ОРДЕНА ЛЕНИНА
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
АКАДЕМИИ НАУК СССР

В.Ф. Турчин

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ РЕФАЛ

II. Формальное описание и принципы реализации рефала

Препринт № 43 за 1971 г

Москва

ОРДЕНА ЛЕНИНА ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ

АКАДЕМИИ НАУК СССР

В.Ф.ТУРЧИН

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ РЕФАЛ

2. ФОРМАЛЬНОЕ ОПИСАНИЕ И ПРИНЦИПЫ РЕАЛИЗАЦИИ РЕФАЛА

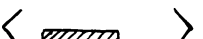
Москва, 1971 г.

Часть I ФОРМАЛЬНОЕ ОПИСАНИЕ ЯЗЫКА

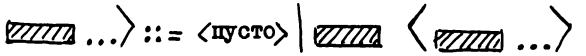
Язык описываемый в настоящей главе, можно назвать абстрактным рефалом; мы исключаем из этого описания те черты используемого на практике языка, которые могут в какой-то степени варьироваться при реализации, отражая либо особенности конкретной вычислительной системы, либо специфическую направленность данной реализации, либо, наконец, изменения, вызванные разработкой новых методов реализации.

I. Синтаксис

Значительную часть синтаксиса мы опишем с помощью бэкусовской нормальной формы, расширенной путем добавления конструкции



где заштрихованный прямоугольник может быть заменен на любое слово, образованное из метасинтаксических переменных и основных символов. Каждая такая конструкция есть метасинтаксическая переменная, которую можно было бы определить так:



где

⟨пусто⟩

Иначе говоря, эта конструкция изображает цепочку, полученную повторением цепочки произвольное число раз (включая ноль).

I.I. Знаки

⟨знак⟩ ::= ⟨собственный знак⟩ | ⟨объектный знак⟩

⟨собственный знак⟩ ⟨скобка⟩ | ⟨управляющий знак⟩

⟨скобка⟩ ⟨символьная скобка⟩

| ⟨объектная скобка⟩

| ⟨функциональная скобка⟩

⟨символьная скобка⟩

⟨объектная скобка⟩ (|)

⟨функциональная скобка⟩ ⫷ | .

⟨управляющий знак⟩ § | ⫾ | ⟨указатель переменной⟩

⟨указатель переменной⟩ :: S | W | E

Объектным знаком может быть любой знак, отличный от собственного знака. Предполагается, что алфавит объектных знаков конечен, хотя он и не фиксируется в описании языка.

I.2. Символы и выражения.

⟨символ⟩ ⟨объектный знак⟩
| ⟨составной символ⟩
| ⟨символ обмена⟩

⟨составной символ⟩ ' ⟨объектная цепочка⟩ '

⟨объектная цепочка⟩ :: ⟨ ⟨объектный знак⟩ ... ⟩

⟨символ обмена⟩ ' § ' | ' ⫾ '

⟨выражение⟩ ⟨ ⟨терм⟩ ... ⟩

⟨терм⟩ ⟨символ⟩
| ⟨свободная переменная⟩
| (⟨выражение⟩)
| ⫷ ⟨выражение⟩ .

⟨свободная переменная⟩ ::=
⟨указатель переменной⟩ ⟨спецификатор⟩ ⟨идентификатор⟩

⟨спецификатор⟩ ::= ⟨пусто⟩
| ⟨составной символ⟩
| (⟨типовое выражение⟩)

⟨идентификатор⟩ ::= ⟨объектный знак⟩

Выражение, которое не содержит знаков конкретизации (но вообще говоря содержит свободные переменные), мы будем называть типовым выражением. Выражение, не содержащее свободных переменных (но вообще говоря содержащее знаки конкретизации) будем называть рабочим выражением. Выражение, которое не содержит ни свободных переменных, ни знаков конкретизации, называется объектным выражением. В следующей таблице перечислены частные случаи понятия выражение и указано, какие знаки они могут содержать (+), и какие не могут (-).

Таблица 2.I

	$ W E	K .	' ( )	Объектные знаки
объектная цепочка				+
объектное выражение			+	+
рабочее выражение		+	+	+
типовое выражение	+		+	+
выражение	+	+	+	+

Областью действия знака K называется выражение, начинающееся непосредственно за этим знаком K , и кончающееся перед парной ему конкретизационной точкой. Ведущим знаком K в некотором выражении называется первый знак K , в области действия которого не содержится ни одного знака K .

I.3. Предложения

<предложение>::= §<комментарий> <левая часть> <правая часть>

<комментарий>::= <типовое выражение>

<левая часть>::= ≤ <типовое выражение> ≥

<правая часть>::= <выражение>

<предложение-комментарий> §<комментарий>

<параграф>::= <предложение> | <предложение-комментарий>

<набор предложений> (<параграф>...)

Приведенные определения образуют общие рамки синтаксиса предложений, но не являются исчерпывающими, то есть не всякое предложение, порожденное этой контекстно-свободной грамматикой, является синтаксически правильным. Дополнительные требования формулируются в п.4. В действительности, синтаксис предложений является существенно контекстно-зависимым.

I.4. Рефал-объекты

<рефал-объект>::= <знак> | <выражение> | <набор предложений>

2 Метакодовое преобразование

Вместе с алфавитом объектных знаков должно быть определено метакодовое преобразование μ, которое каждому рефал-объекту сопоставляет некоторую объектную цепочку, называемую метакодом данного рефал-объекта. Преобразование μ должно обладать однозначным обратным преобразованием μ^{-} (обратное метакодовое преобразование). Для обоих преобразований должны быть определены выполняющие их алгоритмы.

3. Рефал-машина

Рефал-машина - это абстрактное кибернетическое устройство, состоящее из двух потенциально бесконечных запоминающих устрой-

ств – поля памяти и поля зрения – и процессора, преобразующего их содержимое. В каждый момент времени поле памяти содержит некоторый набор предложений, а поле зрения – некоторое рабочее выражение. Работа рефал-машины представляет из себя последовательность однотипных действий, называемых шагами.

Выполнение шага начинается с нахождения в поле зрения ведущего знака $\underline{K}$ Терм, начинающийся с ведущего знака $\underline{K}$ называется конкретизируемым выражением. Если в поле зрения нет знаков $\underline{K}$, рефал-машина приходит в состояние нормальной остановки. Найдя ведущий знак $\underline{K}$, рефал-машина исследует конкретизируемое выражение.

3.1. Если конкретизируемое выражение есть

$$\underline{K}$$

то оно заменяется на метакод набора предложений, находящегося в данный момент в поле памяти.

3.2. Если конкретизируемое выражение имеет вид

$$\underline{K}\ '\ni\ <\mathcal{E}>\ .$$

где $<\mathcal{E}>$ – некоторое выражение, то рефал-машина совершает над $<\mathcal{E}>$ обратное метакодовое преобразование M^{-1}
Если результат есть правильно построенный набор предложений, он помещается в поле памяти рефал-машины вместо набора предложений, стоявшего там прежде. В противном случае происходит аварийная остановка.

3.3. Если за ведущим знаком $\underline{K}$ следует не символ обмена, то рефал-машины выбирает первое предложение, находящееся в поле

памяти, и выполняет алгоритм синтаксического отождествления (см.п.4) конкретизируемого выражения с левой частью данного предложения. Если отождествление невозможно, та же процедура производится со вторым, третьим и т.д. предложением, пока не будет найдено первое подходящее предложение, то есть такое, что для него синтаксическое отождествление заканчивается успешно. При выполнении алгоритма отождествления независимо от его результата может, вообще говоря, меняться и конкретизируемое выражение, и содержимое поля памяти. Если подходящего предложения не найдено, происходит аварийная остановка. Найдя первое подходящее предложение, рефал-машина заменяет в поле зрения конкретизируемое выражение на правую часть предложения, в которой все свободные переменные заменены на те значения, которые они получили в процессе отождествления.

Во всех перечисленных случаях (3.I-3.2), если рефал-машина не останавливается, то шаг считается выполненным и машина переходит к выполнению следующего шага.

Допустим, что в поле зрения рефал-машины находится некоторое выражение $\langle \mathcal{E}_k \rangle$ Тогда о всех последующих состояниях поля зрения (т.е. выражениях, его заполняющих) говорят, что они порождены выражением $\langle \mathcal{E}_k \rangle$ при данном (начальном) поле памяти.

4. Алгоритм синтаксического отождествления.

4.I. Элементы и узлы.

Символы, скобки[x)], свободные переменные и знаки $\underline{<}$, $\doteq$, $\supseteq$ будем называть элементами левой части и конкретизируемого выра-

x) Говоря просто "скобки", мы всегда имеем в виду только обычные, то есть объектные скобки: (,).

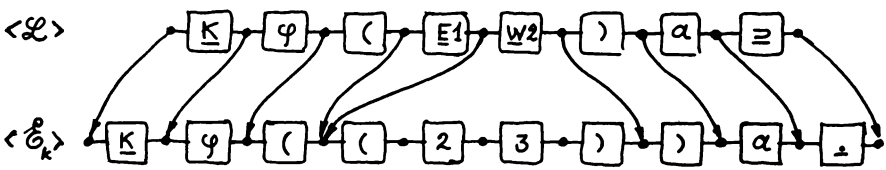
жения. Промежутки между элементами будем называть узлами. Два узла, ограничивающих элемент, будем называть его левым и правым концами.

Если узлу левой части $<\mathcal{L}>$ поставлен в соответствие узел конкретизируемого выражения $<\mathcal{E}_k>$, то будем говорить, что первый узел спроектирован на второй, и называть второй узел проекцией первого. При проектировании узлов в процессе синтаксического отождествления должны выполняться следующие два условия:

4.I.I. Каждому узлу в $<\mathcal{L}>$ может соответствовать не более одной проекции в $<\mathcal{E}_k>$.

4.I.2. Если узел k' в $<\mathcal{L}>$ лежит правее узла k, то проекция k' в $<\mathcal{E}_k>$ не может лежать левее проекции k.

Пример проектирования узлов показан на Фиг.2.I.



Фиг.2.I. Проектирование узлов левой части $<\mathcal{L}>$ на узлы конкретизируемого выражения $<\mathcal{E}_k>$.

4.2. Начало процесса. Классификация свободных переменных.

4.2.I. Выполнение алгоритма синтаксического отождествления начинается с того, что правый конец знака $\underline{K}$ и левый конец знака $\equiv$ в левой части $<\mathcal{L}>$ проектируются, соответственно, на правый конец знака $\underline{K}$ и левый конец знака $.$ в конкретизируемом выражении $<\mathcal{E}_k>$. Каждое последующее действие рефал-машины есть либо проектирование элемента, либо перестройка. Правила проектирования элементов (п.4.3) таковы, что они однозначно определяют порядок, в котором будут проектироваться элементы, образующие левую часть $<\mathcal{L}>$. Тем самым каждому элементу приписывается определенный

проекционный номер. Проекционные номера элементов зависят только от вида самой левой части, но не от выражения на которое левая часть проектируется, и не меняются при перестройке.

