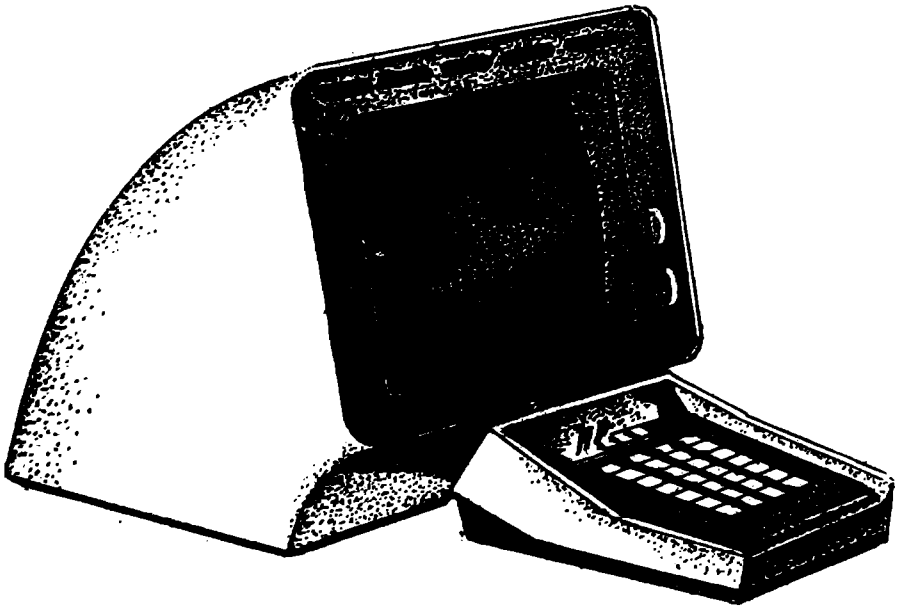
Ю.М. Вишняков



Системное программирование. Конечные распознаватели

ГОСУДАРСТВЕННЫЙ КОМИТЕТ РСФСР
ПО ДЕЛАМ НАУКИ И ВЫСШЕЙ ШКОЛЫ

ТАГАНРОГСКИЙ РАДИОТЕХНИЧЕСКИЙ ИНСТИТУТ
имени В. Д. КАЛМЫКОВА

Ю. М. ВИШНЯКОВ

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ КОНЕЧНЫЕ РАСПОЗНАВАТЕЛИ

Учебное пособие

Таганрог 1991

УДК 681.3.06(075.8)+681.3.062(075.8)

Системное программирование. Конечные распознаватели: Учебное пособие/Ю. М. Вишняков; Таганрог радиотехн. ин-т. Таганрог, 1991. 74 с.
ISBN 5—230—16544—8

Рассматриваются вопросы построения конечных распознавателей в виде автоматных моделей на основе регулярных грамматик и выражений. Обсуждаются автоматы общего вида с магазинной памятью и их области использования. Излагаются подходы к решению проблемы идентификации неавтоматными методами.

Рецензенты:

Кафедра прикладной математики и вычислительной техники Ростовского института сельхозмашиностроения.

Р. А. Ашинянц — доцент кафедры АМ-7 Московского института приборостроения.

ISBN 5—230—16544—8

С О Д Е Р Ж А Н И

ВВЕДЕНИЕ

Зародившись в недрах цифровой техники, конечный автомат перерос ее рамки и превратился в одно из фундаментальных понятий дискретной математики. И сегодня трудно указать области технической кибернетики, где в той или иной мере , явно или неявно , это понятие не использовалось бы.

Относительно области системного программирования автоматы тесно смыкаются с синтаксическими методами анализа и перевода и находят применение благодаря следующим свойствам:

ряд математических положений из теории автоматов позволяет эффективно формализовать процесс решения задач из области трансляции;

программная реализация автоматов на ЭВМ достаточно проста и эффективна и использует фиксированный, как правило, небольшой объем памяти.

Термин "конечный автомат", в зависимости от подразумеваемой предметной области использования употребляется в разных смыслах. Но всегда общим в этих определениях выступает обработка входных последовательностей символов некоторого алфавита, а различия проявляются в определении выходных функций и наполнении их физическим смыслом.

В дальнейшем изложение будет ориентировано на модель автомата, распознающего некоторое подмножество последовательностей входных символов. Эту модель чаще всего называют конечным распознавателем и кладут в основу конструирования сканеров или лексических анализаторов.

Настоящая работа ставит своей целью раскрытие понятия конечного распознавателя, являющегося одним из фундаментальных в системном программировании и синтетически вобравшим в себя формальные грамматики и языки, регулярные множества и выражения, а также конечные автоматы.

Автор глубоко убежден, что результативное обучение может быть достигнуто только на пути от простого к

сложному, через раскрытие физического смысла понятий, их взаимосвязей, и старался, где это возможно, избегать излишней формализации. Насколько это удалось судить читателям.

2. ОСНОВНЫЕ ПОНЯТИЯ ИЗ ТЕОРИИ ФОРМАЛЬНЫХ ГРАММАТИК И ЯЗЫКОВ

2.1 Алфавит,цепочка,катенация

А л ф а в и т A - это некоторое непустое множество элементов, называемых символами.
Например, A= { 0,1,2 } , здесь 0,1,2 -символы алфавита A.

Ц е п о ч к а - это некоторая последовательность символов алфавита A, возможно повторяющихся. Цепочки 012, 001201, 1120 составлены из символов алфавита A. Для обозначения цепочки обычно используют малые буквы. Число входящих в цепочку букв называют ее длиной. Так для цепочки x=012210 длина, обозначаемая через |x|, равна 6.

Вводится так называемая пустая цепочка, обозначаемая через Л. Она не содержит ни одного символа и имеет длину |Л|= 0.

Над цепочками можно проводить операцию катенации (соединения). Пусть x = 012 и y = 210 , тогда катенация xy = 012210 , а катенация yx = 210012. Очевидно, что Лx = xЛ = x.

Повторяющиеся катенации одной и той же цепочки называются ее степенью.

$$x^0 = Л \quad \text{(допущение)}$$

$$x^1 = x$$

$$x^2 = xx$$

$$x^n = \underbrace{xx.....x}_{n\text{-раз}}$$

Грамматика, сентенциальная форма, язык

Формально грамматика определяется следующим образом.

Грамматикой G[Z] называют непустое множество продукционных правил, где Z - начальный символ.

Продукционное правило (продукция, правило вывода) есть упорядоченная пара (U,x), записываемая в виде

U x,

где U левая часть правила (в нашем случае символ),

x непустая цепочка, образующая правую часть правила.

Грамматика задается на алфавите V , называемом словарем. Сам словарь V состоит из двух непересекающихся подмножеств V_T и V_N , где V_T - словарь терминальных символов и V_N - словарь нетерминальных символов. Терминальный символ может встречаться только в правой части правил, а нетерминальный - как в левой , так и правой частях правил.

Обычно в грамматиках для наглядности в нетерминальные символы вкладывают смысловую нагрузку , и тогда их имена представляют какое-нибудь понятие или его аббревиатуру. В этом случае терминальные символы заключаются в угловые скобки. В качестве примера приведена грамматика G[<число>],описывающая целые десятичные числа:

```
<число>::= <чс>
<чс>   ::= <чс><цифра>
<чс>   ::= <цифра>
<цифра>::= 0
<цифра>::= 1
.................
<цифра>::= 9
```

Здесь словарь V = {<число>,<чс>,<цифра>,0,1,...,9}; V_T = {0,1,...,9}; V_N= {<число>,<чс>,<цифра>}.

Если правила содержат одинаковые левые части, то их объединяют следующим образом:

U::= x ; U::= y ; U::= z -----> U::= x!y!z

Символ ! несет смысл логической связки ИЛИ. Такое представление грамматики называется нотацией Бэкуса-Наура. Модифицированная грамматика G[<число>] теперь запишется в виде

<число>::= <чс>
<чс> ::= <чс><цифра>!<цифра>
<цифра>::= 0!1!...!9

Определим символ => .Пусть задана некоторая грамматика G[Z] и в ней имеется правило U::= u . Если имеется цепочка x= aUb , то из нее путем замены U на u можно получить новую цепочку y= aub ,что записывается как

x=> y

В этом случае говорят, что цепочка x порождает цепочку y, или цепочка y выводима из цепочки x, или вывод цепочки y из цепочки x, или y редуцируется к x.

Если при выводе цепочки y из x подстановка выполняется только один раз, то это простой вывод или вывод длины 1. Если существует последовательность простых выводов и подстановка созерается n раз, то это вывод длины n, что записывается как

x=>+ y

С понятием вывода очень тесно связано математическое понятие отношения. Действительно, если рассмотреть некоторое непустое множество цепочек, то на нем можно построить множество пар цепочек, правая из которых в паре выводима из левой. Тогда в терминах отношений :

=> - отношение простого вывода;
=>+ - транзитивное замыкание отношения простого вывода ;
=>* - рефлексивное транзитивное замыкание отношения простого вывода.

Раскроем физический смысл отношений. Пусть заданы две цепочки a и b из некоторого конечного множества. Тогда отношение:

a=> b представляет множество всех упорядоченных пар цепочек, для которых a всегда порождает b;

a=>+ b представляет множество всех упорядоченных пар цепочек, включающее отношение простого вывода, отношения вывода длины 2,3,...,n,...;

a=>* b представляет объединение отношения a=> b и отношения a = a.

С е н т е н ц и а л ь н а я ф о р м а. Пусть задана некоторая грамматика G[Z], тогда цепочка y, выводимая из Z, называется сентенциальной формой, что записывается как

$$Z \Rightarrow^{*} y$$

П р е д л о ж е н и е. Если сентенциальная форма y состоит только из терминальных символов, то она называется предложением.

Я з ы к. Языком L грамматики G[Z] называется множество всех предложений

$$L = \{\, y \quad y - \text{предложение} \,\}.$$

Проинтерпретируем все эти понятия на примере грамматики G[<число>].

Здесь начальным символом является <число>.

<число>=> <чс> - это простой вывод, где x = <число>, y = <чс> и использовано правило вывода <число>:: = <чс>;

<число>=> <чс>=> <чс><цифра>=> <цифра><цифра>=> 1<цифра>=> 15 - это вывод <число>=>+ 15 и длина его равна 5. здесь 15 является предложением.

Очевидно, что языком грамматики G[число] является множество целых чисел.

Ф р а з а. Это понятие является одним из ключевых в синтаксическом анализе.

Пусть задана грамматика G[Z] и в ней для некоторого нетерминала U имеется выводимая из него подцепочка u:

$$U \Rightarrow^{+} u$$

Пусть имеется сентенциальная форма

$$x = a\ u\ b,$$

тогда в этой цепочке u является фразой для нетерминала U, если при замене u на U образуется новая сентенциальная форма y = a U b, т.е.

$$Z \Rightarrow^{*} a\ U\ b$$

Фраза является простой, если вывод $U \Rightarrow u$ является простым, что эквивалентно наличию в грамматике правила U::= u.

Например, для грамматики G[число] рассмотрим сентенциальную форму

$$x = \langle чс \rangle\ \langle цифра \rangle$$

а) проверим <чс> - фраза для нетерминала <число>? Выполним подстановку в соответствии с правилом <число>::= <чс>, получим новую цепочку

$$y = \langle число \rangle\ \langle цифра \rangle,$$

однако цепочка y не является сентенциальной формой, следовательно, <чс> не фраза;

б) проверим <цифра> - фраза для нетерминала <чс>? Выполним подстановку в соответствии с правилом <чс>::= <цифра>, получим новую цепочку:

y = <чс><чс>.

Здесь y также не является сентенциальной формой, отсюда <цифра> не фраза;

в) проверим, вся ли цепочка x =<чс><цифра> является фразой для нетерминала <чс>?
Выполним подстановку в соответствии с правилом <чс>:: = <чс><цифра> получим новую цепочку

y = <чс>.

Цепочка y является сентенциальной формой, т.е. <число>=> <чс>, следовательно вся цепочка x является фразой и причем простой.

2.3. Понятие разбора, синтаксические деревья

Основная задача синтаксического анализа состоит в установлении факта принадлежности некоторой входной цепочки языку грамматики, а сам процесс установления принадлежности называется разбором.

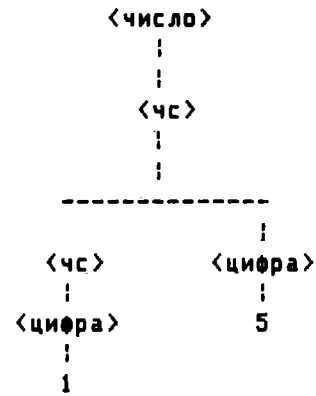
Разбор может строиться двояко. Либо, продвигаясь от начального символа, строится из него вывод цепочки x, либо предпринимается попытка редуцировать цепочку x к начальному символу грамматики. В первом случае разбор называется нисходящим, а во втором - восходящим.

