Федеральное государственное автономное образовательное учреждение

высшего образования «Санкт-Петербургский государственный университет аэрокосмического приборостроения» (ГУАП)

ОТЧЕТ ЗАЩИЩЕН

Ст. преподаватель Рогачев С.А. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Должность Ф.И.О дата, подпись

**Отчет**

**о лабораторной работе №5**

**«Синтез конечных автоматов»**

по дисциплине «Теория вычислительных процессов»

ОТЧЕТ ВЫПОЛНИЛ:

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ф.И.О. группа дата, подпись

Санкт-Петербург

2024

**Цель работы**

Построить конечный автомат, который осуществляет проверку входного слова на допустимость в заданном регулярном выражении используя алгоритм синтеза конечных автоматов.

**Вариант 10:**

<x|<b|d>>m<a>(b|d)

**Ход работы**

Алгоритм синтеза КНА

Примем следующее правило отметки состояний КНА:

Состояние qi КНА, определяемое подмножеством множества основных мест регулярных выражений R1,…, Rp, отмечается множеством, содержащим все те и только те выражения R1,…, Rp, конечные места которых подчинены хотя бы одному основному месту из числа мест, входящих в подмножество для qi.

Из предыдущего изложения следует следующее утверждение: Слово а в алфавите Х регулярных выражений тогда и только тогда переводит КНА из начального состояния q1 в состояние qj, отмеченное произвольным множеством, содержащим любое из заданных рег. выражений Ri, тогда начальное место выражения Ri связано с конечным местом этого выражения словом a.

Пусть Х – алфавит для R (без «(», «)», «|», «&», «<», «>»)

Основным местом в R назовем место, непосредственно слева от которого стоит символ (буква) алфавита Х, а также начальное место.

Место непосредственно справа от которого стоит буква алфавита Х, будем называть предосновным местом.

Правила подчинения мест в регулярном выражении

1. Начальные места всех термов (букв) многочлена, помещенного в обычные или итерационные скобки, подчинены месту, расположенному непосредственно слева от открывающей скобки
2. Место, расположенное непосредственно справа от закрывающей скобки, подчинено конечным местам всех термов многочлена, заключенного в эти скобки; а в случае итерационных скобок – еще и месту, расположенному непосредственно слева от соответствующей открывающей скобки.
3. Начальные места всех термов многочлена, заключенного в итерационные скобки, подчинены месту, расположенному непосредственно справа от соответствующей закрывающей скобки.
4. Если место с подчинено месту b, а место b подчинено месту а, то место с подчинено месту а.
5. Каждое место подчинено самому себе.
6. Других случаев подчинения мест в R нет.

Процесс синтеза КНА

1. Разметка мест:
   1. Определите основные и предосновные места для регулярного выражения.
   2. Установите связи подчинения в соответствии с правилами подчинения мест.
2. Инициализация автомата:
   1. Создайте начальное состояние q1q\_1q1​, включающее все основные места начальных термов регулярного выражения.
3. Построение переходов:
   1. Для каждого символа aaa алфавита XXX:
   2. Определите, какие состояния qjq\_jqj​ можно достигнуть из состояния qiq\_iqi​ при прочтении символа aaa. Это возможно, если начальное место выражения RiR\_iRi​ связано с конечным местом символом aaa.
4. Определение конечных состояний:
   1. Состояние является конечным, если любое место, входящее в qiq\_iqi​, связано с конечным местом всего регулярного выражения.