4.2.2. До сих пор, говоря о свободных переменных, мы фактически имели в виду вхождения свободной переменной, опуская слово "вхождение" для краткости. Теперь нам надо будет в некоторых случаях различать собственно свобсдную переменную (как семантическое понятие) и ее различные вхождения (синтаксические объекты). Свободную переменную можно определить как совокупность всех вхождений свободной переменной с одним и тем же идентификатором. В одном предложении не могут встр ечаться вхождения свободных переменных с различными указателями, но одинаковыми идентификаторами. В правую часть могут входить только те свободные переменные, которые входят в левую часть.

4.2.3. То вхождение свободной переменной в левую часть предложения, которое проектируется первым, называется главным вхождением, остальные - повторными. Проекция главного вхождения переменной есть ее значение.

4.2.4. Если в главном вхождении переменной спецификатор пустой, переменная называется базисной, если спецификатор непустой, переменная называется рекурсивной. Повторные вхождения переменных и их вхождения в правую часть должны иметь пустой спецификатор. Спецификатор рекурсивной переменной может содержать лишь повторные вхождения свободных переменных.

4.3. Правила проектирования элементов

Элемент может проектироваться только в том случае,когда хотя бы один из его концов уже спроектирован. Если спроектированы оба конца элемента, то это еще не значит, что спроектирован элемент: необходимо еще формальное действие проектирования элемента.

Проектирование элемента приводит либо к положительному резуль-

тату, когда оба конца элемента получают проекции или подтверждаются их ранее установленные проекции) и элемент считается спроектированным на выражение, ограниченное проекциями его концов, либо к отрицательному результату-констатации тупика, когда данный элемент не может быть спроектирован без перепроектирования уже спроектированных элементов.

По своей очередности в ряду проектирований элементы левой части предложения делятся на четыре группы (см.пп. 4.3.1 - 4.3.4). Сначала рефал-машина исследует, можно ли проектировать (не путать "проектировать" и "спроектировать"!) какой-либо элемент первой группы очередности. Если таковых нет, исследуются элементы второй группы и т.д. Если можно проектировать несколько элементов одной группы, машина выбирает из них первый слева по своему положению в левой части предложения. Если проектирование приводит к положительному результату, совершается следующий акт проектирования (начиная с пробы элементов первой группы очередности) и т.д.
Если проектирование приводит к тупику, осуществляется попытка перестройки (см.п.5).

4.3.1. К первой группе очередности относятся скобки, у которых спроектирована парная скобка, и все элементы, у которых уже оказались спроектированными (в результате проектирования других элементов) оба конца, за исключением главных вхождений рекурсивных переменных. Свободные переменные выражения, относящиеся к этой группе, называются закрытыми переменными. Проектирование закрытой переменной дает всегда положительный результат, состоящий просто в подтверждении наличия проекций у концов элемента. Скобка, у которой спроектирована парная скобка, проектируется на скобку, парную к проекции парной скобки. Для остальных элементов результат проектирования будет положительным при условии выполнения следующих требо-

ваний.

4.3.I.I. Проекции скобок и символов должны быть тождественны им самим.

4.3.I.2. Проекцией переменной $\underline{S}\langle a\rangle$ и $\underline{W}\langle a\rangle$ может быть любой символ и терм, соответственно.

4.3.I.3. Проекция повторного вхождения переменной должна совпадать с проекцией ее главного вхождения.

В противном случае имеет место тупик.

4.3.2. Ко второй группе очередности относятся следующие элементы, у которых спроектирован один из концов: скобки, символы, главные вхождения переменных вида $\underline{S}\langle a\rangle$ и $\underline{W}\langle a\rangle$ ($\langle a\rangle$ здесь и дальше означает произвольный объектный знак) и повторные вхождения переменных. Проектирование дает положительный результат, если второй конец элемента может быть спроектирован таким образом, чтобы были соблюдены требования 4.I.I-2 с одной стороны, и требования 4.3.I.I - 3 с другой. В противном случае имеет место тупик.

4.3.3. Третью группу очередности образуют главные вхождения рекурсивных переменных символа и терма, у которых спроектирован один или два конца.

Элемент вида $\underline{S}(\langle\mathcal{E}\rangle)\langle a\rangle$, где $\langle\mathcal{E}\rangle$ здесь и дальше — произвольное выражение, сначала проектируется как $\underline{S}\langle a\rangle$, если это возможно. Затем его проекция, которую мы обозначим через $\langle S_0\rangle$, заменяется на выражение

(*) $$\underline{\leq}\langle\mathcal{E}'\rangle\langle S_0\rangle\underline{.}$$

где $\langle\mathcal{E}'\rangle$ получается из $\langle\mathcal{E}\rangle$ заменой всех свободных переменных на их значения. Процесс отождествления временно приостанавливается и рефал-машина пускается в ход, причем только что введенный знак $\underline{\leq}$ становится, очевидно, ведущим. Работа рефал-

машины продолжается до тех пор, пока выражение, порожденное выражением (*) содержит хотя бы один знак К. Когда знаков К в нем больше нет, рефал-машина исследует результат. Если он имеет вид

$$(\langle \mathcal{E}_1 \rangle)$$

где $\langle \mathcal{E}_1 \rangle$ - произвольное выражение, то переменная $\underline{S}\langle a \rangle$ принимает значение $\langle \mathcal{E}_1 \rangle$ скобки, окружающие $\langle \mathcal{E}_1 \rangle$ уничтожаются, и процесс отождествления продолжается. Если результат имеет вид

$$\langle S_\pi \rangle \langle \mathcal{E}_2 \rangle$$

где $\langle S_\pi \rangle$ и $\langle \mathcal{E}_2 \rangle$ - произвольные символ и выражение, то $\langle S_\pi \rangle$ уничтожается и объявляется тупиковая ситуация. Если результат не укладывается ни в ту, ни в другую схему, происходит аварийная остановка.

Элемент вида $\underline{W}(\langle \mathcal{E} \rangle)\langle a \rangle$ трактуется аналогично, надо только вместо $\underline{S}$ и $\langle S_0 \rangle$ подставить $\underline{W}$ и $\langle W_0 \rangle$ - произвольный терм; ($\langle S_\pi \rangle$ остается символом).

Элементы вида $\underline{S}'\langle ab \rangle'\langle a \rangle$ и $\underline{W}'\langle ab \rangle'\langle a \rangle$, где $\langle ab \rangle$ - объектная цепочка, трактуются как $\underline{S}(\text{'}\langle ab \rangle\text{'})\langle a \rangle$ и $\underline{W}(\text{'}\langle ab \rangle\text{'})\langle a \rangle$ соответственно.

4.3.4. Главные вхождения переменных выражения: всех рекурсивных и тех базисных, которые не вошли в п.4.3.1 образуют четвертую, и последнюю, группу очередности.

Главное вхождение рекурсивной переменной выражения проектируется только после того, как тем или иным путем спроектирован его левый конец (спроектирован ли правый конец, не имеет значения). Проектирование переменной вида $\underline{E}(\langle \mathcal{E} \rangle)\langle a \rangle$ осуществляется следующим образом, От проекции левого конца элемента до ближайшего узла в поле зрения, на который уже спроектирован какой-либо узел левой части $\langle \mathcal{L} \rangle$, выделяется выражение, которое мы обозначим через $\langle \mathcal{E}_0 \rangle$. Оно заме-

няется на

$$(**) \qquad \leq \langle \mathcal{E}' \rangle \langle \mathcal{E}_0 \rangle$$

после чего рефал-машина запускается, аналогично тому, как было описано в п.4.3.3. Если результат имеет вид

$$(\langle \mathcal{E}_1 \rangle) \langle \mathcal{E}_2 \rangle$$

то скобки, окружающие $\langle \mathcal{E}_1 \rangle$ удаляются, и переменная $\underline{E}\langle a \rangle$ принимает значение $\langle \mathcal{E}_1 \rangle$ - при том дополнительном условии, что это не противоречит п. 4.I.I. (противоречие может возникнуть, если правый конец элемента уже спроектирован). Это случай успешного проектирования. Если присваивание переменной значения $\langle \mathcal{E}_1 \rangle$ противоречит п. 4.I.I, имеет место тупик. Тупик имеет место также в том случае, если результат конкретизации $(**)$ имеет вид

$$\langle S_n \rangle \langle \mathcal{E}_3 \rangle$$

Символ $\langle S_n \rangle$ из поля зрения удаляется. Переменная вида $\underline{E}\,'\langle a b \rangle'\langle a \rangle$ трактуется как $\underline{E}(\,'\langle a b \rangle\,')\langle a \rangle$.

Базисная переменная выражения, у главного вхождения которой спроектирован только левый конец, в то время как ни один элемент из первых трех групп очередности проектироваться уже не может, называется <u>открытой.</u> Её проектирование всегда приводит к положительному результату, состоящему в том, что правый конец проектируется на тот же узел, что и левый конец, и таким образом, переменная принимает значение <пусто> Это, конечно, только <u>начальное</u> значение открытой переменной. Непустое значение открытые переменные принимают в результате <u>перестройки</u>.

4.4. Перестройки и конец процесса

Перестройка может быть названа также удлинением открытых переменных. Перестройка производится, когда проектирование некоторого элемента, проекционный номер которого мы обозначим через i_0 приводит к тупику. Она выполняется так. Находится главное вхождение открытой переменной с максимальным проекционным номером i_1, не превышающим i_0, и результаты проектирования элементов с номерами $i > i_1$ аннулируются. Затем проекция правого узла открытой переменной с номером i_1 переносится на один терм вправо, если это не противоречит условиям п.п. 4.I.I÷2 (переменная удлиняется), и проектирование продолжается, начиная с элемента номер $i_1 + 1$. Если удлинение переменной номер i_1 невозможно, делается попытка удлинить открытую переменную с максимальным проекционным номером i_2 не превышающим i_1 и т.д. Если невозможно удлинить ни одной открытой переменной, процесс синтаксического отождествления заканчивается с результатом "отождествление невозможно".

Если спроектированы все элементы левой части предложения, отождествление считается успешно выполненным.

Задачи

2.I. В следующих левых частях надписать над элементами их проекционные номера, и классифицировать вхождения свободных переменных:

А) К φ Е1 ÷ Е2 (W 3 А (* *) Е1 В С) ≥

Б) К α Е А (Е 1)(W 2)(Z)(S 3) ≥

В) К α Е А (W К Е 1)(W L W 2)(X Z)(S М S 3) ≥

Г) К 'БР' (Е1 ÷ Е2 ÷ Е3) Е1 ÷ Е2 (Е 3) ≥

Д) К φ Е А (Е 1) (W 'КС' 2) (А) (S ('СУ' 14) 3) ≥

Е) К φ Е А (Е1) (W ('КС' Е1) 2) (А) (S (β Е А) 3) ≥

Ж) К 'Фи' Е1 (Е7 Е 'Сил' 666) Е2 ≥

3) K α E 'ид'1 W2 + E3 + E (β E3) A EB =

2.2. В следующих левых частях указать синтаксические ошибки, если они есть:

А) K 'сус' (E1 - E 'ид'2 - E3) W1 W2 W3 =

Б) K φ W ('X' S A)1 (E2 SA E3) =

В) K φ W ('X' S A)1 (E2 * E3) S A =

Г) K φ S1 S (α S1)2 S (α S 2)3 S (α S 3)4 =

Д) K α EA + (W1 E 'сус'A E2) EB =

Е) K E 'ид'1 W2 + E3 (E1) =

Ж) K E1 W2 + E3 (E 'ид' 1) =

2.3. В следующих предложениях указать синтаксические ошибки, если они есть:

А) § K φ E (A)1 + EB = EB + EA

Б) § K φ E 'ид'1 + E2 = E 'ид'1 - E2

В) § K x = K K K x . . .

