ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ

Сборник трудов
1965 г. Института математики СО АН СССР Выпуск I8

АВТОМАТИЗАЦИЯ СИНТЕЗА ДИСКРЕТНЫХ АВТОМАТОВ НА ОСНОВЕ АЛГОРИТМИЧЕСКОГО ЯЗЫКА ЛЯПАС

А.Д. Закревский

I. Введение

I. Задача синтеза, являющаяся основной задачей теории дискретных автоматов, или релейных устройств,в каждом конкретном случае может быть разложена на некоторую совокупность других,более простых задач. Эти задачи, которые могут оказаться все же довольно сложными, обычно входят в класс задач, называемых комбинаторными, неарифметическими или логическими.Мы будем пользоваться последним из данных названий. К логическим задачам, решаемым при синтезе дискретных автоматов, относятся, в частности, задачи эквивалентных преобразований булевых выражений с целью оптимизации представляемых этими выражениями структур автоматов, решение систем логических уравнений, некоторые задачи теории графов и теории кодирования,операции со строчками символов (поиск, подстановка, расширение и т.д.) и т.п. Как правило, эти задачи характеризуются необходимостью просмотра многих вариантов, перебор которых, в общем случае, неизбежен и может быть лишь сокращен в какой-то степени путем совершенствования алгоритмов решения.

В связи с этим возникает проблема автоматизации синтеза, решение которой позволит ликвидировать существующий в настоя-

щее время заметный разрыв между теорией, в рамках которой разрабатываются алгоритмы синтеза, и инженерной практикой конструирования реальных автоматов, в которой известные методы синтеза используются далеко не в полной степени, поскольку овладение многими из них представляет нелегкую для инженера задачу.

2. Учитывая широту классов логических задач, разумно ориентироваться на решение поставленной проблемы на базе универсальных цифровых вычислительных машин (УЦВМ) с программным управлением. Представляется перспективной задача создания универсальных логических машин, то есть машин с программным управлением, предназначаемых специально для решения логических, а не численных задач. Как показано, например, в работах[1]-[2], такие машины, обладая тем же уровнем структурной сложности, что и УЦВМ, при решении логических задач могут оказаться эффективнее порядка на два. Однако с технической точки зрения более реальной является ориентация на многочисленный парк существующих УЦВМ, хотя они и предназначаются, в первую очередь, для проведения численных расчетов.

Известен ряд успешных опытов по использованию УЦВМ для решения задач синтеза, описанных, например, в[3]-[4]. Вместе с тем эти опыты говорят о малой производительности натурального программирования логических задач, то-есть составления вручную программ непосредственно в языке машины. Значительно лучшие результаты можно получить, автоматизировав трудоемкие этапы программирования интересующих нас задач.

3. Здесь уместно отметить появившуюся в последнее время тенденцию к разработке наряду с универсальными программирующими языками типа FORTRAN и ALGOL -60 языков, ориентированных на решение задач тех или иных классов, например, инженерных, экономических, лингвистических и т.д.[5].Такие специализированные языки оказываются значительно более эффективными и удобными в области их применения. Задачи логического характера, к которым относятся и задачи синтеза, имеют свою специфику и также требуют своего языка.

К настоящему времени известно о ряде ведущихся в соответствующем направлении работ, как правило, по пути расширения уже построенных языков[6]-[8]и др. Независимо от них исследования по автоматизации программирования логических задач были начаты в конце 1962 года в Сибирском физико-техническом институте при Томском государственном университете. Отдельные этапы этих исследований были отражены в[9]-[13]. В настоящей статье дается лишь беглое описание полученных результатов и обсуждаются перспективы дальнейшего развития появившегося направления.

4. В основу системы автоматизации программирования логических задач был положен специально для этого разработанный язык ЛЯПАС (логический язык представления алгоритмов синтеза). При его разработке наряду с обычными требованиями к языкам - компактность представления алгоритмов, близость к языку публикаций - учитывались и такие требования, как максимальное использование некоторых особенностей современных УЦВМ, важных при решении логических задач, и потенциальная простота трансляции на машинные языки. Иными словами, язык и соответствующая ему программирующая система (ПС) разрабатывались параллельно и оптимизировались в совокупности. Рассматривая эту совокупность в дальнейшем, будем называть её **системой ЛЯПАС**.

Система ЛЯПАС обладает рядом особенностей, позволивших по-новому подойти к решению некоторых старых задач теории программирования.

Язык ЛЯПАС является многоступенчатым, и ПС имеет характер цепи, звеньями которой служат трансляторы между соседними уровнями языка. Такая структура языка придает ему большую гибкость и упрощает задачу создания ПС и её дальнейшего совершенствования. Различие между уровнями носит качественный характер. К настоящему времени разработаны два из них.

Первый уровень ЛЯПАСа является наиболее близким к языкам современных УЦВМ и обеспечивает возможности наилучшего согласования с ними. На этом уровне, предназначенном для представления мелкой структуры алгоритмов, можно относительно легко составлять непосредственно программы, объем которых эквивалентен двум - трем сотням машинных команд. На втором уровне представляются программы со сложной иерархической структурой,опирающиеся на расширяющуюся систему операторов второго уровня и библиотеку соответствующих подпрограмм. Эквивалентный объем этих программ может быть значительно выше.

Лишь один из блоков ПС, а именно транслятор-I, то есть транслятор с первого уровня ЛЯПАСа, должен быть представлен в машинном языке. В остальном вся ПС, включая и библиотеку программ, выражается в ЛЯПАСе и поэтому является универсальной, то-есть не привязанной к конкретным машинам.

Стоит также отметить, что в системе ЛЯПАС предусмотрена возможность отладки программ с выдачей соответствующей информации в языке ЛЯПАС.

2. Первый уровень ЛЯПАСа

I. Основными объектами операций в ЛЯПАСе служат множества, задаваемые в некоторых конечных пространствах и представляемые булевыми векторами. За размерность ρ этих векторов по ряду соображений удобно выбирать степень двойки, откуда следует, что для большинства современных универсальных машин целесообразно положить $\rho = 32$, назвав эту размерность с т а н д а р т н о й. Векторы стандартной размерности представляют значения переменных, индексов и констант, цепочки таких векторов – комплексы, выбор нумерованных элементов из которых осуществляется с помощью индексов, значения которых в данном случае интерпретируются как натуральные числа.

В отличие от $ALGOL_a$-60 [I4] в рассматриваемом языке символика операндов фиксируется. П е р е м е н н ы е обозначаются через

$\underline{a}, \underline{b}, \underline{c}, \underline{d}, \underline{e}, \underline{f}, \underline{g}, \underline{h}, \underline{i}, \underline{j}, \underline{k}, \underline{l}, \underline{m}, \underline{n}, \underline{p}, \underline{q}$,

$\underline{r}, \underline{s}, \underline{t}, \underline{u}, \underline{v}, \underline{w}, \underline{x}, \underline{y}, \underline{z}, \underline{ж}, \underline{ц}, \underline{ф}, \underline{ш}, \underline{э}, \underline{ю}, \underline{я}$,

и н д е к с ы – через

$a, b, c, d, e, f, g, h, i, j, k, l, m, n, p, q$,

$r, s, t, u, v, w, x, y, z, ж, ц, ф, ш, э, ю, я$,

к о м п л е к с ы – через

$\underline{A}, \underline{B}, \underline{C}, \underline{D}, \underline{E}, \underline{F}, \underline{G}, \underline{H}, \underline{I}, \underline{J}, \underline{K}, \underline{L}, \underline{M}, \underline{N}, \underline{P}, \underline{Q}$,

$\underline{R}, \underline{S}, \underline{T}, \underline{U}, \underline{V}, \underline{W}, \underline{X}, \underline{Y}, \underline{Z}, \underline{Ж}, \underline{Ц}, \underline{Ф}, \underline{Ш}, \underline{Э}, \underline{Ю}, \underline{Я}$.

