Лабораторная работа №2

Грамматики и конечные автоматы

Цель работы: изучить методы и средства, используемые при построении лексических анализаторов, в основе которых лежат регулярные грамматики.

Краткие теоретические сведения

Практически во всех трансляторах (и в компиляторах, и в интерпретаторах) в том или ином виде присутствует большая часть перечисленных ниже процессов:

- лексический анализ
- синтаксический анализ
- семантический анализ
- генерация внутреннего представления программы
- оптимизация
- генерация объектной программы.

В конкретных компиляторах порядок этих процессов может быть несколько иным, некоторые из них могут объединяться в одну фазу, другие могут выполнятся в течение всего процесса компиляции. В интерпретаторах и при смешанной стратегии трансляции некоторые этапы могут вообще отсутствовать.

В этой работе рассматриваются методы и средства, которые обычно используются при построении лексических анализаторов. В основе таких анализаторов лежат регулярные грамматики, поэтому рассмотрим грамматики этого класса более подробно. В дальнейшем под регулярной грамматикой будем понимать леволинейную грамматику.

Для грамматик этого типа существует алгоритм определения того, принадлежит ли анализируемая цепочка языку, порождаемому этой грамматикой (*алгоритм разбора*):
(1) первый символ исходной цепочки $a_1a_2...a_n\perp$ заменяем нетерминалом A, для которого в грамматике есть правило вывода $A \rightarrow a_1$ (другими словами, производим «свертку» терминала a_1 к нетерминалу A);
(2) затем многократно (до тех пор, пока не считаем признак конца цепочки) выполняем следующие шаги: полученный на предыдущем шаге нетерминал A и расположенный непосредственно справа от него очередной терминал a_i исходной цепочки заменяем нетерминалом B, для которого в грамматике есть правило вывода $B \rightarrow Aa_i$ (i = 2, 3,.., n);

Это эквивалентно построению дерева разбора восходящим методом: на каждом шаге алгоритма строим один из уровней в дереве разбора, поднимаясь от листьев к корню.

При работе этого алгоритма возможны следующие ситуации:
(1) прочитана вся цепочка; на каждом шаге находилась единственная нужная «свертка»; на последнем шаге свертка произошла к символу S. Это означает, что исходная цепочка $a_1a_2...a_n\perp \in L(\mathbf{G})$.
(2) прочитана вся цепочка; на каждом шаге находилась единственная нужная «свертка»; на последнем шаге «свертка» произошла к символу, отличному от S. Это означает, что исходная цепочка $a_1a_2...a_n\perp \notin L(\mathbf{G})$.
(3) на некотором шаге не нашлось нужной «свертки», т.е. для полученного на предыдущем шаге нетерминала A и расположенного непосредственно справа от него очередного терминала a_i исходной цепочки не нашлось нетерминала B, для которого в грамматике было бы правило вывода $B \rightarrow Aa_i$. Это означает, что исходная цепочка $a_1a_2...a_n\perp \notin L(\mathbf{G})$.
(4) на некотором шаге работы алгоритма оказалось, что есть более одной подходящей «свертки», т.е. в грамматике разные нетерминалы имеют правила вывода с одинаковыми

правыми частями, и поэтому непонятно, к какому из них производить «свертку». Это говорит о *недетерминированности разбора.* Анализ этой ситуации будет дан ниже.

Допустим, что разбор на каждом шаге детерминированный. Для того, чтобы быстрее находить правило с подходящей правой частью, зафиксируем все возможные «свертки» (это определяется только грамматикой и не зависит от вида анализируемой цепочки). Это можно сделать в виде таблицы, строки которой помечены нетерминальными символами грамматики, столбцы – терминальными. Значение каждого элемента таблицы – это нетерминальный символ, к которому можно свернуть пару «нетерминал-терминал», которыми помечены соответствующие строка и столбец.

Пример 1. Для грамматики **G** ({a, b, ⊥}, {S, **A, B, C**}, **P**, S), такая таблица будет выглядеть следующим образом:

P: $S \to C\perp$
$C \to Ab \mid Ba$
$A \to a \mid Ca$
$B \to b \mid Cb$

	a	b	⊥
C	A	B	S
A	-	C	-
B	C	-	-
S	-	-	-

Знак «-» ставится в том случае, если для пары «терминал-нетерминал» «свертки» нет.

