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

Препринты ИПМ • Препринт № 48 за 1971 г.

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

В.Ф. Турчин

Программирование на языке РЕФАЛ. 4. Использование рекурсивных переменных в языке Рефал

Рекомендуемая форма библиографической ссылки: Турчин В.Ф. Программирование на языке РЕФАЛ. 4. Использование рекурсивных переменных в языке Рефал // Препринты ИПМ им. М.В.Келдыша. 1971. № 48. 48 с.
https://library.keldysh.ru/preprint.asp?id=1971-48

ОРДЕНА ЛЕНИНА
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
АКАДЕМИИ НАУК СССР

В.Ф. Турчин

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ РЕФАЛ

IV Использование рекурсивных переменных в языке рефал

Препринт № 48 за 1971 г

Москва

ОРДЕНА ЛЕНИНА ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ

АКАДЕМИИ НАУК СССР

В.Ф.ТУРЧИН

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ РЕФАЛ

4. ИСПОЛЬЗОВАНИЕ РЕКУРСИВНЫХ ПЕРЕМЕННЫХ В ЯЗЫКЕ РЕФАЛ

М о с к в а , 1 9 7 1 г.

В настоящем выпуске мы снимем первое из ограничений базисного рефала - ограничение свободных переменных базисными свободными переменными. Использование рекурсивных переменных требует более сложного рефал-транслятора, но зато значительно увеличивает изобразительную силу языка. Оно позволяет, в частности, отделить описание синтаксиса языковых объектов от описания их семантики - правил преобразования.

I. Предикативные переменные

Пусть надо просмотреть алгебраическое выражение и разделить его на аддитивные члены, то-есть подвыражения основного уровня скобочной структуры, разделенные знаками + и - . Результат разделения надо зафиксировать, заключив аддитивные члены в скобки.

Если бы не было знака -, и аддитивные члены отделялись бы друг от друга только знаком +, решение можно было бы записать очень просто:

```
§ K φ E1 + E2 = (E1) + K φ E2 .

§ K φ E2 = (E2)
```

Если мы остаемся в рамках базисного рефала, то наличие второго разделителя - знака минус, заметно усложняет задачу. Используя же рекурсивные переменные, мы можем и во втором случае описать функцию φ по аналогии с первым случаем. Для этого введем синтаксическое понятие знак аддитивной операции, определив его с помощью стандартного рефал-предиката 'АДДОП' :

§ 46.I К АДДОП' + ≡ (+)

§ 46.2 К 'АДДОП' - ≡ (-)

§ 46.3 К 'АДДОП' ⊆1 ≡ ¬ ⊆1

Теперь задача решается предложениями:

§ 47.I К φ Е1 ⊆ 'АДДОП' А Е2 ≡ (Е1) ⊆ А К φ Е2 .

§ 47.2 К φ Е2 ≡ (Е2)

Можно ввести более общий предикат, который устанавливает, не является ли проверяемый символ одним из данного списка символов. Присвоим этому предикату детерминатив 'ИЗ' Согласно форме использования предикатов в рекурсивных переменных проверяемый символ должен стоять в конце аргумента:

§ 48.I К 'ИЗ' Е1 ⊆ А Е2 ⊆ А ≡ (⊆А)

§ 48.2 К 'ИЗ' Е1 ⊆ А ≡ ¬ ⊆ А

Теперь вместо § 47 можем написать

§ 49.I К φ Е1 ⊆ ('ИЗ' + -) А Е2 ≡ (Е1) ⊆ А К φ Е2 .

§ 49.2 К φ Е2 ≡ (Е2)

Предикат 'ИЗ' удобен для реализации в виде машинной операции. Реализовав его таким образом, мы получаем эффективный инструмент для деления символов на синтаксические типы. Заметим, что в соответствии с формальным описанием рефала список символов, фигурирую-

щий в главном вхождении рекурсивной переменной (+ -, в нашем случае) должен при каждой проверке переписываться в поле зрения, а после использования уничтожаться. Рефал-компилятор может избежать этих напрасных действий, записав постоянную часть аргумента предиката в поле зрения один раз, а затем закапывая и выкапывая его по мере надобности. Тем не менее машинная операция будет более эффективна (хотя бы из-за возможности использования кодировки символов). Если список символов длинный, и встречается в левых частях предложений несколько раз, то лучше все-таки завести специальный предикат; описать же его можно через предикат 'ИЗ' . В нашем случае

```
§ К 'АДДОП' ≤A ≡ К 'ИЗ' + - ≤A
```

Рекурсивные свободные переменные символа и терма, использующие стандартные предикаты , можно назвать проверяемыми переменными. Использование в процессе отождествления рекурсивных функций выражается здесь только в проверке того, удовлетворяет ли данный символ или терм необходимым условиям. Рекурсивную переменную выражения можно назвать отщепляемой переменной. Здесь начальным объектом работы является весь еще не отождествленный отрезок выражения, а в результате работы начальная часть этого отрезка отщепляется и становится значением свободной переменной.

Рассмотрим такую задачу. Пусть мы имеем арифметическое выражение, записанное в синтаксисе алгола-60, и надо выделить все идентификаторы, отметив их меткой 'ИД' в начале, и заключив в скобки. Например текст

```
sum1 + 0.5/x
```

должен быть преобразован к виду

$$('\text{ид}'\ sum1) + 0.5/('\text{ид}'\ x)$$

Очевидно, необходимо прежде всего определить синтаксическое понятие идентификатора. В описании алгола-60 идентификатор определяется в бэкусовской нормальной форме следующим образом:

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

Грамматика, заданная в бэкусовской нормальной форме, является, по терминологии математической лингвистики, порождающей грамматикой, то-есть алгоритмом для построения всех правильных объектов данного синтаксического типа (в данном случае, идентификаторов). Кроме порождающих грамматик, известны также распознающие грамматики, то-есть алгоритмы, определяющие является ли предъявленный объект правильно построенным объектом данного синтаксического типа. Рекурсивная функция, нужная нам для решения задачи, не является, конечно, порождающей грамматикой. Однако она не является, строго говоря, и распознающей грамматикой. Скорее, её можно назвать анализирующей грамматикой. Нам надо построить такую функцию - придадим ей детерминатив 'ИД' , - которая определяет, может ли от данного выражения быть отщеплен (с левого конца) идентификатор, и если может, то заключает его в скобки. В случае невозможности она, подобно стандартному рефал-предикату приписывает спереди знак ¬.

Например,

```
<К 'ид' sum1 + 0.5 / x .
```

должно превратиться в

```
( sum1 ) + 0.5 / x
```

а

```
<К 'ид' + 0.5 / x
```

- в

```
¬ + 0.5 / x
```

Итак, распознается не принадлежность объекта к данному типу, а возможность отщепления объекта данного типа, которая одновременно вызывает и само отщепление. Это важный момент, который необходимо твердо запомнить.

Определив рекурсивную переменную выражения таким образом, мы ограничиваемся лишь теми грамматиками, которые допускают синтаксический анализ слева направо без возвратов (точнее говоря, мы ограничиваем синтаксические типы, изображаемые непосредственно свободными переменными, ибо на рефале, разумеется, можно описать любой алгоритм анализа, включающий возвраты). Однако это ограничение совершенно не обременительно, так как обычно только такие грамматики и используются на практике при создании искусственных языков. С другой стороны, анализирующие функции ближе к тем интуитивным понятиям, с которыми мы подходим к восприятию текста на формализованном языке. Например, при чтении текста

```
sum1 + 0.5 / x
```

мы не склонны воспринимать объекты *з*, *зи* и *зит* как идентификаторы (хотя они формально являются таковыми), а пробегаем взглядом до знака + и видим сразу идентификатор *зит1*. Это связано с тем, что семантически важен именно анализ всего текста, а не формальная принадлежность того или иного подвыражения к тому или иному множеству.

Наконец, анализирующий алгоритм включает в себя распознающий алгоритм как частный случай: можно определить анализирующий алгоритм так, чтобы он давал положительный ответ только в том случае, когда отщепляемая часть совпадает со всем аргументом.

Для описания функции 'ИД' воспользуемся синтаксическими понятиями (предикатами) 'БУКВА' и 'ЦИФРА' Если аргумент начинается с буквы, эту букву сразу можно отнести к идентификатору, а затем добрать остаток - последовательность букв и цифр. Результат в этом случае положителен. Если первый знак - не буква, результат отрицательный.

§ 50.I К 'ИД' С 'БУКВА' А Е1 ≡ К 'ДОБОР' (С А) Е1 .

§ 50.2 К 'ИД' Е1 ≡ ¬ Е1

§ 5I.I К 'ДОБОР' (Е И) С 'БУКВА А Е1 ≡
К 'ДОБОР' (Е И С А) Е1 .

§ 5I.2 К 'ДОБОР' (Е И) С 'ЦИФРА' А Е1 ≡
К 'ДОБОР' (Е И С А) Е1 .

