Московский ордена Трудового Красного Знамени
физико-технический институт

Курочкин В.М., Столяров Л.Н.,
Сушков Б.Г., Флеров Ю.А.

КУРС ЛЕКЦИЙ



ТЕОРИЯ И РЕАЛИЗАЦИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

1973 ГОД

Глава 3. Методы трансляции.

Интуитивно под трансляцией понимается механизм перевода с одного формального языка на другой формальный язык, при этом каждое слово x входного языка $\mathcal{L}_x$ переводится в некоторое слово y выходного языка $\mathcal{L}_y$ (обычно это язык команд ЭВМ) в соответствии с заданным смыслом (семантикой). Механизм перевода должен быть реализован на ЭВМ, поэтому к нему предъявляются требования эффективности, другими словами он должен быть задан некоторым алгоритмом.

Транслятором, вообще говоря, мы будем называть алгоритм, реализующий отображение множества T^* (T алфавит входного языка) в множество U^* (U - алфавит выходного языка), либо отображение $\mathcal{L}_x \subseteq T^*$ на $\mathcal{L}_y \subseteq U^*$, либо отображение $\mathcal{L}_x \subseteq T^*$ в U^*.

В связи с тем, что мы будем заниматься методами трансляции языков, естественно формулировать алгоритм перевода не в общем виде (например, в виде нормального алгоритма либо в виде машины Тьюринга), а использовать для построения трансляторов уже изученные выше конструкции - магазинные автоматы и порождающие КС-грамматики.

§1. Метод трансляции, использующий магазинный преобразователь.

Определение 3.1.1. Магазинным преобразователем называется упорядоченная восьмерка $МП = (Z, K, T, U, \delta, z_0, q_0, \mathcal{E})$

Символы Z, K, T имеют тот же смысл, что и в магазинном автомате; $U = \{b_1, b_2, \dots\}$-конечный выходной алфавит;

$\mathcal{E}$ - знак окончания работы преобразователя (это может быть либо пустой магазин - $\mathcal{E} = \Phi$, либо множество заключительных состояний - $\mathcal{E} = F$);

δ -отображение (вообще говоря не однозначное) множества $Z \times K \times T \to Z^* \times K \times U^*$.

Для МП ситуация характеризуется упорядоченной четверкой (ω, P, x, y) где $\omega \in Z^*$, $P \in K$, $x \in T^*$, $y \in U^*$. Один такт работы такого преобразователя заключается в смене ситуации и имеет следующий вид: $(\bar{\omega} Z, P, ax, y) \vdash (\bar{\omega}\omega, q, x\bar{y}y)$, $\bar{\omega} \in Z, y \in U^*$. Такой переход возможен, если имеется $(\omega, q, y) \in \delta(Z, P, a)$, при этом может быть $a = \Phi$. Работа преобразователя задается таблицей перехода следующего вида: $(Z, P, a, \Phi) \vdash (\omega, q, \Phi, y)$ Это означает, что МП читает один левый символ на входе (см. рис. 3.1.), переходит из состояния P в состояние q, записывает цепочку ω вместо Z (верхнего символа магазина) и записывает справа на выходе цепочку y.

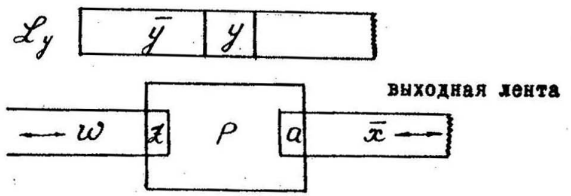


Рис. 3.1.

Заметим, что если игнорировать выход МП, то он превращается

в обычный магазинный автомат.

Пусть задан некоторый магазинный преобразователь.

$МП = (Z, K, T, U, \delta, z_0, q_0, F)$

Мы будем говорить, что заданный МП преобразует (переводит) язык $\mathcal{L}_x$ в язык $\mathcal{L}_y$, если для каждого $x \in \mathcal{L}_x$ существует переход $(z_0, q_0, x, \phi) \vdash^* (u, q, \phi, y)$ где $y \in \mathcal{L}_y$, $q \in F$ - заключительное состояние, останов МП происходит в результате перехода в заключительное состояние (сравни с $\mathcal{F}$ - автоматом); либо

$$(z_0, q_0, x, \phi) \vdash^* (\phi, p, \phi, y)$$

где $y \in \mathcal{L}_y$, останов МП происходит по пустому магазину.

Мы будем рассматривать детерминированные МП, переводящие каждое слово $x \in \mathcal{L}_x$ однозначным образом. Очевидно, что МП задает некоторое отображение $\mathcal{L}_x$ в U^* и при этом реализует однозначное в одну сторону ($\mathcal{L}_x \to \mathcal{L}_y$) соответствие между x и y.

Мы будем записывать это следующим образом:

$$МП(\mathcal{L}_x) = \mathcal{L}_y$$

Пример: 3.I.2. Построим МП, осуществляющий следующее отображение: на вход МП подается КС - язык $\mathcal{L}_x = \{a^n b^n c\}$ $n \geq 1$, на выходе МП получается язык $\mathcal{L}_y = \{a^n b^n c^n\}$.

$МП(\{z_0, \alpha\}, \{0,1,2,3\}, \{a,b,c\}, \{a,b,c\}, \gamma, z_0, \phi)$.

отображение γ зададим так, чтобы магазинный автомат, входящий в МП, допускал язык $\mathcal{L}_x = \{a^n b^m c\}$, $m, n \geq 1$.

Таблица переходов такого МП будет выглядеть следующим образом;

1. $(z_0, 0, a, \phi) \vdash (z_0\alpha, 1, \phi, a)$
2. $(\alpha, 1, a, \phi) \vdash (\alpha\alpha, 1, \phi, a)$
3. $(\alpha, 1, b, \phi) \vdash (\alpha, 2, \phi, b)$
4. $(\alpha, 2, b, \phi) \vdash (\alpha, 2, \phi, b)$
5. $(\alpha, 2, c, \phi) \vdash (\alpha, 3, \phi, \phi)$
6. $(\alpha, 3, \phi, \phi) \vdash (\phi, 3, \phi, c)$
7. $(z_0, 3, \phi, \phi) \vdash (\phi, 3, \phi, \phi)$

МП $(\{a^n b^m c\}) = \{a^n b^m c^n\}$

Посмотрим по таблице преобразования любого слова $x \in \{a^n b^m c\}$ в слово $y \in \{a^n b^m c^n\}$

$(z_0, 0, a^n b^m c, \phi) \vdash (z_0\alpha, 1, a^{n-1} b^m c, a) \vdash^*$

$\vdash^* (z_0\alpha^n, 1, b^m c, a^n) \vdash (z_0\alpha^n, 2, b^{m-1} c, a^n b) \vdash^*$

$\vdash^* (z_0\alpha^n, 2, c, a^n b^n) \vdash (z_0\alpha^n, 3, \phi, a^n b^n) \vdash^*$

$\vdash^* (z_0, 3, \phi, a^n b^m c^n) \vdash (\phi, 3, \phi, a^n b^m c^n)$

Если на тот же МП подать язык $\mathcal{L}_x = \{a^n b^n c\}$, то на выходе будет язык $\mathcal{L}_y = \{a^n b^n c^n\}$

Из примера 3.I.2. видно, что, подавая на МП КС-язык, на выходе можно получить и не КС - язык.

Имеется, однако, интересный случай, когда МП сохраняет свойство выходного языка быть контекстно - свободным.

Это имеет место, если МП получается добавлением выхода к некоторому магазинному автомату, допускающему КС - язык.

Теорема 3.I.3. $\mathcal{L}_x$, допускаемый МП, преобразуется им в КС - язык $\mathcal{L}_y$.

Док-во: Пусть имеется некоторый магазинный преобразователь

$$МП = (Z, K, T, U, \delta, z_0, q_0, \delta)$$

Построим вспомогательный магазинный автомат A (он может быть типа F либо типа N)

$$A = (Z \cup U,\ K \cup \bar{K},\ T \cup U,\ \delta,\ z_0,\ q_0, \delta),$$

где $\bar{K}$ - дубликат K, $Z \cap U = \phi$; $T \cap U = \phi$;

правила определяются так.

1. Если $(z, P, a, \phi) \vdash (w, q, \phi, y)$ в исходном МП, то им соответствуют правила в вспомогательном автомате A. $(z, P, a) \vdash (wy^R, \bar{q}, \phi)$, где y^R - инверсия цепочки y. состоянию q МП соответствует состояние $\bar{q}$ в A;

2. Кроме того автомат A имеет "сокращающие" правила типа: $(b, \bar{q}, b) \vdash (\phi, \bar{q}, \phi)$ для $b \in U, \bar{q} \in \bar{K}$

3. Правила "замены состояния":

$$(z, \bar{q}, \phi) \vdash (z, q, \phi) \text{ для } z \in Z$$

4. Закон окончания работы A тот же, что и для МП.

Язык, допускаемый автоматом A, обозначим через $\mathcal{L}$.

Докажем две леммы для вспомогательного автомата A.

Лемма 3.I.4. Пусть слово $x \in \mathcal{L}_x$ читается МП за κ тактов: $x = a_1 a_2 \dots a_\kappa$, где a_i может быть и пустым символом; в i-ом такте на выходе МП появляется некоторая подцепочка $y_i \in U^*$. Тогда слово $y = a_1 y_1 a_2 y_2 \dots a_\kappa y_\kappa$ допускается вспомогательным автоматом A, т.е. $y \in \mathcal{L}$.

Док - во. Если на вход МП подать слово x, то за κ тактов МП окончит работу. Это можно описать следующей последовательностью переходов из ситуации в ситуацию:

$$(z_0, q_0, a_1 \dots a_\kappa, \phi) \vdash (w_1, q_1, a_2 \dots a_\kappa, y_1) \vdash$$
$$\vdash (w_2, q_2, a_3 \dots a_\kappa, y_1 y_2) \vdash \dots \vdash (w_\kappa, q_\kappa, \phi, y_1 y_2 \dots y_\kappa),$$

где q_κ - заключительное состояние.

В автомате A этой последовательности ситуаций будет соответствовать последовательность работ

$$(z_0, q_0, a_1 y_1 \dots a_\kappa y_\kappa) \vdash (w_1 y_1^R, \bar{q}_1, y_1 a_2 \dots y_\kappa) \vdash^* ,$$

применяя далее правила "сокращения" и "замены" получим,

$$\vdash^* (w_1, q_1, a_2 y_2 \dots y_\kappa) \vdash (w_2 y_2^R, \bar{q}_2, y_2 a_3 \dots y_\kappa) \vdash^*$$
$$\vdash^* (w_2, q_2, a_3 y_3 \dots y_\kappa) \vdash \dots \vdash (w_\kappa y_\kappa^R, \bar{q}_\kappa, y_\kappa) \vdash^*$$
$$\vdash^* (w_\kappa, \bar{q}_\kappa, \phi) \vdash^* (w_\kappa, q_\kappa, \phi).$$

Если A является F - автоматом, то он закончит работу после перехода в заключительное состояние $\bar{q}_\kappa$. Если A является N - автоматом, то он закончит работу после перехода в ситуацию $\vdash^* (\phi, q, \phi)$.

Таким образом A допускает слово $a_1 y_1 \dots a_\kappa y_\kappa = y$

Лемма доказана.

Лемма 3.I.5. Пусть есть автомат A и $y \in \mathcal{L}$, а $y = y_1 y_2 \dots y_\kappa$ разбиение слова y при его чтении автоматом A, которое соответствует переходам из состояния $\bar{q} \in \bar{K}$ в состояние $q_i \in K$.

