МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования  
«Гомельский государственный технический университет имени П.О. Сухого»

Факультет автоматизированных и информационных систем

Кафедра «Информатика»

Специальность 1-40 04 01 «Информатика и технологии программирования»

Отчет  
по лабораторной работе № 6  
по теме «Построение детерминированного конечного автомата»  
по дисциплине «Методы трансляции»

Составил: студент гр. ИП-31 09.05.2023 Коваленко А.И.

(подпись, дата)

Принял: доцент кафедры ИТ Кравченко О.А.

(подпись, дата)

Гомель 2023

**Цель работы**: проведение синтаксического анализа конструкции языка программирования с помощью детерминированного конечного автомата (ДКА).

**Задание**

Дано общее описание конструкции некоторого языка программирования. Задана конкретная конструкция. Предполагается, что проведен лексический анализ этой конструкции, в результате которого составлен новый (упрощенный) алфавит исходного языка. Данный алфавит характерен тем, что в нем каждое служебное слово, каждый идентификатор и каждое число, входящие в конструкцию языка программирования, рассматриваются как отдельные символы.

Требуется:

1) составить ДКА, который анализирует синтаксическую правильность заданной конструкции, т. е. соответствие структуры конструкции общему описанию;

2) разработать программу (на одном из изученных языков программирования), которая моделирует работу ДКА;

3) проверить работу программы при анализе правильной и неправильной конструкции.

**Вариант индивидуального задания:**

**Вариант 15.**

Оператор цикла с постусловием языка программирования имеет следующую структуру:

*repeat оператор\_присваивания; оператор\_присваивания;*

*оператор\_присваивания; until логическое\_выражение;*

Логическое выражение, входящее в оператор цикла, имеет следующую структуру:

*элемент операция\_отношения элемент*

В качестве элемента может быть переменная или число.

Операция отношения – это одна из шести операций: =, <, >, <=, >=, <>.

Оператор присваивания имеет одну из следующих структур:

переменная := выражение

переменная := алгебраическое\_сложение выражение

переменная := выражение алгебраическое\_сложение выражение

Алгебраическое сложение – это одна из операций + или -.

Выражение – это единственный множитель или два множителя, между которыми стоит операция умножения (\*) или деления (/).

В качестве множителя может выступать переменная или число.

Пробелы могут стоять в конструкции в любом месте. Каждое служебное слово должно быть выделено пробелами.

Алфавит языка после лексического анализа:

Служебные слова: repeat, until.

Знаки: ;, :=, =, <, >, +, -, \*, /, пробел.

Переменные: a1, a2, a3, a4, a5, a6.

Числа: r1, r2, r3, r4, r5, r6.

Пример правильной конструкции:

*repeat a3:= r1-a2\*a5; a2:= r1+a2/r4; a6:=-a1; until a2 <= r5;*

Пример неправильной конструкции:

*repeat a3:= r1-a2\*a5; a2:= r1+a2/r4; a6:=-a1 until a2 <= r5;*

**Ход выполнения.**

После лексического анализа был построен детерминированный конечный автомат (ДКА) для оператора цикла с постусловием языка программирования.