§ 5I.3 К 'ДОБОР' Е1 ≡ Е1

Теперь процедуру выделения идентификаторов можно записать так:

§ 52.I К 'ВЫДИД' Е1 Е 'ИД' И Е2 ≥ К α Е1 .

('ИД' Е И) К 'ВЫДИД Е2 .

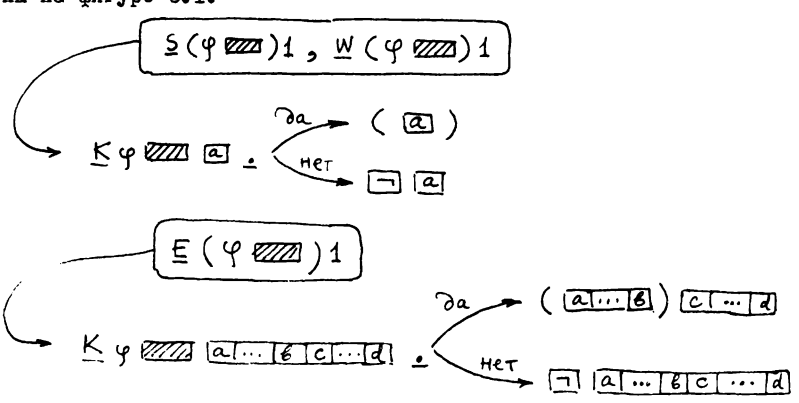
§ .2 К 'ВЫДИД Е1 ≥ К α Е1 .

§ 53 Процедура α вносит процедуру 'ВЫДИД' в скобки .

.I К α Е1 (Е2) Е3 ≥ Е1 (К 'ВЫДИД' Е2 .) К α Е 3 .

§ .2 К α Е1 ≥ Е1

Рекурсивные переменные, которые мы приводили в этом разделе в качестве примеров, обладают тем свойством, что их использование, подобно использованию одних базисных переменных, не меняет конкретизируемого выражения в процессе отождествления. Такие рекурривные переменные мы будем называть предикативными. Чтобы рекурсивная переменная была предикативной, надо чтобы соответствующая рекурсивная функция обладала определенными свойствами, которые схематически изображены на фигуре 5.I.



Фиг. 5.I Предикативные переменные

Несколько слов об использовании непредикативных переменных будет сказано в конце главы.

Задачи

5.1. Описать функцию 'ДОБОР' введя синтаксическое понятие 'НЕБЦ' - не буква и не цифра, и выразив его через понятия 'БУКВА' и 'ЦИФРА'

5.2. В языке алгол-60 понятие числа имеет следующее синтаксическое описание (сделаны очевидные сокращения):

<целбеззн> ::= <цифра> | <целбеззн> <цифра>
<целое> ::= <целбеззн> | + <целбеззн> | - <целбеззн>
<правдробь> . <целбеззн>
<порядок> 10 <целое>
<десчисло> ::= <целбеззн> | <правдробь>
| <целбеззн> <правдробь>
<числобеззн> ::= <десчисло> | <порядок> | <десчисло><порядок>
<число> ::= <числобеззн> | +<числобеззн> | -<числобеззн>

Преобразовать это описание, введя понятие

<знак> + | | <пусто>

и в соответствии с ним описать функцию отщепления числа от начала выражения с детерминативом 'ЧИСЛО' рассчитанную на образование рекурсивной переменной Ε 'ЧИСЛО' <а> .

2. Опорный узел рекурсивной переменной выражения

Ближайший справа к главному вхождению рекурсивной переменной выражения узел, который уже спроектирован в момент проектирования этой переменной, называется её опорным узлом. Отрезок конкретизируемого выражения, вводимый в аргумент функции φ, имеет своим левым концом проекцию левого конца рекурсивной переменной, а правым концом - проекцию опорного узла. Поэтому для правильного понимания процесса отождествления очень важно для каждой рекурсивной переменной выражения (точнее, ее главного вхождения) помнить, где расположен ее опорный узел.

Рассмотрим такую левую часть предложения:

$$\leq \varphi \; \underline{\text{Е}}1 \; \underline{\text{Е}}\text{'ид' и} \downarrow \text{с } 55 \geq$$

Здесь точкой и стрелкой отмечен опорный узел переменной Е'ИД'И. Пусть конкретизируемое выражение имеет вид:

$$\leq \varphi \; z + a b c \; 55$$

В процессе отождествления сначала будет спроектирована цепочка с 55. Затем открытой переменной ЕI будет придано пустое значение и начнется процесс отщепления идентификатора путем конкретизации выражения

$$\leq \text{'ид } z + a b$$

В результате получится

$$(z) + a b$$

Переменная ЕИ должна была бы теперь получить значение z но это противоречит тому, что ее правый конец уже спроектирован на узел между символами b и c. Следовательно, производится перестройка - удлинение значения ЕI, которое производится до тех пор, пока ЕI

не принимает значения Z+, а E И - значения ab. Отождествление успешно выполнено.

Теперь допустим, что мы модифицировали левую часть следующим образом

$$\underline{K}\,\varphi\,\underline{E}1\;\underline{E}\;'\text{ИД}'\,И\,c\,55\;\underline{E}2^{\blacktriangledown}\underline{=}$$

рассчитывая выделить первый идентификатор, за которым следует цепочка с55. Однако, расчет оказывается неверным из-за перемещения опорной точки переменной E 'ИД' И. Пусть конкретизируемое выражение имеет вид:

$$\underline{K}\,\varphi\,Z+a\,b\,c\,55+u\;\underline{.}$$

Процесс отождествления начинается с набора открытой переменной EI. После двукратного удлинения рефал-машина начинает отщепление идентификатор с символа a

$$\underline{K}\;'\text{ИД}'\;a\,b\,c\,55+u\;\underline{.}$$

Однако результатом будет

$$(a\,b\,c\,55)+u$$

Цепочка с55 оказывается включенной в состав переменной E 'ИД' И, "проглоченной" ею, почему в конечном счете отождествление будет найдено невозможным.

Легко сообразить, что если в левую часть предложения входит комбинация E 'ИД' И $<C>$ E1, где $<C>$ - любая непустая цепочка, состоящая из букв и цифр,*) то отождествление будет заведомо невозможным, какой бы вид ни имело конкретизируемое выражение. Действительно, цепочка $<C>$ не может примыкать к идентификатору, иначе она была бы неминуемо "проглочена" им! Эта парадоксальная ситуация на первый взгляд кажется неприятностью, ограничивающей использование рекурсив-

*) a E1 представляет главное вхождение переменной

ных переменных. Однако в этом ограничении мы увидим глубокий смысл, если снова вспомним, что рекурсивные переменные выражения подразумевают не алгоритм распознавания (подобно рекурсивным переменным символа и терма), а алгоритм анализа. Поэтому такую переменную надо всегда воспринимать в комбинации с её опорным узлом. За опорным узлом может быть все, что угодно. Но элементы, стоящие перед опорным узлом, становятся объектами работы дважды: в процессе анализа и при продолжении отождествления. Это порождает своего рода автоматический контроль. Если бы мы захотели выделить первый идентификатор, за которым следует знак возведения в степень ↑ то достаточно было бы написать левую часть:

$$\leq \varphi E1 \in \text{'ид'} \text{ и } \uparrow E2 \geq$$

Но заменяя знак ↑ на цепочку $< C >$, мы совершаем, в сущности, синтаксическую ошибку, ибо согласно нашему определению понятия 'ИД' как алгоритма отщепления идентификатора, цепочка $< C >$ должна быть отнесена к значению рекурсивной переменной.

3. Перевод в базисный рефал

Подобно тому, как открытые переменные могут быть устранены путем введения вспомогательных функций, могут быть устранены и рекурсивные переменные. Таким образом программа, использующая рекурсивные переменные, переводится в базисный рефал. Общий принцип перевода - такой же, как и при устранении открытых переменных: взятие переменных в скобки и описание действий над ними, которые совершаются в процессе отождествления. При этом может оказаться необходимым устранить и некоторые открытые базисные переменные.

Рассмотрим в качестве примера перевод в базисный рефал следующей программы:

§ 54.I К φ Е1 Е 'иД' А + Е 'иД' В Е2 = Е1 К φ ЕА+ЕВ . К φ Е2.

§ 54.2 К φ Е1 = Е1

Мы будем пользоваться функцией иД' отщепляющей идентификатор, как она описана в предложениях §§ 50, 5I. Перевести это описание в базисный рефал с использованием стандартных предикатов 'БУКВА' и 'НЕБЦ' (см. задачу 5.I) мы предоставим читателю.

Переведем предложения § 54 на базисный рефал "в лоб", руководствуясь общим принципом. Получаем:

§ 55 Заводится сумка для Е1, вначале пустая, и делается попытка отщепить идентификатор ЕА:

К φ Е1 = К α ( ) К 'иД' Е1 .