2.4. Можно ли быть уверенным, что выражение, подставляемое в поле зрения при использовании предложения:

§ K φ S1 (E A : E B (S 'X'2) S 3) E5 = S1 S2 S3

состоит ровно из трех символов?

2.5. Назовем областью определения функции φ множество выражений <ℰ> таких, что выражение

K φ < ℰ > .

порождает за конечное число шагов рефал-машины объектное выражение.

Доказать, что если функции β и γ определены на любом выражении, то функция α

§ K α S1 S2 E3 = K α (K β S1 .) S2 K γ E3 .

§ K α E1 = K β E1 .

обладает тем же свойством.

2.6. То же для функции:

$$\S\ \underline{K}\alpha\ (\underline{E}1\ \underline{W}2)\ \underline{E}3 = \underline{K}\alpha\ (\underline{E}1)\ \underline{K}\beta\ \underline{W}2\ .\ \underline{K}\gamma\ \underline{E}3 \leftarrow \leftarrow$$

$$\S\ \underline{K}\alpha\ \underline{E}1 = \underline{E}1$$

2.7. Доказать алгоритмическую полноту рефала, сопоставив каждому нормальному алгорифму Маркова эквивалентную рекурсивную функцию, описанную на рефале.

Часть 2 ПРИНЦИПЫ РЕАЛИЗАЦИИ РЕФАЛА

Чтобы выполнять с помощью электронной вычислительной машины алгоритмы, описанные на рефале, надо написать программу, имитирующую работу рефал-машины. Входной информацией для этой программы будет набор предложений, предназначенный для поля памяти рефал-машины, (алгоритм) и выражение, помещаемое перед началом работы в поле зрения (начальные данные). Если набор предложений сохраняет в процессе работы вычислительной машины свой натуральный вид, или претерпевает лишь небольшую модификацию, то такая программа называется, по сложившейся терминологии, интерпретатором для данного языка (рефала). Если набор предложений подвергается перед началом работы существенной переработке, то-есть, рассматривается, по существу, как информация для создания нового текста на машинном или машинно-ориентированном языке, то такая программа носит название компилятора. Интерпретаторы и компиляторы объединяют под общим наименованием трансляторов с данного языка. В настоящей главе мы изложим основные принципы создания трансляторов с рефала для современных вычислительных машин (см. [4])[x]. Эти принципы общи для интерпретаторов и компиляторов, и касаются, главным образом, расположения в реальной вычислительной машине информации, образующей содержимое поля зрения абстрактной рефал-машины. Знание этих принципов необходимо не только разработчику рефал-транслятора, но и тому, кто им пользуется, если он желает написать на рефале эффективно работающую программу.

---

x) Список литературы см. в 5-ом выпуске данной серии.

I. Размещение информации в поле зрения

Действия, совершаемые рефал-машиной над содержимым поля зрения, заключаются в удалении одних подвыражений и замене их другими. Поэтому информацию в поле зрения надо хранить таким образом, чтобы эти действия не приводили к ненужной переписи тех частей поля зрения, которые не затрагиваются непосредственно данной подстановкой. Это достигается с помощью так называемой списковой структуры памяти, отведенной под поле зрения. Такая структура позволяет избежать и других необязательных для выполнения алгоритма действий (просмотров), которые входят в формальное описание абстрактной рефал-машины. В рефал- трансляторе списковая структура выглядит следующим образом.

Основной единицей структуры является звено; память, отведенная под поле зрения, представляет из себя совокупность некоторого числа звеньев. Звено - это группа двоичных разрядов, допускающая прямую адресацию. В машинах с достаточно большой ячейкой основной памяти звеном может являться одна ячейка. Так обстоит дело, например, в случае машин М-20, БЭСМ-6 и др. В машинах серии ИБМ-360 в качестве звена может служить последовательность из шести или восьми байтов. Звено состоит из четырех полей: поля признака π и трех адресных полей A_1, A_2, A_3, (см. фиг. 3.I). Размер адресных полей должен быть таким, чтобы они могли содержать адрес любого звена, входящего в поле зрения. Поля A_2 и A_3 служат для связывания звеньев в линейную последовательность; поле A_2 всегда содержит адрес предыдущего звена в этой последовательности, поле

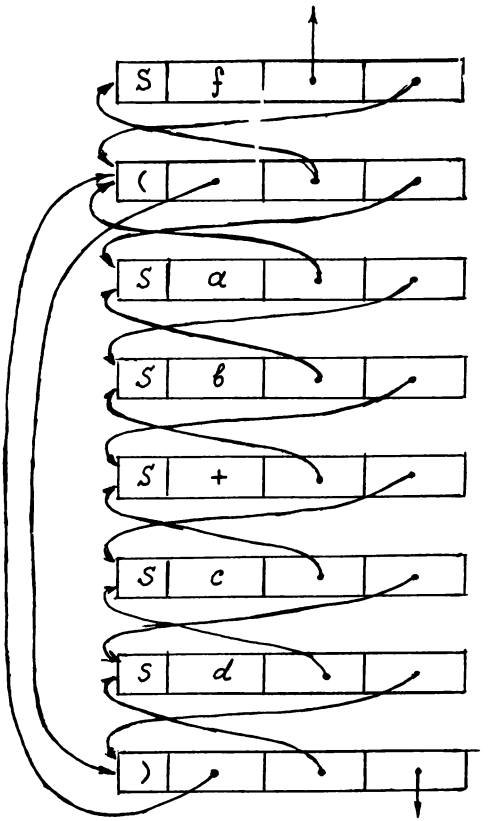
Фиг. 3.I Строение звена.

A_3 содержит адрес следующего звена. Содержимое поля A_1 зависит от значения признака π, которое в свою очередь зависит от того, какому рефал-объекту соответствует данное звено. Звено может изображать либо символ, либо скобку (левую или правую), либо знак конкретизации;(о конкретизационной точке см. ниже). Признак π, следовательно, должен принимать по крайней мере четыре значения, отличающие эти четыре случая. Кроме того, при различных реализациях может быть введено несколько типов символов (например, буквы, числа, составные символы) и для быстрого их распознавания отведено несколько разрядов в поле признака.

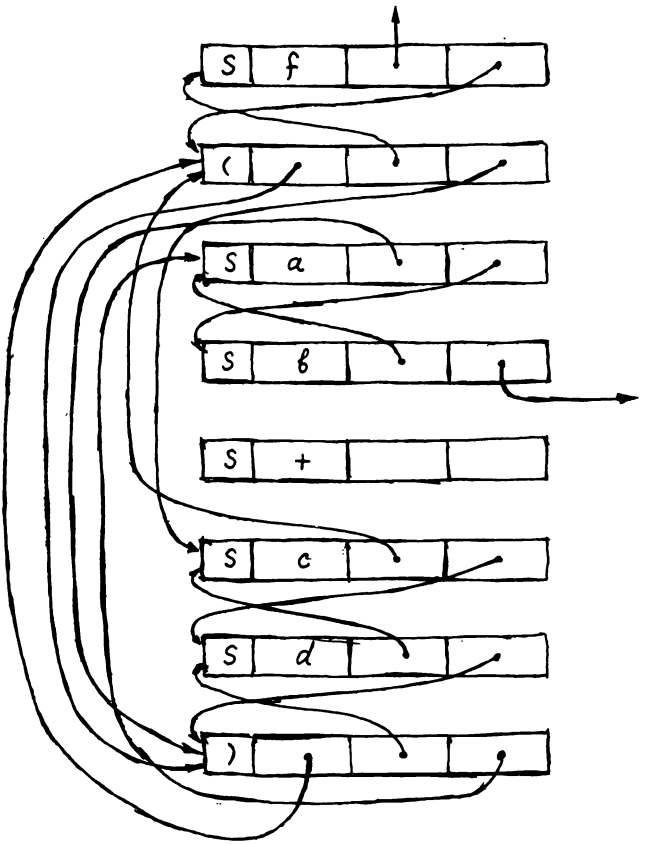
Если звено изображает символ, то поле A_1 содержит код этого символа. Если звено изображает скобку, то поле A_1 содержит адрес парной скобки.

Благодаря такому представлению информации становится возможным осуществлять весьма значительные преобразования текста - вставки, перестановки и вычеркивания - не переписывая частей текста, а лишь изменяя адреса связи. Пусть, например, в поле зрения находится выражение $f(ab+cd)$, представленное восемью подряд идущими звеньями в оперативной памяти вычислительной машины (см. фиг.3.2). Допустим, что нам надо преобразовать это выражение в $f(cd)ab$. Для этого достаточно "перешить" цепочку ab и удалить символ + . В результате получается структура, показанная на фиг.3.3. Ясно, что если бы вместо цепочек ab и cd были сколь угодно длинные цепочки, число операций, необходимых, чтобы выполнить преобразование, осталось бы тем же самым.

Звено, содержащее символ + , оказалось лишним, на рисунке 3.3 оно лишено связей. Это звено подшивается к "свободной памяти" -



Фиг. 3.2 Представление в поле зрения выражения $f(ab+cd)$.



Фиг. 3.3 Представление в поле зрения выражения $f(cd)ab$,полученного трансформацией структуры,изображенной на фиг. 3.2 .

— цепочке звеньев, не несущих в данный момент информации о поле зрения. Если в выражение, содержащееся в поле зрения (в дальнейшем мы это выражение будем для краткости называть просто полем зрения) требуется сделать вставку, то необходимые звенья, уже связанные в цепочку, отщепляются от той же "свободной памяти". Итак, звенья, отведенные под поле зрения, всегда образуют две непрерывные цепочки: одна изображает собственно поле зрения, другая представляет свободную память. Действия транслятора над полем зрения сводятся к перешивке частей внутри первой цепочки и обмену частями между первой и второй цепочками. В результате этих действий физическое расположение звеньев в памяти машины теряет всякую связь (если она существовала вначале) с логическим порядком звеньев, отражающим порядок расположения объектов в поле зрения абстрактной рефал-машины.

Благодаря тому, что в звене, изображающем скобку, хранится адрес парной скобки, становится возможным непосредственное проектирование парной скобки в процессе отождествления без просмотра содержимого этой пары скобок, а путем простого перехода по адресу. Становится также возможным отождествление терма, начинающегося со скобки, без просмотра содержимого скобок.

Рассмотрим следующий пример. Пусть типовое подвыражение

```
1  3  4   8  7  10  9  6  5  2
(  A  S1  E2 (  E3  W4 )  W5 )
```

левой части предложения надо спроектировать на объектное подвыражение

```
      8      10       5
3 4       7       9 6       2
( A B C D ( A + B * ) ( x y z ) )
```

поля зрения. В типовом подвыражении мы пометили элементы в том порядке, в котором они будут проектироваться; в объектном подвыражении указаны проекции элементов типового подвыражения.