Тогда 1. $y_i = a_i y_i'$ ($a_i \cup y_i'$ могут быть пустыми)

2. $x = a_1 a_2 \dots a_\kappa \in \mathcal{L}_x$ подается на вход МП, а $y' = y_1' y_2' \dots y_\kappa' \in \mathcal{L}_y$ выдается на выходе МП.

Док - во: Для определенности будем считать, что A есть F - автомат. Слово y допускается автоматом A, следовательно имеется последовательность переходов, приводящая автомат в заключительное состояние при пустой входной ленте.

$(z_0, q_0, y_1 \dots y_k) \overset{*}{\vdash}$

$(w_1, \bar{q}_1, y_2 \dots y_k) \vdash (w_1, q_1, y_2 \dots y_k) \overset{*}{\vdash}$

$(w_2, q_2, y_3 \dots y_k) \vdash (w_2, q_2, y_3 \dots y_k) \overset{*}{\vdash}$

. . . . . .

. . . . . .

. . . . . .

$(w_{k-1}, \bar{q}_{k-1}, y_k) \vdash (w_{k-1}, q_{k-1}, y_k) \overset{*}{\vdash}$

$(w_k, \bar{q}_k, \phi) \vdash (w_k, q_k, \phi)$

q_k – заключительное состояние.

Заменив в последовательности $y = y_1 y_2 \dots y_k$ все $y_i = a_i y_i'$, можно записать работу автомата в следующем виде:

$(w_{i-1}, q_{i-1}, y_i) = (w_{i-1}, q_{i-1}, a_i y_i') \vdash$,

(применяя правило "сокращения" и "замены" состояний)

$\vdash (w_i, \bar{q}_i, y_i') \overset{*}{\vdash} (w_i, \bar{q}_i, \phi) \vdash (w_i, q_i, \phi)$.

Эти переходы А соответствуют таким переходам в МП:

$(w_{i-1}, q_{i-1}, a_i, \phi) \vdash (w_i, q_i, \phi, y_i')$.

Так как закон окончания для А и МП одинаков, то $x = a_1 a_2 \dots a_k \in \mathcal{L}_x$ будет преобразовано в $y = y_1' y_2' \dots y_k'$.

Лемма доказана.

Итак, мы доказали, что построенный вспомогательный автомат А допускает язык $\mathcal{L}$ и потому $\mathcal{L}$ есть КС – язык (лемма 3.I.4.), а магазинный преобразователь переводит $x \in \mathcal{L}_x$ в слова $y' = y_1' y_2' \dots y_k' \in \mathcal{L}_y$. (лемма 3.I.3.)

Если из $y \in \mathcal{L}$ вычеркнуть все $a \in T$, то получим язык $\mathcal{L}_y$, который тоже будет контекстно – свободным.

Теорема доказана.

Возникает следующий вопрос, для всякой ли пары языков $\mathcal{L}_x \subseteq T^*$ и $\mathcal{L}_y \subseteq U^*$ может быть построен транслятор в виде магазинного преобразователя. Ответ на этот вопрос – отрицателен. Ниже будут приведены примеры пар языков, для которых невозможно построить магазинный преобразователь.

Пример 3.I.6. Построить МП, преобразующий язык $\mathcal{L}_x = a^n$ в язык $\mathcal{L}_y = b^{2^{n-1}}$, для $n \geq 1$. Покажем, что МП для такого преобразования построить нельзя.

Предварительно заметим, что в этом случае МП должен был бы - в каждом i – м такте своей работы записывать на выходной ленте точную копию всей выходной информации, накопленной за все $i-1$ тактов. На самом деле, рассмотрим таблицу:

такты	x	y
I такт	a	b
2 такт	$a\,a$	$b\,b$
3 такт	$a\,a\,a$	$b\,b\,bb$
4 такт	$a\,a\,a\,a$	$bb\ bb\ bb\ bb$
. . . . .	. . . . .	. . . . .
. . . . .	. . . . .	. . . . .
$i-1$ такт	a^{i-1}	$b^{2^{i-2}}$
i такт	a^i	$b^{2^{i-2}} b^{2^{i-2}} = b^{2^{i-1}}$

Предположим, что можно построить такой МП, который на каждом такте мог бы запоминать информацию

выходной ленте. Такое запоминание нельзя осуществить при помощи состояний МП, т.к. множество состояний конечно. Далее предположим, что такое запоминание делается на магазинной ленте. Пусть информация о состоянии выходной ленты на $i-1$-м такте работы МП записана в магазине в виде некоторой конечной цепочки ω_{i-1} и мы её можем использовать для построения "копии" на i - м такте. При этом мы были бы вынуждены стереть всю цепочку ω_{i-1} и тем самым уничтожить всю информацию о предыдущем состоянии выходной ленты.

Специфика использования магазина не позволяет в этом случае запоминать информацию выхода МП.

Таким образом, можно заключить, что нельзя построить МП, преобразующий язык $\mathcal{L}_x = a^n$ в язык $\mathcal{L}_y = b^{2^{n-1}}$.

При построении реальных трансляторов нас интересует, вообще говоря, не просто существование какого - либо магазинного преобразователя для пары языков $\mathcal{L}_y = МП(\mathcal{L}_x)$, а такого, который реализует заданное соответствие между $x \in \mathcal{L}_x$ и $y \in \mathcal{L}_y$.

Оказывается, что не всегда можно реализовать заданное соответствие в классе магазинных преобразователей.

Пример 3.1.7. Построить МП, переводящий формулу алгебры логики, записанную при помощи операции $\neg$ (отрицание), $\circ$ (конъюнкция), $\vee$ (дизъюнкция) в эквивалентную формулу алгебры логики, записанную при помощи операций $\oplus$ (сложение по mod_2), $\circ$ (конъюнкция), 1 (const 1)

Отметим, что в этом примере $\mathcal{L}_x$ и $\mathcal{L}_y$ могут быть заданы КС - грамматиками G_x, G_y.

$$T = \{ \vee, \circ, \neg, (,), \ x_1, \ldots, x_n \}$$

$$U = \{ \oplus, \circ, 1, (,), \ x_1, \ldots, x_n \}$$

Пример слова $x \in \mathcal{L}x$: $((\neg((x_1) \vee (x_2))) \cdot (\neg(x_1) \vee \neg(x_2))$.

Пример слова $y \in \mathcal{L}y$: $(((x_1) \oplus 1) \oplus ((x_1) \cdot (x_2) \oplus 1)) \circ (x_2)$.

Оказывается нельзя построить МП, который реализовал бы заданное соответствие.

Покажем, почему это нельзя сделать.

Выпишем известные тождества:

1). $\neg F = F \oplus 1$

2). $F_1 \vee F_2 = F_1 \oplus F_2 \oplus F_1 \circ F_2 \oplus 1$

Рассмотрим пример перевода слова $X = (x_1 \vee x_2) \vee x_3$.

(некоторые скобки опущены для простоты рассмотрения).

Необходимое преобразование, которое должен выполнить МП можно описать следующей таблицей:

Такты МП	1	2	3	4	5	6	7
Входная лента	(	x_1	$\vee$	x_2	)	$\vee$	x_3
Выходная лента	(	x_1	$\oplus x_1$	$\oplus x_1 \circ x_2$	)	$\oplus F_1$	$\oplus F_1 \circ x_3$
Обозначение подцепочки	⌣		F_1		⌣		
Примечание			по тождеству 2)	по тождеству 2)		необходимо копировать F_1	

Из таблицы видно, что МП должен уметь капировать предыдущую выходную информацию, а в примере 3.I.6. было показано, что МП не может этого делать.

Из приведенных примеров видно, что магазинный преобразователь имеет серьезные ограничения. Несмотря на это, при реализации трансляторов для языков программирования он часто применяется. Это можно объяснить двумя причинами. Обычно предложения на языке программирования переводятся транслятором в программу, написанную на автокоде либо непосредственно в машинных кодах, при этом на последовательность команд не накладывается никакого синтаксического ограничения, т.е., другими словами, делается МП такой, который отображает $\mathcal{L}_x$ в U^*.

В этом случае не возникает вопроса о существовании МП для пары языков, т.к. $\mathcal{L}_y \leq U^*$ определяется выбранной семантикой уже заложенной в правила МП. Кроме того, реализация транслятора в виде МП достаточно проста, особенно, если ЭВМ позволяет выполнять команды, в так называемом, магазинном режиме.

Пример 3.I.7. Перевод арифметического выражения в П О Л И З с помощью МП.

Было обнаружено (Я. Лукашевич), что если знаки арифметических операций записывать всегда перед двумя операндами, то для определения порядка операций скобки становятся излишними. Такой метод записи арифметических выражений назвали польской записью.

Если знаки операций стоят после двух операндов, то такая запись арифметического выражения называется польской инверсной записью (ПОЛИЗ).

Например: $a+b\times(c+d)\times(k-\ell)a$ - скобочная запись

$abcd+\times k\ell-\times+$ - П О Л И З

Пусть арифметическое выражение ограничено справа символом / .

МП = $(Z, K, T, U, \gamma, z_0, q_0, \delta = N)$

$Z = \{z_0, (,$ знаки$\}$

$K = \{q_0,$ знаки, $), 1\}$

$U = \{$ буквы, знаки$\}$ $T = U \cup \{(,), 1\}$

Схема работы преобразователя

Z	K	T	Z	K	T	U	примечание
z	q_0	буква	z	q_0		буква	
z	q_0	(	z(	q_0			
z	q_0	проч	z	проч			т.е.{), знак,1}
(	)		ϕ	q_0			
знак	)		ϕ	)		знак	
знак I	знак2		знакIзнак2	q_0			$\angle$
			ϕ	знак2			$\geqslant$
проч	знак		проч.знак	q_0			т.е.{z, (}
знак	1		ϕ	1		знак	
z	1		ϕ				конец работы

Если на вход посылать команды машины БЭСМ-6 вместо

(считывание магазинное)	СМ	буква	буква
(магазинная арифметическая операция)	I7	знак	знак

то получается программа для вычислительной машины.

§2. Синтаксически управляемая трансляция (СУТ).

СУТ в самом общем виде есть преобразование $\mathcal{L}_x$ в $\mathcal{L}_y$ при помощи конечного числа различных специальных подстановок в правила КС - грамматики G_x для языка $\mathcal{L}_x$.

Пусть язык $\mathcal{L}_x$ задан КС - грамматикой $G_x=(V,T,P,S)$, кроме того задан алфавит U.

Каждому правилу

$$P_i=(A\longrightarrow\alpha_{i1}\alpha_{i2}\dots\alpha_{ik})\in P$$

ставится в соответствие одно (или несколько при недетер - минированном переводе) слово.

$$\Psi_i\in\{U\cup\{\beta_{i1}\dots\beta_{ir}\dots\beta_{it}\}/\beta_{ir}\notin T\}^*$$

где t - количество вхождений нетерминальных символов в правую часть правила P_i (может быть одних и тех же символов),

β_{ir} - соответствует некоторому вхождению нетерминального символа; разным вхождениям одного и того же нетерминального символа соответствуют различные значения r .

Опишем несколько свойств такого соответствия.

I. $\varphi_i\neq\phi$ для всех правил грамматики G_x исключая быть может терминальные правила вида

$$A_i\rightarrow\varphi_i\,/\,\varphi_i\in T^*$$

2. Для терминальных правил вида $A_i\rightarrow\varphi_i\,/\,\varphi_i\in T^*$

$$\{\beta_{i1}\dots\beta_{it}\}=\phi\quad и\quad\varphi_i=V\in U^*$$

Пример 3.2.I.

$$A_i\rightarrow A_a\,AB\rightsquigarrow\beta_1\;\beta_2\,y\,\beta_3 y$$
$$\rightsquigarrow\beta_1.\,\beta_1\;\beta_2\,y\beta_3\,\beta_1$$

Все соответствия, приведенные в примере, правильно построены. Первому вхождению A соответствует β_1 ; второму вхождению A соответствует β_2 , вхождению B соответствует β_3 .

