ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

УПРАВЛЕНИЕ ДИСТАНЦИОННОГО ОБУЧЕНИЯ И ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

Учебно-методическое пособие

по дисциплине

«Теория автоматов и формальных языков»

Автор
Романенко Е. А.

Ростов-на-Дону, 2019

Аннотация

Учебно-методическое пособие предназначено для студентов очной формы обучения направления 09.03.04 «Программная инженерия» и 02.03.03 «Математическое обеспечение и администрирование информационных систем»

Авторы

Старший преподаватель
каф. ПОВТиАС
Романенко Е.А.

Оглавление

1. ЛАБОРАТОРНАЯ РАБОТА №1: АВТОМАТЫ-РАСПОЗНАВАТЕЛИ. ДЕТЕРМИНИЗАЦИЯ АВТОМАТА

1.1. Теоретическая часть

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):

Первый символ исходной цепочки a1a2...an⊥ заменяем нетерминалом A, для которого в грамматике есть правило вывода A → a1 (другими словами, производим «свертку» терминала a1 к нетерминалу A);

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:

• прочитана вся цепочка; на каждом шаге находилась единственная нужная «свертка»; на последнем шаге свертка произошла к символу S. Это означает, что исходная цепочка $a_1a_2...a_n \perp \in L(G)$.

• 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 $a_1a_2...a_n \perp L(G)$.

• на некотором шаге не нашлось нужной «свертки», т.е. для полученного на предыдущем шаге нетерминала A и расположенного непосредственно справа от него очередного терминала ai исходной цепочки не нашлось нетерминала B, для которого в грамматике было бы правило вывода $B \rightarrow Aa_i$. Это означает, что исходная цепочка $a_1a_2...a_n \perp L(G)$.

• 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 (таблица 1) будет выглядеть следующим образом:

Таблица 1

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:

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.

Соединяем эти состояния дугами по следующим правилам:

а) для каждого правила грамматики вида W → t соединяем дугой состояния H и W (от H к 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 представлена на рисунке 1.

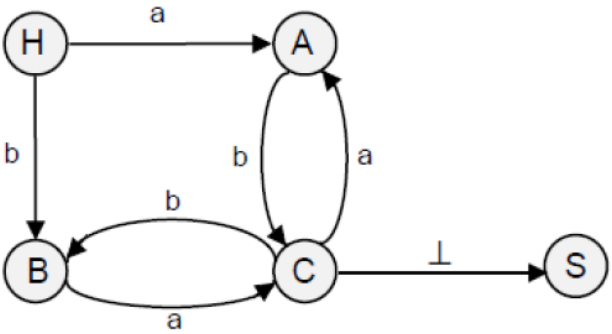


Рисунок 1 — Диаграмма состояний для грамматики G из примера 1

Алгоритм разбора по диаграмме состояний:

• объявляем текущим состояние H;

• 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):

• 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).

• 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).

• на некотором шаге не нашлось дуги, выходящей из текущего состояния и помеченной очередным анализируемым символом. Это означает, что исходная цепочка не принадлежит L(G).

• 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, V_T, F, H, S), где K – конечное множество состояний; V_T – конечное множество допустимых входных символов; F – отображение множества $K \times V_T \rightarrow K$, 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(A1, a2) = A_2; …;

F (A_n-2, a_n-1) = A_n-1; F(A_n-1, a_n) = S, где $a_i \in V_T$, $A_j \in K$, 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:

• если из одного состояния в другое выходит несколько дуг, помеченных разными символами, то будем изображать одну дугу, помеченную всеми этими символами;

• непомеченная дуга будет соответствовать переходу при любом символе, кроме тех, которыми помечены другие дуги, выходящие из этого состояния.

• 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 оди-наковые 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 $K \times V_T$ в множество подмножеств K; $H \subset K$ – ко-

нечное 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, V_T, 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 Ai ∈ K.

2. Отображение F′ определим как F′ ([$A_1A_2...A_n$], t) = [B1B2...Bm], где для каждого 1 j m, F(Ai, t) = Bj для каких-либо 1 i 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 [...Si...], Si ∈ 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]}

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]

1.2. Задание к лабораторной работе

Общие задания:

1. Дана регулярная грамматика с правилами:

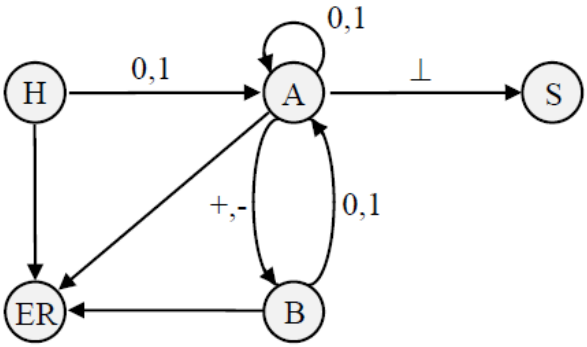
$S \rightarrow S_0 \mid S_1 \mid P_0 \mid P_1$

$P \rightarrow N.$

$N \rightarrow 0 \mid 1 \mid N_0 \mid N_1$

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.



Необходимо:

• Осуществить разбор цепочек 1011⊥, 10+011⊥ и 0–101+1⊥.

• Восстановить регулярную грамматику, по которой была построена данная ДС.

• Какой язык порождает полученная грамматика?

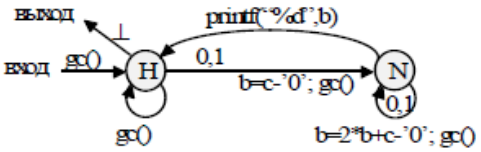
3. Написать леволинейную регулярную грамматику, эквивалентную данной праволинейной, допускающую детерминированный разбор.

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

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

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

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

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 → C⊥ B → B1 | 0 | D0 C → B1 | C1 D → D0 | 0 Определить язык, который она порождает; построить ДС; написать на Си анализатор.
6	Дана грамматика: S → C⊥ C → B1 B → 0 | D0 D → B1 Определить язык, который она порождает; построить ДС; написать на Си анализатор.
7	Дана грамматика: S → A0 A → A0 | S1 | 0 Определить язык, который она порождает; построить ДС; написать на Си анализатор.
8	Дана грамматика: S → 0A | 1S A → 0A | 1B B → 0B | 1B | ⊥ Определить язык, который она порождает; построить ДС; написать на Си анализатор.
9	Дана грамматика: S → B⊥ A → B1 | 0 B → A1 | C1 | B0 | 1 C → A0 | B1 Определить язык, который она порождает; построить ДС; написать на Си анализатор.
10	Дана грамматика: S → 0S | 0B B → 1B | 1C C → 1C | ε Определить язык, который она порождает; построить ДС; написать на Си анализатор.
11	Дана грамматика: G = ({K, L, M, N}, {0, 1, $}, V, K), где V: K→1L \| 0N L→0M N→1L \| 1M M→$ 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→1C | 0D C→0D | 0S | 1 D→1C | 1S | 0 Определить язык, который она порождает; построить ДС; написать на Си анализатор.

1.3. Контрольные вопросы

1. Какие процессы присутствуют во всех трансляторах?
2. Алгоритмы построения недетерминированного и детерминированного конечного автомата?
3. Какие состояния выделяются в автомате?
4. Что такое диаграмма состояний? Опишите алгоритм разбора по диаграмме состояний.

2. ЛАБОРАТОРНАЯ РАБОТА №2: МИНИМИЗАЦИЯ АВТОМАТА МИЛИ. ПРЯМОЕ ПРОИЗВЕДЕНИЕ АВТОМАТОВ. ЭКВИВАЛЕНТНОСТЬ АВТОМАТОВ

2.1. Теоретическая часть

Две булевы функции F_1 и F_2 эквивалентны, если на всех возможных наборах входных переменных они принимают одинаковые значения. Поскольку число наборов у булевых функций конечно, то, перебрав их все, можно проверить, эквивалентны ли F_1 и F_2. Можно также привести F_1 и F_2 к совершенной нормальной форме и сравнить их представления.

Иная ситуация с конечными автоматами. Два конечных автомата эквивалентны, если реализуемые ими отображения вход-выход эквивалентны. Конечный автомат реализует отображение бесконечного множества входных последовательностей сигналов в бесконечное множество выходных последовательностей сигналов. Поэтому автоматные отображения нельзя сравнить простым перечислением их значений на всех возможных аргументах.

Конечные автоматы $A = \{X_A, Y_A, Q_A, \delta_A, \lambda_A\}$ и $B = \{X_B, Y_B, Q_B, \delta_B, \lambda_B\}$ называются эквивалентными, если выполняются два условия:

а) их входные алфавиты совпадают: $X_A = X_B = X$;

б) реализуемые ими автоматные отображения совпадают:

$$(\forall \chi \in X^*)\ \lambda_A^*(q_A, \chi) = \lambda_B^*(q_B, \chi).$$

Эквивалентность автоматов означает, что любое автоматное отображение, реализуемое одним из них, может быть реализовано другим; иначе говоря, их возможности по реализации преобразования входной инфор- мации в выходную совпадают.

Два автомата называются эквивалентными, если они реализуют одинаковые отображения входных слов в выходные на всей области определения отображения. Автоматы Мили и Мура могут быть преобразованы друг в друга, т. е. для всякого автомата Мура может быть построен эквивалентный ему автомат Мили и, наоборот, для всякого автомата Мили – эквивалентный ему автомат Мура.

Определить эквивалентность двух автоматов перебором их реакций на всех входных словах достаточно сложно, так как входных слов бесконечно много. Однако существует достаточно простой метод решения этой проблемы, основанный на понятии прямого произведения автоматов.

Прямым произведением конечных автоматов $A = \{X, Y_A, Q_A, \delta_A, \lambda_A\}$ и $B = \{X, Y_B, Q_B, \delta_B, \lambda_B\}$ с одинаковым входным алфавитом X называется автомат $A \times B = \{X, Y_A \times Y_B, Q_A \times Q_B, \delta_{A\times B}, \lambda_{A\times B}\}$, где:

а) $(\forall q_A \in Q_A)\ (\forall q_B \in Q_B)\ (\forall x \in X)\ \delta_{A\times B}((q_A, q_B)\, x) = (\delta_A(q_A, x), \delta_B(q_B, x))$;

б) $(\forall q_A \in Q_A)\ (\forall q_B \in Q_B)\ (\forall x \in X)\ \lambda_{A\times B}((q_A, q_B)\, x) = (\lambda_A(q_A, x), \lambda_B(q_B, x))$.

Иначе говоря, конечный автомат, являющийся прямым произведением двух конечных автоматов, в качестве своих состояний имеет пары состояний исходных автоматов, выходной алфавит – множество пар выходных символов автоматов множителей, а функции выходов и переходов определены покомпонентно. Таким образом, это просто два стоящих рядом невзаимодействующих конечных автомата, синхронно работающих на одном общем входе (рисунок 1). Прямое произведение автоматов называется также синхронной композицией (рисунок 1).

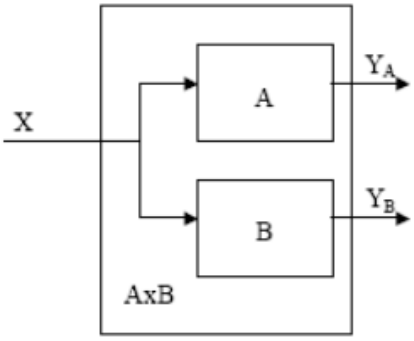


Рисунок 1 — Прямое произведение автоматов

Теорема Мура: Два конечных автомата A = {X, YA, QA, δA, λA} и B = {X, YB, QB, δB, λB} с одинаковым входным алфавитом X являются эквивалентными тогда и только тогда, когда для любого достижимого состояния (qA, qB) в их прямом произведении A×B справедливо ($\forall$ x∈X) λA(qA, x)= = λB(qB, x).