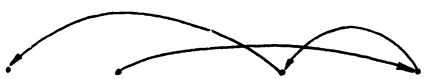
Мы исходим из предположения, что в начале отождествления тем или иным способом спроектирована левая скобка 1, то-есть, в поле зрения найдено соответствующее звено. Из поля A_1, этого звена транслятор выбирает адрес и по этому адресу находил звено, соответствующее правой скобке 2, а выбирая адрес из поля A_3, находит звено, которое должно соответствовать следующему за скобкой 1 элементу типового выражения, то-есть, элементу 3. В соответствии с алгоритмом отождествления в первую очередь должен проектироваться элемент 3. Транслятор убеждается, что соответствующее звено изображает символ, и что поле A_1 содержит код символа A. Выбирая адрес следующего звена, транслятор придает свободной переменной $\underline{S}1$ значение символа B Так как за $\underline{S}1$ следует $\underline{E}2$ транслятор переходит на правый конец типового выражения, где возможно продолжение проектирования. Выбирая поле A_2 из звена 2, транслятор находит звено, которое должно отойти к проекции элемента 5. Выяснив, что это звено изображает правую скобку, транслятор находит парную ей левую скобку, полностью определяя тем самым проекцию переменной $\underline{W}5$ Снова выбирая поле A_2, транслятор находит звено, соответствующее элементу 6, (правая скобка), а используя A_1 находит проекцию левой скобки 7. Теперь и левый и правый конец свободной переменной $\underline{E}2$ оказываются спроектированными (закрытая переменная). Опираясь на проекцию правой скобки 6, транслятор проектирует свободную переменную терма $\underline{W}4$ на символ $*$ после чего переменная $\underline{E}3$ также оказывается спроектированной.

Легко видеть, что если бы в объектном выражении вместо цепочек CD, $A+B$ и xyz стояли бы сколь угодно длинные цепочки, то число операций, необходимых для отождествления, не изменилось бы. Если в типовом выражении нет открытых свободных переменных, то все отождествление выполняется без перестроек (удлинений), путем ряда актов проектирования, и число операций вычислительной машины будет определяться исключительно числом элементов в левой части предложения, вне зависимости от длины конкретизируемого выражения в поле зрения. При наличии открытых переменных, значения которых можно определить только путем последовательного - терм за термом - удлинения, время, затрачиваемое на выполнение отождествления, разумеется, зависит от вида конкретизируемого выражения. Однако при правильной записи алгоритмов на рефале, открытые переменные употребляются лишь в тех случаях, когда последовательное удлинение, то-есть, просмотр выражений в поле зрения, необходим по самому смыслу алгоритма, так что избежать его нет ни возможности, ни необходимости. Подчеркнем, что число операций, совершаемых при наборе открытой переменной, пропорционально не числу знаков в объектном выражении, а числу термов в нем. При наборе выражения, как и при проектировании, транслятор совершает переход по адресу A_1, не заходя в скобки.

Теперь коснемся представления в поле зрения функциональных скобок. Согласно формальному описанию языка, рефал-машина должна в начале каждого шага просматривать поле зрения в поисках ведущего знака конкретизации $\underline{K}$. Ясно, что буквальное выполнение этого требования повлечет серьезную потерю эффективности транслятора. К счастью, определение ведущего знака $\underline{K}$ позволяет избежать ненужных просмотров поля зрения. Достигается это следующим образом.

Если в выражение входит более одного знака $\underline{K}$ то для них можно ввести отношение порядка, которое мы назовем <u>старшинством</u>. Самым младшим знаком $\underline{K}$ объявим ведущий знак $\underline{K}$ Следующим по старшинству будет тот знак $\underline{K}$ который станет ведущим, если удалить из выражения ведущий знак $\underline{K}$ и парную ему конкретизационную точку. Продолжая эту процедуру, распределим по старшинству все знаки конкретизации. Для каждого знака $\underline{K}$ определяем непосредственно следующий за ним по старшинству знак $\underline{K}$ как тот знак, который станет ведущим, если удалить данный знак $\underline{K}$ и все младшие знаки $\underline{K}$ вместе с парными точками.

Итак, все знаки конкретизации в поле зрения можно связать в непрерывную цепь стрелками, которые от каждого знака $\underline{K}$ ведут к следующему за ним по старшинству. Приведем такой пример:



$$\underline{K}\alpha\,BB\,\underline{K}\rho(xy).\,\underline{K}\gamma(yx)\,\underline{K}\delta \qquad BB.$$

Чтобы зафиксировать эту цепь в трансляторе, надо с каждым знаком конкретизации связать <u>четыре</u> адреса: адреса предыдущего и последующего звеньев, адрес парной точки и адрес следующего по старшинству знака конкретизации. Так как звено содержит три адреса, можно достичь этого, отображая знак $\underline{K}$ на два звена: собственно знак конкретизации с адресом следующего знака $\underline{K}$ в поле A_1 и обычную левую скобку, содержащую в поле A_1 адрес конкретизационной точки. Конкретизационная точка будет при этом тождественна обычной правой скобке. Иначе говоря терм

$$\underline{K}<\mathfrak{B}>.$$

будет записываться в поле зрения как

$$\lambda\,(<\mathfrak{C}>)$$

где λ - "непарный" знак конкретизации, относящийся к следующему за ним терму.[x] Можно пойти и по другому пути: изображать в поле зрения функциональные скобки в точности так же, как и обычные (объектные) скобки, а адреса тех левых скобок, которые на самом деле суть знаки $\underline{\kappa}$ вынести в отдельный стэк - "стэк конкретизаций". Так или иначе, но в трансляторе должны храниться адреса знаков $\underline{\kappa}$ в поле зрения, связанные друг с другом в цепь по старшинству. Пусть в поле зрения знаки $\underline{\kappa}$ связаны в цепь по старшинству. Если терм, начинающийся с ведущего знака $\underline{\kappa}$, заменить на любое рабочее выражение, в котором знаки $\underline{\kappa}$ также связаны в цепь по старшинству, и связать старший знак $\underline{\kappa}$ вставляемого выражения с тем знаком $\underline{\kappa}$ который по старшинству непосредственно следовал за ведущим знаком $\underline{\kappa}$ в поле зрения, то и результирующее выражение будет обладать правильной цепью по старшинству знаков $\underline{\kappa}$ Отсюда следует такой алгоритм выбора транслятором ведущего знака $\underline{\kappa}$

1. В начале шага из поля A_1 ведущего знака конкретизации выбирается адрес следующего за ним по старшинству знака конкретизации.

2. Если в правой части используемого предложения нет знаков $\underline{\kappa}$ то знак $\underline{\kappa}$ найденный в п.1, объявляется ведущим.

---

x/ Такое представление можно было бы принять и при записи алгоритмов на бумаге, как это и было сделано в ранних работах [1,2]. Однако оно менее удобно, так как порождает нагромождение одинаковых скобок.

3. Если в правой части используемого предложения есть знаки $\underline{K}$, то в выражении, порождаемом в поле зрения как копия правой части, знаки $\underline{K}$ связываются в цепь по старшинству. Легко понять, что это требует числа операций, зависящего лишь от вида правой части предложения, но не от вида и размеров значений свободных переменных.

4. Старший знак $\underline{K}$ в полученном выражении связывается со знаком $\underline{K}$,найденным в п.I. Младший знак $\underline{K}$ становится ведущим.

Проанализируем работу рефал-транслятора с точки зрения эффективности выполнения алгоритма на примере простейшей функции, заменяющей знаки + (на верхнем скобочном уровне выражения) на знаки -

$$\S\,8.1\ \ \underline{K}\alpha\ \underline{E}1+\underline{E}2 \ =\ \underline{E}1-\underline{K}\alpha\ \underline{E}2\ .$$

$$\S\,8.2\ \ \underline{K}\alpha\ \underline{E}1 \ =\ \underline{E}1$$

Свободная переменная $\underline{E}1$ - открытая, поэтому сначала транслятор придает ей значение <пусто> и, продолжая отождествление, исследует, не является ли следующее звено (то-есть первое звено после детерминатива α) знаком + Если это не так, транслятор пропускает один терм, снова проверяет, не следует ли за ним знак + и т.д., пока не обнаруживает первый знак + Обнаружив его, транслятор заканчивает отождествление (остальная часть конкретизируемого выражения отходит к переменной $\underline{E}2$ без просмотра) и приступает к замене. Чтобы выполнить её, достаточно перенести (путем изменения адресов связи - "перешивки") группу $\underline{K}\alpha$, поместив ее непосредственно за знаком + , и заменить этот знак на то-есть изменить код в поле A_1 соответствующего звена. Так как знак $\underline{K}$ - единственный, он как был, так и остается ведущим. Следующий шаг начи-

нается с просмотра оставшейся части объектного выражения - значения переменной $\underline{\in} 2$. Когда, дойдя до конца выражения, транслятор не обнаруживает знака +, он переходит к использованию второго предложения. Здесь отождествление происходит сразу же, без просмотра аргумента; группа $\underline{<}\alpha$ и точка $\doteq$ удаляются, и работа заканчивается.

Таким образом, никаких лишних просмотров или переписей обрабатываемых выражений транслятор не совершает. Если бы мы писали вручную программу замены + на - при том же представлении информации в памяти машины (списковая структура), то наша программа совершала бы те же самые действия просмотра и замены кодов, что и рефал-транслятор. Отличие состояло бы только в том, что в программе, написанной вручную, адрес начала непросмотренной цепочки просто запоминался бы в ячейке или регистре, в то время как рефал-транслятор запоминает его путем перешивания группы $\underline{<}\alpha$. Это приводит к некоторой потере эффективности. Если, например, знаком + оказывается в среднем каждый десятый терм, то число операций, совершаемых машиной, возрастает на 15-20%, независимо от размеров обрабатываемого материала.

Эта ситуация является типичной. Использование рефала, как и любого проблемно-ориентированного языка, всегда приводит к определенному снижению эффективности работы машины по сравнению с наилучшими программами, написанными на машинно-ориентированном языке. Однако при правильном программировании потеря эффективности будет постоянной, не зависящей от объема материала, и приемлемой по величине (хотя, конечно, она может быть и гораздо больше, чем 20%).

2. Объектные знаки и составные символы

Объектные знаки при реализации рефала интерпретируются как группы двоичных разрядов, имеющих, как правило, определенное внешнее представление на устройствах ввода-вывода. Так как одна и та же машина может быть оборудована различными устройствами ввода-вывода, желательно считать объектным знаком любую последовательность двоичных разрядов фиксированной длины. Для машин с байтовой структурой памяти объектный знак - это произвольный байт (восемь двоичных разрядов).

По формальному описанию рефала различие между составными символами и объектными знаками проявляется только в формировании рекурсивных и базисных переменных. При реализации удобно трактовать составной символ как адрес ячейки в памяти машины, являющийся ключом для извлечения всей информации, связанной с данным символом. Эта информация включает, прежде всего, последовательность объектных знаков, составляющих тело символа. Кроме того, если данный составной символ является детерминативом некоторой группы предложений, надо указать адрес первого предложения этой группы.

