МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Белгородский государственный технологический университет

им. В.Г. Шухова»

ИНСТИТУТ ЭНЕРГЕТИКИ, ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

И УПРАВЛЯЮЩИХ СИСТЕМ

Кафедра программного обеспечения вычислительной

техники и автоматизированных систем

Лабораторная работа №3

по дисциплине: Теория алгоритмов и формальных языков

тема: «Регулярные языки и конечные распознаватели»

|  |
| --- |
| Выполнила:  Ст. группы ПВ-31  Ковалев Павел |
| Проверил:  Рязанов Ю. Д.  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.  Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Белгород 2021

**Цель работы:** изучить основные способы задания регулярных языков, способы построения, алгоритмы преобразования, анализа и реализации конечных распознавателей.

**Задания к работе**

**Задание №1**

Построить минимальный детерминированный конечный распознаватель заданного языка (смотреть варианты заданий).

**Задание №2**

Написать программу-распознаватель компиляционного и интерпретационного типа.

Исходные данные: строка.

Результат:

«Допустить» - если строка представляет собой цепочку заданного языка

«Отвергнуть» - в противном случае.

**Задание №3**

Написать программу, которая оставляет в исходном текстовом файле только те строки, которые представляют собой цепочки заданного языка.

**Задание №4**

Написать программу, которая исключает из исходного текстового файла строки, являющиеся цепочками заданного языка.

**Вариант 2**

**Задача**

Язык строковых констант.

Строковая константа – последовательность символов, которая заключена в кавычки.

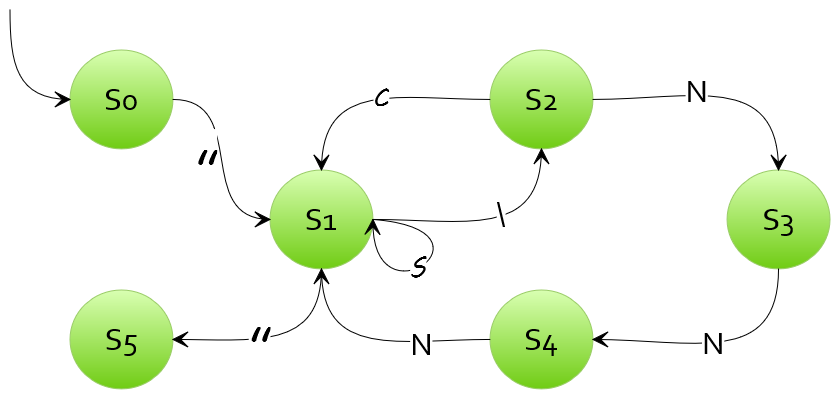
Последовательности символов «\n», «\t», «\b», «\r», «\f», «\’», «\”», «\\» и символ «\», за которым следует трехразрядное восьмеричное число, считаются одним символом, а одиночный символ «\» недопустим

**Выполнение**

**Задание №1**

Введем следующие обозначения:

* S – множество символов, включающее в себя различные буквы и цифры (за исключением символов «\» и «″»);
* C – символы преобразования «n», «t», «b», «r», «f», «’», «”», «\»;
* N – цифры в диапазоне от 0 до 7.



Исходный распознаватель, построенный по графу.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | ↓ |
|  | S0 | S1 | S2 | S3 | S4 | S5 |
| ″ | S1 | S5 |  |  |  |  |
| S |  | S1 |  |  |  |  |
| \ |  | S2 |  |  |  |  |
| C |  |  | S1 |  |  |  |
| N |  | S1 | S3 | S4 | S1 |  |

Произведем минимизацию полученного распознавателя. Построим таблицу переходов для класса 0-эквивалентных состояний.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | K1 | | | | | | K2 |
|  | S0 | S1 | S2 | S3 | S4 |  | S5 |
| ″ | K1 | K2 | K1 | K1 | K1 | K1 | K1 |
| S | K1 | K1 | K1 | K1 | K1 | K1 | K1 |
| \ | K1 | K1 | K1 | K1 | K1 | K1 | K1 |
| C | K1 | K1 | K1 | K1 | K1 | K1 | K1 |
| N | K1 | K1 | K1 | K1 | K1 | K1 | K1 |

Построим таблицу переходов для класса 1-эквивалентных состояний.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | K1 | | | | | K2 | | K3 |
|  | S0 | S2 | S3 | S4 |  | | S1 | S5 |
| ″ | K2 | K1 | K1 | K1 | K1 | | K3 | K1 |
| S | K1 | K1 | K1 | K1 | K1 | | K2 | K1 |
| \ | K1 | K1 | K1 | K1 | K1 | | K1 | K1 |
| C | K1 | K2 | K1 | K1 | K1 | | K1 | K1 |
| N | K1 | K1 | K1 | K2 | K1 | | K2 | K1 |

Построим таблицу переходов для класса 1-эквивалентных состояний.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | K1 | K2 | K3 | K4 | K5 | K6 | K7 |
|  | S0 | S2 | S3 | S4 |  | S1 | S5 |
| ″ | K6 | K5 | K5 | K5 | K5 | K7 | K5 |
| S | K5 | K5 | K5 | K5 | K5 | K6 | K5 |
| \ | K5 | K5 | K5 | K5 | K5 | K2 | K5 |
| C | K5 | K6 | K5 | K5 | K5 | K5 | K5 |
| N | K5 | K3 | K4 | K6 | K5 | K6 | K5 |

Нет смысла продолжать, т. к. на следующем шаге мы получим то же самое количество классов эквивалентности, что и количество состояний в исходной таблице (после того, как из таблицы минимизации будет удален пустой столбец).

