Реализация конечного автомата для регулярного языка

Тотиков Сармат   
Группа: 21.Б13-пу

**Задание:**

1. Разработать распознающий КА для цепочек определенного в задании регулярного языка над заданным алфавитом и реализовать его в виде подпрограммы.
2. Разработать программу поиска цепочек этого языка в последовательности символов, используя подпрограмму пункта  
     
   Вход: последовательность символов алфавита регулярного языка задания  
      
   Выход: последовательность строк формата номер: цепочка  
     
   **номер** – номер позиции начала цепочки языка в последовательности символов   
   **цепочка** – найденная цепочка языка в последовательности символов   
     
   Если цепочек языка не найдено, то на выходе отображается **«цепочек не найдено»**

Строка символов **a, b, (, ), %**, начинающаяся с префикса %и заканчивающаяся суффиксом %, между которыми располагается последовательность одинарных **(….)** или двойных **((.....))** парных скобок, содержащих внутри цепочку четных символов **a**, за которой следует нечетное число символов **b**, например **%(aabbb)(b)((aab))%.**

**Диаграмма переходов и таблица переходов конечного автомата:**

Какой тип Вашего КА: ДКА или НКА? Ответьте!

Построил ДКА. Описание переходов отображенных и не отображенных на диаграмме представил и объяснил ниже.

Дорогой Сармат! Я Вас такому не учил, ДКА – ДЕТЕРМИНИРОВАННЫЙ КА, а, следовательно, он точно знает, в какое ОДНО состояние должен перейти, когда получает символ алфавита!

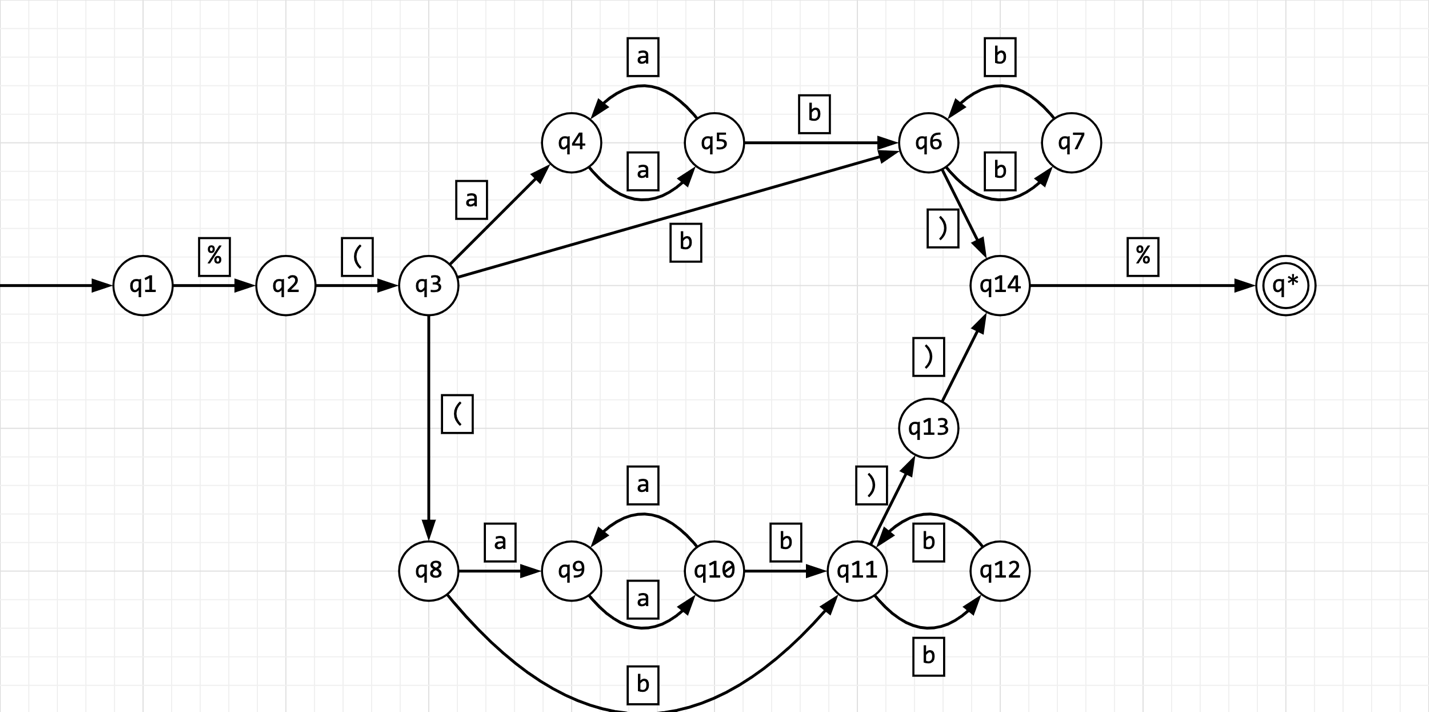
Диаграмма:  


