2. ЛЕКСИЧЕСКИЙ АНАЛИЗ

Этап лексической обработки текста исходной программы выделяется в отдельный этап работы компилятора, как с методическими целями, так и с целью сокращения общего времени компиляции программы. Последнее достигается за счет того, что исходная программа, представленная на входе компилятора в виде непрерывной последовательности символов, на этапе лексической обработки преобразуется к некоторому стандартному виду, что облегчает дальнейший анализ. При этом используются специализированные алгоритмы преобразования, теория и практика построения которых проработана достаточно глубоко.

2.1. Задачи и функционирование блока лексического анализа

Под *лексическим анализом* будем понимать процесс предварительной обработки исходной программы, на котором основные лексические единицы программы — *лексемы*: ключевые (служебные) слова, идентификаторы, метки, константы приводятся к единому формату и заменяются условными кодами или ссылками на соответствующие таблицы, а комментарии исключаются из текста программы.

Выходами лексического анализа являются поток образов лексем-дескрипторов и таблицы, в последних хранятся значения выделенных в программе лексем.

Дескриптор — это пара вида: (<тип лексемы>, <указатель>),

где <тип лексемы> — это, как правило, числовой код класса лексемы, который означает, что лексема принадлежит одному из конечного множества классов слов, выделенных в языке программирования;

<указатель> — это может быть либо начальный адрес области основной памяти, в которой хранится адрес этой лексемы, либо число, адресующее элемент таблицы, в которой хранится значение этой лексемы.

Количество классов лексем (т.е. различных видов слов) в языках программирования может быть различным. Наиболее распространенными классами являются:

- ❑ идентификаторы;
- ❑ служебные (ключевые) слова;
- ❑ разделители;
- ❑ константы.

Могут вводиться и другие классы. Это обусловлено в первую очередь той ролью, которую играют различные виды слов при написании исходной программы и, соответственно, при переводе ее в машинную программу. При этом наиболее предпочтительным является разбиение всего множества слов, допускаемого в языке программирования, на такие классы, которые бы не пересекались между собой. В этом случае лексический анализ можно выполнить более эффективно. В общем случае все выделяемые классы являются либо конечными (ключевые слова, разделители и др.) — классы фиксированных для данного языка программирования слов, либо бесконечными или очень большими (идентификаторы, константы, метки) — классы переменных для данного языка программирования слов.

С этих позиций коды образов лексем (дескрипторов) из конечных классов всегда одни и те же в различных программах для данного компилятора. Коды же образов лексем из бесконечных классов различны для разных программ и формируются каждый раз на этапе лексического анализа.

В ходе лексического анализа значения лексем из бесконечных классов помещаются в таблицы соответствующих классов. Конечность таблиц объясняет ограничения, существующие в языках программирования на длины (и соответственно число) используемых в программе идентификаторов и констант. Необходимо отметить, что числовые константы перед помещением их в таблицу могут переводиться из внешнего символьного во внутреннее машинное представление. Содержимое таблиц, в особенности таблицы идентификаторов, в дальнейшем пополняется на этапе семантического анализа исходной программы и используется на этапе генерации объектной программы.

Рассмотрим основные идеи, которые лежат в основе построения лексического анализатора, проблемы, возникающие при его работе, и подходы к их устранению.

Первоначально в тексте исходной программы лексический анализатор выделяет последовательность символов, которая по его предположению должна быть словом в программе, т.е. лексемой. Может выделяться не вся последовательность, а только один символ, который считается началом лексемы. Это наиболее ответственная часть работы лексического анализатора. Более просто она реализуется для тех языков программирования, в которых слова в программе отделяются друг от друга специальными разделителями (например, пробелами), либо запрещено использование служебных слов в качестве переменных, либо классы лексем опознаются по вхождению первого (последнего) символа лексемы.

После этого (или параллельно с этим) проводится идентификация лексемы. Она заключается в сборке лексемы из символов, начиная с выделенного на предыдущем этапе, и проверки правильности записи лексемы данного класса.

Идентификация лексемы из конечного класса выполняется путем сравнения ее с эталонным значением. Основная проблема здесь — минимизация времени поиска эталона. В общем случае может понадобиться полный перебор слов данного класса, в особенности для случая, когда выделенное для опознания слово содержит ошибку. Уменьшить время поиска можно, используя различные методы ускоренного поиска:

- ❑ метод линейного списка;
- ❑ метод упорядоченного списка;
- ❑ метод расстановки и другие [6].

Для идентификации лексем из бесконечных (очень больших) классов используются специальные методы сборки лексем с одновременной проверкой правильности написания лексемы. При построении этих алгоритмов широко применяется формальный математический аппарат — теория регулярных языков, грамматик и конечных распознавателей. Практический аспект этой теории для целей лексического анализа будет рассмотрен в п. 2.2.

При успешной идентификации значение лексемы из бесконечного класса помещается в таблицу идентификации лексем данного класса. При этом необходимо предварительно проверить: не хранится ли там уже значение данной лексемы, т.е. необходимо проводить просмотр элементов таблицы. Если ее там нет, то значение помещается в таблицу. При этом таблица

должна допускать расширение. Опять же для уменьшения времени доступа к элементам таблицы она должна быть специальным образом организована, при этом должны использоваться специальные методы ускоренного поиска элементов.

После проведения успешной идентификации лексемы формируется ее образ — дескриптор, он помещается в выходной поток лексического анализатора. В случае неуспешной идентификации формируются сообщения об ошибках в написании слов программы.

В ходе лексического анализа могут выполняться и другие виды лексического контроля, в частности, проверяется парность скобок и других парных символов.

Выходной поток с лексического анализатора в дальнейшем поступает на вход синтаксического анализатора. Имеется две возможности их связи:

- раздельная связь, при которой выход лексического анализатора формируется полностью и затем передается синтаксическому анализатору;
- нераздельная связь, когда синтаксическому анализатору требуется очередной образ лексемы, он вызывает лексический анализатор, который генерирует требуемый дескриптор и возвращает управление синтаксическому анализатору.