§ 55А.I Случай успешного отщепления идентификатора ЕА, за которым следует знак +; попытка отщепить идентификатор ЕВ:

К α (Е1)(ЕА)+ Е2 = К β (Е1)(ЕА) + К иД' Е2 . .

§ 55А.2 Случай успешного отщепления идентификатора, за которым не следует знак + ; удлинение Е1:

К α(Е1)(W X ЕА)Е2 = К α(Е1 W X) К 'иД' Е А Е2 . .

§ 55А.3 Случай невозможности отщепления идентификатора ЕА; удлинение Е1:

К α(Е1) – W X Е2 = К α (Е1 W X) К 'иД' Е2 . .

§ 55A.4 То же, что в 55A.3, но удлинение невозможно; используется предложение 54.2

$$\underline{K}\,\alpha\,(\underline{E}1)\neg \;=\; \underline{E}1$$

§ 55B.I Случай успешного отщепления идентификатора $\underline{E}B$; используется правая часть предложения 54.I:

$$\underline{K}\,\beta(\underline{E}1)(\underline{E}A)+(\underline{E}B)\underline{E}2 \;=\; \underline{E}1\;\underline{K}\,\psi\,\underline{E}A+\underline{E}B\;.\;\underline{K}\,\varphi\,\underline{E}2\;.$$

§ 55B.2 Случай невозможности отщепления идентификатора $\underline{E}B$, удлинение $\underline{E}I$:

$$\underline{K}\,\beta\,(\underline{E}1)(\underline{W}\chi\;\underline{E}A)+\neg\;\underline{E}2 \;=\; \underline{K}\,\alpha\,(\underline{E}1\;\underline{W}\chi)\underline{K}\,'\text{нд}'\;\underline{E}A+\underline{E}2\;..$$

Эта программа заставляет рефал-машину совершать в точности те же действия, которые она совершала бы, выполняя программу § 54. Изучая критические программу § 55, мы видим, что некоторые из действий машины явно излишни. Во-первых, согласно § 55A.2 машина, отщепив идентификатор $\underline{E}A$ и не найдя за ним знака +, удлиняет $\underline{E}I$ на один терм (кстати, он фактических всегда является символом, хотя это несущественно) и снова отщепляет идентификатор. Это бессмысленно, так как если идентификатор и отщепится, то за ним снова будет следовать знак, отличный от + . В этой ситуации можно сразу удлинить $\underline{E}I$ на весь идентификатор $\underline{E}A$:

$$\text{§ 55A.2}\;\;\underline{K}\,\alpha\,(\underline{E}1)(\underline{E}A)\underline{E}2 \;=\; \underline{K}\,\alpha\,(\underline{E}1\;\underline{E}A)\underline{K}\,'\text{нд}'\;\underline{E}2\;.\;.$$

(Можно было бы сразу добавить к $\underline{E}I$ $\underline{E}A$ еще один терм из $\underline{E}2$ – тот самый, который не является знаком + , ибо он не является также и буквой , однако это потребует дополнительного предложения для случая, когда $\underline{E}2$ пусто).

Еще один источник бессмысленных действий-предложение 55В.2. Его можно переписать следующим образом:

§ 55В.2 K р (E1)(E A) E2 = K α (E1 E A+) K ИД' E2 . .

Итак, подобно тому как это было в случае открытых базисных переменных, при устранении рекурсивных переменных можно устранить и источники потери эффективности, если они присутствовали в исходной записи. Рефал предоставляет возможности для разделения двух аспектов программирования: принципиально решения задачи и повышения эффективности программы. Краткая и выразительная форма записи, включающая, если необходимо, несколько открытых или рекурсивных переменных выражения, имеет все преимущества на начальной стадии составления и отладки программы, а также как "человечески ориентированный" документ. Когда задача принципиально решена, можно заняться преобрезованием программы в целях повышения ее эффективности. При этом значительная часть работы может быть выполнена с помощью машины же. Алгоритм устранения открытых и рекурсивных переменных может быть описан на рефале. Введя в машину программу в краткой форме записи, (например, § 54), программист сможет получить безошибочный текст этой же программы в развернутой форме, (§ 55) и устранять источники неэффективности, как было показано выше. Это - минимальный уровень автоматизации. Однако, вполне вероятно, что можно разработать достаточно мощные алгоритмы устранения неэффективности без участия человека.

4. Просмотры справа налево

Мы связываем рекурсивную переменную выражения с процедурой отщепления подвыражения от левого конца выражения. Это всегда

верно в том смысле, что результатом применения функции φ фигурирующей в главном вхождении переменной будет $(\langle \mathcal{E}_1 \rangle)\langle \mathcal{E}_2 \rangle$, а не $\langle \mathcal{E}_1 \rangle(\langle \mathcal{E}_2 \rangle)$, и значением переменной станет подвыражение $\langle \mathcal{E}_1 \rangle$ а не $\langle \mathcal{E}_2 \rangle$ однако это не исключает той возможности, что конкретизация функции φ производится путем просмотра не слева направо, а справа налево. Эта возможность дает способ краткой записи алгоритмов, включающих просмотры справа налево.

Пусть надо в заданном выражении найти последний (на основном уровне скобочной структуры) знак аддитивной операции: + или − . Для этого введем функцию с детерминативом $'R'$, которая отщепляет выражение до последнего знака из заданной последовательности знаков. Эту последовательность, в соответствии с форматом рекурсивных переменных, будем помещать в первую сумку аргумента. Например, результатом концентрации

$$\underline{K}\,'R'(+-)\,a+b-c+d$$

должно быть выражение

$$(a+b-c)+d$$

Выполнять процедуру $'R'$ будем путем просмотра термов справа налево:

§ 56 $\quad \underline{K}\,'R'\,\underline{W}1\,\underline{E}2 \supseteq \underline{K}\,'R1'\,\underline{W}1\,(\underline{E}2)$.

§ 56A.1 $\underline{K}\,'R1'\,(\underline{E}1 \subseteq A\,\underline{E}2)(\underline{E}3 \subseteq A)\underline{E}4 \supseteq (\underline{E}3) \subseteq A\,\underline{E}4$

§ 56A.2 $\underline{K}\,'R1'\,\underline{W}1\,(\underline{E}3\,\underline{W}A)\underline{E}4 \supseteq \underline{K}\,'R1'\,\underline{W}1\,(\underline{E}3)\,\underline{W}A\underline{E}4$.

§ 56A.3 K 'R' W 1 ( ) E 4 = ¬ E 4

Отметим, что найденный символ, мы оставляем вне скобок, поэтому он не войдет в значение рекурсивной перменной. Переменная ⊏ ('R'(+-)) <a> принимает значение до первого справа знака + , исключительно. Теперь наша задача решается левой частью предложения:

K φ E ('R'(+-)) 1 S A E 2 =

После успешного отождествления переменная S A будет спроектирована на последний знак + или - в конкретизируемом выражении.

При создании рефал-транслятора функцию 'R' рекомендуется реализовать как машинную операцию. Тогда простые обратные просмотры, имеющие переменную вида E ('R'(<C>)) <a> будут выполняться так же эффективно, как прямые просмотры с использованием открытой переменной. Это впрочем не снимает неэквивалентности прямого и обратного направлений в более сложных случаях.

Задача 5.3. Описать процедуру 'ПОЛ' перевода алгебраического выражения из обычной формы записи в польскую. Элементарные выражения - символы; знаки операций: +, -, x, /, ↑ Операции + и - - только бинарные.

5. Рекурсия в левой части предложения

До сих пор мы встречались лишь с одним видом рекурсии: когда правая часть предложения, определяющего правило конкретизации некоторой функции φ содержит прямой или косвенный вызов самой этой функции φ . Наличие рекурсивных переменных позволяет осу-

ществить рекурсию в процессе отождествления, используя в левой части предложения - в **спецификаторах** рекурсивных переменных - ту самую функцию φ , которая определяется данным предложением, или другие функции, которые вызывают функцию φ Понятие вызова одной функцией другой функции, введенное в предыдущей главе, мы расширим теперь - с учетом рекурсивных переменных - включив тот случай, когда вызываемая функция входит как детерминатив (первый символ спецификатора) рекурсивной переменной, встречающейся в левой части предложения, определяющего вызывающую функцию.

Покажем, как используется рекурсия в левой части предложения, на примере определения идентификатора. Начнем с того, что несколько изменим классическое определение идентификатора, приведенное выше; почему это необходимо, станет ясно позже. Разделим идентификатор на две части: букву и "хвост" - последовательность (возможно, пустую), состоящую из букв и цифр. Получаем следующий синтаксис, описанный в бэкусовской нормальной форме:

< идентификатор > <буква > < хвост >

<хвост> ::= <буква> < хвост > | <цифра> <хвост> | <пусто>

Теперь будем составлять предложения на рефале, стараясь как можно ближе следовать описанию в бэкусовской форме. При этом тот факт, что в противоположность бэкусовской форме, наша программа - анализирующая, проявляется в следующих двух правилах:

I) Правая часть бэкусовской формы переходит в левую часть рефал-предложения. Альтернативные варианты правой части бэкусовской формы порождают отдельные предложения, относящиеся к данной функции.