Наглядно разбор интерпретируется специальным графом, называемым синтаксическим деревом. С синтаксическим деревом связывается также ряд понятий, содержание которых раскрывается ниже

1. Узлы синтаксического дерева сопоставляются с нетерминальными и терминальными символами.
2. Узлы дерева связаны ветвями, отображающими вывод.
3. Корнем дерева называется узел, не имеющий входящих ветвей. Корень дерева соответствует начальному символу грамматики.
4. Концевые узлы. Узлы, не имеющие подчиненных узлов.
5. Куст узла. Это множество подчиненных этому узлу других узлов.

6. Поддерево. Это часть дерева, имеющая свой корень.

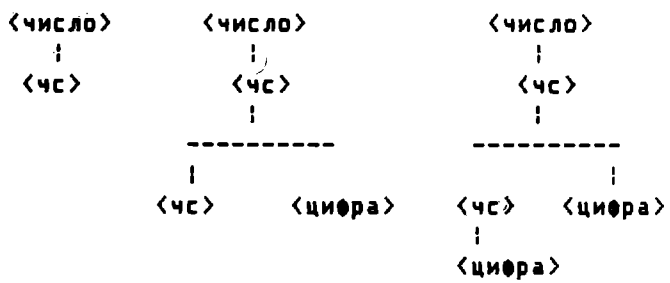
Проинтерпретируем данные понятия на примере дерева, соответствующего выводу <число>=>+ 15 в грамматике G[число]:

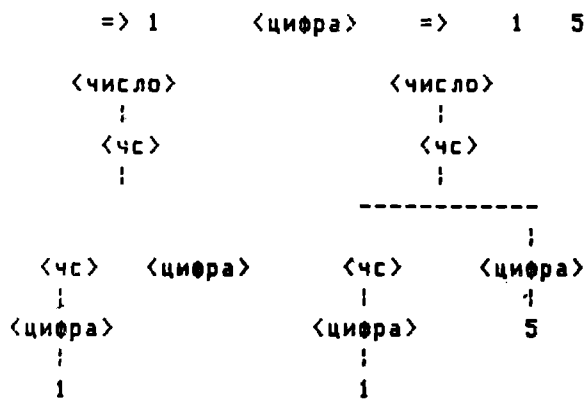


Здесь узел <число> является корнем дерева. Узлы 1 и 5 являются концевыми. В куст узла <чс> входят узлы <чс> и <цифра>. Куст узла <число> образует единственный узел <чс>.

Следующий пример показывает связь между выводом и синтаксическим деревом:

<число>=> <чс>=> <чс> <цифра>=> <цифра> <цифра>=>





Из этого примера очевидно утверждение о том, что каждому выводу однозначно соответствует дерево, обратное чаще всего не выполняется. Действительно, в выводе <число>=>+ 15 на промежуточном этапе из подцепочки <цифра> <цифра> могут быть построены выводы:

<цифра> <цифра>=> 1 <цифра>=> 15;
<цифра> <цифра>=> <цифра> 5=> 15.

Таким образом, одно и то же синтаксическое дерево может соответствовать более чем одному выводу.

Синтаксические деревья являются инструментом, позволяющим наглядно интерпретировать разбор. Действительно:

концевые узлы указывают на выводимую (редуцируемую) сентенциальную форму;

куст узла соответствует подцепочке, являющейся простой фразой для данного узла;

концевые узлы поддерева образуют фразу для корня этого поддерева.

Отсюда отсечение куста или его присоска эквивалентны построению простого вывода.

В контексте данного изложения укажем на ряд частных определений, не требующих детальных пояснений, которые используются в восходящем синтаксическом анализе.

П р о с т о й в ы в о д

$$x\ U\ y \Rightarrow x\ u\ y$$

называется каноническим, если y подцепочка, состоящая из терминалов.

В ы в о д

$$V \Rightarrow^{+} W$$

называется каноническим, если он состоит из последовательности простых канонических выводов.

Определение. Сентенциальная форма, имеющая канонический вывод, называется канонической.

Определение. Самая левая простая фраза сентенциальной формы называется основной.

Обычно для обозначения канонического вывода вместо символа $\Rightarrow$ используется символ $\succ$

2.4. Классификация языков

В 1959 году американский ученый Н. Хомский предложил классифицировать языки по типу правил подстановки порождающей грамматики. Порождающая грамматика определяется четверкой:

$$(V, V_T, P, Z),$$

где V - алфавит, V_T - словарь терминальных символов, P множество продукционных правил, Z начальный символ.

С учетом этого классификация выглядит следующим образом:

0 грамматики имеют правила

$$u ::= v,$$

где $u \in V^{+}$ и $v \in V^{*}$

0 - грамматики называют еще грамматиками фразовой структуры. Они моделируют естественные языки;

1 грамматики имеют правила вида

$$xUy :: = xuy ,$$

где $U \in V \setminus V_T$; $u \in V^+$; $x \in V^*$; $y \in V^*$

Грамматики данного класса называются контекстно-чувствительными, поскольку подстановка u вместо U может быть выполнена только в контексте x y;

2 грамматики имеют правила вида

$$U :: u ,$$

где $U \in V \setminus V_T$; $u \in V^+$

Эти грамматики называют контекстно свободными (КС-грамматиками). Чаще всего именно этот тип грамматик моделирует языки программирования;

3 грамматики имеют правила вида

$U :: T$ или $U :: = WT$,

где $T \in V_T$; $U \in V \setminus V_T$; $W \in V \setminus V_T$.

Грамматики этого класса называют А-грамматиками или автоматными. Они порождают так называемые регулярные языки, разбор предложений которых может быть выполнен конечным автоматом. Подробно всем этим вопросам будут посвящены следующие разделы.

3. КОНЕЧНЫЕ АВТОМАТЫ

Настоящий раздел знакомит читателя с понятием конечного автомата (КА), раскрывает его связь с автоматными грамматиками и регулярными выражениями.

3.1. Детерминированный конечный автомат

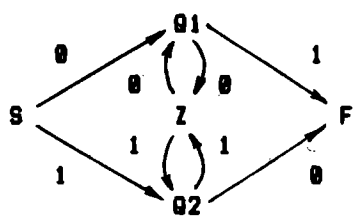
Детерминированный конечный автомат (ДКА)- это пятерка

(K, V , M, S, Z),

где K - алфавит состояний; V - входной алфавит, M - функция переходов (представляет отображение K * V→K); S - начальное состояние; Z - множество заключительных состояний.

Для понимания ДКА лучше всего представить в виде устройства, имеющего конечное число внутренних состояний K. Под воздействием цепочек символов входного алфавита ДКА изменяет текущее состояние, реализуя функции переходов M. В нашем случае ДКА должен распознать некоторое множество цепочек, образующее входной язык, а остальные - отвергнуть. Для этого часть состояний Z⊂K отмечается в качестве допустимых, а остальные - отвергающих. Свою работу ДКА начинает всегда с некоторого состояния S , называемого начальным.

Очень удобно представить ДКА диаграммой состояний. Диаграмма состояний - это ориентированный граф, вершины которого сопоставлены с состояниями КА, а дуги взвешены входными символами. Ниже представлена диаграмма состояний КА, который допускает цепочки символов входного алфавита (0,1), состоящие из любого числа пар одинаковых символов



Здесь S - начальное состояние, Z - допускающее (конечное) состояние, Q1, Q2- состояния автомата, F - состояние ошибки.

Функция переходов M определяет последующее состояние КА Q через текущее состояние Q и входной символ C , т.е.

$$Q_{l}^{t+1} = M(Q_{m}^{t}, C^{t}).$$

Обычно строчные символы дискретного времени опускают. В примере функция переходов M имеет вид:

M(S,0) = Q1; M(S,1) = Q2; M(Q1,0) = Z; M(Q1,1) = F;

M(Q2,0) = F; M(Q2,1) = Z; M(Z,0) = Q1; M(Z,1) = Q2;

M(F,0) = M(F,1) = F.

Другой наиболее удобный и распространенный способ для машинной реализации представляет задание функции переходов матрицами переходов. Столбцы матрицы отождествляются с символами входного алфавита, строки- с состояниями, отдельный элемент задает частичную функцию переходов M(Q,C). Ниже представлена матрица переходов. К ней приписан один "лишний" столбец меток, который содержит символы Y и N, отмечающие , допускающие и отвергающие состояния.

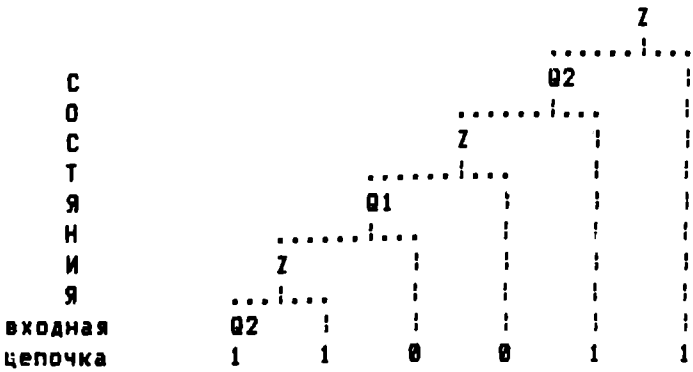
K\V	0	1	метка
S	Q	Q	N
Q	Z	F	N
Q	F	Z	N
Z	Q	Q	Y
F	F	F	N

Разберем как работает автомат над входными цепочками:

входная цепочка	1	1	0	0	1	1
последовательность состояний	S → Q2 → Z → Q1 → Z → Q2 → Z					

Данная цепочка допускается, поскольку последнее состояние Z является допускающим.

Представим проверку цепочки несколько по иному:



Такое графическое представление есть не что иное, как синтаксическое дерево, а его построение можно интерпретировать как разбор в некоторой грамматике G. Нетрудно убедиться в том, что грамматика является автоматной и имеет вид

$$Z ::= Q2\ 1 \mid Q1\ 0$$
$$Q2 ::= Z1 \mid 1$$
$$Q1 ::= Z0 \mid 0$$

Здесь Z, Q1 , Q2 - нетерминалы; 1,0 - терминалы.

При построении конечных распознавателей решают задачу обратного перехода от грамматики к конечному автомату. Он проводится по следующему алгоритму:

1. Отождествить нетерминалы с состояниями автомата, терминалы - со входными символами;
2. Ввести дополнительное начальное состояние S.
3. Для всех правил вида

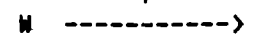
$$Q = T\ ,$$

где $Q \in V_M$ $T \in V_T$, построить фрагмент диаграммы состояний вида

4. Для всех правил вида

$$Q ::= WT,$$

где $Q \in V_N$, $W \in V_N$, $T \in V_T$, построить фрагмент диаграммы состояний вида

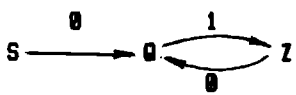


5. Разметить состояния на отвергающие и допускающие.

Пример. Пусть задана грамматика G[Z] вида

$$Z ::= Q1$$
$$Q ::= 0 | Z0$$

Состояния эквивалентного автомата K={Z,Q}. Символы входного алфавита V ={0,1}. Допускающее состояние Z. Диаграмма состояний имеет вид



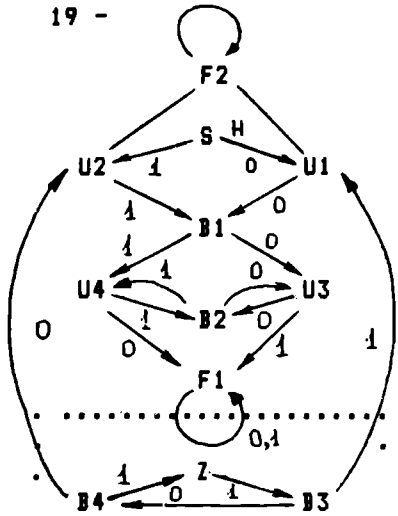
3.2. Минимизация автоматов

В математической логике одну и ту же булеву функцию может представлять множество дизъюнктивных нормальных форм, одна из которых является минимальной. Здесь можно аналогию распространить на автоматы, когда одно и то же множество входных цепочек может быть распознано некоторым множеством автоматов. В этом смысле все они являются эквивалентными, но различаются числом состояний и функциями переходов.

Дальнейшее изложение посвящено построению минимального по числу состояний КА из некоторого исходного представления.

Пусть задан некоторый КА A, функция переходов и диаграмма состояний которого представлены ниже.

		0	1	
н	S	U1	U2	N
	U1	B1	F2	N
	U2	F2	B1	N
	B1	U3	U4	Y
	F2	F2	F2	N
	U3	B2	F1	N
	U4	F1	B2	N
	B2	U3	U4	Y
	F1	F1	F1	N
	Z	-	B3	Y
	B3	B4	U1	Y
	B4	U2	Z	Y



В автомате S - начальное состояние, B1, B2, Z, B3, B4 - допускающие состояния, остальные - отвергающие. Читателю предлагается самостоятельно убедиться, что КА допускает цепочки, состоящие из пар 00 и 11.

