Why to study finite automata

<u>**Finite automata**</u> (aka <u>**finite-state machines**</u>) are extensively used in computer science and data networking applications.

Here are some examples of the areas where they are used:

- programs for spell checking
- pattern matching in search engines
- compiler design
- specifying network protocols
- chip design
- speech recognition
- transforming text using markup languages like XML
- ...

Preliminaries: alphabets and words

- An **alphabet** is a finite set S of symbols.

  FOR EXAMPLE: – $\{a, b, c, \ldots, z\}$

  – $\{0, 1\}$

  – $\{\Box, \Diamond, \heartsuit\}$

- A **word** or **string** (over an alphabet S) is a finite sequence of symbols from S, written without commas.

  FOR EXAMPLE: – $tortoise$, $azwzax$

  – 11111111110000000000, 000110,

  – $\heartsuit\heartsuit\Box$, $\Box\Diamond\Box\Box\Diamond\heartsuit$,

  – the **empty word**, denoted by $\boxed{\varepsilon}$, is a word over *any* alphabet (but we may assume that ε is *NOT a symbol* of any of our alphabets)

More on words

- The **length** of a word w is

$$|w| = \text{the number of symbols in } w$$

FOR EXAMPLE: $|azwza| = 5$, $|\heartsuit\heartsuit\square| = 3$, $|\varepsilon| = 0$

- The **concatenation** of words x and y (written xy):

  the word x followed by the word y

  - $w^n = \overbrace{ww\dots w}^{n}$

    FOR EXAMPLE: $(\heartsuit\square)^0 = \varepsilon$, $(01)^3 = 010101$, $a^4 = aaaa$

  - $x\varepsilon = \varepsilon x = x$, for every word x

- If $w = xy$ then x is a **prefix** of w, and y a **suffix** of w.

  FOR EXAMPLE: tor is a prefix and se is a suffix of $tortoise$

  $tortoise$ is **both** a prefix and a suffix of $tortoise$

Preliminaries: languages

A **language** (over an alphabet S) is a set of words over S.

FOR EXAMPLE:

(1) $S = \{a, b, c, \ldots, z\}$

- L_1 = all English words
- L_2 = all Latin words
- $L_3 = \{kdpekvg,\ leih,\ hkiiw,\ wowiszk\}$

(2) $S = \{0, 1\}$

- $L_4 = \{001,\ 101010,\ 111,\ 1001\}$
- $L_5 = \{0^n 1^m \mid n$ is an even, m is an odd number$\}$

Which program is more complicated?

Task 1 *Design a computer program that, for any input word over the Latin alphabet, outputs* 1 *if the length of the word is divisible by* 3*, and otherwise outputs* 0*.*

Task 2 *Design a computer program that sorts* (*in ascending order*) *and outputs the result for any input sequence* $a_1, a_2, \ldots, a_n$ *of numbers, where* n *is any natural number.*

A possible measure: the amount of **memory** needed.

- Program 1 requires *constant* memory, regardless of the input length.
- Program 2 must memorise the entire input list, and that can be of any *arbitrary* length.

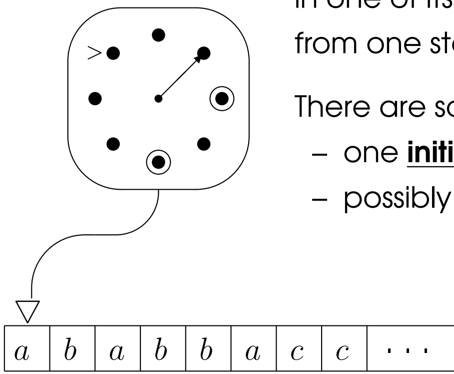
Finite automaton

A *theoretical model* for programs using a constant amount of memory regardless of the input form.

Finite control device: In any moment it can be in one of its **states**. It is hard-wired how it changes from one state to another.

There are some special states:

- one **initial state**,
- possibly more **favourable states**.