| < | x | I | < | b | I | d | > | > | m | < | a | > | ( | b | I | d | ) |

0 | 1 | | 2 | 3 | | 4 | 5 | | 6 | 7 |

0 0 0 0 0 0 4 4 4 4 6

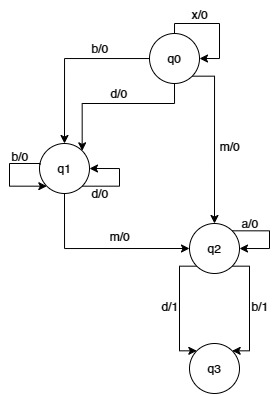
1 1 1 1 1 1 5 5 5 5 7

2 2 2 2 2 2

3 3 3 3 3 3

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| X\Q | q0 | q1 | q2 | q3 | q4 | q5 | q6 | q7 |
| x | q1/0 | q1/0 | q1/0 | q1/0 | - | - | - | - |
| b | q2/0 | q2/0 | q2/0 | q2/0 | q6/1 | q6/1 | - | - |
| d | q3/0 | q3/0 | q3/0 | q3/0 | q7/1 | q7/1 | - | - |
| m | q4/0 | q4/0 | q4/0 | q4/0 | - | - | - | - |
| a | - | - | - | - | q5/0 | q5/0 | - | - |

Ориентированный граф(мультиграф)



Матрица переходов:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **q\x** | **x** | **b** | **d** | **m** | **a** |
| **q0** | q0 | q1 | q1 | q2 | - |
| **q1** | - | q1 | q1 | q2 | - |
| **q2** | - | q3 | q3 | - | q2 |
| **q3** | - | - | - | - | - |

Матрица выходов:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **q\x** | **x** | **b** | **d** | **m** | **a** |
| **q0** | 0 | 0 | 0 | 0 | - |
| **q1** | - | 0 | 0 | 0 | - |
| **q2** | - | 1 | 1 | - | 0 |
| **q3** | - | - | - | - | - |

Автоматная матрица

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **q\q** | **q0** | **q1** | **q2** | **q3** |
| **q0** | x\0 | b\0, d\0 | m\0 | - |
| **q1** | - | b\0, d\0 | m\0 | - |
| **q2** | - | - | a\0 | b\1, d\1 |
| **q3** | - | - | - | - |

Результат работы программы:

q0 q1 q2 q3 q4 q5 q6 q7

x q1/0 q1/0 q1/0 q1/0 - - - -

b q2/0 q2/0 q2/0 q2/0 q5/1 - - q5/1

d q3/0 q3/0 q3/0 q3/0 q6/1 - - q6/1

m q4/0 q4/0 q4/0 q4/0 - - - -

a - - - - q7/0 - - q7/0

Минимизируем этот КНА:

1)Делим состояния по входным символам:

q0, q1, q2, q3

q4, q7

q5, q6

2) q0-> q0

q1, q2, q3 ->q1

q4, q7 - >q2

q5, q6 -> q3

Состояния повторились - заканчиваем минимизацию, новая таблица переходов/выходов:

q0 q1 q2 q3

x q0/0 - - -

b q1/0 q1/0 q3/1 -

d q1/0 q1/0 q3/1 -

m q2/0 q2/0 q2/0 -

a - - - -

Новая матрица соответствует матрице, составленной в предыдущей лабораторной работе.

**Вывод**

В ходе работы были закреплены алгоритмы синтеза конечных автоматов по регулярным выражениям.

Листинг программы:

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

#include <algorithm>

using namespace std;

struct row*//заголовок строки таблицы 4 шага. Содержит для облегчения дальнейших шагов позиции символа*

{

    string symbol; *//символ регулярного выражения*

    vector <int> place; *//позиции символа в регулярном выражении*

};

void read\_file(string*&* st, string name);

void markdown(vector <vector <int>>*&* delimiter, string str, int begin, int end, int*&* count, vector<row>*&* r);

void dop(vector <vector <int>>*&* delimiter, string str, int begin, int end);

void automaton(vector<vector<vector <int>>>*&* cell, vector<row>rOfTable, vector<vector <int>>*&* cOfTable, vector <vector <int>> delimiter, vector<bool>*&* correct);

void write\_to\_file(vector<vector<vector <int>>> cell, vector<row>rOfTable, vector<vector <int>>cOfTable, vector<bool>correctly);

int closing\_bracket(string str, int begin);

int find(vector<row> r, char s[1]);

int main()