М о щ н о с т и $\mathfrak{G}_a, \mathfrak{G}_b, \ldots, \mathfrak{G}_я$ комплексов $\underline{A}, \underline{B}, \ldots, \underline{Я}$, соответственно, являются, вообще говоря, переменными. Комплексы задаются перечислением своих элементов (например, $\underline{A} \equiv \equiv \{\underline{a}_0, \underline{a}_1, \ldots \underline{a}_{\mathfrak{G}_a - 1}\}$), причем через $\underline{a}_b$ обозначается $[b]$-ый элемент комплекса $\underline{A}$, где $[b]$ – натуральное число, двоичный код которого совпадает с кодом значения индекса b (подразумевается позиционный двоичный код, используемый в современных машинах). Заметим, что в принятой системе обозначений запись $\underline{a}_b$ эквивалентна записи $\underline{ab}$.

Константы подразделяются на стандартные и натуральные.

Набор с т а н д а р т н ы х к о н с т а н т состоит из следующих 32-разрядных кодов, каждый из которых, как и любой другой п р о с т о й с и м в о л ЛЯПАСа, имеет для своего обозначения соответствующий 9-разрядный код. Кроме того, этот набор оформляется в виде четырех комплексов, что позволяет оперировать со стандартными константами как с элементами комплексов.

c_0	– I000 0000 0000 0000 0000 0000 0000 0000
c_1	– 0I00 0000 0000 0000 0000 0000 0000 0000
c_2	– 00I0 0000 0000 0000 0000 0000 0000 0000
	....
c_{31}	– 0000 0000 0000 0000 0000 0000 0000 000I
d_0	– I0I0 I0I0 I0I0 I0I0 I0I0 I0I0 I0I0 I0I0
d_1	– II00 II00 II00 II00 II00 II00 II00 II00
d_2	– IIII 0000 IIII 0000 IIII 0000 IIII 0000
d_3	– IIII IIII 0000 0000 IIII IIII 0000 0000
d_4	– IIII IIII IIII IIII 0000 0000 0000 0000
e_0	– 0I0I 0I0I 0I0I 0I0I 0I0I 0I0I 0I0I 0I0I
e_1	– 00II 00II 00II 00II 00II 00II 00II 00II
e_2	– 0000 IIII 0000 IIII 0000 IIII 0000 IIII
e_3	– 0000 0000 IIII IIII 0000 0000 IIII IIII
e_4	– 0000 0000 0000 0000 IIII IIII IIII IIII
f_0	– 0000 0000 0000 0000 0000 000I I000 0000
f_1	– 0000 0000 0000 0000 0000 000I II00 0000
f_2	– 0000 0000 0000 0000 0000 000I III0 0000
f_3	– 0000 0000 0000 0000 0000 000I IIII 0000
f_4	– 0000 0000 0000 0000 0000 000I IIII I000
f_5	– 0000 0000 0000 0000 0000 000I IIII II00
f_6	– 0000 0000 0000 0000 0000 000I IIII III0
f_7	– 0000 0000 0000 0000 0000 000I IIII IIII
f_{10}	– IIII IIII IIII IIII IIII IIII IIII IIII
f_{11}	– IIII IIII IIII IIII IIII IIII IIII IIII

Особое значение имеет последняя из констант f_{11} . Она представляется в любой машине единицами во всех разрядах ячейки памяти, независимо от числа этих разрядов, в то время как все другие константы могут иметь единицы только в 32 разрядах, выделенных определенным образом. Предназначается константа f_{11} главным образом для использования в системе автоматического программирования.

Введем также I28 **натуральных констант**, задаваемых непосредственно с помощью кодов программы. За символы этих констант примем представления соответствующих им натуральных чисел в восьмеричной системе счисления.

Таким образом,

0 - 0000 0000 0000 0000 0000 0000 0000 0000
I - 0000 0000 0000 0000 0000 0000 0000 000I
2 - 0000 0000 0000 0000 0000 0000 0000 00I0
....
7 - 0000 0000 0000 0000 0000 0000 0000 0III
I0 - 0000 0000 0000 0000 0000 0000 0000 I000
....
I76 - 0000 0000 0000 0000 0000 0000 0III IIIO
I77 - 0000 0000 0000 0000 0000 0000 0III IIII

2. Условимся, что переменные и индексы, представляемые буквами $ж, ѣ, ф, ш, э, ю, я$, будут иметь специальное назначение (в основном - в системе автоматического программирования), поэтому употреблять их при программировании наравне с другими переменными и индексами не разрешается. Например, из введенных переменных особо выделяется $\underline{Я}$ - она служит **датчиком случайных значений**, в связи с чем нельзя присваивать ей какое-либо значение программным путем. Полагается, что при обращении к переменной $\underline{Я}$ её значение представляется случайным набором нулей и единиц, в котором вероятности 0 и I одинаковы и в котором отсутствует межразрядная корреляция.

Как это было уже показано, при перечислении стандартные константы разбиваются на четыре комплекса C, D, E и F, относящиеся к **специальным комплексам**

$$A, B, C, D, E, F, G, K, I, J, K, L, M, N, P, Q .$$

Специальные комплексы обладают фиксированным значением, которое для некоторых из них будет раскрыто в этом параграфе, для других - при описании системы автоматического программирования. Мощность специальных комплексов фиксирована.

Определим **оперативный комплекс** K как такой комплекс, значение i -го элемента которого представляется в i -ой ячейке оперативного запоминающего устройства машины. При таком определении задача размещения информации и ЗУ машины становится задачей установления определенного соответствия между оперативным комплексом и другими комплексами и отдельными переменными, представляющими перерабатываемую информацию.

Комплекс A примем за **комплекс начал** всех комплексов $\underline{A}, \underline{B}, \ldots, \underline{Я}$ в оперативном комплексе, т.е. положим, что элемент a_0 будет представлять номер того элемента оперативного комплекса, который отождествляется с элементом $\underline{a_0}$, элемент a_1 ставится в аналогичное соответствие с элементом $\underline{b_0}$ и т.д.

Комплекс B назовем **комплексом мощностей** комплексов $\underline{A}, \underline{B}, \ldots, \underline{Я}$: значением b_0 будет служить c_a - мощность множества $\underline{A}$, $[b_1]=c_b$, $[b_2]=c_c$ и т.д.

Через G обозначим **комплекс простых переменных**, положив $g_0=\underline{a}, g_1=\underline{b}, g_2=\underline{c}$ и т.д. Аналогично определим **комплекс индексов** H и **комплексы дополнительных переменных** T и **индексов** J .

3. Программа первого уровня ЛЯПАСа выражается последовательностью символов, однозначно разлагающейся на символы и составные символы, представляющие **операции и метки**. В основу набора операций положена гипотеза о существенной связности отображаемых вычислительных процессов, оправдывающая специализацию одной из переменных на фиксировании результата очередной операции. Эта переменная, называемая **собственной переменной** $\underline{c}$, играет, кроме того, роль левого операнда во многих операциях. Набор операций выбран на основе учета как удобств оперирования с этим набором при программировании, так и простоты выражения выбранных операций через элементарные операции типичной УЦВМ. В этот набор включены, например, покомпонентные логические операции, непосредственно реализуемые практически в любой УЦВМ, операции последовательного поиска слева единиц в значении операнда, операция инверсии порядка следования компонент, операция подсчета числа единичных компонент и т.п. Включены в набор и четыре арифметических операции над натуральными числами, определяемых по модулю 2^{32}.

Для удобства последующего изложения классифицируем операнды, положив

$\pi_1 \in \{\underline{а}, \underline{б}, \ldots, \underline{я}\}$

$\pi_2 \in \{а, б, \ldots, я\}$

$\pi_3 \in \{0, 1, \ldots, 177\}$

$\pi_4 \in \{c_0, c_1, \ldots, c_{37}, d_0, d_1, \ldots, d_4, e_0, e_1, \ldots, e_4, f_0, f_1, \ldots, f_n\}$

$\pi_5 \in \{\underline{А}, \underline{Б}, \ldots, \underline{Я}, А, Б, \ldots, Г\}$

$\pi_6 \in \{\underline{А}, \underline{Б}, \ldots, \underline{Я}, А, Б, Г, Ж, \ldots, Г\}$

$\pi_7 \in \{\underline{А}, \underline{Б}, \ldots, \underline{Я}\}$

Будем считать, что если $\eta \in \{\eta_1, \eta_2, \ldots, \eta_m\}$ и $\zeta \in \{\zeta_1, \zeta_2, \ldots, \zeta_n\}$, то $\eta\zeta \in \{\eta_1\zeta_1, \eta_1\zeta_2, \ldots, \eta_1\zeta_n, \eta_2\zeta_1, \eta_2\zeta_2, \ldots, \eta_2\zeta_n, \ldots, \eta_m\zeta_1, \eta_m\zeta_2, \ldots, \eta_m\zeta_n\}$.