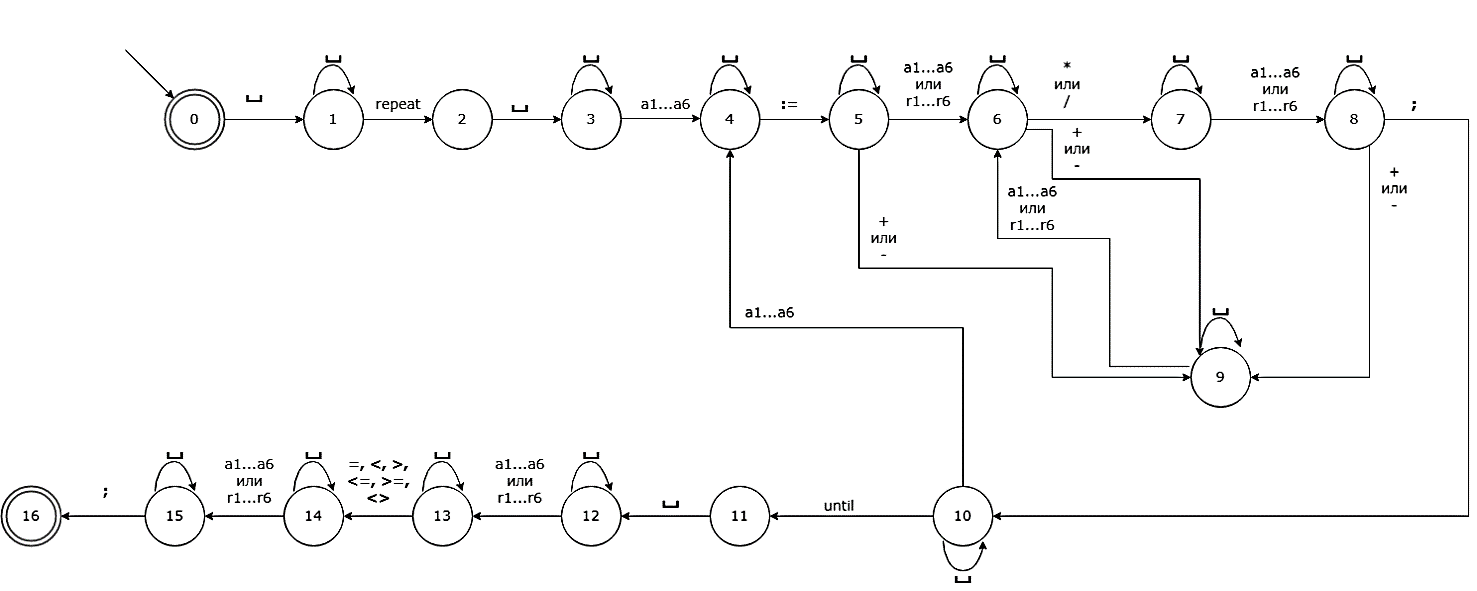


Рисунок 1 – ДКА для оператора цикла с постусловием

Прим. если у состояния встречается символ, не описанный для перехода к следующим, следует полагать, что он приводит к состоянию ошибки (error). Эти переходы не отображены для лучшей читаемости автомата.

Так же была составлена таблица состояний.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Состояние | пробел | repeat | until | a1…a6 | r1…r6 | := | +,- | \*,/ | Опер отнош. | ; |
| 0 | 1 | e | e | e | e | e | e | e | e | e |
| 1 | 1 | 2 | e | e | e | e | e | e | e | e |
| 2 | 3 | e | e | e | e | e | e | e | e | e |
| 3 | 3 | e | e | 4 | e | e | e | e | e | e |
| 4 | 4 | e | e | e | e | 5 | e | e | e | e |
| 5 | 5 | e | e | 6 | 6 | e | 9 | e | e | e |
| 6 | 6 | e | e | e | e | e | 9 | 7 | e | 10 |
| 7 | 7 | e | e | 8 | 8 | e | e | e | e | e |
| 8 | 8 | e | e | e | e | e | 9 | e | e | 10 |
| 9 | 9 | e | e | 6 | 6 | e | e | e | e | e |
| 10 | 10 | e | 11 | 4 | e | e | e | e | e | e |
| 11 | 12 | e | e | e | e | e | e | e | e | e |
| 12 | 12 | e | e | 13 | 13 | e | e | e | e | e |
| 13 | 13 | e | e | e | e | e | e | e | 14 | e |
| 14 | 14 | e | e | 15 | 15 | e | e | e | e | e |
| 15 | 15 | e | e | e | e | e | e | e | e | 16 |
| 16 | e | e | e | e | e | e | e | e | e | e |
| Error | e | e | e | e | e | e | e | e | e | e |