{

    setlocale(LC\_ALL, "rus");

    string str;

*//считывает содержимое файла в строку*

    read\_file(str, "input.txt");

    vector <vector <int>> delimiter(str.size() + 1, vector<int>());

    int count = 1;

    vector<row>rOfTable;*//заголовки строк таблицы*

    vector<bool>correct;

    vector<vector <int>>cOfTable;*//заголовки столбцов таблицы*

    cOfTable.push\_back(vector < int > {0});

    correct.push\_back(false);

*// Запуск алгоритма разбиения на дизъюнктивные члены и построения конечного автомата*

    markdown(delimiter, str, 0, str.size(), count, rOfTable);

    vector<vector<vector <int>>> cell(rOfTable.size(), vector<vector <int>>());*//первый вектор-строка, второй-столбец, третий - ячейка*

    automaton(cell, rOfTable, cOfTable, delimiter, correct);

*// Запись результатов в файл*

    write\_to\_file(cell, rOfTable, cOfTable, correct);

    system("pause");

*return* 0;

}

*// Функция записи результатов в файл*

void write\_to\_file(vector<vector<vector <int>>> cell, vector<row>rOfTable, vector<vector <int>>cOfTable, vector<bool>correct)

{

    ofstream file("result.txt");

    vector<vector<int>>::iterator it;

*// Запись заголовков столбцов*

    for (int i = 0; i < cOfTable.size(); i++)

    {

        file << '\t' << i;

    }

*// Запись таблицы переходов в файл*

    for (int i = 0; i < rOfTable.size(); i++)

    {

        file << '\n' << rOfTable[i].symbol << '\t';

        for (int j = 0; j < cOfTable.size(); j++)

        {

            if (cell[i][j].empty())

                file << "-\t";

            else

            {

*// Поиск позиции в заголовке столбца*

                it = find(cOfTable.begin(), cOfTable.end(), cell[i][j]);

                file << it - cOfTable.begin() << '/';

                if (correct[it - cOfTable.begin()] == true)

                    file << '1';

                else

                    file << '0';

                file << '\t';

            }

        }

    }

    file.close();

}

*// Функция построения конечного автомата*

void automaton(vector<vector<vector <int>>>*&* cell, vector<row>rOfTable, vector<vector <int>>*&* cOfTable, vector <vector <int>> delimiter, vector<bool>*&* correct)

{

    int column = 0;

*// Перебор столбцов*

    while (column < cOfTable.size())

    {

*// Проверка, является ли состояние конечным*

        for (int i = 0; i < cOfTable[column].size(); i++)*//состояние является конечным?*

        {

            for (int j = 0; j < delimiter[delimiter.size() - 1].size(); j++)

            {

                if (cOfTable[column][i] == delimiter[delimiter.size() - 1][j])

                {

                    correct[column] = true;

                    break;

                }

            }

            if (correct[column] == true)

                break;

        }

*// Построение таблицы переходов для каждой строки*

*// Создание ячеек в таблице автомата*

        for (int i = 0; i < rOfTable.size(); i++)*//каждая строка*

        {

            cell[i].push\_back(vector<int>());

*//для каждого вхождения символа во входную строку*

            for (int j = 0; j < rOfTable[i].place.size(); j++)

            {

*//дл€ каждой позиции в заголовке столбца*

                for (int k = 0; k < cOfTable[column].size(); k++)

                {

                    if (find(delimiter[rOfTable[i].place[j]].begin()*/\*предосновное место этого символа i в месте j\*/*, delimiter[rOfTable[i].place[j]].end(), cOfTable[column][k]) != delimiter[rOfTable[i].place[j]].end()

*//в этом предосновном месте найдена позиция из заголовка столбца*

                        && find(cell[i][column].begin(), cell[i][column].end(), delimiter[rOfTable[i].place[j] + 1][0]) == cell[i][column].end())

*//и позиции основного места нет в ячейки*

                        cell[i][column].push\_back(delimiter[rOfTable[i].place[j] + 1][0]); *//значит добавляем позицию основного места в ячейку*