Но чаще информацию о возможных свертках представляют в виде *диаграммы состояний (ДС)* – неупорядоченного ориентированного помеченного графа, который строится следующим образом:

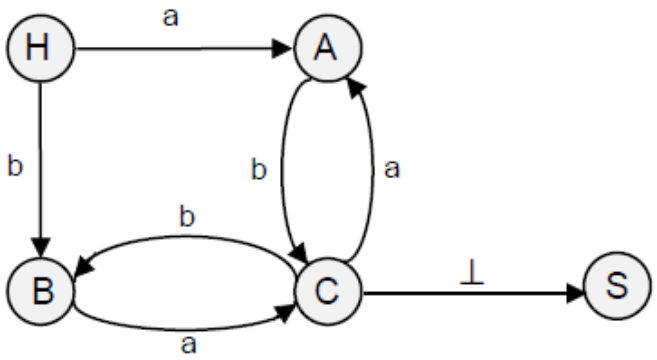
(1) строятся вершины графа, помеченные нетерминалами грамматики (для каждого нетерминала – одну вершину), и еще одну вершину, помеченную символом, отличным от нетерминальных (например, H). Эти вершины называют *состояниями*. H – начальное состояние.

(2) соединяем эти состояния дугами по следующим правилам:

а) для каждого правила грамматики вида W → t соединяем дугой состояния H и W (от H к W) и помечаем дугу символом t;

б) для каждого правила W → Vt соединяем дугой состояния V и W (от V к W) и помечаем дугу символом t;

Диаграмма состояний для грамматики **G** из примера 1:



Алгоритм разбора по диаграмме состояний:

(1) объявляем текущим состояние H;

(2) затем многократно (до тех пор, пока не считаем признак конца цепочки) выполняем следующие шаги: считываем очередной символ исходной цепочки и переходим из текущего состояния в другое состояние по дуге, помеченной этим символом. Состояние, в которое мы при этом попадаем, становится текущим.

При работе этого алгоритма возможны следующие ситуации (аналогичные ситуациям, которые возникают при разборе непосредственно по регулярной грамматике):

(1) прочитана вся цепочка; на каждом шаге находилась единственная дуга, помеченная очередным символом анализируемой цепочки; в результате последнего перехода оказались в состоянии S. Это означает, что исходная цепочка принадлежит L(**G**).
(2) прочитана вся цепочка; на каждом шаге находилась единственная «нужная» дуга; в результате последнего шага оказались в состоянии, отличном от S. Это означает, что исходная цепочка не принадлежит L(**G**).
(3) на некотором шаге не нашлось дуги, выходящей из текущего состояния и помеченной очередным анализируемым символом. Это означает, что исходная цепочка не принадлежит L(**G**).
(4) на некотором шаге работы алгоритма оказалось, что есть несколько дуг, выходящих из текущего состояния, помеченных очередным анализируемым символом, но ведущих в разные состояния. Это говорит о *недетерминированности разбора*. Анализ этой ситуации будет приведен ниже.

Диаграмма состояний определяет конечный автомат, построенный по регулярной грамматике, который допускает множество цепочек, составляющих язык, определяемый этой грамматикой. Состояния и дуги ДС – это графическое изображение функции переходов конечного автомата из состояния в состояние при условии, что очередной анализируемый символ совпадает с символом-меткой дуги.

Среди всех состояний выделяется начальное (считается, что в начальный момент своей работы автомат находится в этом состоянии) и конечное (если автомат завершает работу переходом в это состояние, то анализируемая цепочка им допускается).

Таким образом, *конечный автомат (КА)* – это пятерка (K, VT, F, H, S), где K – конечное множество состояний; VT – конечное множество допустимых входных символов; F – отображение множества $K \times VT \rightarrow K$, определяющее поведение автомата; отображение F часто называют функцией переходов; $H \in K$ – начальное состояние; $S \in K$ – заключительное состояние (либо конечное множество заключительных состояний). F(A, t) = B означает, что из состояния A по входному символу t происходит переход в состояние B.

Конечный автомат *допускает цепочку* $a_1a_2...a_n$, если $F(H,a_1) = A_1$; $F(A_1,a_2) = A_2$; …; $F(A_{n-2},a_{n-1}) = A_{n-1}$; $F(A_{n-1},a_n) = S$, где $a_i \in VT$, $A_j \in K$, j = 1, 2 , …, n–1; i = 1, 2, ..., n; H – начальное состояние, S – одно из заключительных состояний.