Задача минимизации заключается в отыскании автомата с минимальным числом состояний среди всех автоматов, реализующих заданные условия работы. Под заданными условиями работы понимаются входной и выходной алфавиты автоматов. Задача минимизации возникает по той причине, что преобразование входного алфавита в выходной можно реализовать с помощью автоматов с разным числом состояний. Так как внутреннее состояние определяет объем памяти устройства, в процессе минимизации состояний определяется минимальное число элементов памяти. Для определения числа элементов памяти используется соотношение

$$N = \left| \log_2 M \right|,$$

где N – число элементов памяти,

M – число состояний автомата,

$|\cdot|$ – взятие наибольшего ближайшего целого.

Из приведенного соотношения видно, что при минимизации не всегда можно обеспечить уменьшение числа элементов памяти. Однако и в этом случае не следует пренебрегать минимизацией внутренних состояний, так как уменьшение числа состояний без сокращения числа элементов памяти приводит к появлению состояний, использование которых в дальнейшем позволяет упростить структуру комбинационной части автомата.

Известные методы минимизации дают точное решение только для полных автоматов. При минимизации частичных автоматов возникают трудности, связанные с перебором множества вариантов реализации автомата. Здесь рассматриваются две группы методов. Одна группа – методы, позволяющие быстро и эффективно с помощью ЭВМ найти автомат, близкий к минимальному, т.е. приближенные методы. Другая группа методов обеспечивает точное решение задачи для автоматов определенного вида, например потенциальных, т.е. автоматов, описываемых каноническими уравнениями. Общий подход во всех методах минимизации автоматов общего типа заключается в последовательном уменьшении заданного числа состояний до тех пор, пока ав-

томат реализует заданные условия работы.

Метод Ауфенкампа и Хона: Основная идея этого метода состоит в разбиении всех состояний исходного абстрактного автомата на попарно непересекающиеся классы эквивалентных состояний и замене каждого класса эквивалентности одним состоянием. Получающийся в результате минимизации автомат имеет столько состояний, на сколько классов эквивалентности разбиваются состояния исходного автомата.

Состояния a_m и a_s называются эквивалентными ($a_m \sim a_s$), если $\lambda(a_m, \xi) = \lambda(a_s, \xi)$ для всевозможных входных слов ξ. Состояния a_m и a_s называются k - эквивалентными ($a_m \overset{k}{\sim} a_s$), если $\lambda(a_m, \xi) = \lambda(a_s, \xi)$ для всевозможных входных слов длины k.

Минимизация автомата Мили

Алгоритм минимизации автомата Мили $S = \{A, Z, W, \delta, \lambda, a_1\}$ состоит из следующих шагов:

Находятся последовательные разбиения π_1, π_2, ..., π_k, π_{k+1} множества A на классы одно-, двух-, … , K-, K+1- эквивалентных состояний до тех пор, пока на каком-то (K+1) шаге не окажется, что $\pi_k = \pi_{k+1}$.

Одноэквивалентными будут состояния с одинаковыми столбцами в таблице выходов. Состояния будут двухэквивалентными, если они одноэквивалентны и под действием одинаковых входных сигналов попадают в одинаковые одноэквивалентные классы.

В каждом классе эквивалентности разбиения π выбирается по одному состоянию, в результате чего получается множество A' состояний минимального автомата $S' = \{ A', Z, W, \delta', \lambda', \}$, эквивалентного автомату S.

Для определения функции переходов δ' и функции выходов λ' автомата S' в таблице переходов и выходов вычёркиваются столбцы, соответствующие не вошедшим в A' состояниям. В оставшихся столбцах не вошедшие в множество A состояния заменяются на эквивалентные.

В качестве начального состояния a_1' выбирается состояние, эквивалентное состоянию a_1. В частности, удобно за a_1' принимать само состояние a_1.

Пример 1. Минимизировать полностью определённый автомат Мили S_1, заданный таблицами переходов и выходов (таблице 1 и 2).

Таблица 1 Таблица 2

	a_1	a_2	a_3	a_4	a_5	a_6
z_1	a_3	a_4	a_3	a_4	a_5	a_6
z_2	a_5	a_6	a_5	a_6	a_1	a_2

	a_1	a_2	a_3	a_4	a_5	a_6
z_1	w_1	w_1	w_1	w_1	w_1	w_1
z_2	w_1	w_1	w_2	w_2	w_1	w_1

Решение. 1. По таблице выходов (см. табл. 2) находим разбиение π_1 на классы одноэквивалентных состояний, объединяя одинаковые столбцы:

$\pi_1 = \{B_1, B_2\} = \{\{a_1, a_2, a_5, a_6\},\{ a_3, a_4\}\}$.

Для сокращения числа скобок будем использовать надчёркивания, а элементы множества под чертой разделять точками:

$\pi_1 = \{B_1, B_2\} = \{\overline{1.2.5.6},\ \ \overline{3.4}\}$.

Строим таблицу π_1 (таблица 3), заменяя состояния в таблице переходов исходного автомата (см. табл. 3.1) соответствующими классами одноэквивалентности.

Таблица 3

	B_1				B_2	
	a_1	a_2	a_5	a_6	a_3	a_4
z_1	B_2	B_2	B_1	B_1	B_2	B_2
z_2	B_1	B_1	B_1	B_1	B_1	B_1

По табл. 3 получим разбиение π_2 на классы 2-эквивалентных состояний (таблица 4): $\pi_2 = \{C_1, C_2, C_3\} = \{\overline{1.2},\ \overline{5.6},\ \overline{3.4}\}$.

Таблица 4

	C_1		C_2		C_3	
	a_1	a_2	a_5	a_6	a_3	a_4
z_1	C_3	C_3	C_2	C_2	C_3	C_3
z_2	C_2	C_2	C_1	C_1	C_2	C_2

Разбиение π_3 получаем аналогично (см. табл. 4): $\pi_3 = \{D_1, D_2, D_3\} = \{\overline{1.2},\ \overline{5.6},\ \overline{3.4}\}$.

Оно полностью совпадает с π_2. Процедура завершена. Разбиение $\pi_3 = \pi_2 = \pi$ есть разбиение множества состояний автомата Мили S_1 на классы эквивалентных между собой состояний.

2. Из каждого класса эквивалентности произвольно выбираем по одному состоянию: $A' = \{a_1, a_3, a_6\}$.

3. Строим таблицы переходов и выходов минимального автомата (таблица 5, 6, 7).

Таблица 5

	a_1	a_2	a_3	a_4	a_5	a_6
z_1	a_3	a_4	a_3	a_4	a_5	a_6
z_1	a_5 a_6	a_6	a_5 a_6	a_6	a_1	a_2 a_1

Таблица 6

	a_1	a_3	a_6
z_1	a_3	a_3	a_6
z_2	a_6	a_6	a_1

Таблица 7

	a_1	a_3	a_6
z_1	w_1	w_1	w_1
z_1	w_1	w_2	w_1

2.2. Задание к лабораторной работе

Минимизировать полный автомат Мили, используя метод Ауфенкампа и Хона.

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
z_1	1, 1	1, -	1, -	1, -	1, -	1, -	1, -	1, -
z_2	7, -	7, -	4, -	4, -	4, -	4, -	7 w_1	4, -
z_3	2, -	2,w_1	5, -	2, -	5, 1	5, -	5, -	5, -
z_4	6, -	3, -	3,w_2	3, -	3, -	6, 1	8, -	8,w_2

2.3. Контрольные вопросы

1. В чем заключается основная идея метода Метод Ауфенкампа и Хона?
2. В чем заключается задача минимизации?
3. Из каких шагов состоит алгоритм минимизации автомата Мили?
4. При каком условии два конечных автомата эквивалентны?
5. Что такое произведение автоматов?

3. ЛАБОРАТОРНАЯ РАБОТА №3: МИНИМИЗАЦИЯ АВТОМАТА МУРА. ВЗАИМОСВЯЗЬ АВТОМАТА МИЛИ И МУРА

3.1. Теоретическая часть

Автомат Мура (абстрактный автомат второго рода) в теории вычислений — конечный автомат, выходное значение сигнала в котором зависит лишь от текущего состояния данного автомата, и не зависит напрямую, в отличие от автомата Мили, от входных значений.

Минимизация автоматов Мура основана на тех же принципах, что и минимизация автоматов Мили. Для табличного описания эта процедура алгоритмизирована и состоит из трёх шагов.

Шаг 1 Распространение неопределённости выходов на таблицу переходов. Если в автомате Мура для некоторого состояния выходной сигнал не определён, то в это состояние он не может попасть под действием допустимого входного слова. Переход в таблице, соответствующий этому состоянию, исключается, а в остальных клетках таблицы переход в исключённое состояние заменяется специальным символом, например, прочерком.

Шаг 2 Исключение недостижимых состояний. Если нет ни одного слова, приводящего автомат в состояние a_i, отличающееся от начального, то такое состояние исключается вместе с соответствующими переходами таблицы переходов.

Шаг 3 Нахождение совместимых состояний автомата Мура. Состояния автомата Мура являются 0-совместимыми, если, не считая неопределённых отметок, они отмечены одинаковыми выходными сигналами. Состояния являются i - совместимыми для любого i = 1;2;... , если они 0-совместимы и автомат, переходя из них, перерабатывает допустимые слова длиной i одинаково. Про-

цедура расщепления классов обязательно закончится за ограниченное количество шагов и, следовательно, более нерасщепляющиеся классы образуют конечные классы совместимых состояний. После нахождения совместимых состояний автомата строится минимизированный нормализованный автомат Мура.

При минимизации полностью определённых автоматов Мура вводится понятие 0-эквивалентности состояний и разбиение множества состояний на 0-эквивалентные классы. 0-эквивалентными являются одинаково отмеченные состояния. Если два состояния автомата Мура 0-эквивалентны и под действием одинаковых входных сигналов попадают в 0-эквивалентные состояния, то они называются 1-эквивалентными. Все дальнейшие классы эквивалентности для автомата Мура определяются аналогично рассмотренному выше для автомата Мили.

Пример 1. Минимизировать полностью определённый автомат Мура S2, заданный отмеченной таблицей переходов (таблица 1)

Таблица 1 — Таблица переходов

	w_1	w_1	w_3	w_3	w_3	w_2	w_3	w_1	w_2	w_2	w_2	w_2
	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}
z_1	a_{10}	a_{12}	a_5	a_7	a_3	a_7	a_3	a_{10}	a_7	a_1	a_5	a_2
z_2	a_5	a_7	a_6	a_{11}	a_9	a_{11}	a_6	a_4	a_6	a_8	a_9	a_8

По таблице 1 находим разбиение π_0 на классы 0-эквивалентных состояний, отыскивая одинаково отмеченные состояния:

$$\pi_0 = \{A_1, A_2, A_3\} = \{\overline{1.2.8},\quad \overline{6.9.10.11.12},\quad \overline{3.4.5.7}\}.$$

Строим таблицу π_0 (таблица 2), заменяя состояния в таблице переходов (таблица 1) соответствующими классами 0-эквивалентности.