                }

            }

*// Сортировка ячейки*

            sort(cell[i][column].begin(), cell[i][column].end());

*// Проверка наличия нового заголовка в таблице*

*// Если заголовка нет в таблице, добавляем его*

            if (find(cOfTable.begin(), cOfTable.end(), cell[i][column]) == cOfTable.end() && !cell[i][column].empty())*//если такого заголовка нет*

            {

                cOfTable.push\_back(cell[i][column]);*//добавляем*

                correct.push\_back(false);

            }

        }

        column++;

    }

}

*//строит таблицу разделителей, которая содержит информацию о позициях символов и их отношениях.*

void markdown(vector <vector <int>>*&* delimiter, string str, int begin, int end, int*&* count, vector<row>*&* r)*// count = 1*

{

    vector<int>dMemberB;

    vector<int>dMemberE;

    int check;

    for (int i = begin; i < end; i++)

    {

        if (i == begin)

        {

*// Начальная обработка символа в регулярном выражении*

            if (i == 0)

            {

                delimiter[0].push\_back(0);

*// Если первый символ не является одной из скобок*

                if (str[0] != '<' && str[0] != '>' && str[0] != '(' && str[0] != ')')

                    dMemberB.push\_back(0);

            }

            else

            {

*// Копируем предыдущие разделители в новый блок*

                for (int j = 0; j < delimiter[i - 1].size(); j++)

                    delimiter[i].push\_back(delimiter[i - 1][j]);

                dMemberB.push\_back(i);*//начальные места дизъюнктивных членов*

            }

        }

        if (str[i] != '<' && str[i] != '>' && str[i] != '(' && str[i] != ')' && str[i] != '|')*//отмечаю основные места*

        {

*// Отмечаем основные места*

            check = find(r, &str[i]);

            if (check == -1)

                r.push\_back(row{ { str[i] },{ i } });

            else

                r[check].place.push\_back(i);

            delimiter[i + 1].push\_back(count);

            count++;

        }

        if (i > 0)

        {

            if (str[i] != '>' && str[i] != ')' && str[i] != '|' && str[i - 1] == '|')

*//Индекс места перед любыми скобками распростран€етс€ на начальные места*

*//всех дизъюнктивных членов, записанных в этих скобках*

            {

                if (begin == 0)

                {

                    for (int j = 0; j < delimiter[begin].size(); j++)

                        delimiter[i].push\_back(delimiter[begin][j]);

                }

                else

                {

                    for (int j = 0; j < delimiter[begin - 1].size(); j++)

                        delimiter[i].push\_back(delimiter[begin - 1][j]);

                }

                dMemberB.push\_back(i);*//начальные места дизъюнктивных членов*

                dMemberE.push\_back(i - 1);

            }

        }

        if (i == end - 1)

        {

            if (str[end - 1] == ')' || str[end - 1] == '>')

*//отмечаю конец последнего диз члена в скобках*

                dMemberE.push\_back(end - 1);

            for (int j = 0; j < dMemberE.size(); j++)

*//Индекс конечного места любого дизъюнктивного члена, заключенного в любые скобки,*

*//распространяется на место, непосредственно следующее за этими скобками.*

            {

                for (int k = 0; k < delimiter[dMemberE[j]].size(); k++)

                    delimiter[end].push\_back(delimiter[dMemberE[j]][k]);

            }

            if (str[end - 1] == '>')

            {

                for (int j = 0; j < delimiter[begin - 1].size(); j++)

*//индекс места перед итерационными скобками распространяется на место,*

*//непосредственно следующее за этими скобками*

                    if (find(delimiter[end].begin(), delimiter[end].end(), delimiter[begin - 1][j]) == delimiter[end].end())

                        delimiter[end].push\_back(delimiter[begin - 1][j]);

                for (int j = 0; j < dMemberB.size(); j++)

