Автоматные алгоритмические модели

Лекция 6

План лекции

1. Машина Тьюринга.
2. Конечные автоматы.
3. Простейшая реализация конечного автомата.
4. Реализация конечного автомата на языке C#.
5. Библиотека Stateless.

Введение

Автоматные модели можно отнести к наиболее простым алгоритмическим моделям.

Машина Тьюринга

Машина Тьюринга - модель вычислительного механизма (1936)

А. Тьюринг
(1912–1954)

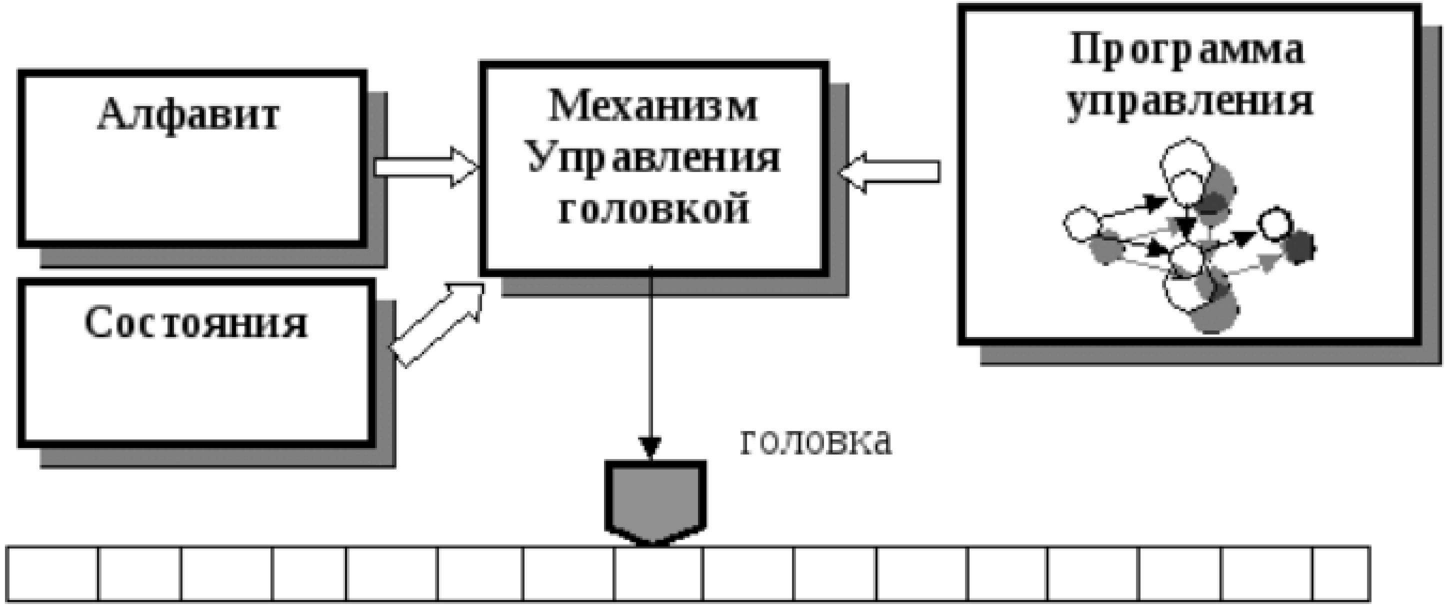


Рис. 1. Машина Тьюринга

Машина Тьюринга

В состав МТ входят:

1. Память в виде бесконечной в обе стороны ленты, разбитой на ячейки, в каждую из которых может быть записан один символ из алфавита *A*=$\{a_0, a_1, \dots, a_n\}$, причем a_0 является пустым символом.
2. Головка записи/чтения, способная читать и записывать символ в текущей ячейке.
3. Автомат, сдвигающий ленту влево или вправо относительно головки. Автомат подчиняется множеству команд *D* = {*L, R, H*}, что означает ”влево”, ”вправо”, ”на месте”.
4. Устройство управления (УУ), осуществляющее выработку команд на движение ленты, запись символов в текущую ячейку памяти в зависимости от текущего символа ячейки и текущего состояния машины, задаваемого символом из алфавита *Q* = $\{q_0, q_1, \dots, q_m\}$.