Таблица 2

	A_1			A_2					A_3			
	a_1	a_2	a_8	a_6	a_9	a_{10}	a_{11}	a_{12}	a_3	a_4	a_5	a_7
z_1	A_2	A_2	A_2	A_3	A_3	A_1	A_3	A_1	A_3	A_3	A_3	A_3
z_2	A_3	A_3	A_3	A_2	A_2	A_1	A_2	A_1	A_2	A_2	A_2	A_2

По таблица 2 получим разбиение π_1 на классы 1-

эквивалентных состояний (таблица 3):

$$\pi_1 = \{B_1, B_2, B_3, B_4\} = \{\overline{1.2.8},\ \overline{6.9.11},\ \overline{10.12},\ \overline{3.4.5.7}\}$$

Таблица 3

	B_1			B_2			B_3		B_4			
	a_1	a_2	a_8	a_6	a_9	a_{11}	a_{10}	a_{12}	a_3	a_4	a_5	A_7
z_1	B_3	B_3	B_3	B_4	B_4	B_4	B_1	B_1	B_4	B_4	B_4	B_4
z_2	B_4	B_4	B_4	B_2	B_2	B_2	B_1	B_1	B_2	B_2	B_2	B_2

Из таблице 3 видно, что $\pi_2 = \pi_1 = \pi$. То есть поиск классов эквивалентности завершён. Из каждого класса эквивалентности произвольно выбираем по одному элементу: $A' = \{a_1, a_6, a_{10}, a_3\}$. Строим отмеченную таблицу переходов минимального автомата S_2 (таблица 4).

Таблица 4

	w_1	w_3	w_2	w_2
	a_3	a_1	a_6	a_{10}
z_1	a_{10}	a_3	a_3	a_1
z_2	a_3	a_6	a_6	a_1

Пусть S_A – исходный автомат Мура; S_B – автомат Мили, который является целью преобразования. Автомат Мура характеризуется шестиэлементным множеством:

$S_A = (X_A, Q_A, Y_A, \delta_A, \lambda_A, q_{1A})$

Построим автомат Мили: $S_B = (X_B, Q_B, Y_B, \delta_B, \lambda_B, q_{1B})$. Четыре первые компоненты автомата Мили, а также начальное состояние определяются исходя из равенств:

$X_B = X_A$,

$Q_B = Q_A$,

$Y_B = Y_A$,

$\delta_B = \delta_A$,

$q_{1B} = q_{1A}$.

Различие состоит в функциях выходов. Они определяются так: если в автомате Мура $\delta_A(q_i, x_j) = q_s$ и $y_k = \lambda_A(q_s)$, то в автомате Мили $\lambda_B(q_i, x_j) = y_k$ (рисунок 1).

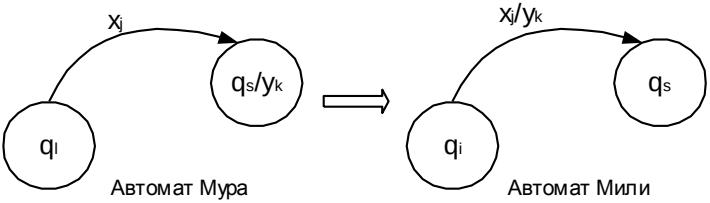


Рисунок 1 — Фрагмент преобразования автомата Мура в автомат Мили

В общем случае, если в вершину q_s входят несколько дуг, то выходной сигнал y_k, записанный в вершине q_s, переносится в обозначения всех дуг, входящих в данную вершину.

Преобразование автомата Мили в автомат Мура.

При этом преобразовании в графе автомата Мили не должно быть вершин, в которые не входит ни одна дуга, но которые в то же время имеют хотя бы одну выходящую дугу.

$S_A = (X_A, Q_A, Y_A, \delta_A, \lambda_A, q_{1A})$ – исходный автомат Мили;

$S_B = (X_B, Q_B, Y_B, \delta_B, \lambda_B, q_{1B})$ – целевой автомат Мура.

Для алфавитов автомата S_B будут справедливы следующие равенства:

$X_B = X_A$,

$Y_B = Y_A$.

Для определения множества Q_B каждому состоянию $q_s \in Q_A$ поставим в соответствие множество Q_s всевозможных пар вида (q_s, y_t), где y_t – выходной сигнал, приписанный дуге, входящей в q_s.(фрагмент графа изображен на рисунке 2).

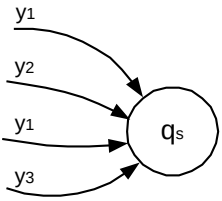


Рисунок 2 — Вершина q_s с входящими дугами

В этом случае Q_s представляет собой множество пар вида:

$Q_s = \{(q_s, y_l), (q_s, y_2), (q_s, y_3)\}$

В общем виде, если D – множество дуг, входящих в вершину q_s, то Q_s определяется следующим образом:

$Q_s = \{(q_s, y_t) | \ y_t \in D\}$

Итак, число элементов Q_s равно точному числу самых раз-

личных выходных сигналов при дугах автомата S_A, входящих в состояние q_s. Множество состояний автомата S_B получим как теоретико-множественное объединение множеств Q_s, ассоциированных со всеми состояниями q_s исходного автомата.

Вывод: число состояний в автомате Мура в среднем больше, чем число состояний в автомате Мили.

Функция λ_B определяется так: каждому состоянию автомата S_B, представляющему собой пару (q_s, y_t), ставится соответствие выходной сигнал y_t.

Функция δ_B определяется следующим образом: если в автомате Мили S_A есть переход $\delta_A(q_i, x_j) = q_s$ и при этом выдается выходной сигнал $\lambda_A(q_i, x_j) = y_p$, то в автомате Мура S_B будет переход из множества Q_i состояний, порожденных состоянием q_i , в состояние (q_s, y_p) под воздействием входного сигнала x_j (рисунок 3).

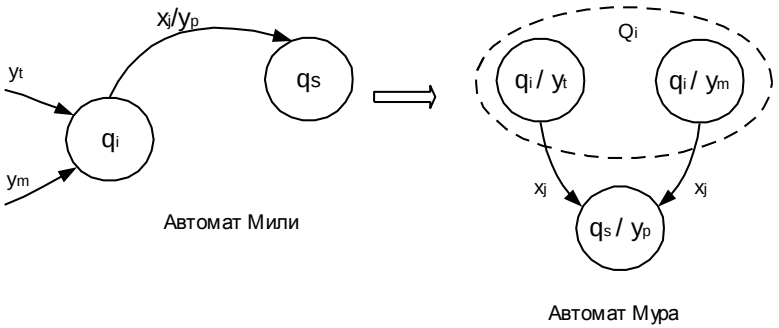


Рисунок 3 — Преобразование автомата Мили в автомат Мура

В качестве начального состояния q_{1B} можно взять любое состояние из множества Q_1, порожденного состоянием q_{1A}.

3.2. Задание к лабораторной работе

Минимизировать полный автомат Мура, используя метод Ауфенкампа и Хона. В таблице 5 приведены варианты перестановки столбцов для определения вариантов заданий.

Таблица 5

	w_2	w_1	w_1	w_2	w_2
	a_1	a_2	a_3	a_4	a_5
z_1	a_2	a_4	a_2	a_4	a_2
z_2	a_3	a_5	a_1	a_5	a_4

N вар	a_1	a_2	a_3	a_4	a_5
1	1	2	3	4	5
2	2	1	3	4	5
3	2	3	1	4	5
4	2	3	4	1	5
5	2	3	4	5	1
6	1	3	2	4	5
7	1	3	4	2	5
8	1	3	4	5	2
9	1	2	3	5	4
10	1	2	4	3	5
11	1	2	4	5	3
12	2	1	4	3	5
13	2	1	5	4	3
14	2	1	5	3	4
15	3	1	2	4	5
16	3	1	4	2	5
17	3	1	5	4	2
18	3	1	2	5	4

3.3. Контрольные вопросы

1. Что такое автомат Мура?
2. Что такое эквивалентность состояний?
3. В каком автомате (Мили или Мура) число состояний больше и почему?
4. Шаги минимизации автомата Мура?

4. ЛАБОРАТОРНАЯ РАБОТА №4: МАШИНА ТЬЮРИНГА

4.1. Теоретическая часть

Машина Тьюринга представляет собой устройство для выполнения алгоритмов преобразования информации. В теории алгоритмов машина Тьюринга используется как средство для описания алгоритмов. Считается, что если алгоритм решения некоторой задачи существует, то этот алгоритм может быть реализован на машине Тьюринга, т.е. для этого алгоритма может быть построена машина Тьюринга. С точки зрения теории цифровых автоматов машина Тьюринга (более строго - ее блок управления) представляет собой универсальный преобразователь информации.

Машина Тьюринга (МТ) может иметь структуру, показанную на рисунке 1.

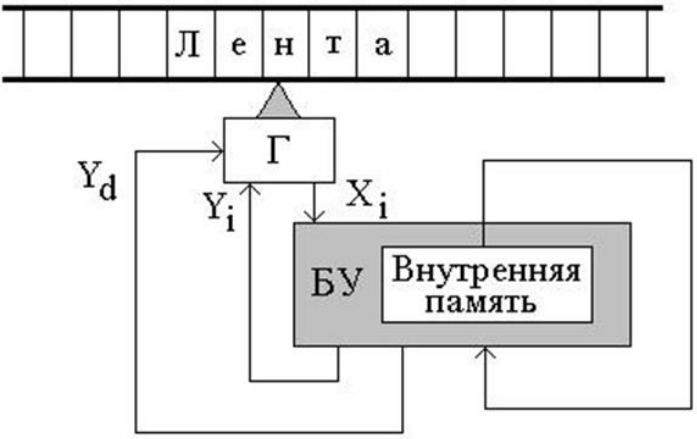


Рисунок 1 — Структура машины Тьюринга

В состав машины входят:

1. лента, на которой записаны исходные данные и на которую записываются результаты решения задачи;
2. головка записи/чтения информации (Г);
3. блок управления (БУ).

Лента состоит из отдельных ячеек. В каждую ячейку может быть записан символ из некоторого алфавита. На ленту предварительно записывается исходная информация. В процессе работы МТ с ленты с помощью головки считывается символ, находящийся над головкой (текущий символ X_i). При считывании информация в ячейке стирается. После считывания символа X_i в результате работы машины на данном шаге на ленту вместо X_i записывается новый символ Y_i. Лента считается бесконечной в одну или обе стороны и является внешней памятью машины. Головка служит для чтения и записи информации и с помощью привода может перемещаться вдоль ленты вправо или влево на одну ячейку на каждом шаге. В каждый момент времени для записи или чтения доступна только одна ячейка ленты.

Блок управления организует работу машины в целом. Он анализирует считываемую информацию и управляет записью символов Y_i на ленту, а также перемещением головки. Блок управления имеет внутреннюю память. Информация во внутренней памяти представляет собой состояние машины Тьюринга. Реакция машины на считанный символ X_i зависит не только от значения этого символа, но и от состояния машины, которое на каждом шаге ее работы может изменяться. Новое состояние машины определяется значением символа X_i и старым состоянием машины.

Перед началом работы на ленту наносится исходная информация. Головка устанавливается под ячейкой ленты, в которой записан первый символ. Машина переводится в начальное состояние. Процесс преобразования информации в машине Тьюринга состоит из отдельных шагов. На каждом шаге машина выполняет следующие элементарные операции:

1. чтение символа Xi из ячейки, под которой размещена головка;
2. анализ считанного символа в соответствии с алгоритмом решения задачи;
3. запись в ячейку вместо символа Xi нового символа Yi ;
4. перемещение головки на одну ячейку влево или вправо
5. переход машины в новое состояние (запись новой информации во внутрен-нюю память).

Машина Тьюринга представляет собой бесконечный авто-

мат, благодаря бесконечной (потенциально) ленте, разбитой на ячейки. В ячейках записы-ваются символы некоторого алфавита МТ. Имеется также конечный автомат с головкой записи и считывания (ГЗЧ). ГЗЧ обозревает одну ячейку ленты в текущий момент дискретного времени.

Функции ГЗЧ: считывание символа из обозреваемой ячейки; запись символа

в обозреваемую ячейку; передвижение влево или вправо на одну ячейку.