Для более удобной работы с диаграммами состояний введем несколько соглашений:
a) если из одного состояния в другое выходит несколько дуг, помеченных разными символами, то будем изображать одну дугу, помеченную всеми этими символами;
b) непомеченная дуга будет соответствовать переходу при любом символе, кроме тех, которыми помечены другие дуги, выходящие из этого состояния.
c) введем состояние ошибки (ER); переход в это состояние будет означать, что исходная цепочка языку не принадлежит.

По диаграмме состояний можно написать анализатор для регулярной грамматики.

Для грамматики из примера 1 анализатор будет таким:

```
#include <stdio.h>
int scan_G()
{
      enum state {H, A, B, C, S, ER}; /* множество состояний */
      enum state CS; /* CS - текущее состояние */
      FILE *fp;/*указатель на файл, в котором находится анализируемая
      цепочка */
      int c;
      CS = H;
      fp = fopen("data","r");
      c = fgetc(fp);
      do {
            switch(CS)
```

```
            {
            case H:
                  if(c == 'a')
                  {
                        c = fgetc(fp);
                        CS = A;
                  }
                  else if(c == 'b')
                  {
                        c = fgetc(fp);
                        CS = B;
                  }
                  else CS = ER;
                  break;
            case A:
                  if(c == 'b')
                  {
                        c = fgetc(fp);
                        CS = C;
                  }
                  else CS = ER;
                  break;
            case B:
                  if(c == 'a')
                  {
                        c = fgetc(fp);
                        CS = C;
                  }
                  else CS = ER;
                  break;
            case C:
                  if(c =='a')
                  {
                        c = fgetc(fp);
                        CS = A;
                  }
                  else if(c == 'b')
                  {
                        c = fgetc(fp);
                        CS = B;
                  }
                  else if(c == '⊥') CS = S;
                  else CS = ER;
                  break;
            }
      } while(CS != S && CS != ER);
      if(CS == ER) return -1;
      else return 0;
}
```

При анализе по регулярной грамматике может оказаться, что несколько нетерминалов имеют одинаковые правые части, и поэтому неясно, к какому из них делать «свертку» (см. описание алгоритма). В терминах диаграммы состояний это означает, что из одного состояния выходит несколько дуг, ведущих в разные состояния, но помеченных одним и тем же символом.

Пример 2. Для грамматики **G** ({a,b, ⊥}, {S,**A,B**}, **P**, S), где

P:

$S \rightarrow A\perp$

$A \rightarrow a \mid Bb$
$B \rightarrow b \mid Bb$
разбор будет недетерминированным (т.к. у нетерминалов A и B есть одинаковые правые части – Bb). Такой грамматике будет соответствовать недетерминированный конечный автомат.

Недетерминированный конечный автомат (НКА) – это пятерка (K, VT, F, H, S), где K – конечное множество состояний; VT – конечное множество допустимых входных символов; F – отображение множества $K \times VT$ в множество подмножеств K; $H \subset K$ – конечное множество начальных состояний; $S \subset K$ – конечное множество заключительных состояний. $F(A, t) = \{B_1, B_2, ..., B_n\}$ означает, что из состояния A по входному символу t можно осуществить переход в любое из состояний B_i, $i = 1, 2, ..., n$.

В этом случае можно предложить алгоритм, который будет перебирать все возможные варианты «сверток» (переходов) один за другим; если цепочка принадлежит языку, то будет найден путь, ведущий к успеху; если будут просмотрены все варианты, и каждый из них будет завершаться неудачей, то цепочка языку не принадлежит. Однако такой алгоритм практически неприемлем, поскольку при переборе вариантов мы, скорее всего, снова окажемся перед проблемой выбора и, следовательно, будем иметь «дерево отложенных вариантов».

Один из наиболее важных результатов теории конечных автоматов состоит в том, что класс языков, определяемых недетерминированными конечными автоматами, совпадает с классом языков, определяемых детерминированными конечными автоматами. Это означает, что для любого НКА всегда можно построить детерминированный КА, определяющий тот же язык.