Второй вариант характерен для однопроходных трансляторов. Таким образом, процесс лексического анализа может быть достаточно простым, но в смысле времени компиляции оказывается довольно долгим. Как указывается в [9], больше половины времени, затрачиваемого компилятором на компиляцию, приходится на этап лексического анализа. На рис. 2.1. для приведенной там же программы показан возможный вариант содержимого таблиц и выхода лексического анализатора. При этом в качестве кодов типов лексем используются числа:

10 — ключевое слово;

20 — разделитель;

30 — идентификатор;

40 — константа.

Вход лексического анализатора:

```
PROGRAM PRIMER;
  VAR X,Y,Z : REAL;
  BEGIN
        X:=5;
        Y:=5;
        Z:=X+Y;
  END;
```

Внутренние таблицы лексического анализатора:

Таблица ключевых слов

№ п/п	Ключевое слово
1	Program
2	Begin
3	End
4	For
5	Real
6	Var
...	...

Таблица разделителей

№ п/п	Разделитель
1	;
2	,
3	+
4	—
5	/
6	*
7	:
8	=
9	.
...	...

Выход лексического анализатора:

Таблица идентификаторов

№ п/п	Идентификатор
1	Primer
2	X
3	Y
4	Z

Таблица констант

№ п/п	Знач. константы
1	5

Дескрипторный текст:

(10,1) (30,1) (20,1)
(10,6) (30,2) (20,2) (30,3) (20,2) (30,4) (20,7) (10,5) (20,1)
(10,2)
(30,2) (20,7) (20,8) (40,1) (20,1)
(30,3) (20,7) (20,8) (40,1) (20,1)
(30,4) (20,7) (20,8) (30,2) (20,3) (30,3) (20,1)
(10,3) (20,9).

Рис. 2.1. Пример содержимого таблиц и выхода лексического анализатора

2.2. Регулярные языки, конечные автоматы и лексический анализ

Разработка лексического анализатора выполняется достаточно просто, если при этом воспользоваться хорошо разработанным математическим аппаратом. Таким аппаратом является теория регулярных языков и конечных автоматов. В рамках этой теории классы однотипных лексем (идентификаторы, константы и т. д.) рассматриваются как формальные языки (язык идентификаторов, язык констант и т.д.), множество предложений которых описывается с помощью соответствующей порождающей грамматики. При этом языки эти настолько просты, что они порождаются простейшей из грамматик — регулярной грамматикой (грамматика класса 3 в иерархии Хомского). Построенная регулярная грамматика является источником, по которому в дальнейшем конструируется вычислительное устройство, реализующее функцию распознавания предложений языка, порождаемого данной грамматикой. Для регулярных языков таким устройством является конечный автомат. Ниже будут рассмотрены основные понятия теории регулярных языков и конечных автоматов и ее практическое использование при разработке алгоритмов идентификации лексем.

Определение 2.2.1. Порождающая грамматика G = <N, T, P, S>, правила которой имеют вид: A → aB или C → b, где A, B, C ∈ N; a, b ∈ T называется регулярной (автоматной).

Язык L(G), порождаемый автоматной грамматикой, называется автоматным (регулярным) языком или языком с конечным числом состояний.

Пример 2.2.1. Класс идентификаторов, если идентификатором является последовательность, состоящая из букв и цифр, и первым символом идентификатора может быть только буква, описывается следующей порождающей регулярной грамматикой G = <N, T, P, S>, в которой

N = {I, K}, T = {б, ц}, S = {I},
P = { 1. I → б
2. I → бK
3. K → бK
4. K → цK
5. K → б
6. K → ц }

Здесь б, ц — обобщенные терминальные символы для обозначения букв и цифр соответственно.

Процесс порождения идентификатора «ббцбц» описывается следующей последовательностью подстановок:

$$I \overset{2}{\Rightarrow} бК \overset{3}{\Rightarrow} ббК \overset{4}{\Rightarrow} ббцК \overset{3}{\Rightarrow} ббцбК \overset{6}{\Rightarrow} ббцбц$$

Необходимо отметить, что данная грамматика не единственная, которая описывает язык идентификаторов.

Однако основной задачей ЛА является не порождение лексических единиц, а их распознавание. Математической моделью процесса распознавания регулярного языка является вычислительное устройство, которое называется *конечным автоматом* (КА). Термин «конечный» подчеркивает то, что вычислительное устройство имеет фиксированный и конечный объем памяти и обрабатывает последовательность входных символов, принадлежащих некоторому конечному множеству. Существуют различные типы КА, если функцией выхода КА (результатом работы) является лишь указание на то, допустима или нет входная последовательность символов, такой КА называют *конечным распознавателем*. Именно этот тип КА в дальнейшем мы и будем рассматривать.

Определение 2.2.2. Конечным автоматом называется следующая пятерка:

$A = \langle V, Q, \delta, q_0, F \rangle$, где $V = \{a_1, a_2, ..., a_m\}$ — входной алфавит (конечное множество символов);

$Q = \{q_0, q_1, ..., q_{n-1}\}$ — алфавит состояний (конечное множество символов);

$\delta: Q \times V \to Q$ — функция переходов;

$q_0 \in Q$ — начальное состояние конечного автомата;

$F \subseteq Q$ — множество заключительных состояний.

На содержательном уровне функционирование КА можно представить следующим образом.

Имеется бесконечная лента, разбитая на ячейки, в каждой из которых может находиться один символ из V. На ленте записана цепочка $\alpha \in V^*$. Ячейки слева и справа от цепочки не заполнены. Имеется конечное устройство управления (УУ) с читающей головкой, которое может последовательно считывать символы с ленты, передвигаясь вдоль ленты слева направо. При этом УУ может находиться в каком-либо одном состоянии из Q. Начинает свою работу УУ всегда в начальном состоянии $q_0 \in Q$, а завершает в одном из заключительных состояний $F \subseteq Q$. Каждый раз, переходя к новой ячейке на ленте, УУ переходит

в новое состояние в соответствии с функцией δ. Схематически конструкция КА показана на рис.2.2.