$A_i\rightarrow A_a\,AB\rightsquigarrow\beta_1 y\beta_2 y$ - неправильное соответствие

Слово ψ будем называть семантикой правила $A\rightarrow\xi$ и записывать

$$A\rightarrow\varphi\rightsquigarrow\psi$$

Далее определим процесс перевода x в V , где $V\in\mathcal{L}_y\leq U^*$, как некоторую последовательность подстановок ψ_i в ψ_j , которая "управляется" выводом слова x в исходной КС - грамматике G_x .

Определим процесс перевода индуктивно.

I. Пусть x выводится в грамматике G_x ровно за один шаг, т.е. $(A \to x) \in P$, то переводом x будет сама семантика этого правила; т.к. x не содержит нетерминальных символов, то $\gamma = v \in U^*$.

$$A \to x \sim v$$

2. Пусть перевод сделан для всех выводов и дан вывод $A \overset{*}{\Rightarrow} x$ длины n. Рассмотрим первый шаг этого вывода $A \to \varphi = \alpha_1 \alpha_2 \dots \alpha_j \dots \alpha_k \overset{*}{\Rightarrow} x = x_1 x_2 \dots x_k$,

где $\alpha_j \overset{*}{\Rightarrow} x_j$

Если α_j - нетерминальный символ, то вывод имеет длину меньше n и по условиям индукции для него существует перевод $\alpha_j \overset{*}{\Rightarrow} x_j \approx v_j$

В слове γ из правила перевода $A \to \varphi \sim \gamma$ заменим каждое вхождение β_j, соответствующее α_j - му, на v_j.

Полученные таким образом слова $v \in U^*$ будем называть переводом вывода $A \overset{*}{\Rightarrow} x$. Этот факт мы запишем в виде: $A \overset{*}{\Rightarrow} x \approx v$

Совокупность правил перевода будем обозначать через

$$Q = \{A \to \varphi \sim \gamma \mid A \to \varphi \in P\}$$

Определение 3.2.2. Назовем упорядоченную тройку $\langle G_x, Q, U \rangle$ синтаксически управляемым транслятором (СУТ). Применение соответствующей операции перевода к некоторому КС - языку $\mathcal{L}_x$ будем обозначать

$$\mathcal{L}_y = СУТ(\mathcal{L}_x).$$

Для СУТ, так же как и для магазинных преобразователей, имеется частный случай, когда КС - язык $\mathcal{L}_x$ всегда переводится в КС - язык $\mathcal{L}_y$.

Теорема 3.2.3. Если число вхождений каждого β_j в слове γ не превосходит 1, то $\mathcal{L}_y = СУТ(\mathcal{L}_x)$ является КС - языком.

Рассмотрим более подробно ограничения на семантику правил, которые накладывает условие теоремы. Условие означает, что при переводе некоторого нетерминального символа $A_i \in \{V - T\}$ не может быть сделано одновременно несколько замен символов $\beta_{ij} \in \gamma$. Это позволяет свести СУТ к некоторому преобразованию исходной грамматики G_x и выводу в этой преобразованной грамматике слова $v \in \mathcal{L}_y$.

Пусть задан СУТ тройкой $\langle G_x, Q, U \rangle$, удовлетворяющий условиям теоремы 3.2.3., где $G_x = (V_в, T, P, S)$ $V_в$ - вспомогательный алфавит $V_в = \{V - T\}$, и задан вывод $S \overset{*}{\Rightarrow} x$ в грамматике G_x.

Правила грамматики G_x имеют семантику такого вида:

$$A_i \to \gamma_i = \alpha_{i1} \dots \alpha_{ij} \dots \alpha_{ik} \sim \gamma_{i1} \dots \gamma_{i2} \dots \gamma_{ik} = \gamma_i.$$

Задано взаимнооднозначное соответствие между символами слова γ_i и γ_i так, что некоторому α_{ij} соответствует γ_{i2} для всех $j = 1,2,\dots k$ и $2 = 1,2,\dots k$.

Образуем правила новой грамматики $G_y = \langle V_в, U, P', S \rangle$, каждому i - му правилу грамматики G_x будет соответствовать единственное правило в грамматике G_y, и левые части правил совпадают.

Введем обозначения: $P_i' = (A_i \to \gamma_{i1} \dots \gamma_{i2} \dots \gamma_{ik}) \in P'$

$$\gamma_{i2} = \begin{cases} \alpha_{ij}, & \text{если соответствующий } \gamma_{i2} \text{ символ } \alpha_{ij} \in V_в \\ v_{i2} \in U^*, & \text{если соответствующий } \gamma_{i2} \text{ символ } \alpha_{ij} \in T \end{cases}$$

Очевидно, что построенная грамматика G_y является КС - грамматикой.

Теперь, учитывая условия теоремы, что при переводе A_i не может быть сделано одновременно несколько замен, и определение перевода в СУТ, можно сделать вывод: операция СУТ при условиях теоремы эквивалентна выводу слова y в КС - грамматике G_y, и для вывода слова y применяется та же последовательность подстановок нетерминальных символов что и в исходной грамматике G_x.

Теорема доказана.

Пример 3.2.4. $\mathcal{L}_x = \{a^n b^n\}$

$$Q = \begin{cases} 1. \quad S \to asb \rightsquigarrow c^2 \beta d^3 \\ 2. \quad S \to ab \rightsquigarrow \omega \end{cases}$$

Построим грамматику G_y.

$$1.\ S \to c^2 S d^3$$

$$2.\ S \to \omega$$

Пусть задано слово $x = a^3 b^3$ и его вывод в G_x получается последовательным применением подстановок правых частей правил I.I.2.

Применим ту же последовательность подстановок в G_y и получим слово $V = c^6 \omega d^9$

Не трудно видеть, что $\mathcal{L}_y = СУТ(\mathcal{L}_x) = \{c^{2n} \omega\, d^{3n}\}$ и является КС - языком.

Приведем пример, когда $\mathcal{L}_y = СУТ(\mathcal{L}_x)$ не является КС - языком.

Пример 3.2.5. $G_x = (V, T, P, S)$

$$V = T \cup \{S, A\}, \quad T = \{a\}, \quad U = \{a, b\}$$

$$Q = \begin{cases} 1. \quad S \to A \rightsquigarrow \beta \quad \beta \quad \beta & \mathcal{L}_x = \{a^n / n \geq 1\} \\ 2. \quad A \to a \rightsquigarrow a & \\ 3. \quad A \to aA \rightsquigarrow a\beta & \mathcal{L}_y = \{a^n b a^n b a^n\} \end{cases}$$

Читателю предоставляется самому доказать, что $\mathcal{L}_y$ не КС - язык.

Пусть $x = a^3$ и оно выводимо применением правил подстановок в следующей последовательности 1,3,3,2.

Соответствующий перевод строится так:

1. $\beta \quad b \quad \beta \quad b \quad \beta$
3. $a\beta \quad b \quad a\beta \quad b \quad a\beta$
3. $aa\beta \quad b \quad aa\beta \quad b \quad aa\beta$ } — одновременная замена всех β на $a\beta$
2. $aaa \quad b \quad aaa \quad b \quad aaa$ } — одновременная замена всех β на a

Соотношение магазинного преобразователя и СУТ определяется следующей теоремой.

Теорема 3.2.6. Для каждого магазинного преобразователя МП $= (\Sigma, K, T, U, \gamma, z_0, q_0, \delta)$ существует эквивалентный СУТ.

Эквивалентность означает:

1. Язык $\mathcal{L}_x$, допускаемый МП, совпадает с языком, порождаемым грамматикой G_x СУТ.

2. Совокупность слов $\mathcal{L}_y$, в которые МП может перевести слова $\mathcal{L}_x$, совпадает с $\mathcal{L}_y = СУТ(\mathcal{L}_x)$.

3. Соответствие между $x \in \mathcal{L}_x$ и $y \in \mathcal{L}_y$ которое реализует МП, реализуется и в СУТ.

Док - во: Можно считать, что автомат в МП является автоматом типа $\mathcal{N}$.

Построим грамматику G_x эквивалентную

магазинному автомату, входящему в МП, так как это делалось в ЛЛ1. $G_x = (V, T, P, S)$.

$$V = \{ A_{zpq} \mid z \in Z,\ p, q \in K \} \cup \{S\}$$

$$P = \{ A_{zpq} \to a A_{z_k z q_{k-1}} A_{z_{k-1} q_{k-1} q_{k-2}} \dots A_{z_2 q_2 q_1} A_{z_1 q_1 q} \mid$$

$$\mid (z, p, a) \vdash (z_1 z_2 \dots z_k, z, \phi) \text{ в автомате } N \} \cup$$

$$\cup \{ S \to A_{z_0 q_0 p} \mid p \in K \}$$

Теорема NG утверждает, что $L(G_x) = L(N)$

т.е. язык допускаемый автоматом N и язык порождаемый грамматикой, построенной таким образом совпадают.

Дополним грамматику G_x до СУТ.

Если в МП есть переход такого типа

$$(z, p, a, \phi) \vdash (z_1 \dots z_k, z, \phi, v) \qquad v \in U^*,$$

то соответствующему правилу грамматики приписывается следующая семантика:

$$A_{zpq} \to a A_{z_k z q_{k-1}} A_{z_{k-1} q_{k-1} q_{k-2}} \dots A_{z_1 q_1 q} \rightsquigarrow v \beta_2 \beta_3 \beta_4 \dots \beta_{k+1}$$

Для правила $S \to A_{z_0 q_0 p} \rightsquigarrow \beta_1$.

Рассмотрим частные случаи:

Если $(z, p, a, \phi) \vdash (\phi, z, \phi, v)$ -стирание магазина,

то $A_{zpz} \to a \rightsquigarrow v$.

Если $(z, p, \phi, \phi) \vdash (\phi, z, \phi, v)$ - холостой такт,

то $A_{zpz} \to \phi \rightsquigarrow v$.

Лемма 3.2.7. Если существует переход в МП

$$(z, p, x, \phi) \vdash^* (\phi, q, \phi, v)$$

то существует перевод в СУТ, построенном по МП

$$A_{zpq} \overset{*}{\Longrightarrow} x \approx v.$$

Док - во: индукцией по числу тактов работы МП.

I. Пусть такой переход выполняется за один такт, тогда $(z, p, x, \phi) \vdash (\phi, q, \phi, v)$ и $x = a$ состоит ровно из одного символа. Для такого перехода правило перевода в СУТ будет иметь вид:

$$A_{zpq} \to a \rightsquigarrow v$$

2. Пусть лемма доказана для всех тактов работы МП меньших n , докажем предложение леммы для n тактов. На вход МП подается слово x , а на выходе в результате перевода получается слово $v = yv'$

Другими словами имеет место следующий переход: $(z, p, x, \phi) \vdash^* (\phi, q, \phi, v)$. Разобьем этот вывод на две части, одна часть будет соответствовать переходу за один первый такт работы МП, так что $x = ax'$ и $v = yv'$, а другая часть соответствует всей последующей работе МП. Рассмотрим отдельно переход: $(z_1 \dots z_k, z, x', y) \vdash^* (\phi, q, \phi, yv')$.

Заметим, что МП последовательно должен стирать символы в магазине $z_k, z_{k-1}, \dots z_1$, при этом МП переходит через некоторое состояния $q_{k-1}, q_{k-2}, \dots q_1, q$ (которые, вообще говоря, могут и совпадать). Работу по стиранию одного z_i символа из магазина назовем шагом работы МП (шаг может быть равен одному или нескольким тактам работы МП).