Таблица переходов:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **%** | **(** | **A** | **b** | **)** |
| ** q1** | q2 |  |  |  |  |
| **q2** |  | q3 |  |  |  |
| **q3** |  | q8 | q4 | q6 |  |
| **q4** |  |  | q5 |  |  |
| **q5** |  |  | q4 | q6 |  |
| **q6** |  |  |  | q7 | q14 |
| **q7** |  |  |  | q6 |  |
| **q8** |  |  | q9 | q11 |  |
| **q9** |  |  | q10 |  |  |
| **q10** |  |  | q9 | q11 |  |
| **q11** |  |  |  | q12 | q13 |
| **q12** |  |  |  | q11 |  |
| **q13** |  |  |  |  | q14 |
| **q14** | q14 |  |  |  |  |
| **q\*** |  |  |  |  |  |
|  |  |  |  |  |  |

Если НКА, то исправьте таблицу переходов: он находится во множестве состояний. А последняя строка зачем? Разъясните? Где у Вас на диаграмме автомата состояние  **?   
В диаграмме не отображаются переходы в дьявольское состояние, однако в случае если в каком-то из состояний мы получим символ, переход по которому не отображен в диаграмме, мы перейдем в дьявольское состояние.**

Куда перейдет Ваш ДКА из состояния, например, q5, если придет символ "("?  
В данном случае он перейдёт в дьявольское состояние. В подобных случаях аналогично.

Для ДКА переходы для каждого состояния по всем символам должны быть определены! Исправьте таблицу переходов и дайте к диаграмме переходов пояснение (если это ДКА), куда выполняется переход из состояний по символам, которые отсутствуют для данного состояния на диаграмме.

**Дорогой Сармат! Проясните все определения и дайте правильные ответы на мои вопросы.**

**Состояния:**

|  |
| --- |
| ** q1 – начальное состояние** |
| **q2 – после прочтения префикса** % |
| **q3 – после прочтения одной открывающей скобки (** |
| **q4 – после прочтения первого символа** а **или после каждого нечетного символа** а |
| **q5 - после прочтения второго** а **или после каждого четного символа** а |
| **q6 – после первого символа** b **или после каждого нечетного** |
| **q7 – после каждого четного символа** b |
| **q8 – после второй открывающей скобки (** |
| **q9 – после первого символа** а **или после каждого нечетного** |
| **q10 - после каждого четного символа** а |
| **q11 – после первого символа** b **или после каждого нечетного** |
| **q12 - после каждого четного символа** b |
| **q13 – после первой закрывающей скобки )** |
| **q14 - после второй закрывающей скобки )** |
| **q\* - конечное состояние после суффикса %** |
| - **дьявольское состояние** |

**Регулярное выражение, задающее язык задания:**

Символы алфавита: %, a, b, (, )

Окрашу их в красный цвет для большей читаемости.  
**%** ( **(** (**aa**)\* **b** (**bb**)\* **) + ((** (**aa**)\* **b** (**bb**)\* **))** ) **%**

РВ верно!

Программа, описание и тесты проверил, все правильно.

**Листинг программ:**

#include <iostream>

// Функция образующая необходимый конечный автомат.

// Она обрабатывает скобку от '%' до '%' и выдает результат true или false

**bool** KA(std::string in)

{

**if**(in[0] == in[1]) **return** **false**; // проверка на цепочку вида: "%%"

**char** c; // c - бегунок по тексту

**bool** two\_br\_opened = **false**; // проверка на открытие двух скобок

**bool** one\_br\_opened = **false**; // проверка на открытие одной скобки

**int** a\_counter = 0; // проверка на четность 'a'

**int** b\_counter = 0; // проверка на нечетность 'b'

**int** idx = 1; // индекс перемещения по строке

c = in[idx]; // передаем бегунку значение символа после '%'

// Цикл, который проходится по переданной строке

**do**

{

c = in[idx];

**if**(c == '(') // Если символ '(', то проверяем одинарная скобка или двойная

{

**if**(in[idx + 1] == '(' )

{

// Если есть уже открытые скобки, тогда строка нас не устраивает.