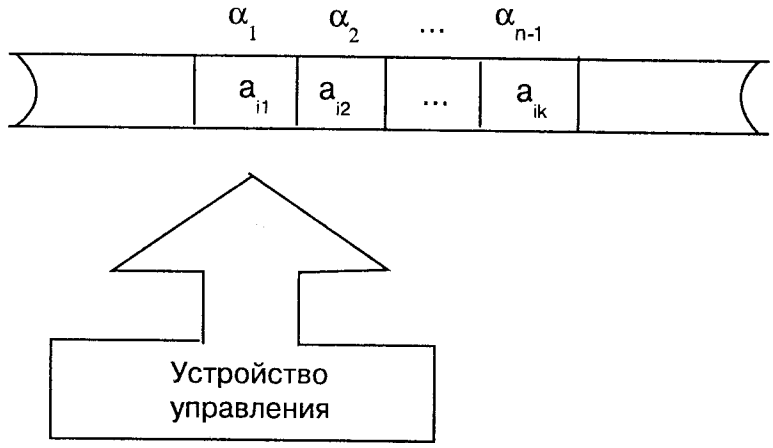


Рис. 2.2. Схема конечного автомата

Отображение δ (функцию переходов КА) можно представить различными способами. Основные из них:

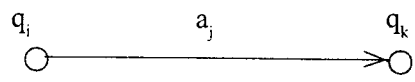
- совокупность команд;
- диаграмма состояний;
- матрица переходов.

Команда конечного автомата записывается следующим образом:

$$(q_i, a_j) \rightarrow q_k, \text{ где: } q_i, q_k \in Q; a_j \in V.$$

Данная команда обозначает, что КА находится в состоянии q_i, читает с ленты символ a_j и переходит в состояние q_k.

Графически команда представляется в виде дуги графа, идущей из вершины q_i в вершину q_k и помеченной символом a_j входного алфавита:



Графическое представление всего отображения δ называют *диаграммой состояний* конечного автомата. Диаграмма состояний в этом случае — это ориентированный граф, вершинам которого поставлены в соответствие символы из множества состояний Q, а дугам — команды отображения δ.

Если КА оказывается в ситуации (q_i, a_j), которая не является левой частью какой-либо команды, то он останавливается. Если же УУ считает все символы цепочки α, записанной на ленте, и при этом перейдет в заключительное состояние $q_r \in F$, то говорят, что цепочка α *допускается конечным автоматом.*

Матрица переходов КА строится следующим образом: столбцы матрицы соответствуют символам из входного алфавита, строки — символам из алфавита состояний, а элементы матрицы соответствуют состояниям, в которые переходит КА для данной комбинации входного символа и символа состояния.

Как было указано ранее, КА является хорошей математической моделью для представления алгоритмов распознавания лексем в ЛА, в особенности для лексем из бесконечных классов. При этом источником, по которому строится КА, является регулярная грамматика.

Пусть задана регулярная грамматика G = <N, T, P, S>, правила которой имеют вид : $A_i \rightarrow a_j A_k$ или $A_i \rightarrow a_j$, где $A_i, A_k \in N$ и $a_j \in T$.

Тогда конечный автомат A = <V, Q, δ, q_0, F>, допускающий тот же самый язык, что порождает регулярная грамматика G, строится следующим образом [7]:

1) V = T;
2) $Q = N \cup \{Z\}$, $Z \notin N$ и $Z \notin T$, Z — заключительное состояние КА;
3) $q_0 = \{S\}$;
4) F = {Z};
5) Отображение δ строится в виде:
   - ❑ каждому правилу подстановки в грамматике G вида $A_i \rightarrow a_j A_k$ ставится в соответствие команда $(A_i, a_j) \rightarrow A_k$;
   - ❑ каждому правилу подстановки вида $A_i \rightarrow a_j$ ставится в соответствие команда $(A_i, a_j) \rightarrow Z$;

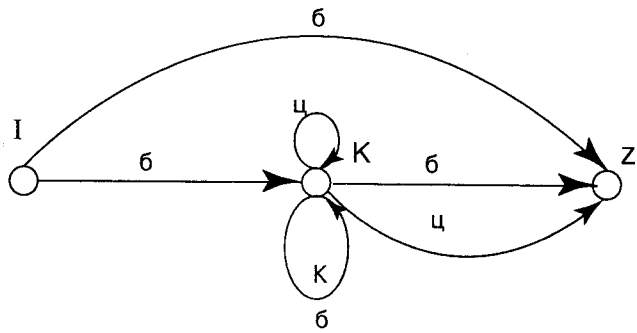
Пример 2.2.2. Построить КА для грамматики из примера 2.2.1. Имеем A = <V, Q, δ, q_0, F>, где

1) V = T = {б, ц}
2) $Q = N \cup \{Z\}$ = {I, R, Z}
3) q_0 = {S} = {I}
4) F = {Z}

5) δ: а) в виде совокупности команд:

(I, б) → Z
(I, б) → K
(K, б) → K
(K, ц) → K
(K, б) → Z
(K, ц) → Z

б) в виде диаграммы состояний:



Допустим и обратный переход, так конечному автомату A = <V, Q, δ, q_0, F> можно поставить в соответствие регулярную грамматику G = <N, T, P, S>, у которой:

1) T = V;

2) N = Q;

3) S = {q_0};

4) Множество правил подстановки P строится следующим образом: каждой команде автомата (q_i, a_j) → q_k ставится в соответствие правило подстановки $q_i \rightarrow a_j q_k$, если $q_k \in Q$, либо еще одно правило $q_i \rightarrow a_j$, если $q_k \in F$.