Проинтерпретируем некоторые понятия на данном примере.

Недостижимые состояния. Среди состояний автомата могут находиться такие, которые недостижимы из начального состояния ни для какой входной цепочки. Соответствующие строки в таблице переходов, а также вершины и связи в диаграмме состояний можно удалить. В примере строки недостижимых состояний выделены рамкой.

Тривиальный алгоритм удаления недостижимых состояний заключается в построении новой матрицы переходов на основе старой следующим образом:

1. Внести в новую матрицу строку начального состояния S:

	0	1
S		

2. На основе старой матрицы переходов вписать в элементы матрицы новы" состояния:

	1
S	U1 U2

3. Дописать новые строки для состояний U1 и U2

	0	1
S	U1	U2
U1		
U2		

4. Вписать в элементы строк U1 , U2 новые состояния в соответствии с функцией переходов

	0	1
S	U1	U2
U1	B1	F2
U2	F2	B1

Процесс продолжать до тех пор, пока не прекратится дописывание строк новых состояний. Все недостижимые состояния будут отсутствовать в новой матрице переходов.

Эквивалентность состояний

Два состояния Q и L являются эквивалентными, если:

- оба они допускающие или отвергающие (условие подобия);
- любая цепочка t переводит их в эквивалентные состояния P и R:

$$(Q \xrightarrow{t} P;\ L \xrightarrow{t} R)$$ (условие преемственности).

Таким образом, два состояния являются эквивалентными, если не существует различающей их цепочки.

Отметим, что эквивалентность состояний является

эквивалентностью в математическом смысле. Отношение рефлексивно (каждое состояние эквивалентно само себе), симметрично (если Q эквивалентно L, то L эквивалентно Q), транзитивно (если Q эквивалентно L, а L эквивалентно P, то Q эквивалентно P).

Эквивалентные состояния можно объединить в одно, тем самым сокращая лишние состояния.

Проверим эквивалентность состояний B1 и B2

Для этих состояний выполняется условие подобия оба они допускающие. Проверку преемственности удобно проводить путем построения таблицы эквивалентности. Здесь столбцы соответствуют входным символам ,

1 шаг

	0	1
B1, B2		

Отмечаем пару состояний B1 и B2, которую проверяем на эквивалентность.

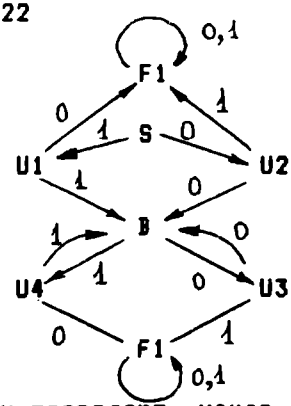
2 шаг

	0	1
B1, B2	U3	U4

В клетки в соответствии с функцией переходов вписываем пары преемников для входных символов 0 и 1 соответственно. Преемники эквивалентны в силу свойства рефлексивности. Отсюда B1, B2 эквивалентны. Если эквивалентность установить не удалось, то надо дописать строки преемников, и проверить их на эквивалентность и т.д.

Объединяем состояния B1 и B2 в B. Теперь новый автомат имеет вид:

	0	1		
S	U1	U2	¦	N
U1	B	F2	¦	N
U2	F2	B	¦	N
B	U3	U4	¦	Y
F2	F2	F2	¦	N
U3	B	F1	¦	N
U4	F1	B	¦	N
F1	F1	F1	¦	N



Таблицы эквивалентности позволяют, начав проверку с одной пары, выяснить попутно эквивалентность и других пар.

Тем не менее полной уверенности в том, что выявлены все эквивалентные состояния, нет. Минимизация автоматов этим способом достаточно трудоемка, поскольку всегда возникает вопрос, а все ли состояния проверены на эквивалентность.

Приведем более эффективный метод поиска и объединения эквивалентных состояний, называемый иногда методом разбиения. Его суть заключается в следующем.

1. Вначале состояния автомата разбиваются на блоки, один из которых включает допускающие состояния, другой- отвергающие. В нашем случае разбиение имеет вид

P0 = ({S, U1, U2, F2, U3, U4, F1}, {B1, B2})

1-й блок 2-й блок

Очевидно, ни одно из состояний 1-го блока не эквивалентно ни одному из состояний 2-го блока, поскольку не выполняется условие подобия. Эквивалентные состояния могут содержаться в пределах одного блока (условие подобия).

2. Разобьем P0 относительно входного символа 0.

По матрице переходов в 1- м блоке состояниями-преемниками для (S, U2, F2, U4, F1) являются состояния 1-го блока, а для (U1, U3) состояния-преемники находятся во втором блоке. Следовательно, U1 и U3 не могут быть эквивалентны ни одному из состояний: S, U2, F2, U4, F1.

Следовательно, 1-й блок в P0 разбивается на два.
Аналогичную процедуру проведем над 2-м блоком P0

P1 = ((S, U2, F2, U4, F1), (U1, U3), (B1, B2))

1-й блок 2-й блок 3-й блок

3. Точно так же разобьем P1 относительно символа 1:

P2 = ((S, F2, F1), (U2, U4), (U1, U3), (B1, B2)).

4. Проведем разбиение P2 относительно символа 0:

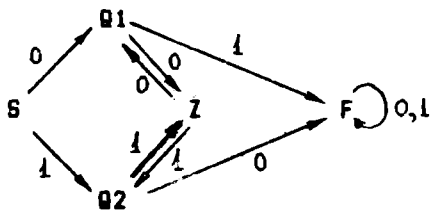
P3 = ((S), (F1, F2), (U2, U4), (U1, U3), (B1, B2)).

Дальнейшее разбиение невозможно, следовательно, блоки содержат эквивалентные состояния. Введем имена блоков

(S) ---> S; (F1, F2) ---> F; (U2, U4) ---> Q2;
(U1, U3) ---> Q1; (B1, B2) ---> Z

и произведем замену всех эквивалентных состояний в матрице переходов на новые имена, получим новую матрицу переходов и соответствующую ей диаграмму состояний:

	0	1
S	Q1	Q2
Q1	Z	F
Q2	F	Z
F	F	F
Z	Q1	Q2



Итак, исходный КА имеет 12 состояний. Минимальный КА имеет только 5 состояний, а сравнение диаграмм состояний исходного и минимизированного КА наглядно демонстрирует настоятельную необходимость этапа минимизации.

В общем случае правило построения нового разбиения можно сформулировать следующим образом. При образовании нового блока в него надо включать только те состояния,

которые имеют преемников в пределах одного блока старого разбиения.

3.3. Недетерминированный конечный автомат

Недетерминированный конечный автомат (НКА)-это пятерка

$$(K, V, M, S, Z),$$

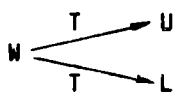
где K алфавит состояний, V - входной алфавит, M - функция переходов, представляющая отображение KxV ---> в подмножество K; S множество начальных состояний ($S \subset K$), Z - множество заключительных состояний.

Обратим внимание на принципиально различные определения в ДКА и НКА таких понятий, как начальные состояния (одно в ДКА и несколько в НКА) и функции переходов.

Как правило, НКА порождаются регулярными грамматиками, которые содержат правила вида

U:: WT и L:: = WT,

что соответствует следующему фрагменту диаграммы состояний:



Частичная функция переходов этого случая имеет вид

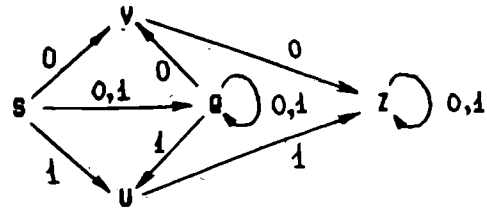
$$M(W,T) = \{U,L\}.$$

Цепочка t допускается НКА, если в диаграмме состояний найдется для нее путь из некоторого состояния $P \in S$ в состояние $Q \in Z$.

Пусть задана регулярная грамматика G[Z]:

```
Z::   U1|V0|Z0|Z1
U::   Q1|1
V:: = Q0|0
Q::   Q0|Q1|0|1
```

Построим соответствующую диаграмму состояний:



Здесь:
- алфавит состояний K = {S, Q, V, U, Z};
- входной алфавит V = {0,1};
- множество заключительных состояний {Z};
- множество начальных состояний {S};
- функция переходов, заданная перечислением:

M(S,0) = {V,Q}; M(V,0) = {Z}; M(U,0) = 0; M(Q,0) = {Q,V};
M(S,1) = {Q,V}; M(V,1) = 0; M(U,1) = {Z}; M(Q,1) = {U,Q};
M(Z,0) = {Z}; M(Z,1) = {Z}.

Покажем разбор входной цепочки, реализуемый данным НКА:

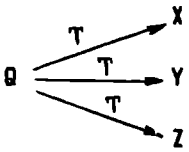
Номер текущего шага	Текущее состояние	Остаток входной цепочки	Возможные преемники	Выбор преемника
1	S	01001	V,Q	Q
2	Q	1001	U,Q	Q
3	Q	001	V,Q	V
4	V	01	Z	Z
5	Z	1	Z	Z

Здесь построение правильного пути всегда связано с выбором требуемого состояния из нескольких альтернатив. При этом выборе не всегда является таким наглядным, как в примере, а программная реализация НКА такой уж тривиальной.

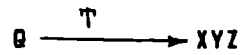
3.4. Преобразование НКА в ДКА

Это преобразование основано на замене альтернативных путей разбора в НКА одним путем в эквивалентном ему ДКА.

НКА



ДКА



Три альтернативных состояния X, Y, Z в НКА представляются одним [XYZ] в ДКА, которое представляет первые три состояния. Итак, если НКА представляется пятеркой

$$(K, V_T, M, S, Z),$$

то эквивалентный ДКА имеет вид

$$(K', V_T, M', S', Z').$$

Алфавит состояний K' определим через подмножества алфавита K. Элементы K' будем обозначать через [S1 S2 Sl], где S1, S2,...,Sl-состояния из K. Таким образом, число элементов в K' равно числу подмножеств в K.

Функция переходов M'

$$M'([S1,S2,\ldots,Sl],T) = [R1,R2,\ldots,Rt],$$

если в НКА

$$M(\{S1,S2,\ldots,Sl\},T) = \{R1,R2,\ldots,Rt\}.$$

Начальное состояние S' = [S1,S2,...,Sp], если в НКА S = {S1,S2,...,Sp}.

Множество заключительных состояний Z образуется из элементов, в которых присутствует хотя бы одно состояние из Z ДКА.

Как НКА, так и эквивалентный ему ДКА, построенный по вышеприведенным правилам, допускают одни и те же входные цепочки.

Тривиальный алгоритм построения ДКА заключается

следующем:

1. Построить множество состояний K' путем перечисления всех подмножеств K.
2. Выделить из множества K' начальное состояние S' и множество заключительных состояний Z'.
3. Создать пустую матрицу переходов, строки которой пометить элементами из K', а столбцы – символами входного алфавита.
4. Заполнить матрицу переходов.

Тривиальный алгоритм хотя и прост, тем не менее построенный ДКА практически всегда содержит много недостижимых состояний. Поэтому лучше при построении ДКА воспользоваться алгоритмом, устраняющим сразу и недостижимые состояния:

1. Построить начальное состояние S' и отметить им первую строку матрицы переходов.
2. Вычислить все состояния K', которые могут быть достигнуты из данного состояния.
3. Если во вновь вычисленных состояниях не встречаются такие, которые отмечали ранее строки матрицы переходов, то для них завести новые строки и повторить п.2.
4. Выделить в матрице переходов заключительные состояния Z'.

Проведем построение ДКА для НКА (разд. 3.3).

1. S' = [S]

	0	1
[S]		

2. Продолжить вычисления состояний для строк [VQ] и [UQ]:

	0	1
[S]	[VQ]	[UQ]
[VQ]		
[UQ]		

3. Продолжить вычисления состояний для строк [VQZ], [UQZ]:

	0	1
[S]	[VQ]	[UQ]
[VQ]	[VQZ]	[UQ]
[UQ]	[VQ]	[UQZ]
[VQZ]		
[UQZ]		

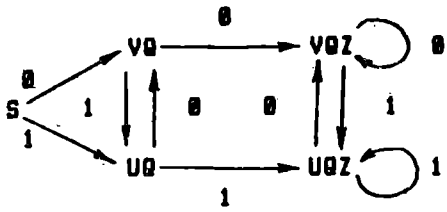
4. Продолжить вычисления состояний для строк [VQZ], [UQZ]:

	0	1
[S]	[VQ]	[UQ]
[VQ]	[VQZ]	[UQ]
[UQ]	[VQ]	[UQZ]
[VQZ]	[VQZ]	[UQZ]
[UQZ]	[VQZ]	[UQZ]

5. Дописывание новых строк не производится. Матрица переходов построена:

	0	1
[S]	[VQ]	[UQ]
[VQ]	[QZ]	[Q]
[UQ]	[Q]	[QZ]
[QZ]	[VQZ]	[UQZ]
[Q]	[VQ]	[UQ]
[VQZ]	[VQZ]	[QZ]
[UQZ]	[QZ]	[UQZ]

ДКА имеет следующую диаграмму состояний:



Здесь [S] - начальное состояние; Z' = {[VQZ],[UQZ]} - заключительные состояния.