**Задание №2**

(программа-распознаватель компиляционного и интерпретационного типа)

Модуль header.h

#include <algorithm>

#include <iostream>

#include <iterator>

#include <fstream>

#include <conio.h>

#include <string>

#include <vector>

#include <list>

using namespace std;

#pragma warning(disable: 4996)

bool IsLetter(char ch);

// S – множество символов, включающее в себя различные буквы и цифры (за исключением символа «\»);

bool symb\_is\_S(char ch);

// C – символы преобразования «n», «t», «b», «r», «f», «’», «”», «\»;

bool symb\_is\_C(char ch);

// N – цифры в диапазоне от 0 до 7.

bool symb\_is\_N(char ch);

//считывание данных из файла

unsigned input\_data(std::string const& fName, std::list <std::string>& dataList);

//удаление строки

bool remove\_line(const char\* filename, size\_t index);

//------------Компиляционный распознователь--------------------//

bool recognizer\_compilation\_mode(std::string const& str);

//-------------------------------------------------------------//

//-----------Интерпретанционный распознователь-----------------//

bool recognizer\_interpretation\_mode(std::string const& str);

void prepareForInterpret(int\*\*\* T, int\* m, int\* n);

int F(char x);

//-------------------------------------------------------------//

#pragma once

Модуль Main.cpp

#include <iostream>

#include "header.h"

int main()

{

std::list<std::string> \_dataList;

unsigned \_n;

int num\_str = 0;

if ( (\_n = input\_data("C:/test/data.txt", \_dataList)) != 0)

{

std::list<std::string>::const\_iterator \_ci = \_dataList.cbegin();

for (; \_ci != \_dataList.cend(); \_ci++)

{

//нужное раскоментировать

// =>компилятор

bool flag = recognizer\_compilation\_mode(\*\_ci);

// =>интерпретатор

//bool flag = recognizer\_interpretation\_mode(\*\_ci);

num\_str++;

std::cout << "|--------------------------------------|"

<< std::endl << "\tstring: "

<< std::string(\*\_ci) << std::endl

<< "\tstatus: " << (flag ? "TRUE" : "FALSE")

<< std::endl;

}

}

else std::cout << "ERROR! FILE IS EMPTY!" << std::endl;

\_getch();

return 0;

}

bool IsLetter(char ch)

{

return ((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z'));

}

bool symb\_is\_S(char ch) { return (IsLetter(ch) && ch != '"'); }

bool symb\_is\_C(char ch) {

return (ch == 'n' || ch == 't' || ch == 'b' ||

ch == 'r' || ch == 'f' || ch == '\''||

ch == '"' || ch == '\\');

}

bool symb\_is\_N(char ch) { return (ch >= '0' && ch <= '7'); }

unsigned input\_data(std::string const& fName, std::list<std::string>& dataList)

{

std::ifstream \_fin;

std::string \_buf;

\_fin.open(fName.c\_str());

unsigned \_n;

if (\_fin) {

\_fin >> \_n;

std::getline(\_fin, \_buf);

for (unsigned i = 0; i < \_n; i++) {

std::getline(\_fin, \_buf);

dataList.push\_back(\_buf);

}

\_fin.close();

}

else return 0;

return \_n;

}

//------------Компиляционный распознователь--------------------//

bool recognizer\_compilation\_mode(std::string const& str)

{

unsigned \_condition = 0;

bool \_state = true;

for (unsigned i = 0; i < str.size(); i++)

{

switch (\_condition)

{

//=================================== condition\_0

case 0: {

if (str[i] == '"') \_condition = 1;

else \_state = false;

} break;

//=================================== condition\_1

case 1: {

if (symb\_is\_S(str[i])) \_condition = 1;

else if (str[i] == '"') \_condition = 5;

else if (str[i] == '\\') \_condition = 2;

else \_state = false;

} break;

//=================================== condition\_2

case 2: {

if (symb\_is\_C(str[i])) \_condition = 1;

else if (symb\_is\_N(str[i])) \_condition = 3;

else \_state = false;

} break;

//=================================== condition\_3

case 3: {

if (symb\_is\_N(str[i])) \_condition = 4;

else \_state = false;

} break;

//=================================== condition\_4

case 4: {

if (symb\_is\_N(str[i])) \_condition = 1;

else \_state = false;

} break;

//=================================== condition\_5

case 5: \_state = ((i + 1) == str.size());

}

if ((\_state == false) || (\_condition == 5)) break;

}

return \_state && (\_condition == 5);

}

//-----------Интерпретанционный распознователь-----------------//

bool recognizer\_interpretation\_mode(std::string const& str)

{

unsigned \_condition = 0;

bool \_state = true;

int n, m; int\*\* T;

int s = 0, spred = 0, k = 0;

int i = -1;

char x;

prepareForInterpret(&T, &m, &n);

do {

x = str[++i];//берем из строки s первый символ

x = str[++i];//берем из строки s первый символ

if (F(x) != -1) s = T[F(x)][spred = s];

} while ((s >= 0) && (x != '\0'));

return s == 6;

}

void prepareForInterpret(int\*\*\* T, int\* m, int\* n)

{

int i, j;

FILE\* f = fopen("C:/test/res.txt", "r");

fscanf\_s(f, "%i %i", &(\*m), &(\*n));

\*T = new int\* [\*m];

for (i = 0; i < \*m; ++i) {

(\*T)[i] = new int[\*n];

for (j = 0; j < \*n; ++j) {

fscanf\_s(f, "%i", &((\*T)[i][j]));

}

}

}

int F(char x) {

if (x == '"') return 0;

if (x == '\\') return 2;

if (symb\_is\_C(x)) return 3;

if (symb\_is\_N(x)) return 4;

if (symb\_is\_S(x)) return 1;

return -1;

}

|  |  |
| --- | --- |
| Результат компиляционного распознавателя | Результат интерпретационного распознавателя |
| Оба распознавателя работают верно, следовательно, программа-распознаватель корректна. | |

**Задание №3**

**(**программа, которая оставляет в исходном текстовом файле только те строки, которые представляют собой цепочки заданного языка)

**Задание №4**

(программа, которая исключает из исходного текстового файла строки, являющиеся цепочками заданного языка)

Программа для заданий №3 и №4 полностью аналогична программе предыдущего задания, за исключением того, что добавляется функция удаления необходимой строки из исходного файла.

bool remove\_line(const char\* filename, size\_t index)

{

std::vector<std::string> vec;

std::ifstream file(filename);

if (file.is\_open())

{

std::string str;

while (std::getline(file, str))

vec.push\_back(str);

file.close();

if (vec.size() < index)

return false;

vec.erase(vec.begin() + index);

std::ofstream outfile(filename);

if (outfile.is\_open())

{

std::copy(vec.begin(), vec.end(),

std::ostream\_iterator<std::string>(outfile, "\n"));

outfile.close();

return true;

}

return false;

}

return false;

}

int main()

{

std::list<std::string> \_dataList;

unsigned \_n;

int num\_str = 0;

if ( (\_n = input\_data("C:/test/data1.txt", \_dataList)) != 0)

{

std::list<std::string>::const\_iterator \_ci = \_dataList.cbegin();

for (; \_ci != \_dataList.cend(); \_ci++)

{

bool flag = recognizer\_compilation\_mode(\*\_ci);

//bool flag = recognizer\_interpretation\_mode(\*\_ci);

num\_str++;

std::cout << "|--------------------------------------|"

<< std::endl << "\tstring: "

<< std::string(\*\_ci) << std::endl

<< "\tstatus: " << (flag ? "TRUE" : "FALSE")