В каждый момент времени МТ описывается следующей пятеркой:

(qi, sj, δ(qi, sj), λ(qi, sj), d(qi, sj)),

где

qi – состояние МТ в текущий момент времени;

sj – обозреваемый символ в текущий момент времени;

δ – функция переходов, которая определяет следующее состояние;

λ – функция выходов, определяющая запись нового символа в обозреваемую ячейку;

d – функция, определяющая передвижение головки влево (L) или вправо (R) на один шаг.

Более краткое обозначение элементов пятерки: (qi , sj , qij , sij , dij).

Тезис Тьюринга: Любой процесс, который было бы естественно назвать эффективной процедурой может быть реализован МТ.

Следует подчеркнуть, что тезис – это, в общем случае, правдоподобное утверждение, которое не обязательно математически строго доказано. Однако многие научные тезисы действенны, так как прошли проверку временем и практикой.

Примеры машин Тьюринга для конкретных вычислений.

Счетчик четности единиц представлен на рисунке 2.

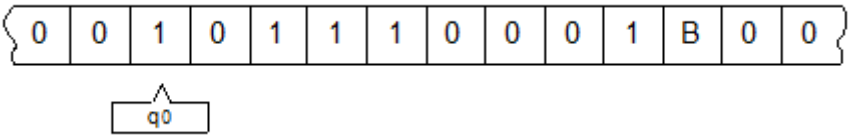


Рисунок 2 — МТ для счетчика четности

На рисунке 2 показана МТ в начальном состоянии q_0, при

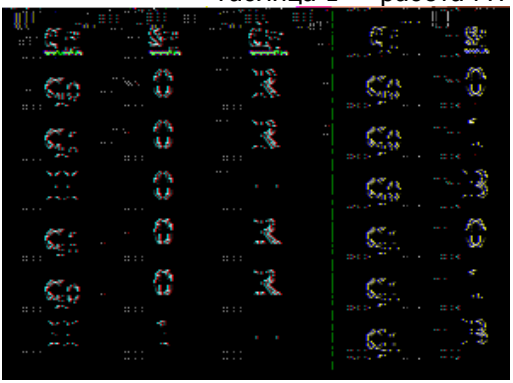
этом обозревается первый символ двоичной последовательности, которая заканчивается ограничителем B. Далее на каждом такте ГЗЧ продвигается вправо, заменяя все символы символом "0", причем смена состояний (q_0 на q_1 и обратно) происходит всякий раз, когда ГЗЧ обнаружит единицу. Таким образом, состояние q_0 вязано с четным числом обнаруженных единиц, а состояние q_1 – с нечетным. Останов МТ (переход в заключительное состояние H) происходит по достижении символа B, при этом на его место записывается "0", если число единиц в исходной последовательности было четным, и "1", если это число было нечетным.

После завершения процесса вычислений ГЗЧ будет указывать на эту ячейку с заключительной информацией, а во всех остальных ячейках ленты будут записаны нули.

Представим работу МТ двумя способами: таблицей (таблица 1) и графом (рисунок 3).

Строки таблицы содержат все "пятерки", описывающие функционирование МТ.

Таблица 1 — работа МТ



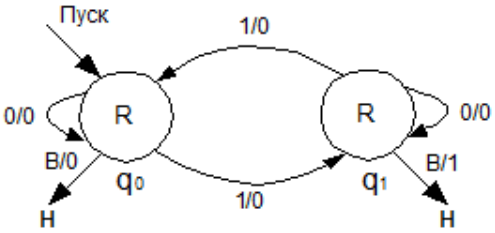


Рисунок 3 — Граф счетчика четности единиц

В вершинах графа явно указан символ, обозначающий движение ГЗЧ вправо в каждом из двух состояний. Заключительное состояние обозначено на рисунке буквой H – "halt" (останов).

2. Машина Тьюринга для проверки правильности скобочных выражений.

Заметим, что конечный автомат не может решить поставленную задачу для скобочного выражения произвольной длины. МТ решает задачу в общем случае, так как наличие неограниченной ленты эквивалентно неограниченному объему памяти.

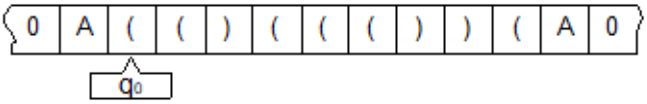


Рис. 17. МТ для проверки скобочных выражений

Скобочное выражение заключено между левым и правым ограничителями, обозначенными символом A. В начальном состоянии автомата ГЗЧ обозревает первый символ скобочного выражения (рисунок 4). Реализация вычислений представлена графом.

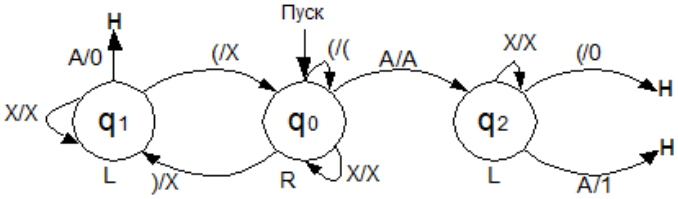


Рисунок 4 — Граф МТ для проверки скобочных выражений

Работа машины: сначала ГЗЧ движется вправо до первой правой скобки, заменяет ее символом X, переходит в состояние q_1 и движется влево до ближайшей левой скобки, заменяет ее символом X, переходит в состояние q_0, и челночное движение повторяется.

Если машина, находясь в состоянии q_1, достигает левого символа A, то печатает "0" и останавливается – скобочное выражение неправильно.

Если машина, находясь в состоянии q_0, достигает правого символа A, не обнаружив больше правой скобки, то переходит в заключительное состояние q_2, связанное с движением влево, и в

последний раз просматривает последовательность: не осталась ли непарная левая скобка. Если по пути встретится левая скобка, то машина печатает "0" и останавливается. При достижении в состоянии q_2 левого символа A, машина печатает "1" и останавливается – скобочное выражение правильно.

4.2. Задание к лабораторной работе

Поставнока задачи:

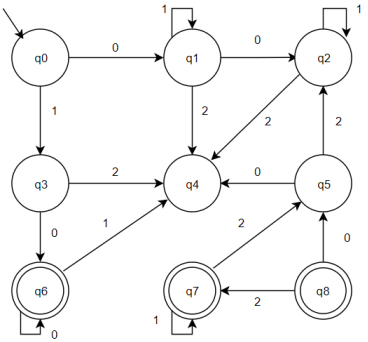
Разработать программное средство, реализующщее следующие функции.

1) вывод исходного автомата и вывод его на экран
2) устранение недостижимых состояние
3) построение классов эквивалентных состояний
4) вывод минимального автомата

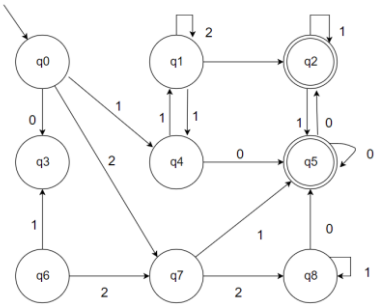
Так же всё прорешать вручную.

Условия по вариантам:

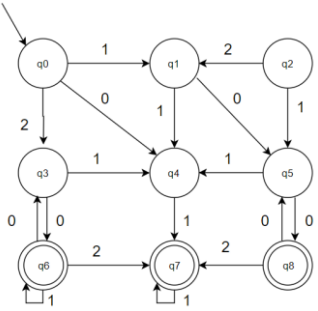
1)



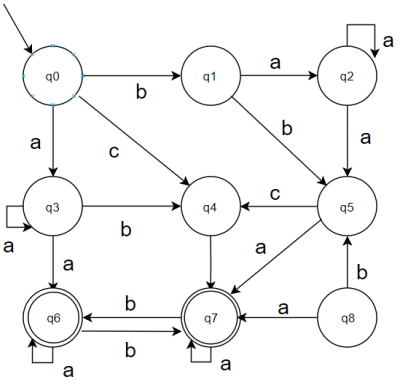
2)



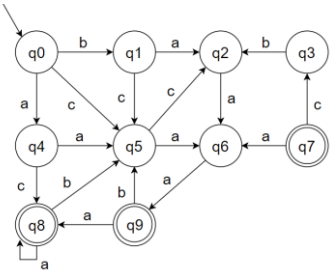
3)


q0
q1
q2
q3
q4
q5
q6
q7
q8

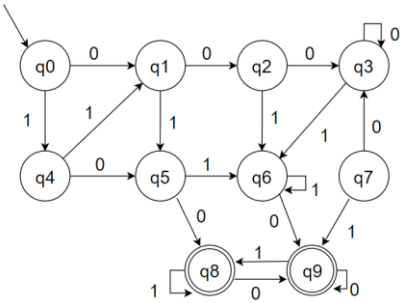
4)


q0
q1
q2
q3
q4
q5
q6
q7
q8

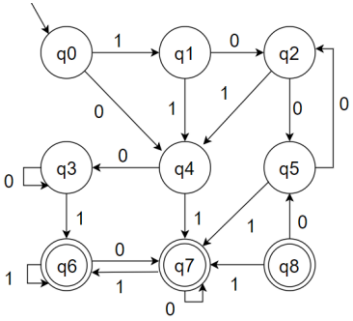
5)


q0
q1
q2
q3
q4
q5
q6
q7
q8
q9

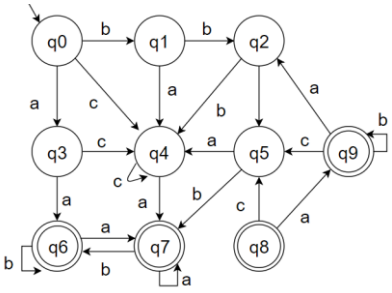
6)


q0
q1
q2
q3
q4
q5
q6
q7
q8
q9

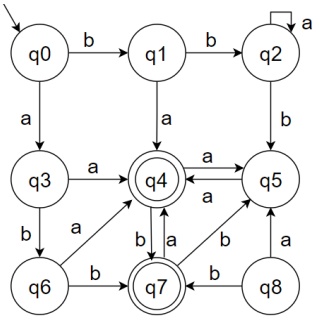
7)


q0
q1
q2
q3
q4
q5
q6
q7
q8

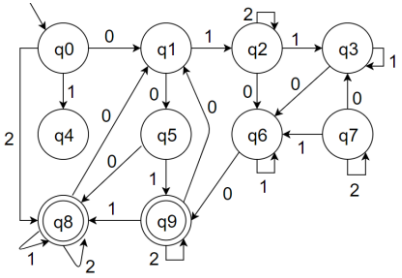
8)


q0
q1
q2
q3
q4
q5
q6
q7
q8
q9

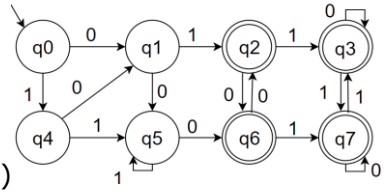
9)



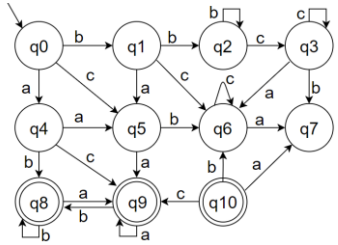
10)



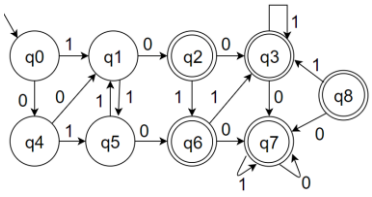
11)