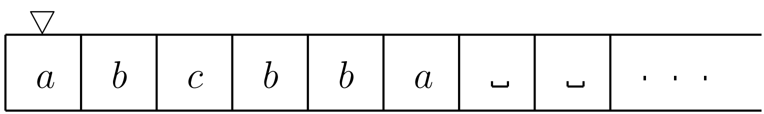


Reading head

Input tape: It is divided into cells, having a leftmost cell, but as long as we want to the right. Each cell may contain one character of the **input alphabet**.

Finite automaton: how it starts

- The finite control device is in its unique initial state.
- The tape contains a finite word of the input alphabet, the <u>**input**</u>, at its left end. The remainder of the tape contains only blank cells.
- The reading head is positioned on the leftmost cell of the input tape, containing the first character of the input word.



Finite automaton: how it works

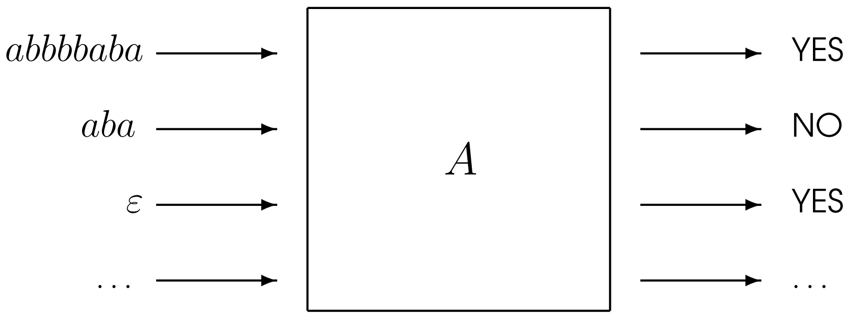
- At regular time intervals, the automaton
  - reads one character from the input tape,
  - moves the reading head one cell to the right, and
  - changes the state of its control device.
- The control device is hard-wired such that the next state depends
  on (1) the previous state, and
  on (2) the character read from the tape.
- As the input is finite, at some moment the reading head reaches the end of the input word (that is, the first blank cell).

  If at this moment the control device is in a favourable state,
  then the input word is **accepted** by the automaton.

  Otherwise, the input word is not accepted (**rejected**).

Finite automata: what they are used for

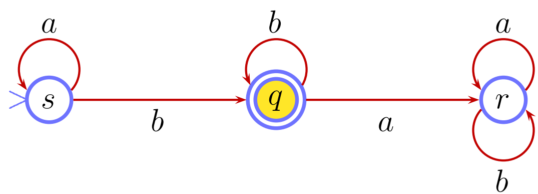
- Each finite automaton is a kind of 'recognition' or 'decision' device over all possible words of its input alphabet.
- Each automaton can be 'tried' on infinitely many input words, and gives a YES/NO answer each time.



'Visual' representation of finite automata: state transition diagrams

We can represent the hard-wired control device of a finite automaton by a directed multigraph:

- with the vertices representing the states,
- and the arrow-edges being labelled by symbols of the input alphabet.

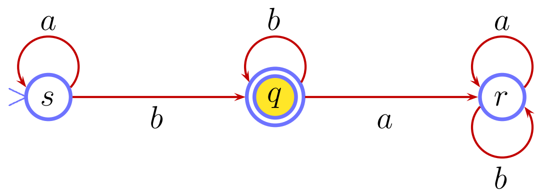


- The initial state is marked by $>$.
- The favourable states are double-circled.
- Each arrow represents a possible **transition**, hard-wired in the control device:

  Say, an arrow from state q to state r labelled by symbol a indicates that, when the head is reading a and the control device is in state q, then it should move next to state r.

Finite automata: an example

Automaton A_1 :



- Input 1: $abba$

  Computation: $(s, abba),\ (s, bba),\ (q, ba),\ (q, a),\ (r, \varepsilon)$

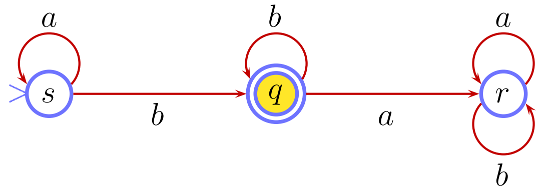
  $\rightsquigarrow$ word $abba$ is **rejected**