Положим далее, что

$\pi_8 \in \{\pi_1, \pi_2\}$

$\pi_9 \in \{\pi_2, \pi_3\}$

$\pi_{10} \in \{\pi_1, \pi_2, \pi_3, \pi_4\}$

$\pi_{11} \in \{\pi_1, \pi_2, \pi_6 \pi_5\}$

$\pi \in \{\pi_{10}, \pi_5 \pi_9\}$

Операторы подразделяются на вычислительные операторы, операторы присвоения значения, операторы переходов и операторы обмена с внешней средой. В программе они (то есть их символы) входят в составные символы, представляющие операции и содержащие, кроме них, символы подвергающихся действию операндов допустимых типов.

4. Обозначим через π^i i-ую двоичную компоненту вектора π, положив, что в нижеследующих выражениях i может принимать любое значение из $\{0, 1, \ldots, \rho-1\}$.

Специфику первого уровня ЯПАСа определяют следующие вычислительные операторы.

$\vee$ - оператор дизъюнкции

Этот оператор порождает операцию $\vee\pi$, результат которой, фиксируемый очередным значением τ собственной переменной, представляет покомпонентную дизъюнкцию предшествующего значения $\underline{\tau}$ собственной переменной и правого операнда π. Формальное выражение этого определения имеет вид:

$$\vee\pi \sim \tau^i = \underline{\tau}^i \vee \pi^i.$$

Например, если $\underline{\tau}$ = 01000110 и π = 10001100, то при реализации операции $\vee\pi$ фиксируется значение τ = 11001110. Короче, 01000110 $\vee$ 10001100 = 11001110 (с целью упрощения иллюстрирующих примеров мы положим в них ρ = 8).

Аналогично определяются

$\wedge$ - оператор конъюнкции

и

$\oplus$ - оператор дизъюнкции с исключением.

Примеры: 01000110 $\wedge$ 10001100 = 00000100,
01000110 $\oplus$ 10001100 = 11001010.

$+_\rho$ - оператор сложения по модулю 2^ρ

$$+\pi \sim [\tau] = [\underline{\tau}] + [\pi] \bmod 2^\rho$$

Пример: 01000110 + 10001100 = 11010010.

Аналогично определяются

$-_\rho$ - оператор вычитания по модулю 2^ρ

и

X - оператор перемножения по модулю 2^ρ

Примеры: 01000110 - 10001100 = 10111010,
01000110 x 10001100 = 01001000.

$\overline{X}$ - оператор перемножения с округлением

$$\overline{X}\pi \sim [\tau] = ([\underline{\tau}] \cdot [\pi] - [\underline{\tau} \times \pi]) : 2^\rho.$$

Пример: 01000110 $\overline{\text{X}}$ 10001100 = 00100110.

: - оператор деления

Выражение: π означает, что новым значением τ собственной переменной должен служить остаток от деления $[\underline{\tau}]$ на $[\pi]$. В то же время при реализации операции деления целая часть частного всегда представляется значением индекса $я$.

Пример: 00001101 : 00000101 = 00000011, а $я$ получает значение 00000010.

Заметим, что в операции деления недопустимо значение π = 00...0, что требуется учитывать при программировании.

$\neg$ - оператор инверсии покомпонентной

$$\neg \sim \tau^i = \overline{\underline{\tau}^i}$$

Пример: 01000110 ¬ = 10111001.

$\mathbb{I}$ - оператор инверсии порядка

$$\mathbb{I} \sim \tau^i = \underline{\tau}^{p-i-1}$$

Пример: 01000110 $\mathbb{I}$ = 01100010.

$\vdash$ - оператор нахождения номера левой единицы

$$\vdash \sim [\tau] = i_{min} \quad \text{при} \quad \underline{\tau}^i = 1.$$

Пример: 00010110 $\vdash$ = 00000011.

При использовании этого оператора следует учитывать, что 00000000 $\vdash$ = 10000000 $\vdash$ = 00000000.

$\vdash$ - оператор нормализации

Действие этого оператора выражается в сдвиге операнда влево до появления единицы в крайнем левом разряде. Освобождающиеся разряды заполняются нулями.

Пример: 00010110 $\vdash$ = 10110000.

Действие следующих двух операторов аналогично, за исключением того, что величина сдвига задается значением $[\tau]$.

< - оператор сдвига влево

> - оператор сдвига вправо

Примеры: 01101010 < 00000100 = 10100000,

01101010 > 00000101 = 00000011.

$\frown$ - оператор циклического сдвига

$$\frown \tau \sim \tau^i = \underline{\tau}^{i+[\tau] \bmod p}.$$

Пример: 01101010 $\frown$ 00100100 = 10100110.

Следующий оператор в какой-то степени выходит за рамки наших построений и вводится для того, чтобы сделать возможным более полное использование памяти машины, а именно, полное использование запоминающих ячеек в тех случаях, когда они содержат более 32 двоичных разрядов, а также полное использование пропускных способностей входного и выходного каналов связи машины.

$\rightleftharpoons$ - оператор неограниченного сдвига

Выражение $\rightleftharpoons \tau$ может быть интерпретировано следующим образом: все содержимое регистра, в котором хранится значение переменной τ, подвергается сдвигу на $q = [\tau] \bmod 2^6 - 2^5$ разрядов, причем положительному значению q соответствует

сдвиг влево, отрицательному - вправо, а освобождающиеся при сдвиге разряды заполняются нулями.

Эта операция предназначается преимущественно для использования в системе автоматического программирования и для широкого применения не рекомендуется.

∇ - оператор взвешивания

При реализации операции $\nabla [\tau]$ принимает значение, равное числу единиц в коде предшествующего значения $\underline{\tau}$.

Пример: 01101010 ∇ = 00000100.

$\curlyvee$ - оператор отражения

$$\curlyvee \tau \sim \tau^i = \begin{cases} \underline{\tau}^{i+2^n}, & \text{если } (i)^n = 0; \\ \underline{\tau}^{i-2^n}, & \text{если } (i)^n = 1; \end{cases}$$

и если $n = [\tau] \in \{0,1,2,3,4\}$, а $(i)^n$ - n-ый справа разряд двоичного позиционного кода числа i (если начинать счет с нуля). Если же $[\tau] > 4$, то при реализации операции $\curlyvee \tau$ переменная τ должна принять значение 00...0.

Если рассматривать 32-разрядный код - значение $\underline{\tau}$ собственной переменной - как линейную развертку пятимерного куба, то действие оператора отражения можно представить как зеркальный обмен четырехмерными гранями, соответствующими различным значениям одной из пяти двоичных переменных, задаваемой операндом τ.

Примеры: 01001101 $\curlyvee$ 00000000 = 10001110,
01001101 $\curlyvee$ 00000001 = 00010111, 01001101 $\curlyvee$ 00000010 = 11010100.

5. Следующая группа операторов объединяется под общим названием операторов присвоения значений.

$\Rightarrow$ - оператор присвоения

Операция $\Rightarrow \tau_n$ заключается в присвоении операнду τ_n значения $\underline{\tau}$. При этом значение собственной переменной сохраняется: $\tau = \underline{\tau}$. Заметим, что выражение τ также представляет операцию, а именно - операцию присвоения значения операнда τ собственной переменной: $\tau = \tau$.

Обобщим оператор $\Rightarrow$ на случай обмена информацией между группами операндов типа τ_{10} и комплексами типа τ_7, положив, что выражение

$$(\xi_1, \xi_2 \ldots \xi_k) \Rightarrow \tau_7,$$

где $\xi_i \in \{\pi_{10}\}$, реализуется путем расширения комплекса π_7 за счет добавления k элементов, принимающих значения операндов ξ_1 , $\xi_2, \ldots, \xi_k$, с сохранением порядка их следования.

Выражение

$$\pi_7 \Rightarrow (\xi_1 \xi_2 \ldots \xi_k),$$

где $\xi_i \in \{\pi_8\}$, имеет обратный смысл, представляя "распаковку" комплекса π_7 с конца, производимую в таком порядке, что после реализации, например, выражения

$$(\underline{a}\ \underline{b}\ m\ \rho) \Rightarrow \mathcal{R}\ \mathcal{R} \Rightarrow (\underline{a}\ \underline{b}\ m\ \rho)$$

операнды $\underline{a}, \underline{b}, m, \rho$ принимают прежние значения.