Запишем пошаговую работу МП.

$\{(z_1 \dots z_k; z, x', y) \vdash^{*} (\phi, q, \phi, y v')\} =$

$= \{(z_1 \dots z_k, z, x'y) \vdash^{*} (z_1 \dots z_{k-1}, q_{k-1}, x_{k-1} x_{k-2} \dots x_1, y v_k)$

$\dots \vdash^{*} (z_1 \dots z_{k-2}, q_{k-2}, x_{k-2} \dots x_1, y v_k v_{k-1}) \vdash^{*} \dots$

$\dots \vdash^{*} (z_1, q_1, x_1, y v_k v_{k-1} \dots v_2) \vdash^{*}$

$\vdash^{*} (\phi, q, \phi, y v_k v_{k-1} \dots v_1) \}$

где $x' = x_k x_{k-1} \dots x_1$, $x_i \in T$ либо $x_i = \phi$

$v' = v_k v_{k-1} \dots v_1$, $v_i \in U^*$ либо $v_i = \phi$.

Каждый i - ый шаг работы МП характеризуется изменением составляющих ситуаций, что можно записать в следующем виде:

$$(z_i, q_i, x_i, \phi) \vdash^{*} (\phi, q_{i-1}, \phi, v_i)$$

Каждый шаг работы МП является частью всего вывода, таким образом для отдельного шага справедливо предположение индукции.

Следовательно можно записать:

$$A_{z_i} q_i q_{i-1} \overset{*}{\Longrightarrow} x_i \rightsquigarrow v_i$$

Первому такту работы МП соответствует правило грамматики Gx и перевод

$$A_{zpq} \to a A_{z_k z q_{k-1}} \dots A_{z_1 q_1 q} \rightsquigarrow y \beta_2 \beta_3 \dots \beta_{k+1}.$$

Заменяя β_i на v_{i-1}, получим $A_{zpq} \overset{*}{\Longrightarrow} x \rightsquigarrow v$.

Лемма доказана.

Лемма 3.2.6. Если в СУТ, построенной по МП, существует перевод $A_{zpq} \overset{*}{\Longrightarrow} x \rightsquigarrow v$, то существует эквивалентный переход в исходном МП $(z, p, x, \phi) \vdash^{*} (\phi, q, \phi, v)$.

Док - во: проведем индукцией по длине вывода $A \overset{*}{\Longrightarrow} x$

Пусть вывод делается за один шаг, т.е. $A_{zpq} \to a \rightsquigarrow v$

то в МП существует переход $(z, p, a, \phi) \vdash (\phi, q, \phi, v)$.

Предположим, что лемма доказана для всех переводов длинны меньше n.

Пусть в СУТ имеется перевод

$$A_{zpq} \to a A_{z_k z q_{k-1}} \dots A_{z_1 q_1 q} \overset{*}{\Longrightarrow} x \rightsquigarrow v$$

длины n.

Назовем шагом перевода, перевод каждого нетерминального символа в правой части правила, этот факт запишем следующим образом: $A_{z_i q_i q_{i-1}} \overset{*}{\Longrightarrow} x_i \rightsquigarrow v_i$

Тогда вывод x из A_{zpq} и перевод v можно представить последовательностью из k шагов.

$$x = a x_k x_{k-1} \dots x_1 = a x'$$

$$v = y v_k v_{k-1} \dots v_1 = y v'$$

Заметим, что шаг перевода, есть перевод длины меньше n и для него выполнены условия индукции, т.е. каждому шагу перевода в СУТ есть эквивалентный переход в МП. По построению: правилу перевода

$$A_{zpq} \to a A_{z_k z q_{k-1}} \dots A_{z_1 q_1 q} \rightsquigarrow y \beta_2 \beta_3 \dots \beta_{k+1}$$

соответствует переход в МП

$$(z, p, a, \phi) \vdash (z_1 \dots z_k, z, \phi, y).$$

Учитывая предыдущее, можно записать работу МП следующим образом:

$$(z, p, ax', \phi) \vdash (z_1 \dots z_k, z, x', y) \vdash^{*}$$

$$\vdash^{*} (z_1 \dots z_{k-1}, x_{k-1} x_{k-2} \dots x_1, y v_k) \vdash^{*} \dots$$

$$\dots \vdash^{*} (z_1, q_1, x_1, y v_k v_{k-1} \dots v_2) \vdash^{*}$$

$$\vdash^{*} (\phi, q, \phi, y v_k v_{k-1} \dots v).$$

Лемма доказана.

Выше был приведен метод построения СУТ по заданному МП, оказалось, что так построенный СУТ производит "не меньший" перевод нежели исходный МП (лемма 3.2.7.) - МП($\mathcal{L}x$) $\leq$ СУТ($\mathcal{L}x$), с другой стороны построенный СУТ делает и "не больше", чем МП (лемма 3.2.8.) МП($\mathcal{L}x$) $\geq$ СУТ($\mathcal{L}x$).

Таким образом, утверждения леммы 3.2.7. и леммы 3.2.8. доказывают теорему 3.2.6. Обратное не верно. Мы уже встречались с примерами отображений, которые невозможно реализовать с помощью МП, теперь покажем как их можно все-таки сделать при помощи СУТ.

Пример 3.2.9. (см. пример 3.I.7.) СУТ, переводящий формулу алгебры логики, записанную при помощи операций $\neg$ (отрицание), $\circ$ (конъюнкция), $\vee$ (дизъюнкция) в эквивалентную формулу алгебры логики, записанную при помощи операций $\oplus$ (сложение mod_2), $\circ$ (конъюнкция), 1 (const 1).

G_x	семантика
1. $S \to F$	$\rightsquigarrow \beta_1$
2. $F \to F \cdot F$	$\rightsquigarrow \beta_1 \circ \beta_2$
3. $F \to \neg F$	$\rightsquigarrow \beta_1 \oplus 1$
4. $F \to (F \vee F)$	$\rightsquigarrow \beta_1 \oplus \beta_2 \oplus \beta_1\beta_2 \oplus 1$
5. $F \to x_1$	$\rightsquigarrow x_1$
. . . . .	. . . .
. . . . .	. . . .
. . . . .	. . . .
n+5. $F \to x_n$	$\rightsquigarrow x_n$

Построение перевода некоторого слова предоставляется читателю.

Пример 3.2.IO. СУТ, реализующий соответствие

$$\mathcal{L}_x = \{a^n\} \rightsquigarrow \mathcal{L}_y = \{\beta^{2^{n-1}}\}$$
$$S \to Sa \rightsquigarrow \beta\beta$$
$$S \to a \rightsquigarrow \beta.$$

Не смотря на бо́льшую"мощность" метода СУТ по сравнению с МП, существуют такие соответствия между $\mathcal{L}x$ и $\mathcal{L}y$, которые не могут быть реализованы при помощи СУТ. Приведем один пример.

Пример 3.2.II. Построить СУТ, реализующий перевод

$$\mathcal{L}_x = \{a^n\} \rightsquigarrow \mathcal{L}_y = \{a^n b^n c^n\}$$

Покажем невозможность построения СУТ для такого перевода. Грамматика G_x состоит из двух правил.

1. $S \to aS$
2. $S \to a$

Заметим, что $\mathcal{L}_y = \{a^n b^n c^n\}$ – не КС - язык, следовательно, семантика I-го правила может быть задана словом ψ_1, содержащим не менее двух одинаковых символов β. Семантика для второго правила должна быть задана словом ψ_2, которое в обязательном порядке содержит буквы a, b, c, и, кроме того, одинаковое их количество, иначе при переводе мы не получили бы соответствие $a^n \rightsquigarrow a^n b^n c^n$. Например, семантика правил может быть такой:

1. $S \to aS \rightsquigarrow \psi_1 = \beta\beta\beta$
2. $S \to a \rightsquigarrow \psi_2 = abc$

При таких условиях на семантику синтаксически управляемая трансляция не позволяет получить слова y,

у которых сначала следовали только символы a , за ними только b и за ними только c .

В заключение приведем несколько примеров применений метода СУТ для некоторых фрагментов ALGOL 60.

Пример 3.2.12. Перевод арифметического выражения в ПОЛИЗ. Операции $\times$ и $+$, буква (δ), число (z).

$B1$ - выражение, t - переменная.

1. $B1 \to \delta \rightsquigarrow \delta$
2. $B1 \to z \rightsquigarrow z$
3. $B1 \to (B) \rightsquigarrow \beta_3$
4. $t \to B1 \rightsquigarrow \beta_2$
5. $t \to t \times B1 \rightsquigarrow \beta_1 \beta_2 \times$
6. $B \to t \rightsquigarrow \beta_1$
7. $B \to t + B \rightsquigarrow \beta_1 \beta_3 +$

выражение - $(\delta + z) \times \delta + \delta$

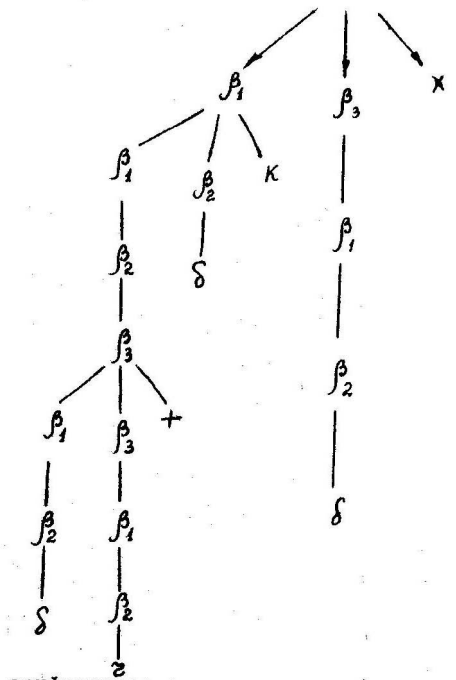
$$B \underset{7}{\Rightarrow} t + B \underset{5}{\Rightarrow} t + B1 + B \underset{4}{\Rightarrow} B1 \times B1 + B \underset{3}{\Rightarrow} (B) \times B1 + B \underset{7}{\Rightarrow}$$

$$(t + B) \times B1 + B \underset{4}{\Rightarrow} (B1 + B) \times B1 + B \underset{1}{\Rightarrow} (\delta + B) \times B1 + B \underset{6}{\Rightarrow}$$

$$(\delta + t) \times B1 + B \underset{4}{\Rightarrow} (\delta + B1) \times B1 + B \underset{2}{\Rightarrow} (\delta + z) \times B1 + B \underset{1}{\Rightarrow}$$

$$(\delta + z) \times \delta + B \underset{6}{\Rightarrow} (\delta + z) \times \delta + t \underset{4}{\Rightarrow} (\delta + z) \times \delta + B1 \underset{1}{\Rightarrow} (\delta + z) \times \delta + \delta$$

ПОЛИЗ



арифметическое выражение в ПОЛИЗ	машинная программа
	СМ δ
	СМ z
	17 АС
$\delta z + \delta \times + \delta +$	СМ δ
	17 АЧ
	СМ δ
	17 АС

Пример 3.2.13. Язык операторов присваивания.

$a:=b:=\ldots:=c:=B$; B - выражение

	семантика
$Q \to \delta 1:=B$	$\rightsquigarrow \beta_3:=\beta_4$
$Q \to \delta 1:=0$	$\rightsquigarrow \beta_5:=\beta_4$
$\delta 1 \to a$	$\rightsquigarrow a$
. . . . .	
$\delta 1 \to c$	$\rightsquigarrow c$

Дерево перевода

$\beta_5:=\beta_4$
|
$\beta_3:=\beta_4$
|
$B:=b:=a$

Машинная программа - В 34 В 34

З Ч - запись числа.

Глава 4.

РЕАЛИЗАЦИЯ.