**if**(two\_br\_opened == **true** || one\_br\_opened == **true**)

{

//std::cout << "Ошибка в расстановке скобок. Не подходит. " << std::endl;

**return** **false**;

}

two\_br\_opened = **true**;

idx += 2;

c = in[idx];

} **else** {

// Если есть уже открытые скобки, тогда строка нас не устраивает.

**if**(two\_br\_opened == **true** || one\_br\_opened == **true**)

{

//std::cout << "Ошибка в расстановке скобок. Не подходит. " << std::endl;

**return** **false**;

}

one\_br\_opened = **true**; //Показываем, что уже есть открытые скобки.

++idx; //Смещаемся на символ вперед. Это либо 'a', либо 'b'

c = in[idx];

}

// // проверка на наличие букв в скобках

// if((in[idx] == 'a') || (in[idx] == 'b'))

// {

// std::cout << "�' скобках есть буквы. �'с�' проходит. " << std::endl;

// } else {

// std::cout << "Ошибка. Пустые скобки. Не подходит. " << std::endl;

// return false;

// }

// Проверка на наличие букв в скобках и на то, что скобки открыты.

**if**((c == 'a' || c == 'b') && (one\_br\_opened || two\_br\_opened))

{

// Считываем 'a'

**while**(c == 'a')

{

++a\_counter;

++idx;

c = in[idx];

}

// Проверяем количество 'a' на четность и на то, что есть хотя бы одна 'b'

**if**((a\_counter % 2 != 0) || (c != 'b'))

{

//std::cout << "Нечетное количество 'a'. Не подходит. " << std::endl;

**return** **false**;

}

// считываем 'b'

**while**(c == 'b')

{

++b\_counter;

++idx;

c = in[idx];

// проверяем на то, что последовательность символов верна

**if**(in[idx] == 'a')

{

//std::cout << "'a' после 'b'. Не подходит. " << std::endl;

**return** **false**;

}

}

// Проверяем количество 'b' на нечетность

**if**(b\_counter % 2 == 0)

{

//std::cout << "Четное количество 'b'. Не подходит. " << std::endl;

**return** **false**;

}

// проверка на закрывающие скобки.

**if**(c == ')')

{

//проверка на двойную закрывающую скобку.

**if**(in[idx + 1] == ')')

{

// показываем, что двойная скобка закрылась.

two\_br\_opened = **false**;

idx += 2;

c = in[idx];

// проверка на случай, если у нас одна открывающая скобка, но две закрывающие

**if**(one\_br\_opened == **true**)

{

//std::cout << "Ошибка в расстановке скобок. Не подходит. " << std::endl;

**return** **false**;

}

// удаляем счетчики под 'a' и 'b'

a\_counter = 0;

b\_counter = 0;

} **else** {

// показываем, что одинарная скобка закрылась.

one\_br\_opened = **false**;

++idx;

c = in[idx];

// проверка на случай, когда у нас две открывающие, но одна закрывающая скобка

**if**(two\_br\_opened == **true**)

{

//std::cout << "Ошибка в расстановке скобок. Не подходит. " << std::endl;

**return** **false**;

}

// удаляем счетчики под 'a' и 'b'

a\_counter = 0;

b\_counter = 0;

}

} **else** {

// Если закрывающих скобок не будет, то цепочка не подходит.

//std::cout << "Ошибка в расстановке скобок. Нет закрывающей. Не подходит. " << std::endl;

**return** **false**;

}

} **else** {

// Если проверка на непустые скобки не пройдена.

//std::cout << "Ошибка. Пустые скобки. Не подходит. " << std::endl;

**return** **false**;

}

}

// проверка на случай, если нет открывающих скобок, но есть закрывающие

**if**(in[idx - 1] == '%' && in[idx] != '(')

{

//std::cout << "Ошибка в расстоновке скобок. Не подходит. " << std::endl;

**return** **false**;

}

} **while**(c != '%');

//std::cout << "Всё подходит. " << std::endl;

**return** **true**;

}

// Функция, которая считывает поступившую строку и отправляет в КА '%' до '%'

**void** recognition(std::string in)

{

std::string out = ""; // строка, которая передается КА

**char** c; // бегунок по строке

**int** place = 0; // место начала строки

**bool** isempty = **true**; // переменная, показывающая, что подходящих цепочек нет.

// цикл по строке

**for**(**int** idx = 0; idx < in.length(); ++idx)

{

c = in[idx];

// Начала обработки строки.