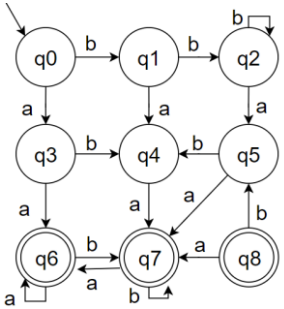
12)



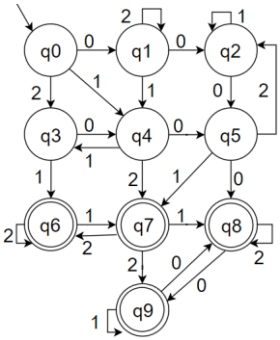
13)



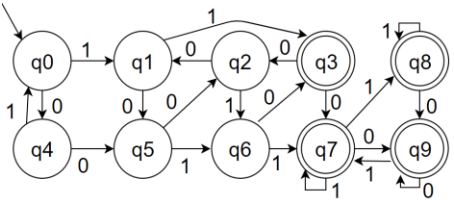
14)



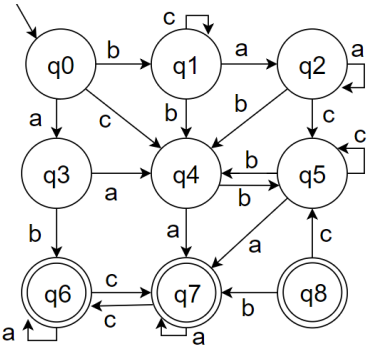
15)



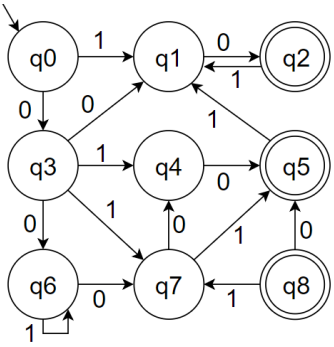
16)



17)



18)



4.3. Контрольные вопросы

1. Что из себя представляет машина Тьюринга?
2. Какова структура машины Тьюринга?
3. Опишите составные части машины Тьюринга.
4. Какие операции машина Тьюринга выполняет на каждом шаге процесса преобразования информации?

5. ЛАБОРАТОРНАЯ РАБОТА №5: РЕГУЛЯРНЫЕ МНОЖЕСТВА И РЕГУЛЯРНЫЕ ЯЗЫКИ. НАХОЖДЕНИЕ ЛЕКСИЧЕСКОГО НОМЕРА СЛОВА

5.1. Теоретическая часть

Язык L = <A, S> как множество лексем, т.е. $\alpha \in L \subset A^*$, можно задать перечислением (это возможно только для конечных языков) или описанием (с привлечением алгоритмических процедур порождения- распознавания языковых выражений в алфавите A).

Далее под порождающей процедурой понимается формальная грамматика, а распознающая процедура реализуется автоматом. Сказанное выше сведем в таблицу 1.

Таблица 1 — формальная грамматика

Тип языка по Хомскому	Порождающая грамматика Хомского <T,H,J,P>	Распознающий автомат <A, Q, R>
$L(\sigma_0)$	σ_0=<T,H, J, {$\xi \to \eta$}> $\xi, \eta \in (T \cup H)^*$	U_0 =<A,Q, {$q_i a_j \to q_e a_p d_k$}> $a_i, a_p \in H$; $q_i, q_i \in Q$, $d_k \in \{ d_п, d_н, d_п \}$
$L(\sigma_1)$	σ_1= < T,H,J,{x, Ax_2-$x_1\eta x_2$}> $A \in H, \eta \in (T \cup H)^*$	Автомат с линейно-ограниченной памятью
$L(\sigma_2)$	σ_2= <T,H,J,{$a \to \eta$}>, $A \in H, \eta \in (T \cup H)^*$	Магазинный автомат
$L(\sigma_3)$	σ_3= <T,H,J{$A \to aB, C-b$}>, a,b $\in$T, A,B,C, $\in$H	U_3= <A,Q, {$q_i, a_i \to q_e$}> $a_i \in A$; $q_i, q_i \in Q$

Очевидно, что классы языков L (σ_i) = { $\alpha \in T^*$: J | σ_i α} является иерархией , т.е. L (σ_3) $\subset$ L (σ_2) $\subset$ L (σ_1) $\subset$ L (σ_0).

Иерархией являются и распознающие автоматы.

Следуя приведенной таблице, можно говорить, что:

1) регулярный язык L (σ_3) = {$\alpha \in T^*$: J | σ_3 α}, σ_3 <T, H, J, P>, J $\in$ H, порождаемый грамматикой с конечным числом состояний σ_3 и распознаваемый конечным автоматом U_3, является самым простым в математическом плане;

2)бесконтекстный язык L (σ_2) порождаемый К.С. грамматикой σ_2, распознается магазинным автоматом U_2,(бесконечным автоматом, внутренняя структура которого представляет собой стековую память).

3)контекстный язык L (σ_1) распознается Л. О. автоматом U_1 (бесконечный автомат с линейно-ограниченной памятью).

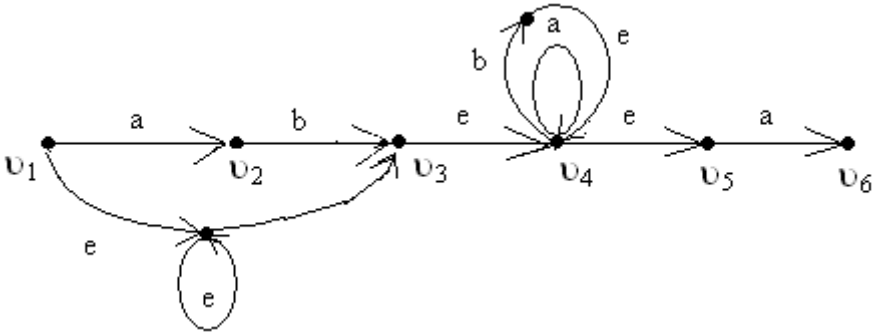
4)произвольный формальный язык, т.е. L (σ_0), распознается машиной Тьюринга

Примечание:

В алгебраической теории автоматов описанием регулярных языков (множеств, событий) являются формулы алгебры регулярных событий A_S= <S_E, $\cup$, $\cdot$,*> (здесь S_E – класс регулярных событий, а {$\cup$, $\cdot$,*} – регулярные операции). По заданному регулярно-

му множеству можно построить источник, частным случаем которого является графовое задание конечного автомата.

Пример: пусть R = (a·b +c*) · (a+b·c)*· a, построить источник, если R может быть записано в виде $R=R_1 \cdot R_2 \cdot R_3$, где $R_1=(a \cdot b +c^*)$, $R_2=(a + b \cdot c)^*$, $R_3 = a$:



по заданному регулярному выражению или заданному источнику можно построить регулярное множество. Для рассмотренного примера: $E=(\{a\} \cdot \{b\} \cup \{c\})^* \cdot (\{a\} \cup \{b\} \cdot \{c\})^*$

Одно и то же регулярное множество описывается различными (эквивалентными) регулярными выражениями (и источниками). Так, в примере, используя аксиомы полукольца, получаем другие эквивалентные формулы.

Два языка эквивалентны, если множества их лексем (то есть лексику) совпадают.

Две грамматики эквивалентны, если они порождают один и один и тот же язык.

Рассмотрим структуру рекурсивных определений, которые применяются как средство строгого математического описания классов объектов. Рекурсивное определение некоторого класса *K* включают следующие части:

База – определяет один или несколько простейших объектов класса *K*.

Рекурсия – состоит из одного или нескольких пунктов. В каждом пункте говорится, что если определенные виды объектов принадлежат классу *K*, то и другие объекты, построенные из первых по некоторому правилу, также принадлежат классу *K*. Число пунктов рекурсии соответствует числу правил.

Ограничение – устанавливает, что никакие объекты, кроме тех, которые построены посредством применения базы и рекур-

сии, не принадлежат классу *K*.

Регулярные языки задаются с помощью регулярных выражений, с помощью праволинейной грамматики, с помощью детерминированного и недетерминированного конечного автомата.

1.1. Регулярные множества и регулярные выражения

Пусть Σ – конечный алфавит. Регулярное множество в алфавите Σ определяется рекурсивно следующим образом:

∅ (пустое множество) – регулярное множество в алфавите Σ;

{e} – регулярное множество в алфавите Σ;

{a} – регулярное множество в алфавите для каждого a ∈ Σ;

если P и Q – регулярные множества в алфавите, то таковыми же являются и множества

(а) P ∪ Q;

(б) PQ;

(в) P*.

ничто другое не является регулярным множеством в алфавите Σ.

Регулярные выражения в алфавите Σ и регулярные множества, которые они обозначают, определяются рекурсивно следующим образом:

∅ – регулярное выражение, обозначающее регулярное множество ∅;

e – регулярное выражение, обозначающее регулярное множество {e},

если a ∈ Σ, то a – регулярное выражение, обозначающее регулярное множество {a},

если p и q – регулярные выражения, обозначающие регулярные множества P и Q соответственно, то:

(а) (p + q) – регулярное выражение, обозначающее P ∪ Q;

(б) (pq) – регулярное выражение, обозначающее PQ;

(в) (p)*– регулярное выражение, обозначающее P*.

ничто другое не является регулярным выражением.

Несколько примеров регулярных выражений и обозначаемых ими множеств:

1 обозначает {01};

0* обозначает {0}*;

(0+1)* обозначает {0, 1}*;

(0+1)*011 обозначает множество всех цепочек, составленных из нулей и единиц и оканчивающихся цепочкой 011;

(a+b) (a + b + 0 + 1)* обозначает множество всех цепочек

из {0, 1, a, b}*, начинающихся с буквы a или b.

1.2. Праволинейные грамматики

Формально грамматика задается четверкой G = (N, Σ, P, S), где N – конечное множество нетерминальных символов;

Σ – конечное множество терминальных символов, непересекающееся с N;

P – конечное множество правил;

S – начальный или исходный символ.

Различают регулярные грамматики с левосторонними продукциями вида A → Bx или A → Ax или A → x, и регулярные грамматики с правосторонними продукциями вида A → xB, или A → xA, или A → x, где A, B ∈ N, x ∈ Σ*.

1.3. Конечный автомат

Конечным автоматом называется пятерка A = (Q, Σ, ☐, q_0, F), где Q = {q_0, q_2, …, q_n}– конечное множество состояний устройства управления;

Σ = {a_1, a_2, …, a_k} – конечный входной алфавит;

☐ – функция переходов, отображающая Q во множество подмножеств множества Q, или другими словами, функция переходов по данному текущему состоянию и текущему входному символу указывает все возможные следующие состояния;

q_0 – начальное состояние устройства управления;

F – множество заключительных состояний, причем F ∈ Q.

Конечный автомат называется детерминированным, если множество ☐ (q, a) содержит не более одного состояния для любых q ∈ Q и a ∈ Σ. Если множество ☐ (q, a) содержит более одного состояния, то автомат называется недетерминированным.

1.4. Эквивалентность языков, определяемых праволинейной грамматикой и конечным автоматом. Рассмотрим теперь произвольную автоматную грамматику G = (N, Σ, P, S) с правосторонними продукциями:

C → xB, или C → x, где C, B ∈ N, x ∈ Σ*. Построим для нее недетерминированный конечный автомат:

A = (Q, Σ, ☐, q_0, F), где алфавиты Σ в грамматике и автомате совпадают, Q = N ∪ {D}, причем символ D не должен содержаться в N, q_0 = S, F = {D}, а ☐ определяется следующим образом:

1) каждой продукции вида C → x ставится в соответствие команда ☐ (C, x) = D;

2) каждой продукции вида C → xB ставится в соответствие команда ☐ (C, x) = B;

3) других команд нет.