Такая интерпретация составного символа подсказывает следующее ограничение входного языка: детерминативами предложений могут быть только составные символы. Заметим, что в формальном описании рефала вообще нет понятия детерминатива, и никаких ограничений на вид левой части не накладывается: на уровне абстрактного языка они не оправданы. При реализации можно потребовать, чтобы левая часть начиналась с символа - детерминатива. Если вдобавок ограничить детерминативы составными символами, то они, фактически, пре-

Вращаются в ссылку на подпрограмму данной функции. Это ограничение не обременительно, но для реализации на вычислительной машине очень удобно, так как порождает четкое разделение: объектный знак - это константа, раз навсегда определенный код, а составной символ - это метка, которая принимает значение конкретного адреса лишь перед началом работы машины, и может, вообще говоря, принимать различные значения от одного запуска транслятора к другому. Имея это в виду, мы будем составные символы называть также метками. Итак, код символа (содержимое поля A_1) есть либо код объектного знака, либо адресное значение метки. Эта система весьма удобна для связи программ, написанных на рефале, с другими программами, идущими на той же вычислительной машине, ибо она дает возможность все необходимые адреса связи ввести в поле зрения в качестве составных символов (меток). Использоваться эти адреса будут с помощью "машинных процедур" (см. ниже).

В настоящей книге мы принимаем ограничение детерминативов составными символами, но чтобы сохранить возможность краткой записи предложений оставляет за собой право заменять составные символы греческими буквами. При записи составных символов в их полном виде мы будем соблюдать еще несколько ограничений: тело символа должно быть образовано только из букв и цифр, начинаться оно должно с буквы, а символы, которые мы рассматриваем как различные, должны обязательно различаться в своих первых шести знаках. Мы рекомендуем и читателям приучить себе к этим ограничениям. Они вытекают из того, что при реализации рефала удобно превращать составные символы непосредственно в метки автокода (языка ассемблера), поэтому обычные ограничения на метки автокода переходят в ограничения на составные символы.

3. Метакод - *А*

В формальном описании рефала метакод введен как средство обмена информацией между полем памяти и полем зрения. Он необходим потому, что поле памяти содержит такие собственные знаки, которые не могут входить в поле зрения. При реализации рефала аналогичная проблема возникает, когда нужно осуществить обмен информацией между полем зрения и устройствами ввода-вывода. Ибо поле зрения может содержать собственные знаки рефала - скобки (трех видов), в то время как для устройств ввода-вывода, а также для обмена информацией между рефал-транслятором и другими программами, вся информация представляется последовательностью групп двоичных разрядов фиксированной длины, то-есть объектных знаков. Мы будем считать для определенности, что эти группы разрядов - байты. Таким образом, для всех инстанций, внешних по отношению к рефал-транслятору, информация, которую они получают от рефал-транслятора и передают ему, есть объектные цепочки - последовательности байтов. Если зафиксировать несколько байтов, которые всегда воспринимать как собственные знаки рефала, то это порождает неудобства, так как накладывает ограничения на входную информацию, которую может обрабатывать рефал-транслятор. Поэтому мы примем за принцип, что любой байт должен рассматриваться как объектный знак. Но тогда необходим метакод, для преобразования содержимого поля зрения в последовательность объектных знаков и обратно. Чтобы уменьшить число основных определений, это преобразование разумно объединить с обменом между полем памяти и полем зрения. Поэтому метакод, определенный в формальном описании рефала, преобразует набор предложений (а тем самым и любой рефал-объект) не в рабочее или объектное выражение, а в

объектную цепочку. Имея такой метакод, можно, если это представится необходимыми, осуществить и частичное преобразование, не затрагивающее символьных и/или объектных скобок.

Мы опишем два метакода, которые будем называть метакод-А и метакод-Б.

Метакод-А строится таким образом, чтобы быть максимально приближенным к той записи рефал-объектов на бумаге, которой мы пользовались до сих пор и будем в настоящей книге пользоваться и впредь. В этой записи бóльшая часть собственных знаков рефала изображается путем подчеркивания простых знаков, которые могут использоваться как объектные знаки. Основной принцип метакода-А состоит в том, что такие знаки преобразуются в пару объектных знаков: знак подчеркивания и "содержательный" знак (тот, который подчеркивается).

Знак подчеркивания имеется на многих устройствах ввода-вывода. На печати он выглядит как низко расположенная черточка _ Кроме того, он обладает той особенностью, что у обычных печатающих устройств не вызывает сдвига каретки, так что следующий за ним знак печатается на том же месте и оказывается, таким образом, подчеркнутым. Отсюда следует соглашение о набивке текста на перфорирующем устройстве: встречая подчеркнутый знак, пробивают сначала подчеркивание, а затем этот знак. Как и всякий объектный знак, подчеркивание имеет определенный код - байт. Мы будем для наглядности изображать байт, соответствующий объектному знаку, квадратиком, в котором помещен печатный вид знака. Следовательно, подчеркиванию соответствует байт [_] а тройка байтов [_] [S] [A] будет выглядеть на печати как S A

Итак, подчеркнутый знак, встречающийся в некотором тексте допускает двойственную интерпретацию: как пара объектных знаков (при

машинной реализации - байтов), или как один особый знак (например, собственный знак рефала), о представлении которого в машине надо еще договориться. Мы будем говорить, что в первом случае текст воспринимается как натуральный код, а во втором случае - как метакод. Полное определение натурального кода, а точнее, истолкования данного текста как натурального кода, таково:

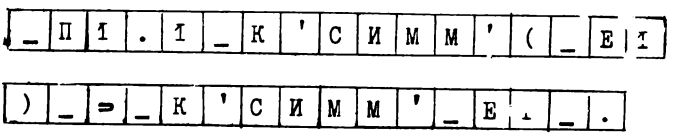
- Если некоторая последовательность знаков воспринимается как натуральный код, то каждый простой (то-есть не подчеркнутый) знак воспринимается как объектный знак, имеющий данное графическое представление, а подчеркнутый знак - как пара объектных знаков: подчеркивание и простой знак.

Теперь мы потребуем, чтобы всякий текст, который мы пишем, мог быть истолкован как натуральный код. Это требование нужно для того, чтобы наши тексты могли быть введены в машину, и вообще, могли рассматриваться как объект работы. Но в то же время мы не намерены менять введенную выше систему записи произвольных рефал-объектов. Мы просто объявили, что все эти записи были записями рефал-объектов в метакоде-А. Если пожелаем, мы можем истолковать их как записи в натуральном коде, тогда они предстанут в виде объектных цепочек, которые можно ввести в машину, переводя каждый объектный знак в соответствующий байт. Впрочем, мы сделаем одно исключение. Так как знак § обычно отсутствует в устройствах ввода-вывода, мы дадим ему другое представление - П (подчеркнутое П). Теоретически мы будем считать это представление основным, а знак § рассматривать как полный его эквивалент, удобный при типографском (а также рукописном и машинописном) представлении текста.

Например, предложение

§1.1 ≤'симм' (Е1) = ≤'симм' Е1 .

будет вводиться в машину в виде следующей последовательности байтов:



Прежде чем приступить к строгому определению метакода-А, мы внесем некоторые дополнения в натуральный код. Согласно нашему определению, всякий "законный" текст, истолковываемый в натуральном коде, представляет цепочку объектных знаков. Для собственных знаков рефала в натуральном коде не осталось мест. Теперь мы введем их временно как некие "незаконнче" знаки, которые принципиально непредставимы на языке машины, и которые не будут использоваться практически, а служат лишь для внесения ясности. Мы будем изображать собственные знаки рефала в натуральном коде, обводя кружком их изображения в метакоде-А: (≤) , (() и т.п. Ясно, что для практического использования такая система совершенно не пригодна. она служит лишь для создания реперного, базисного представления рефал-объектов. Мысля в натуральном коде, мы можем представлять себе собственные знаки как обведенные кружком. До тех пор, пока мы не объявили все наши прежние записи на рефале записями в метакоде-А, мы воспринимали их как "натуральный код", в котором используется десять выделенных знаков - собственных знаков рефала. Теперь мы не имеем права воспринимать эти записи как натуральный код, ибо, например, круглая скобка согласно нашему определению должна будет истолковываться как объектный знак с графическим видом "(", а не как собственный знак рефала, но если мысленно, обвести все собственные знаки

кружками, то такое восприятие снова станет законным. Мысленное обведение собственных знаков - это алгоритм перевода метакода-А в натуральный код.

Полное определение метакода-А дано в таблице 3.I. Проще всего было бы изображать все собственные знаки в метакоде-А подчеркнутыми знаками. Однако мы сделали исключение для рефал-кавычки (символьной скобки) и объектных скобок, ибо они встречаются очень часто и их смысл в рефале соответствует общепринятому, поэтому подчеркивать их было бы неудобно. Итак, семь собственных знаков рефала изображаются в метакоде-А подчеркнутыми знаками, то-есть парой объектных знаков, а три собственных знака: ⓞ , ⓒ и ⓒ - одним объектным знаком. Из требования однозначности обратного преобразования следует, что три объектных знака: ' , (и) , являющиеся изображениями собственных знаков, должны иметь в метакоде отличные от них изображения. То же относится и к самому знаку подчеркивания, иначе, например, пара объектных знаков _ K осталась бы неизменной при переводе в метакод, а при обратном преобразовании превратилась бы в знак конкретизации. Поэтому все четыре объектных знака изображаются в метакоде подчеркнутыми знаками. Остальные объектные знаки не претерпевают изменений при переходе в метакод.

Мы сосредоточили внимание на сопоставлении метакода и натурального кода как разных способов изображения одного и того же рефал-объекта. Этот аспект выражается сравнением первого и второго столбцов таблицы 3.I. Однако не надо забывать, что первичный смысл введения метакода - это определение метакодового преобразования, отображающего множество всех рефал-объектов на некоторое его подмножество, содержащее лишь объектные цепочки (но не все объектные

Таблица 3.1 Метакод-А

Натуральный код	Метакод-А, представленный в натуральном коде		Метакод-А, представленный в метакоде-А
	на бумаге	в машине	
§	П	_ П	V П
=	⊃	_ :'	V ⊃
S	S	_ S	V S
W	W	_ W	V W
E	E	_ E	V E
K	K	_ K	V K
.	.	_ .	V
'	'	'	Q
(	(	(	L
)	)	)	R
'	Q	_ Q	V Q
(	L	_ L	V L
)	R	_ R	V R
	V	_ V	V V
A	A	A	A

цепочки). Преобразование, задаваемое метакодом-А, будем обозначать через μ_A Обратное преобразование будем обозначать через μ_A^{-1} преобразование μ_A , примененное дважды, через μ_A^2 и т.д.

Любое преобразование можно описать, пользуясь любым способом изображения объектов. Преобразование μ_A описывается в натуральном коде следующим образом: берем заданный рефал-объект (записанный в натуральном коде), преобразуем его в метакод-А, пользуясь первым и вторым столбцом таблицы 3.I, и воспринимаем результат как натуральный код. Машинное восприятие второго столбца представлено в третьем столбце таблицы 3.I, поэтому преобразование μ_A наиболее отчетливо предстанет как переход от столбца I к столбцу 3.