Таблица 1 – Таблица состояний ДКА для оператора цикла с постусловием

**Листинг программы:**

**Файл Form1.cs:**

using System.Security.Cryptography;

namespace MTLab

{

public partial class Form1 : Form

{

private string \_rightString = "repeat a3:= r1-a2\* a5; a2:= r1+a2/r4; a6:=-a1; until a2 <= r5;";

private string \_falseString = "repeat a3:= r1-a2\* a5; a2:= r1+a2/r4; a6:=-a1 until a2 <= r5;";

// Определяем алфавит

//List<string> alphabet = new List<string>() { " ", "repeat", "until", "a1", "a2", "a3", "a4", "a5", "a6", "r1", "r2", "r3", "r4", "r5", "r6",

//":=", "+", "-", "\*", "/", "=", "<", ">", "<=", ">=", "<>", ";"};

List<string> alphabet = new List<string>() { "␣", "repeat", "until", "a1...a6", "r1...r6",

":=", "+,-", "\*,/", "=,<,>,<=,>=,<>", ";"};

public Form1()

{

InitializeComponent();

Actions();

}

private Dictionary<string, Dictionary<string, string>> Actions()

{

// Очищаем таблицу

dataGridView1.Rows.Clear();

dataGridView1.Columns.Clear();

// Создаём ДКА

Dictionary<string, Dictionary<string, string>> dfa = new Dictionary<string, Dictionary<string, string>>();

//dfa.Add("0", new Dictionary<string, string>() { { "␣", "1" } });

//dfa.Add("1", new Dictionary<string, string>() { { "␣", "1" }, { "repeat", "2" } });

//dfa.Add("2", new Dictionary<string, string>() { { "␣", "2" }, { "a1...a6", "3" } });

//dfa.Add("3", new Dictionary<string, string>() { { "␣", "3" }, { ":=", "4" } });

//dfa.Add("4", new Dictionary<string, string>() { { "␣", "4" }, { "a1...a6", "5" }, { "r1...r6", "5" }, { "+,-", "8" } });

//dfa.Add("5", new Dictionary<string, string>() { { "␣", "5" }, { "\*,/", "6" }, { "+,-", "8" }, { ";", "9" } });

//dfa.Add("6", new Dictionary<string, string>() { { "␣", "6" }, { "a1...a6", "7" }, { "r1...r6", "7" } });

//dfa.Add("7", new Dictionary<string, string>() { { "␣", "7" }, { "+,-", "8" }, { ";", "9" } });

//dfa.Add("8", new Dictionary<string, string>() { { "␣", "8" }, { "a1...a6", "5" }, { "r1...r6", "5" } });

//dfa.Add("9", new Dictionary<string, string>() { { "␣", "9" }, { "a1...a6", "3" }, { "until", "10" } });

//dfa.Add("10", new Dictionary<string, string>() { { "␣", "10" }, { "a1...a6", "11" }, { "r1...r6", "11" } });

//dfa.Add("11", new Dictionary<string, string>() { { "␣", "11" }, { "=,<,>,<=,>=,<>", "12" } });

//dfa.Add("12", new Dictionary<string, string>() { { "␣", "12" }, { "a1...a6", "13" }, { "r1...r6", "13" } });

//dfa.Add("13", new Dictionary<string, string>() { { "␣", "13" }, { ";", "14" } });

//dfa.Add("14", new Dictionary<string, string>() { });

//dfa.Add("error", new Dictionary<string, string>() { });

dfa.Add("0", new Dictionary<string, string>() { { "␣", "1" } });

dfa.Add("1", new Dictionary<string, string>() { { "␣", "1" }, { "repeat", "2" } });

dfa.Add("2", new Dictionary<string, string>() { { "␣", "3" } });

dfa.Add("3", new Dictionary<string, string>() { { "␣", "3" }, { "a1...a6", "4" } });

dfa.Add("4", new Dictionary<string, string>() { { "␣", "4" }, { ":=", "5" } });