Алгоритм построения детерминированного КА по НКА

Вход: M = (K, VT, F, H, S) – недетерминированный конечный автомат.
Выход: M’ = (K’, VT, F’, H’, S’) – детерминированный конечный автомат, допускающий тот же язык, что и автомат M.
Метод:
1. Множество состояний K’ состоит из всех подмножеств множества K. Каждое состояние из K’ будем обозначать $[A_1A_2...A_n]$, где $A_i \in K$.
2. Отображение F’ определим как $F'([A_1A_2...A_n], t) = [B_1B_2...B_m]$, где для каждого $1 \le j \le m$, $F(A_i, t) = B_j$ для каких-либо $1 \le i \le n$.
3. Пусть $H = \{H_1, H_2, ..., H_k\}$, тогда $H' = [H_1, H_2, ..., H_k]$.
4. Пусть $S = \{S_1, S_2, ..., S_p\}$, тогда S’ – все состояния из K’, имеющие вид $[...S_i...]$, $S_i \in S$ для какого-либо $1 \le i \le p$.

В множестве K’ могут оказаться состояния, которые недостижимы из начального состояния, их можно исключить.

Пример 2.

Пусть задан НКА M = ({H, A, B, S}, {0, 1}, F, {H}, {S}), где

F(H, 1) = B, F(B, 0) = A, F(A, 1) = B, F(A, 1) = S,

тогда соответствующий детерминированный конечный автомат будет таким:

K’ = {[H], [A], [B], [S], [HA], [HB], [HS], [AB], [AS], [BS], [HAB], [HAS], [ABS], [HBS], [HABS]}

F’([A], 1) = [BS]
F’([H], 1) = [B]
F’([B], 0) = [A]
F’([HA], 1) = [BS]
F’([HB], 1) = [B]
F’([HB], 0) = [A]
F’([HS], 1) = [B]
F’([AB], 1) = [BS]
F’([AB], 0) = [A]
F’([AS], 1) = [BS]
F’([BS], 0) = [A]
F’([HAB], 0) = [A]
F’([HAB], 1) = [BS]
F’([HAS], 1) = [BS]
F’([ABS], 1) = [BS]
F’([ABS], 0) = [A]
F’([HBS], 1) = [B]
F’([HBS], 0) = [A]

F’([HABS], 1) = [BS] F’([HABS], 0) = [A]
S’ = {[S], [HS], [AS], [BS], [HAS], [ABS], [HBS], [HABS]}
Достижимыми состояниями в получившемся КА являются [H], [B], [A] и [BS], поэтому остальные состояния удаляются.
Таким образом, M’ = ({[H], [B], [A], [BS]}, {0, 1}, F’, H, {[BS]}), где F’([A], 1) = [BS] F’([H], 1) = [B] F’([B], 0) = [A] F’([BS], 0) = [A]

Задания

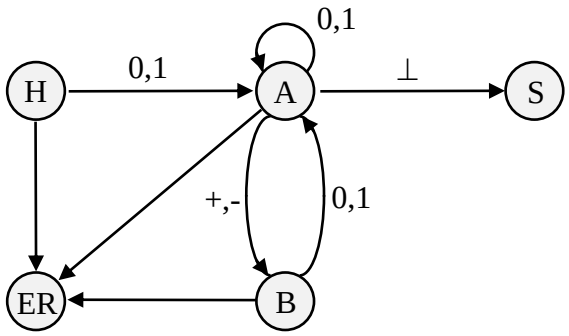
Общие

1. Дана регулярная грамматика с правилами:

$S \rightarrow S0 \mid S1 \mid P0 \mid P1$
$P \rightarrow N.$
$N \rightarrow 0 \mid 1 \mid N0 \mid N1$

Построить по ней диаграмму состояний и использовать ДС для разбора цепочек : 11.010 , 0.1 , 01. , 100 Какой язык порождает эта грамматика ?

2. Дана ДС.



a) Осуществить разбор цепочек 1011⊥, 10+011⊥ и 0–101+1⊥.
b) Восстановить регулярную грамматику, по которой была построена данная ДС.
c) Какой язык порождает полученная грамматика?

3. Написать леволинейную регулярную грамматику, эквивалентную данной праволинейной, допускающую детерминированный разбор.