*//индекс места за итерационными скобками распространяется*

*//на начальные места всех дизъюнктивных членов, заключенных в итерационные скобки*

                {

                    for (int k = 0; k < delimiter[end].size(); k++)

                    {

                        if (find(delimiter[dMemberB[j]].begin(), delimiter[dMemberB[j]].end(), delimiter[end][k]) == delimiter[dMemberB[j]].end())

                            delimiter[dMemberB[j]].push\_back(delimiter[end][k]);

                    }

*// Если символ в начальной позиции - открывающая скобка, значит,*

*// надо добавить на начальные места дизъюнктивных членов новые индексы*

                    if (str[dMemberB[j]] == '<' || str[dMemberB[j]] == '(')*//значит надо добавить на начальные места диз членов новые индексы*

                    {

                        dop(delimiter, str, dMemberB[j] + 1, closing\_bracket(str, dMemberB[j]));

                    }

                }

            }

        }

*// Обработка скобок*

        if (str[i] == '<' || str[i] == '(')

        {

            int zSc = closing\_bracket(str, i);

            markdown(delimiter, str, i + 1, zSc + 1, count, r);

            i = zSc; *// Пропускаем уже обработанный блок*

        }

    }

}

void dop(vector <vector <int>>*&* delimiter, string str, int begin, int end)

{

*// Функция добавления начальных мест дизъюнктивных членов в таблицу разделителей*

    for (int i = begin; i < end; i++)

    {

        if (i == begin)

        {

*//добавляем в начальное место, сразу за скобками*

            for (int j = 0; j < delimiter[begin - 1].size(); j++)

            {

                if (find(delimiter[begin].begin(), delimiter[begin].end(), delimiter[begin - 1][j]) == delimiter[begin].end())

                    delimiter[begin].push\_back(delimiter[begin - 1][j]);

            }

        }

        if (str[i] != '>' && str[i] != ')' && str[i] != '|' && str[i - 1] == '|')*// добавляем всем остальным диз членам в скобках*

        {

*// Добавляем всем остальным дизъюнктивным членам в скобках*

            for (int j = 0; j < delimiter[begin - 1].size(); j++)

            {

                if (find(delimiter[i].begin(), delimiter[i].end(), delimiter[begin - 1][j]) == delimiter[i].end())

                    delimiter[i].push\_back(delimiter[begin - 1][j]);

            }

        }

        if ((str[i] == '<' || str[i] == '(') && (str[i - 1] == '|' || str[i - 1] == '(' || str[i - 1] == '<'))

        {

*// Если встречаем открывающую скобку внутри разделителей*

            int zSc = closing\_bracket(str, i);

*// Рекурсивно вызываем функцию для обработки вложенной скобки*

            dop(delimiter, str, i + 1, zSc);

*// Пропускаем уже обработанный блок*

            i = zSc;

        }

    }

}

*// Функция поиска символа в таблице разделителей*

int find(vector<row> r, char s[1])

{

    for (int j = 0; j < r.size(); j++)

    {

        if (s[0] == r[j].symbol[0])

*return* j;

    }

*return* -1;

}

*// Функция разбора скобок*

int closing\_bracket(string str, int begin)

{

    int count = 1;

    string brackets;

*// Определение типа скобок (<> или ())*

    if (str[begin] == '<')

        brackets = "<>";

    else

        brackets = "()";

*// Поиск соответствующей закрывающей скобки*

    while (true)

    {

        begin++;

        if (str[begin] == brackets[0])

            count++;

        if (str[begin] == brackets[1])

        {

            count--;

            if (count == 0)

*return* begin;*// Возвращаем индекс закрывающей скобки*

        }

    }

}

void read\_file(string*&* st, string name)

{

    string temporary;

    ifstream file(name);

    if (!file.is\_open())

        cout << "Файл не может быть открыт!\n";

    else

    {

        string str;

        getline(file, str);

        if (str.empty()) cout << "Файл пуст!" << endl;

        else

        {

            st = str;

        }

    }

    file.close();

*return*;

}