В предыдущей главе была рассмотрена задача перевода (трансляции) с одного языка L_1 на другой L_2 в предположении, что известны некоторые локальные законы перевода (СУТ). Эти локальные законы можно рассматривать как придание смысла в языке L_2 некоторым понятиям и конструкциям из языка L_1. Однако проблема формального описания семантики (т.е. смысла объектов) языков программирования пока не решена и поэтому при разработке трансляторов одной из основных и, повидимому, самых трудных задач является отыскание в машинном языке ("язык L_2") конструкций, эквивалентных (т.е. дающих при работе ЭВМ желаемый с точки зрения языка L_1 и записанного в нем алгоритма-программы результат) конструкциями языка L_1. Или, как принято говорить, задача состоит в реализации конструкций (или всего)языка L_1 в языке L_2.

Сложность реализации в очень большой степени зависит от того, насколько схожи языки L_1 и L_2. Так, сравнительно просто реализовывать на ЭВМ языки ассемблеров, программы в которых представляют собой, как правило, те же машинные программы, записанные лишь с использованием более удобной символики. С другой стороны, появились машины, внутренний язык которых подобен языку Алгол-60 и содержит многие элементы последнего; для таких машин упрощается задача трансляции с Алгол-60 и похожих на него языков. Однако в большинстве случаев язык программирования (L_1) и машинный язык (L_2) имеют мало общего. Как уже говорилось, общих методов реализации не существует, и мы разберем ниже достаточно типичный и достаточно слож-

ный случай реализации основных элементов языка АЛГОЛ-60 на машине с привычной ("классической") структурой команд. Следуя идеям предыдущей главы, мы будем давать способы "локального" перевода. В целом, материала 3 и 4 глав достаточно, чтобы можно было написать синтаксически управляемый транслятор с языка АЛГОЛ-60 на какую-либо из реальных ЭВМ.

В целях упрощения изложения и придания ему большей строгости мы несколько ограничим свою задачу, произведя в языке АЛГОЛ-60 ряд модификаций, а именно:

I. Мы не будем учитывать тонкостей, связанных с наличием двух типов арифметических величин - *real* и *integer* .

2. Мы не будем допускать целых меток.

3. При записи указателя переключателя мы будем использовать не квадратные, а круглые скобки с тем, чтобы он больше походил на указатель функции.

4. Мы будем считать, что в языке нет величин типа *own* .

Усложнения, которые необходимо сделать в нижеследующем изложении для "полного" АЛГОЛа-60, весьма невелики и мы надеемся, что они будут по силам любому читателю, который разберется во всех тонкостях описываемой здесь реализации.

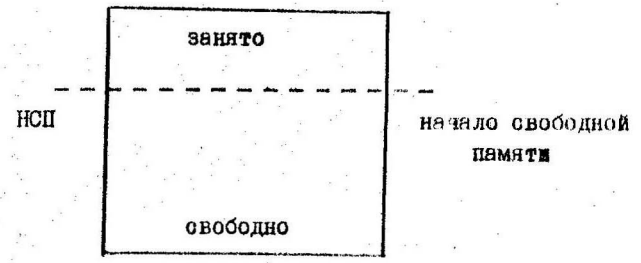
§ I. Магазинный принцип использования памяти.

Структура языка АЛГОЛ-60 хорошо согласуется с магазинным принципом использования памяти в масштабах всей программы (в гл.2 магазинный принцип демонстрировался на примере трансляции арифметических выражений). Одна из основных особенностей АЛГОЛа-60 состоит в том, что память для величин, описанных в каком-либо блоке, может выделяться лишь в те

моменты выполнения алгол-программы, когда управление входит (через *begin*) в этот блок и находится в начале блока, т.е. в области описаний. При выходе из этого блока (через соответствующий *end* или с помощью оператора перехода *go to* ...) память, выделенную для этих величин, можно освободить. Все входы и выходы из блоков осуществляются в соответствии с принципом: из двух блоков, в которые вошло управление, раньше оно выйдет из того блока, в которое вошло позже. Это позволяет использовать один магазин для размещения в нем всех величин, описанных в программе: при входе в блок в магазине выделяется секция памяти для величин, описанных в этом блоке; при выходе из блока эта секция освобождается. Обращение к процедуре (или к процедура-функции) мы будем трактовать как вход в блок, и все что в дальнейшем говорится о блоках, будет относиться также и к процедурам.

В некотором регистре (назовем его НСП) будет постоянно храниться номер первой свободной ячейки памяти, а всю память (точнее весь магазин) мы будем рассматривать, как один одномерный массив П[I : N]. Таким образом в любой момент занята память от П[I] до П[НСП-I] и свободна от П[НСП] до П[N];

Массив П (магазин)



Вначале, т.е. перед работой программы, НСП-I. Номер какой-либо ячейки магазина мы будем называть адресом этой ячейки. Последняя занятая ячейка магазина, т.е. П [НСП-I], будет часто упоминаться в нашем изложении и поэтому для краткости мы будем называть ее верхушкой магазина. Секция, выделяемая для блока, состоит из 5 частей. Все ячейки первых 3 частей секции будут нумероваться в пределах самой секции начиная с - I. Этот номер будет в дальнейшем называться относительным адресом соответствующей ячейки или величины, которая в ней находится.

№ по порядку	Название чисел	Относительный адрес
I	Результат	-I
2	Связь	0
3	Величины	I + m
4	Массивы	
5	Рабочая	

Результат будет содержать значение процедуры функции в тех случаях, когда блоком является процедура-функция.

Связь содержит некоторую информацию, учитываемую при запуске и завершении работы блоков и описываемую далее.

Величины - это часть секции, содержащая столько ячеек, сколько в блоке описано простых переменных, массивов и меток. В ячейке, соответствующей простой переменной, будет находиться значение этой переменной; в ячейке, соответствующей массиву - т.к. указатель массива, т.е. некоторая описываемая

далее информация о массиве; в ячейке, соответствующей метке - опять-таки описываемая далее величина, которую мы будем называть знаменем метки. Размер m этой части секции определяется "статически", т.е. по внешнему виду программы и мы будем считать, что при входе в блок нам каждый раз это число известно. М а с с и в ы - это часть магазина, выделенная под все массивы, описанные в блоке. Отметим, что размер этой части секции определяется только в момент входа в соответствующий блок, а точнее - после того, как будут просмотрены и обработаны описания всех массивов этого блока.

Р А Б О Ч А Я - часть секции, выделяемая для хранения промежуточных результатов при вычислении встречающихся в блоке выражений (например, так, как это описывалось в предыдущей главе). Нижняя граница рабочей части секции изменяется на протяжении работы блока. Перед выполнением каждого оператора рабочая часть пустая. В процессе вычисления выражения, например, правой части оператора присваивания

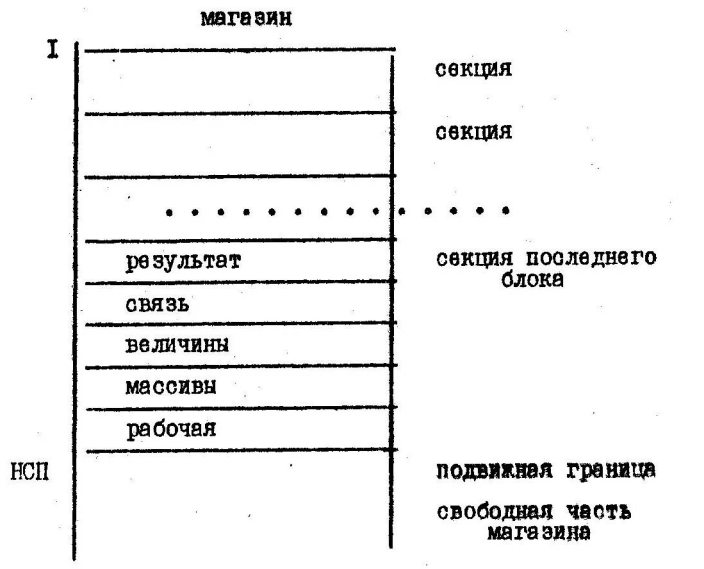
$$x := a+b+c\times(d+e),$$

рабочая часть будет последовательно содержать значения

a	a в	a+в	a+в c	a+в c d	a+в c d e	a+в c d+e	a+в c×(d+e)	a+в+c×(d+e)

и ее длина будет, таким образом, меняться от 0 до 4.

Итак, структура магазина и секции последнего блока схематично такова:



Примечание I. Максимальный размер, которого достигает рабочая часть секции, может быть вычислен при анализе текста соответствующего блока. Поэтому ее можно было бы сделать постоянной (максимальной) длины и объединить с частью в е л и ч и н ы . Однако, в этом случае, в противоположность описываемому, использование памяти было бы менее экономным.

Примечание 2. Секции будут образовываться также при обращении к процедуре (в этом случае m будет равно числу параметров процедуры) и при вычислении некоторых фактических параметров (см. § I2). В последнем случае секция будет состоять только из результата, связи и рабочей части.

§ 2. Адресация величин.

Каждому блоку припишем некоторое натуральное число, которое мы будем называть уровнем этого блока. Уровень самого внешнего блока равен I. Уровень блока, содержащегося непосредственно в блоке уровня ℓ , равен $\ell + I$. В частности, уровень процедуры на I больше уровня блока, в котором она описана; уровень блока-тела процедуры на 2 больше уровня блока, в котором описана процедура. У с л о в н ы м а д р е с о м любой величины блока ℓ будем называть пару (ℓ, d), где d – номер ячейки, выделяемый ей в части в е л и ч и н ы соответствующей секции (это может быть, например, просто порядковый номер, если подряд пронумеровать все простые переменные, массивы и метки данного блока). Если в какой-то момент управление находится в блоке уровня ℓ , то доступны должны быть все величины, описанные в этом блоке и в охватывающих блоках уровней $\ell - I$, $\ell - 2, \ldots, 2, I$. Секции соответствующих блоков к этому времени находятся, конечно, в магазине. Зная адреса нулевых ячеек этих секций $c_\ell, c_{\ell-1}, \ldots, c_2, c_1$ (для простоты мы будем называть их н а ч а л а м и с е к ц и й) мы можем получить адрес любой величины с условным адресом (i, d): это будет $c_i + d$, а соответствующая ячейка – $П[c_i + d]$. Введем в рассмотрение одномерный массив УРОВЕНЬ $[I: \ldots]$, или для более компактной записи последующих формул, $У[I: \ldots]$, где верхняя граница определяется сложностью программы и равна максимальному уровню блока. Будем считать, что в каждый момент у вектора $У$ первые ℓ (ℓ – уровень текущего

блока) компоненты равны, соответственно, c_ℓ, $c_{\ell-1}$, ..., c_1. Тогда адрес величины с условным адресом (i, d) будет $V[i]+d$, а соответствующая ячейка – $П[V[i]+d]$. Можно предполагать, что преобразование (i, d) в $V[i]+d$ осуществляется с помощью какой-либо вспомогательной подпрограммы или, как во многих машинах, аппаратно, и прямо писать команду вида

считать (i, d)

если, например, надо выбрать из памяти значение величины x, имеющей условный адрес (i, d). Основной задачей, которую надо будет решать при каждом переходе с одного уровня на другой (т.е. при входе в блок, выходе из него и при обращениях к параметрам) – это корректировка вектора V, точнее – соответствующего числа ℓ первых компонент этого вектора, что должно обеспечить возможность обращения по вышеприведенной формуле к любым величинам, в области действия описаний которых находится управление программой.

§ 3. Структура машины.

