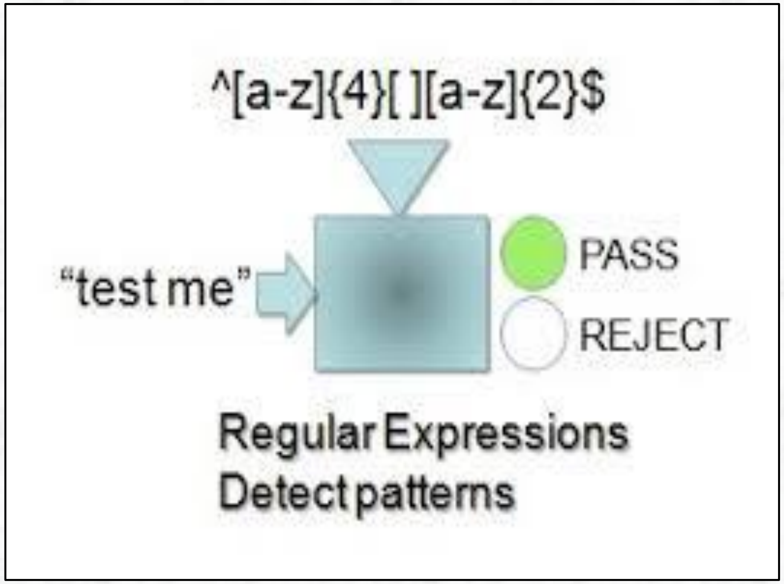
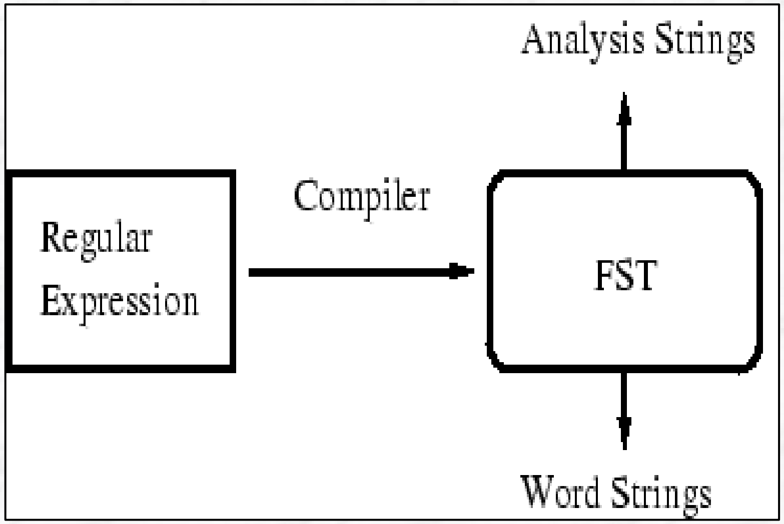
Basic Text Processing





Finite State Automata (FSA) and Regular Expression (Regex)

Instructor : Dr. Hanaa Bayomi Ali
Mail : h.mobarz @ fci-cu.edu.eg

Outlines

- Corpora
- Tokenization
- Regular expression
- Finite state automat

Corpora (plural of : Corpus**)**

- **A corpus** is a collection of text in a machine readable format.

- A corpus should be:

  - It must be a large body of text
  - It needs to be representative of language
  - Must be in machine-readable form
  - Acts as a standard reference
  - Often annotated

Corpora (plural of : Corpus) cont.

- **Annotation** is additional linguistic information

- Why Annotation ?

  1. Manual examination of corpus
  2. Reusability of annotations
  3. Multi-functionality
  4. Objective record of analysis
  5. Annotation process **is** corpus analysis

Corpora (plural of : Corpus) cont.

- **Annotation** is additional **linguistic information**

  - **Part of speech (POS)**

    Noun, Verb, Adjective, etc.

    Ex. ***present***, it may be a *noun* (= 'gift'), a *verb* (= 'give someone a present') or an *adjective* (= 'not absent')

  - **Grammatical Roles**

    Subject, Object, Time, Location, etc.

  - **Word sense (Concept)**

    Person, Institution, Animal, Color, Sentiment, etc.

  - **polarity**

    Positive, negative, neutral , etc.

- **Corpus maybe annotated or not, if annotated, annotated with one or more with the above tags/labels**

Corpora (plural of : Corpus) cont.