2) Так как функции рефала суть функции отщепления определенных объектов, левая часть предложения должна заканчиваться базисной переменной E <a> (произвольное выражение).

Руководствуясь этими правилами, получаем:

§ 57.I K 'ИД' S 'БУКВА' Б E 'ХВОСТ' Х E1 = (S Б E Х) E1

§ 57.2 K 'ИД' E1 = ¬ E1

§ 58.I K 'ХВОСТ' S 'БУКВА' Б E 'ХВОСТ' Х E1 = (S Б E Х) E1

§ 58.2 K 'ХВОСТ' S 'ЦИФРА' Ц E 'ХВОСТ' Х E1 = (S Ц E Х) E1

§ 58.3 K 'ХВОСТ' E1 = ( ) E1

Определение идентификатора - не рекурсивное, оно выражается через два понятия ('БУКВА' и 'ХВОСТ'), расположенные ниже в иерархии функций, ибо они ни прямо, ни косвенно не вызывают функции 'ИД' Определение хвоста содержит два предложения с рекурсией в левой части. Третье предложение не содержит ни рекурсивных переменных в левой части, ни знаков конкретизации в правой части. Оно - концевое. Заметим, что поскольку понятие ⟨хвост⟩ в порождающем (а также в распознающем) варианте включает пустую цепочку, в анализирующем варианте оно никогда не приводит к отрицательному результату: отщепление пустой цепочки возможно всегда, и так как соответствующее предложение имеет наиболее общий вид, оно должно стоять на последнем месте.

Как же работают предложения 58.I и 58.2? Их задачей является

отщепление от заданного выражения "хвоста". В процессе отождествления прежде всего распознается буква или цифра. Затем весь остаток снова отдается функции 'ХВОСТ' для отщепления хвоста. Таким образом, при каждом следующем обращении к функции 'ХВОСТ' её аргумент становится меньше ровно на один символ. Следовательно, рекурсия не может продолжаться до бесконечности. Рано или поздно, эти предложения окажутся неприменимыми и сработает заключительное предложение. Программируя на рефале, вообще, необходимо следить за тем, чтобы при каждом следующем обращении к рекурсивной функции её аргумент, или хотя бы один из подаргументов в случае жесткого формата, становился меньше, чем он был при предыдущем обращении. Это дает гарантию, что программа не зациклится.

Если бы за основу рефал-предложения мы взяли синтаксическое определение, записанное в виде:

⟨хвост⟩ ⟨хвост⟩ ⟨буква⟩

то получили бы предложение

```
§ К 'ХВОСТ' Е 'ХВОСТ' Х З 'БУКВА' Б Е1 = (Е Х З Б) Е1
```

которое заставить рефал-машину без конца повторять одно и то же обращение к функции 'ХВОСТ' Теперь понятно, почему мы не могли взять за основу классическое определение идентификатора: оно породило бы зацикливающуюся программу. Два правила перевода бэкусовской нормальной формы в рефал-предложение надо дополнить еще одним правилом, касающимся предварительного преобразования бэкусовской формы.

3) Бэкусовская нормальная форма должна быть приведена к такому виду, чтобы перед каждым вхождением определяемого понятия ⟨S⟩

в вариант правой части находились хотя бы одно понятие < R >, расположенное в иерархии понятий ниже понятия < S > и никогда не порождающее пустой цепочки.

Используя эти правила, можно описывать синтаксические понятия на рефале в виде, весьма близком к бэкусовской нормальной форме. В то же время надо помнить, что эти правила являются лишь рекомендациями, но отнюдь не строго определенным алгоритмом преобразования. Существует глубокое различие между порождающими грамматиками в бэкусовской форме и процедурами анализа, которые описываются на рефале.

Что же можно сказать об эффективности выполнения алгоритма, описанного с использованием рекурсии в левой части предложения? Приводит ли такая рекурсия к дополнительным затратам времени ?

Ответ на этот вопрос в значительной степени зависит от реализации языка. При каждом последовательном обращении к функции 'ХВОСТ' рефал-транслятор будет тратить определенное время на то, чтобы прервать процесс отождествления и перейти к выполнению конкретизации. Однако, эти потери можно сделать небольшими. Каких-либо принципиальных источников серьезной потери эффективности использование рекурсии в левой части, и вообще, рекурсивных переменных, не порождает.

Более определенно можно оценить влияние рекурсии в левой части на эффективность, если перевести в базисный рефал программу, использующую рекурсию и сравнить её с программой, написанной на базисном рефале специально для данного случая.

Переведем описание функции 'ХВОСТ', § 58.

§ 59.I К'ХВОСТ' S A E1 = К'ХВ1' К'БУКВА' S A . E1 .

§ 59.2 К'ХВОСТ' E1 = ( ) E1

§ 60:I К'ХВ1' ¬ S A E1 = К'ХВ2' К'ЦИФРА' S A . E1 .

§ 60.2 К'ХВ1' (S A) E1 = К'ХВ3' (S A) К'ХВОСТ' E1 .

§ 6I.I К'ХВ2' ¬ S A E1 = ( ) S A E1

§ 6I.2 К'ХВ2' (S A) E1 = К'ХВ3' (S A) К'ХВОСТ' E1 . .

§ 62.I К'ХВ3' (S A)(E1) E2 = (S A E1) E2

На базисном рефале можно написать более эффективную программу. Для этого надо вначале завести сумку для отщепляемого "хвоста", а затем отправлять в нее каждый следующий символ, пока он является буквой или цифрой.

Отличие программы §§ 59÷62 от более эффективной программы состоит в том, что в соответствии с предложениями § 60.2 и § 6I.2 (которые как раз и возникают в результате перевода левых частей предложений § 58.I и § 58.2, содержащих рекурсию) рефал-машина, найдя букву или цифру, заключает её в скобки и отправляется на отщепление остальной части хвоста, которая также заключается в скобки. Слияние этих двух пар скобок в одну пару осуществляется функцией 'ХВ3' Без этого-то слияния и обходится более эффективная программа, сразу помещающая букву или цифру в сумку. Таким образом, при переводе в базисный рефал рекурсия в левой части

выражается в создании лишней пары скобок, необходимой чтобы отграничить рекурсивно меняющийся объект, с последующим уничтожением этой пары скобок. Если, например, конкретизируется выражение

$$\underline{\kappa}\,{}'\text{хвост}'\ a\,l\,2 + b\ .$$

то после всех проверок в поле зрения оказывается выражение:

$$\underline{\kappa}\,{}'\text{хвз}'(a)\,\underline{\kappa}\,{}'\text{хвз}'(l)\,\underline{\kappa}\,{}'\text{хвз}'(2)()+b\quad .$$

которое после трех шагов рефал-машины дает результат

$$(a\,l\,2)+b$$

Итак, на каждый символ, входящий в "хвост", приходится один лишний шаг рефал-машины. Число полезных шагов на символ равно,- если считать, что предикаты конкретизируются за один шаг, - трем для буквы и пяти для цифры. Следовательно, потерю эффективности можно оценить величиной порядка 25%. Следует отметить также разбухание обрабатываемого выражения в процессе работы.

6. Синтаксис и семантика

Используя рекурсивные переменные можно в значительной степени разделить синтаксический и семантический аспекты задачи, то-есть описывать алгоритмы преобразования языковых объектов в форме, инвариантной относительно изменения деталей синтаксиса. Для иллюстрации этих возможностей мы возьмем алгоритм упорядочивания произвольного числа объектов, между которыми введено транзитивное и рефлексивное отношение порядка. Упорядочивание оказывается необходимым в различных задачах и по отношению к различным типам объектов: упорядочивание слов в списках, упорядочивание аддитивных членов полинома, мультипликативных членов терма и т.п. Задача состоит в том, чтобы

описать алгоритм упорядочивания один раз в достаточно общем виде и использовать его затем для объектов различного синтаксиса при различным образом введенном отношении порядка.

Начнем с построения самого алгоритма, для чего возьмем простейшие объекты - буквы, которые надо упорядочить в алфавитном порядке. Пусть исходный список букв такой:

Б Ж А К Б Г Я Н Р О Г

Простейший алгоритм состоит в создании сумки для упорядоченной части списка и "протаскивании" каждой следующей буквы через этот список до того места, где она перестает нарушать порядок. Вначале список состоит из одной первой буквы:

(Б) Ж А К Б Г Я Н Р О Г

Выберем какой-то момент в середине работы, например:

(А Б Б Ж К) Г Я Н Р О Г

Чтобы "протащить" букву Г надо сравнить её с "К", затем с "Ж", затем с "Б". Последнее сравнение дает благоприятный результат, и мы помещаем "Г" между "Б" и "Ж".