dfa.Add("5", new Dictionary<string, string>() { { "␣", "5" }, { "a1...a6", "6" }, { "r1...r6", "6" }, { "+,-", "9" } });

dfa.Add("6", new Dictionary<string, string>() { { "␣", "6" }, { "+,-", "9" }, { "\*,/", "7" }, { ";", "10" } });

dfa.Add("7", new Dictionary<string, string>() { { "␣", "7" }, { "a1...a6", "8" }, { "r1...r6", "8" } });

dfa.Add("8", new Dictionary<string, string>() { { "␣", "8" }, { "+,-", "9" }, { ";", "10" } });

dfa.Add("9", new Dictionary<string, string>() { { "␣", "9" }, { "a1...a6", "6" }, { "r1...r6", "6" } });

dfa.Add("10", new Dictionary<string, string>() { { "␣", "10" }, { "until", "11" }, { "a1...a6", "4" } });

dfa.Add("11", new Dictionary<string, string>() { { "␣", "12" } });

dfa.Add("12", new Dictionary<string, string>() { { "␣", "12" }, { "a1...a6", "13" }, { "r1...r6", "13" } });

dfa.Add("13", new Dictionary<string, string>() { { "␣", "13" }, { "=,<,>,<=,>=,<>", "14" } });

dfa.Add("14", new Dictionary<string, string>() { { "␣", "14" }, { "a1...a6", "15" }, { "r1...r6", "15" } });

dfa.Add("15", new Dictionary<string, string>() { { "␣", "15" }, { ";", "16" } });

dfa.Add("16", new Dictionary<string, string>() { });

dfa.Add("error", new Dictionary<string, string>() { });

// Создаем таблицу состояний

Dictionary<string, Dictionary<string, string>> stateTable = new Dictionary<string, Dictionary<string, string>>();

// Заполняем таблицу состояний

foreach (string state in dfa.Keys)

{

Dictionary<string, string> row = new Dictionary<string, string>();

foreach (string symbol in alphabet)

{

string nextState;

try

{

nextState = dfa[state][symbol];

}

catch (Exception ex)

{

nextState = "e";

}

row.Add(symbol, nextState);

}

stateTable.Add(state, row);

}

// Создаём datagridview

dataGridView1.Columns.Add("Condition", "Состояние");

dataGridView1.Columns["Condition"].Width = 85;

foreach (var el in alphabet)

{

dataGridView1.Columns.Add(el, el);

dataGridView1.Columns[el].Width = 62;

}

// Заполняем datagridview

foreach (var el in stateTable)

{

dataGridView1.Rows.Add();

dataGridView1["Condition", dataGridView1.RowCount - 1].Value = dataGridView1.RowCount - 1;

foreach (var innerEl in el.Value)

{

dataGridView1[innerEl.Key, dataGridView1.RowCount - 1].Value = innerEl.Value;

}

}

dataGridView1["Condition", dataGridView1.RowCount - 1].Value = "Error";

return dfa;

}

private void button1\_Click(object sender, EventArgs e)

{

Dictionary<string, Dictionary<string, string>> dfa = Actions();

List<string> result = new List<string>();

string buffer = "";

foreach (char c in \_rightString)

{

if (alphabet.Contains(c.ToString()))

{

if (!string.IsNullOrWhiteSpace(buffer))

{

result.Add(buffer.Trim());

buffer = "";

}

result.Add(c.ToString().Trim());

}

else if (c == ':')

{

if (!string.IsNullOrWhiteSpace(buffer) && alphabet.Contains(buffer))

{

result.Add(buffer.Trim());

buffer = "";

}

buffer += c;

}

else

{

buffer += c;

}

}

if (!string.IsNullOrWhiteSpace(buffer))

{

result.Add(buffer.Trim());

}

bool flag = true;

try

{

result = result.ConvertAll(s => string.IsNullOrWhiteSpace(s) ? " " : s);

for (int i = 0, j = 0; i < dfa.Keys.Count() && j < result.Count(); j++)