Пример. Пусть язык задан регулярным выражением (a+b) (a + b + 0 + 1)*, грамматика G = (Q, Σ, ☐, q_0, F), порождающая строки языка задается следующим образом:

N = {S, L}, Σ = {a, b, 0, 1},

P = {S→aL, S→bL, S→a, S→b, L→aL, L→bL, L→0L, L→1L, L→a, L→b, L→0, L→1}.

Вывод строки a01b0 в данной грамматике будет выглядеть следующим образом:

S→ aL
→ a0L
→ a01L
→ a01bL
→ a01b0

Построим для грамматики G конечный автомат

A = (Q, Σ, ☐, q_0, F),

Σ = {a, b, 0, 1}, Q = {S, L} ∪ {D}, q_0 = S, F = {D},

(S, a) = L; (S, b) = L; (S, a) = D; (S, b) = D;

(L, a) = L; (L, 0) = L; (L, 0) = D;

(L, b) = L;

(L, 1) = L; (L, a) = D; (L, b) = D; (L, 1) = D;

Распознавание строки a01b0 данным автоматом будет выполнено следующим образом:

(S, a01b0) ⊢ (L, 01b0)
⊢ (L, 1b0)
⊢ (L, b0)
⊢ (L, 0)
⊢ (D, ε)

Теперь рассмотрим произвольный недетерминированный конечный автомат

A = (Q, Σ, ☐, q_0, F).

Такому автомату можно поставить в соответствие следующую автоматную грамматику:

G = (N, Σ, P, S), где алфавиты Σ в грамматике и автомате совпадают; N = Q, S = q_0, а множество P строится следующим образом:

1) каждой команде автомата (q_i, a_j) = q_k ставится в соответствие продукция $q_i \rightarrow a_j q_k$, если q_i, $q_k \in Q$, $a_j \in \Sigma$;

2) каждой команде (q_i, a_j) = q_k ставится в соответствие еще одна продукция $q_i \rightarrow a_j$, если $q_k \in F$;

3) других продукций нет.

Рассмотрим конечный детерминированный автомат A1, допускающий все цепочки из символов 0 и 1, которые начинаются и

оканчиваются 1. A1 = ({p, q, r}, {0, 1}, □, p, {r}), где □ задается следующими командами:

(p, 1) = q; (q, 0) = q; (q, 1) = r; (r, 0) = q; (r, 1) = r;

Этому автомату поставим в соответствие следующую автоматную грамматику:

G= (N, Σ, P, S)

Σ = {0, 1}, N = {p, q, r}, S = p,

P = {p→ 1q; q→ 0q; q→ 1r; q→ 1; r→ 0q; r→ 1r; r→ 1}

5.2. Задание к лабораторной работе

Без компьютера:

Опишите язык, определяемый данным регулярным выражением, приведите примеры строк, принадлежащих и не принадлежащих языку.

Для выбранного регулярного выражения определите конечный автомат, допускающий строки заданного языка.

Определите последовательность конфигураций автомата при распознавании строк, принадлежащих и не принадлежащих языку.

Определенному выше конечному автомату поставьте в соответствие праволинейную грамматику, порождающую строки заданного языка.

При помощи компьютера:

Напишите на языке программирования программу, моделирующую работу конечного автомата или праволинейной грамматики.

Варианты заданий регулярного языка:

№ вар.	Регулярное выражение
1	$(a^* + b^*)c$
2	$a(a+b)^*b$
3	$(aa+bb)^+$
4	$a^n b^m c^k$ где $n,m,k>0$
5	$(a+b)^* aa(a+b)^*$
6	$a^+ b^+ c^* b$
7	$((ba^*)(a^* b))^+$
8	$(10+1)^*(10)^+(1+10)^*$
9	$0(0+1)^*0+1(0+1)^*1$
10	$((1^*0)(1^*0)(1^*0))^+$

5.3. Контрольные вопросы

1. С помощью чего задаются регулярные языки?
2. Что включает в себя рекурсивное определение класса?
3. Какие языки называются эквивалентными?
4. Идентифицируйте элементы выражения A = (Q, Σ, ☐, q_0, F).

6. ЛАБОРАТОРНАЯ РАБОТА №6: СВОЙСТВА РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ

6.1. Теоретическая часть

Определим над множествами цепочек символов из алфавита V операции конкатенации и итерации следующим образом:

PQ - конкатенация $P \in V^*$ и $Q \in V^*$: $PQ = \{pq \ \forall p \in P, \forall q \in Q\}$;

P* - итерация $P \in V^*$: $P^* = \{p^* \ \forall p \in P\}$.

Тогда для алфавита V регулярные множества определяются рекурсивно:

1. $\varnothing$ — регулярное множество;
2. {λ} — регулярное множество;

3. {a} — регулярное множество $\forall a \in V$;

4. если P и Q — произвольные регулярные множества, то множества $P \cup Q,\ PQ$ и P^* также являются регулярными множествами;

5. ничто другое не является регулярным множеством.

Фактически регулярные множества — это множества цепочек символов над заданным алфавитом, построенные определенным образом (с использованием операций объединения, конкатенации и итерации). Все регулярные языки представляют собой регулярные множества.

Регулярные выражения. Свойства регулярных выражений.

Регулярные множества можно обозначать с помощью регулярных выражений. Эти обозначения вводятся следующим образом:

1. 0 — регулярное выражение, обозначающее $\varnothing$;

2. λ — регулярное выражение, обозначающее {λ};

3. a — регулярное выражение, обозначающее {a} $\forall a \in V$;

4. если p и q - регулярные выражения, обозначающие регулярные множества P и Q, то p + q, pq, p* - регулярные выражения, обозначающие регулярные множества $P \cup Q,\ PQ$ и P^* соответственно.

Два регулярных выражения α и β равны α = β, если они обозначают одно и то же множество.

Каждое регулярное выражение обозначает одно и только одно регулярное множество, но для одного регулярного множества может существовать сколь угодно много регулярных выражений, обозначающих это множество.

При записи регулярных выражений будут использоваться круглые скобки, как и для обычных арифметических выражений. При отсутствии скобок операции выполняются слева направо с учетом приоритета. Приоритет для операций принят следующий: первой выполняется итерация (высший приоритет), затем конкатенация, потом — объединение множеств (низший приоритет).

Если α, β и γ — регулярные выражения, то свойства регулярных выражений можно записать в виде следующих формул:

1.

$$\lambda + \alpha\alpha^* = \lambda + \alpha^*\alpha = \alpha^*(\lambda + \alpha^+ = \alpha^*)$$

2. $\alpha + \beta = \beta + \alpha$
3. $\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$
4. $\alpha(\beta + \gamma) = \alpha\beta + \alpha\gamma$
5. $(\beta + \gamma)\alpha = \beta\alpha + \gamma\alpha$
6. $\alpha(\beta\gamma) = (\alpha\beta)\gamma$
7. $\alpha + \alpha = \alpha$
8. $\alpha + \alpha^* = \alpha^*$
9. $\lambda + \alpha^* = \alpha^* + \lambda = \alpha^*$
10. $0^* = \lambda$
11. $0\alpha = \alpha 0 = 0$
12. $0 + \alpha = \alpha + 0 = \alpha$
13. $\lambda\alpha = \alpha\lambda = \alpha$
14. $(\alpha^*)^* = \alpha^*$

Все эти свойства можно легко доказать, основываясь на теории множеств, так как регулярные выражения — это только обозначения для соответствующих множеств.

Следует также обратить внимание на то, что среди прочих свойств отсутствует равенство $\alpha\beta = \beta\alpha$, то есть операция конкатенации не обладает свойством коммутативности. Это и не удивительно, поскольку для этой операции важен порядок следования символов.

Свойства регулярных языков

Множество называется замкнутым относительно некоторой

операции, если в результате выполнения этой операции над любыми элементами, принадлежащими данному множеству, получается новый элемент, принадлежащий тому же множеству. Например, множество целых чисел замкнуто относительно операций сложения, умножения и вычитания, но оно не замкнуто относительно операции деления — при делении двух целых чисел не всегда получается целое число. Регулярные множества (и однозначно связанные с ними регулярные языки) замкнуты относительно многих операций, которые применимы к цепочкам символов. Например, регулярные языки замкнуты относительно следующих операций:

- пересечения;
- объединения;
- дополнения;
- итерации;
- конкатенации;
- гомоморфизма (изменения имен символов и подстановки цепочек вместо символов).

Поскольку регулярные множества замкнуты относительно операций пересечения, объединения и дополнения, то они представляют булеву алгебру множеств. Существуют и другие операции, относительно которых замкнуты регулярные множества. Вообще говоря, таких операций достаточно много.

Проблемы, разрешимые для регулярных языков.

Регулярные языки представляют собой очень удобный тип языков. Для них разрешимы многие проблемы, неразрешимые для других типов языков. Например, доказано, что разрешимыми являются следующие проблемы:

- Проблема эквивалентности. Даны два регулярных языка L1(V) и L2(V). Необходимо проверить, являются ли эти два языка эквивалентными.
- Проблема принадлежности цепочки языку. Дан регулярный язык L(V) и цепочка символов $\alpha \in V^*$. Необходимо проверить, принадлежит ли цепочка данному языку.
- Проблема пустоты языка. Дан регулярный язык L(V). Необходимо проверить, является ли этот язык пустым, то есть найти хотя бы одну цепочку $\alpha \neq \lambda$, такую что .

Эти проблемы разрешимы вне зависимости от того, каким из трех способов задан регулярный язык. Следовательно, эти проблемы разрешимы для всех способов представления регуляр-

ных языков: регулярных множеств, регулярных грамматик и конечных автоматов. На самом деле достаточно доказать разрешимость любой из этих проблем хотя бы для одного из способов представления языка, тогда для остальных способов можно воспользоваться алгоритмами преобразования, рассмотренными выше.

Для регулярных грамматик также разрешима проблема однозначности — доказано, что для любой регулярной грамматики можно построить эквивалентную ей однозначную регулярную грамматику.

6.2. Задание к лабораторной работе

Используя описанные в теоретической части свойства регулярных выражений, привести доказательства описанных свойств при помощи теории множеств. Каждый студент должен доказать три свойства регулярных выражений, в соответствии со своим вариантом.

6.3. Контрольные вопросы

1. Для чего нужны регулярные выражения?
2. Дайте характеристику известным вам свойствам регулярных выражений?
3. Назовите свойства регулярных языков.
4. При каком условии регулярные выражения равны (эквивалентны)?

7. ЛАБОРАТОРНАЯ РАБОТА №7: ГРАММАТИКИ. ЯЗЫКИ. ГРАММАТИКИ ХОМСКОГО. КЛАССИФИКАЦИЯ ГРАММАТИК

7.1. Теоретическая часть

Цель работы:

- закрепить понятия «алфавит», «цепочка», «формальная грамматика» и «формальный язык», «выводимость цепочек», «эквивалентная грамматика»;
- сформировать умения и навыки распознавания типов формальных языков и грамматик по классификации Хомского, по-

строения эквивалентных грамматик.

Определение 1.1. Алфавитом V называется конечное множество символов.

Определение 1.2. Цепочкой α в алфавите V называется любая конечная последовательность символов этого алфавита.

Определение 1.3. Цепочка, которая не содержит ни одного символа, называется пустой цепочкой и обозначается ε.

Определение 1.4. Формальное определение цепочки символов в алфавите V:

1) ε - цепочка в алфавите V;

2) если α - цепочка в алфавите V и a – символ этого алфавита, то αa – цепочка в алфавите V;

3) β - цепочка в алфавите V тогда и только тогда, когда она является таковой в силу утверждений 1) и 2).

Определение 1.5. Длиной цепочки α называется число составляющих ее символов (обозначается | α |).