Различают детерминированные и недетерминированные конечные автоматы. КА называется *недетерминированным* КА (НДКА), если в диаграмме его состояний из одной вершины исходит несколько дуг с одинаковыми пометками. Например, КА из примера 2.2.2. является НДКА. Хотя НДКА располагает на первый взгляд большими возможностями, чем детерминированный КА, классы языков, которые они допускают, в действительности совпадают [1].

И по описанию НДКА всегда можно построить описание детерминированного КА (ДКА). При этом НДКА A с входным алфавитом V, множеством состояний

$$Q = \{q_0, q_1, ..., q_{n-1}\}$$

и функцией переходов δ преобразуется в ДКА A′, у которого:

1. $V' = V$;
2. $q_0' = \{q_0\}$;
3. $Q' = \{\varnothing, \{q_0\}, \{q_1\}, ..., \{q_0, ..., q_{n-1}\}\}$;
4. $\delta'(q', a) = \bigcup_{q \in q'} \{\delta(q, a)\}$

5) Заключительные состояния F′ — все подмножества Q, содержащие хотя бы одно заключительное состояние автомата A.

Пример 2.2.3. [7]. Пусть регулярный язык порождается грамматикой G = <N, T, P, S>, у которой N = {S, R}, T = {a, b}, S = {S},

$$P = \{ \begin{array}{l} S \to aS \mid aR \\ R \to bR \mid a \end{array} \}$$

Построим НДКА A = <V, Q, δ, q_0, F>. Имеем V = T = {a, b}; Q = {S, R, Z}; F = {Z}, отображение представлено на рис.2.3 в виде диаграммы состояний.

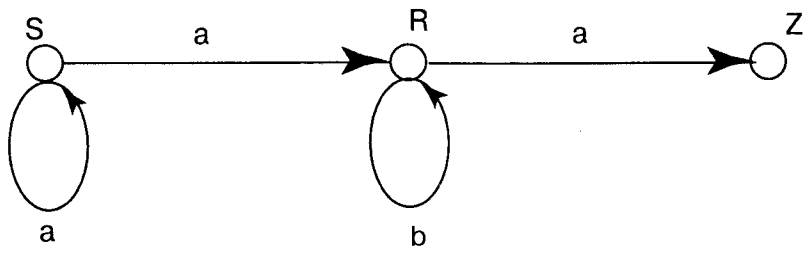


Рис. 2.3. Диаграмма состояний НДКА из примера 2.2.3.

Построим эквивалентный ему ДКА $A' = \langle V', Q', \delta', q_0', F' \rangle$. Его компонентами будут:

$V' = V = \{a, b\}$;

$Q' = \{\{S\}, \{R\}, \{S, R, Z\}\}$; остальные 4 состояния можно отбросить, так как ни в одно из них автомат попасть не может;

$q_0' = \{S\}$;

$F' = \{\{S, R, Z\}\}$;

функцию переходов δ' можно задать с помощью следующих команд:

$(\{S\}, a) \rightarrow \{S, R, Z\}$;

$(\{S, R, Z\}, a) \rightarrow \{S, R, Z\}$;

$(\{S, R, Z\}, b) \rightarrow \{R\}$;

$(\{R\}, b) \rightarrow \{R\}$;

$(\{R\}, a) \rightarrow \{S, R, Z\}$.

Представление δ' в виде диаграммы состояний показано на рис. 2.4.

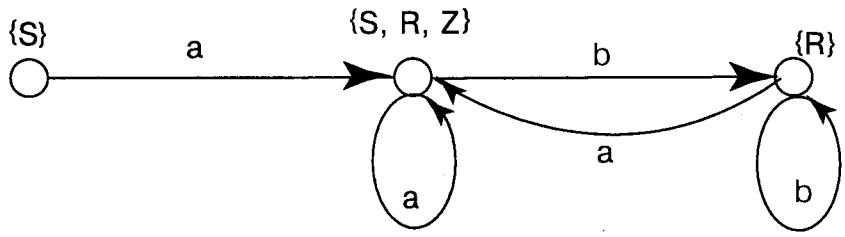


Рис. 2.4. Диаграмма состояний ДКА

Построим по описанию ДКА A' регулярную грамматику $G' = \langle N', T', P', S' \rangle$. Для этого дадим новые обозначения элементам множества Q':

$\{S\} — S'$, $\{R\} — R'$, $\{S, R, Z\} — K'$.

Имеем $N' = \{S', R', K'\}$, $T' = \{a, b\}$, $S' = \{S'\}$,

$P' = \{S' \rightarrow aK'$

$K' \rightarrow aK' \mid bR' \mid a$

$R' \rightarrow bR' \mid aK' \mid a\}$.

Для практических целей необходимо, чтобы конечный распознаватель играл более активную роль и сам определял момент окончания входной последовательности символов с выдачей сообщения о правильности или ошибочности входной цепочки. Для этих целей входная цепочка считается ограниченной справа концевым маркером ⊢ и в диаграмму состояний КА вводятся интерпретированные состояния. Будем обозначать эти состояния соответственно: Z — «допустить входную цепочку», O — «запомнена ошибка во входной цепочке», E — «отвергнуть входную цепочку». Состояния Z и E являются заключительными, и в них КА переходит при прочтении концевого маркера ⊢ соответственно после обработки правильной или ошибочной входной цепочки. Состояние O является промежуточным, в него КА переходит из любого допустимого состояния КА при обнаружении ошибки во входной цепочке и остается в нем до поступления концевого маркера ⊢ , после чего осуществляется переход в состояние E — «отвергнуть входную цепочку».

На рис. 2.5 приведена модифицированная диаграмма состояний конечного распознавателя из примера 2.2.2.