Минимизируем ДКА методом разбиения.

1. P0 = ({[S],[VQ],[UQ]}, {[VQZ],[UQZ]})
   1-й блок 2-й блок

2. Разобьем P0 относительно входного символа 0:

   P1 = ({[S],[UQ]}, {[VQ]}, {[VQZ],[UQZ]})

3. Разобьем P1 относительно 1:

   P2 = ({[S]}, {[UQ]}, {[VQ]}, {[VQZ],[UQZ]})

4. Разобьем P2 относительно 0:

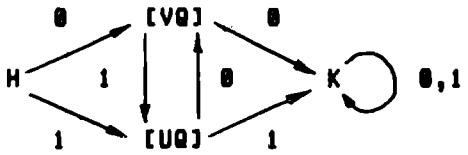
   P3 = ({[S]},{[UQ]},{[VQ]},{[VQZ],[UQZ]}.

   Разбиение P2 = P3.

Следовательно, дальнейшее разбиение на блоки надо прекратить. Состояния [VQZ] и [UQZ] являются эквивалентными.

Введем новые имена для [S] - H, для [VQZ] и [UQZ] - K, после чего матрица переходов минимизированного ДКА примет вид:

	0	1
H	[VQ]	[UQ]
[VQ]	K	H
[UQ]	H	K
K	K	K



Здесь H - начальное состояние, K - заключительное.

4. РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ И КОНЕЧНЫЕ АВТОМАТЫ

4.1 Модифицированная форма Бэкуса-Наура и регулярные выражения

В разделе 2 были рассмотрены основные понятия теории формальных грамматик, основанной на нотации Бэкуса-Наура. Иногда для представления синтаксиса грамматику модифицируют путем введения дополнительных метасимволов типа: { } - фигурные скобки и ( ) - круглые скобки.

Если имеется запись вида {a}, где a - некоторая цепочка, то это означает, что цепочка a может отсутствовать либо повторяться любое число раз. В НБФ для этой цели служит рекурсия. Например, в грамматике В[<число>] имеется правило вида

<чс>:: = <чс><цифра>|<цифра>

В модифицированной НБФ это же правило можно записать следующим образом:

<чс>:: = {<цифра>} <цифра>

Иногда можно встретить запись вида {a} , что определяет число вхождений цепочки a, которых может быть от двух до пяти.

Метасимволы "круглые скобки" используются точно так же как и в арифметических выражениях. Приведенные ниже записи

являются эквивалентными:

ab|ac|ad -----> a(b|c|d)

В теории автоматов используются так называемые регулярные множества (РМ) и для их представления регулярные выражения (РВ).Рассмотрим эти понятия.

Пусть задан некоторый алфавит S = {S1, S2,...,Sn} , тогда:

- 0, Л, S1, S2,..., Sn- регулярные выражения на множестве S;
- если e - регулярное выражение, то (e), {e} также регулярные выражения;
- если e1 и e2 - регулярные выражения , то e1e2 , e1|e2 также являются регулярными выражениями.

РВ e порождает РМ |e|, которое является его значением:

- |0| = 0;
- |Л| = {Л};
- |Si| = {Si};
- |(e)| = |e|;
- |e1e2| = |e1| |e2| {xy x |e1|; y |e2|};
- |e1|e2| = |e1| U |e2| = 0;
- |{e}| = |e|* - т.е. итерация |e|.

Сопоставляя модифицированную НБФ и РВ, нетрудно заметить, что РВ тождественны правой части правил грамматки. Так,правило E:: = a{1} и РВ a{1} задают одно и то же множество цепочек вида

a; a1, a11, a111,...

Однако построение конечного автомата из модифицированной НБФ по рассмотренному в разделе 3 способу требует ее преобразования к НБФ. Это не всегда является тривиальной задачей, что покажем на следующем примере.

Пусть имеется грамматика G[Z] в виде Z::={a|b}c.

Требуется ее привести к НБФ, где правила имеет вид U::= VT|T. Пусть A::={a|b}, тогда B[Z] имеет вид

Z::=Ac - соответствует правилу НБФ;
A::={a|b} - не соответствует.

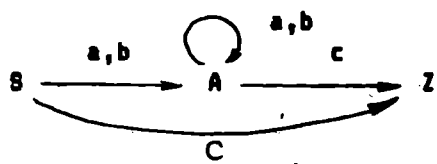
Поскольку A:: = {a|b|aa|ab|ba|...}, то, используя рекурсивность, B[Z] преобразуем к виду

Z:: = Ac
A:: = A|a|b|Aa|Ab

Однако здесь требует развязки правило вида A::= Λ , которое в таком виде в НБФ не используется. Это правило задает нулевое вхождение A в правиле Z:: = Ac. Вводим новое правило Z:: = c для компенсации A::= Λ , после чего B[Z] примет вид

Z:: = Ac|c
A:: = Aa|Ab|a|b

Эквивалентная диаграмма состояний КА теперь строится просто и имеет вид



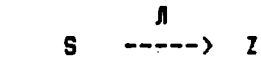
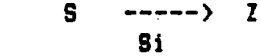
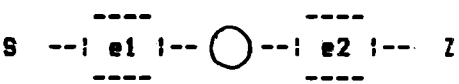
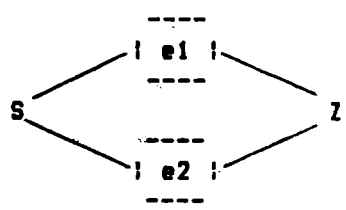
Читателям предлагается убедиться самостоятельно, что преобразование грамматики Z:: = {(a)b}c к НБФ вида рассмотренным способом невозможно.

4.2. Система переходов

Рассмотрим специфическую диаграмму состояний применительно к РВ, которую назовем системой переходов (СП). Пусть e ∈ некоторое РВ, тогда ему соответствует следующая СП:

Эта СП имеет одно начальное и одно заключительное состояния. Под воздействием цепочек символов, являющихся значениями РВ e, происходит их распознавание конечным распознавателем, который переходит из состояния S в состояние Z.

Типовые базовые конструкции СП для РВ:

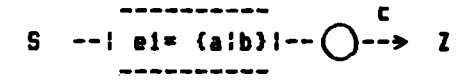
Регулярное выражение	Система переходов
∅	
Λ	
S	
e1 e2	
e1 e2	
(e)	

Поскольку составное РВ можно всегда представить в виде композиции более простых, то, проводя аналогичную процедуру над исходной СП, можно всегда ее представить в виде композиции элементарных СП. Например:

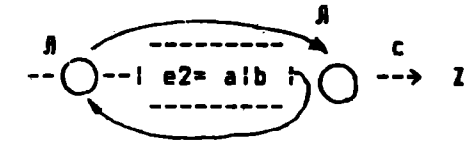
Анализируемая часть РВ	Система переходов
e = (a|b)c	

Анализируемая часть РВ Система переходов

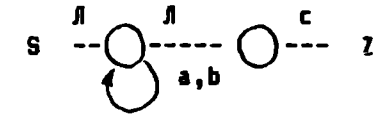
e1 = (a|b)



e2 = a|b



e3 a, e4 = b

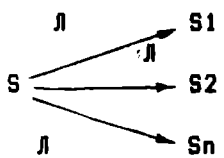


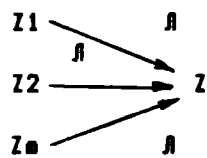
Дальнейшая декомпозиция невозможна.

С м ы с л д у г и Л. Если два состояния A и B в СП связаны дугой Л , то в данный момент может быть текущим как состояние A, так и B. Нетрудно убедиться, что распознаватель, заданный СП, допускает все цепочки, порождаемые соответствующим РВ.

4.3. Преобразование системы переходов в конечный автомат

Преобразование диаграммы состояний в СП с учетом физического смысла дуги Л является тривиальным. Пусть имеется диаграмма состояний НКА, в которой S1,S2,...,Sn - начальные состояния, а Z1,Z1,...,Zm-заключительные. Введем для эквивалентной СП начальное состояние S и заключительно Z .Если из S провести дуги во все S1,S2,...,Sn, а из Z1,Z2,...,Zm - в Z, то диаграмма состояний преобразуется в эквивалентную СП, что и показано ниже:

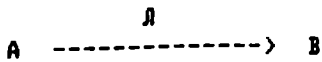




Преобразование СП в диаграмму состояний основывается

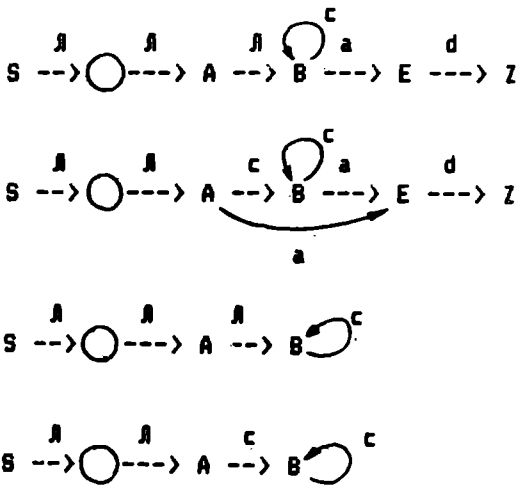
на этом же свойстве Л-дуг и связано с их последовательным исключением. Алгоритм преобразования выглядит следующим образом:

1. В СП найти некоторое состояние, из которого исходит дуга Я , продвигаться по дугам Л , пока не встретится ситуация, когда из некоторого состояния А в состояние В имеется дуга Л , а из В не исходит дуг



Если такой ситуации не окажется, то перейти к п.3 алгоритма.

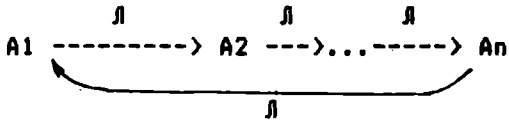
2. Если имеется последовательность дуг Л ,ведущих из начального состояния S в A ,а следовательно,и в B ,то B сделать начальным состоянием. Если B — заключительное состояние, то A сделать также заключительным состоянием. Убрать дугу Л из A в B и на вершину A навесить все дуги, исходящие из B. Пример ниже иллюстрирует такие ситуации.



S-начальное состояние
Z-заключит. состояние
S,B-нач. состояния
Z-заключит. состояние

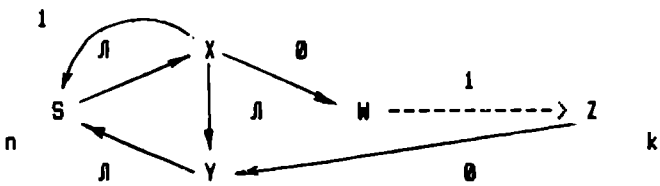
S-начальное состояние
B-заключит. состояние
S-начальное состояние
A,B-заключит. состояния

. Если отсутствует ситуация,отмеченная в п.1 алгоритма, о обязательно имеется ситуация вида

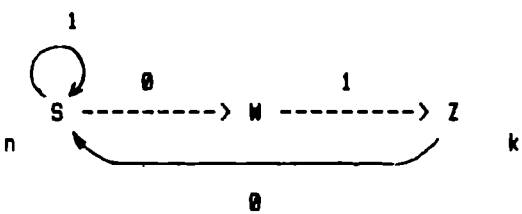


этом случае все дуги Л убираются, а все Ai отождествляются с одним состоянием A.Если одно из состояний Ai было начальным или заключительным, то состояние A также становится начальным или заключительным соответственно.

Например, следующая СП



преобразуется к виду



Здесь в S стягиваются состояния S,X,Y, и оно становится начальным.

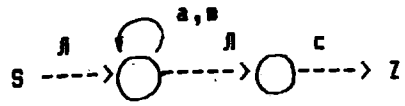
Алгоритм повторяется до тех пор, пока не останется ни одной дуги Л. В этом случае СП преобразуется к эквивалентной диаграмме состояний НКА.

В качестве иллюстрации приведем построение КА для РВ вида e={a|b}.

Регулярное выражение

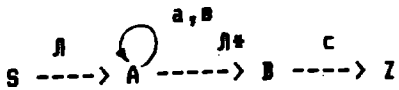
e=(a|b)c

Система переходов

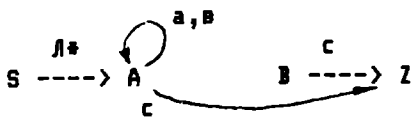


Здесь S-начальное состояние, Z-заключительное состояние.