Машина Тьюринга

Строение машины Тьюринга

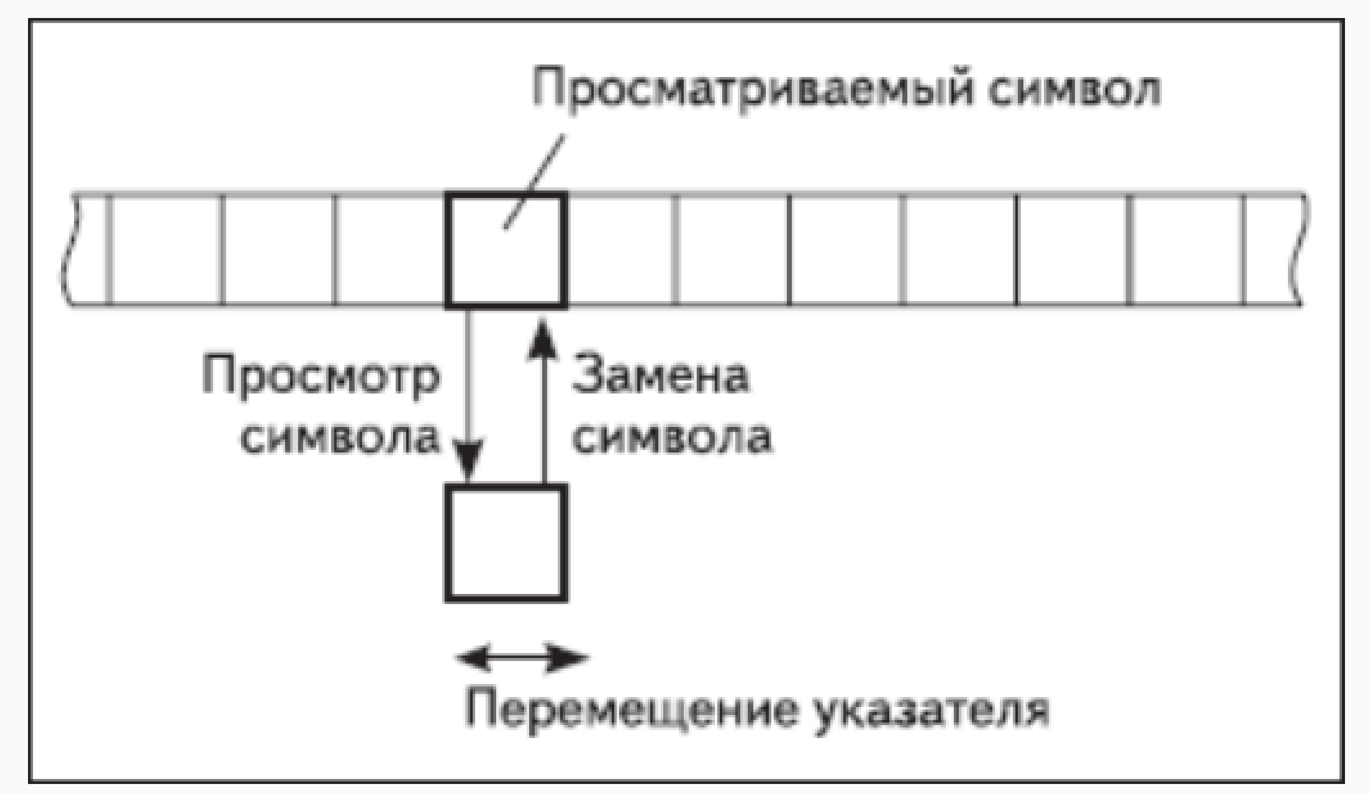


Рис. 2. Строение и работа машины Тьюринга

Машина Тьюринга

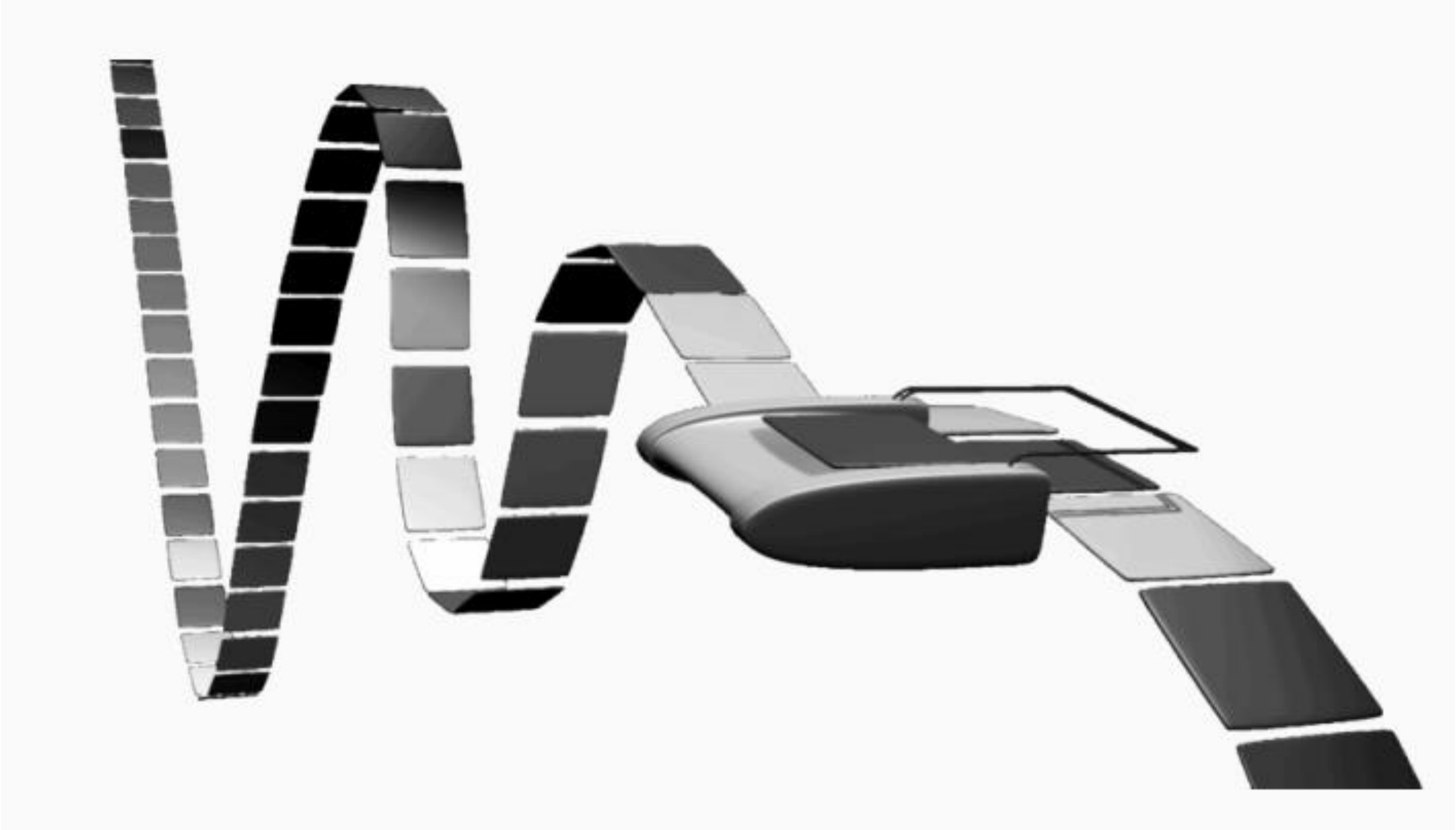


Рис. 3. Современное изображение МТ

Машина Тьюринга

Принцип работы МТ:

1. В начальный момент на ленте находится непустое слово (входное слово), УУ находится в начальном состоянии q_0, а головка записи/чтения находится над левым символом входного слова.
2. На каждом шаге работы головка считывает текущий символ с ленты и в зависимости от него и текущего состояния могут выполняться следующие действия:

- в текущую ячейку записывается новый символ;
- изменяется состояние УУ;
- автомат осуществляет сдвиг ленты по команде или остается на месте

Машина Тьюринга

Принцип работы МТ:

3. Если в результате некоторого количества шагов МТ переходит в состояние, при котором состояние УУ не меняется, символ не меняется и движения ленты не происходит, то говорят, что МТ переходит в заключительное состояние останова. При этом слово, оставшееся на ленте и будет является результатом работы алгоритма.
4. Действия МТ записываются в виде таблицы правил, которые имеют вид

$$q_i a_j \to q_{i1} a_{j1} d_k$$

Набор правил и является программой, по которой работает МТ.