- Input 2: $aabb$

  Computation: $(s, aabb),\ (s, abb),\ (s, bb),\ (q, b),\ (q, \varepsilon)$

  $\rightsquigarrow$ word $aabb$ is **accepted**

More on how automaton A_1 works



- <u>Input 3:</u> $bbaaa$

  <u>Computation:</u> $(s, bbaaa),\ (q, baaa),\ (q, aaa),\ (r, aa),\ (r, a),\ (r, \varepsilon)$

  $\rightsquigarrow$ word $bbaaa$ is **rejected**

- <u>Input 4:</u> bbb

  <u>Computation:</u> $(s, bbb),\ (q, bb),\ (q, b),\ (q, \varepsilon)$

  $\rightsquigarrow$ word bbb is **accepted**

- <u>Input 5:</u> ε

  <u>Computation:</u> (s, ε)

  $\rightsquigarrow$ word ε is **rejected**

Deterministic Finite Automaton (DFA): a summary

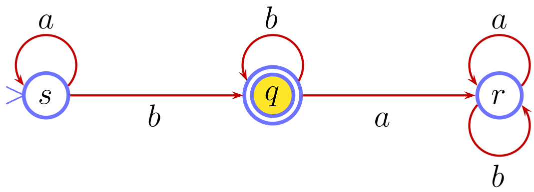
In order to describe a $\boxed{\textbf{DFA}}$ we need to describe **5** things:

- its **states**
- its **input alphabet**,
- its (unique) **initial state**,
- its **favourable** (or **accepting**) **states** (there can be none, or more than one)
- its **transition function**: for every (state, input symbol) pair we have to tell what the next state should be

A word w is **accepted** by DFA A if the computation of A on input w ends up in some *favourable state.*

Otherwise, w is **rejected** by A.

DFAs: what does 'deterministic' mean?



Observe that for each state and each symbol, there is a *unique* arrow coming out of the state labelled by the symbol.

(Here *unique* means two things: there is one, but not more than one.)

In other words, the pair

(current state, symbol read)

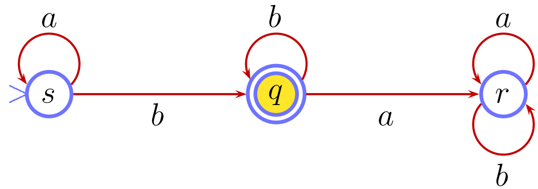
uniquely determines the next state.

This is why **D**FAs are called **deterministic**.

(Later we will also consider automata that are NOT like this.)

Another representation of DFAs: the transition table

A_1 :



states: s, q, r input alphabet: $\{a, b\}$ initial state: s favourable states: q

The **transition table** is another way of representing the transition function of A_1:

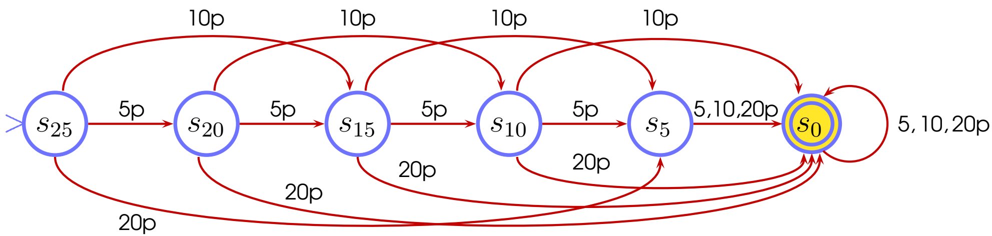
	a	b
s	s	q
q	r	q
r	r	r

Observe again: for each 'cell' in the transition table there is a *unique* state to put in. In other words, the pair

(current state, symbol read)

uniquely determines the next state. (This is why DFAs are called **deterministic**.)