<< std::endl;

//**Необходимое раскоментировать**

//Задание №3

//if (!flag)

//Задание №4

if (flag!=0)

{

remove\_line("C:/test/data1.txt", num\_str);

num\_str--;

}

}

}

else std::cout << "ERROR" << std::endl;

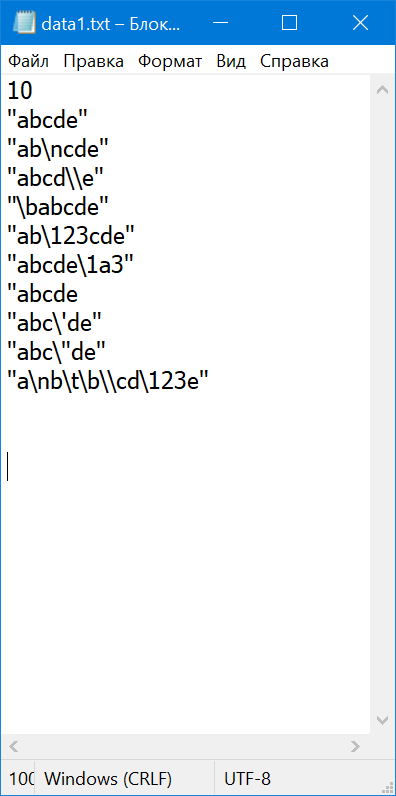
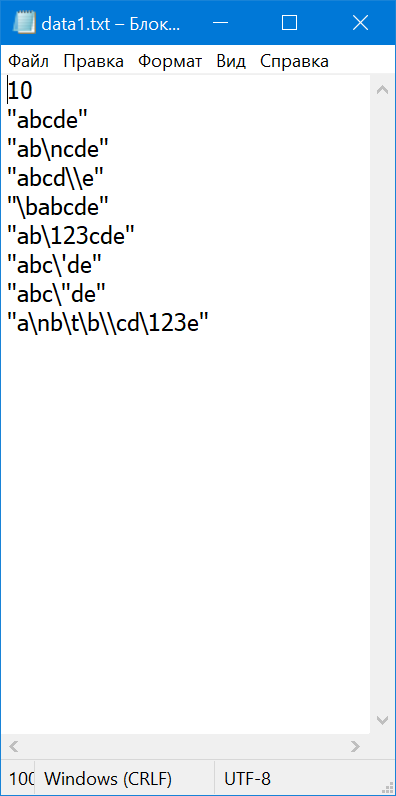
\_getch();

return 0;

}

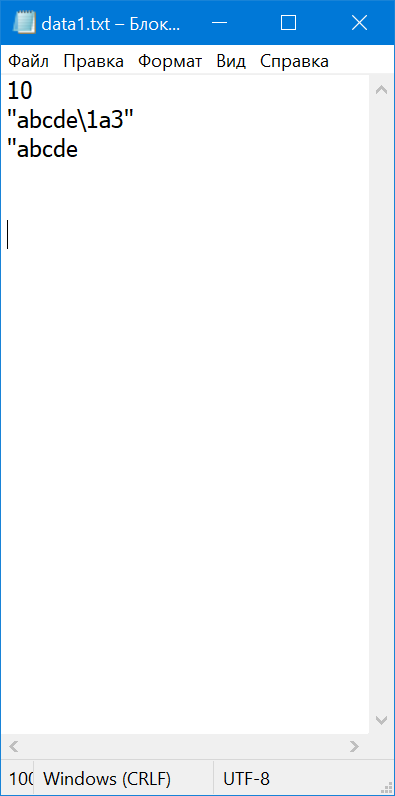
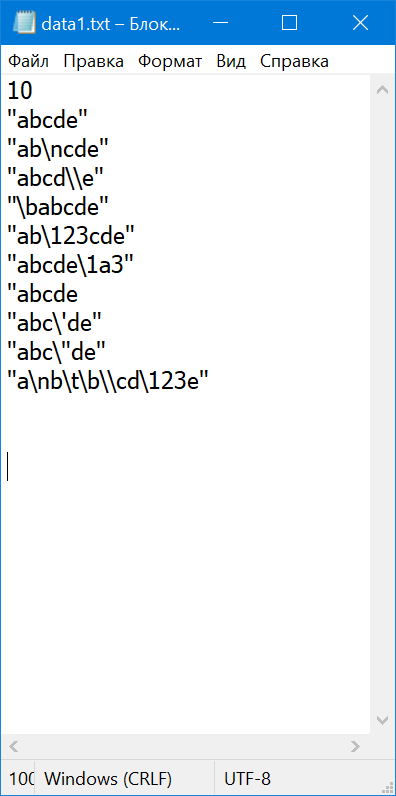
**Результат работы программы №3**

(оставляет в файле строки, являющиеся цепочками заданного языка)



***Результат работы программы №4***

(исключает из файла строки, являющиеся цепочками заданного языка)



**Вывод:**

В ходе проделанной лабораторной работы были изучены основные способы задания регулярных языков, способы построения, алгоритмы преобразования, анализа и реализации конечных распознавателей.

**Конечные распознаватели. Основные понятия.**

Определить принадлежность заданной цепочки заданному регулярному языку можно с помощью *конечного распознавателя*.

***Конечный распознаватель***– это конечный автомат, определяемый пятёркой A = {S, X, s0, δ, F}, где:

* S – конечное непустое множество состояний;
* X – конечное непустое множество входных символов;
* s0 ∈ S – начальное состояние;
* δ: S×X→S – функция переходов, которая паре (состояние, входной символ) ставит в соответствие новое состояние (состояние перехода);
* F ⊆ S – множество допускающих состояний.

Конечный распознаватель можно представить в виде устройства с одним входом, которое в момент времени t может находиться в некотором состоянии stS, а на вход поступает символ xtX из входной цепочки. В дальнейшем будем говорить, что распознаватель в момент времени t находится в состоянии st и обрабатывает символ xt. В соответствии с функцией переходов распознаватель перейдёт в новое состояние st+1 = δ(st, xt), st+1S и в следующий момент времени t+1 будет находиться в состоянии st+1 и обрабатывать следующий символ xt+1 входной цепочки.

В начальный момент времени t=0 распознаватель находится в начальном состоянии s0 и обрабатывает первый символ входной цепочки. На каждом шаге распознаватель обрабатывает новый (следующий за предыдущим) символ цепочки. Если после обработки последнего символа входной цепочки распознаватель окажется в допускающем состоянии skF, то входная цепочка ***принадлежит*** языку (*допускается распознавателем*), иначе – не принадлежит языку (*отвергается распознавателем*).