Обозначим через V* множество, содержащее все цепочки в алфавите V, включая пустую цепочку ε, а через V+ - множество, содержащее все цепочки в алфавите V, исключая пустую цепочку ε.

Пример 1.1. Пусть V = {1, 0}, тогда V* = {ε, 0,1, 00, 01,10,11, 000, K}, а V + = {0,1, 00, 01,10,11, 000, K}.

Определение 1.6. Формальной грамматикой называется четверка вида:

G = (V_T, V_N, P, S), (1.1) где V_N - конечное множество нетерминальных символов грамматики
(обычно прописные латинские буквы);

V_T - множество терминальных символов грамматики (обычно строчные латинские буквы, цифры, и т.п.), $V_T \cap V_N = \emptyset$;

P – множество правил вывода грамматики, являющееся конечным подмножеством множества $(V_T \cup V_N) + \times (V_T \cup V_N)^*$; элемент (α, β) множества P называется правилом вывода и записывается в виде α→β (читается: «из цепочки α выводится цепочка β»);

S - начальный символ грамматики, $S \in V_N$.

Для записи правил вывода с одинаковыми левыми частями вида

α →β_1, α →β_2, K, α →β n используется сокращенная форма записи

α →β1 | β 2 |K| β n

Пример 1.2. Грамматика G_1 = ({0, 1}, {A, S}, P1, S), где множество P_1 состоит из правил вида: 1) S→ 0A1; 2) 0A→ 00A1; 3)

A→ε.

Определение 1.7. Цепочка β ∈ (V_T ∪ V_N)* непосредственно выводима из цепочки α ∈ (V_T ∪ V_N)+ в грамматике G = (V_T , V_N , P, S) (обозначается: α⇒β), если α = ξ1γξ2 и β = ξ1δξ2, где ξ1, ξ2, δ ∈ (V_T ∪ V_N)*, γ ∈(V_T ∪ V_N)+ и правило вывода γ → δ содержится во множестве P.

Определение 1.8. Цепочка β ∈ (V_T ∪ V_N)* выводима из цепочки α ∈ (V_T ∪ V_N)+ в грамматике G = (V_T , V_N , P, S) (обозначается α⇒*β), если существует последовательность цепочек $γ_0$, $γ_1$, K,γ n (n≥0) такая, что α = $γ_0$ ⇒ $γ_1$ ⇒ K⇒ γ n = β . Пример 1.3. В грамматике G_1 S⇒*000111, т.к. существует вывод

S⇒ 0A1 ⇒ 00A11 ⇒ 000A111 ⇒ 000111.

Определение 1.9. Языком, порожденным грамматикой G = (V_T, V_N, P, S), называется множество всех цепочек в алфавите V_T, которые выводимы из начального символа грамматики S с помощью правил множества P, т.е. множество L(G) = {α ∈V_T* | S ⇒ *α}.

Пример 1.4. Для грамматики G_1 L(G_1) = {0n1n | n>0}.

Определение 1.10. Цепочка α ∈ (V_T ∪ V_N)* , для которой существует вывод S⇒*α, называется сентенциальной формой в грамматике G = (V_T , V_N, P, S) .

Определение 1.11. Грамматики G_1 и G_2 называются эквивалентными, если L(G_1) = L(G_2) .

Пример 1.5. Для грамматики G_1 эквивалентной будет грамматика G2 = ({0, 1}, {S}, P_2, S), где множество правил вывода P_2 содержит правила вида

S→ 0S1 | 01.

Классификация грамматик по Хомскому

Тип 0. Грамматика G = (V_T, V_N, P, S) называется грамматикой типа 0, если на ее правила вывода не наложено никаких ограничений, кроме тех, которые указаны в определении грамматики.

Тип 1. Грамматика G = (V_T, V_N, P, S) называется контекстно-зависимой грамматикой (КЗ-грамматикой), если каждое правило вывода из множества P имеет вид α→β, где α ∈ (VT ∪ VN)+, β ∈ (VT ∪ VN)* и |α| ≤ |β|.

Тип 2. Грамматика G = (V_T, V_N, P, S) называется контекстно-свободной грамматикой (КС-грамматикой), если ее правила вывода имеют вид: A → β, где A∈VN и β ∈V *.

Тип 3. Грамматика G = (V_T, V_N, P, S) называется регулярной грамматикой (Р-грамматикой) выровненной вправо, если ее правила вывода имеют вид A → aB | a, где a ∈VT ; A, B ∈ V_N .

Грамматика G = (V_T, V_N, P, S) называется регулярной грамматикой (Р-грамматикой) выровненной влево, если ее правила вывода имеют вид A → Ba | a, где a ∈ VT; A, B ∈V_N .

Определение 1.12. Язык L(G) называется языком типа k, если его можно описать грамматикой типа k, где k – максимально возможный номер типа грамматики.

Соотношение типов грамматик и языков представлено на рисунке 1.

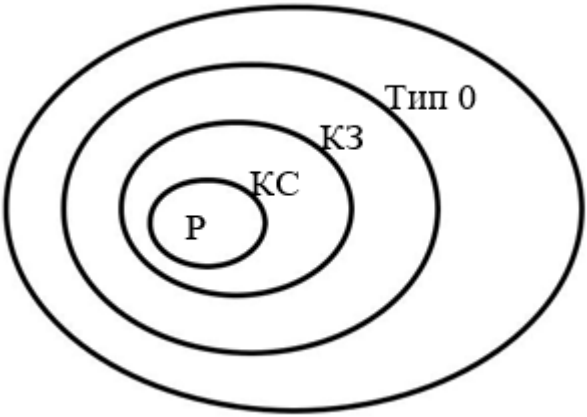


Рисунок 1 – Соотношение типов формальных языков и грамматик

Р – регулярная грамматика;

КС – контекстно-свободная грамматика; КЗ – контекстно-зависимая грамматика; Тип 0 – грамматика типа 0.

КС – контекстно-свободная грамматика; КЗ – контекстно-зависимая грамматика;

Тип 0 – грамматика типа 0.

Пример 1.6. Примеры различных типов формальных языков и грамматик по классификации Хомского. Терминалы будем обозначать строчными символами, нетерминалы – прописными буквами, начальный символ грамматики – S.

а) Язык типа 0 L(G) = {a2bn2 −1 | n ≥1} определяется грамматикой с правилами вывода:

1) S → aaCFD; 2) AD → D;
3) F → AFB | AB; 4) Cb → bC;
5) AB → bBA; 6) CB → C;
7) Ab → bA; 8) bCD → ε.

б) Контекстно-зависимый язык L(G) = {anbncn | n≥1} определяется грамматикой с правилами вывода:

1) S → aSBC | abc ; 2) bC → bc;

3) CB → BC; 4) cC → cc;

5) BB → bb.

в) Контекстно-свободный язык L(G) = {(ab)n(cb)n | n>0 } определяется грамматикой с правилами вывода:

1) S → aQb | accb;

2) Q → cSc.

г) Регулярный язык L(G) = {ω⊥ | ω ∈ {a, b}+, где нет двух рядом стоящих a} определяется грамматикой с правилами вывода:

1) S → A⊥ | B⊥;

2) A → a | Ba;

3) B → b | Bb | Ab.

7.2. Задание к лабораторной работе

При выполнении лабораторной работы следует реализовать следующие действия:

1) составить грамматику, порождающую формальный язык, заданный в соответствии с вариантом;

2) определить тип формальной грамматики и языка по классификации Хомского;

3) разработать программное средство, распознающее тип введенной пользователем грамматики по классификации Хомского.

Варианты индивидуальных заданий представлены в таблице 1.1.

Таблица 1.1

Вариант	Формальный язык
1	$L(G)=\{a^n b^m c^k \mid n, m, k>0\}$
2	$L(G)=\{(ab)^n (cb)^m \mid n, m\geq 0\}$
3	$L(G)=\{0^n (10)^m \mid n, m\geq 0\}$
4	$L(G)=\{wcwcw \mid w\in\{a, b\}^+\}$
5	$L(G)=\{c^{2n} d^n \mid n>0\}$
6	$L(G)=\{l+l\text{-}l \mid l\in\{a, b\}^+\}$
7	$L(G)=\{(10)^{n-1}(01)^{n+1} \mid n>0\}$
8	$L(G)=\{(ac)^n \mid n>0, a\in\{b, d\}, c\in\{+, -\}\}$
9	$L(G)=\{\perp(010)^n\perp \mid n>0\}$
10	$L(G)=\{a_1 a_2 \ldots a_n a_n \ldots a_2 a_1 \mid a_i\in\{0, 1\}\}$
11	$L(G)=\{a_1 a_2 \ldots a_n a_1 a_2 \ldots a_n \mid a_i\in\{c, d\}\}$
12	$L(G)=\{ab.b \mid a_i\in\{+, -\}, b\in\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}^+\}$

7.3. Контрольные вопросы

1. Что такое цепочка в алфавите? Что такое пустая цепочка и как она обозначается?
2. Какой вид имеет формальная грамматика?
3. Что из себя представляет язык, порождённый грамматикой G = (VT, VN, P, S)?
4. Назовите классификацию грамматик по Хомскому.

8. ЛАБОРАТОРНАЯ РАБОТА №8: ПОСТРОЕНИЕ И АНАЛИЗ КС-ГРАММАТИК.

8.1. Теоретическая часть

Контекстно-свободной грамматикой (КС-грамматикой) называется система G = (V_T, V_N, P, S), где V_T, V_N – непересекающиеся конечные множества терминальных и нетерминальных символов (терминалов и нетерминалов) соответственно; $S \in V_N$ – некоторый выделенный символ, называемый аксиомой грамматики, P – конечное множество правил. Также терминальные символы называются основными, нетерминальные — вспомогательными символами.

Каждое правило в КС-грамматике имеет вид $A \to \alpha$, где A

$\in V_N$ и $\alpha \in (V_N \cup V_T)^*$

Для слов α и β из $(V_T \cup V_N)^*$ будем говорить, что β непосредственно выводимо из α (и записывать $\alpha \Rightarrow \beta$), если α и β можно представить в виде $\alpha = \alpha_1 A \alpha_2$, $\beta = \alpha_1 \gamma \alpha_2$ для некоторых $\alpha_1, \alpha_2 \in (V_T \cup V_N)^*$, и в грамматике G имеется правило $A \to \gamma$. В этом случае говорят, что слово α получено из β применением правила $A \to \gamma$.

Рефлексивное транзитивное замыкание отношения $\Rightarrow$ назовем отношением выводимости и обозначим через $\Rightarrow^*$. Таким образом, $\alpha \Rightarrow^* \beta$, если слово β может быть выведено из слова α путем применения конечное число раз правил грамматики.

Язык, порождаемый грамматикой G, определим как множество всех слов в терминальном алфавите, выводимы из аксиомы грамматики S:

$L(G) = (\alpha : S \Rightarrow^* \alpha, \alpha, \alpha \in V_T^*)$. Язык L называется КС-языком, если существует КС-грамматика G такая, что L = L(G).

Пример 1.1.1. Пусть задана КС-грамматика. Пусть задана КС-грамматика $G_1 = /\{a,b\}, \{S\}, P, S \setminus$, где P состоит из правил:

$$\underset{(1)}{S \to aSbS} \mid \underset{(2)}{bSaS} \mid \underset{(3)}{\lambda} \ (\lambda - \text{пустое слово}).$$

Грамматика G_1 порождает множество всех слов в алфавите $\{a, b\}$, содержащих одинаковое число букв a и b.

Рассмотрим слово $\alpha = babbaa$ и процесс его вывода из аксиомы S:

$S \Rightarrow bSaS \Rightarrow baSbSaS \Rightarrow babSaS \Rightarrow babSa \Rightarrow babbSaSa \Rightarrow babbaa \in L$.