Машина Тьюринга. Добавление единицы

Реализуем функцию увеличения числа на единицу: *f(n) = n + 1*

Запись правил (программы)

1. $\mathbf{q_0}* \rightarrow \mathbf{q_1R}$ - если текущий символ $*$, а состояние q_0, то перейти в состояние q_1 и сдвинуться вправо.
2. $\mathbf{q_11} \rightarrow \mathbf{q_1R}$ - если текущее состояние q_1, а символ в ячейке 1, то не изменяя состояние сдвинуться вправо.
3. $\mathbf{q_1}* \rightarrow \mathbf{q_21}$ - если текущее состояние q_1, а символ в ячейке $*$, то записать в ячейку 1 и остаться на месте.
4. $\mathbf{q_21} \rightarrow \mathbf{q_2L}$ - если текущее состояние q_2, а символ в ячейке 1, то сдвинуться влево.
5. $\mathbf{q_2}* \rightarrow \mathbf{q_3}*$ - состояние q_3 является состоянием *останова*

Машина Тьюринга. Добавление единицы

Итак, если на вход МТ подается лента с записанным входным словом

$$*111*$$

то после работы программы на ленте останется слово

$$*1111$$

Машина Тьюринга. Сложение двух чисел

Запись правил (программы)

1. $\mathbf{q_0 *} \rightarrow \mathbf{q_1 R}$
2. $\mathbf{q_1 1} \rightarrow \mathbf{q_1 R}$
3. $\mathbf{q_1 *} \rightarrow \mathbf{q_2 1}$
4. $\mathbf{q_2 1} \rightarrow \mathbf{q_2 R}$
5. $\mathbf{q_2 *} \rightarrow \mathbf{q_3 L}$
6. $\mathbf{q_3 1} \rightarrow \mathbf{q_4 *}$
7. $\mathbf{q_4 *} \rightarrow \mathbf{q_5 L}$
8. $\mathbf{q_5 1} \rightarrow \mathbf{q_5 L}$
9. $\mathbf{q_5 *} \rightarrow \mathbf{q_6 *}$

Состояние q_6 в этой программе является состоянием останова.

Машина Тьюринга. Реализация умножения

Рассмотрим программу для умножения двух чисел в унарной системе счисления.

Входное слово имеет вид

$*$111..111×111...11 = $*$

где символы $*$ обозначают границы входного слова, единицы задают разряды перемножаемых чисел, а x - знак умножения (и разделитель чисел).

Полученное в результате перемножения число будет записано справа от символа равенства =.

Машина Тьюринга. Реализация умножения

Запись правил (программы)

q0∗→q0R	q4a→q4aR
q01→q0R	q4=→q4=R
q0×→q1×R	q41→q41R
q11→q2aR	q4∗→q51R
q21→q21L	q5∗→q2∗L
q2a→q2aL	q6a→q61R
q2=→q2=L	q6×→q7×R
q2×→q3×L	q7a→q7aR
q31→q4aR	q71→q2aR
q3a→q3aL	q7=→q8=L
q3∗→q6∗R	q8a→q81L
q4×→q4×R	q8×→q9H

Машина Тьюринга. Реализация умножения

Так, при входном слове

$$*111\times 11 = *$$

Получаем выходное слово

$$*111\times 111 = 111111*$$

за 152 шага.

Машина Тьюринга в природе

Для производства белков в клетке с помощью сложно устроенного фермента — РНК-полимеразы — считывается информация с ДНК, своего рода информационной ленты машины Тьюринга.

Здесь, правда, не происходит перезапись ячеек самой ленты, но в остальном процесс весьма похож:

РНК-полимераза садится на ДНК и двигается по ней в одном направлении, при этом она синтезирует нить РНК — нуклеиновой кислоты, сходной с ДНК.