Другими словами, конечный распознаватель допускает входную цепочку, если она переводит его из начального состояния в одно из допускающих. Множество (возможно, бесконечное) всех цепочек, допускаемых конечным распознавателем А, образует язык L(A), ***допускаемый*** распознавателем A.

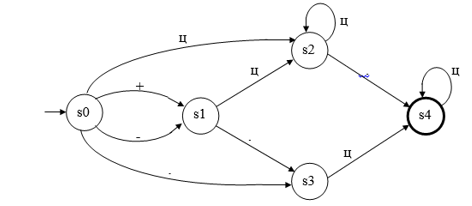
Конечный распознаватель можно задать таблицей, строки которой соответствуют входным символам, а столбцы – состояниям. Первый столбец таблицы соответствует начальному состоянию, а столбцы, соответствующие допускающим состояниям, будем отмечать символом «1». Если распознаватель из состояния sj при обработке символа xi переходит в состояние sk, то в клетке таблицы на пересечении строки xi и столбца sj записывается состояние sk.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | **1** |
|  | **s0** | **s1** | **s2** | **s3** | **s4** |
| **ц** | **s2** | **s2** | **s2** | **s4** | **s4** |
| **.** | **s3** | **s3** | **s4** |  |  |
| **+** | **s1** |  |  |  |  |
| **-** | **s1** |  |  |  |  |

Данная таблица представляет конечный распознаватель, который может находиться в одном из шести состояний: S = {s0, s1, s2, s3, s4, Error} и четыре входных символа: X = {«ц», «.», «+», «-»}. Состоянию **Error** не соответствует столбец таблицы. Это особое состояние, из которого невозможен переход в какое-либо состояние, отличное от него, а, следовательно, из него недостижимо допускающее состояние. Если в процессе обработки входной цепочки распознаватель переходит в состояние ошибки, то цепочка отвергается независимо от необработанной её части. **Переходам в состояние ошибки соответствуют пустые клетки.** Допускающее состояние F = {S4} помечено символом «1».

Конечный распознаватель можно задать ориентированным графом. Вершинам графа соответствуют состояния, а дугам – переходы. В вершинах записываются состояния, а на дугах – входные символы. Если распознаватель из состояния sj при обработке символа xi переходит в состояние sk, то из вершины sj проводится дуга, отмеченная символом xi, в вершину sk. Вершина, соответствующая состоянию ошибки, и дуги, ведущие в состояние ошибки, в графе не изображаются (подразумеваются). Начальное состояние отмечается стрелочкой, а допускающие состояния выделяются жирной линией.

Граф распознавателя, представленного выше таблицей.



Конечный распознаватель, который не может одновременно находиться более чем в одном состоянии, называется ***детерминированным****.* Конечный распознаватель, представленный в качестве примера, является детерминированным, т.к. имеет только одно начальное состояние и функция переходов задаёт единственное следующее состояние перехода для любой пары (состояние, символ).

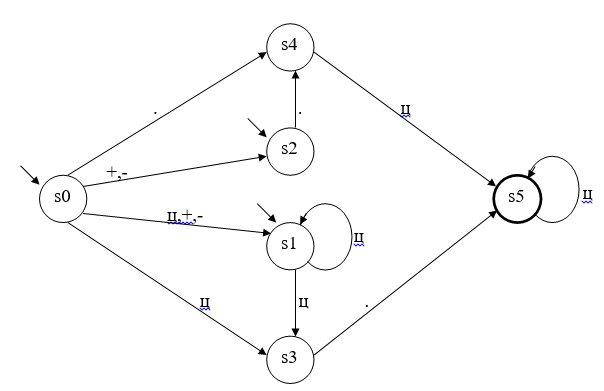
Конечный распознаватель, который может одновременно находиться более чем в одном состоянии, называется ***недетерминированным****.* Недетерминированный распознаватель может иметь несколько начальных состояний и может перейти из состояния при обработке входного символа более чем в одно состояние. К недетерминированному конечному распознавателю нельзя применить определение, приведенное для конечного распознавателя.

**Конечный недетерминированный распознаватель**определяется пятёркой A = {S, X, S0, δ, F}, где:

* S – конечное непустое множество состояний;
* X – конечное непустое множество входных символов;
* S0 ⊆ S – множество начальных состояний;
* δ: S×X→2S, где 2S обозначает булеан S, т.е. множество всех подмножеств множества S – функция переходов, которая паре (состояние, символ) ставит в соответствие подмножество состояний;
* F ⊆ S – множество допускающих состояний.

Недетерминированный распознаватель, подобно детерминированному, может быть задан таблицей или графом. В клетках таблицы недетерминированного распознавателя записываются множества состояний переходов, а столбцы, соответствующие начальным состояниям, отмечаются символом «↓». В графе недетерминированного распознавателя из одной вершины могут выходить различные дуги, отмеченные одним и тем же входным символом.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **↓** | **↓** | **↓** |  |  | **1** |
|  | **s0** | **s1** | **s2** | **s3** | **s4** | **s5** |
| **ц** | **s1,s3** | **s1,s3** |  |  | **s5** | **s5** |
| **.** | **s4** |  | **s4** | **s5** |  |  |
| **+** | **s1,s2** |  |  |  |  |  |
| **-** | **s1,s2** |  |  |  |  |  |

Недетерминированный распознаватель может из состояния перейти в пустое множество состояний. Если при обработке входной цепочки множество текущих состояний в некоторый момент времени окажется пустым, то распознаватель прекращает работу и цепочка отвергается.

Цепочка *допускается* недетерминированным

конечным распознавателем, если после обработки последнего символа входной цепочки он будет находиться во множестве состояний, содержащих в себе хотя бы одно из допускающих.

***Детерминированные конечные распознаватели являются подклассом недетерминированных конечных распознавателей.***

Недетерминированный распознаватель, который может перейти на некотором шаге из состояния в состояние без обработки символа входной цепочки, называется ***недетерминированным конечным распознавателем с ε-переходами***.

*Конечный недетерминированный распознаватель с ε-переходами* определяется

пятеркой A = {S, X, S0, δ, F}, где:

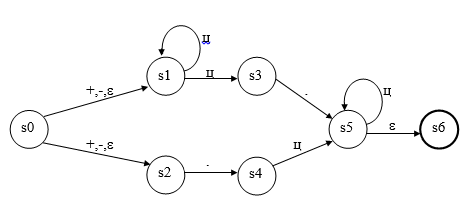
* S – конечное непустое множество состояний;
* X – конечное непустое множество входных символов;
* S0 ⊆ S – множество начальных состояний;
* δ: S×{X∪ε}→2S, где 2S обозначает булеан S, т.е. множество всех подмножеств множества S – функция переходов, которая паре (состояние, символ) или паре (состояние, ε) ставит в соответствие подмножество состояний;
* F ⊆ S – множество допускающих состояний.