Сейчас было бы уместно описать язык, на который осуществляется перевод, т.е. систему команд машины. Мы, однако, не будем педантичны и язык машины, в традиционном его понимании задавать не будем. Вместо этого реализация будет даваться в некоторых условных операциях, которые на любой конкретной машине легко осуществлять с помощью одной или нескольких команд, либо с помощью обращения к подпрограмме. Нужные нам операции будут вводиться постепенно. Здесь же мы

уточним 2 из них

1. С Ч И Т А Т Ь (i, d)

содержимое ячейки с условным адресом (i, d) направляется в первую свободную ячейку магазина, т.е. в $П[НСП]$, после чего НСП увеличивается на 1 (так как свободная память теперь будет начинаться со следующей ячейки).

В *алгольной* записи это эквивалентно операторам:

$П[НСП] := П[V[i]+d]$;

НСП := НСП + 1 ;

2. З А П И С А Т Ь (i, d)

содержимое последней занятой ячейки магазина (т.е. верхушки магазина) пересылается в ячейку с условным адресом (i, d). Верхушка магазина освобождается. В *алгольной* записи это означает:

$П[V[i]+d] := П[НСП-1]$;

НСП := НСП-1 ;

Если (i, d) условный адрес величины α, то для наглядности мы будем часто писать С Ч И Т А Т Ь α или З А П И С А Т Ь α. Точно также С Ч И Т А Т Ь 5 будет означать обращение к ячейке, в которой записана константа 5.

3. Арифметические операции всегда выполняются над двумя последними занятыми ячейками магазина. Результат помещается в предпоследнюю, а последняя освобождается. Например, Д Е Л Е Н И Е означает :

$П[НСП-2] := П[НСП-2] / П[НСП-1]$;

НСП := НСП - 1 ;

Не связывая себя заранее размерами ячеек и форматом хранимой в них информации, мы будем иногда помещать в одну ячейку несколько элементов информации, давая им различные названия. Например, если в ячейку П $[1]$ нам надо поместить 2 адреса α и β, означающие адреса 2-х секций- секция I и секция 2, то мы будем писать операторы

$$П\ [1] := (\alpha, \beta);$$

или

$$П\ [1] \text{ . секция I: } = \alpha;$$

$$П\ [1] \text{ . секция 2: } = \beta;$$

Первый оператор означает изменение содержимого всей ячейки П$[1]$на (α, β). Второй означает изменение той части ячейки П $[1]$, которая названа "секция I", на α. Третий оператор изменяет значение части "секция 2 " на β. В изложении будет фигурировать такое с ч е т ч и к к о м а н д (сокращенно СК), обозначающий номер, а точнее - адрес текущей выполняемой команды.

Для удобства приводим список операций, которыми мы будем пользоваться в этой главе, с указанием параграфа, где эта операция описана.

Операция	§
С Ч И Т А Т Ь α	3
З А П И С А Т Ь α	3
С Ч И Т А Т Ь П О А Д Р Е С У В м а г а з и н е	4
С Ф О Р М И Р О В А Т Ь М А С С И В	4
А Д Р Е С Э Л Е М Е Н Т А М А С С И В А	4
З А П И С А Т Ь П О А Д Р Е С У В М А Г А З И Н Е	4
К О П И Я М А С С И В А	I 3

В Х О Д В Б Л О К $1, r, m$, вход	5
В Ы Х О Д И З Б Л О К А r	8
П Е Р Е Х О Д	7
В Ы Х О Д В П Р О Ц Е Д У Р У $1, r, m$, вход	5, II
В Ы Х О Д И З П Р О Ц Е Д У Р Ы r	9
П А Р А М Е Т Р r, m	I2
К О Н Е Ц П А Р А М Е Т Р А e	I2
К О Н Е Ц О П И С А Н И Й r	5

§ 4. Адресация элементов массива.

Считая, что элементы массива

$$A[i_1 : I_1, i_2 : I_2, \ldots, i_s : I_s]$$

лексикографически упорядочены по индексам и расположены в последовательных ячейках начиная с ячейки $a_н$, легко подсчитать номер β ячейки, в которой будет находиться элемент $A[j_1, j_2, \ldots j_s]$:

$$\beta = a_н + (j_s - i_s) + (j_{s-1} - i_{s-1})(I_s - i_s + 1) +$$
$$(j_{s-2} - i_{s-2})(I_{s-1} - i_{s-1} + 1)(I_s - i_s + 1) + \ldots$$
$$(j_1 - i_1)(I_2 - i_2 + 1)(I_3 - i_3 + 1) \ldots (I_s - i_s + 1)$$

Это выражение целесообразно преобразовать к виду

$$\beta = a_o + j_s + j_{s-1} d_s + j_{s-2} d_{s-1} d_s + \ldots + j_1 d_2 d_3 \ldots d_s, \quad (*)$$

где $d_t = I_t - i_t + 1$ -диапазон изменения t-го индекса,

$$a_o = a_н - (i_s + i_{s-1} d_s + i_{s-2} d_{s-1} d_s + \ldots + i_1 d_2 d_3 \ldots d_s)$$

Как легко видеть, a_o вычисляется по той же формуле, что и β но для нулевых индексов, т.е. это номер ячейки, в которой находится элемент $A[0, 0, 0, \ldots 0]$. Таким образом, для того чтобы можно было вычислять адрес любого элемента

массива

$$A[j_1, j_2, \ldots, j_s],$$

надо, помимо его индексов $j_1, j_2, \ldots, j_s$, знать следующие величины:

$$\alpha_0, \delta_2, \delta_3, \ldots, \delta_s$$

Эти значения должны быть вычислены и запомнены в момент обработки описания массива А, т.е. при входе в соответствующий блок. Совокупность величин

$$s, \delta_1, \delta_2, \ldots, \delta_s$$

будем называть ф у н к ц и е й о т о б р а ж е н и я м а с с и в а . Эти величины будут располагаться в последовательных ячейках памяти и адрес первой из них (т.е. адрес S) будем называть адресом функции отображения массива, обозначая его через α_{φ_1}. Формировать функцию отображения массива, адреса $\delta_0, \alpha_H, \delta_\varphi$ и выделять место для массива будет описываемая ниже операция С Ф О Р М И Р О В А Т Ь М А С С И В . Перед выполнением этой операции вычисляются и помещаются в магазин последовательно все $2s$ границы и число s :

$$i_1, I_1, i_2, I_2, \ldots, i_s, I_s, s$$

затем выполняется операция

С Ф О Р М И Р О В А Т Ь М А С С И В

Зная, что размерность массива равна s (число в верхушке магазина), эта операция образует функцию отображения массива $s, \delta_1, \ldots \delta_s$ на месте первых $s+1$ величин из последовательности $i_1, I_1, \ldots$. Вслед за этим она выделяет место для массива (т.е. увеличивает НСП еще на $\delta_1 \times \delta_2 \times \ldots \times \delta_s$) и помещает в очередную свободную ячейку магазина три адреса α_0, a_H, a_φ . В виде формул все это можно записать так :

П [НСП - 2s - 1] := s ;
П [НСП - 2s] := $I_1 - i_1 + 1$;
П [НСП - 2s + 1] := $I_2 - i_2 + 1$;
. . . . . .
П [НСП - s - 1] := $I_s - i_s + 1$;
НСП := НСП - s + $\delta_1 \times \delta_2 \times \ldots \times \delta_s$;
П [НСП] := (a_0, a_H, a_φ);
НСП := НСП + 1 ;

Если условный адрес массива А был (i, δ) , то надо еще выполнить операцию

З А П И С А Т Ь (i, δ)

Тройку (a_0, a_H, a_φ) мы для краткости будем называть указателем массива А.

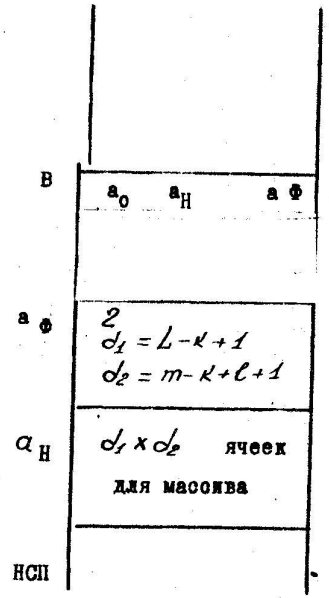
Пример: Пусть дано описание массива

$$array\ B[k : \ell, k-\ell : m];$$

оно превратится в следующие операции

С Ч И Т А Т Ь k
С Ч И Т А Т Ь ℓ
С Ч И Т А Т Ь k
С Ч И Т А Т Ь ℓ
В Ы Ч Е С Т Ь
С Ч И Т А Т Ь m
С Ч И Т А Т Ь 2
С Ф О Р М И Р О В А Т Ь М А С С И В
З А П И С А Т Ь B

Память будет выглядеть следующим образом :



Функция отображения массива B

Получение адреса элемента массива осуществляется с помощью операции

А Д Р Е С Э Л Е М Е Н Т А М А С С И В А

Перед ее выполнением надо в магазин поместить значения индексов и указатель массива:

$$j_1, j_2, \ldots, j_s \ (a_0, a_H, a_\Phi)$$

Операция А Д Р Е С Э Л Е М Е Н Т А М А С С И В А, обратившись по адресу a_Φ узнает размерность массива и, следовательно, число индексов в магазине и коэффициентов в функции отображения. Произведя вычисления по формуле (*), она получит нужный адрес ℓ и поместит его на место j_1, уменьшив НСП на s. Коротко:

НСП : = НСП - s ;

П[НСП] : = ℓ ;

Если необходим не адрес, а значение элемента массива, то проще всего ввести еще одну операцию

С Ч И Т А Т Ь П О А Д Р Е С У В М А Г А З И Н Е,

которая аналогична операции С Ч И Т А Т Ь, но адрес ячейки берет из верхушки магазина:

П [НСП - I] : = П [П [НСП - I]] ;

Пример: Оператор A [K] : = A[$k+\ell$] + D будет реализован операциями

С Ч И Т А Т Ь K
С Ч И Т А Т Ь A
А Д Р Е С Э Л Е М Е Н Т А М А С С И В А } адрес A[K]
С Ч И Т А Т Ь K
С Ч И Т А Т Ь ℓ
С Л О Ж И Т Ь } адрес
С Ч И Т А Т Ь A
А Д Р Е С Э Л Е М Е Н Т А М А С С И В А } A[K+ℓ]
С Ч И Т А Т Ь П О А Д Р Е С У В М А Г А З И Н Е
С Ч И Т А Т Ь D
С Л О Ж И Т Ь } A[K+ℓ] + D
З А П И С А Т Ь П О А Д Р Е С У В М А Г А З И Н

Последняя операция примера

З А П И С А Т Ь П О А Д Р Е С У В М А Г А З И Н Е

означает следующее:

П [П [НСП - 2]] : = П [НСП - I] ;

НСП : = НСП - 2 ;

§ 5. Вход в блок.

При входе в блок (или процедуру) необходимо:

а) образовать новую секцию в памяти

б) запомнить ячейку с в я з ь этой секции

с) скорректировать вектор У Р О В Е Н Ь

d) передать управление на первую команду блока.

Операции входа в блок в качестве параметров задаются:

ℓ – уровень блока, в котором мы находимся перед входом в блок;

r – уровень блока, в который мы входим (если это обычный блок, то $r = \ell+1$; если процедура, то $r \leq \ell+1$, обращение к параметрам-процедурам будет рассмотрено в § I2);

m – число величин, описанных в блоке;

вход – адрес первой команды блока; его имеет смысл задавать лишь для процедур; для единообразия, мы будем задавать его и для обычных блоков; в большинстве реализаций, в частности, и у нас он на 1 больше номера текущей команды.