Преобразование μ_A можно описать и в метакоде-А. Для этого и начальную, и конечную стадию описанного выше преобразования надо изобразить в метакоде-А. Это преобразование описывается переходом от столбца 2 к столбцу 4 в таблице 3.I. Четвертый столбец, если рассматривать его как натуральный код, получается из первого столбца преобразованием μ_A^2

Теперь мы можем сформулировать алгоритм преобразования μ_A^{-1} и убедиться в его однозначности. Это преобразование выполняется следующим образом. Выбирается очередной знак текста. Если это не подчеркивание, то он преобразуется путем движения справа налево в таблице 3.I. (столбец 3 - столбец I при записи в натуральном коде, столбец 4 - столбец 2 при записи в метакоде-А). Если это подчеркивание, то выбирается следующий знак и полученная пара преобразуется с помощью таблицы 3.I таким же образом.

Отношение к нашим текстам на рефале как к метакоду облегчает использование нескольких различных обозначений для одного собствен-

ного знака рефала: преобразование $\mathcal{M}_A^{-1}$ которое подразумевается, каждое из них переводит в один и тот же собственный знак. Мы уже говорили об эквивалентности знака § и пары знаков П. Введем еще варианты для трех собственных знаков (таблица 3.I.I)

Таблица 3.I.I Варианты метакода-А

П	≥	S	W
§	~	C	T

Изображения указателей переменных C и T удобны, когда текст на рефале печатается на машинке с русским алфавитом. В качестве основного и единственного (подобно E) это обозначение не принято потому, что в рукописном тексте подчеркнутое T слишком похоже на латинское I

Основным (фактически, единственным) способом изображения рефал-объектов в настоящей книге будет (и был) метакод-А. Он же принимается за основу при выполнении функций обмена 'П' и '≥'. Исключение будет представлять лишь следующий подраздел, где мы рассмотрим альтернативный метакод. Следовательно, основными столбцами в таблице 3.I, определяющими преобразование $\mathcal{M}_A$ являются второй и четвертый столбцы. Натуральный код мы решительно и бесповоротно изгоняем из записей рефал-текстов на бумаге, так что, например, круглые скобки (и) , всегда будут изображать рефал-скобки, а объектные знаки, которые имеют графические изображения (и) , будут изображаться подчеркнутыми знаками L и R
В то же время запись в метакоде-А можно воспринимать таким образом, что вовсе не замечать ее метакодового характера. Прежде всего,

можно мысленно обвести кружком собственные знаки и преобразовать выделенные объектные знаки $\underline{Q}$, $\underline{L}$, $\underline{R}$, $\underline{V}$ к их графическому виду:

( ), _ . Тогда мы получим запись рефал-объекта в натуральном коде . Однако, это слишком большая нагрузка для воображения, она имеет смысл лишь в трудных ситуациях, когда необходимо начинать "от печки" - от базисного представления в натуральном коде. Нормальным способом восприятия записи в метакоде-А является способ, который мы назовем восприятием записи как натурального вида рефал-объекта. Чтобы воспринимать так запись в метакоде-А, достаточно рассматривать подчеркнутый знак как один знак, и, конечно, запомнить список собственных знаков. Правда, при этом четыре выделенных объектных знака записываются не совсем так, как они выглядят при печати их машиной, но это не имеет большого значения; это различие можно отнести к перекодировке графических изображений знаков при реализации рефала. Да и используются эти знаки не слишком часто: в тех лишь случаях, когда сам объект работы является записью на метакоде. До тех пор пока мы не ввели понятие метакода, мы воспринимали записи в метакоде-А именно как натуральный вид рефал-объектов, а с подчеркнутыми объектными знаками просто не встречались, поэтому не было случая провести различие между натуральным видом и натуральным кодом.

Итак, натуральным кодом, в том точном смысле, который был придан этому понятию выше, мы не будем пользоваться при изображении рефал-объектов на бумаге. Однако при обмене информацией с машиной натуральный код сохраняет свое значение наравне с метакодом, и очень важно указать режим обмена: происходит ли он в натуральном коде, или в метакоде. На этом вопросе следует остановиться подробнее.

Применяя преобразование μ_A к объектной цепочке, мы получаем, вообще говоря, другую цепочку. Это, несомненно, является неприятностью. Ведь метакод строится для преобразования рефал-объектов, содержащих собственные знаки, в объектные цепочки. Хотелось бы, чтобы объектные цепочки, поскольку они не содержат собственных знаков, не изменялись при этом преобразовании. Быть может, тот факт, что они изменяются - это недостаток нашего метакода, и его можно исправить?

Легко видеть, что это не так. Любое метакодовое преобразование μ, обладающее однозначным обратным преобразованием μ^{-1}, будет обладать этим свойством. Действительно, допустим, что μ преобразует любую объектную цепочку $< \mathcal{C} >$ в самое себя. Возьмем выражение $< \mathcal{C}_1 >$, содержащее хотя бы один собственный знак. Тогда результат воздействия μ на $< \mathcal{C}_1 >$, который мы обозначим через $< \mathcal{C}_1 >$, не совпадает с $< \mathcal{C}_1 >$. Применим теперь μ к $< \mathcal{C}_1 >$ Так как $< \mathcal{C}_1 >$ - объектная цепочка, результат будет снова $< \mathcal{C}_1 >$ Таким образом, преобразование μ, примененное к двум разным объектам: $< \mathcal{C}_1 >$ и $< \mathcal{C}_1 >$ дает один и тот же результат, следовательно обратное преобразование неоднозначно.

Итак, нам придется примириться с тем, что преобразования μ_A и μ_A^{-1} вообще говоря меняют объектные цепочки. И следует учесть, что при вводе в машину объектной цепочки необходимо указывать, как её воспринимать: как готовый объект работы, где каждый байт есть объектный знак, или как метакод-А, к которому надо предварительно применить преобразование μ_A^{-1} ? При формировании поля памяти информация, в конечном счете, всегда рассматривается как метакод.

Но при обмене информацией с полем зрения оба типа ввода-вывода могут быть полезными, в зависимости от того, как мы желаем трактовать кавычку и круглые скобки во вводимой информации. Например, если последовательность байтов

(	A	B	C	)

вводится в машину как метакод, то первый и пятый байты порождают в поле зрения звенья-скобки, связанные друг с другом адресами в полях A_1 Если та же последовательность вводится как натуральный код, то эти байты, как и три остальных, порождают звенья-символы, у которых в полях A_1 хранятся эти самые байты.

Задачи

3.1. Совершить преобразование $\mathcal{M}_A$ над набором предложений §§ 4Д.1 - 4Д.3 (см.стр. I/37).

3.2. Совершить преобразование $\mathcal{M}_A^{-1}$ над объектной цепочкой

$$\underline{V}\ K\ \underline{Q}\ \text{ПЕРВСИМ}\ \underline{Q}\ \underline{L}\ A\ B\ C\ \underline{R}\ \underline{V}$$

3.3. Совершить преобразование $\mathcal{M}_A^3$ над выражением $\underline{E}I$.

3.4. Сколько знаков содержит результат применения преобразования $\mathcal{M}_A^n$ к выражению (A)?

3.5. В поле зрения рефал-машины находится выражение

$$\underline{K}\ \varphi\langle\mathcal{E}_1\rangle\langle\mathcal{E}\rangle\langle\mathcal{E}_2\rangle\ .$$

где выражение $\langle\mathcal{E}\rangle$ получено вводом текста

$$\text{'сил'}\ (A+B) + \underline{L}\ \underline{Q}\ K\text{У}\ \underline{Q}\ 4\ \underline{R}$$

рассматриваемого как натуральный код, а относительно $\langle\mathcal{E}_1\rangle$ и $\langle\mathcal{E}_2\rangle$ известно только, что они не содержат подвыражений, тождественных $\langle\mathcal{E}\rangle$.

Описать функцию φ так, чтобы конкретизация приводила к удалению выражения $\langle\mathcal{E}\rangle$.

3.6. То же, если приведенный текст вводится как метакод.

3.7. Значительная часть рефал-компилятора пишется на самом же рефале. Эта программа рассматривает набор предложений, подлежащий компиляции, как натуральный код некоторой объектной цепочки. Предполагая, что пользователю разрешено употреблять оба варианта обозначений для указателей переменных, написать для рефал-компилятора программу перекодировки, приводящую рефал-предложения к такому виду, когда используются только указатели $\underline{S}$ и $\underline{W}$

4. Метакод-Б

Метакод-Б строится на принципе, в извесном смысле противоположном принципу метакода-А. Разграничение собственных и объектных знаков достигается выделением не собственных, а объектных знаков: они заключаются в кавычки. Кавычка в метакоде-Б является выделенным объектным знаком, она не имеет ничего общего с рефал-кавычкой. Рефал-кавычка преобразуется в метакоде-Б в косую черту ("слэш"), которая является рядовым объектным знаком. Такая перестановка, странная на первый взгляд, диктуется стремлением выдержать запись рефал-объектов в метакоде-Б в духе традиций, сложившихся в практическом программировании. Если в метакоде-А запись на рефале близка по стилю к записи математических формул, то в метакоде-Б она приближается по форме к типичному тексту на формализованном языке программирования.

Набор предложений в метакоде-Б можно записать в бланковом и и безбланковом формате. Основной формой, в которой используется метакод-Б, является запись в бланковом формате; текст при этом пи-

мется на специальном бланке для набивки перфокарт. Однако мы начнем с описания безбланкового формата, так как он сохраняет больше связей с натуральным кодом, и рассматривается как однозначно определенный, в то время как в бланковой записи могут быть те или иные вариации, зависящие от конкретной реализации рефала.

На набор предложений, записываемый в метакоде-Б, накладываются следующие ограничения:

I) Предложения не должны содержать комментариев.

2) Составные символы должны быть образованы только из букв и цифр.

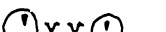
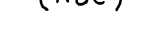
3) Предложения с общим детерминативом должны следовать подряд.

Преобразование $/\!\!/_Б$, переводящее рефал-объект в объектную цепочку метакода-Б, осуществляется следующим образом. Все собственные знаки превращаются в объектные знаки в соответствии с таблицей 3.2. Объектные цепочки, образующие составные символы, и идентификаторы свободных переменных остаются неизменными. Все остальные объектные знаки группируются в цепочки, ограниченные в исходном рефал-объекте с обеих сторон либо собственным знаком, либо идентификатором переменной, либо концом рефал-объекта. Каждая такая цепочка, если она состоит не из одних кавычек, заключается в кавычки, а каждая кавычка, входящая в нее, превращается в пару кавычек. Если цепочка состоит из одних кавычек, то каждая кавычка по-прежнему превращается в пару кавычек, но цепочка в целом уже в кавычки не заключается.

Задачи

3.8. Записать в метакоде-Б набор предложений §§ 4Д.I - 4Д.3 (см. стр. I/37).