Легко видеть, что такой алгоритм требует в общем случае (при случайном распределении букв) числа операций сравнения, пропорционального n^2, где n - число упорядочиваемых объектов. Между тем хорошо известно, что можно построить алгоритм, упорядочивающий n объектов за $c\ n \log n$ сравнений, и существует несколько таких алгоритмов. Мы воспользуемся алгоритмом, основанном на слиянии упорядоченных списков.

Пусть дано два упорядоченных списка из n и m элементов, например:

(А Б Б Ж К) (Г Н Р Я)

Поместим перед ними пустую вначале сумку и будем формировать в ней результат слияния списков:

( ) (А Б Б Ж К) (Г Н Р Я)

Ясно, что для того, чтобы определить первую букву результата достаточно сравнить первые буквы обоих списков. В нашем случае, это буква А и мы переносим её в сумку:

(А) (Б Б Ж К) (Г Н Р Я)

Теперь мы сравниваем первые буквы оставшихся списков и так поступает до тех пор, пока один из списков не окажется пустым:

(А Б Б Г Ж К) ( ) (Н Р Я)

Теперь мы можем перенести другой из исходных списков в сумку результата:

(А Б Б Г Ж К Н Р Я)

Число сравнений, необходимое для слияния списков, не меньше, чем меньше из чисел n и m, и не больше чем их сумма $n+m$

Теперь сведем задачу упорядочивания к некоторому числу слияний упорядоченных списков. Будем считать, что вначале мы имеем n списков из одного элемента каждый. Очевидно, список из одного элемента - всегда упорядоченный. Будем сливать их попарно: первый со вторым, третий с четвертым и т.д. Получаем:

(Б Ж)(А К) (Б Г) (Н Я) (О Р) (Г)

То, что последний список состоит из одного элемента, нисколько

не должно нас беспокоить. Повторяем процедуру попарного слияния вновь полученных списков до тех пор, пока не останется один список, который и представляет результат:

(А Б Ж К) (Б Г Н Я) (Г О Р)

(А Б Б Г Ж К Н Я) (Г О Р)

(А Б Б Г Г Ж К Н О Р Я)

Сколько же операций сравнения требует этот алгоритм? Так как при каждом укрупнении списков, кроме последнего, их максимальная длина возрастает вдвое, число укрупнений равно минимальному целому числу, большему чем $\log_2 n$, то-есть оно не больше чем $\log_2 n + 1$. Число сравнений при каждом укрупнении, не превышает суммарной длины всех списков, которая всегда остается равной n. Итак, число сравнений не больше, чем $n(\log_2 n + 1)$

Опишем теперь этот алгорим на рефале. Сохраним в качестве объектов упорядочения символы и будем считать, что отношение порядка определено предикатом 'ПОРЯДОК' § 23. Сначала надо описать процедуру слияния упорядоченных списков. Формат обращения:

$$\leftarrow \text{'СЛИУС'}\ (\leftarrow 1)(\leftarrow 2)\ .$$

Такой формат позволяет обойтись без вспомогательного предложения, заводящего сумку. Мы будем помещать результат между детерминативом и сумкой первого списка без всяких скобок. Таким образом формат описания 'СЛИУС' есть

$$\S\ \leftarrow \text{'СЛИУС'}\ \leftarrow R\ (\leftarrow 1)(\leftarrow 2) =$$

а формат обращения представляет частный случай, когда аргумент $\leftarrow R$

пустой.

§ 63 Слияние упорядоченных списков символов

§ .1 К 'СЛИУС' ЕR (ЕА Е1)(ЕВ Е2) ≡

К 'СЛИУС1' К 'ПОРЯДОК' ЕА ЕВ. ЕR (Е1)(Е2).

§ .2 К 'СЛИУС' ЕR ( )(Е2) ≡ ЕR Е2

§ .3 К 'СЛИУС' ЕR (Е1)( ) ≡ ЕR Е1

§/А.1 К 'СЛИУС1' (ЕА ЕВ) ЕR (Е1)(Е2) ≡

К 'СЛИУС' ЕR ЕА (Е1)(ЕВ Е2).

§/А.2 К 'СЛИУС1' ¬ ЕА ЕВ ЕR (Е1)(Е2) ≡

К 'СЛИУС' ЕR ЕВ (ЕА Е1)(Е2).

Первое укрупнение отличается от остальных тем, что первичными объектами являются не списки, заключенные в скобки, а символы. Поэтому функцию упорядочивания 'УПОРЯД' мы определим как вертикальную последовательность функций первого укрупнения 'ПЕРУ' и укрупнения в цикле 'УКРУП'

§ 64 Упорядочивание последовательности символов

К 'УПОРЯД' Е1 ≡ К 'ПЕРУ' (Е1).

§ 64/А Первое укрупнение

.1 К 'ПЕРУ' ЕR (ЕА ЕВ Е1) ≡

К 'ПЕРУ' ЕR (К 'СЛИУС' (ЕА)(ЕВ).) (Е1).

§ .2 К 'ПЕРУ' ЕR (ЕА) ≡ К 'УКРУП' (ЕR (ЕА)).

§ .3 Κ 'ПЕРУ' ΕR ( ) ≡ Κ 'УКРУП' (ΕR).

§ 64/Б Укрупнение в цикле

.I Конец работы Κ 'УКРУП' ((ΕR)) ≡ ΕR

§ .2 Κ 'УКРУП' ΕR (WA WB Ε1) ≡

Κ 'УКРУП' ΕR (Κ 'СЛИУС' WA WB .) (Ε1).

§ .3 Κ 'УКРУП' ΕR (WA) ≡ Κ 'УКРУП' (ΕR WA).

§ .4 Κ 'УКРУП' ΕR ( ) ≡ Κ 'УКРУП' (ΕR).

Функция, описанная таким образом, годится только для упорядочения последовательности символов. Чтобы она могла оперировать с объектами более сложного синтаксиса, надо модифицировать её, введя в описание синтаксическое понятие < объект > , которое мы будем изображать свободной переменной вида Ε 'ОБ' <a> Функция 'ОБ', остающаяся в описании упорядочивания неопределенной, должна быть описана дополнительно, и может определяться по-разному в различных приложениях. Следовательно, первое, что мы сделаем - это заменим в левой части § 63.I переменные SA и S B на Ε'ОБ'A и Ε'ОБ' B

Но это не все. Необходимо еще изменить определение процедуры 'ПОРЯДОК' чтобы она допускала аргументы более общего вида. Простейшим вариантом было бы установить такой формат:

Κ 'ПОРЯДОК' (ΕA)(ΕB).

Однако это определение нехорошо тем, что заведомо требует двух пар скобок - даже в том случае, когда они не нужны, как например,

при объектах упорядочивания, являющихся символами или термами рефала. Поэтому в дополнение к понятию объекта мы введем понятие терм-объекта, изображаемое переменной W 'ТОБ'<a> , и две функции: 'ОБТОБ' , преобразующую объект в терм-объект, и 'ТОБОБ' совершающую обратное преобразование. Терм-объекты являются аргументами функции 'ПОРЯДОК'

Прежде чем двигаться дальше, мы введем соглашения об описании синтаксиса функций, то-есть формата обращений к ним и результата их предельной конкретизации. Для обозначения синтаксического типа объектов будем использовать рекурсивные переменные в виде их главных вхождений. Форматы обращения и результаты объединим в одном квазипредложении. Например, синтаксис функций 'ОБТОБ' и 'ТОБОБ' будет описываться квазипредложениями

```
# K 'ОБТОБ' E 'ОБ'A = W 'ТОБ'B

# K 'ТОБОБ' W 'ТОБ'A = E 'ОБ'B
```

Знак # служит для отделения одного квазипредложения от другого. Синтаксические описания функций не входят в формальный текст на рефале, это - вспомогательная информация типа комментариев. Они бывают полезны в процессе составления программ на рефале для согласования синтаксиса языковых объектов в момент описания функции и в момент её использования.

Синтаксис функции 'ПОРЯДОК' опишется так:

```
# K 'ПОРЯДОК' W 'ТОБ'A W 'ТОБ'B = (W 'ТОБ'A W 'ТОБ'B)
                                 | - W 'ТОБ'B W 'ТОБ'A
```

Черта | служит, как и в бэкусовской нормальной форме, для отделения вариантов правой части; совпадение идентификаторов переменных в правой и левой части указывает на неизменность объекта.

Итак, процедура слияния упорядоченных списков произвольных объектов получает следующее описание:

§ 65 Слияние упорядоченных списков

.I $\underline{\leq}$ 'СЛИУС' $\underline{\in}$R ($\underline{\in}$'ОБ'A $\underline{\in}$1)($\underline{\in}$'ОБ'B $\underline{\in}$2) $\underline{\geq}$

$\underline{\leq}$ 'СЛИУС1' $\underline{\leq}$ 'ПОРЯДОК' $\underline{\leq}$ 'ОБТОБ' $\underline{\in}$A. $\underline{\leq}$ 'ОБТОБ' $\underline{\in}$B.