Таблица, описывающая недетерминированный распознаватель с ε-переходами, отличается от таблицы недетерминированного распознавателя без ε-переходов наличием дополнительной строки, которая соответствует пустому ε-символу. В графе распознавателя ε-переходу соответствует дуга, отмеченная символом «ε».

Таблица недетерминированного распознавателя с ε-переходами:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **↓** |  |  |  |  |  | **1** |
|  | **s0** | **s1** | **s2** | **s3** | **s4** | **s5** | **s6** |
| **ц** |  | **s1,s3** |  |  | **s5** | **s5** |  |
| **.** |  |  | **s4** | **s5** |  |  |  |
| **+** | **s1,s2** |  |  |  |  |  |  |
| **-** | **s1,s2** |  |  |  |  |  |  |
| **ε** | **s1,s2** |  |  |  |  | **s6** |  |

Граф недетерминированного распознавателя с ε-переходами.



***Недетерминированные конечные распознаватели являются подклассом недетерминированных конечных распознавателей с ε-переходами.***

Два конечных распознавателя (детерминированных, недетерминированных с ε-переходами или без них) называются ***эквивалентными***, если равны допускаемые ими языки. Процедуру получения недетерминированного конечного распознавателя без ε-переходов, эквивалентного заданному недетерминированному конечному распознавателю с ε-переходами назовём устранением ε-переходов. Процедуру получения детерминированного конечного распознавателя, эквивалентного заданному недетерминированному конечному распознавателю назовём преобразованием недетерминированного конечного распознавателя в детерминированный.

**Преобразование недетерминированных конечных**

**распознавателей с ε-переходами в детерминированные**

Преобразование недетерминированных конечных распознавателей с ε-переходами в детерминированные выполняется в два этапа:

1. устранением ε-переходов;
2. преобразованием недетерминированного конечного распознавателя в детерминированный.

При устранении ε-переходов будем использовать понятие «ε-замыкание».

***ε-замыканием состояния***s∈S (или, по-другому, ε(s)) назовём множество всех состояний, которые достижимы из s по ε-переходам.

Само состояние s принадлежит ε-замыканию состояния s (s∈ε(s)). Если из состояния s нет ε-переходов, то ε-замыкание состояния s состоит только из одного состояния s (другими словами, ε(s)={s}). Для недетерминированного конечного распознавателя с ε-переходами для примера, приведенного далее, определены следующие ε-замыкания: ε(s0) = {s0, s1, s2}, ε(s1) = {s1}, ε(s2) = {s2}, ε(s3) = {s3}, ε(s4) = {s4}, ε(s5) = {s5, s6}, ε(s6) = {s6}.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **↓** |  |  |  |  |  | **1** |
|  | **s0** | **s1** | **s2** | **s3** | **s4** | **s5** | **s6** |
| **ц** |  | **s1,s3** |  |  | **s5** | **s5** |  |
| **.** |  |  | **s4** | **s5** |  |  |  |
| **+** | **s1,s2** |  |  |  |  |  |  |
| **-** | **s1,s2** |  |  |  |  |  |  |
| **ε** | **s1,s2** |  |  |  |  | **s6** |  |

***Алгоритм устранения ε-переходов.***

1. Множество ε-замыканий недетерминированного конечного распознавателя с ε-переходами есть множество состояний недетерминированного конечного распознавателя без ε-переходов.
2. По входному символу x∈X недетерминированный конечный распознаватель без ε-переходов переходит из состояния, соответствующего ε(si), в множество состояний, соответствующих ε-замыканиям состояний переходов из каждого состояния sj∈ε(si) по входному символу x.
3. Начальными состояниями недетерминированного конечного распознавателя без ε-переходов являются состояния, соответствующие ε-замыканиям состояний sj∈ε(s0), где s0∈S0, т.е. ε-замыкания состояний, достижимых из начальных по ε-переходам.
4. Допускающими состояниями недетерминированного конечного распознавателя без ε-переходов являются состояния, соответствующие ε-замыканиям состояний ε(s), в которые включаются допускающие состояния недетерминированного конечного распознавателя с ε-переходами.
5. Состояния недетерминированного конечного распознавателя без ε-переходов, недостижимые из начальных, исключаются из множества состояний.

Алгоритм устранения ε-переходов будем представлять построением таблицы недетерминированного конечного распознавателя без ε-переходов. Для недетерминированного конечного распознавателя с ε-переходами имеем:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **↓** | **↓** | **↓** |  |  | **1** | **1** |
|  | **ε(s0) {s0,s1,s2}** | **ε(s1) {s1}** | **ε(s2) {s2}** | **ε(s3) {s3}** | **ε(s4) {s4}** | **ε(s5) {s5, s6}** | **ε(s6) {s6}** |
| **ц** | **ε(s1), ε(s3)** | **ε(s1), ε(s3)** |  |  | **ε(s5)** | **ε(s5)** |  |
| **.** | **ε(s4)** |  | **ε(s4)** | **ε(s5)** |  |  |  |
| **+** | **ε(s1), ε(s2)** |  |  |  |  |  |  |
| **-** | **ε(s1), ε(s2)** |  |  |  |  |  |  |

Обозначив ε(si) через si и исключив недостижимое из начальных состояние ε(s6) получим таблицу недетерминированного конечного распознавателя без ε-переходов:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **↓** | **↓** | **↓** |  |  | **1** |
|  | **s0** | **s1** | **s2** | **s3** | **s4** | **s5** |
| **ц** | **s1,s3** | **s1,s3** |  |  | **s5** | **s5** |
| **.** | **s4** |  | **s4** | **s5** |  |  |
| **+** | **s1,s2** |  |  |  |  |  |
| **-** | **s1,s2** |  |  |  |  |  |

***Алгоритм преобразования недетерминированного конечного распознавателя в детерминированный*.**

1. В качестве начального состояния детерминированного конечного распознавателя установить состояния, соответствующие множеству S0 начальных состояний недетерминированного конечного распознавателя и включить его в множество S’ состояний детерминированного конечного распознавателя (S’={S0}).
2. Пока в множестве S’ есть состояние, для которого состояния перехода не определены, определить их следующим образом: по входному символу x∈X детерминированный конечный распознаватель переходит из состояния, соответствующего подмножеству Si⊆S, в состояние, соответствующее подмножеству Sj⊆S, элементами которого являются состояния переходов из каждого состояния sk∈Si по входному символу x, состояние перехода Sj включить в множество S’ состояний детерминированного конечного распознавателя.
3. Допускающими состояниями детерминированного конечного распознавателя являются состояния, соответствующие подмножествам состояний недетерминированного конечного распознавателя, в которые входит хотя бы одно допускающее состояние недетерминированного конечного распознавателя.