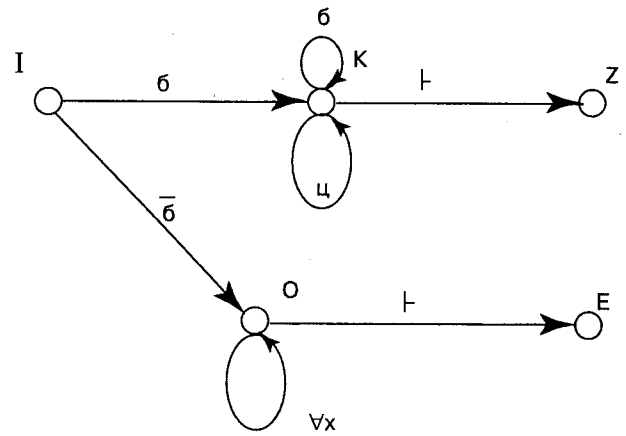


Рис. 2.5. Модифицированная диаграмма состояний КА для распознавания идентификаторов (б̄ — не буква)

В общем случае может быть предложен следующий порядок конструирования лексического анализатора [6]:

Шаг 1. Выделить во входном языке L(G) на основании описания его синтаксиса с помощью КС-грамматики G множество классов лексем L_i, $1 \le i \le k$, $k \ge 1$.

Шаг 2. Построить для каждого класса лексем L_i автоматную грамматику G_i, порождающую язык L_i, т.е. $L_i = L(G_i)$.

Шаг 3. Для каждой автоматной грамматики G_i построить диаграмму состояний D_i — неформальную модель распознавателя языка $L(G_i)$.

Шаг 4. Определить условия выхода из ЛА (переход ЛА в начальное состояние) при достижении конца произвольной лексемы из каждого класса лексем $L(G_i)$.

Шаг 5. Разбить символы входного алфавита на непересекающиеся классы.

Шаг 6. Построить матрицу переходов лексического анализатора.

Шаг 7. Выбрать формат и код образов лексем-дескрипторов вида (<тип лексем>, <адрес в таблице>).

Шаг 8. Запрограммировать семантические подпрограммы и распознаватель по склеенной диаграмме переходов.

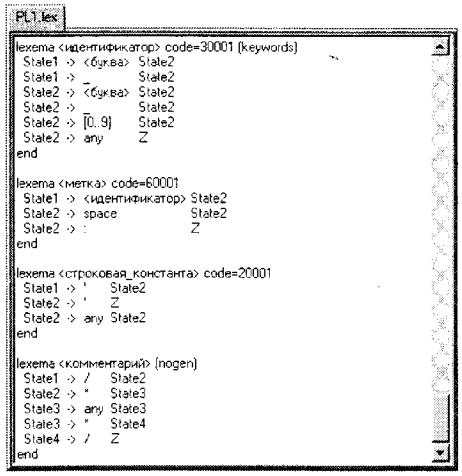
2.3. Практикум по лексическому анализу

Для закрепления знаний по теории конечных распознавателей и приобретения навыков и умений использования теории при разработке средств лексического анализа языка, необходимо проведение практикума по программированию конечных распознавателей языков и лексических анализаторов. При этом целесообразно формальное описание лексики языка программирования проводить не только с одним «карандашом», но уже на этапе разработки модели ЛА привлечь компьютер.

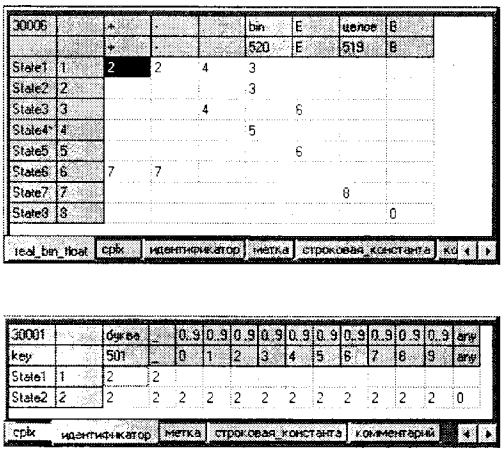
Процесс конструирования модели ЛА может быть достаточно легко автоматизирован. Для этого необходим язык, который позволял бы выполнять описание модели ЛА в терминах конечных распознавателей. В общем случае в этом языке должны присутствовать средства для описания множества входных символов ЛА, его выходных символов (дескрипторов), множества состояний и правил перехода автоматов-распознавателей лексем соответствующих классов из одних состояний в другие.

В учебном пособии предлагается язык описания лексики, нотация которого максимально приближена к математическому описанию конечных распознавателей. С помощью этого языка можно описывать как конечные распознаватели, так и модели лексических анализаторов. Интерпретатор с языка описания лексики (ЯОЛ) входит в состав разработанной авторами программной системы «КАШТАН». Разработка и отладка модели ЛА в системе осуществляется в

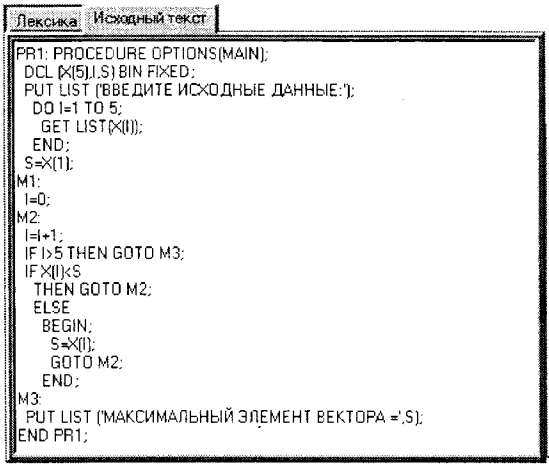
интерактивном режиме в рамках оконного интерфейса Windows. На рис.2.6 приведен пример содержимого окон при моделировании процесса лексического анализа языка ПЛ/1. Технология применения системы «КАШТАН» описана в главе 5 данного учебного пособия. Здесь же будет рассмотрен собственно язык описания лексики и правила интерпретации предложений на этом языке.