Таблица 3.2 Метакод-Б

Натуральный код	Метакод-Б
	#
	⊃
	S
	W
	E
	K
	
	/
	(
	)
	''
	'('
	')'
	'/'
	/XY/
	('AB')
	'(ABC)'
	A'''
	E1S2

3.9. Следующее предложение, записанное в метакоде-Б:

К/Фи/Е1/СИЛ/ ('А+В')' + ("КУ"4)' Е2 ⊃ Е1 Е2

записать в метакоде-А.

3.10. Сколько знаков содержит результат применения преобразования $/M_Б^n$ к объекту, изображаемому в метакоде-А в виде (А)? Сравнить с задачей 3.4.

Основные принципы записи метакода-Б на бланках (детали формата, как говорилось, могут варьироваться) таковы.

1. Знак # опускается. Каждое предложение начинается с новой перфокарты. Если предложение не закончилось на данной перфокарте, в 72-й позиции ставится знак продолжения, и следующая перфокарта рассматривается как продолжение.

2. Знак ⊃ , отделяющий левую часть предложений от правой, опускается. Вместо него пробивается один или несколько подряд идущих пробелов.

3. Знак К, с которого начинается левая часть предложения, опускается.

4. Если в первой позиции пробит пробел, он воспринимается как повторение детерминатива предыдущего предложения.

5. Если в первой позиции пробита звездочка, вся перфокарта воспринимается как комментарий и не влияет на работу машины.

5. Машинные процедуры

При описании абстрактной рефал-машины предполагается, что мы можем непрерывно следить за состоянием поля памяти и поля зрения. При воплощении рефал-машины на реальной вычислительной машине это, разумеется, не так. Поэтому необходимы какие-то процедуры, позволяющие выводить из машины информацию по мере необходимости,

не прекращая работу рефал-транслятора. Может оказаться также полезным вводить какую-то информацию в рефал-машину в процессе её работы. Далее, при обработке больших массивов информации необходимо осуществлять обмен между оперативной памятью и внешними запоминающими устройствами: барабанами, дисками, лентами. Наконец, при выполнении некоторых преобразований (например, арифметических действий) имеет смысл забыть на время о языке рефал и передать управление программе, написанной в кодах вычислительной машины. Все эти действия выполняются при реализации рефала с помощью машинных процедур. Так как машинных процедур может быть сколько угодно, и даже те из них, которые необходимы практически всегда (например, печать), могут иметь различные модификации, ни одна из этих процедур не включена в описание абстрактной рефал-машины. Однако возможность использования машинных процедур является непременной и существенной чертой системы программирования на рефале.

Каждой машинной процедуре, подобно процедуре, описанной на рефале, соответствует определенный составной символ - детерминатив. Предполагается, что транслятор содержит список детерминативов всех задействованных машинных процедур. Машинная процедура выполняется тогда, когда ведущим термом в поле зрения становится терм вида:

$$\leq \ '\langle \mathcal{M} \rangle' \ \langle \mathcal{E} \rangle .$$

где $'\langle \mathcal{M} \rangle'$ - детерминатив данной машинной процедуры, а $\langle \mathcal{E} \rangle$ - произвольное выражение (содержащее необходимую информацию). Таким образом, с учетом наличия машинных операций алгоритм выполнения шага рефал-машины должен быть уточнен следующим образом:

Найдя ведущий знак конкретизации, машина исследует, не является ли непосредственно следующий за ним символ детерминативом

машинной процедуры. Если это так, то выполняется соответствующая машинная процедура, то-есть управление передается на некоторую подпрограмму, заданную набором кодов вычислительной машины. С точки зрения абстрактной рефал-машины выполнение шага этим заканчивается, а дальнейшее преобразование информации и возвращение к нормальному началу следующего шага рефал-машины возлагается на программу машинной процедуры.

Аппарат машинных процедур служит для передачи управления от абстрактной рефал-машины к конкретной вычислительной машине и обратно. Он позволяет сочетать использование рефала с использованием всех других средств программирования, которые имеются в распоряжении на данной машине.

Набор машинных процедур - дело конкретной реализации рефала. Однако мы введем несколько машинных процедур, которые будем считать стандартными и необходимо присутствующими при любой реализации.

Это, прежде всего, машинная процедура печати с детерминативом 'ПЕЧ' . Когда ведущим становится терм

$$\underline{\kappa}\ \text{'ПЕЧ'}\ \langle \mathcal{E}_1 \rangle \underline{.}$$

рефал-транслятор печатает выражение $\langle \mathcal{E}_1 \rangle$ и уничтожает в поле зрения функциональные скобки с детерминативом, оставляя вместо конкретизируемого выражения одно выражение $\langle \mathcal{E}_1 \rangle$. Мы определяем процедуру печати таким образом для того, чтобы в процессе работы рефал-машины можно было печатать выражения, не портя этих выражений. Часто бывает нужна и другая процедура - выводящая выражение на печать и уничтожающая его в поле зрения. Этой процедуре мы присвоим стандартный детерминатив 'ВЫВ' (то-есть "вывести из поля зрения на печать"), и опишем ее на рефале, используя проце-

дуру 'ПЕЧ'

§ M1 К'ВЫВ' Е1 ⩾ К 'СТЕРЕТЬ' К 'ПЕЧ' Е1 . .

§ M2 К'СТЕРЕТЬ' Е1 ⩾

Две важных машинных процедуры с детерминативами 'ЗК' и 'ВК' называемые "закапыванием" и "выкапыванием", предназначены для оперативного присваивания значений переменным и замены переменных на их значения. Обращение к процедуре закапывания имеет вид:

К 'ЗК' < $\mathcal{C}$ > = < $\mathcal{E}$ > .

где < $\mathcal{C}$ > - произвольная последовательность букв и цифр, а < $\mathcal{E}$ > - произвольное выражение. Это обращение может быть прочитано, как "закопать выражение < $\mathcal{E}$ > под именем < $\mathcal{C}$ >". Выполняется эта процедура следующим образом. Звено в поле зрения, предшествующее ведущему знаку К, соединяется со звеном, следующим за парной функциональной точкой, так что всё конкретизируемое выражение исключается из выражения, стоящего в поле зрения. Однако к свободной памяти отходят только звенья, содержащие рефал-объекты К 'ЗК' < $\mathcal{C}$ > = . Звенья, фиксирующие выражение < $\mathcal{E}$ >, остаются нетронутыми; выражение < $\mathcal{E}$ > исчезает из поля зрения абстрактной рефал-машины, но остается в памяти реальной вычислительной машины в том самом месте и в том самом виде, в котором оно было прежде. Объектная цепочка < $\mathcal{C}$ > вместе с адресами начального и конечного звеньев выражения < $\mathcal{E}$ > запоминается в специальной области памяти, отведенной для процедур 'ЗК' и 'ВК' Транслятор как бы закапывает выражение < $\mathcal{E}$ > и забивает над ним колышек, помеченный именем < $\mathcal{C}$ >

Обращение к процедуре выкапывания имеет вид

К 'ВК' < $\mathcal{C}$ > .

и читается: "выкопать последнее выражение, закопанное под именем

<C>" . Выполняется процедура 'ВК' следующим образом. Из области памяти, отведенной для процедур 'ЗК' и 'ВК' выбираются адреса, связанные с последним закапыванием под именем <C> и с их помощью конкретизируемое выражение заменяется на закопанное выражение. Тройка, состоящая из имени <C> и адресов двух концов закопанного выражения, стирается, так что следующее выкапывание с тем же именем <C> даст выражение, закопанное предпоследним. Если от транслятора требуют выкопать выражение, указывая имя, под которым в данный момент ничего не закопано (в частности, было закопано, но уже выкопано), то конкретизируемое выражение заменяется на пустое выражение.

Принципиально, процедуры 'ЗК' и 'ВК' можно целиком описать на рефале, используя лишь те средства, которые входят в Формальное описание. Точнее говоря,это будут процедуры, эквивалентные процедурам 'ЗК' и 'ВК' и использующие обмен информацией между полем зрения и полем памяти. Однако при практическом использовании рефала очень важно, что машинная реализация этих процедур так, как это описано выше, приводит к весьма быстрому их выполнению, не требующему переписи, и даже просмотра, закапываемых и выкапаваемых выражений. Это следует учитывать при составлении программ на рефале. С другой стороны, закопанные выражения продолжают занимать место в памяти, отведенной под поле зрения, и это может вызвать нехватку памяти. Поэтому процедуры 'ЗК' и 'ВК' рекомендуется использовать лишь для оперативного запоминания, а для более долговременного запоминания информации использовать другие машинные процедуры, которые хранят ее в сжатом виде, или выносят на внешние запоминающие устройства.

Пример. Допустим, что нам надо хранить список "выделенных" букв, который время от времени будет использоваться, а также каким-то образом изменяться (например, пополняться). Введем имя СПИСВЫД, по которым будем закапывать этот список. Начальное формирование списка выделенных букв может быть осуществлено конкретизацией:

К'ЗК' СПИСВЫД = АБВГ .

Чтобы приписать справа к списку выделенных букв три буквы КЛМ, надо выполнить конкретизацию

К'ЗК' СПИСВЫД = К'ВК' СПИСВЫД . КЛМ .

Допустим, надо удалить из списка букву В. Для этого сначала определим процедуру (функцию) удаления произвольной буквы 'УДАЛ' В аргументе этой функции будем помещать в скобки выражение, из которого букву надо удалять.

§ К'УДАЛ' $ A (Е1 $ A Е2) ≡ Е1 Е2

§ К'УДАЛ' $ A (Е1) ≡ Е1

Второй параграф относится к случаю, когда нужной буквы в выражении нет.

Удаление буквы В из списка букв достигается конкретизацией

К'ЗК' СПИСВЫД = К'УДАЛ' В (К'ВК' СПИСВЫД .) . .

Поставим задачу описать предикат 'ВЫДЕЛ' определяющий является ли данная буква "выделенной", то-есть входит ли она в СПИСВЫД . Здесь необходима вспомогательная функция 'ПРОВ' , проверяющая, входит ли буква в данное выражение:

§9 К'ВЫДЕЛ' СА ≡ К'ПРОВ' СА(К'ВК' СПИСВЫД .) .

§10.1 К'ПРОВ' СА(Е1 СА Е2) ≡ 'ИСТИНА'
К'ЗК' СПИСВЫД=Е1 СА Е2 .

§10.2 К'ПРОВ' СА(Е1) ≡ 'ЛОЖЬ' К'ЗК' СПИСВЫД=Е1 .

Процедура 'ПРОВ' , после того как она использует список выделенных букв, снова закапывает его под именем СПИСВЫД иначе первое же использование предложений § I0.I или § I0.2 привело бы к безвозвратной утере списка.

Для повышения эффективности работы вычислительной машины можно реализовать как машинные процедуры наиболее простые и часто используемые процедуры, которые первоначально были описаны на рефале. Их описание на рефале удобно сохранить как эталон, служащий для правильного использования этих процедур при составлении програмы на рефале. Человек, имеющий перед глазами этот эталон, может вовсе не знать, выполняются ли данные процедуры в общем порядке или как написанные в машинном коде подпрограммы.