Введем имена для непомеченных состояний и через Л* отметим первую удаляемую Л - дугу:

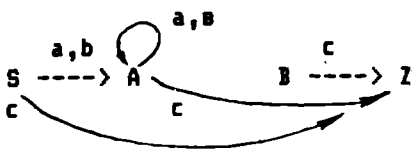


После удаления Л- дуги получим:



Начальные состояния-S,B
Z-заключит. состояние

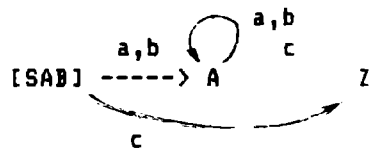
Устраним следующую дугу Л:



S,A,B-начальные состояния
Z-заключит. состояние

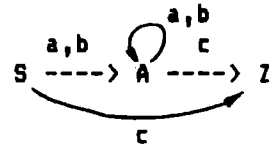
Полученная диаграмма состояний представляет НКА. Преобразуем данный НКА в эквивалентный ДКА. [SAB]-начальное состояние ДКА, а матрица переходов , построенная в соответствии с разделом 3.3, и соответствующая диаграмма состояний имеют вид

	a	b	c
[SAB]	A	A	Z
A	A	A	Z
Z	.	.	.



Перекодируем [SA]] в S, получим:

	a	b	c
S	A	A	Z
A	A	A	Z
Z			



Данному ДКА соответствует грамматика вида

Z::= Ac!c
A::= Aa!Ab!a!b

Таким образом преобразование модифицированной НБФ к НБФ возможно на пути РВ-СП-НКА-ДКА-грамматика. Отметим, что здесь не рассмотрены вопросы преобразования ДКА в грамматику, которая может быть как левосторонней, так и правосторонней.

5. ПРЕОБРАЗОВАНИЕ РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ В КОНЕЧНЫЙ АВТОМАТ НА ОСНОВЕ ОБРАТНОЙ ПОЛЬСКОЙ ЗАПИСИ

Одну из ключевых ролей в системном программировании играет обратная польская запись (ОПЗ), названная так в честь польского математика Я. Лукасевича, который впервые использовал ее в задачах математической логики. ОПЗ обладает двумя замечательными свойствами:

- отсутствием скобок;
- возможностью вычисления выражения путем однократного просмотра ОПЗ слева направо.

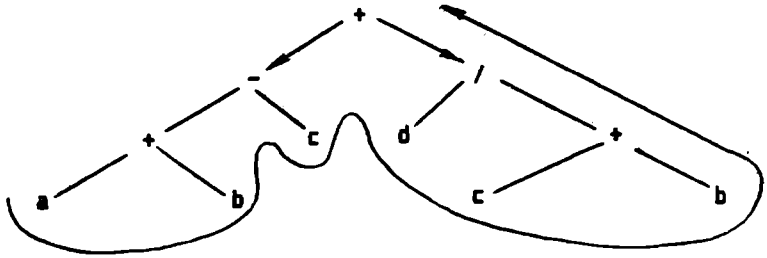
Благодаря этим свойствам ОПЗ широко применялась в построении первых компиляторов, основанных на прямых методах трансляции.

5.1 Основные понятия

Пусть задано арифметическое выражение

(a + b) / c + d / (e + b)

Представим его в виде дерева, в узлах которого разместим операции, а в концевых узлах - операнды.



Если обходить это дерево и записывать последовательно его символы, начиная с левого нижнего концевого узла, причем узел рассматривать только после того,как рассмотрены все исходящие из него ветви, то искомая последовательность символов будет представлять ОПЗ. В нашем случае ОПЗ арифметического выражения имеет вид

a b + c / d e b + / +

Вычисление ОПЗ осуществляется по следующему алгоритму.

1. Посмотреть очередной символ ОПЗ (просмотр осуществляется слева направо).
2. Если символ - операнд, то рассматривается следующий символ.
3. Если символ - к-арная операция, то из входной последовательности выбираются предшествующие к символов-операндов, над ними выполняется эта операция и результаты записываются на место самого первого операнда во временную переменную ri .Символ операции и все операнды, кроме модифицированного первого, из ОПЗ удаляются.

Например, вычислим следующую ОПЗ:

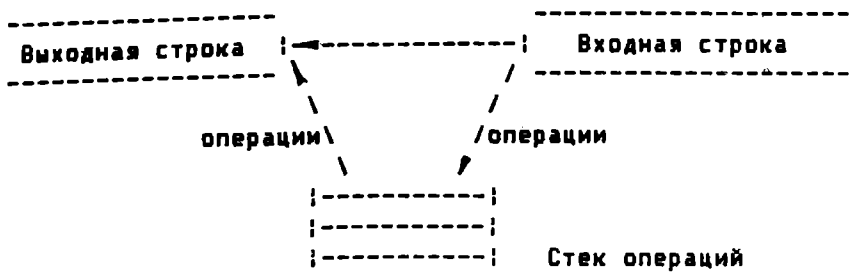
N	Состояние входной строки ОПЗ	Действие
1	a b + c / d e b + / +	Просмотреть след. символ
2	a b + c / d e b + / +	Просмотреть след. символ
3	a b + c / d e b + / +	r1 = a + b
4	r1 c / d e b + / +	Просмотреть след. символ
5	r1 c / d e b + / +	r1 = r1 / c
6	r1 d e b + / +	Просмотреть след. символ
7	r1 d e b + / +	Просмотреть след. символ
8	r1 d e b + / +	Просмотреть след. символ
9	r1 d e b + / +	r2 = e + b
10	r1 d r2 / +	r1 = d / r2
11	r1 r2 +	r1 = r1 + r2
	Вычисление ОПЗ окончено	

Один из эффективных алгоритмов по переводу арифметических выражений в ОПЗ предложил голландский ученый Е.В.Дейкстра. В его основу положен стек и таблица приоритетов операций, которые являются обратными привычным вычислительным приоритетам.

Операции, ограничители	Приоритет
(	0
)	1
+ , -	2
* , /	3
!	4

Здесь 0 - самый высокий приоритет, 4 - самый низкий.

Приведенный ниже рисунок иллюстрирует действие алгоритма.



1. Входная строка строится посимвольно слева направо. Если символ- операнд, то он переписывается в выходную строку, если - операция, то совершается следующая последовательность действий:
   а) при пустом стеке операция помещается в его вершину;
   б) если приоритет у анализируемого символа ниже, чем у операции в вершине стека, то он добавляется в стек.
   в) если приоритет у анализируемого символа равен или больше, чем у операции в вершине стека, то из стека в выходную строку последовательно выталкиваются все знаки с меньшим или равным приоритетом в сравнении с анализируемым символом. После этого входной символ добавляется в вершину стека.
2. Обработка скобок. Скобка (всегда записывается в вершину стека. Если анализируемый символ), то из стека выталкиваются все операции до (. Сами скобки уничтожаются и в выходную строку не попадают.
3. Если обработаны все символы входной строки, оставшиеся знаки стека дописываются в выходную строку. Например, в нашем случае процесс получения ОПЗ имеет вид:

Вых. стр.	a	b	+	c	/	d	e	b	+	/	+			
Сос- тоя- ния стека	(	(	+	+	/	/	+	+	/	(	(	+	+	/
		(	(				+	/	/	(	(	+		
							+	+	/	/				
								+	+					
Вх. стр.	(a + b) / c + d / (e + b)													

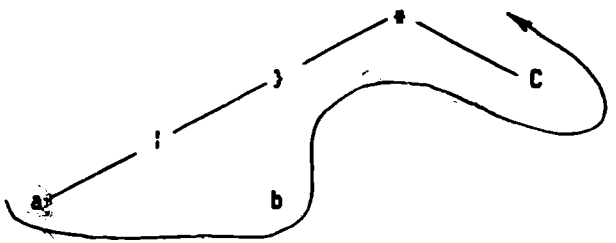
5.2. Обратная польская запись регулярного выражения

Нет никаких запретов для введения понятия ОПЗ применительно к регулярным выражениям. Более того , в дальнейшем убедимся, что именно ОПЗ позволяет провести построение конечных автоматов большой размерности, наглядное представление которых в виде диаграмм состояний невозможно.

Введем понятие ОПЗ регулярного выражения, для чего выделим следующие операции:

* - операция сцепления. Ее будем употреблять для сцепления отдельных символов РВ e1 и e2 . Так, e1e2 теперь будем записывать в виде e1 * e2.
| - операция ИЛИ. Смысл не изменяется.
} одноместная (унарная) операция итерации, которая эквивалентна { }. Так, РВ { a } будет иметь запись вида a}.

Пусть задано РВ e = {a/b}c. Построим дерево и выполним его обход по ранее рассмотренным правилам. ОПЗ будет иметь вид



a b | } c *

Преобразование РВ к ОПЗ можно также выполнить на основе алгоритма Дейкстры, для чего определим приоритеты операции следующим образом.

Операция	Приоритет
(,)	0
{ , (	1
|	2
* .	3

В алгоритме обработка операций производится следующим образом:

- круглые скобки обрабатываются без изменений;
- { открывающая фигурная скобка размещается в вершину стека.
- } фигурная закрывающая скобка выталкивает из стека все

знаки операций до ближайшей (включительно и выталкивает в выходную строку операцию).

Например, получим ОПЗ РВ e (a|b)cd(b|d). Введем в него операцию *., после чего оно преобразуется к виду

e = (a b) * c * d * (b d).

Процесс получения ОПЗ выглядит следующим образом:

Вых. строка	a	b	)	c	*	d	*	b	d	*
Стек	(	(			*	*	*	*	*					*
		(	(							*	*	(	(	
												*	*	
Вх.стр	(	a	b	)	*	c	*	d	*	(	b	d	)	

Таким образом, ОПЗ РВ e имеет вид

a b) c * d * b d *

5.3. Интерпретация операций. Вычисление ОПЗ регулярного выражения

Следует оговорить сразу, что будем понимать под вычислением ОПЗ. Для этого наполним это понятие физическим смыслом на конкретном примере.

Пусть задано ОПЗ предыдущего примера. Воспользуемся стандартной методикой вычисления ОПЗ.

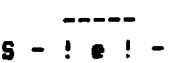
Остаток ОПЗ	Действия
a b) c * d * b d *	просмотреть следующий символ
a b) c * d * b d *	----*----*----*----*----
a b) c * d * b d *	построить систему переходов R1= a | b

Остаток ОПЗ	Действия
	a,b S ------> Z
R1) c * d * b d *	выполнить итерацию над R1 ; R1 =(R1)
	Л Л S -----> Q1 -----> Z a,b
R1 c * d * b d *	просмотреть очередной символ
R1 c * d * b d *	построить СП R1 = R1 *c
	Л Л c S ---> Q1 ---> Q2 ---> Z a,b
R1 d * b d *	просмотреть очередной символ
R1 d * b d *	построить СП R1 = R1 * d
	Л Л c d S --> Q1 --> Q2 --> Q3 --> Z a,b
R1 b d *	просмотреть очередной символ
R1 b d *	просмотреть очередной символ
R1 b d *	построить СП R2 =b!d
	b,d S ------> Z
R1 R2 *	построить СП R1 R1 * R2
	Л Л c d S --> Q1 --> Q2 --> Q3 --> Q4 --> Z a,b
	вычисление окончено

Таким образом, построение СП может быть выполнено по всем правилам вычисления ОПЗ. Здесь неявно введены множество имен состояний Q={Q1,Q2,...} для идентификации вновь появляющихся состояний и множества объектов R={R1,R2,...}, играющих роль вспомогательных элементов для размещения промежуточных результатов. Интерпретация Ri зависит от формы представления СП (либо в форме графа, либо в матричной форме). Поскольку представление СП переходов большой размерности в виде графа теряет наглядность, то в дальнейшем будем вести речь о матричном представлении. Для этой цели определим матричные операции над РВ.

Пусть задано РВ e Для него СП и матрица переходов имеют вид:

Система переходов

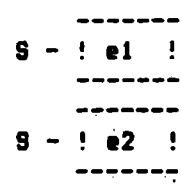


Матрица системы переходов

	e
S	Z
Z	

Операции сцепления * соответствуют:

а) Системы переходов



Результат

б) Матрицы системы переходов

	e
S	Z
Z	

*

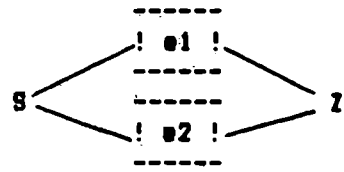
	e
S	Z
z	

Результат

	e	e
S	Q1	
Q1		Z
Z		

Операции ! соответствуют:

а) S - ! e1 ! - Z
S - ! e2 ! - Z

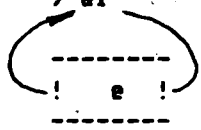


б)

! e1 !	! e2 !	! e1 ! e2 !
S ! Z !	S! Z !	S ! Z ! Z !
Z !	Z! !	Z ! ! !

Операции (e) соответствуют:

а) S -----> Q1 -----> Z



в)

! e !	! e ! Л !
S ! Z !	S ! ! Q1 !
Z ! !	Q1! Q1 ! Z !
	Z ! ! !

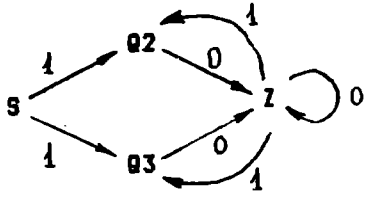
Здесь во всех операциях Q1 является очередным свободным именем из списка Q .