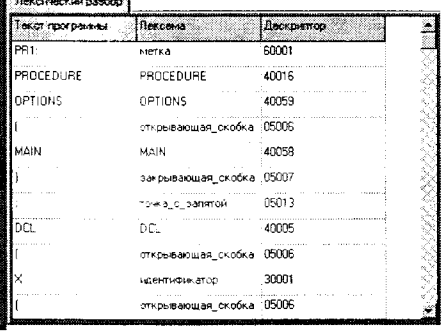
а) программа на ЯОЛ



б) управляющие таблицы ЛА



в) вход лексического анализатора



г) выход лексического анализатора

Рис. 2.6. Пример содержимого окон при моделировании лексического анализатора с помощью системы «КАШТАН»

ЯЗЫК ОПИСАНИЯ ЛЕКСИКИ

Язык описания лексики (ЯОЛ) предназначен для автоматизированной разработки и исследования (тестирования) модели лексического уровня описания языка программирования. Результатом интерпретации программы на ЯОЛ является поток образов классов лексем, выделенных в исходном языке. В отличие от реального ЛА интерпретатор с ЯОЛ не формирует выходных таблиц ЛА и поэтому коды дескрипторов различаются в нем на уровне имен классов лексем.

ЯОЛ можно использовать и для программирования собственно конечных распознавателей. В этом случае на выходе интерпретатора формируются стандартные выходы детерминированного конечного распознавателя: ДОПУСТИТЬ или ОТВЕРГНУТЬ. Данный вариант использования ЯОЛ хорошо подходит для моделирования синтаксического контроля предложений языка программирования.

Лексический уровень ЯОЛ.

В ЯОЛ выделены следующие классы лексем:

- ключевые слова;
- директивы;
- идентификаторы;
- разделители.

К *ключевым словам* относятся: **KeyWords**, **EndWords**, **Lexema**, **End**, **Space**, **Any**, **Code**, **NoGen**. Задавать их можно как с помощью прописных, так и строчных букв. *Идентификатором* является любая последовательность букв и цифр. Для идентификаторов имеют различие прописные и строчные буквы.

К *разделителям* относятся: угловые скобки < >; круглые скобки ( ); квадратные скобки **[ ]**; две точки ..; знак равно =; переход –>.

К директивам относятся: **%NOUPCASE**, **%UPCASE**.

Синтаксис ЯОЛ в форме нотации БНФ имеет вид:

<Программа на ЯОЛ> ::= <описание ЛА> | <описание ДКА>
<описание ЛА>::=<список вспомогательных лексем>
<список ключевых слов><список основных лексем>
<список вспомогательных лексем>::=<описание лексемы > |
<список вспомогательных лексем><описание лексемы > |λ
<список ключевых слов> ::=**KeyWords** <список слов> **EndWords** |λ
<список слов> ::=<ключевое слово> <<код лексемы>> |
<ключевое слово> <<код лексемы>> <список слов>

<ключевое слово>::=**идентификатор**
<код лексемы> ::=**идентификатор**
<список основных лексем>::=<описание основной лексемы > | <список основных лексем><описание основной лексемы > |λ
<описание основной лексемы> ::= **Lexema** <имя лексемы> <опции> <список команд>**End**
<описание вспомогательной лексемы> ::= **Lexema** <имя лексемы><список команд> **End**
<имя лексемы> ::=**идентификатор**
<список команд> ::=<команда> | <команда> <список команд>
<команда> ::= <состояние> -> <условие перехода> <состояние>
<условие перехода> ::= **символ** | <диапазон> | < <имя лексемы> > | **Space** | **Any**
<диапазон> ::= **[символ .. символ]**
<опции> ::=**Code** = <код лексемы> <параметр> | <параметр>
<код лексемы>::=**идентификатор**
<параметр> ::= (**KeyWords**) | (**NoGen**) | λ.

Программа на ЯОЛ представляет из себя последовательность описаний основных и вспомогательных лексем. Описание лексемы задается в виде команд *конечного детерминированного распознавателя*, при этом в одной строке записывается одна команда.

Лексема является *основной*, если после успешного ее распознавания на выходе ЛА формируется ее образ — дескрипторный код. Если дескрипторный код для лексемы не формируется,то она является *вспомогательной*. В программе на ЯОЛ должна присутствовать всегда по крайней мере одна вспомогательная и одна основная лексемы.

Признаком *основной* лексемы является наличие в ее заголовке опции **code**=<код лексемы>, здесь <код лексемы> — образ лексемы в выходном потоке — дескриптор. Лексемы могут входить в состав других лексем (как основных, так и вспомогательных). Лексема может быть: префиксом, находиться внутри или быть суффиксом другой лексемы. При описании лексем необходимо соблюдать условие: описание лексемы должно всегда предшествовать ее использованию.

Программирование моделей лексического анализа.

Правила описания и интерпретации предложений ЯОЛ рассмотрим на конкретных примерах описания лексем различных классов.

Пусть необходимо сконструировать модель ЛА для распознавания идентификаторов, в записи которых можно использовать только прописные или строчные буквы латинского алфавита. В случае поступления на вход ЛА правильного