- **Types of Corpora**

1-Specialized corpus:

Language (Arabic)

(Modern Standard Arabic or Classical Arabic or Dialect)

Time ,Location,Domain (Medicine, Agriculture, etc.),Written or spoken ,Monolingual or Multilingual corpus, Parallel corpus

2-General corpus:

British National Corpus (BNC)

Corpus Important Tool (Tokenizer)

- **Tokenization (Splitting):** is the process of breaking a stream of text up into words, phrases, symbols, or other meaningful elements called *tokens*

- All contiguous strings of alphabetic characters are part of one token; likewise with numbers

- Tokens are separated by delimiter(s) such as:
  - Whitespace characters, such as a space, tab or line break
  - Punctuation characters

What about Arabic tokenization?

أرأيتَهم

أنلزمكموها (Shall we compel you to accept it)

Corpus Important Tool (Tokenizer)

- **Tokenization (Splitting):** is the process of breaking a stream of t s, or other meaningful

- All contig are part of one token; li

- Tokens are
  Whitespac ak
  Punctuatio

What about Arabic tokenization?

أرأيتَهم

أنلزمكموها (Shall we compel you to accept it)

Corpus Important Tool (Tokenizer) (Cont')

- Python Examples:

```
x = 'blue,red,green'
print (x.split(","))
```

Output

```
['blue', 'red', 'green']
```

```
x = 'blue red green'
print (x.split(" "))
```

Output

```
['blue', 'red', 'green']
```

Regular expression

1- Find all phone numbers in a text, e.g., occurrences such as

EX: When you call (614) 292-8833, you reach the fax machine.

2-Look up the following words in a dictionary: **(Regex Dictionary)**

EX1: adjectives ending in ly (**homely**)

Ex2: words ending in the suffix ship

Adjectives (**midship**)
Nouns (**membership**)
Verbs (**worship**)

Regular expression

3- Find multiple adjacent occurrences of the same word in a text, as in

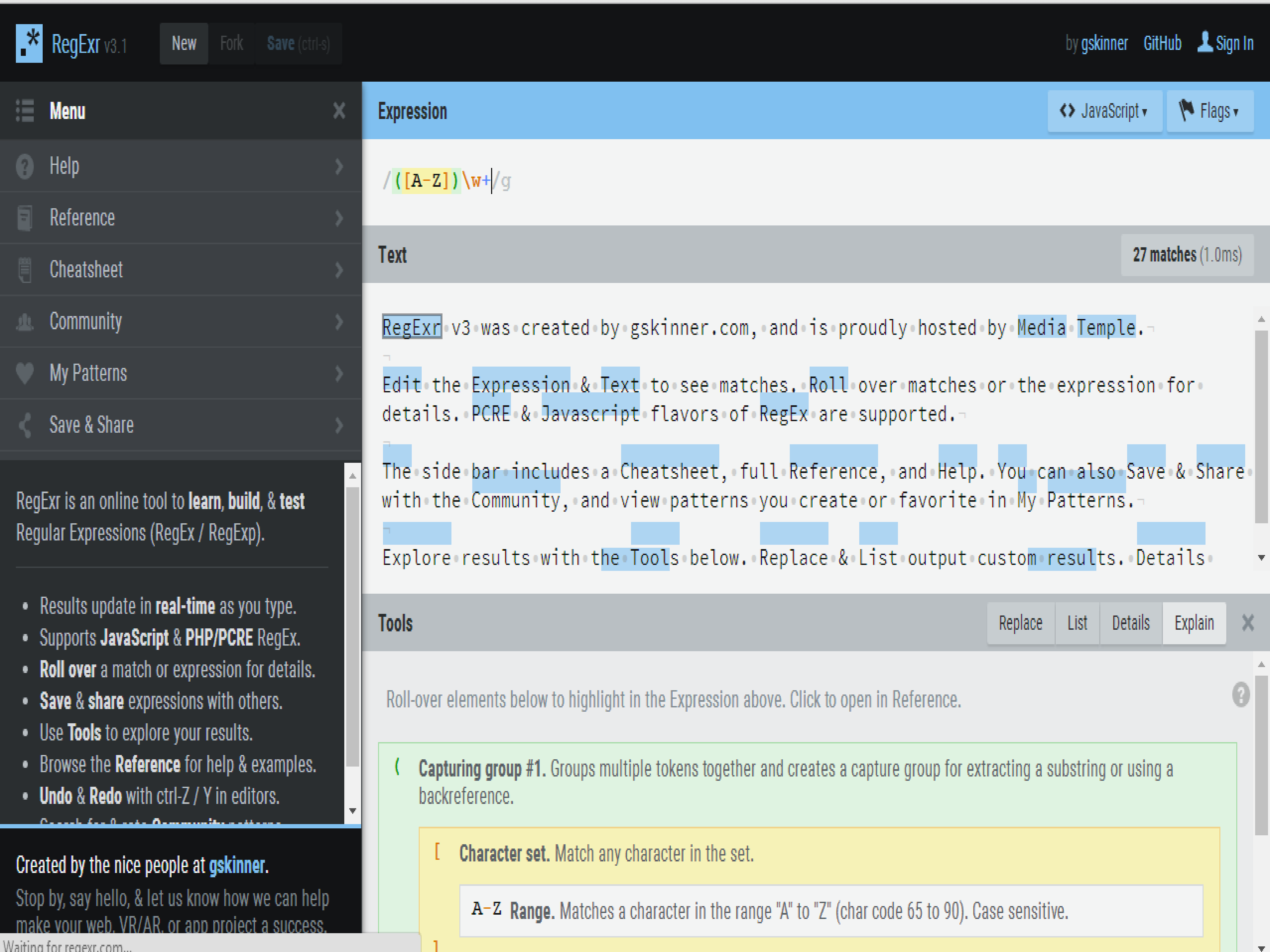
I read the **the** book.