Таким образом, операция выглядит:

В Х О Д В Б Л О К $\ell, r, m,$ вход

или

В Х О Д В П Р О Ц Е Д У Р У $\ell, r, m,$ вход

Прежде, чем описывать все действия, которые надо выполнить при входе в блок, уточним содержимое ячейки с в я з ь. Ее значение состоит из 5 компонент:

I) Номер уровня данной секции (т.е. той, которой принадлежит эта ячейка). Сокращенно мы будем обозначать эту компоненту NY.

2) Адрес предыдущей секции, т.е. секции, которая работала непосредственно перед данной. Адресом секции мы будем всегда называть адрес ячейки с в я з ь этойсекции. Описываемую компоненту мы будем называть д и н а м и ч е с к и м п р е д ш е с т в е н н и к о м или ДИНПРЕД.

3) Адрес секции, уровень которой на I меньше уровня данной секции. Эта информация нужна для корректировки вектора УРОВЕНЬ и она называется с т а т и с т и ч е с к и м п р е д ш е с т в е н н и к о м или СТАТПРЕД.

4) Адрес возврата, т.е. номер команды, которой надо передать управление после выхода из блока через end. Эта компонента, называемая ВОЗВРАТ, заполняется только для процедур (и для описываемых в § II параметров). В некоторых реализациях программа каждого блока составляется независимо, т.е. она выносится из программы охватывающего блока, а в последнем остается лишь операция обращения к вынесенному блоку. При таком варианте компонента ВОЗВРАТ заполняется и для обычных блоков.

5) Начало р а б о ч е й ч а с т и секции или НРr.

Итак, связь состоит из:

NY

ДИНПРЕД

СТАТПРЕД

ВОЗВРАТ

НРr

Из всего вышесказанного уже приблизительно ясно какие действия выполняет операция В Х О Д В Б Л О К и В Х О Д В П Р О Ц Е Д У Р У. Запишем их явно (секция располагается

начиная с ячейки НСП, связь попадает в ячейку НСП + I).

	примечания
П [НСП + I]. $NУ$: = r ;	пункт в)
П [НСП + I]. ДИНПРЕД : = $У[l]$;	"
П [НСП + I]. СТАТПРЕД : = $У[r-1]$;	"
П [НСП + I]. ВОЗВРАТ : = $cк+1$;	"
$У[r]$: = НСП + I ;	пункт с)
НСП : = НСП + 2 + m ;	пункт а)
СК : = вход	пункт d)

Не заполнена лишь компонента НРr связи. Однако ее значение определится лишь после того, как будут обработаны описания всех массивов блока. Поэтому в конце описаний ставится операция

К О Н Е Ц О П И С А Н И Й , r

где r -уровень данного блока. В результате ее выполняется необходимое присваивание

П[$У[r]$]. НРr : = НСП ;

Значение этой компоненты (т.е.НРr) используется только при передачах управления на метки данного блока, а поэтому несколько "запоздалое" определение ее значения ничему повредить не может.

Коррекция вектора $У$ состоит в данном случае в изменении значения только одной компоненты $У[r]$. Этого достаточно, так как $У[r-1]$..., $У[1]$ уже имеют необходимые значения: они соответствуют тем секциям уровней $r-1, \dots, 1$ которые были доступны и из предыдущего блока уровня l .

§ 6. Обработка описаний.

Описания простых переменных игнорируются. Реализация описаний массивов рассмотрена в § 4.

Тела процедур транслируются так же, как и блоки, а о трансляции заголовка процедуры будет сказано в § I3. Так как описание процедур при входе в блок выполнять не надо, то перед каждым из них ставится операция передачи управления на конец описания, т.е. обхода этого описания.

Описание переключателя превращается в описание процедуры-функции, выдающей значение типа метка по следующей схеме: описание переключателя

switch $s := d1, d2, \dots, dt$;

соответствующее описание процедуры:

```
label procedure s(α) value α; integer α;
s: = if α=1 then d1 else
     if α=2 then d2 else...
     if α=t then dt else null ;
```

Что такое значение типа метка (в частности, значения именующих выражений $d1, d2, \dots, dt$, а также null , сказано далее в этом же параграфе. Используемый здесь идентификатор α - это некоторый служебный идентификатор, который нигде в описании переключателя не встречается.

Значения типа метка - это пара $(c, к)$, где $к$ - адрес оператора, помеченного данной меткой, а c -адрес начала секции, соответствующей некоторому запуску данного блока. Заметим, что нетрудно привести пример программы на АЛГОЛе, в которой в процессе ее выполнения будет одновременно

существовать несколько значений $(c_1, к)$, $(c_2, к)$ с одинаковыми $к$, но разными c. *nuee* -это некоторое условное значение отличное от любого возможного реального значения типа метка.

При обработке описаний блока необходимо добавить операции, которые будут присваивать каждой величине, соответствующей каждой описанной в данном блоке метке (как об этом сказано в § I), значение типа метка

$$(у[р], к)$$

где $к$ - адрес команды, с которой начинается оператор, помеченный этой меткой, а $у[р]$ - адрес секции, соответствующей рассматриваемому блоку. Об использовании этих значений будет сказано в § 7.

§ 7. Выход из блока посредством оператора *goto*

Мы будем считать, что значение именующего выражения в операторе *goto* вычислено и находится в верхушке магазина:

П [НСП - I] = $(с, к)$

Задача, которую надо выполнить операции ПЕРЕХОД, состоит в том, чтобы:

а) освободить память от ненужных более величин (т.е. от секций, соответствующих блокам, из которых мы выходим),

в) скорректировать вектор УРОВЕНЬ,

с) передать управление на нужную команду.

Реализация пункта а) :

НСП := П [с] НРг

Реализация пункта в) :

пусть П [с], *NУ*:= $р$, тогда

$у[р] := с$;

$у[р-1] := П[у[р]]$; СТАТПРЕД;

$у[р-2] := П[у[р-1]]$; СТАТПРЕД;

. . . . . .

$у[1] := П[у[2]]$; СТАТПРЕД;

Реализация пункта с) :

$сК := к$

Пояснения: Должен быть запущен блок с секцией, начинающейся в ячейке $с$. В этой ячейке находится с в я з ь, т.е. информация, которая позволяет установить необходимое значение НСП (это НР2, т.к. перед выполнением каждого оператора блока рабочая часть секции пуста) и компоненты вектора УРОВЕНЬ (самая старшая компонента, номер $р$ которой узнается из с в я з и, делается равной $с$, а значение всех остальных получаются из цепочки статических предшественников).

Задача I. Убедиться, что при передаче управления внутри одного блока с помощью оператора *goto* можно пользоваться той же схемой и, следовательно, той же операцией ПЕРЕХОД.

Задача 2. Читателю предоставляется самому выяснить, какие возможны упрощения в операции ПЕРЕХОД, если

I) Оператор *goto* находится в области действия метки, на которую передается управление.

2) Выполнение оператора *goto* не связано с выходом из какого-либо блока.

Задача 3. Последовательность присваеваемый компонентом вектора $У$ можно проводить не для всех компонент $у[р]$, $у[1]$ а для меньшего числа (ясно, что на протяжении решения всей задачи $у[1]$ не меняется). Выяснить, сколько присваиваний необходимо делать.

Примечание: В языке АЛГОЛ-60 много тонких мест. К их числу относится и такая фраза . "При неопределенном указателе переключателя оператор перехода эквивалентен пустому оператору". Ее следует понимать следующим образом: если мы получим *пусто* в качестве значения именующего выражения, то никаких действий, перечисленных в пунктах а) - с), делать не надо. Ясно, как это реализовать: операция ПЕРЕХОД прежде всего должна выполнять соответствующую проверку.

§ 8. Выход из блока через *end*

При выходе из блока через его *end* необходимо лишь ликвидировать одну (последнюю) секцию. Вектор УРОВЕНЬ корректировать, очевидно, не надо, так как должен сохраняться доступ к секциям всех старших блоков. Таким образом операция

ВЫХОД ИЗ БЛОКА z,

где z - уровень текущего блока, реализуется так:

НСП := $У[z]$ -I

§ 9. Выход из процедуры через *end*

При выходе из процедуры через *end* , т.е. при нормальном завершении работы процедуры, необходимо, как и для блока (§ 8) , ликвидировать последнюю секцию. Так как, однако, процедура могла быть процедурой-функцией, то в магазине надо сохранить ее результат. Кроме того, надо скорректировать вектор УРОВЕНЬ аналогично тому, как это было сделано в § 7. Итак,

ВЫХОД ИЗ ПРОЦЕДУРЫ z

реализуется следующим образом :

пусть П $[У[z]]$. ДИНПРЕД $\equiv c$;
и П $[c]$. $NУ := P$;
тогда НСП := $У[z]$;

$У[P] := c$;
$У[P-1] := П[У[P]]$. СТАТПРЕД;
$У[P-2] := П[У[P-1]]$. СТАТПРЕД;
. . . . . .
$У[1] := П[У[z]]$. СТАТПРЕД;
$cK := П[У[z]]$. ВОЗВРАТ

Чтобы после выполения оператора процедуры в магазине не "застревал" р е з у л ь т а т , т.е. первая ячейка ликвидируемой секции, после каждого оператора процедуры надо уменьшить НСП на I :

НСП := НСП - I ;

Задача: Решить задачу, аналогичную задаче 3 из § 7.

§ IO. Значение, вычисляемое процедурой-функцией.

В теле процедуры функции f должен встречаться оператор присваивания с идентификатором f в левой части. Если уровень процедуры равен z , то условным адресом f считается $(z,-1)$. Тогда операция ЗАПИСАТЬ $(z,-1)$ направить вычисленное значение в ячейку соответствующей секции с относительным адресом - I, т.е. в самую первую ячейку этой секции. При выходе из процедуры этот результат, т.е. значение функции, остается в верхушке магазина.

§ II. Обращение к процедуре.

При обращении к процедуре (как с помощью оператора процедуры, так и к процедуре-функции) необходимо в дополнение

к тому, что сказано в § 5 об операции ВХОД В ПРОЦЕДУРУ выполнить еще некоторые действия, связанные с возможным наличием у процедуры фактических параметров.

При обращении к процедуре задается некоторая информация о фактических параметрах этого обращения; эту информацию о каждом параметре мы так и будем называть ф а к т и ч е с к и й п а р а м е т р. Операция ВХОД В П Р О Ц Е Д У Р У будет перерабатывать каждый фактический параметр в то, что мы будем называть а р г у м е н т о м, и помещать аргумент в ту ячейку выделяемой для процедуры секции, которая соответствует рассматриваемому параметру (это i -ая ячейка части "величины" секции для i -го по счету параметра процедуры). Отметим, что фактические параметры образуются транслятором и, следовательно, для данного обращения к процедуре при любом его выполнении будут одними и теми же, однако аргументы, которые образуются каждый раз лишь в момент выполнения этого обращения, будут, как это мы увидим далее, различными.

Мы будем различать следующие типы фактических параметров.
Тип 1: простая переменная, идентификатор массива, метка,
Тип 2: выражение (арифметическое, логическое или именующее), которое нельзя отнести к типу 1.
Тип 3: идентификатор процедуры или переключателя.
Тип 4: формальный параметр.
Для каждого фактического параметра типа 2 необходимо иметь программу (программа фактического параметра), которая вычисляет значение этого параметра или его адрес (если это, например, переменная с индексами). Для единообразия мы

будем предполагать, что программа фактического параметра всегда вычисляет адрес и выдает его в верхушке магазина. Для фактического параметра вида, например, $a+b$ это будет означать, что вычисляется сумма $a+b$, затем она помещает ся в какую-либо служебную ячейку, а в магазин направляется в качестве результата работы программы фактического параметра адрес этой служебной ячейки. Очевидно, что составление программы для фактического параметре типа 2 осуществляется практически так же, как и для правой части оператора присваивания, о чем говорилось в предыдущей главе.