Готовая РНК, отсоединяясь от фермента, несёт информацию к клеточным органеллам, в которых производятся белки.

Машина Тьюринга в природе

Ещё более похож на машину Тьюринга процесс исправления ошибок в ДНК — её репарация.

Здесь ДНК-полимераза вместе с другими белками двигается по ленте ДНК и считывает обе её половинки (геномная ДНК, как известно, представляет собой две переплетенных нити, несущих одну и ту же информацию).

Если информация в половинках не совпадает, ДНК полимераза принимает одну из них за образец и «правит» другую.

Конечные автоматы

Абстрактный автомат А задается шестеркой: *A = (S, I, O, Fs, Fo,* s_0):

1. *S* = s_1, …, s_k – множество состояний (алфавит состояний);
2. *I* = i_1, …, i_M – множество входных символов (входной алфавит);
3. *O* = O_1, …, O_N – множество выходных символов (выходной алфавит);
4. *Fs* : *S×I→S* – функция переходов, отображающая *DFs* ⊆ *S×I в S*. Функция *Fs* некоторым парам состояние – входной символ (s_k, i_m) ставит в соответствие состояние автомата s_i = *Fs*(s_k, i_m), s_i∈ S ;
5. Fo : *S×I→O* – функция выходов, отображающая *DFo* ⊆ *S×I* в O. Функция Fo некоторым парам состояние – входной символ (s_k, i_m) ставит в соответствие выходные символы автомата, O_n = *Fs*(s_k, i_m), O_n∈ O ;
6. s_0 ∈ *S* – начальное состояние автомата.