Последовательность номеров правил грамматики, с помощью которой слово выводится из аксиомы S, называется выводом слова.

Для $\alpha = babbaa$ рассмотренный вывод имеет вид $и(\alpha)$= 2133233. *Левым (правым) выводом* называется вывод, в котором каждое правило применяется к самому левому (самому правому) вхождению нетерминала в слово. $w_l(\alpha)$ = 2132333 - левый вывод слова α, $w_r(\alpha)$ = 2312333 - правый.

Для КС-грамматик существует удобный графический способ представления вывода – *дерево вывода*. Дерево строится следующим образом.

Корень дерева помечается аксиомой грамматики.

Если некоторая вершина дерева помечена нетерминалом *A*, и в процессе вывода к этому нетерминалу применяется правило $A \Rightarrow b_{i1}\ b_{i2}\ b_{ik}$, то из вершины в следующий ярус проводится k ребер, и полученные вершины следующего яруса помечаются слева направо буквами b_{i1} , b_{i2} , , b_{ik}. При применении правил вида $A \Rightarrow \lambda$ из вершины, помеченной нетерминалом *A*, в следующий ярус проводится одно ребро и новая вершина помечается символом пустого слова λ.

Для слова $\alpha \in L(G_1)$ листья дерева помечены символами терминального алфавита либо символом λ. Само слово α получается обходом кроны дерева слева направо. На рисунке 1 для грамматики G_1 изображено дерево вывода слова $\alpha = babbaa$.

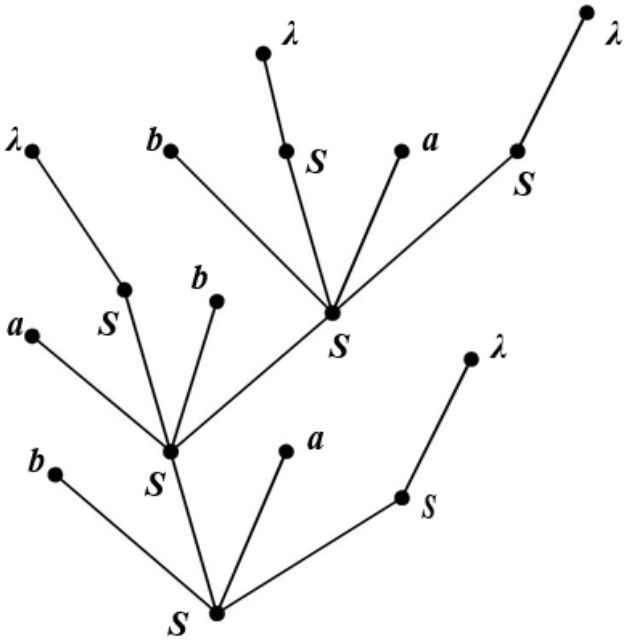


Рисунок 1 — Дерево вывода для слова $\alpha = babbaa$

КС-грамматика G называется грамматикой с однозначным выводом (или однозначной грамматикой), если каждое слово из $L(G)$ имеет единственный левый вывод. В противном случае КС-грамматика называется грамматикой с неоднозначным выводом.

Пример 1.1.2. Рассмотрим грамматику $G_2 = /\{ a, +, *, (,$

$)\{\{,E\}, P, E\backslash$, где P содержит следующие четыре правила:

$(1)\quad E \to E + E$

$(2)\quad E \to E * E$

$(3)\quad E \to (E)$

$(4)\quad E \to a.$

Слово $\alpha = a + a * a$ из $L(G_2)$ имеет два различных левых вывода $w_l(\alpha) = 14244$ и $w_l'(\alpha) = 21444$, и соответственно два различных дерева вывода. Поэтому G_2 - грамматика с неоднозначным выводом, представленном на рисунок 2.

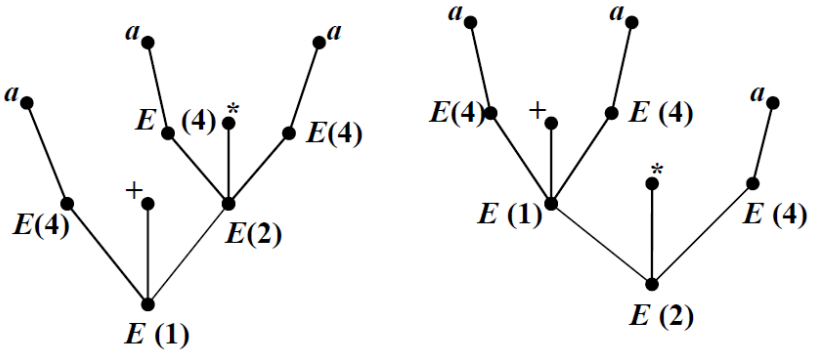


Рисунок 2 — Деревья вывода для слова $\alpha = a + a * a$

Пример 1.1.3. Пусть $L = \{a^n b^n c^k \mid n, k \geq 0\}$. Требуется построить КС-грамматику G_3, для которой $L = L(G_3)$. Введем два нетерминальных символа A и B. Нетерминал A будем использовать для порождения подслова $\alpha = a^n b^n$ с помощью правила $A \to aAb$; последовательно применяя это правило n раз, получим слово $a^n Ab^n$. Добавим правило $A \to \lambda$, чтобы иметь возможность удалить нетерминал A из $a^n Ab^n$. Нетерминал B будем использовать для порождения $\alpha_2 = c^k$, последовательно применяя k раз правило $B \to cB$. Для удаления нетерминала B добавим правило $B \to \lambda$. Чтобы получить слово $\alpha = \alpha_1\alpha_2 \in L$, добавим аксиому грамматики S и правило $S \to AB$. Заметим, что пустое слово принадлежит L, его можно получить так:

$$S \Rightarrow AB \Rightarrow \lambda B \Rightarrow \lambda\lambda = \lambda.$$

Таким образом, мы построили грамматику $G_3 = /\{a,b,c\}\{,S, A, B\}, P, S\backslash$, где P состоит из следующих правил:

$S \to AB$

$A \to aAb \mid \lambda$

$B \to cB \mid \lambda.$

8.2. Задание к лабораторной работе

В задачах 1.1.1 – 1.1.21 построить контекстно-свободную грамматику, порождающую заданный язык.

1.1.1. $L = \left\{0^i 1^j \mid i \geq j\right\}.$

1.1.2. $L = \left\{0^i 1^j \mid i < j\right\}.$

1.1.3. $L = \left\{0^i 1^j 2^k \mid i, j, k \geq 1,\ i = j\right\}.$

1.1.4. $L = \left\{0^k 1^i 2^k \mid i, k \geq 1\right\}.$

1.1.5. $L = \left\{\alpha\alpha^R \mid \alpha \in \{a,b\}^*\right\}$, где α^R - обращение слова α.

1.1.6. $L = \left\{\alpha c\alpha^R \mid \alpha \in \{a,b\}^*\right\}.$

1.1.7. $L = \left\{\alpha\alpha^R\beta\beta^R \mid \alpha, \beta \in \{a,b\}^*\right\}.$

1.1.8. $L = \left\{\alpha c\alpha^R c\beta c\beta^R \mid \alpha, \beta \in \{a,b\}^*\right\}.$

1.1.9. $L = \left\{\alpha \mid \alpha \in \{a,b\}^*, |\alpha|_a = |\alpha|_b\right\}.$

1.1.10. $L = \left\{\alpha \mid \alpha \in \{a,b\}^*, |\alpha|_a < |\alpha|_b\right\}.$

1.1.11. $L = \left\{\alpha \mid \alpha \in \{a,b\}^*, |\alpha|_a \geq |\alpha|_b\right\}.$

1.1.12. $L = \left\{\alpha \mid \alpha \in \{a,b\}^*, |\alpha|_a \neq |\alpha|_b\right\}.$

1.1.13. $L = \left\{0^i 1^j 2^{2(i+j)} \mid i, j \geq 0\right\}.$

1.1.14. $L = \{0^i 2^{3(i+j)} 1^j \mid i, j \geq 0\}$.

1.1.15. $L = \{2^{2(i+j)} 0^i 1^j \mid i, j \geq 0\}$.

1.1.16. $L = \{0^i 1^j \mid i \leq j \leq 2i\}$.

1.1.17. $L = \{0^i 1^j \mid j \leq i \leq 2j\}$.

1.1.18. $L = \{0^i 1^j 2^k \mid i = 2j \text{ или } j = 2k\}$.

1.1.19. $L = \{0^i 1^j 2^k \mid i = 2j \text{ или } k = 2j\}$.

1.1.20. $L = \{0^i 1^j 2^k \mid i = j \text{ или } j \neq 2k\}$.

1.1.21. $L = \{0^i 1^j 2^k \mid i \geq 2j \text{ или } j \geq 2k\}$.

1.1.22 Пусть $G = \{a,b\}\{,S, A, B\}, P, S$ — КС-грамматика, где P состоит из правил:

$S \to AB$

$A \to Aa \mid bB$

$B \to a \mid Sb$

Для слова a, принадлежащего $L(G)$, построить дерево вывода, левый и правый выводы.

а) $\alpha = baabaab$;

б) $\alpha = bbaaaba$;

в) $\alpha = babaabb$.

1.1.23 Пусть $G = \{a, +,*, (,)\}\{, E,T , F\}, P, E$ — КС-грамматика, где P состоит из правил:

$E \to E + T \mid T$

$T \to T * F \mid F$

$F \to (E) \mid a$

Грамматика G порождает множество арифметических выражений, содержащих переменную a, знаки арифметических операций +, * и правильно расставленные скобки.

Для слова a, принадлежащего $L(G)$, построить дерево вывода, левый и правый выводы.

а) $\alpha = a + a *(a + a + a)$;

б) $\alpha = (a * (a + a)+ a) * a$;

в) $\alpha = a * a + (a * a + a)* a$;

г) $\alpha = a * a *(a + a + a)$;
д) $\alpha = (a + a)*(a + a * a)$;
е) $\alpha = (a + a *(a + a))* a$.

8.3. Контрольные вопросы

1. Что называется контекстно-свободной грамматикой?
2. Что такое отношение выводимости и как оно обозначается?
3. Что такое левый (правый) вывод? Какие отличие между грамматикой с однозначным и неоднозначным выводом?
4. Какие вы знаете способы графического представления КС-грамматик?

СПИСОК ЛИТЕРАТУРЫ

1. Джон Хопкрофт, Раджив Мотвани, Джеффри Ульман. Введение в теорию автоматов, языков и вычислений (Introduction to Automata Theory, Languages, and Computation). — М.: «Вильямс», 2002. — С. 528. — ISBN 0-201-44124-1.

2. Белоусов А. И., Ткачев С. Б. Дискретная математика. — М.: МГТУ, 2006. — 743 с. — ISBN 5-7038-2886-4.

3. Джон Хопкрофт, Раджив Мотвани, Джеффри Ульман. Глава 8. Введение в теорию машин Тьюринга // Введение в теорию автоматов, языков и вычислений = Introduction to Automata Theory, Languages, and Computation. — М.: Вильямс, 2002. — 528 с. — ISBN 0-201-44124-1.

4. Карпов Ю. Г. Теория автоматов. — Питер, 2003. — ISBN 5-318-00537-3.

5. Гладкий А. В. Формальные грамматики и языки. — М.: Наука, 1973. — 368 с.

6. Кревский И. Г., Селивёрстов М. Н., Григорьева К. В. Формальные языки, грамматики и основы построения трансляторов: Учебное пособие / Под ред. А. М. Бершадского. — Пенза: Изд-во Пенз. гос. ун-та, 2002. — 124 с.