Пусть заданы две системы переходов в виде

а) S ---0---> Q1 ---1---> Z
Q1 ---0---> Q1

R1	0	1
S	Q1,Z	
Q1		Z
Z	!	.

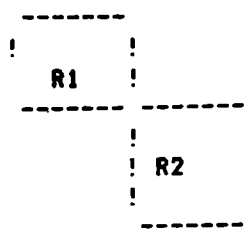
б)



R2	0	1
S		Q2,Q3
Q2	Z	
Q3	Z	
Z	Z	Q2,Q3

Вычислим R1 * R2:

- построим новую матрицу по схеме



	0	1	0	1
S	Q1,Z			
Q1		Z		
Z				
S				Q2,Q3
Q2			Z	
Q3			Z	
Z			Z	Q2,Q3

- введем состояние Q4. Переименуем Z в матрице R1 и S в матрице R2 в Q4:

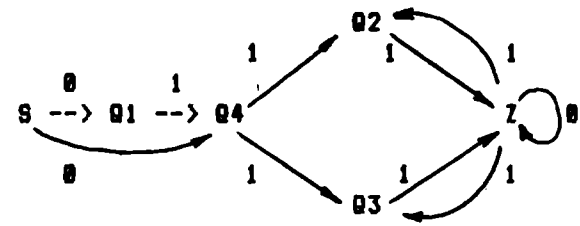
	0	1	0	1
S	Q1,Q4			
Q1		Q4		
Q4				
Q4				Q2,Q3
Q2			Z	
Q3			Z	
Z			Z	Q2,Q3

-объединим строки Q4:

	0	1	0	1
S	Q1,Q4			
Q1		Q4		
Q4			Q2,Q3	
Q2			Z	
Q3			Z	
Z			Z	Q2,Q3

-объединим одинаковые столбцы:

	0	1
S	Q1,Q4	
Q1		Q4
Q4		Q2,Q3
Q2	Z	
Q3	Z	
Z	Z	Q2,Q3



Вычислим R1|R2:

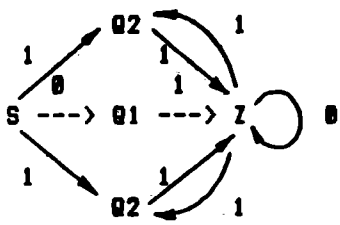
-построим матрицу по схеме

R1	R2

	0	1	0	1
S	Q1,Z			Q2,Q3
Q1		Z		
Q2			Z	
Q3			Z	
Z			Z	Q2,Q3

-объединим одинаковые столбцы

	0	1
S	Q1,Z	Q2,Q3
Q1		Z
Q2	Z	
Q3	Z	
Z	Z	Q2,Q3



Вычислим (R1):

-выберем очередное свободное имя Q2 и переименуем состояния S и Z. Получим матрицу R1 в виде

R1	0	1
Q2	Q1,Q2	
Q1		Q2
Q2		

-объединим одноименные строки:

R1	0	1
Q2	Q1,Q2	
Q1		Q2

строим новую матрицу R по схеме

R	Л	
S	Q2	
R1		
Q2	Z	
Z		

R	0	1	
S			Q2
Q2	Q1,Q2		
Q1		Q1	
Q2			Z
Z			

Искомая матрица получается после объединения одноименных столбцов и одноименных строк.

В качестве примера приведем вычисление ОПЗ (с целью упрощения анализ операторов опустим):

a b) c * d * b d) *

```
N   Остаток ОПЗ          Действия
-------------------------------------------------------
1.! ab!}c*d*bd!*     R1=a   b:
                   !- построить матрицу R1:
                              !  a  ! b !
                             -------------
                              S! Z     Z
                              Z!
2.! R1}c*d*bd!*      R1= (R1):
                   !- выбрать очередное имя Q1 из Q.
                   !- Переименовать S и Z в R1 в Q1
                              !  a  ! b !
                             -------------
                             Q1!Q1   !Q1
                             Q1!     !
                   !- объединить одинаковые строки
                              !  a  ! b !
                             -------------
                             Q1!Q1   !Q1 !
                   !- построить новую матрицу
                   !         !  a  ! b !
                             -----------------
                   !         S           ! Q1
                             Q1!Q1   !Q1
  !                !         Q1!     !
                   !         Z!      !
                   !- объединить одинаковые строки
                              !  a  ! b !
                             -----------------
                             S              Q1
                             Q1!Q1   !Q1
                   !          Z!     !
3.!R1c*d*bd!*      ! R1=R1 * C:
  !                !- дописать к матрице R1 матрицу c
                              !  a  ! b !   ! c
                             -------------------
                             S !          ! Q1!
                             Q1!Q1   !Q1   Z !
                              Z!     !
                             ------------------
                   .         S .     .   . . !Z
```

```
                !-выбрать новое состояние Q2 из Q.
                  Переименовать в матрице R1 состояние
                  Z  и в матрице с состояния S в
                  состояния Q2.
                           !  a     b        c
                         --------------------
                         S                Q1!
                         Q1!Q1    !Q1     Q2!
                         Q2!      !         !
                         --------------------
                         Q2!                !Z
                         Z !
                    объединить одинаковые строки
                           !  a   ! b !    ! c
                         --------------------
                         S                Q1!
                         Q1!Q1    !Q1     Q2!
                         Q2!      !         !Z
                         Z !                !
R1d*bd!*            R1=R1*d:
                !-дописать к матрице R1 матрицу d
                           !  a   ! b !    ! c !d
                         -------------------------
                !        S                Q1!
                         Q1!Q1    !Q1     Q2!
                         Q2!      !         !Z
                         Z !      !         !
                         -------------------------
                         S                       Z
                         Z
                !-выбрать новое состояние Q3 из Q.
                !Переименовать в матрице R1 состояние
                !Z  и состояние S в матрице d  в
                !состояние Q3.Объединить одинаковые
                !строки.
                !             a     b        c !d
                         -------------------------
                         S        !      Q1!
                         Q1!Q1    !Q1    Q2!
                         Q2!                !Q3
                         Q3!                !      Z
                         Z !
```

5.!R1bd!*
!
!
R2=b!d:
построить матрицу R2

	b	d
S	Z	Z
Z		

6.!R1R2*
!
R1=R1*R2:
!-дописать к матрице R1 матрицу R2

	a	b		c	d	b	d
S			Q1				
Q1	Q1	Q1	Q2				
Q2				Q3			
Q3					Z		
Z							
S						Z	Z
Z							

!-выбрать новое состояние Q4 из Q.
переименовать состояние Z в матрице R1 и состояние S в матрице R2 в Q4.
объединить одинаковые строки.

	a	b		c	d	b	d
S			Q1				
Q1	Q1	Q1	Q2				
Q2				Q3			
Q3					Q4		
Q4						Z	Z
Z							

!-объединить одинаковые столбцы

	a	b	c	d	
S					Q1
Q1	Q1	Q1			Q2
Q2			Q3		
Q3				Q4	
Q4		Z		Z	
Z					

7. Вычисление окончено.

СП , построенная по матрице СП , совпадает с предыдущим результатом. Таким образом, введение понятия ОПЗ для РБ позволяет достаточно просто и эффективно организовать

процесс построения СП для сложных РВ. Дальнейшее преобразование матрицы СП в диаграмму состояний НКА и далее в ДКА, а также их минимизация затруднений не вызывают. Читателю в качестве упражнения предлагается построить эти алгоритмы.

6. АВТОМАТЫ С МАГАЗИННОЙ ПАМЯТЬЮ

Модель конечного автомата, введенная ранее, может решать ограниченный круг задач, требующих фиксированного объема памяти. Однако в задачах, подобных обработке парных скобок в арифметических выражениях, такая модель не работает. В этом случае используют более мощную модель автомата с магазинной памятью (МП - автомат), раскрытию которой посвящено последующее изложение.

6.1. Задание нерегулярных множеств цепочек

В разделе 4 были описаны регулярные множества цепочек. Они получаются, если над символами некоторого алфавита A проводить операции *, ¦, ( ). Более того, применение операций произведения, объеденения и итерации к регулярным множествам также порождает регулярные множества. Дополним эти операции еще двумя способами, позволяющими получать нерегулярные множества.

а) Использование степеней над символами алфавита.

$$\{\ 1^n\ 0^n \quad n>0\ \}.$$

Цепочки этого множества имеют вид

10
1100
....
11...1 00...0
n n

Для множества

$$\{\ (11)^n\ (00)^m\ \mid\ n>0,\ m>0\ \}$$

цепочки имеют вид

1100 n=m=1

11111100 n=3;m=1.

Таким образом, использование указателей степеней как целочисленных переменных определяет конфигурацию цепочек нерегулярного множества

б) Операция обращения цепочки.

Если w некоторая цепочка abc , то ее обращение, обозначаемое как w , равно cba Здесь r - знак обращения цепочки.

Например, цепочки множества

$$(w^r w \quad w \quad (0|1))$$

имеют вид

$$w^r \quad w$$

$$0 \quad 0$$

0111 1110

6.2 Определение МП-автомата

МП-автомат определяется пятью объектами:

1) алфавитом входных символов (обычно в него включают концевой маркер ^);
2) алфавитом магазинных символов, включающим маркер дна Z;
3) алфавитом состояний, включающим начальное состояние;
4) управляющим устройством, которое каждой комбинации:
   - входного символа,
     магазинного символа,
   - символа состояния

ставит в соответствие ВЫХОД или ПЕРЕХОД.
Здесь ВЫХОД - это конец распознавания;
ПЕРЕХОД - выполняется действие над магазином, состоянием и входом;
5) начальным содержимым магазина.

В МП-автомате запрещены операции, которые запрашивают входной символ после концевого маркера и выталкивают маркер дна магазина.

Во время ПЕРЕХОДА возможны следующие действия:
-ВЫТОЛКНУТЬ- верхний символ выталкивается из магазина;
-ВТОЛКНУТЬ(А) - в магазин вталкивается символ А;
-СОСТОЯНИЕ(S) - следующим текущим состоянием становится S;
-СДВИГ- осуществляется просмотр следующего текущего символа входной цепочки;
-ДЕРЖАТЬ- текущий входной символ остается прежним.

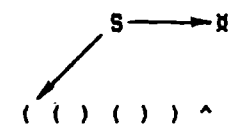
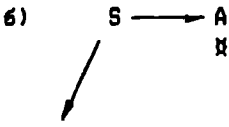
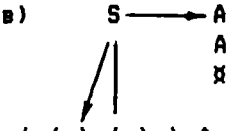
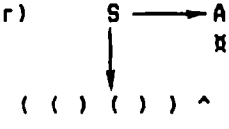
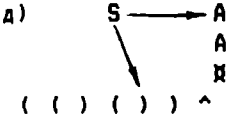
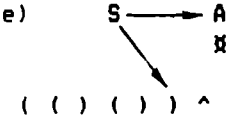
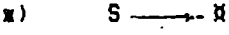
Если ВЫХОД имеет два состояния ДОПУСТИТЬ и ОТВЕРГНУТЬ, то МП-автомат называется МП-распознавателем. Для обозначения рассмотренных действий используются сокращения: ВЫТОЛК, ВТОЛК(А), СОСТ(S), ДЕРЖ, ОТВЕРГ, ДОПУСТ.

В качестве примера рассмотрим МП-автомат, осуществляющий разбор скобок в арифметических выражениях, который определяется следующим образом:

1) входное множество {(,),^ };
2) множество магазинных символов { А,Ж };
3) множество состояний { S }, где S - начальное состояние;
4) переходы.

Информация	Переходы
(,А,S	ВТОЛК(А), СОСТОЯНИЕ (S), СДВИГ;
(,Ж,S	ВТОЛК(А),СОСТОЯНИЕ (S);
),А,S	ВЫТОЛК(А),СОСТОЯНИЕ (S),СДВИГ;
),Ж,S	ОТВЕГНУТЬ;
^,Ж,S	ОТВЕГНУТЬ;
^,Ж,S	ДОПУСТИТЬ.