$\leftrightarrow$ - оператор обмена

Операция $\xi_i \leftrightarrow \xi_j$, где $\xi_i, \xi_j \in \{\pi_H\}$, реализуется путем обмена значений между операндами ξ_i и ξ_j . При этом ε принимает новое значение операнда ξ_j .

$\Leftarrow$ - оператор натурального присвоения

Выражение $\pi_H \Leftarrow \psi_1 \psi_2 \psi_3 \psi_4$, где ψ_j - произвольные символы языка (векторы-константы с девятью двоичными компонентами) реализуется путем присвоения операнду π_H значения, определяемого векторами ψ_j следующим образом:

$$\pi_{11}^i = \begin{cases} \psi_1^{i+4} & \text{при } 0 \leqslant i < 5 \\ \psi_2^{i-5} & \text{при } 5 \leqslant i < 14 \\ \psi_3^{i-14} & \text{при } 14 \leqslant i < 23 \\ \psi_4^{i-23} & \text{при } 23 \leqslant i < 32 \end{cases}$$

При этом τ принимает новое значение операнда π_H .
Договоримся подчеркивать такие символы ЛЯПАСа, которые совпадают со своими кодами (например, коду 065 ставится в соответствие символ $\underline{065}$). Тогда, например, для присвоения переменной $\underline{a}$ значения

IOII OOII OOII OIII IOOI OOII OIOI OIOI

достаточно реализовать операцию

$$\underline{a} \Leftarrow \underline{026}\ \underline{315}\ \underline{711}\ \underline{525}.$$

Δ - оператор положительного элементарного приращения

$$\Delta\, \pi_H \sim \pi_{11} + 1 \Rightarrow \pi_H .$$

Пример: если π_H = IOOIIOII, то результатом воздействия оператора Δ на операнд π_H явится принятие операндом π_H значения IOOIIIOO.

$\overline{\Delta}$ - оператор отрицательного элементарного приращения

$$\overline{\Delta}\, \pi_H \sim \pi_H - 1 \Rightarrow \pi_H .$$

Пример: если π_H = IOOIIOII, то результатом воздействия оператора $\overline{\Delta}$ на операнд π_H явится принятие операндом π_H значения IOOIIOIO.

O - оператор присвоения нулевого значения

Результатом воздействия оператора O на операнд π_H всегда является принятие операндом π_H значения 00000000.

$\overline{O}$ - оператор присвоения максимального значения

$$\overline{O}\, \pi_H \sim O \urcorner \Rightarrow \pi_H .$$

Результатом воздействия оператора $\overline{O}$ на операнд π_H всегда является принятие операндом π_H значения IIIIIIII.

6. Последовательность символов языка ЛЯПАС , называемая $\mathcal{A}$ -программой (или просто программой), отражает некоторый вычислительный процесс, в котором последовательно реализуются входящие в программу операторы.

Программа состоит из предложений, отделенных друг от друга символами $\S$, сопровождаемыми номером последующего предложения, который представляется соответствующей натуральной константой. Каждое предложение может содержать произвольное число символов и должно включать лишь один символ $\S$ входящий в предложение в качестве левого символа. При представлении программы на бумаге условимся для удобства каждое новое предложение начинать с новой строки.

Стандартный порядок реализации операторов во времени, при котором вслед за очередным оператором реализуется оператор, символ которого в $\mathcal{A}$ -программе следует за символом очередного оператора, может нарушаться только при встрече следующих операторов управления процессом реализации программы, называемых операторами переходов.

$\rightarrow$ - оператор безусловного перехода

Пара символов $\rightarrow \pi_3$ означает, что непосредственно вслед за реализацией предшествующей части программы начнется реализация предложения номер $[\pi_3]$.

$\circ\!\!-$ – оператор условного перехода по нулю

Этот оператор становится эквивалентным по действию оператору $\rightarrow$ при следующих условиях:

а) если оператору $\circ\!\!-$ непосредственно предшествует оператор + или –, при реализации которых оказывается, что $[\xi]\pm[\pi] \neq [\xi \pm \pi]$, т.е. при $[\xi]+[\pi] \geqslant 2^{\rho}$ (при операторе +), или $[\xi] < [\pi]$ (при операторе –), где ξ и π – соответственно левый и правый операнды, подвергающиеся действию оператора + или – ;

б) если оператору $\circ\!\!-$ не предшествует оператор + или – и если последним значением переменной τ является 00000000.

В остальных случаях стандартный порядок реализации программы не нарушается.

$\vdash$ – оператор условного перехода по единице

Действие этого оператора противоположно действию оператора $\circ\!\!-$.

Следующие два оператора вводятся, главным образом, для упрощения работы с подпрограммами. Введем понятие глубины реализации программы, положив, что началу реализации программы соответствует нулевая глубина и что глубина может меняться лишь при встрече операторов $\#\!\!-$ и $!$.

$\#\!\!-$ – оператор ухода

Выражение $\#\!\!- \pi_3 \alpha$, где α – некоторое произвольное выражение в языке ЛЯПАС, означает, что глубина реализации должна быть увеличена на единицу, "координаты" выражения α должны быть поставлены в соответствие новому значению глубины и запомнены и должен быть совершен переход к реализации предложения номер $[\pi_3]$.

$!$ – оператор возвращения

При встрече оператора $!$ производится переход к реализации того выражения, "координаты" которого запомнены на текущей глубине, вслед за чем глубина реализации уменьшается на единицу. Реализация любой программы начинается с нулевой глубины.

$\dagger$ – оператор перехода к машинному языку

Пара символов $\dagger \pi_8$ говорит о переходе к реализации некоторой программы на машинном языке, первая из команд которой представляется $[\pi_8]$-ым элементом оперативного комплекса. Чтобы обеспечить возвращение к реализации $\mathcal{A}$-программы, начиная с символа, следующего за символами $\dagger \pi_8$, упомянутую программу на машинном языке следует кончать командой безусловной передачи управления в ячейку ОЗУ, отведенную под хранение значений переменной $\underline{10}$.

Этот оператор может оказаться весьма полезным в тех ситуациях, когда целесообразно выражение какого-либо вычислительного процесса непосредственно в языке машины. Его использование требует ознакомления с некоторыми характеристиками системы автоматического программирования (размещение информации в ОЗУ и пр.). Условимся, что при реализации перехода по любому из рассмотренных операторов значение собственной переменной τ не меняется. Следующие два оператора обладают комбинированным характером.

$\mathcal{X}$ – оператор перебора единиц

$$\pi_{11}\,\mathcal{X}\,\pi_3\pi_2 \sim \pi_{11}\circ\!\!-\pi_3\pi_{11}\vdash \Rightarrow \pi_2 \subset \pi_2 \oplus \pi_{11} \rightarrow \pi_{11}.$$

При реализации этой операции в значении операнда π_{11} удаляется левая единица, а её координата сообщается индексу π_2 и собственной переменной τ. Если же исходным значением операнда π_{11} служит 00...0, начинается реализация предложения номер $[\pi_3]$, а τ, π_{11} и π_2 принимают значение 00...0.

Пример: если исходное значение операнда π_{11} = 01011010 (значение индекса π_2 может быть произвольным), то после реализации операции $\pi_{11}\,\mathcal{X}\,\pi_3\pi_2$ операнд π_{11} примет значение 00011010, а индекс π_2 – значение 00000001.

$\mathcal{Z}$ – оператор случайного перебора единиц

Этот оператор отличается от предыдущего лишь тем, что очередной выбор единицы в коде операнда π_{11} происходит случайно, с равными вероятностями выбора каждой единицы.

7. Операторы обмена информацией с внешней средой

Следующие операторы играют вспомогательную роль, обеспечивая, главным образом, обмен информацией между запоминающим устройством реализующей алгоритм машины и внешней средой.

$*$ - о п е р а т о р п е ч а т и э л е м е н т а

Реализация оператора $*$ заключается в выдаче текущего значения собственной переменной τ на печать. Учитывая особенности современных вычислительных машин, потребуем, чтобы значение τ выдавалось в восьмеричном коде.