Конечные автоматы

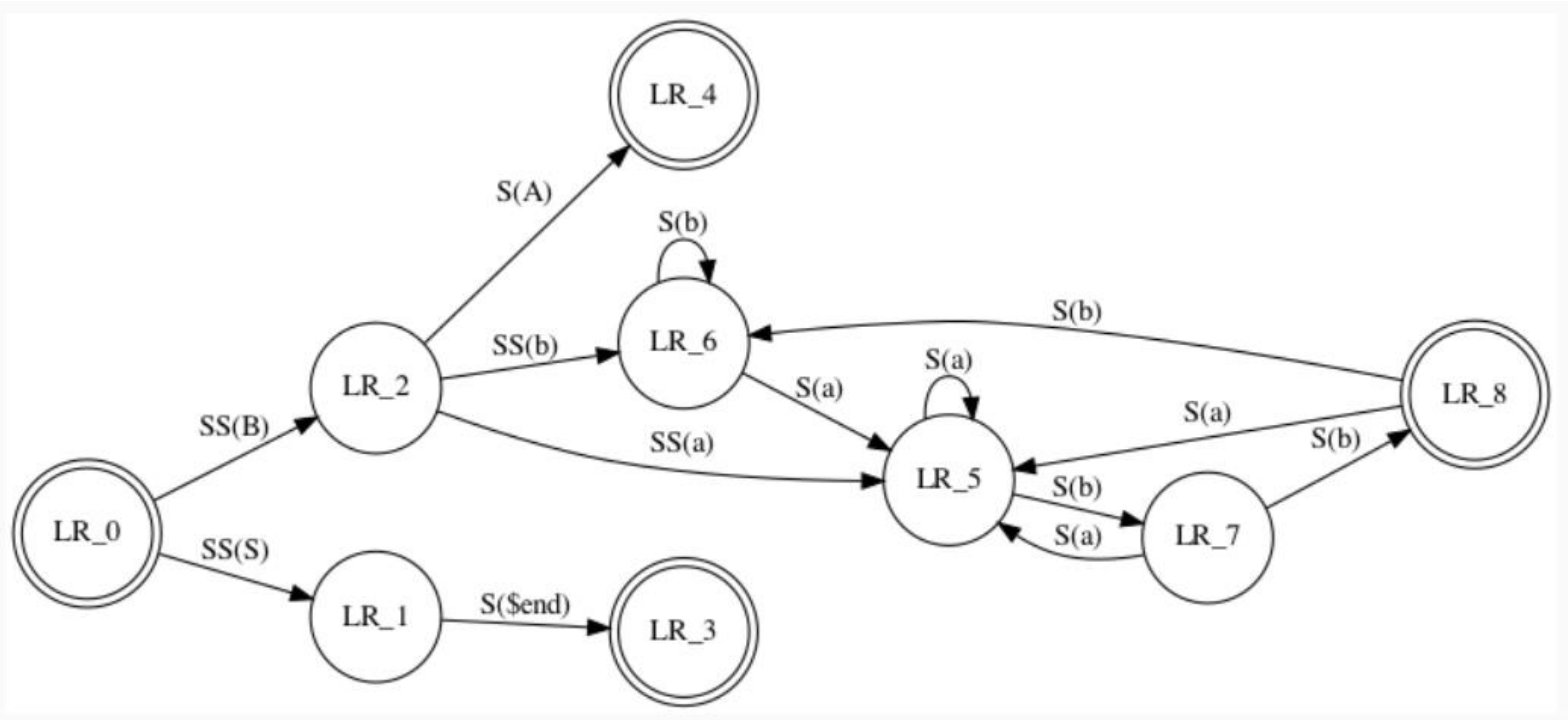


Рис. 4. Схема конечного автомата

Конечные автоматы

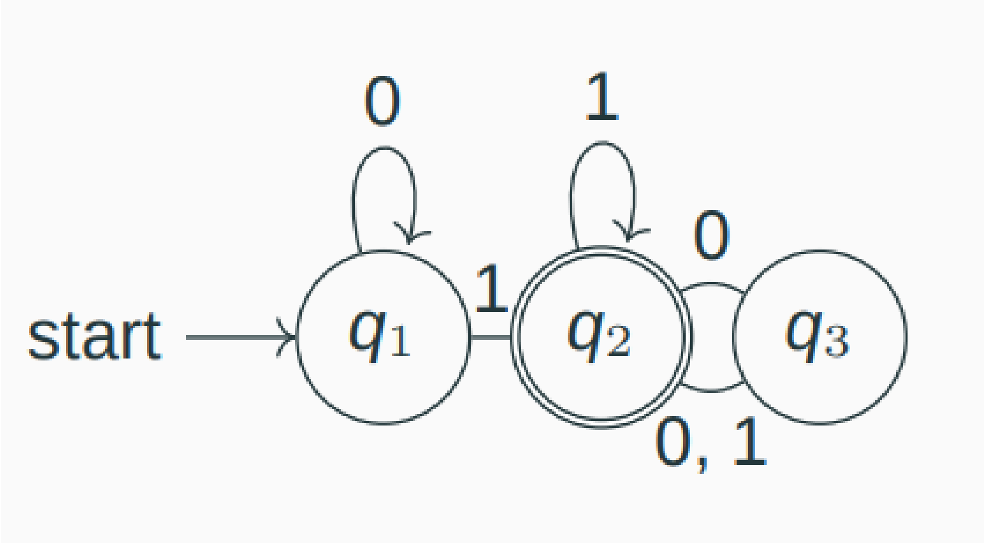


Рис. 5. Схема конечного автомата

Конечные автоматы

КА можно задавать таблицей переходов:

Текущее состояние	След. состояние при 0	След. состояние при 1
a	a	b
b	c	a
c	b	c

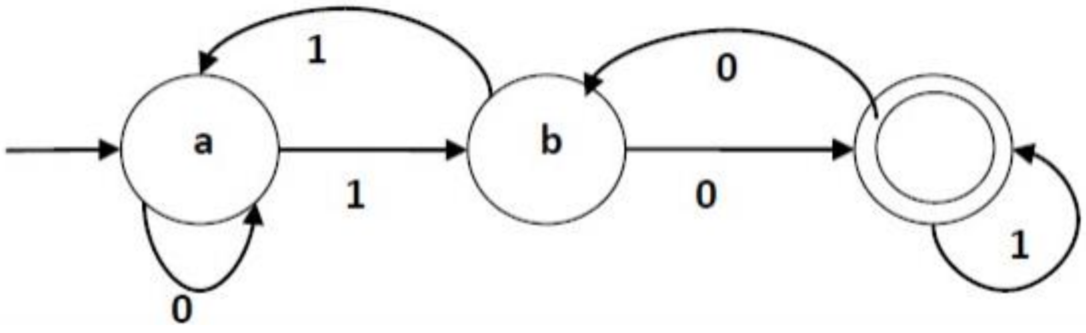


Рис. 6. Схема конечного автомата, представленного на таблице

Простейшая реализация конечного автомата. C++

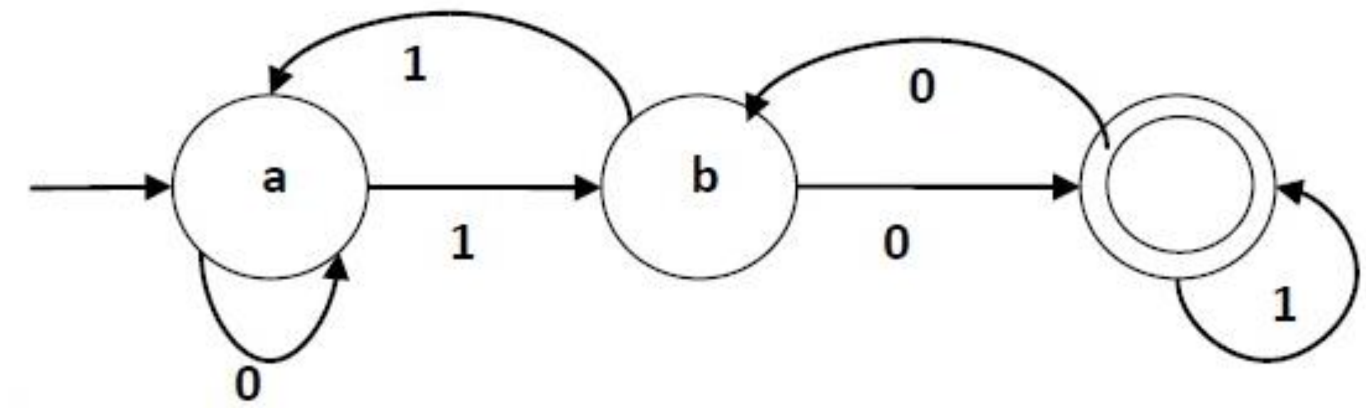
Задача

Подсчитать количество слов в строке. Слова могут разделяться любым количеством пробелов.

```
int wordsCount(char *buf) {
    int count=0;
    bool inWord=false;
    while(*buf) {
        if(*buf!=' ' && inWord==false) {
            count++;
            inWord=true;
        }
        else if(*buf==' ' && inWord==true)
            inWord=false;
    }
    return count;
}
```

Реализация конечного автомата на языке C#.

Реализуем КА, задаваемый следующей диаграммой (см. рис.6):



Реализация конечного автомата на языке C#.

Для начала определим класс State для состояния автомата:

```
class State
 {
     public string Name;
     public Dictionary<char, State> Transitions;
     public bool IsAcceptState;
 }
```

Реализация конечного автомата на языке C#.

Далее, определим 3 объекта типа State:

```
public static State a = new State()
{
  Name = "a",
  IsAcceptState = false,
  Transitions = new Dictionary<char, State>()
};
static public State b = new State()
{
  Name = "b",
  IsAcceptState = false,
  Transitions = new Dictionary<char, State>()
};
static public State c = new State()
{
  Name = "c",
  IsAcceptState = true,
  Transitions = new Dictionary<char, State>()
};
```

Реализация конечного автомата на языке C#.

Свяжем состояния функциями перехода:

```
a.Transitions['0'] = a;
a.Transitions['1'] = b;
b.Transitions['0'] = c;
b.Transitions['1'] = a;
c.Transitions['0'] = b;
c.Transitions['1'] = c;
```

Реализация конечного автомата на языке C#.

Свяжем состояния функциями перехода:

```
static public bool? Run(IEnumerable<char> s)
{
  State current = InitialState;
  foreach (var c in s) // цикл по всем символам
  {
    current = current.Transitions[c]; // меняем состояние на то, в которое у нас переход
    if (current == null)              // если его нет, возвращаем признак ошибки
        return null;
        // иначе переходим к следующему
  }
  return current.IsAcceptState;         // результат true если в конце финальное состояние
}
```

Реализация конечного автомата на языке C#.