5. Начальное состояние магазина Ж

Структурное представление МП-автомата	Линейное представление МП-автомата		
а)	Магазин	Состояние	Остаток вх.цепочки
	Ж	[S]	(( )( ))^
б) 	ЖA	[S]	( )( ))^
в) 	ЖAA	[S]	)( ))^
г) 	ЖA	[S]	( ))^
д) 	ЖAA	[S]	))^
е) 	ЖA	[S]	)^
ж) 	Ж	[S] допустить	^

На практике очень часто встречаются автоматы с одним состоянием. Оно не влияет на работу автомата, и его можно опустить. В этом случае устройство управления можно представить в виде решающей таблицы:

Алфавит магазина	Входной алфавит (	)	^
A	ВТОЛК(A) СДВИГ	ВЫТОЛК(A) СДВИГ	ОТВЕРГНУТЬ
Z	ВТОЛК(A) СДВИГ	ОТВЕРНУТЬ	ДОПУСТИТЬ

В дальнейшем о подобных таблицах будем говорить как стандартном представлении МП-автомата с одним состояниек

Рассмотрим другой пример МП-автомата, разбирающег цепочки множества:

$$\{\, 0^n 1^n \quad n > 0 \,\}$$

Определение МП-автомата:

1) Входное множество {0,1,^ };
2) множество магазинных символов {Z,A};
3) множество состояний {S1,S2}.S1-начальное состояние;
4) переходы:
- при появлении 0 в магазин вталкивается символ Z, МП-автомат находится в состоянии S1-фаза 1;
- при появлении 1 из магазина выталкивается символ Z. Автомат переходит в состояние S2-фаза2,

Пример разбора.
Цепочка правильная

Магазин	Состояние	Остаток вх.цепочки
1 A	[S1]	000111^
2 AZ	[S1]	00111^
3 AZZ	[S1]	0111^

```
4 ¤ZZZ        [S1]            111^
5 ¤ZZ         [S2]             11^
6 ¤Z          [S2]              1^
7 ¤           [S2]               ^
8  допустить
Цепочка неправильная

Магазин     Состояние       Остаток
                            вх.цепочки
1 ¤           [S1]            0001011^
2 ¤Z          [S1]             001011^
3 ¤ZZ         [S1]              01011^
4 ¤ZZZ        [S1]               1011^
5 ¤ZZ         [S2]                011^
6 отвергнуть
```

Здесь различаются две фазы работы МП-автомата. Это фаза вталкивания символа Z в магазин , соответствующая состоянию [S1], и фаза выталкивания - [S2] . Для каждой фазы строятся свои решающие таблицы, которые имеют вид:

Фаза 1 [S1]

	0	1	^
Z	сост.(S1) втолк.(Z) сдвиг	сост.(S2) втолк.	отвергнуть
¤	сост.(S) втолк.(Z) сдвиг	отвергнуть	отвергнуть

Фаза 2 [S2]

	0	1	^
Z	отвергнуть	сост.(S2) вытолкнуть сдвиг	отвергнуть
¤	отвергнуть	отвергнуть	допустить

6.3 Расширение операций над магазином

МП-автоматы, у которых операции над магазином производятся только над одним верхним символом типа ВЫТОЛКНУТЬ, ВТОЛКНУТЬ, называются примитивными. Однако при реализации эти операции являются слишком мелкими, и их укрупняют. Более того,в анализ состояния магазина попадает не один верхний символ, а несколько. Все типы МП-автомата можно всегда свести к модели МП-автомата, обрабатывающего только один верхний символ автомата. Введем расширенную операцию ЗАМЕНИТЬ над магазином. Эта операция заключается в выталкивании верхнего символа магазина и выполнении затем нескольких вталкиваний. Операция ЗАМЕНИТЬ(ABC) осуществляет следующие действия:

- выталкивается верхний символ магазина;
- вталкивается в магазин последовательность вида(ABC)

Проиллюстрируем эту операцию на примере разбора множества

$$\{0^n 1^n \quad n>0\}$$

Определение автомата имеет вид:

1) входной алфавит { 0,1,^};
2) алфавит магазинных символов{X,Z,H};
3) используется одно состояние, которое опускаем;
4) управляющее устройство

	0	1	^
X	ЗАМЕНИТЬ(XZ) СДВИГ	ВТОЛКНУТЬ (Z)	ОТВЕРГНУТЬ
Z	ОТВЕРГНУТЬ	ВЫТОЛКНУТЬ СДВИГ	ОТВЕРГНУТЬ
H	ОТВЕРГНУТЬ	ОТВЕРГНУТЬ	ДОПУСТИТЬ

5) начальное состояние магазина X.

Рассмотрим , как осуществляется разбор множества цепочек.

N сост.	Состояние магазина	Остаток цепочки
1	#X	000111^
2	#ZX	00111^
3	#ZZX	0111^
4	#ZZZX	111^
5	#ZZZ	111^
6	#ZZ	11^
7	#Z	1^
8	#	^
9	ДОПУСТИТЬ	

Здесь фаза 1(анализ нулей) характеризуется наличием в вершине магазина символа X. При смене текущего входного символа с 0 на 1 происходит выталкивание символа X без смены текущего символа (п.4,п.5), а далее разбор осуществляется обычным образом.

6.4 Перевод входных цепочек

Дополним операции над состоянием, магазином и входом операцией над выходом

ВЫДАТЬ (ABC)

На выход МП-автомата выдается цепочка ABC. Использование операции ВЫДАТЬ проиллюстрируем на примере. Пусть на вход МП-автомата поступают цепочки из произвольно расположенных 0 и 1. Требуется на выходе построить цепочку вида 1 0 . Организуем работу автомата следующим образом.Если во входной последовательности встречается 1, то она перепускается на выход. Все нули запоминаются в магазине. По прочтении концевого маркера нули выталкиваются на выход.Здесь реализация МП-автомата требует всего одного состояния. Решающая таблица имеет вид:

	0	1	^
0	ВТОЛК(0) СДВИГ	ВЫДАТЬ(1) СДВИГ	ВТОЛК ВЫДАТЬ(0)
#	ВТОЛК(0) СДВИГ	ВЫДАТЬ(1) СДВИГ	ДОПУСТИТЬ

Проведем разбор входной цепочки:

N	Состояние магазина	Остаток вх. цепочки	Выходная цепочка
1	¤	01011^	
2	¤0	1011^	
3	¤0	011^	
	ВЫДАТЬ(1)		
4	¤00	11^	1
	ВЫДАТЬ(1)		
5	¤00	1^	11
	ВЫДАТЬ(1)		
6	¤00		111
	ВЫДАТЬ(0)		
7	¤0		1110
	ВЫДАТЬ(0)		
8			11100
	ДОПУСТИТЬ		

В МП-автоматах следует проводить анализ на предм выявления зацикливаний. Примером цикла могут служи следующие фрагменты решающей таблицы и разбор входн цепочки этим автоматом.

	0
A	ВТОЛК(B) ! ДЕРЖ
B	ВЫТОЛК ! ДЕРЖ

N	Состояние магазина	Остаток вх.цепо
...	...	...
i	A	01
i+1	AB	01
i+2	A	01
...	...	...
j	AB	01
j+1	A	01

При использовании МП-автоматов для трансляции на быть всегда уверенным в отсутствии циклов.

7. ИДЕНТИФИКАЦИЯ

Пусть задан некоторый входной алфавит символов и на м построено каким-либо образом множество слов (цепочек) Это множество А называется распознаваемым множеством. ебуется построить распознаватель, который определял бы личие входного слова х в распознаваемом множестве. кая задача получила название проблемы идентификации, а нечный распознаватель, решающий данную задачу, часто зывают идентифицирующим автоматом.

7.1. Построение идентифицирующего автомата методом префиксов

Это построение основано на введении для каждой цепочки, которая может выступать префиксом в другой цепочке, отдельного состояния. Рассмотрим метод на имере.

Пусть задан входной алфавит {Б,Ы,Т,Л,А,К,О,С,У,^}, е ^ -маркер конца цепочки.Распознаваемое множество имеет д {БЫТ, БЫК, БЫЛ, БАК, БАКУ, КОСА, ОСА}. Начинаем строение таблицы переходов автомата,отмечая в ней первое чальное состояние Л. Оно соответствует случаю, когда на оде пустая цепочка.

	Б	Ы	Т	Л	А	К	О	С	У	^
Л	Б					К	О			

ишем в соответствующие элементы таблицы состояния Б, К,), так как они являются начальными символами префиксов ов из А. Для новых состояний заведем строки в таблице и строим следующие состояния:

	Б	Ы	Т	Л	А	К	О	С	У	^
Л	Б					К	О			
	БЫ				БА					
						КО				
О								ОС		

Продолжая построение таблицы переходов, получим ее в окончательном виде:

	Б	Ы	Т	Л	А	К	О	С	У	^
Л	Б					К	О			
Б		БЫ			БА					
К						КО				
О								ОС		
БЫ			БЫТ	БЫЛ		БЫК				
БА						БАК				
КО								КОС		
ОС		ОСЫ								
БЫТ										"БЫТ"
БЫЛ										"БЫЛ"
БЫК										"БЫК"
БАК									БАКУ	"БАК"
КОС					КОСА					
ОСЫ										"ОСЫ"
БАКУ										"БАКУ"
КОСА										"КОСА"

Элементы,заключенные в кавычки,означают,что распознаватель нашел соответствующее слово. Пустые элементы соответствуют состоянию ошибки.

Отметим,что представление идентифицирующего автомата матрицей переходов ввиду ее разреженности требует достаточно больших затрат памяти. Особенно это свойство будет проявляться, когда распознаваемое множество A содержит большое число длинных слов.

Другой способ задания функции переходов — это ее представление в виде связанного списка. Реквизиты элемента такого списка имеют следующий вид:

Текущее состояние	Входные буквы	Указатели новых состояний

Построение списка переходов напоминает построение матрицы переходов и производится следующим образом:

	Б	2
1	К	3
	О	4
	ОШИБКА	

Здесь первый реквизит - это текущее состояние, второй- входные буквы, третий- новые очередные состояния. ОШИБКА - это все остальные входные буквы ,приводящие в состояние ошибки. Далее для состояний 2,3,4 строим аналогичные элементы и продолжаем процесс до завершения, после чего список переходов принимает вид

	Б	2	Б....
1	К	3	К....
	О	4	О....
	ОШИБКА		
2	Ы	5	БЫ...
	А	6	БА...
	ОШИБКА		
3	О	7	КО...
	ОШИБКА		
4	С	8	ОС...
	ОШИБКА		

```
-------------------------
 5    !   Т     !  9   БЫТ...
      !   Л     !  10  БЫЛ...
      !   К     !  11  БЫК...
      ! ОШИБКА  !
-------------------------
 6    !   К     !  12  БАК...
      ! ОШИБКА  !
-------------------------
 7    !   С     !  13  КОС...
      ! ОШИБКА  !
-------------------------
 8    !   Ы     !  14  ОСЫ...
      ! ОШИБКА  !
-------------------------
 9    !   ^     !  "БЫТ"
      ! ОШИБКА  !
-------------------------
10    !   ^        "БЫЛ"
      ! ОШИБКА  !
-------------------------
11    !   ^     !  "БЫК"
      ! ОШИБКА  !
-------------------------
12    !   У        15  БАКУ.
      !   ^     !  "БАК"
      ! ОШИБКА  !
-------------------------
13    !   А     !  16  КОСА.
      ! ОШИБКА  !
-------------------------
14    !   ^     !  "ОСЫ"
      ! ОШИБКА  !
-------------------------
15    !   ^     !  "БАКУ"
      ! ОШИБКА  !
-------------------------
16    !   ^     !  "КОСА"
      ! ОШИБКА  !
-------------------------
```

В качестве комментария рядом с новым состоянием написан разбираемый префикс. Многоточие указывает на продолжение распознавания. Слово, заключенное в кавычки, означает,что слово идентифицировано.

Итак, представление автомата в виде списка переходов является компактным в сравнении с матричным. Однако если возникает потребность в расширении распознаваемого множества, то построение списка переходов надо выполнить заново, что является недостатком.

В реальной практике системного программирования случаи расширения не так ух редки. Более того,расширение часто приходится выполнять в процессе распознавания. В этом случае необходимо иметь возможность расширять список переходов.

Выделим в списке переходов отдельный элемент, например:

	Б	2	начало 2-го списка
1	К	3	
	О	4	
	ОШИБКА		

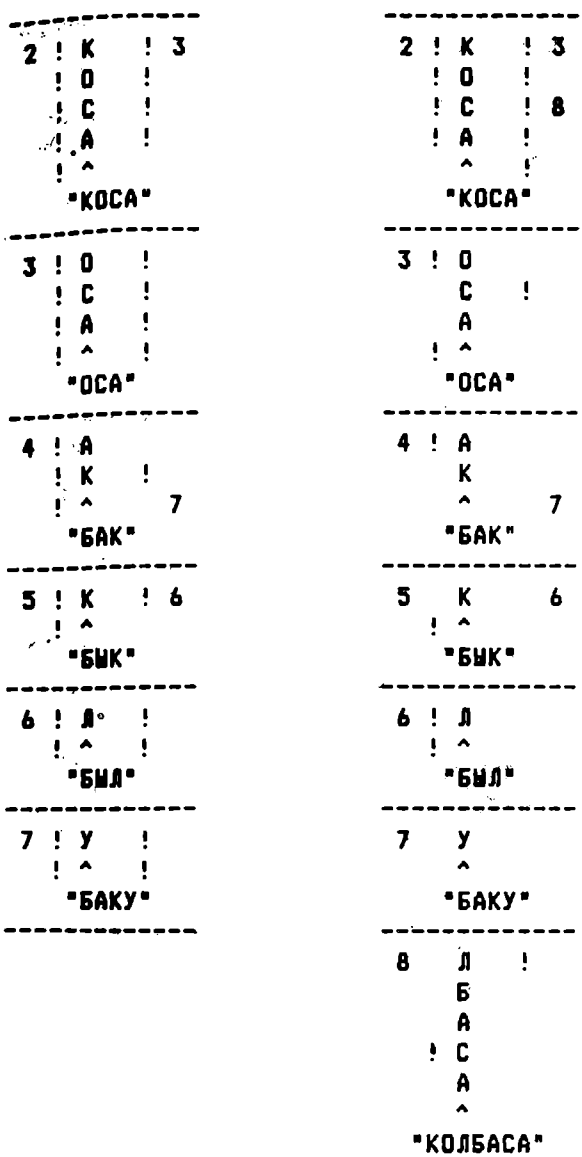
1-й список

1-й список это следующие альтернативные буквы для проверки, если проверка на равенство с данным символом отрицательна. Список задается явно в последовательных ячейках;

2-й список это указатель на следующее состояние перехода, если входной символ совпал проверяемым. Список задается неявно в виде ссылочных указателей.