Алгоритм преобразования недетерминированного конечного распознавателя в детерминированный будем представлять построением таблицы детерминированного конечного распознавателя, первый столбец которой отмечается множеством начальных состояний недетерминированного конечного распознавателя. Множества, соответствующие состояниям перехода, заносятся в соответствующие строки. Если получено множество, которым не отмечен ни один столбец таблицы, то добавляется новый столбец и отмечается этим множеством. Построение таблицы заканчивается, когда будут определены состояния перехода для всех столбцов.

Для недетерминированного конечного распознавателя имеем:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **↓** |  |  |  | **1** |
|  | **{s0,s1,s2}** | **{s1,s3}** | **{s4}** | **{s1,s2}** | **{s5}** |
| **ц** | **{s1,s3}** | **{s1,s3}** | **{s5}** | **{s1,s3}** | **{s5}** |
| **.** | **{s4}** | **{s5}** |  | **{s4}** |  |
| **+** | **{s1,s2}** |  |  |  |  |
| **-** | **{s1,s2}** |  |  |  |  |

Обозначив состояние, соответствующее i-му столбцу таблицы через si-1, получим таблицу детерминированного конечного распознавателя:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **↓** |  |  |  | **1** |
|  | **s0** | **s1** | **s2** | **s3** | **s4** |
| **ц** | **s1** | **s1** | **s4** | **s1** | **s4** |
| **.** | **s2** | **s4** |  | **s2** |  |
| **+** | **s3** |  |  |  |  |
| **-** | **s3** |  |  |  |  |

**Минимизация конечных детерминированных распознавателей**

Конечный детерминированный распознаватель, содержащий наименьшее количество состояний по сравнению с любым, эквивалентным ему распознавателем, называется *минимальным.* Множество всех эквивалентных между собой распознавателей называется *классом эквивалентных распознавателей.* Класс эквивалентных распознавателей может содержать несколько распознавателей с одинаковым, наименьшим числом состояний по сравнению с любым другим распознавателем из этого класса, но все они *изоморфны*, т.е. различаются только названиями состояний. Процесс получения минимального распознавателя, эквивалентного заданному, называется *минимизацией конечного распознавателя.*

Конечный распознаватель является минимальным, если он не содержит эквивалентных между собой состояний и состояний, недостижимых из начального. В связи с этим, процесс минимизации конечных детерминированных распознавателей выполняется в два этапа:

1. поиск и устранение состояний, недостижимых из начального;
2. поиск и замена эквивалентных между собой состояний одним состоянием.

**ПЕРВЫЙ ЭТАП МИНИМИЗАЦИИ**

***Алгоритм поиска состояний, недостижимых из начального.***

1. Определить множество состояний, достижимых из начального, используя алгоритм обхода графа (например, в ширину или в глубину).
2. В результате исключения из множества состояний распознавателя состояний, достижимых из начального, получим множество состояний, недостижимых из начального.

***Алгоритм устранения состояний, недостижимых из начального*.**

1. Исключить из множества состояний распознавателя состояния, недостижимые из начального.

При табличном способе задания распознавателя исключаются столбцы, соответствующие состояниям, недостижимым из начального, а при графовом способе – вершины, соответствующие исключаемым состояниям, и выходящие из них дуги.

**ВТОРОЙ ЭТАП МИНИМИЗАЦИИ**

Представленный далее способ заключается в нахождении классов эквивалентных состояний, которые в дальнейшем становятся состояниями минимального распознавателя.

***Класс эквивалентных состояний*** – это множество всех эквивалентных между собой состояний конечного распознавателя. Состояния, для которых не существует различающих цепочек длины k, называются *k-эквивалентными*. Множество всех k-эквивалентных между собой состояний называются *классом k-эквивалентных состояний*.

Для нахождения классов эквивалентных состояний нужно последовательно находить классы 0-, 1-, …, k-эквивалентных состояний, пока классы k-эквивалентных состояний не совпадают с классами (k-1)-эквивалентных состояний. При совпадении классов k- и (k-1)-эквивалентных состояний полученные классы представляют собой классы эквивалентных состояний.

Множество всех допускающих состояний образуют класс 0-эквивалентных состояний, т.к. все они допускают пустые цепочки. Множество всех отвергающих состояний образуют другой класс 0-эквивалентных состояний, т.к. все они отвергают пустые цепочки.

Для того, чтобы состояния были k-эквивалентными, они должны быть k-1-эквивалентными и под действием одинаковых входных символов должны переходить в k-эквивалентные состояния. Процесс нахождения классов эквивалентных состояний можно представить построением таблиц переходов из состояний в классы k-эквивалентных состояний.

**Пример**. Таблица переходов конечного распознавателя.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **↓** |  |  |  | **1** |  |  | **1** | **1** |  |
|  | **s0** | **s1** | **s2** | **s3** | **s4** | **s5** | **s6** | **s7** | **s8** |  |
| **ц** | **s6** | **s5** | **s6** | **s4** | **s4** | **s6** | **s5** | **s8** |  |  |
| **.** | **s3** | **s3** | **s3** |  |  | **s7** | **s8** |  |  |  |
| **+** | **s1** |  |  |  |  |  |  |  |  |  |
| **-** | **s2** |  |  |  |  |  |  |  |  |  |

Последний столбец в таблице соответствует состоянию ошибки, а пустые клетки – переходам в состояния ошибки. Отвергающие состояния {s0, s1, s2, s3, s5, s6} объединяем в класс «К1» 0-эквивалентных состояний, а допускающие состояния {s4, s7, s8} – в класс «К2».