$\overset{*}{*}$ - о п е р а т о р п е р ф о р а ц и и э л е м е н т а

Этот оператор представляет выдачу текущего значения τ на перфорацию. Он обладает известными удобствами, например, в том случае, когда выдаваемая при реализации алгоритма информация предназначена для последующего ввода в машину.

В том и только в том случае, когда оператор $*$ или $\overset{*}{*}$ следует непосредственно за символом комплекса (например, в случае $\underline{A*}$) на печать или перфорацию выдаются последовательно значения всех элементов комплекса.

$\vdash$ - о п е р а т о р в в о д а

Подлежащая обработке на машине информация оформляется в виде последовательности констант, которая заранее разбивается на порции. В начале каждой порции указывается её размер и номер некоторого элемента оперативного комплекса машины. При реализации оператора $\vdash$ производится ввод очередной порции - входящие в неё константы в порядке их следования становятся значениями элементов той части оперативного комплекса, которая начинается с отмеченного элемента и совпадает с вводимой порцией.

Кроме оперативного комплекса, соответствующего оперативной памяти машины, введем в рассмотрение внешний комплекс, соответствующий дополнительной, внешней памяти машины. Обмен информацией между этими двумя комплексами может производиться с помощью следующих двух операторов.

$/$ - о п е р а т о р п о с ы л к и в о в н е ш н и й к о м п л е к с

Выражение $/\xi_1\,\xi_2\,\xi_3$, где $\xi_1, \xi_2, \xi_3 \in \{\tau_8\}$, означает, что $[\xi_2]$ рядом расположенных элементов оперативного комплекса, начинающихся с элемента номер $[\xi_1]$, передают свои значения $[\xi_2]$ рядом расположенным элементам внешнего комплекса, начинающимся с элемента номер $[\xi_3]$.

$\backslash$ - о п е р а т о р п о с ы л к и и з в н е ш н е г о к о м п л е к с а.

Этот оператор представляет передачу информации в обратном направлении: $[\xi_2]$ элементов внешнего комплекса, начиная

с элемента номер $[\xi_3]$, передают свои значения соответствующей серии элементов оперативного комплекса, начинающейся с элемента номер $[\xi_1]$.

8. Назовем α - ц е п о ч к о й произвольную конечную последовательность составных символов α :

$$\alpha \in \{\tau, \rho_1,\rho_2\tau, \rho_3\tau_3, \rho_4\tau_H, *, \overset{*}{*}, \to\tau_H, \vdash\tau_8, \tau_5 *, \tau_5 \overset{*}{*}, \vdash,$$
$$/\tau_8\tau_8\tau_8, /\tau_8\tau_8\tau_8, \tau_7 \to (\tau_8\tau_8\ldots\tau_8), (\tau_{10}\tau_{10}\ldots\tau_{10}) \to \tau_7,$$
$$\tau_{11}\mathcal{X}\tau_3\tau_2, \tau_{11}\mathcal{E}\tau_3\tau_2, \tau_H \leftarrow\!\!\!\!\!\!\;\psi\psi\psi\psi, \dot{\tau}_H \leftrightarrow \tau_H, .^{!}, \xi\tau_3\},$$

где

$$\rho_1 \in \{\vdash, \vdash, \neg, I, \nabla\}$$
$$\rho_2 \in \{+, -, \vee, \wedge, \oplus, \times, \overline{\vee}, >, <, -, =, \vee, :\}$$
$$\rho_3 \in \{-, \circ\!\!-, \vdash, \to\}$$
$$\rho_4 \in \{O, \bar{O}, \Delta, \bar{\Delta}\}.$$

Символы $\S\tau_3$ разбивают α - цепочку на предложения, номерами которых служат значения символа τ_3 в стоящей в начале каждого предложения метке $\S\tau_3$.

Назовем п р о г р а м м о й такую α -цепочку, которая удовлетворяет следующим условиям.

Во-первых, α -цепочка должна замыкаться символом " . ", играющим двойную роль : с одной стороны, он служит признаком конца алгоритма (реализация алгоритма прекращается при встрече этого символа), с другой стороны, символом " . " всегда замыкается выражение, представляющее алгоритм.

Во-вторых, номера предложений в данной α -цепочке не должны повторяться и должны образовывать такое множество N, чтобы для всех встречающихся в α -цепочке сочетаний $\rho_3\tau_3$, $\mathcal{X}\tau_3$ и $\mathcal{E}\tau_3$ выполнялось условие $[\tau_3] \in N$.

В-третьих, накладываются определенные ограничения на ориентированный граф переходов, представленных в рассматриваемой α-цепочке. Вершины этого графа ставятся в соответствие предложениям программы, а дуги - возможным переходам по операторам ρ_3, $\mathcal{X}$ и $\mathcal{E}$ и переходам по непосредственному следованию предложений (в случае, если предшествующее предложение не оканчивается символом $\to$, $\to$ или !). Отметим дуги, соответствующие переходам по оператору $\to$, и вершины, соответствующие предложениям, оканчивающимся символом ! . В построенном таким образом графе не должно существовать контуров, содержащих отмеченные дуги. В графе, полученном из первого пу-

тем выбрасывания отмеченных дуг, не должно существовать путей, идущих от вершины, соответствующей начальному предложению (эта вершина не должна быть отмеченной) к какой-либо из отмеченных вершин.

Сформулированные условия могут рассматриваться как синтаксис первого уровня ЛЯПАСа. Конечно, не любая из удовлетворяющих ему α - цепочек представляет достаточно разумную программу. Тем не менее любая из таких α- цепочек называется программой и должна однозначно восприниматься транслятором с первого уровня ЛЯПАСа на язык конкретной машины. Поэтому можно сказать, что данный синтаксис задает область определения функций транслятора. Что же касается повышения степени разумности рассматриваемых программ, то она достигается применением более детализированного синтаксиса, на который опирается блок поиска синтаксических ошибок, входящий в состав программирующей системы и описанный в [13].

В порядке уточнения способа определения текущих значений собственной переменной τ положим, что τ сохраняет прежнее значение при операциях $*, \overset{*}{*}, \to\pi_{11}, o\!-\pi_3, \vdash\pi_3, |\pi_8$ и при входе в новое предложение, отмечаемом меткой $\S\pi_3$. При реализации операций $\pi_5 *, \pi_5 \overset{*}{*}, |, /\pi_8\pi_8\pi_8, /\pi_8\pi_8\pi_8, \pi_7 \to$ $\to(\pi_8\pi_8\ldots\pi_8), (\pi_{10}\pi_{10}\ldots\pi_{10})\to\pi_7$ принимает нулевое значение. Началу реализации программы соответствует нулевое значение τ.

9. Чтобы обеспечить возможность слежения за порядком использования оперативного комплекса при реализации программы, введем специальный символ $?$. Наличие его после сочетания $\pi_7\pi_2$, где $\pi_7 \in \{\underline{\mathcal{A}}, \underline{\mathcal{B}}, \ldots, \underline{\mathcal{Я}}\}$ и $\pi_2 \in \{a, \mathcal{б}, \ldots, я\}$, служит сигналом для проверки комплекса π_7 на "наползание" в оперативном комплексе на другие операнды и принятия мер по ликвидации такового, если оно будет иметь место. Сочетание $\mathcal{B}\pi_{12}??$, где $\pi_{12} \in \{0, 1, \ldots, 37\}$, служит сигналом для нахождения в оперативном комплексе места для того комплекса, информация о котором содержится в символах $\mathcal{B}\pi_{12}$.

Реализация указанных операций возлагается на соответствующий блок системы автоматического программирования.

Уточняя описанный выше синтаксис первого уровня ЛЯПАСа, заметим, что удовлетворяющие ему цепочки символов могут быть расширены за счет замены $\pi_7\pi_2$ на $\pi_7\pi_2?$ и $\mathcal{B}\pi_{12}$ на $\mathcal{B}\pi_{12}??$ Однако, повторная замена подобного рода, приводящая, например, к получению сочетания вида $\pi_7\pi_2??$ недопустима.