. $\underline{\in}$R ($\underline{\in}$1)($\underline{\in}$2) .

§ .2 $\underline{\leq}$ 'СЛИУС' $\underline{\in}$R ( )($\underline{\in}$2) $\underline{\geq}$ $\underline{\in}$R $\underline{\in}$2

§ .3 $\underline{\leq}$ 'СЛИУС' $\underline{\in}$R ($\underline{\in}$1)( ) $\underline{\geq}$ $\underline{\in}$R $\underline{\in}$1

§ 65A.I $\underline{\leq}$ 'СЛИУС1' ($\underline{W}$A $\underline{W}$B) $\underline{\in}$R ($\underline{\in}$1)($\underline{\in}$2) $\underline{\geq}$

$\underline{\leq}$ 'СЛИУС' $\underline{\in}$R $\underline{\leq}$ 'ТОБОБ' $\underline{W}$A. ($\underline{\in}$1)($\underline{\leq}$ 'ТОБОБ' $\underline{W}$B. $\underline{\in}$2) .

§ .2 $\underline{\leq}$ 'СЛИУС1' $\neg$ $\underline{W}$A $\underline{W}$B $\underline{\in}$R ($\underline{\in}$1)($\underline{\in}$2) $\underline{\geq}$

$\underline{\leq}$ 'СЛИУС' $\underline{\in}$R $\underline{\leq}$ 'ТОБОБ' $\underline{W}$B. ($\underline{\leq}$ 'ТОБОБ' $\underline{W}$A. $\underline{\in}$1)($\underline{\in}$2) .

Синтаксис функции 'СЛИУС'

$\underline{\leq}$ 'СЛИУС' ($\underline{\leq}$ $\underline{\in}$'ОБ' ...>)($\underline{\leq}$ $\underline{\in}$'ОБ'...>) $\underline{\geq}$ $\underline{\leq}$ $\underline{\in}$'ОБ'...>

Через $\underline{\leq}$ <гл.вх. б. и.> ...> где <гл.вх. б. и.> - главное вхождение рекурсивной переменной без идентификатора, мы обозначаем синтаксический объект, являющийся последовательностью из произвольного числа объектов <гл. вх. б. и.> .

Процедура первого укрупнения модифицируется следующим образом:

```
§ 66.I ⊑'ПЕРУ' ⊑R (⊑'ОБ'A ⊑'ОБ'B ⊑1) ≥
        ⊑'ПЕРУ' ⊑R (⊑'СЛИУС'(⊑A)(⊑B).) (⊑1).
§ .2  ⊑'ПЕРУ' ⊑R (⊑'ОБ'A) ≥ ⊑'УКРУП' (⊑R (⊑A)).
§ .3  ⊑'ПЕРУ' ⊑R ( ) ≥ ⊑'УКРУП' (⊑R).
```

Определения функций 'УПОРЯД' и 'УКРУП' остаются без изменений.

Синтаксис функции 'УПОРЯД'

```
# ⊑'УПОРЯД' ⟨⊑'ОБ'...⟩ ≥ ⟨⊑'ОБ'...⟩
```

Следовательно, аргумент этой функции должен состоять из однородных элементов - объектов. Если, например, мы хотим упорядочивать члены в алгебраическом выражении, его надо записывать, не опуская знак + перед первым членом. Тогда объект упорядочивания можно описать функцией:

```
§ 67.I ⊑'ОБ' + ⊑A + ⊑1 ≥ (+⊑A) + ⊑1
§ 67.2 ⊑'ОБ' + ⊑A ≥ (+⊑A)
§ 67.3 ⊑'ОБ' ≥ ¬
```

Знак + входит в состав объекта; только таким образом можно обеспечить однородность упорядочиваемых элементов. Заметим вообще, что однородность синтаксиса всегда упрощает алгоритмы обработки. В тех случаях, когда на входе и выходе алгоритма желательно иметь

привычный неоднородный синтаксис, например, сумму членов без первого знака +, оказывается, как правило, разумным ввести специальное входное и выходное преобразование; в данном случае - приписывание и убирание знака +.

Функции преобразования 'ОБ' в 'ТОБ' и обратно описываются в данном случае следующим образом:

§ 67/1 ⊑ 'ОБТОБ' + ⊑A ⊇ (⊑A)

§ 67/2 ⊑ 'ТОБОБ' (⊑A) ⊇ +⊑A

Можно было бы, конечно, и сохранить знак + в терм-объекте.

Если надо упорядочивать члены, разделенные знаками + или -, в §§ 67, 67/1, 67/2 надо заменить знак + на переменную ⊆('ИЗ'(+-))⟨a⟩ Для упорядочивания слов, разделенных запятыми, напишем:

§ 68.1 ⊑ 'ОБ' ⊑A, ⊑' ⊇ (⊑A,)⊑'

§ 68.2 ⊑ 'ОБ' ⊇ ¬

§ 68/1 ⊑ 'ОБТОБ' ⊑A, ⊇ (⊑A)

§ 68/2 ⊑ 'ТОБОБ'(⊑A) ⊇ ⊑A,

В данном случае требование однородности синтаксиса предписывает ставить запятую в конце списка слов. Убирая запятую из терм-объекта, мы сводим формат процедуры 'ПОРЯДОК' к двум сумкам, в которых помещаются сравниваемые слова.

Наконец, для упорядочивания символов, мы получаем тривиальный синтаксис объектов:

§ 69.I $\underline{\leq}$ 'ОБ' $\leq A \in^1 \geq (\leq A) \in^1$

§ 69.2 $\underline{\leq}$ 'ОБ' $\geq \neg$

§ 69/I $\underline{\leq}$ 'ОБТОБ' $\leq A \geq \leq A$

§ 69/2 $\underline{\leq}$ 'ТОБОБ' $\leq A \geq \leq A$

Описание процедуры упорядочивания с переменным синтаксисом объектов (§ 64, § 66, § 65, § 65А, § 64Б), пока оно не дополнено описанием синтаксических функций, определяет, собственно говоря, не рекурсивную функцию, а лишь рекурсивную схему функции. Его можно скомбинировать с любой группой предложений, описывающих функции 'ОБ', 'ОБТОБ' и 'ТОБОБ' ; тогда получится программа для упорядочивания объектов данного синтаксиса. В частности, если использовать предложения §§ 69,69/I, 69/2, получим программу, эквивалентную нашей исходной программе упорядочивания символов (§ 64, § 64А, § 63, § 63А, § 64Б).

Ясно, однако, что рекурсивная схема, дополненная описанием синтаксиса объектов, не только состоит из большего числа предложений, но и работать будет заметно медленнее. Является ли эта потеря эффективности неизбежной?

Отнюдь нет. Её можно избежать путем некоторого преобразования программы, а именно путем устранения рекурсивных переменных и прогонки. И то, и другое может производиться автоматически, а в результате этого преобразования получится такая же программа, как

если бы она была написана специально для данного синтаксиса. Таким образом, программист может писать программу в сильно расчлененном вида и свободно использовать алгоритм с переменным синтаксисом, а приведет программу к эффективному виду за него машина.

Мы продемонстрируем этот процесс на примере сочетания процедуры слияния упорядоченных списков, §§ 65, 65А, с синтаксисом списка слов, §§ 68, 68/I, 68/2.

Для сокращения записи заменим детерминативы на греческие буквы:

'СЛИУС'	φ
'СЛИУСІ'	ψ
'ПОРЯДОК'	π
'ОБ'	ω
'ОБТОБ'	τ
'ТОБОБ'	σ

Вспомогательные функции будем обозначать начальными греческими буквами.

Устраняем рекурсивные переменные в § 65.