Таблица переходов в классы 0-эквивалентных состояний.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **K1** | | | | | | | **К2** | | |
|  | **s0** | **s1** | **s2** | **s3** | **s5** | **s6** |  | **s4** | **s7** | **s8** |
| **ц** | **К1** | **К1** | **К1** | **К2** | **К1** | **К1** | **К1** | **К2** | **К2** | **К2** |
| **.** | **К1** | **К1** | **К1** | **К1** | **К2** | **К2** | **К1** | **К1** | **К1** | **К1** |
| **+** | **К1** | **К1** | **К1** | **К1** | **К1** | **К1** | **К1** | **К1** | **К1** | **К1** |
| **-** | **К1** | **К1** | **К1** | **К1** | **К1** | **К1** | **К1** | **К1** | **К1** | **К1** |

* Распознаватель из состояния s0 под действием входного символа «ц» переходит в состояние s6, которое принадлежит классу K1, поэтому в таблице на пересечении столбца s0 и строки «ц» записано K1.
* Распознаватель из состояния s1 под действием входного символа «+» переходит в состояние ошибки, которое принадлежит классу K1, поэтому в таблице на пересечении столбца s1 и строки «+» записано K1.
* Распознаватель из состояния s3 под действием входного символа «ц» переходит в состояние s4, которое принадлежит классу K2, поэтому в таблице на пересечении столбца s3 и строки «ц» записано K2.
* Распознаватель из состояния ошибки под действием любого входного символа переходит в состояние ошибки, которое принадлежит классу K1. В связи с этим, в таблице в столбце, соответствующем состоянию ошибки, записаны K1 для всех входных символов.

По таблице переходов в классы 0-эквивалентных состояний определяем классы 1-эквивалентных состояний. Состояния будут 1-эквивалентными, если они 0-эквивалентны и под действием одинаковых входных сигналов переходят в состояния, принадлежащие одинаковым классам 0-эквивалентных состояний, т.е. 1-эквивалентным состояниям в таблице переходов в классы 0-эквивалентных состояний соответствуют одинаково заполненные столбцы. Объединяем 1-эквивалентные состояния в классы и строим таблицу переходов в классы 1-эквивалентных состояний для определения 2-эквивалентных состояний.

Таблица переходов в классы 1-эквивалентных состояний.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **K1** | | | | **K2** | **K3** | | **K4** | | |
|  | **s0** | **s1** | **s2** |  | **s3** | **s5** | **s6** | **s4** | **s7** | **s8** |
| **ц** | **K3** | **K3** | **K3** | **K1** | **K4** | **K3** | **K3** | **K4** | **K4** | **K4** |
| **.** | **K2** | **K2** | **K2** | **K1** | **K1** | **K4** | **K4** | **K1** | **K1** | **K1** |
| **+** | **K1** | **K1** | **K1** | **K1** | **K1** | **K1** | **K1** | **K1** | **K1** | **K1** |
| **-** | **K1** | **K1** | **K1** | **K1** | **K1** | **K1** | **K1** | **K1** | **K1** | **K1** |

В классе К1 2-эквивалентными состояниями будут s0, s1 и s2. Состояние ошибки – отдельный класс 2-эквивалентных состояний. Остальные классы 1-эквивалентных состояний также являются и 2-эквивалентными. Классы 2-эквивалентных состояний не совпадают с классами 1-эквивалентных состояний, поэтому строим таблицу переходов в классы 2-эквивалентных состояний для определения классов 3-эквивалентных состояний.

Таблица переходов в классы 2-эквивалентных состояний.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **K1** | | | **K2** | **K3** | **K4** | | **K5** | | |
|  | **s0** | **s1** | **s2** |  | **s3** | **s5** | **s6** | **s4** | **s7** | **s8** |
| **ц** | **K4** | **K4** | **K4** | **K2** | **K5** | **K4** | **K4** | **K5** | **K5** | **K5** |
| **.** | **K3** | **K3** | **K3** | **K2** | **K2** | **K5** | **K5** | **K2** | **K2** | **K2** |
| **+** | **K1** | **K2** | **K2** | **K2** | **K2** | **K2** | **K2** | **K2** | **K2** | **K2** |
| **-** | **K1** | **K2** | **K2** | **K2** | **K2** | **K2** | **K2** | **K2** | **K2** | **K2** |

В классе К1 3-эквивалентными состояниями будут s1 и s2. Состояние перехода s0 – отдельный класс 3-эквивалентных состояний. Остальные классы 2-эквивалентных состояний также являются и 3-эквивалентными. Классы 3-эквивалентных состояний не совпадают с классами 2-эквивалентных состояний, поэтому строим таблицу переходов в классы 3-эквивалентных состояний для определения классов 4-эквивалентных состояний.

Таблица переходов в классы 3-эквивалентных состояний.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **K1** | **K2** | | **K3** | **K4** | **K5** | | **K6** | | |
|  | **s0** | **s1** | **s2** |  | **s3** | **s5** | **s6** | **s4** | **s7** | **s8** |
| **ц** | **K5** | **K5** | **K5** | **K3** | **K6** | **K5** | **K5** | **K6** | **K6** | **K6** |
| **.** | **K4** | **K4** | **K4** | **K3** | **K3** | **K6** | **K6** | **K3** | **K3** | **K3** |
| **+** | **K2** | **K3** | **K3** | **K3** | **K3** | **K3** | **K3** | **K3** | **K3** | **K3** |
| **-** | **K2** | **K3** | **K3** | **K3** | **K3** | **K3** | **K3** | **K3** | **K3** | **K3** |

По таблице видно, что классы 4-эквивалентных состояний совпадают с классами 3-эквивалентных состояний. Из этого следует, что классы 3-эквивалентных состояний представляют собой классы эквивалентных состояний.

***Алгоритм формирования минимального распознавателя.***

1. Множеством состояний минимального распознавателя сделать множество классов эквивалентных состояний исходного распознавателя. Класс эквивалентных состояний исходного распознавателя соответствует состоянию минимального распознавателя.
2. Начальному состоянию минимального распознавателя соответствует класс эквивалентных состояний, содержащий в себе начальное состояние исходного распознавателя.
3. Допускающим состояниям минимального распознавателя соответствуют классы эквивалентных состояний, содержащие в себе допускающие состояние исходного распознавателя.
4. Функция переходов минимального распознавателя определяется следующим образом. Если из состояний класса Ki под действием входного символа x исходный распознаватель переходит в состояния класса Kj, то в минимальном распознавателе состоянием перехода состояния Ki под действием входного символа x сделать состояние Kj.

Если классы эквивалентных состояний определялись с помощью таблиц переходов в классы k-эквивалентных состояний, то для того, чтобы получить таблицу минимального распознавателя достаточно в последней таблице в каждом классе эквивалентных состояний оставить по одному столбцу, исключить строку, в которой записаны названия состояний исходного распознавателя, и отметить начальное и допускающие состояния.