3. Программирование на первом уровне ЛЯПАСа

1. При описании постановки какой-либо задачи и алгоритма её решения разрешим использование латинского алфавита и прочей символики, употребляемой в ЛЯПАСе, но в другом значении. Чтобы согласовать это описание с соответствующей $\mathcal{A}$-программой, договоримся, в частности, пользоваться матрицами C бинарных отношений между элементами двух множеств $\mathcal{A} \equiv \equiv \{a_0, a_1, \ldots, a_{n-1}\}$ и $\mathcal{B} \equiv \{\mathcal{б}_0, \mathcal{б}_1, \ldots, \mathcal{б}_{m-1}\}$, определив их (матрицы) следующим образом: $C = \|\mathcal{A} \prec \mathcal{B}\|$ означает, что $c_{ij} = 1$, если $a_i \prec \mathcal{б}_j$, где $\prec$ - символ некоторого бинарного отношения, и $c_{ij} = 0$ в противном случае. Через $\|\{\mathcal{A}\} \prec \mathcal{B}\|$ будем обозначать однострочную матрицу бинарного отношения $\prec$ между единственным элементом $\mathcal{A}$ множества $\{\mathcal{A}\}$ и элементами множества $\mathcal{B}$.

Положим, что выражение

$$\underline{\mathcal{A}} :: \|\mathcal{A} \prec \mathcal{B}\|$$

означает, что комплекс $\underline{\mathcal{A}}$ представляет матрицу $\|\mathcal{A} \prec \mathcal{B}\|$, то есть строки этой матрицы служат значениями элементов комплекса $\underline{\mathcal{A}}$. Аналогично,

$$\underline{a} :: \|\{\mathcal{A}\} \prec \mathcal{B}\|$$

означает, что значением переменной $\underline{a}$ служит однострочная матрица $\|\{\mathcal{A}\} \prec \mathcal{B}\|$.

Рассмотрим несколько примеров программ на первом уровне ЛЯПАСа, сопровождая их краткими пояснениями.

2. Пусть $\mathcal{B} \subseteq \hat{\mathcal{A}}$, где $\hat{\mathcal{A}}$ - множество всех подмножеств некоторого множества $\mathcal{A}$. Нижней границей множества $\mathcal{B}$ называется его подмножество, состоящее из тех элементов $\mathcal{б}_i \in \mathcal{B}$, для которых нельзя найти таких $\mathcal{б}_j \in \mathcal{B}$, что $\mathcal{б}_j \subset \mathcal{б}_i$.

Предложим один из возможных вариантов программы получения нижней границы множества $\mathcal{B}$, положив, что $\underline{\mathcal{B}} :: \|\mathcal{B} \ni \mathcal{A}\|$.

§ 0 $\quad \bar{o}\, \mathcal{б}\, o\, c$

§ 1 $\quad \Delta\, \mathcal{б} \oplus \mathcal{б}_1\, o\!-\, 4\, \bar{o}\, a$

§ 2 $\quad \Delta\, a \oplus \mathcal{б}_1\, o\!-\, 3\, a \oplus \mathcal{б}\, o\!-\, 2\, \underline{\mathcal{б}}\, \neg \wedge \underline{\mathcal{б}}_a\, o\!-\, 1\!-\, 2$

§ 3 $\quad \underline{\mathcal{б}} \to \underline{c}_c\, \Delta c\!-\, 1$

§ 4 $\quad c \to \mathcal{б}_2.$

Напомним, что элемент b_1 специального комплекса $\mathcal{B}$ представляет мощность комплекса $\underline{\mathcal{B}}$, а элемент b_2, значение которого определяется в конце программы, – мощность комплекса $\underline{\mathcal{C}}$, в котором фиксируется результат работы рассматриваемой программы.

Например, в результате приложения данной программы к комплексу

$\underline{\mathcal{B}}$

IOIIOOIO
OIIOIIOI
IOIIIOII
IIIIOIOI
OIOOOOIO
OIIOOOOI
IOOOIIOO
OOOOIOOO

получается комплекс

$\underline{\mathcal{C}}$

IOIIOOIO
OIOOOOIO
OIIOOOOI
OOOOIOOO

3. Упрощением Квайна мы называем принадлежащий Квайну алгоритм определенного сокращения множества $\mathcal{B} \subseteq \widehat{\mathcal{A}}$, предложенный в качестве одного из этапов минимизации булевых функций в классе дизъюнктивных нормальных форм. Упрощение Квайна заключается в получении такого подмножества $\mathcal{D}$ множества $\mathcal{B}$, элементы которого не поглощаются ядром $\mathcal{C}$ множества $\mathcal{B}$, то есть множеством существенных элементов множества $\mathcal{B}$, или входят в это ядро. При этом элемент $b_i \in \mathcal{B}$ называется существенным элементом множества $\mathcal{B}$, если он содержит некоторый элемент a_j множества $\mathcal{A}$, не содержащийся ни в одном из других элементов множества $\mathcal{B}$. Считается также, что элемент $b_i \in \mathcal{B}$ поглощается множеством $\mathcal{B}_j \subseteq \mathcal{B}$, если $b_i \subseteq \mathcal{B}_j^*$, где $\mathcal{B}_j^*$ объединение всех элементов множества $\mathcal{B}_j$.

Приведем программу упрощения Квайна, в которой

$$\mathcal{B} :: \| \mathcal{B} \ni \mathcal{A} \|$$

а значения переменных $\underline{b}, \underline{c}, \underline{d}$ и $\underline{e}$ определяются в процессе реализации программы следующим образом:

$$\underline{b} :: \| \{\mathcal{B}^*\} \ni \mathcal{A} \|$$
$$\underline{c} :: \| \{\mathcal{B}^{**}\} \ni \mathcal{A} \|,$$

где $\mathcal{B}^{**}$ – множество таких элементов из $\mathcal{A}$, каждый из которых входит не менее чем в два элемента из $\mathcal{B}$,

$$\underline{d} :: \| \{\mathcal{B}^* \setminus \mathcal{B}^{**}\} \ni \mathcal{A} \|$$
$$\underline{e} :: \| \{C^*\} \ni \mathcal{A} \|$$

§ 0 $\bar{o}\, a\, o\, \underline{b}\, o\, \underline{c}\, o\, i$

§ I $\Delta a \oplus b_1\, o\!-\!2\, \underline{b} \wedge \underline{b}_a \vee \underline{c} \to \underline{c}\, \underline{b}_a \vee \underline{b} \to \underline{b} - 1$

§ 2 $\underline{c} \neg \wedge \underline{b} \to \underline{d}\, \bar{o}\, a\, o\, \underline{e}$

§ 3 $\Delta a \oplus b_1\, o\!-\!4\, \underline{b}_a \wedge \underline{d}\, o\!-\!3\, \underline{b}_a \vee \underline{e} \to \underline{e} - 3$

§ 4 $\underline{d} \neg \wedge \underline{e} \to \underline{f}\, \bar{o}\, a$

§ 5 $\Delta a \oplus b_1\, o\!-\!6\, \underline{f} \neg \wedge \underline{b}_a\, o\!-\!5\, \underline{b}_a \to \underline{d}_i \Delta i \to 5$

§ 6 $i \Rightarrow b_3$.

Результат реализации программы представляется комплексом $\underline{\mathcal{D}}$

$$\underline{\mathcal{D}} :: \| \mathcal{D} \ni \mathcal{A} \|$$

4. Ниже приводится программа перехода от матрицы смежности симметрического графа к матрице связей, единичные элементы которой указывают на наличие цепи между соответствующими вершинами графа. Матрица смежности [15] представляется перед реализацией программы комплексом $\underline{\mathcal{A}}$, элементы которого выражают строки матрицы. После реализации нижеприведенной программы комплекс $\underline{\mathcal{A}}$ будет представлять аналогичным образом матрицу связей.

§ 0 $\bar{o}\, i$

§ I $\Delta i \oplus b_0\, o\!-\!3\, \underline{a}_i \Rightarrow \underline{a}$

§ 2 $\underline{a}\, x1_{ij}\, \underline{a}_j \vee \underline{a}_i \Rightarrow \underline{a}_j - 2$

§ 3 .