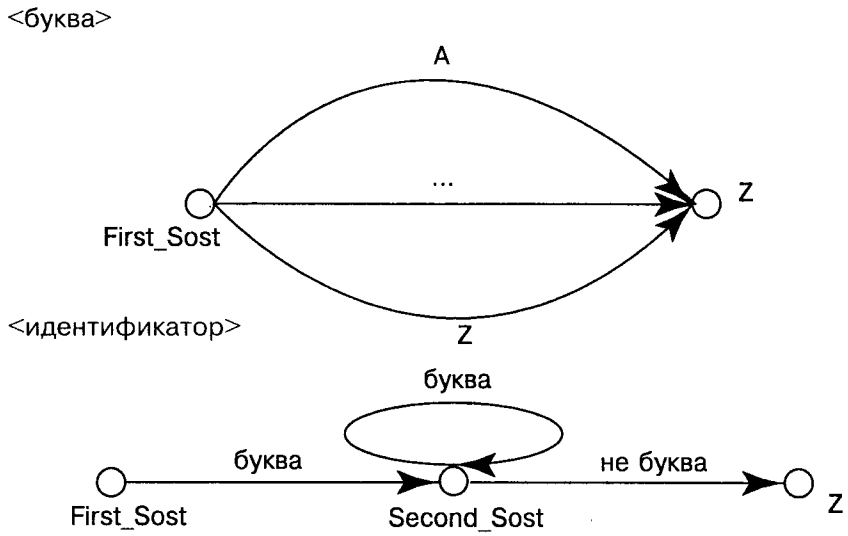
идентификатора на его выходе должен быть сгенерирован дескриптор, имеющий числовой код **101**. На рис.2.7 а), б) и в) показаны основные способы представления модели такого ЛА соответственно в виде диаграммы переходов, программы на ЯОЛ, результатов трансляции ЯОЛ программы — управляющих таблиц конечных распознавателей.

Описание распознавателя (лексемы) всегда начинается с ключевого слова **Lexema** и заканчивается ключевым словом **End**. После слова **Lexema** в угловых скобках указывается имя лексемы. Это имя можно использовать внутри описаний других лексем. Если лексема основная, то в опции **Code** указывается буквенно-цифровая кодировка образа лексемы на выходе ЛА.

Команды переходов распознавателя лексемы задаются в виде:

<**Состояние**> -> <**Условие перехода**> <**Состояние**>.

<**Состояние**> — это любая последовательность из цифр и (или) букв. Причем имя **Z** является зарезирвированным и соответствует *заключительному состоянию* конечного автомата, его нельзя использовать в левой части команды. Имена состояний являются локальными для данного распознавателя, т.е. в распознавателях для различных лексем можно использовать одни и те же идентификаторы для именования внутренних состояний распознавателя. *Начальным состоянием распознавателя* является имя в левой части его *первой команды*.



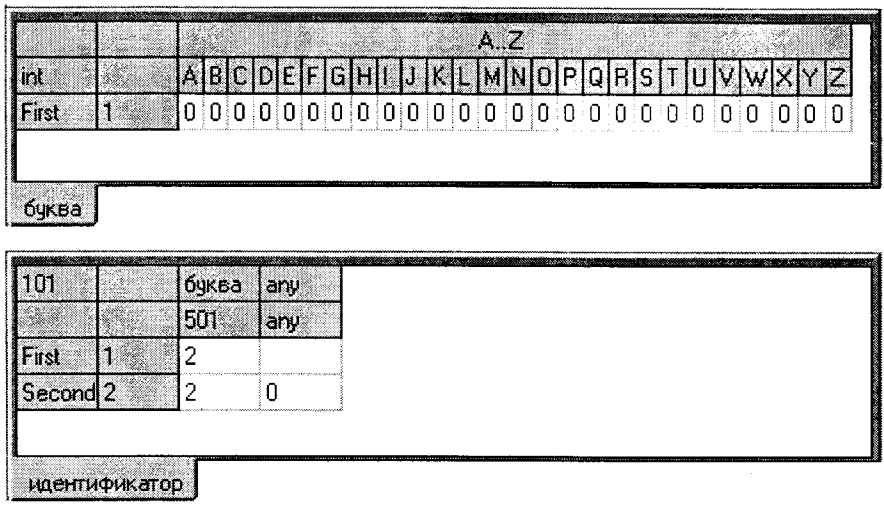
а) диаграммы переходов

Рис. 2.7. Основные способы представления распознавателей

```
Lexema <буква>
    First_Sost -> [A..Z] Z
    First_Sost -> [a..z] Z
End

Lexema <идентификатор> code = 101
    First_Sost -> <буква> Second_Sost
    Second_Sost -> <буква> Second_Sost
    Second_Sost -> Any Z
End
```

б) программа на ЯОЛ



		A..Z																								
int		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
First	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

буква

101		буква	any
		501	any
First	1	2	
Second	2	2	0

идентификатор

в) управляющие таблицы распознавателей лексем

Рис. 2.7 (продолжение). Основные способы представления распознавателей

<Условие перехода> в команде задается либо указанием имени символа из входного алфавита ЛА, либо заданием диапазона значений символов, либо указанием имени ранее описанной лексемы или зарезервированного слова(**Any** или **Space**).

Имена входных символов являются глобальными — одинаковые имена обозначают одно и то же в различных лексемах. *Диапазон значений символов* задается в квадратных скобках и определяет начальный и конечный символы диапазона (в соответствии с кодировкой ASCII), разделенные двумя точками ... Любой входной символ из заданного диапазона будет вызывать переход в следующее состояние распознавателя. *Имя ранее описанной лексемы* записывается в угловых скобках.

На вход интерпретатора описания ЯОЛ может подаваться любое количество примеров цепочек входного языка. При поступлении на вход ЛА ошибочной лексемы, он сразу прекращает работу с выдачей сообщения об ошибке. Лексические единицы друг от друга могут отделяться любым числом пробелов. Пробелы не могут использоваться в качестве префикса лексемы. Пробелы, разделяющие лексемы после распознавания текущей лексемы, удаляются из входной цепочки.

Если в множество цепочек, описываемых конкретной лексемой обязательно входит символ пробел, то для его распознавания применяется ключевое слово **Space**.

Выход из распознавателя текущей лексемы и начало распознавания следующей осуществляется после перехода распознавателя в заключительное состояние **Z**. В качестве условия выхода при переходе в состояние **Z** может использоваться:

- ключевое слово **Any**, в этом случае выход осуществляется по любому символу, не принадлежащему множеству переходов для данного состояния (включая пробел), которые могут быть первыми в других лексемах, при этом все пробелы до первого значащего символа удаляются из входного текста;
- символ, который является последним символом лексемы; если после этого символа во входной строке есть пробелы, то они также удаляются из входного текста;
- имя ранее описанной лексемы.