6. Представление чисел в машине

Простейший способ представления чисел при программировании на рефале - это в виде последовательности знаков, изображающих число в десятичной системе счисления, как установлено, например, в алголе. Напомним эту систему записи, приведя несколько примеров, понятных без пояснений:

$$1971$$
$$-1$$
$$0.000018287036$$
$$2_{10}-16$$
$$-15.25_{10}20$$

(Указатель основания степени $_{10}$ представляет один знак).

При таком представлении число, с точки зрения рефал-транслятора, ничем не отличается от любой другой последовательности объектных знаков. Можно описать на рефале процедуры, выполняющие арифметические

действия над такими числами, можно реализовать их как машинные процедуры. Удобно объединить их в одну процедуру, которой мы присвоим детерминатив 'ВЧСЛ' - вычислить. Как уже отмечалось, при программировании на рефале не имеет значения, как реализуется эта процедура. Условимся непосредственно за детерминативом помещать знак операции (один из пяти: +, -, x, /, ↑) и отделять операнды друг от друга запятой. Например, выполнение конкретизации

К'ВЧСЛ' + - I7, 0.25.

даст выражение

- I6.75

Этот способ представления чисел (который мы будем называть алгольным) весьма расточителен и в отношении памяти машины, и в отношении времени, затрачиваемого на арифметические действия. Так, число I97I будет занимать четыре полных звена. При действиях над так представленными числами надо по очереди выбирать знаки из памяти, после чего либо моделировать десятичную арифметику, либо тратить время на перевод в двоичную систему и обратно.

Недостатки алгольного представления устраняются в специальном представлении чисел, когда поле A_1 звена, изображающего число, содержит двоичный код данного числа, готовый к непосредственному использованию машиной. Мы покажем, как это осуществляется, на примере машин с системой команд IBM-360 и ограничиваясь лишь натуральными числами (фиксированная запятая), ибо в тех задачах, для которых используется рефал (аналитические выкладки и т.п.), как правило, используются целочисленная, а не десятичная арифметика.

Звено в машине IBM-360 занимает 8 байтов. На поля A_2 и A_3 уходит по два байта, на поле A_1 вместе с полем признака остается, следовательно, четыре байта - 32 двоичных разряда. Если звено изображает

число, то значение первого (слева) разряда поля признака есть нуль. Кодировка признаков устроена таким образом, что только звено-число и имеет в этом разряде нуль, остальные типы звеньев содержат в нем единицу. Поэтому оставшиеся разряды поля признака (которое нормально занимает целый байт) используются под код числа. Так как в машине IBM-360 первый разряд чисел с фиксированной запятой кодирует знак числа, причем код 0 соответствует знаку плюс, запись числа в первых четырех байтах звена (одно слово) полностью совпадает с тем видом записи, который требуется системой команд. Действия над числами, умещающимися в одном звене, выполняются как одна машинная операция. Так как одно звено может изображать числа от 0 до $2^{31} - 1 =$ $= 2.147.483.647$, максимальная плотность хранения числовой информации в десять раз больше, чем при алгольном представлении, и только вдвое меньше, чем если бы вся память была забита цифрами. Неудобством этого представления является необходимость перевода чисел из одной системы исчисления в другую при вводе и выводе информации. Звено-число изображается на письме как обычное десятичное число, взятое в рефал-кавычки, например '1971'. Такие составные символы мы будем называть числовыми символами. Синтаксически они являются символами, а от символов-меток отличаются тем, что за первой кавычкой следует не буква.

Числа, бо́льшие чем $2^{31} - 1$, представляются в поле зрения последовательностью из нескольких звеньев-чисе ,'а в тексте на рефале - последовательностью числовых символов, подобно тому как в алгольном представлении записываются числа, большие девяти. Каждое предыдущее звено, входит с весом в 2^{31} раз бо́льшим, чем последующее. Иначе говоря, большие числа представляются в 2^{31}-ичной системе счисления.

При написании программы на рефале бывает полезно отличать числовые символы от других символов. Это достигается использованием рекурсивных переменных со спецификатором 'NS' . В принципе, можно считать, что функция 'NS' описывается набором $2^{31}+1$ предложений:

```
§ К 'NS' '0'  ≡ ('0')
§ К 'NS' '1'  ≡ ('1')
§ К 'NS' '2'  ≡ ('2')

§ К 'NS' '2147483647'  ≡  ('2147483647')
§ К 'NS' S A  ≡ ¬ S A
```

В действительности же, конечно, функция 'NS' реализуется как машинная функция (процедура), проверяющая значение первого двоичного разряда звена.

Используя свободную переменную S 'NS' <a> , можно описать на рефале функцию 'N' отщепляющую от начала заданного выражения (аргумента) натуральное число в специальном представлении, то-есть максимальную последовательность числовых символов:

```
§ II.1 К 'N' S 'NS'1 E1  ≡ К 'CONTN'(S1) E1.
§ II.2 К 'N' E1  ≡ ¬ E1
§ I2.1 К 'CONTN' (E1) S 'NS'2 E3  ≡ К 'CONTN' (E1 S2) E3.
§ I2.2 К 'CONTN' (E1) E2  ≡ (E1) E2
```

Используя эту функцию, можно записывать свободную переменную натурального числа в виде E 'N' <a>.

Отрицательные целые числа мы будем изображать, как обычно, помещая перед натуральным число знак минус. Для действий над целыми числами в специальном представлении введем функцию 'ВЧЦ' - вычисления с целыми, формат которой определим таким же, как формат функции 'ВЧСЛ' . Заметим, что число нуль в качестве аргумента этой функции

должно фигурировать в виде числового символа '0' а не знака 0 (то же относится, конечно, и к любому числу, выражаемому одной цифрой). Операция деления, примененная к целым числам, понимается как деление с остатком. Она дает пару чисел, разделенных запятой: неполное частное(на первом месте) и остаток. Например, результатом конкретизации

```
K 'ВЧЦ' /'I84' ,'I5' .
```

будет

```
'I2' '4'
```

Формальное определение процедуры деления с остатком таково. Результат деления n на m есть пара целых чисел k, r такая, что

$$n = km + r$$

причем $0 \le r < |m|$. Возведение в степень определяется только при неотрицательном показателе.

Пример. Описание процедуры приведения подобных членов — 'ПРИВ' - может содержать предложение

```
§ K 'ПРИВ' E 'N' 1 S A + E 'N' 2 S A =
                K 'ВЧЦ' + E 1 , E 2 . S A
```

При использовании этого предложения для конкретизации

```
K 'ПРИВ' 'I9I7' y + '67' y
```

Получаем

```
'I984' y
```

Процедура 'N' может быть также реализована в виде машинной процедуры, и распознавание чисел может быть введено в процесс отождествления, тогда оно будет выполняться с максимальной эффективностью. Имея рефал-транлятор, обладающий такими возможностями, мы как бы пользуемся языком, в котором с самого начала для чисел введен особый синтаксический тип переменных. В то же время описание алго-

ритма продолжает оставаться абсолютно законным с точки зрения абстрактного рефала, и оно может использоваться другим рефал-транслятором, в котором таких машинных процедур не предусмотрено. Алгоритм будет выполняться совершенно правильно, хотя и с меньшей эффективностью.

Подобно тому как мы ввели для целых чисел специальный способ представления информации, специальные машинные процедуры и синтаксические типы, такие специальные средства могут быть предусмотрены и для десятичных чисел, и вообще, для объектов любой природы. Такая специализация рефал-транслятора может быть осуществлена ради повышения эффективности без всякого изменения текста на рефале, или с незначительными изменениями, затрагивающими лишь процедуры нижнего уровня иерархии.

7. Разбиение на модули

Для описания на рефале сложного алгоритма (рекурсивной функции) обычно вводится некоторое число вспомогательных функций, каждая из которых имеет свой детерминатив. При одновременном использовании (то-есть загрузке в поле памяти) нескольких таких функций может оказаться, что детерминативы двух вспомогательных функций, задуманных как различные, случайно совпадут. Это, конечно, нарушит правильное выполнение алгоритмов. Возникает проблема согласования обозначений. При описании рефала на абстрактном уровне предполагается, что возможные конфликты такого рода разрешаются путем переобозначения, реального или воображаемого. Это типично математический подход. Для программиста, имеющего дело с большими массивами информации, переобозначение - это всегда реальная и чрезвычайно неприятная процедура, которой надо избегать. Поэтому в формализованных языках программирования

устанавливаются определенные правила истолкования обозначений, введенных в разное время и, возможно, разными программистами. Есть две системы истолкования обозначений: модульная и блочная. При реализации рефала используется модульная система.

Набор предложений, загружаемый в поле памяти рефал-транслятора, представляется в виде последовательности рефал-модулей (сокращенно, рефмодулей). Каждый рефал-модуль есть набор предложений, которые связаны друг с другом в некотором смысле теснее, чем с предложениями из других модулей. А именно, если в состав модуля входит группа предложений с некоторым детерминативом $\langle \varphi \rangle$, то встречая детерминатив $\langle \varphi \rangle$ в правой части предложения, мы относим его именно к этой группе предложений, если даже в полном наборе предложений, находящихся в поле памяти, есть предложения с детерминативом $\langle \varphi \rangle$, встречающиеся раньше этой группы. И только в том случае, если в данном модуле предложений с детерминативом $\langle \varphi \rangle$ нет, мы обращаемся к другим модулям. Говоря более формально, при подстановке правой части предложения в поле зрения рефал-транслятор для каждого подставляемого детерминатива запоминает, из какого рефмодуля было взято предложение, и когда детерминатив оказывается в положении, следующем за ведущим знаком К, в первую очередь просматриваются предложения, образующие его собственный модуль. Если среди них оказывается хотя бы одно предложение с совпадающим детерминативом, остальные модули игнорируются; в противном случае просматриваются предложения остальных модулей в том порядке, в котором модули расположены в поле памяти. Из этого соглашения следует, что в начальном состоянии поля зрения должны присутствовать только детерминативы, допускающие однозначную трактовку.

Разбиение программы на рефмодули осуществляется с помощью предложений-комментариев, таким же образом вводится в машину дополнительная информация, потребность в которой возникает при использовании модульной системы. Предложения-комментарии могут содержать также другую информацию, связанную с используемой системой программирования. Конкретный способ задания всей этой информации в предложениях-комментариях зависит всецело от реализации.

Модульная система дает программисту возможность вводить вспомогательные функции, не заботясь об уникальности детерминативов. Для этого достаточно объединить все вспомогательные функции с основной функцией в один рефмодуль.

С О Д Е Р Ж А Н И Е

Часть I ФОРМАЛЬНОЕ ОПИСАНИЕ ЯЗЫКА	3
I.Синтаксис	3
2.Метакодовое преобразование	6
3.Рефал-машина	6
4.Алгоритм синтаксического отождествления	8
Часть 2 ПРИНЦИПЫ РЕАЛИЗАЦИИ РЕФАЛА	I8
I.Размещение информации в поле зрения	I9
2.Объектные знаки и составные символы	30
3.Метакод-А	32
4.Метакод-Б	43
5.Машинные процедуры	46
6.Представление чисел в машине	52
7.Разбиение на модули	57

№ Т-12610 от "20" VII 1971 г. Заказ № 790 Тираж 250 экз.

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