§ 70 $\underline{\kappa}\,\varphi\,\underline{\in} R\,(\underline{\in}1)(\underline{\in}2) \equiv \underline{\kappa}\,\alpha\,\underline{\in} R\,(\underline{\kappa}\,\omega\,\underline{\in}1\,.\,)(\underline{\in}2)\,.$

§ 70А.I $\underline{\kappa}\,\alpha\,\underline{\in} R\,((\underline{\in}A)\underline{\in}1)\,(\underline{\in}2) \equiv$

$$\underline{\kappa}\,\rho\,\underline{\in} R\,((\underline{\in}A)\underline{\in}1\,)(\underline{\kappa}\,\omega\,\underline{\in}2\,.\,)\,.$$

§ 70А.2 $\underline{\kappa}\,\alpha\,\underline{\in} R\,(\neg\,\underline{\in}1)(\underline{\in}2) \equiv \underline{\kappa}\,\gamma\,\underline{\in} R\,(\underline{\in}1)(\underline{\in}2)\,.$

§ 70Б.I $\underline{K}\rho\,\underline{E}R((\underline{E}A)\underline{E}1)((\underline{E}B)\underline{E}2) \geq$

$\underline{K}\psi\underline{K}\pi\underline{K}\tau\,\underline{E}A_{\cdot}\,\underline{K}\tau\,\underline{E}B_{\cdot\cdot}\,\underline{E}R(\underline{E}1)(\underline{E}2)_{\cdot}$

§ 70Б.2 $\underline{K}\beta\underline{E}R((\underline{E}A)\underline{E}1)(\neg\,\underline{E}2) \geq \underline{K}\gamma\,\underline{E}R(\underline{E}A\,\underline{E}1)(\underline{E}2)_{\cdot}$

§ 70В.I $\underline{K}\gamma\,\underline{E}R(\,)(\underline{E}2) \geq \underline{E}R\,\underline{E}2$

§ 70В.2 $\underline{K}\gamma\,\underline{E}R(\underline{E}1)(\,) \geq \underline{E}R\,\underline{E}1$

Начинаем прогонку. Прогоняем § 70, используя § 69. Получаем два предложения:

§ 7I.I $\underline{K}\varphi\,\underline{E}R(\underline{E}A,\underline{E}1)(\underline{E}2) \geq \underline{K}\alpha\,\underline{E}R((\underline{E}A,)\underline{E}1)(\underline{E}2)_{\cdot}$

§ 7I.2 $\underline{K}\varphi\,\underline{E}R(\,)(\underline{E}2) \geq \underline{K}\alpha\,\underline{E}R(\neg\,)(\underline{E}2)_{\cdot}$

Продолжаем прогонку § 7I.I и § 7I.2.

§ 7I.I $\underline{K}\varphi\,\underline{E}R(\underline{E}A,\underline{E}1)(\underline{E}2) \geq$

$\underline{K}\,\beta\,\underline{E}R((\underline{E}A,)\underline{E}1)(\underline{K}\,\omega\,\underline{E}2_{\cdot})_{\cdot}$

§ 7I.2 $\underline{K}\varphi\,\underline{E}R(\,)(\underline{E}2) \geq \underline{K}\gamma\,\underline{E}R(\,)(\underline{E}2)_{\cdot} \geq \underline{E}R\,\underline{E}2$

Прогонка § 7I.I дает расщепление на два предложения:

§ 7I.I.I $\underline{K}\varphi\,\underline{E}R(\underline{E}A,\underline{E}1)(\underline{E}B,\underline{E}2) \geq$

$$\underline{K}\rho\,\underline{E}R\,((\underline{E}A,)\underline{E}1)((\underline{E}B,)\underline{E}2)_{\cdot}\;=$$

$$\underline{K}\psi\,\underline{K}\pi\,\underline{K}c\;\underline{E}A,_{\cdot}\;\underline{K}\tau\,\underline{E}B,_{\cdot\cdot}\;\underline{E}R\,(\underline{E}1)(\underline{E}2)_{\cdot}\;=$$

$$\underline{K}\,\psi\,\underline{K}\,\pi\,(\underline{E}A)(\underline{E}B)_{\cdot}\;\underline{E}R\,(\underline{E}1)(\underline{E}2)_{\cdot}$$

§ 71.1.2 $\underline{K}\,\varphi\,\underline{E}R\,(\underline{E}A,\underline{E}1)()\;=\;\underline{K}\beta\,\underline{E}R\,((\underline{E}A,)\underline{E}1)(\neg)_{\cdot}\;=$

$$\underline{K}\gamma\,\underline{E}R\,(\underline{E}A,\underline{E}1)()_{\cdot}\;=\;\underline{E}R\,\underline{E}A,\underline{E}1$$

На этой стадии прогонки функции φ мы должны остановиться, так как получили три предложения (§ 71.1.1, § 71.1.2, § 71.2), которые выражают функцию φ только через функции ψ и π, которые являются основными, а синтаксические и вспомогательные функции $\omega, \tau, \alpha, \beta, \gamma$ исключены из описания.

Описание функции ψ § 65А, не содержит рекурсивных переменных, но содержит синтаксическую функцию σ § 68/2, которая должна быть исключена путем прогонки:

§ 72.1 $\underline{K}\,\psi\,((\underline{E}A)(\underline{E}B))\,\underline{E}R\,(\underline{E}1)(\underline{E}2)\;=$

$$\underline{K}\,\psi\,\underline{E}R\,\underline{E}A\,,\,(\underline{E}1)(\underline{E}B,\underline{E}2)_{\cdot}$$

§ 72.2 $\underline{K}\,\psi\,\neg\,(\underline{E}A)(\underline{E}B)\,\underline{E}R\,(\underline{E}1)(\underline{E}2)\;=$

$$\underline{K}\,\psi\,\underline{E}R\,\underline{E}B,\,(\underline{E}A,\underline{E}1)(\underline{E}2)_{\cdot}$$

Итак, мы получаем следующую программу, которая не отличается от наидучшей программы, написанной для упорядочения слов, разделенных запятыми (за исключением, быть может, незначительной особенности § 73.2):

§ 73.I К 'слиус' ЕR (ЕA, Е1)(ЕB, Е2) ≡

К 'слиус1' К 'порядок' (ЕA)(ЕB) . ЕR (Е1)(Е2) .

§ 73.2 К 'слиус' ЕR (ЕA, Е1)( ) ≡ ЕR ЕA, Е1

§ 73.3 К 'слиус' ЕR ( )(Е2) ≡ ЕR Е2

§ 73A.I К 'слиус1 ((ЕA)(ЕB)) ЕR (Е1)(Е2) ≡

К 'слиус' ЕR ЕA, (Е1)(ЕB, Е2) .

§ 73A.2 К 'слиус1' ¬ (ЕA)(ЕB) ЕR (Е1)(Е2) ≡

К 'слиус' ЕR ЕB, (ЕA, Е1)(Е2) .

Избранный нами синтаксис объектов требует выделения слова (просмотр) на каждом этапе упорядочения. Можно было бы сэкономить время, если с самого начала заключить слова в скобки и объявить объектом упорядочения терм, а после упорядочения снова заменить скобки запятыми. Это, однако, приводит к удлинению промежуточных результатов (замена одной запятой на пару скобок), которое при малой средней длине слов может оказаться довольно значительным и нежелательным. Поэтому рассмотренный нами синтаксис отнюдь не является бессмысленным.

Вообще же, следует отметить, что проведение различия между

внешним (входным-выходным) и внутренним синтаксисом часто бывает оправдано. Оно, фактически, позволяет избежать многократного синтаксического анализа текста в процессе работы, если синтаксис сложный или требующий просмотров. Например, выделив числа и идентификаторы один раз, и заключив их в скобки с отметкой 'чис' или 'ид', мы можем распознавать их с помощью комбинаций ('чис' Е 1), ('ид' Е 2), не проводя синтаксического анализа, и даже не просматривая их повторно.

Задача 5.4. Дополнить следующую рекурсивную схему:

§ 7I Поэлементный просмотр с накоплением

К φ Е1 ≡ К ψ ( ) Е1 .

§ 7I/A.I К ψ (Е R) Е 'элмн' A Е1 ≡

К ψ К 'преобр' (Е R) Е A . Е1 .

§ 7I/A.2 К ψ (Е R) ≡ Е R

с синтаксисом процедуры преобразования

\# К 'ПРЕОБР' (Е 'ПРОСМОТРЕННАЯ ЧАСТЬ' R) Е 'ЭЛМН' A ≡

(Е 'ПРОСМОТРЕННАЯ ЧАСТЬ' S)

описанием функций 'ЭЛМН' и 'ПРЕОБР' таким образом, чтобы получилось описание функции φ которая в случае, если имеется несколько одинаковых термов собирает их все в одну цепочку, помещаемую на месте первого вхождения. Например конкретизация

```
K φ (ABC(1+2)B12 B(1+2)(1+3)Z) .
```

должна давать результат

```
A B B B C (1+2)(1+2)12(1+3)Z
```

Путем прогонки привести описание к эффективному виду и сравнить с программой, написанной специально.

7. Свободные переменные в спецификаторе рекурсивной переменной

Рассмотрим задачу анализа текста, записанного в жестком формате на перфокартах, вводимых в "вертикальном" режиме. При таком вводе каждая перфокарта порождает ровно 80 байтов (объектных знаков), соответствующих 80-ти колонкам двоичных разрядов на перфокарте. Говоря о "жестком формате", мы подразумеваем, что синтаксически текст разбивается на предложения, состоящие всегда из 80 знаков, а каждое предложение разбивается на поля, следующие всегда из определенного числа знаков каждое. Таким образом, синтаксический анализ включает процесс счета знаков - ситуация не слишком типичная для задач, решаемых с помощью рефала (особенно для сложных задач).