Поменяем списки способом представления, после чего автоматное представление будет иметь вид(список до расширения):

До расширения После расширения



Слева представлен идентифицирующий автомат, в котором произведено расширение распознаваемого множества словом

"КОЛБАСА". Для этого в первичный автомат в состояние 2 для строки С вставляется указатель на новое состояние 8. Само состояние приписывается к списку переходов снизу.

7.2. Неавтоматное решение задачи идентификации

Существуют методы идентификации, отличные от автоматных. Идея заключается в том, что распознаваемое множество хранится в виде таблицы или связанного списка, а входное слово сравнивается с элементами такой структуры хранения по типу справочника. Расширение распознаваемого множества в этом случае никаких затруднений не вызывает.

7.2.1. Метод линейного поиска

Этот метод часто называют линейным поиском, и его суть заключается в том, что входное слово последовательно сравнивается с элементами распознаваемого множества. Хранение множества допускается либо в виде последовательного массива, либо в виде связанного списка.

7.2.2. Метод индексов

Метод индексов иначе называют прямым доступом. Пусть задано распознаваемое множество, состоящее из N элементов, и какая-либо структура хранения (таблица, связанный список). Нумерация последовательных элементов этой структуры образует ее адресное пространство. Если вложить элементы распознаваемого множества в структуру хранения, то каждый из них получит свой адрес или, иначе, индекс.

В рассматриваемом методе между элементами множества и их индексами устанавливается некоторая функциональная связь вида

индекс = F(распознаваемый элемент)

Ее часто называют формулой рандомизации. Тогда доступ к соответствующему адресу может быть осуществлен путем ее вычисления. Например, пусть распознаваемое множество представляет картотеку сотрудников со следующими

реквизитами отдельной карточки:

Табельный номер ! Ф И О ! прочие данные

Тогда формулу рандомизации можно реализовать простейшим образом:

индекс = табельный номер

Однако на практике такие простейшие формулы применяются редко ввиду неэкономного расхода памяти.

В системном программировании при обработке текстовой информации ключевые поля явно не задаются, и приходится устанавливать косвенные индексные зависимости. Например, в языке программирования типа Бейсик идентификаторы задаются в виде <буква>!<буква><цифра>, где <буква> представляет все буквы латинского алфавита, а <цифра> - цифры от 0 до 9. Число букв в алфавите 26, цифр 9, следовательно, всего может быть не более 286 идентификаторов.

Рассмотрим задачу идентификации, когда в качестве распознаваемого множества выступает множество всех идентификаторов языка Бейсик. В качестве структуры хранения выберем таблицу и ее заполнение выполним из следующих соображений. Перенумеруем буквы алфавита (A - 1, B - 2,..., Z - 26), сопоставим с каждым идентификатором индекс следующим образом:

индекс = для односимвольных идентификаторов;
индекс = i + 26 (цифра + 1)для двухсимвольных.

Здесь i - номер буквы алфавита. Тогда индекс для идентификатора A равен 1, для Z9 - 226.

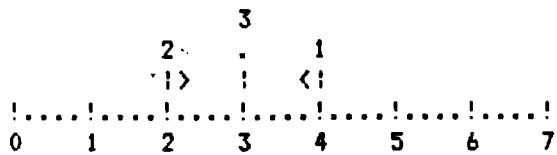
Теперь идентификация осуществляется следующим образом. Входное слово перекодируется в индекс и далее осуществляется сравнение входного слова с элементом таблицы, расположенным по полученному адресу.

Метод индексов требует наличия двух условий:
- фиксированного распознаваемого множества и однозначного его отображения на структуру хранения;
- простоты вычисления формулы рандомизации.

Кроме того, расширение множества является проблематичным.

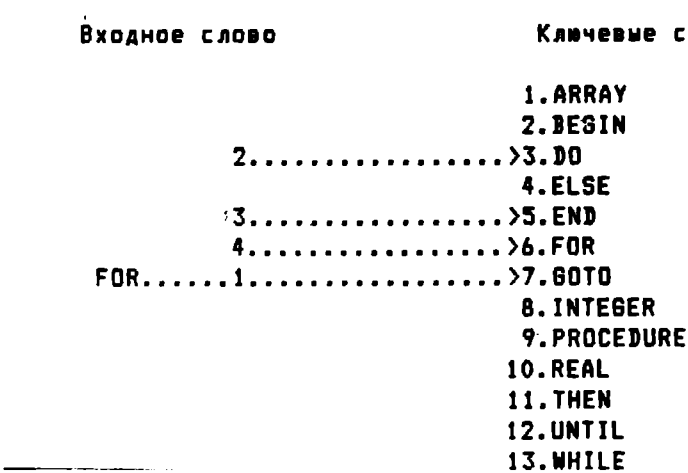
7.2.3. Идентификация по упорядоченному списку

Идентификация существенно сокращается, если распознаваемое множество упорядочено (отсортировано) по какому-либо из признаков. В этом случае возможно применение различных видов поиска, одним из эффективных и простых среди которых является двоичный (логарифмический). Например, пусть распознаваемое множество представляет записи с ключами и упорядочено в порядке их возрастания. Тогда бинарный поиск заключается в последовательном делении отрезка чисел пополам и сужении интервала неопределенности.



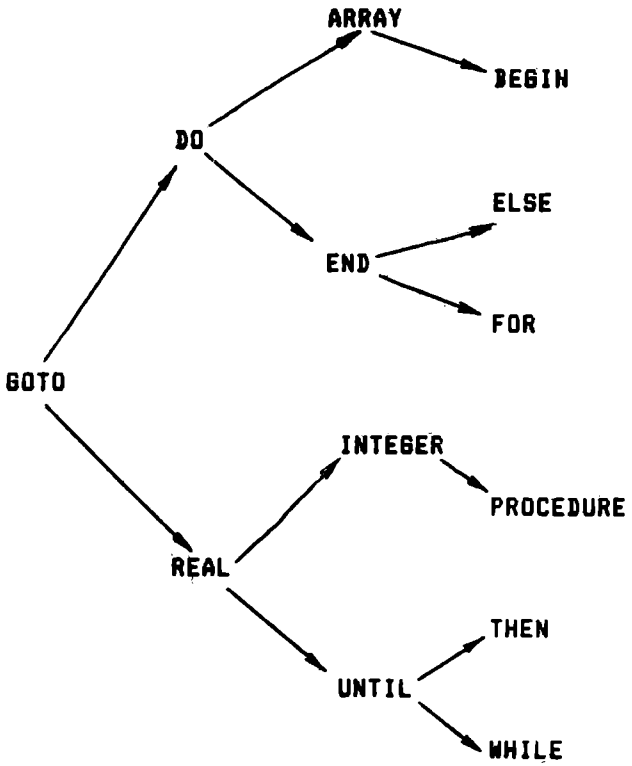
При мощности множества в N элементов число шагов на поиск составляет величину порядка log N.

Пусть распознаваемое множество - это ключевые слова языка программирования. Упорядочим их в лексикографической порядке и разместим в таблице. Здесь бинарный поиск будет проводиться следующим образом:



Если число элементов множества не кратно 2^n, то возникает вопрос, что считать серединой отрезка. Кроме того, расширение множества требует реорганизации таблицы.

Для поиска можно использовать связанный список в виде двоичного дерева. Его корень соответствует середине отрезка, а ссылки указывают на середины "половин" и т.д.



Дерево можно расширять, добавляя новые элементы со стороны концевых узлов. Но в этом случае быстродействие поиска может существенно ухудшиться. В этой связи любое существенное расширение требует реорганизации дерева.

7.2.4. Метод расстановки

Этот способ является в некотором смысле комбинацией

метода индексов и методов поиска по упорядоченным множествам и заключается в следующем. Все распознаваемое множество разбивается на отдельные подмножества,каждому из которых приписывается свой индекс. Подмножества могут размещаться в списках. Для всех списков составляется таблица указателей, которая индексируется. Идентификация осуществляется в два приема:

- вычисляется по входному слову индекс и отыскивается соответствующий список;
- выполняется поиск по списку, и, если слово отсутствует, оно добавляется к списку:

```
0 !.!------->!FOR !.!----->!BEGIN!.!----->!THEN!.!
1 !.!------->!GOTO!.!
2 !.!------->!INTEGER!.!
3 !.!------->!WHILE!.!---->!REAL!.!
4
5 !.!------->!DO!.!------->!ARRAU!.!----->!END!.!
7 !.!------->!PROCEDURE!.!
```

В методе расстановки всегда приходится находить компромисс между длиной списков и размером индексирующей таблицы. Обычно стремятся иметь списки одинаковой длины.

7.3. Обнаружение слов с нечеткими границами

Границы слов, подлежащих идентификации, могут быть как четкими (например, в естественных языках слова в предложении разделяются пробелами), так и нечеткими. Например, следующую запись можно воспринимать как оператор присваивания в языке фортран и как оператор цикла в языке программирования Бейсик:

```
10 FORK = STOP
```

В случае Бейсика запись при вставке пробелов имеет вид:

```
10 FOR K   S TO P
```

Задачу идентификации в случае нечетких границ слов

часто называют задачей обнаружения префиксов. Она достаточно просто решается при следующем ограничении: в распознаваемом множестве ни один элемент не является префиксом для другого. В этом случае можно строить идентифицирующий автомат и в нем предусмотреть выход не по концевому маркеру, а по обнаружению префикса. В случае неавтоматного решения задачи следует поступать следующим образом:

1) завести некоторую текстовую переменную CHAR и присвоить ей значение пустая знаковая строка;
2) из входной строки считать текущий символ и присоединить его к CHAR;
3) провести идентификацию CHAR с распознаваемым множеством. При неудаче поторить с п.2.

ПОСЛЕСЛОВИЕ

В настоящей работе автор постарался осветить многогранность проблемы распознавания и роль автоматных методов в ее решении. В основном весь материал построен на широко известных фундаментальных работах, перешедших в разряд классических, которые приведены в списке литературы, и только по части использования обратной польской записи для построения системы переходов предложил свои соображения. Саму работу надо рассматривать не как справочное руководство, а только как введение в предметную область распознавателей.

Автор будет признателен всем читателям за замечания и предложения по улучшению работы.

СПИСОК ЛИТЕРАТУРЫ

1. Лебедев В.Н. Введение в системы программирования.-М.: Статистика, 1975.-312 с.

2. Донован Дж. Системное программирование.-М.: Мир, 1975.-540 с.

3.Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции.-М.: Мир, 1978, Т.1.- 612 с.

4.Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции,-М.: Мир, 1978, Т.2. -490 с.

5.Грис Д. Конструирование компиляторов для цифровых вычислительных машин.-М.: Мир, 1975.-545 с.

6.Льюис Ф., Розенкранц Д., Стирнз Р. Теоретические основы проектирования компиляторов.-М.: Мир, 1979.-645 с.

7.Вишняков М.М. Системное программирование. Ассемблеры и макрогенераторы: Учебное пособие.-Таганрог: ТРТИ, 1985.-64 с.

8.Перцев А.В. Системное программирование: Учебное пособие.-Таганрог: ТРТИ, 1983.-44 с.

Темплан 1991 г поз. 1495

Вишняков Юрий Муссович

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ КОНЕЧНЫЕ РАСПОЗНАВАТЕЛИ

Учебное пособие

Ответственный за выпуск Вишняков Ю. М.

Редактор Маныч Э. И.

Корректор Проценко И. А.

Художник обложки Рябенко В. И.

Подписано к печати 4.12.1990 г.
Формат 60×84 1/16. Бумага оберточная.
Офсетная печать. Усл. п. л. — 4,8. Уч.-изд. л. — 4,0.
Заказ № 7. Тир. 500 экз.
Цена 20 к.

Редакционно-издательский отдел Таганрогского радиотехнического института им. В. Д. КАЛМЫКОВА
Таганрог, 15, Некрасовский переулок, 44
Типография Таганрогского радиотехнического института им. В. Д. Калмыкова
Таганрог, 15, Энгельса, 1