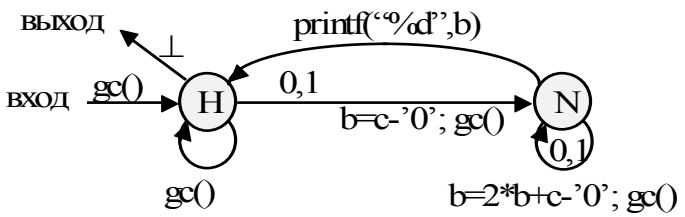
a) $S \rightarrow 0S \mid 0B$
$B \rightarrow 1B \mid 1C$
$C \rightarrow 1C \mid \perp$

b) $S \rightarrow aA \mid aB \mid bA$
$A \rightarrow bS$
$B \rightarrow aS \mid bB \mid \perp$

c) $S \rightarrow aB$
$B \rightarrow aC \mid aD \mid dB$
$C \rightarrow aB$
$D \rightarrow \perp$

d) $S \rightarrow 0B$
$B \rightarrow 1C \mid 1S$
$C \rightarrow \perp$

По вариантам

Вариант	Задание
1	Пусть имеется переменная **c** и функция gc(), считывающая в **c** очередной символ анализируемой цепочки. Дана ДС с действиями:  a) Определить, что будет выдано на печать при разборе цепочки 1+101//p11+++1000/5⊥? b) Написать на Си анализатор по этой ДС.
2	Дана грамматика: G = ({Q, P, R, S}, {0, 1, *, $, /}, V, Q), где V: Q→1P P→*S | $ S→0R R→/Q | $ Определить язык, который она порождает; построить ДС; написать на Си анализатор.
3	Дана регулярная грамматика: $S \to A\perp$ $A \to Ab \mid Bb \mid b$ $B \to Aa$ Определить язык, который она порождает; построить ДС; написать на Си анализатор.
4	Для данной грамматики a) определить ее тип; b) определить, какой язык она порождает; c) написать Р-грамматику, почти эквивалентную данной; d) построить ДС и анализатор на Си. $S \to 0S \mid S0 \mid D$ $D \to DD \mid 1A \mid \varepsilon$ $A \to 0B \mid \varepsilon$ $B \to 0A \mid 0$
5	Дана грамматика: $S \to C\perp$ $B \to B1 \mid 0 \mid D0$ $C \to B1 \mid C1$ $D \to D0 \mid 0$ Определить язык, который она порождает; построить ДС; написать на Си анализатор.
6	Дана грамматика: $S \to C\perp$ $C \to B1$ $B \to 0 \mid D0$ $D \to B1$ Определить язык, который она порождает; построить ДС; написать на Си анализатор.

7	Дана грамматика: $S \rightarrow A0$ $A \rightarrow A0 \mid S1 \mid 0$ Определить язык, который она порождает; построить ДС; написать на Си анализатор.
8	Дана грамматика: $S \rightarrow 0A \mid 1S$ $A \rightarrow 0A \mid 1B$ $B \rightarrow 0B \mid 1B \mid \perp$ Определить язык, который она порождает; построить ДС; написать на Си анализатор.
9	Дана грамматика: $S \rightarrow B\perp$ $A \rightarrow B1 \mid 0$ $B \rightarrow A1 \mid C1 \mid B0 \mid 1$ $C \rightarrow A0 \mid B1$ Определить язык, который она порождает; построить ДС; написать на Си анализатор.
10	Дана грамматика: $S \rightarrow 0S \mid 0B$ $B \rightarrow 1B \mid 1C$ $C \rightarrow 1C \mid \varepsilon$ Определить язык, который она порождает; построить ДС; написать на Си анализатор.
11	Дана грамматика: G = ({K, L, M, N}, {0, 1, $}, V, K), где V: $K \rightarrow 1L \mid 0N$ $L \rightarrow 0M$ $N \rightarrow 1L \mid 1M$ $M \rightarrow \$$ Определить язык, который она порождает; построить ДС; написать на Си анализатор.
12	Дана грамматика: G = ({S, C, D}, {0, 1}, P, S), где P: $S \rightarrow 1C \mid 0D$ $C \rightarrow 0D \mid 0S \mid 1$ $D \rightarrow 1C \mid 1S \mid 0$ Определить язык, который она порождает; построить ДС; написать на Си анализатор.
13	Дана грамматика: $S \rightarrow B0 \mid 0$ $B \rightarrow B0 \mid C1 \mid 0 \mid 1$ $C \rightarrow B0$ Определить язык, который она порождает; построить ДС; написать на Си анализатор.
14	Дана грамматика: $S \rightarrow Sb \mid Aa \mid a \mid b$ $A \rightarrow Aa \mid Sb \mid a$ Определить язык, который она порождает; построить ДС; написать на Си анализатор.
15	Дана грамматика:

	$S \to A0 \mid A1 \mid B1 \mid 0 \mid 1$ $A \to A1 \mid B1 \mid 1$ $B \to A0$ Определить язык, который она порождает; построить ДС; написать на Си анализатор.
16	Дана грамматика: $S \to S0 \mid A1 \mid 0 \mid 1$ $A \to A1 \mid B0 \mid 0 \mid 1$ $B \to A0$ Определить язык, который она порождает; построить ДС; написать на Си анализатор.
17	Дана грамматика: $S \to A\perp$ $A \to A0 \mid A1 \mid B1$ $B \to B0 \mid C0 \mid 0$ $C \to C1 \mid 1$ Определить язык, который она порождает; построить ДС; написать на Си анализатор.
18	Даны две грамматики G1 и G2, порождающие языки L1 и L2. Построить регулярную грамматику для $L1 \cup L2$. Для полученной грамматики построить ДС и написать на Си анализатор. G1: $S \to S1 \mid A0$ $A \to A1 \mid 0$ G2: $S \to A1 \mid B0 \mid E1$ $A \to S1$ $B \to C1 \mid D1$ $C \to 0$ $D \to B1$ $E \to E0 \mid 1$
19	Даны две грамматики G1 и G2, порождающие языки L1 и L2. Построить регулярную грамматику для $L1 \cap L2$. Для полученной грамматики построить ДС и написать на Си анализатор. G1: $S \to S1 \mid A0$ $A \to A1 \mid 0$ G2: $S \to A1 \mid B0 \mid E1$ $A \to S1$ $B \to C1 \mid D1$ $C \to 0$ $D \to B1$ $E \to E0 \mid 1$
20	Дана грамматика: G = ({I, J, K, M, N}, {0, 1, ~, !}, P, I), где P: $I \to 0J \mid 1K \mid 0M$ $J \to \sim K \mid 0M$ $K \to \sim M \mid 0J \mid 0N$ $M \to 1K \mid !$ $N \to 0I \mid 1I \mid !$ Определить язык, который она порождает; построить ДС; написать на Си анализатор.
21	Дана грамматика: G=({S, A, B, C}, {a, b, c}, P, S), где P: $S \to aA \mid bB \mid aC$ $A \to bA \mid bB \mid c$ $B \to aA \mid cC \mid b$ $C \to bB \mid bC \mid a$

	Определить язык, который она порождает; построить ДС; написать на Си анализатор.
22	Дана грамматика: G = ({K, L, M, N}, {a, b, +, –, ⊥ }, P, K), где P: K → aL | bM L → –N | –M M → +N N→aL | bM | ⊥ Определить язык, который она порождает; построить ДС; написать на Си анализатор.
23	Дана грамматика: G = ({X, Y, Z, V, W}, {0, 1, x, y, z}, P, X), где P: X → yY | zZ Y → 1V Z → 0W | 0Y V → xZ | xW | 1 W → 1Y | 0 Определить язык, который она порождает; построить ДС; написать на Си анализатор.
24	Дана грамматика: G = ({S, A, B, C, D}, {a, b, c, d, ⊥ }, P, S), где P: S → aA | bB A → cC | ⊥ C → cC | cA B → dD | ⊥ D → dD |dB Определить язык, который она порождает; построить ДС; написать на Си анализатор.
25	Дана грамматика: G = ({S, A, B, C, D}, {a, b, c, d, #, ⊥ }, P, S), где P: S→aA | bB | cC A→dD B→#D D→dD | dB | ⊥ C→cD Определить язык, который она порождает; построить ДС; написать на Си анализатор.

Литература

1. Молчанов А.Ю. Системное программное обеспечение: Учебник для вузов. 3-е изд. – СПб.: Питер, 2010. – 400 с.
2. Ахо А.В., Лам М. С., Сети Р., Ульман Дж. Д. Компиляторы: принципы, технологии и инструментарий, 2-е изд.: Пер. с англ. – М.: Вильямс, 2008. – 1184 с. (разделы 3.4, 3.6, 3.7)
3. Волкова И.А., Руденко Т.В. Формальные грамматики и языки. Элементы теории трансляции: Учебное пособие. – Издательский отдел факультета вычислительной математики и кибернетики МГУ им. М.В. Ломоносова, 1999. – 62 с.
4. Свердлов С.З. Языки программирования и методы трансляции: Учебное пособие. — СПб.: Питер, 2007. — 638 с. (стр. 221–230)