Тестирование программы:

При подаче на вход строки **”11”** автомат завершает работу в недопустимом состоянии.

При подаче на вход строки **”1110001”** работа автомата завершается успешно

Библиотека Stateless.

Библиотека Stateless упрощает разработку конечных автоматов на языке C#.

Библиотека Stateless.

Рассмотрим пример программы (некоторые директивы using опущены):

```
using Stateless;
using Stateless.Graph;
namespace sm1c
{
    class Program
    {
        enum Trigger { TOGGLE };
        enum State { ON, OFF };
        static StateMachine<State, Trigger> sm1;

        static bool IsLightNeeded() {
            return false;
        }
        static void Main(string[] args) {
            State CurrentState = State.OFF;
            sm1=new StateMachine<State, Trigger>(() => CurrentState, s => CurrentState = s);
            sm1.Configure(State.ON).Permit(Trigger.TOGGLE, State.OFF);

            sm1.Configure(State.OFF).PermitIf(Trigger.TOGGLE, State.ON, () => IsLightNeeded(),"Toggle allowed"
                    .PermitReentryIf(Trigger.TOGGLE, () => !IsLightNeeded(),"Toggle not allowed");

            sm1.Fire(Trigger.TOGGLE);
            string graph = UmlDotGraph.Format(sm1.GetInfo());
            Console.WriteLine(graph);
        }
    }
```

Библиотека Stateless.

В качестве результата, программа выдает описание диаграммы состояний на языке dot:

```
digraph {
 ON -> OFF [label="TOGGLE"];
 OFF -> ON [label="TOGGLE [Toggle allowed]"];
 OFF -> OFF [label="TOGGLE [Toggle not allowed]"];
}
```

Библиотека Stateless.

В качестве результата, программа выдает описание диаграммы состояний на языке dot:

```
digraph {
 ON -> OFF [label="TOGGLE"];
 OFF -> ON [label="TOGGLE [Toggle allowed]"];
 OFF -> OFF [label="TOGGLE [Toggle not allowed]"];
}
```

Библиотека Stateless.

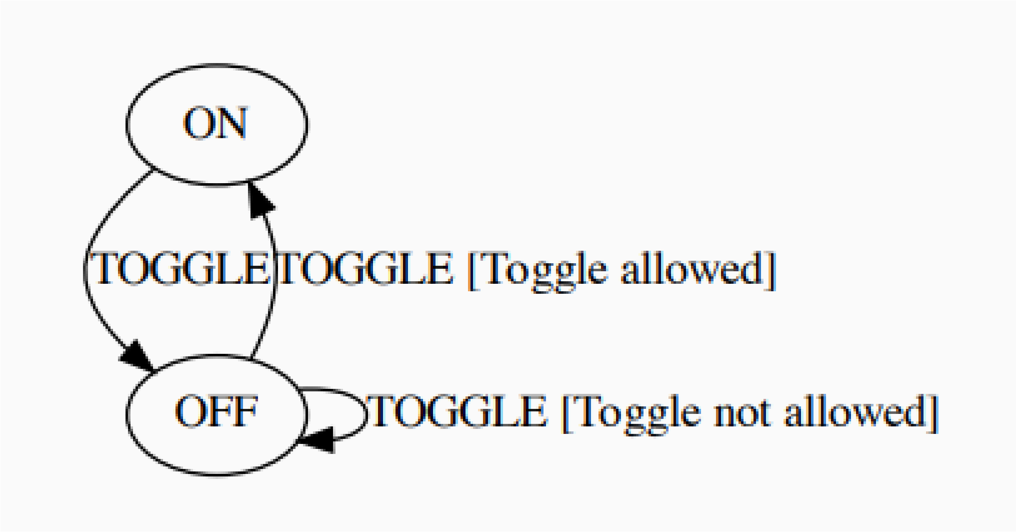


Рис. 7. Диаграмма состояний автомата из примера

Библиотека Stateless.

• **Configure** - метод, задающий состояние автомата.

• **Permit** - метод, устанавливающий связь между событием и результирующим состоянием.

• **PermitIf** - метод, похожий на Permit, но использующий доп. условие.

• **Fire** - метод, вызывающий срабатывание функции перехода.

Спасибо за внимание!!!!!