**if**(c == '%')

{

out += c;

place = idx + 1; // записываем позицию первого символа

++idx;

c = in[idx];

// считываем строку до '%'

**while**(c != '%')

{

out += c;

++idx;

c = in[idx];

**if**(idx > in.length()) **break**; // Проверка на выход из строки

// Необходимо на случай, если в конце нет '%'

}

**if**(idx > in.length()) **break**; // Аналогичная проверка.

--idx;

out += '%';

// Проверка строки.

**if**(KA(out) == **true**)

{

std::cout << place << "\t:\t" << out << std::endl; // выводим строку

isempty = **false**; // показываем, что есть хотя бы одна подходящая цепочка

}

out = "";

place = 0;

}

}

// если цепочек не найдено, то показываем.

**if**(isempty == **true**)

{

std::cout << "Цепочек не найдено" << std::endl;

}

}

**int** main(**int** argc, **const** **char** \* argv[]) {

// Тесты:

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

// Тест из задания:

std::string test\_01 = "%(aabbb)(b)((aab))%";

std::cout << "=== Test 01: ===" << std::endl;

recognition(test\_01);

std::cout << std::endl;

// Вывод:

// === Test 01: ===

// 1 : %(aabbb)(b)((aab))%

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

// Тест нескольких верных цепочек:

std::string test\_02 = "%(aabbb)%(b)%((aab))%";

std::cout << "=== Test 02: ===" << std::endl;

recognition(test\_02);

std::cout << std::endl;

// Вывод:

// === Test 02: ===

// 1 : %(aabbb)%

// 9 : %(b)%

// 13 : %((aab))%

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

// Тест двух верных цепочек и одной неверной:

std::string test\_03 = "%((aabbb))%(bb)%((aaaab))%";

std::cout << "=== Test 03: ===" << std::endl;

recognition(test\_03);

std::cout << std::endl;

// Вывод:

// === Test 03: ===

// 1 : %((aabbb))%

// 16 : %((aaaab))%

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

// Тест нескольких неверных цепочек:

std::string test\_04 = "%((aabbb)%(bb)%((aaaaab))%";

std::cout << "=== Test 04: ===" << std::endl;

recognition(test\_04);

std::cout << std::endl;

// Вывод:

// === Test 04: ===

// Цепочек не найдено

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

// Тест на неверную последовательность букв:

std::string test\_05 = "%(aaba)%";

std::cout << "=== Test 05: ===" << std::endl;

recognition(test\_05);

std::cout << std::endl;

// Вывод:

// === Test 05: ===

// Цепочек не найдено

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

// Тест на пустые скобки:

std::string test\_06 = "%(())%(aabbb)%";

std::cout << "=== Test 06: ===" << std::endl;

recognition(test\_06);

std::cout << std::endl;

// Вывод:

// === Test 06: ===

// 6 : %(aabbb)%

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

// Тест на ошибку с открытием/закрытием скобок:

std::string test\_07 = "%(%((aab))%";

std::cout << "=== Test 07: ===" << std::endl;

recognition(test\_07);

std::cout << std::endl;

// Вывод:

// === Test 07: ===

// 4 : %((aab))%

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

// Тест на пустую строку:

std::string test\_08 = "";

std::cout << "=== Test 08: ===" << std::endl;

recognition(test\_08);

std::cout << std::endl;

// Вывод:

// === Test 08: ===

// Цепочек не найдено

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

// Тест на строку из процентов:

std::string test\_09 = "%%";

std::cout << "=== Test 09: ===" << std::endl;

recognition(test\_09);

std::cout << std::endl;

// Вывод:

// === Test 09: ===

// Цепочек не найдено

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

// Тест на сторонние символы и регистр:

std::string test\_10 = "%(aAbbb)%((aaccb))%";

std::cout << "=== Test 10: ===" << std::endl;

recognition(test\_10);

std::cout << std::endl;

// Вывод:

// === Test 10: ===

// Цепочек не найдено

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

// Тест на строку без скобки в начале и в конце:

std::string test\_11 = "%aab))%(aab%";

std::cout << "=== Test 11: ===" << std::endl;

recognition(test\_11);

std::cout << std::endl;

// Вывод:

// === Test 11: ===

// Цепочек не найдено

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

// Тест на строку с поздним началом:

std::string test\_12 = "Начало цепочки тут -> %((aab))%";

std::cout << "=== Test 12: ===" << std::endl;

recognition(test\_12);

std::cout << std::endl;