5. В связи с задачей синтеза логических схем из мажоритарных элементов, реализующих функцию $\psi(a,b,c) = ab \vee ac \vee bc$, представляет интерес задача ψ – разложения булевых функций. Пусть $X \equiv \{x_0, x_1, \ldots, x_{n-1}\}$ и $\{x,y,z\} \subseteq X$. Назовем булеву функцию $f(X)$ ψ-разложимой по триаде (x,y,z) (такую триаду назовем простой), если существуют такие функции f_1, f_2, f_3, что

$$f(X) = \psi(f_1(X\ \{x\}), f_2(X)\ \{y\}), f_3(X\ \{z\}))$$

В работе [I6] показано, что необходимое и достаточное условие такой разложимости может быть выражено в следующей форме

$$(\varphi \oplus N_x\varphi) \vee (\varphi \oplus N_y \varphi) \vee (\varphi \oplus N_z \varphi) = 0,$$

где

$$\varphi = (f \oplus N_x f) \wedge (f \oplus N_y f) \wedge (f \oplus N_z f),$$

f - исследуемая булева функция и N_a - оператор отражения булевой функции по переменной a (например, $N_a(a\bar{c} \vee \bar{a}b) = \bar{a}\bar{c} \vee ab$).

Любая булева функция n аргументов может быть представлена одним вектором с $\rho = 2^n$ двоичными компонентами, задающими перечень значений функции для всех элементов булева пространства n переменных. Положим, что таким путем функция f будет представляться значением переменной $\underline{f}$ и предложим следующую программу анализа булевой функции на ψ -разделимость по простой триаде.

§ 0 $\underline{f} \vee x \oplus \underline{f} \Rightarrow \underline{a}\,\underline{f} \vee y \oplus \underline{f} \wedge \underline{a} \Rightarrow \underline{a}\,\underline{f} \vee z \oplus \underline{f} \wedge \underline{a} \Rightarrow \underline{a}$

$\vee x \oplus \underline{a} \Rightarrow \underline{b}\underline{a} \vee y \oplus \underline{a} \vee \underline{b} \Rightarrow \underline{b}\underline{a} \vee z \oplus \underline{a} \vee \underline{b} \circ\!\!- \alpha$.

В приведенной программе индексы x, y и z служат для представления номеров тех из двоичных переменных $x_0, x_1, \ldots, x_{n-1}$, которые входят в заданную простую триаду. Выход из программы, соответствующий установлению ψ -разложимости анализируемой функции по этой триаде, условно представлен через α .

4. Программирующая система и понятие о втором уровне ЛЯПАСа

I. В отличие от языков типа *ALGOL -60*, в ЛЯПАСе применяются меры, позволяющие значительно облегчить процесс автоматического программирования ценой введения ряда ограничений, практически не приводящих к существенным затруднениям при исходном программировании. Так, например, чётко определен выбор всех операндов языка, которым раз и навсегда поставлены в соответствие определенные символы, ограничено возможное число предложений в программе первого уровня и т.п. Благодаря этому оказалось практически возможным ценой 2-3 человеко-месяцев создавать транслятор-I для произвольной УЦВМ, обеспечивающий скорость транслирования порядка I00 операций на синтезируемую команду и выдающий достаточно хорошие машинные программы [I7]. Весьма приближенно качество полученных программ можно оценить коэффициентом I,5, выражающим отношение объема (быстродействия) оцениваемой программы к объему (быстродействию) некоторой мыслимой идеальной машинной программы, соответствующей заданному алгоритму.

Механизм отладки программ, согласованный, например, с транслятором-I для машины "Урал-I" [I8], позволяет производить отладку программ непосредственно в языке ЛЯПАС. При этом имитируется работа отлаживаемой программы и выдается информация о траектории реализации (в форме последовательности номеров предложений) и о значениях, принимаемых интересующими отладчика операндами. Объем выводимой информации автоматически оптимизируется.

Интересно отметить универсальность отладочной процедуры, практически одинаковой для всех машин. Эта универсальность позволяет, например, эффективно использовать для отладки алгоритмов, выраженных на ЛЯПАСе, медленнодействующие машины типа "Урал-I", и реализовывать уже отлаженные алгоритмы на других, быстродействующих машинах.

2. Сложные вычислительные процессы обычно оказывается возможным разбивать на такие "куски", которые имеют самостоятельное значение и которым можно поставить в соответствие некоторые операторы. Введение этих операторов в язык и оформление соответствующих им программ в виде так называемых п о д п р о г р а м м, включаемых в постоянно растущую библиотеку, позволяет постепенно увеличивать эффективность обогащаемого таким образом языка, так как все большая часть работы по программированию производится автоматически. Улучшается и представление самих вычислительных процессов - повышается компактность и обозримость программ, ускоряется их составление и облегчается проверка.

Реализация этой достаточно общей идеи на языке ЛЯПАС привела к созданию второго уровня языка, обладающего рядом оригинальных особенностей, связанных с общими характеристиками системы ЛЯПАС.

Например, высокое быстродействие транслятора-I позволило представить всю библиотеку подпрограмм не в машинном языке, а в языке ЛЯПАС, то есть значительно более компактным образом и,

что представляется более важным, в форме, более удобной для компилирования.

На втором уровне ЛЯПАСа представляются алгоритмы с иерархической структурой, включающей Л-операторы, служащие идентификаторами процедур, выраженных находящимися в библиотеке подпрограммами. Структура взаимных связей между подпрограммами, на которые опирается оформляемая на втором уровне программа, представляется логическим деревом, корень которого соответствует самой программе, а прочие вершины - используемым в ней подпрограммам. Основным блоком транслятора-2, то есть транслятора со второго уровня на первый, является компилятор, монтирующий из подпрограмм требуемую программу первого уровня. При работе компилятора производится, в частности, "настройка" используемых подпрограмм, включающая замену формальных параметров фактическими (по терминологии *АЛГОЛа*), которым придается довольно широкий смысл - например, фактическими параметрами могут служить операторы и многосимвольные выражения. Этот процесс хорошо организуется именно в том случае, если он не привязывается к языкам конкретных машин, то есть если подпрограммы заданы в абстрактном языке. С другой стороны, в указанном случае программа компиляции сама становится удобной для её выражения в ЛЯПАСе, как это и сделано в системе ЛЯПАС.

Таким образом, транслятор-2 также можно хранить в памяти в компактной форме, а работу системы в целом организовывать следующим образом: сначала транслятор-2 пропускается через транслятор-I и получает выражение в машинном языке, а затем он обрабатывает исходную программу второго уровня и, пользуясь библиотекой подпрограмм, выраженных в ЛЯПАСе, выдает соответствующую программу на первом уровне, которая, в свою очередь, преобразуется в машинную программу, подготовленную к непосредственной реализации на машине. Разумеется, при наличии достаточно емких и быстродействующих запоминающих устройств блоки транслятора-2 можно хранить и в машинном языке.

3. На втором уровне предусмотрена возможность оперирования с переменными повышенной размерности, значение каждой из которых представляется в совокупности ячеек, объединяемых программно (при выполнении операций над этой переменной). Учитывая заданную в программе второго уровня информацию о том, какие переменные должны иметь повышенную размерность и какую именно, специальный блок транслятора-2 обеспечивает выражение процедур требуемых объединений ячеек средствами первого уровня ЛЯПАСа.

Решается также задача уплотнения информации, или упаковки комплексов, являющаяся немаловажной при недостаточном объеме памяти. При таком уплотнении каждый элемент некоторого выбранного комплекса используется для одновременного представления соответствующих элементов некоторых других комплексов. Наличие в трансляторе-2 специального блока уплотнения информации позволяет сохранить стандартную методику программирования, соответствующую случаю разнесенных комплексов, обеспечивая автоматическое преобразование программы первого уровня к форме, необходимой в ситуации упакованных комплексов. Получаемая форма оптимизируется.

Особый блок транслятора-2 проводит синтаксический анализ исходной программы, позволяя найти часть из допущенных при программировании ошибок. Организация поиска остальных ошибок осуществляется с помощью блока отладки, производящего подготовку программы к отладке путем некоторого её расширения в символическом виде. Корректировка программ, в которых обнаружены ошибки, производится с помощью блока коррекции, специализированного на операциях над строчками символов.

Все перечисленные блоки транслятора-2 так же, как и компилятор, оперирует лишь с выражениями ЛЯПАСа и сами оформлены в ЛЯПАСе. Такой принцип позволил существенно сократить сроки разработки транслятора-2.