Допустим, что анализируется текст на автокоде, каждое предложение которого состоит из следующих полей:

№ поля	Число знаков	Номера позиций	Название поля	Детерминатив
I	8	I - 8	Метка	МЕТКА
2	8	9 -I6	Операция	ОПИЯ
3	30	I7 -46	Операнд	ОПАНД
4	25	47 -7I	Комментарий	КОММ
5	I	72	Признак продолжения	ПРОД
6	8	73 -80	Номер перфокарты	НОМЕР

Для анализа текста нам понадобятся процедуры, отщепляющие заданное число символов. Такие процедуры можно описать, учитывая каждый символ как свободную переменную. Например, понятие 'ТРИСИМ' - три символа - можно описать так

$$\S \underline{\leq} \text{'ТРИСИМ'} \leq 1 \leq 2 \leq 3 \in A \supseteq (\leq 1 \leq 2 \leq 3) \in A$$

$$\S \underline{\leq} \text{'ТРИСИМ'} \in A \supseteq \neg \in A$$

Теперь можно использовать рекурсивную переменную $\in$ 'ТРИСИМ' $\langle a \rangle$ для отщепления большего числа символов. Например, понятие 'ВОСИМ' - восемь символов, получает описание

$$\S \underline{\leq} \text{'ВОСИМ'} \in \text{'ТРИСИМ'} 1 \in \text{'ТРИСИМ'} 2 \leq 3 \leq 4 \in A \supseteq$$

$$(\in 1 \in 2 \leq 3 \leq 4) \in A$$

$$\S \underline{\leq} \text{'ВОСИМ'} \in A \supseteq \neg \in A$$

Однако ясно, что этот путь не слишком перспективен. Его главный недостаток - необходимость специальной функции для каждого числа. Поэтому определим функцию так, чтобы число отщепляемых символов входило в качестве аргумента.

§ 72 К 'СИМОВ' (Е*N*) Е*A* ≡ К 'СИМОВ1' (Е*N*)( ) Е*A* .

§ 72/I.I К 'СИМОВ1' (0) Е*1* ≡ Е*1*

§ .2 К 'СИМОВ1' (Е*N*)(Е*R*) Е*1* Е*A* ≡

К 'СИМОВ1' (К 'ВЫЧ' Е*N*, 1 .)(Е*R* Е*1*) Е*A* .

§ .3 К 'СИМОВ1' (Е*N*)(Е*R*) Е*A* ≡ ¬ Е*R* Е*A*

Теперь Е('СИМОВ'(8)) <a> означает переменную, принимающую значение из восьми очередных символов. Предложение (рефала) отщепляющее и разбивающее на поля очередное предложение текста, имеет левую часть:

К φ Е('СИМОВ' (8))I Е('СИМОВ' (8))2 Е('СИМОВ' (30))3

Е('СИМОВ' (25))4 Е('СИМОВ' (I))5 Е('СИМОВ' (8))6 Е*X* ≡

Можно продвинуться еще дальше по пути создания все более общих функций. У разных автокодов число знаков, отводимых под поля метки, операции и т.д., может оказаться неодинаковым, в то время как алгоритм обработки этих полей - одним и тем же. Этот алгоритм можно описать одной функцией, которая в качестве дополнительного аргумента будет содержать описание расчленения предложения автокода

на шесть перечисленных выше полей. Будем изображать расчленение на поля в виде суммы чисел знаков, отводимых на каждое поле. Например, расчленение, отраженное в приведенной выше таблице, изображается формулой:

$$8 + 8 + 30 + 25 + 1 + 8$$

Формулу расчленения, взятую в скобки, будем помещать непосредственно за детерминативом φ . Итак, обращение к функции φ будет для нашего случая иметь вид:

$$\underline{K}\ \varphi\ (\ 8 + 8 + 30 + 25 + 1 + 8\ )\ <\ T\ >$$

где $<T>$ - обрабатываемый текст на автокоде. Левая часть рефал-предложения, осуществляющего расчленение, будет иметь вид

$$\underline{K}\ \varphi\ (\ \underline{E}A + \underline{E}B + \underline{E}C + \underline{E}D + \underline{E}E + \underline{E}F\ )\ \underline{E}('\text{симов}'(\underline{E}A))1$$
$$\underline{E}('\text{симов}'(\underline{E}B))2\ \underline{E}('\text{симов}'(\underline{E}C))3\ \underline{E}('\text{симов}'(\underline{E}D))4$$
$$\underline{E}('\text{симов}'(\underline{E}E))5\ \underline{E}('\text{симов}'(\underline{E}F))6\ \underline{E}X\ =$$

Задача 5.5. Решить задачу 4.4, использовав рекурсивные переменные, чтобы избежать просмотров высших размерностей.

3. Непредикативные переменные

Рекурсивные переменные были задуманы и введены в рефал для использования их в том частном случае, когда они являются предикативными, то-есть не портят конкретизируемое выражение в процессе отождествления. Но заставить рефал-транслятор проверять переменные на предикативность, то-есть после успешного отождествления, давшего выражение

$$(\langle e_1 \rangle)\langle e_2 \rangle$$

сравнивать $\langle e_1 \rangle \langle e_2 \rangle$ с исходным отрезком, и аналогичным образом поступать при безуспешном отождествлении, - значило бы снизить эффективность транслятора до практически неприемлемой величины. Поэтому, с одной стороны, ошибочное описание функции, используемой в главном вхождении рекурсивной переменной, может привести к совершенно неожиданным последствиям, а с другой стороны, появляется возможность сознательно пользоваться непредикативными переменными, которые вызывают преобразование конкретизируемого выражения в процессе отождествления. Сколь ценна эта возможность? Заметим, что она вступает в противоречие с основными идеями, на которых строится рефал, ибо при использовании непредикативных переменных нельзя рассматривать предложения рефала как альтернативные правила замены левой части на правую, которые в левых частях содержат различные типовые ситуации для заменяемого выражения. При использовании непредикативных переменных не только заменяемое выражение преобразуется до замены в случае успешного отождествления, но и в случае безуспешного отождествления оно поступает на следующее предложение в уже измененном виде. Можно построить сколько угодно примеров, когда из двух тождественных предложений первое не работает, а второе работает. Вот один из таких примеров:

§ 73.I $\underline{\leq}$ Ф $\underline{\in}$ 'ДАЙ 5' А $\underline{\in}$ В $\geq$ $\underline{\in}$ А

§ 73.2 $\underline{\leq}$ Ф $\underline{\in}$ 'ДАЙ 5' А $\underline{\in}$ В $\geq$ $\underline{\in}$ А

§ 74.I $\underline{\leq}$ 'ДАЙ 5' $\underline{\in}$ 1 5 $\geq$ ($\underline{\in}$ 1 5)

§ 74.2 $\underline{\leq}$ 'ДАЙ 5' $\underline{\in}$ 1 $\geq$ $\neg$ $\underline{\in}$ 1 5

Если конкретизируемое выражение не оканчивается на символ 5, предложение § 73.I неприменимо. Второе такое же предложение применимо всегда.

В результате предложения теряют свою ясность и выразительность. Разбор программы превращается в расшифровку замысловатого ребуса. Возьмем четыре простеньких предложения:

§ 75.I $\underline{\underline{K}}\ \varphi\ \underline{\underline{\in}}1\ \underline{\underline{\in}}\ 'УНТ'A\ \underline{\underline{\in}}2\ \underline{\underline{\supseteq}}$

§ 75.2 $\underline{\underline{K}}\ \varphi\ \underline{\underline{\in}}1\ \underline{\underline{\supseteq}}\ \underline{\underline{\in}}1$

§ 76.I $\underline{\underline{K}}\ 'УНТ'\ \underline{\underline{W}}1\ \underline{\underline{\in}}2\ \underline{\underline{\supseteq}}\ \neg\ \underline{\underline{\in}}2$

§ 76.2 $\underline{\underline{K}}\ 'УНТ'\ \underline{\underline{\supseteq}}\ \neg$

Нужно немало потрудиться, чтобы сообразить, что функция φ, описанная тким образом, уничтожает все термы аргумента, имеющие нечетные номера (первый, третий и т.д.) и оставляет неизменными термы с четными номерами.

Задача 5.6. Путем устранения рекурсивных переменных с последующей прогонкой привести описание функции φ (§§ 75, 76) к удобочитаемому виду.

Тем не менее, за программистом остается право использовать непредикативные переменные по своему усмотрению. Иногда бывают очень полезны квазипредикативные переменные. Мы объединяем под этим именем такие рекурсивные переменные, которые хотя и меняют объект отождествления, но меняют его в некотором смысле несущественно. Это, конечно, неформальное понятие. В следующей главе мы встретимся с примерами использования квазипредикативных переменных.

Не исключена возможность, что в будущем найдутся и более оригинальные способы эффективного использования непредикативных переменных.

С О Д Е Р Ж А Н И Е

№ Т12633 от "27" VII 1971 г. Заказ № 800 Тираж 250 экз.

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