{

if (dfa[i.ToString()][result[j]] == "e")

{

flag = false;

break;

}

else

{

i = int.Parse(dfa[i.ToString()][result[j]]);

}

}

}

catch

{

flag = false;

}

if (flag)

label4.Text = "Данная конструкция не правильна";

else

label4.Text = "Данная конструкция правильна";

}

private void button2\_Click(object sender, EventArgs e)

{

Dictionary<string, Dictionary<string, string>> dfa = Actions();

List<string> result = new List<string>();

string buffer = "";

foreach (char c in \_rightString)

{

if (alphabet.Contains(c.ToString()))

{

if (!string.IsNullOrWhiteSpace(buffer))

{

result.Add(buffer.Trim());

buffer = "";

}

result.Add(c.ToString().Trim());

}

else if (c == ':')

{

if (!string.IsNullOrWhiteSpace(buffer) && alphabet.Contains(buffer))

{

result.Add(buffer.Trim());

buffer = "";

}

buffer += c;

}

else

{

buffer += c;

}

}

if (!string.IsNullOrWhiteSpace(buffer))

{

result.Add(buffer.Trim());

}

bool flag = true;

try

{

result = result.ConvertAll(s => string.IsNullOrWhiteSpace(s) ? " " : s);

for (int i = 0, j = 0; i < dfa.Keys.Count() && j < result.Count(); j++)

{

if (dfa[i.ToString()][result[j]] == "e")

{

flag = false;

break;

}

else

{

i = int.Parse(dfa[i.ToString()][result[j]]);

}

}

}

catch

{

flag = false;

}

if (flag)

label4.Text = "Данная конструкция правильна";

else

label4.Text = "Данная конструкция не правильна";

}

}

}

**Результат выполнения:**

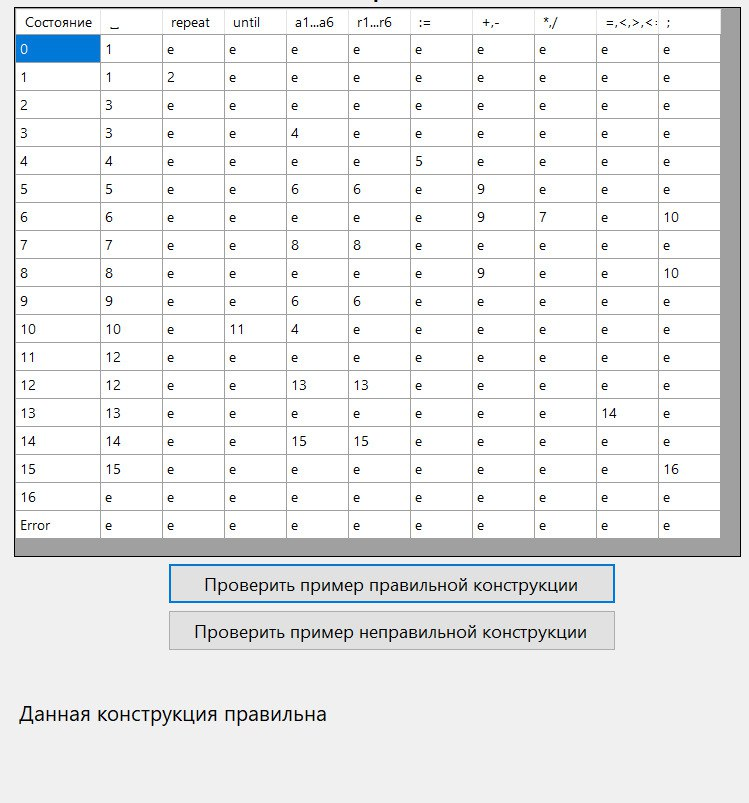


Рисунок 2 – Результат выполнения программы при нажатии на кнопку проверки правильной конструкции

**Вывод:** в ходе выполнения лабораторной работы был проведен синтаксический анализ конструкции языка программирования с помощью детерминированного конечного автомата (ДКА).