Заслуживает внимания и входящая в систему ЛЯПАС программа динамической оптимизации использования памяти, следящая, в частности, за тем, чтобы растущие при реализации какой-либо программы комплексы не "наползли" друг на друга, и в случае, если это происходит, разносящая комплексы. При этом частота появления упомянутых критических ситуаций по возможности минимизируется с целью сокращения расходов дополнительного машинного времени на перемещение комплексов в памяти. Расходы времени на слежение за растущими комплексами могут быть сокращены самим программистом, от которого зависит разбиение всех комплексов на "фиксированные" и "плавающие" - слежение за фиксированными комплексами, привязанными к определенному месту памяти, не производится.

5. Некоторые пути автоматизации синтеза и перспективы развития системы ЛЯПАС

I. Анализ структуры вычислительных процессов, реализуемых при синтезе дискретных автоматов, показывает, что она, как правило, может быть разложена на части, которые носят достаточно общий характер и которые могут войти в алгоритмы решения логических задач весьма широкого класса. Такие части, могущие иметь различную интерпретацию, оформляются в виде п о д п р о г р а м м, включаемых в библиотеку ЛЯПАСа и записываемых в ЛЯПАСе, а реализуемые этими подпрограммами действия представляются символически $\mathcal{A}$ - о п е р а т о р а м и, включение которых во второй уровень ЛЯПАСа является эффективным средством обогащения языка.

К настоящему времени в библиотеку ЛЯПАСа включено более сотни подпрограмм, соответствующих таким операторам широкого назначения. К ним относятся, например, подпрограммы решения следующих задач: транспонирование двоичной матрицы, транзитивное замыкание бинарного отношения, построение матрицы совместимости состояний автомата, нахождение максимальных множеств совместимости, нахождение минимального замкнутого множества совместимости, решение системы логических уравнений, решение системы линейных уравнений над полем Галуа, обход логического дерева, вычисление значения булевой функции, генерирование сочетаний и перестановок, анализ булевой функции на однородность, линейное разделение булевой функции, факторизация булева выражения, анализ булевой функции на разделимость по заданному разбиению, анализ графа на связность и на 2-связность, проверка тождества в алгебре логики, нахождение нижней и верхней границ множества, получение безызбыточных покрытий множества (минимальных внешних устойчивых множеств графа), получение кратчайших покрытий, минимизация дизъюнктивной нормальной формы булевой функции и т.д.

Оперируя подобными операторами как элементами, удалось получить достаточно эффективные программы решения некоторых более сложных задач - минимизации числа состояний автомата, оптимизации алгебраической формы булевой функции по некоторым критериям, синтеза логических схем из пороговых и, в частности, мажоритарных элементов и т.д.

2. Однако, оказывается, что при решении подобных задач, если они не слишком просты, приходится сталкиваться с необходимостью затрат большого объема машинного времени, далеко не всегда практически доступного.

В связи с этим возникает, с одной стороны, проблема развития приближенных методов решения этих задач, позволяющих получать достаточно хорошие решения за приемлемое время, и, с другой стороны, проблема оптимизации алгоритмов по критерию быстродействия машинной реализации. Решение второй проблемы может быть получено лишь в том случае, если будут иметься о ц е н к и э ф ф е к т и в н о с т и подпрограмм, входящих в библиотеку. Например, для некоторых приближенных алгоритмов такие оценки могут выражать отношение качества полученных решений (математические ожидания значений некоторых величин, их дисперсии и т.п.) к стоимости их получения (в частности, стоимостью может служить время решения).

Задача получения оценок эффективности алгоритмов является, в общем случае, довольно трудной и зачастую её не удается решить чисто аналитически, а требуется решать на УЦВМ либо непосредственно экспериментально-статистическим путем, либо комбинируя счет по некоторым формулам со статистическим экспериментом, что представляется наиболее перспективным. Для решения этой задачи в системе ЛЯПАС строится необходимая экспериментальная база, включающая, в частности, генераторы задач с заданными статистическими характеристиками и алгоритмы получения оценок алгоритмов, испытываемых на задаваемом потоке задач. Эти работы создают предпосылки для выполнения цикла исследований по сравнительному анализу различных алгоритмов синтеза и уточнению областей их оптимальности.

3. В свою очередь, накопленные на описанном пути результаты будут использованы при построении м е т а п р о г р а м м синтеза - программ, сочетающих функции анализа постановки подлежащей решению задачи, расчленения алгоритма её решения на этапы и оптимального прохождения каждого из этих этапов за счет выбора соответствующего частного алгоритма, оказывающегося наилучшим в ситуации данного этапа.

Создание комплекса метапрограмм явится, по-видимому, первым шагом на пути следующего расширения системы ЛЯПАС, связанного с введением третьего уровня ЛЯПАСа. Этот уровень, разрабатываемый в настоящее время, будет не языком представления алгоритмов, а языком постановки задач, значительно облегчающим связь с машиной. Соответствующая ему часть программирующей системы, называемая транслятором-3, должна обеспечивать авто-

матический переход от языка постановки задач к языку представления алгоритмов их решения.

Разумеется, построение такого транслятора представляет,в общем случае, очень сложную задачу. Поэтому предполагается, что первоначальный круг задач, на который будет рассчитан третий уровень ЛЯПАСа, будут довольно ограниченным, ориентируясь, в основном, на тематику теории синтеза; затем он будет постепенно расширяться. Работа транслятора-3 будет опираться на обширную библиотеку подпрограмм, в связи с чем предполагается провести своего рода "замыкание" алгоритмов, абстрагированных из области теории синтеза, на решение задач, возникающих при автоматизации программирования. К таким задачам можно отнести, например, задачу минимизации числа предложений или числа операндов в программе, задачу оптимизации структуры алгоритма методами некоторых эквивалентных преобразований, синтез коммутирующих систем (переключателей) и т.д.

Особый интерес представляет задача предварительного исследования статистики входных потоков однотипных задач с последующим синтезом алгоритмов, оптимальных относительно полученной статистики.

Таким образом, транслятор-3 должен реализовывать процессы с а м о п р о г р а м м и р о в а н и я на основе информации, представляемой на третьем уровне ЛЯПАСа и задающей лишь формулировки задач некоторых классов.

Л и т е р а т у р а

1. Unger S.H. A computer oriented toward spatial problems. PIRE,1958,46,N 10,1744-1750.
2. Закревский А.Д. Универсальная система для решения задач типа синтеза релейных схем. Труды СФТИ, 1963, вып.42, 9-37.
3. Научный отчет проблемной лаборатории счетно-решающих устройств ТГУ по теме: "Исследование алгоритмов синтеза цифровых автоматов на УЦВМ", Томск, 1960.
4. Стогний А.А. Минимизация булевых функций на ЭЦМ. Сборник "Вопросы вычислительной техники". Гостехиздат УССР, 1961.
5. IEEE Trans.Electron.Comput., the special issue on computer languages,1964,EC-13,N 4.

6. Wells M.B. Aspects of language design for combinatorial computing. [5], 431-438.
7. Sammet J.E., Bond E.R. Introduction to FORMAC.[5],386-394.
8. Braffort P., Hirschberg D., Mommens J. La programmation des problèmes non numériques. "Rapport CETIS",1964,N 31.
9. Zakrevskiy A.D. On conference papers by S.Valigorskiy and A.Stogny. Translated proceedings of the International Symposium on relay systems and finite automata, Moscow, October 1962, Burroughs Corp.,1964,33-34.
10. Закревский А.Д. ЛЯПАС - логический язык представления алгоритмов синтеза. Материалы научных семинаров по теоретическим и прикладным вопросам кибернетики. Теория автоматов, Киев, 1964.
11. Товштейн М.Я. Инструкция к использованию программирующей программы ПП-ЛЯПАС-1 (там же).
12. Труды Сибирского физико-технического института. 1965,вып.48, (специальный выпуск по языку ЛЯПАС).
13. Логический язык для представления алгоритмов синтеза релейных устройств. Изд. "Наука", М., 1965 (в печати).
14. Лавров С.С. Универсальный язык программирования. Изд."Наука", М., 1964.
15. Берж К. Теория графов и её применения. ИЛ, М., 1962.
16. Закревский А.Д. К синтезу схем из мажоритарных элементов. Труды СФТИ, 1964, вып. 44, 17-23.
17. Торопов Н.Р. Транслятор для машины "Урал-1". В [13].
18. Закревский А.Д. Отладка $\mathcal{L}$-программ на машине "Урал-1". В [13].

Сибирский физико-
технический ин-т.

Поступила в редакцию
23.IУ.1965 г.