Таблица переходов минимального распознавателя.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **↓** |  |  |  |  | **1** |
|  | **K1** | **K2** | **K3** | **K4** | **K5** | **K6** |
| **ц** | **K5** | **K5** | **K3** | **K6** | **K5** | **K6** |
| **.** | **K4** | **K4** | **K3** | **K3** | **K6** | **K3** |
| **+** | **K2** | **K3** | **K3** | **K3** | **K3** | **K3** |
| **-** | **K2** | **K3** | **K3** | **K3** | **K3** | **K3** |

Переобозначим состояния K1 через s0, K2 через s1, K4 через s2, K5 через s3, K6 через s4, состояние ошибки К3 и переходы в него исключим из таблицы.

Окончательная таблица переходов минимального распознавателя.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **↓** |  |  |  | **1** |
|  | **s0** | **s1** | **s2** | **s3** | **s4** |
| **ц** | **s3** | **s3** | **s4** | **s3** | **s4** |
| **.** | **s2** | **s2** |  | **s4** |  |
| **+** | **s1** |  |  |  |  |
| **-** | **s1** |  |  |  |  |

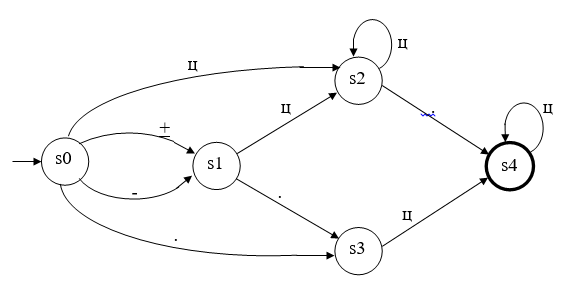
**Программная реализация конечных детерминированных распознавателей**

Рассмотрим два способа программной реализации конечных детерминированных распознавателей: *компиляционный* (программный) и *интерпретационный* (табличный).

**КОМПИЛЯЦИОННЫЙ СПОСОБ**

Пусть задан граф конечного детерминированного распознавателя. Его можно преобразовать в программу распознавания цепочки языка, размер которой пропорционален количеству вершин (состояний) и дуг в графе. Каждому состоянию соответствует часть кода: считывание очередного символа и, если считан не концевой маркер, определение состояния перехода. Процесс распознавания заканчивается, если считан концевой маркер или если состоянием перехода является состояние ошибки. При переходе в состояние ошибки может быть выдано сообщение о типе ошибки. Результат распознавания определяется состоянием, в котором завершён процесс распознавания: если оно допускающее – цепочка допускается, иначе – отвергается.

Рассмотрим два алгоритма реализации следующего конечного детерминированного распознавателя:



***Алгоритм №1***

Код каждого состояния отметим соответствующей меткой. Переменную *x* будем использовать для хранения символа входной цепочки. Анализ переменной *х* выполним с помощью оператора множественного выбора. Переход в новое состояние закодируем оператором безусловного перехода на метку. Концевой маркер обозначим символом «┤».

Псевдокод алгоритма 1 представлен ниже:

s0: ввод (*х*);

выбор (*х*)

*х*∈{«+», «-»} : переход на s1;

*х*∈{«0» .. «9»} : переход на s2;

*х* = «.» : переход на s3;

*x* = «┤» : вывод ( ‘Отвергнуть, цепочка пуста’), конец.

конец выбор;

s1: ввод (*х*);

выбор (*х*)

*х*∈{«0» .. «9»} : переход на s2;

*х* = «.» : переход на s3;

*х*∈{«+», «-»} : вывод ( ‘Отвергнуть, недопустимы два знака подряд’), конец.

*x* = «┤» : вывод ( ‘Отвергнуть, нет значения константы’), конец.

конец выбор;

s2: ввод (*х*);

выбор (*х*)

*х*∈{«0» .. «9»} : переход на s2;

*х* = «.» : переход на s4;

иначе : вывод ( ‘Отвергнуть, должна быть точка’), конец.

конец выбор;

s3: ввод (*х*);

выбор (*х*)

*х*∈{«0» .. «9»} : переход на s4;

*х* = «.» : переход на s4;

иначе : вывод ( ‘Отвергнуть, после точки должна быть цифра’), конец.

конец выбор;

s4: ввод (*х*);

выбор (*х*)

*х*∈{«0» .. «9»} : переход на s4;

*х*∈{«+», «-», «.»} : вывод ( ‘Отвергнуть, последний

символ не может встречаться дважды’), конец.

*x* = «┤» : вывод ( ‘Допустить’), конец.

конец выбор;

***Алгоритм №2*** (см. стр. 84).

**ИНТЕРПРЕТАЦИОННЫЙ СПОСОБ**

В этом способе реализации используется табличный способ задания конечного детерминированного распознавателя. Таблица распознавателя сохраняется в памяти и обрабатывается по определённому, общему для всех распознавателей, алгоритму. Номер состояния распознавателя будем хранить в переменной *S*, символ входной цепочки – в переменной *x*. На каждом шаге распознавания, пока не введён концевой маркер, определяется состояние перехода путём обращения к элементу таблицы распознавателя, расположенному в строке, соответствующей входному символу *х*, и столбце, соответствующем состоянию *S*. С помощью функции L(*x*) поставим в соответствие каждому входному символу номер строки таблицы. Для быстрого вычисления L(*x*) представим эту функцию в табличной форме и сохраним в массиве L таким образом, что элемент L[*x*] содержит номер строки таблицы распознавателя, соответствующей входному символу *х*. В результате, для вычисления значения функции L(*x*) достаточно обратиться к элементу L[*x*]. Переходы в состояние ошибки закодируем отрицательными числами, представляющими собой код ошибки. Сообщения об ошибках будем хранить в таблице ошибок. Выход из процесса распознавания происходит при вводе концевого маркера или достижении состояния ошибки.

Итак, для реализации конечного детерминированного распознавателя интерпретационным способом, необходимо иметь:

1. таблицу переходов конечного детерминированного распознавателя;
2. множество допускающих состояний;
3. массив L для определения строки, таблицы, соответствующей входному символу;
4. таблицу ошибок.

Ниже представлен псевдокод алгоритма интерпретационного способа реализации конечного детерминированного распознавателя.

*S* := 0;

ввод (*х*);

пока (*S* ≥ 0) и (х ≠ «┤») выполнять

*S* := таблица\_переходов[L[*x*], *S*];

ввод (*х*);

конец пока;

если *S* ∈ множество\_допускающих\_состояний

то вывод (‘Допустить’)

иначе вывод(‘Отвергнуть ’, таблица\_ошибок[-*S*]);

Этот алгоритм является универсальным и позволяет получать программы реализации различных распознавателей путём изменения обрабатываемых данных.