Итак, обращение к процедуре с m параметрами будет выглядеть следующим образом:

K	В Х О Д В П Р О Ц Е Д У Р У ℓ, r, m, вход
K+1	Передача управления на продолжение программы
K+2	I-ый фактический параметр
:	2-ой фактический параметр
	. . . . .
	m - ый фактический параметр
	программа i_1-го
	фактического параметра
	. . . . .
	. . . . .
	программа i_s -го
	фактического параметра

Т.е. вслед за обращением к процедуре стоит обход следующей за ним информации о m фактических параметрах, так что после обращения к процедуре из ячейки K и выполнения этой процедуре управление вернется к ячейке $K+1$ и далее - к

продолжению программы. Для тех из фактических параметров, которые относятся к типу 2, даются программы фактических параметров.

Строение фактических параметров

фактический параметр I-го типа содержит следующую информацию:

(тип I , условный адрес)

фактический параметр 2-го типа содержит информацию :

(тип 2, адрес начала программы этого параметра)

фактический параметр 3-го типа содержит информацию :

(тип 3, адрес начала программы этой процедуры - или переключателя-параметра, ее (или его) уровень)

фактический параметр 4-го типа содержит информацию:

(тип 4, условный адрес)

Операция ВХОД В П Р О Ц Е Д У Р У перерабатывает эти фактические параметры в аргументы по следующим правилам:

Фактический параметр	Аргумент
(тип I, i, δ)	(тип I, $y[i]+\delta$)
(тип 2, κ)	(тип 2, $y[e], \kappa$)
(тип 3, κ, q)	(тип 3, $y[e], \kappa, q$)
(тип 4, i, q)	$\Pi[y[i]+\delta]$

здесь (i, δ)-условный адрес фактического параметра I-го и δ-го типа

κ-адрес входа в программу фактического параметра или в программу описания процедуры (а также переключателя),

q - уровень процедуры или переключателя -параметра,

e - текущей уровень, т.е. уровень, на котором выполняется

обращение к процедуре.

Цель переработки состоит в следующем : при обращении из тела процедуры к формальному параметру исследуется информация, содержащаяся в соответствующем аргументе. Для аргументов I-го типа эта информация содержит адрес $y[i]+\delta$ переменной, метки или указателя массива, который должным образом и используется. Для аргументов 2-го типа -адрес κ программы фактического параметра. Однако, чтобы запустить эту программу, необходимо "создать подходящую обстановку", т.е. ту, которая существовала в момент обращения к процедуре. Это можно сделать, зная начало текущей в момент обращения секции, т.е. $y[e]$; связь, содержащаяся в $\Pi[y[e]]$, позволит по цепочке статических предшествований изменить должным образом (т.е. "скорректировать", как мы это называли) вектор УРОВЕНЬ. То же самое можно сказать и об аргументах типа 3: для последующего обращения к процедуре-параметру надо будет восстанавливать обстановку данного обращения, что и позволит сделать $y[e]$. Аргументов 4-го типа, как легко видеть, не будет, так как в момент рассматриваемого обращения в качестве аргумента для фактического параметра 4-го типа будет браться аргумент какого-то предыдущего уже выполненного обращения, который будет I-го, 2-го или 3-го типа.

§ I2. Обращение к параметру в теле процедуры.

В языке АЛГОЛ-60 формальный параметр может использоваться в качестве:

I. Операнда выражения (арифметического, логического или именующего);

2. Идентификтора массива у переменной с индексами.

3. Переменной списка левой части оператора присваивания.

4.Идентификатора процедуры в операторе процедуры (и, аналогично, для указателя функции или указателя переключателя);

5. Фактического параметра.

В случаях I и 2 для неформального параметра мы, пользуясь операцией СЧИТАТЬ, получали в верхушке магазина необходимое значение. Для формального параметра вместо С Ч И Т А Т Ь необходимо будет написать:

С Ч И Т А Т Ь α,
П А Р А М Е Т Р z ,
С Ч И Т А Т Ь П О А Д Р Е С У В М А Г А -
З И Н Е ,

здесь первая операция поместит в верхушке магазина аргумент, соответствующий формальному параметру α ; вторая, подробно описываемая ниже, поместит в верхушке магазина адрес значения фактического параметра (z -текущий уровень). Наконец, третья получит в магазине нужное значение. В случае 3 для неформального параметра использовалась операция З А П И С А Т Ь α . Теперь придется писать •

С Ч И Т А Т Ь α ,
П А Р А М Е Т Р z,
З А П И С А Т Ь П О А Д Р Е С У В
М А Г А З И Н Е .

В случае 4 вместо В Х О Д В П Р О Ц Е Д У Р У z, p, m , вход, придеться писать

С Ч И Т А Т Ь α
П А Р А М Е Т Р z, m

Случай 5 был разобран в § II.

Операция П А Р А М Е Т Р начинает работу с того, что смотрит, какой аргумент находится в верхушке магазина и в зависимости от его типа (I, 2 или 3) выполняет разные действия. Рассмотрим их по порядку.

Аргумент типа I. Пусть П[НСП - I] = (тип I, A). Тогда

П[НСП - I] := A

Аргумент типа 2. Пусть П[НСП - I] = (тип 2, c, K), В этом случае образуется новая секция для того, чтобы программа фактического параметра могла "на ней работать". Секция эта будет состоять только из р е з у л ь т а т а, с в я з и и р а б о ч е й части. Итак, пусть П[c].NY = ℓ. Тогда выполняются следующие действия:

П[НСП + I].NY := ℓ ;
П[НСП + I].ДИНПРЕД := $Y[r]$;
П[НСП + I].СТАТПРЕД := П[c].СТАТПРЕД ;
П[НСП + I].ВОЗВРАТ := $L+1$;
НСП := НСП + 2 ;
$Y[\ell]$:= c ;
$Y[\ell-1]$:= П[$Y[\ell]$].СТАТПРЕД ;
. . . . . .
$Y[1]$:= П[$Y[2]$].СТАТПРЕД ;
CK := K ;

Ясно, что теперь можно запускать программу фактического параметра уровня ℓ, а связь позволит обеспечить возврат на текущий уровень r и продолжение работы, начиная с команды L + I. Само собой это не произойдет, т.к. пока не

сказано, чем должна заканчиваться программа фактического параметра. Об этом надо было бы сказать в § II, однако тогда это делать было преждевременно. Итак, программа фактического параметра, находящаяся на уровне ℓ, получает необходимый адрес и помещает его в ячейку $Y[\ell]-1$, имеющую условный адрес $(\ell, -1)$, после чего выполняется операция

К О Н Е Ц П А Р А М Е Т Р А ℓ

После выполнения этой операции ячейка $Y[\ell]$ -I с искомым адресом станет верхушкой магазина, как это было обусловлено в § II. Сама операция К О Н Е Ц П А Р А М Е Т Р А ℓ совпадает с операцией В Ы Х О Д И З П Р О Ц Е Д У Р Ы и названа здесь другим именем лишь наглядности ради. Аргумент типа 3: Появление такого аргумента означает, что происходит обращение к процедуре (в частности, к функции или переключателю), т.е. что выполняется операция

П А Р А М Е Т Р r, m

а в верхушке магазина находится

(тип 3, c, ℓ, q).

Здесь необходимо выполнить те же действия, что и при операции ВХОД В П Р О Ц Е Д У Р У с той разницей, что динамическим предшественником считается текущий уровень r (и текущая секция), а вектор УРОВЕНЬ корректируется на основе секции с адресом c и соответствующей цепочки статических предшествований; переработка следующих за операцией П А Р А М Е Т Р r, m фактических параметров осуществляется так же, как и в операции ВХОД В П Р О Ц Е Д У Р У . Таким образом обработка аргумента типа 3 предоставляет собой объединение обработки аргумента типа 2 операции П А Р А М Е Т Р r

и переработки фактических параметров операции НХЦ В
П Р О Ц Е Д У Р У r, l, m, K.

§ I3. Обработка заголовка процедуры.

После входа в процедуру управление подает на группу команд, соответствующих заголовку процедуры. Если среди параметров процедуры нет вызываемых по значению, то эта группа команд пуста. Для каждого параметра - не массива a вызываемого по значению, пишутся команды

С Ч И Т А Т Ь a,
П А Р А М Е Т Р r,
С Ч И Т А Т Ь П О А Д Р Е С У В
М А Г А З И Н Е ,
З А П И С А Т Ь a.

После их выполнения в ячейке a вместо аргумента будет находиться значение соответствующего фактического параметра.

Задача: Убедиться, что это так для случаев, когда фактический параметр является а) простой переменной, в) арифметическим выражением, с) именующим выражением.

Для массива, вызываемого по значению, необходимо создать копию в секции, соответствующей заголовку процедуры, а аргумент заменить на соответствующий указатель массива. Функцию отображения можно при этом не менять.

Реализовывать это будем операциями.

С Ч И Т А Т Ь a,
П А Р А М Е Т Р r,
С Ч И Т А Т Ь П О А Д Р Е С У В М А Г А З И Н Е
К О П И Я М А С С И В А ,
З А П И С А Т Ь a.

Операция К О П И Я М А С С И В А выполняет следующие действия : Пусть П $[$НСП - I$]$ = (a_0, a_H, a_ϕ), т.е. это указатель некоторого массива. По a_ϕ узнаем размерность этого массива, диапазоны изменения индексов $d_1, d_2, \ldots, d_s$ а, следовательно, и длину массива $\lambda = d_1 \times d_2 \times \ldots \times d_s$. Пересылаем λ значений из ячеек с адресами $a_0, a_0+1, \ldots, a_0+\lambda-1$ в ячейки с адресами НСП-I, НСП, НСП + I,... НСП - 2 + λ. Это будет копия массива. Образуем новый указатель массива

$$(b_0, b_H, b_\phi),$$

где $b_0 =$ НСП - I ,

$$b_H = b_0 - a_0 + a_H$$

$$b_\phi = a_\phi$$

меняем, наконец, НСП :

НСП : = НСП + λ

засылаем новый указатель массива в ячейку П $[$НСП - I$]$.

ОГЛАВЛЕНИЕ

Глава I. стр.

§ I. Порождающие грамматики 3

§ 2. Языки и грамматики непосредственных составляющих 7

§ 3. Контактно-свободные языки и грамматики 9

§ 4. Операции над *KC*-языками I3

§ 5. Неоднозначные грамматики и неоднозначные языки I6

§ 6. Магазинные автоматы 2I

Глава 2. Синтаксический анализ 35

§ I. Проблема принадлежности слова языку 35

§ 2. Анализ сверху-вниз и анализ снизу-вверх 38

§ 3. Грамматики с отношением предшествования 42

§ 4. Функции предшествования 52

§ 5. Детерминированные автоматы с магазинной памятью 59

§ 6. *LR(K)* - грамматики 7I

Глава 3. Методы трансляции 84

§ I. Метод трансляции, использующий магазинный преобразователь 84

§ 2. Синтаксически управляемая трансляция (СУТ).... 96

Глава 4. Реализация III

§ I. Магазинный принцип использования памяти II2

§ 2. Адресация величин II7

§ 3. Структура машины II8

§ 4. Адресация элементов массива I2I

§ 5. Вход в блок I26

стр.

§ 6. Обработка описаний I29

§ 7. Выход из блока посредством оператора *goto* I30

§ 8. Выход из блока через *end*..... I32

§ 9. Выход из процедуры через *end*..... I32

§ I0. Значение, вычисляемое процедурой-функцией. I33

§ II. Обращение к процедуре I33

§ I2. Обращение к параметру в теле процедуры ... I37

Ротапринт МФТИ *158481*

Объём 6 п.л.

Тираж 500 экз.

Цена 60 коп.

Заказ № 368