Example A_2: vending machine



states: $s_{25}, s_{20}, s_{15}, s_{10}, s_5, s_0$

initial state: s_{25}

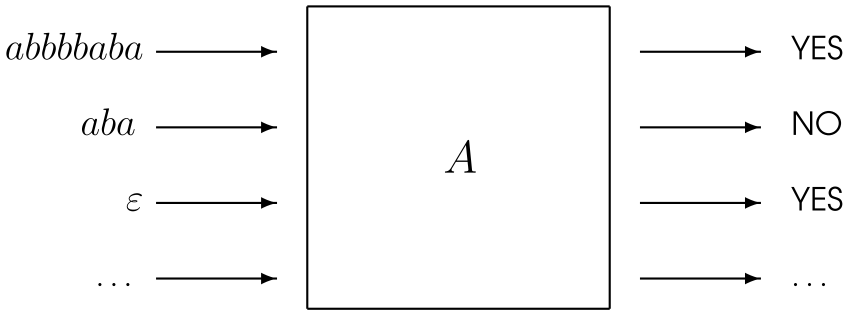
input alphabet: $\{5\text{p},\ 10\text{p},\ 20\text{p}\}$

favourable states: s_0.

Word: any sequence of 5p, 10p and 20p coins

Accepts: all the words such that the sum of the coins is $\geq$ 25p

Languages and DFAs



Any DFA may accept certain words, while may reject others.

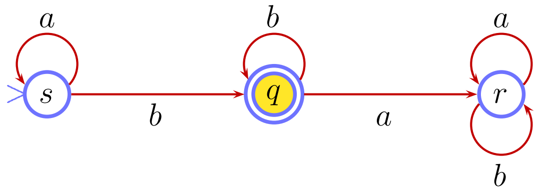
If we collect all words accepted by a DFA A, we obtain a language:

the <u>**language of a DFA**</u> A is

$L(A) =$ <u>the set of all the words over its input alphabet that A **accepts**</u>

DFA A_1 revisited

$\boxed{A_1 :}$



We already know:

- accepted: $aabb,\ bbb$
- rejected: $abba,\ bbaaa,\ \varepsilon$

$$L(A_1) = \text{all the words starting with some (possibly none) } a\text{s followed by at least one } b$$

$$= \{a^n b^m \mid n = 0, 1, 2, \ldots,\quad m = 1, 2, \ldots\}$$

How to find the language of a finite automaton

(1) Experiment with words.

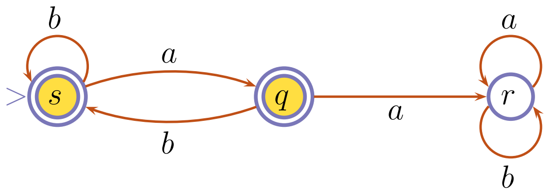
(2) Come up with a language.

(3) Test the suggested language:

- Every word that is accepted by the automaton <u>should be in</u> the suggested language.
- Every word that is rejected by the automaton <u>must not be in</u> the suggested language.

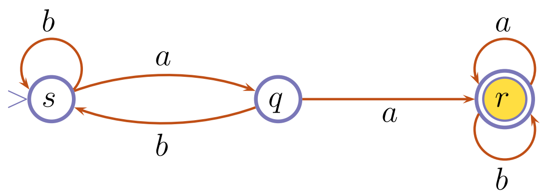
(4) Revise the suggested language if necessary, then go to step (3).

Example: DFA A_3



$L(A_3)$ $=$ all the words that do not contain two consecutive as

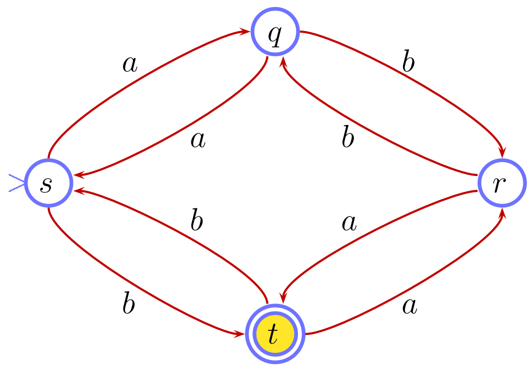
Example: DFA A_4



$L(A_4)$ = all the words that contain two consecutive a s

Observe that $L(A_4) = \{w \mid w \text{ is any word of } a\text{s and } b\text{s}\} - L(A_3)$.

Example: DFA A_5



$L(A_5) =$ all the words that contain an even number of a s and odd number of b s