В ЯОЛ можно запретить передачу образа лексемы — дескриптора на выход ЛА, для этого достаточно в заголовке описания лексемы указать дополнительный параметр (**Nogen**), при этом лексема не потеряет статуса основной. В качестве примера такой ситуации приведено описание лексемы для распознавания комментариев-последовательности символов, ограниченной фигурными скобками { }:

```
lexema <комментарий> (NoGen)
    Sost_1 -> { Sost_2
    Sost_2 -> Any Sost_2
    Sost_2 -> } Z
end
```

Если различные лексемы имеют одинаковые префиксы, то порядок их описания должен соответствовать порядку их включения в обобщенную лексему. Пусть язык L включает в себя три языка, т.е $L=L_1 \bigcup L_2 \bigcup L_3$ и $L_1=\{a\}$, $L_2=\{ab\}$, $L_3=\{abc\}$. Тогда порядок описания лексем должен быть следующим:

```
Lexema <буква>
1->[a..z] Z
end
Lexema <a> code=a
1->a 2
2->Any Z
end
Lexema <ab> code=ab
1->a 2
2->b 3
3->Any Z
end
Lexema <abc> code=abc
1->a 2
2->b 3
3->c Z
end
```

Другой и более надежный способ описания объединения языков, имеющих одинаковые префиксы, использовать в обобщенных описаниях ссылки на ранее описанные лексемы. Такая ситуация приведена в примере описания ЛА для распознавания различных форм описания числовых констант:

```
Lexema <цифра>
1->[0..9] Z
end
lexema <целое без знака> code=int1
1-><цифра> 2
2-><цифра> 2
2->Any Z
end
Lexema <целое со знаком> code=int2
```

```
1->+ 2
1->- 2
2-><целое без знака> 2
2->Any Z
end
Lexema <real dec fixed> code=real_dec_fixed
1-><целое без знака> 2
1-><целое со знаком> 2
1->. 3
2-><целое без знака> 2
2->. 3
3-><целое без знака> 4
3->Any Z
4->Any Z
end
Lexema <real dec float> code=real_dec_float
1-><real dec fixed> 2
2->E 4
4->+ 5
4->- 5
5-><целое без знака> Z
end
```

Здесь при описании числа в экспоненциальной форме в качестве его префикса используется ссылка на описание распознавателя числа с фиксированной точкой.

При поступлении на вход сконструированного на основе этого описания ЛА цепочки

78 -56 +122.45 .09E+11 -234.E+34

на его выходе будет сформирован дескрипторный текст

int1 int2 real_dec_fixed real_dec_float real_dec_float.

В случае же ошибочной входной цепочки (например +23a11) на выходе будет сообщение об ошибке **ERROR**.

В ЯОЛ имеется возможность описывать ключевые (зарезервированные) слова путем их перечисления. Выполняется это в секции **KeyWords ... EndWords** в виде последовательностей пар **<ключевое слово> <код лексемы>**. Секция в программе должна быть единственной, и в программе должен быть описан ровно один распознаватель, в заголовке которого указан параметр (**KeyWords**). Этот параметр указывает на то, что при успешном распознавании лексемы данного класса будет осуществляться проверка принадлежности его к ключевым словам и в случае успеха на выход ЛА будет передаваться код лексемы ключе-

вого слова, а не тот, который задан в опции **Code** при описании класса. Ниже приведен пример описания модели лексического анализатора для языка, который включает в себя ранее определенный класс идентификаторов и ключевые слова **read** и **write**.

```
Lexema <буква>
   First_Sost -> [a..z] Z
End
KeyWords
   read <51>
   write <52>
EndWords
Lexema <идентификатор> code = 101 (KeyWords)
   First_Sost -> <буква> Second_Sost
   Second_Sost -> <буква> Second_Sost
   Second_Sost -> Any Z
End
```

Для входной цепочки **aaaa adfrt read dftg write** на выходе данного ЛА будет сгенерирован дескрипторный код вида **101 101 51 101 52**.

Моделирование поведения собственно ДКА на ЯОЛ проводится аналогично, отличие лишь в том, что программа моделирования ДКА может содержать множество описаний вспомогательных лексем и лишь одно описание основной лексемы с параметром **code**=ДОПУСТИТЬ. Ниже приведен пример описания модели ДКА для распознавания упрощенного варианта записи оператора **with** языка Паскаль.

```
Lexema <буква>
1->[a..z] Z
1->[A..Z] Z
end
Lexema <цифра>
1->[0..9] Z
end
Lexema <Id>
1-><буква> 2
2-><буква>2
2-><цифра>2
2->Any Z
end
Lexema <with>
1->w 2
```

```
2->i 3
3->t 4
4->h Z
end
Lexema <do>
1->d 2
2->o Z
end
Lexema <Sp1>
1->Space Z
end
Lexema <Sp>
1->Space 1
1->Any Z
end
Lexema <with_KA> code=ДОПУСТИТЬ
a1-> <with> a2
a2-><Sp1> a31
a31-><Sp> a3
a3-><Id> a4
a4->, a31
a4-><Sp1> a51
a51-><Sp> a5
a5-><do> a6
a6-><Sp1> a71
a71-><Sp> a7
a7-><Id> Z
end
```

Примеры других программ на языке описания лексики можно найти в каталоге с именем LEXICA системы «КАШТАН». Программы на ЯОЛ хранятся в файлах с расширением **.lex.**

Результаты отладки модели лексического анализатора являются средством спецификации программы лексического анализа, кодирование распознающей части ЛА которой в дальнейшем у обучаемых не вызывает больших трудностей. Примеры программ лексического анализа приведены в Приложении.