// Вывод:

// === Test 12: ===

// 39 : %((aab))%

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

// Тест на строку с поздним концом:

std::string test\_13 = "%((aab))% <- тут был конец программы(->)";

std::cout << "=== Test 13: ===" << std::endl;

recognition(test\_13);

std::cout << std::endl;

// Вывод:

// === Test 13: ===

// 1 : %((aab))%

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

// Тест на сторонний текст в середине программы:

std::string test\_14 = "%((aab))% <- тут нет нужной цепочки ->%((aab))%";

std::cout << "=== Test 14: ===" << std::endl;

recognition(test\_14);

std::cout << std::endl;

// Вывод:

// === Test 14: ===

// 1 : %((aab))%

// 58 : %((aab))%

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

//Тест на всё вместе:

std::string test\_15 = "Made by: Sarmat Totikov %((aab))%((aab))%((aab))%%((aab))% ";

std::cout << "=== Test 15: ===" << std::endl;

recognition(test\_15);

std::cout << std::endl;

// Вывод:

// === Test 15: ===

// 25 : %((aab))%

// 33 : %((aab))%

// 41 : %((aab))%

// 50 : %((aab))%

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

**return** 0;

}

**Краткое описание работы подпрограммы, реализующей конечный автомат, и программы поиска цепочек языка в последовательности символов:**

Функция **КА** – подпрограмма, реализующая конечный автомат.  
Принцип работы:

1. На вход поступает строка, начинающаяся со знака % и окачивающаяся знаком %.
2. Программа проверяет на то, что это цепочка вида «%%».   
   2.1. Если цепочка вида «%%», то цепочка не подходит
3. Далее. Начинаем цикл, который идет до знака ‘%’.
4. Ищем знак ‘(’
5. Определяемся с количеством ‘(‘ или ‘((’.
6. Считываем количество ‘а’. Делаем проверку на четность.
7. Считываем количество ‘b’. Делаем проверку на четность и отсутствие ‘a’ после ‘b’
8. Проверяем закрывающие скобки. Их количество должно соответствовать количество открывающих. (Выявлено на 5 шаге)
9. Повторяем действия, пока не дойдем до знака ‘%’

Функция **recognition** – программа поиска цепочек языка в последовательности символов:

1. На вход поступает строка из множества символов. Игнорируем все знаки, написанные до ‘%’.
2. С первого ‘%’ до следующего ‘%’ считываем строку и запоминаем ее.  
   2.1. Запоминаем место первого ‘%’
3. Запомненную строку посылаем в **КА**.
4. Получаем ответ от **КА**.   
   a. Если ответ удовлетворительный, то выводим позицию начала и саму строку  
   b. Если ответ неудовлетворительный, то выводим «цепочек не найдено»
5. Завершаем программу

**Таблица тестирования программы:**

|  |  |  |
| --- | --- | --- |
| Номер теста: | Входные данные: | Выходные данные: |
| 1 | %(aabbb)(b)((aab))% | 1 : %(aabbb)(b)((aab))% |
| 2 | %(aabbb)%(b)%((aab))% | 1 : %(aabbb)% 9 : %(b)%  13 : %((aab))% |
| 3 | %((aabbb))%(bb)%((aaaab))% | 1 : %((aabbb))%  16 : %((aaaab))% |
| 4 | %((aabbb)%(bb)%((aaaaab))% | Цепочек не найдено |
| 5 | %(aaba)% | Цепочек не найдено |
| 6 | %(())%(aabbb)% | 6 : %(aabbb)% |
| 7 | %(%((aab))% | 4 : %((aab))% |
| 8 | (Пустая строка) | Цепочек не найдено |
| 9 | %% | Цепочек не найдено |
| 10 | %(aAbbb)%((aaccb))% | Цепочек не найдено |
| 11 | %aab))%(aab% | Цепочек не найдено |
| 12 | Начало цепочки тут -> %((aab))% | 39 : %((aab))% |
| 13 | %((aab))% <- тут был конец программы(->) | 1 : %((aab))% |
| 14 | %((aab))% <- тут нет нужной цепочки ->%((aab))% | 1 : %((aab))%  58 : %((aab))% |
| 15 | Made by: Sarmat Totikov %((aab))%((aab))%((aab))%%((aab))% | 25 : %((aab))%  33 : %((aab))%  41 : %((aab))%  50 : %((aab))% |