⇒ Such tasks can be addressed using so-called finite-state machines.

⇒ How can such machines be specified?

Regular expression

- A regular expression is a description of a set of strings, i.e., a language.
- Regular expressions are very powerful to describe **patterns to search in a text**.
- Regular expressions are the key to, **powerful, flexible** and **efficient text processing.**
- A variety of unix tools (**grep, sed**), editors (emacs), and programming languages (***perl, python,JAVA***) incorporate regular expressions.
- Regular expressions themselves, with a general pattern notation almost like *a **mini programming language***, allow you to **describe and parse** text.


RegExr v3.1
New
Fork
Save (ctrl-s)
by gskinner
GitHub
Sign In
Menu
Help
Reference
Cheatsheet
Community
My Patterns
Save & Share
RegExr is an online tool to learn, build, & test Regular Expressions (RegEx / RegExp).
Results update in real-time as you type.
Supports JavaScript & PHP/PCRE RegEx.
Roll over a match or expression for details.
Save & share expressions with others.
Use Tools to explore your results.
Browse the Reference for help & examples.
Undo & Redo with ctrl-Z / Y in editors.
Created by the nice people at gskinner.
Stop by, say hello, & let us know how we can help make your web, VR/AR, or app project a success.
Waiting for regexr.com...
Expression
JavaScript
Flags
/([A-Z])\w+/g
Text
27 matches (1.0ms)
RegExr v3 was created by gskinner.com, and is proudly hosted by Media Temple.
Edit the Expression & Text to see matches. Roll over matches or the expression for details. PCRE & Javascript flavors of RegEx are supported.
The side bar includes a Cheatsheet, full Reference, and Help. You can also Save & Share with the Community, and view patterns you create or favorite in My Patterns.
Explore results with the Tools below. Replace & List output custom results. Details
Tools
Replace
List
Details
Explain
Roll-over elements below to highlight in the Expression above. Click to open in Reference.
(Capturing group #1. Groups multiple tokens together and creates a capture group for extracting a substring or using a backreference.
[Character set. Match any character in the set.
A-Z Range. Matches a character in the range "A" to "Z" (char code 65 to 90). Case sensitive.

Regular expression (Syntax)

Constant regular expressions:

Pattern	*String*
regular	"A section on ***regular*** expressions"
the	"The book of ***the*** life"

Regular expression (Syntax)

Metacharacters :

Metacharacter	*Descriptions*	*Examples*
*	Matches any number of occurrences of the previous characters – **zero or more**	**ad*e matches** ae, ade, adde, addde, etc.
?	Matches at most one occurrence of the previous characters – **zero or one**	**ad?e matches** ae and ade
+	Matches **one or more** occurrences of the previous characters	**ad+e matches** ade, adde,addde, etc.
{n}	Matches **exactly n** occurrences of the previous characters	**ad{2}e matches** adde

Regular expression (Syntax)

Metacharacters :

Metacharacters	*Descriptions*	*Examples*
{n,}	Matches n or more occurrences of the previous characters	**ad{2,}e** **matches** Adde,addde,etc.
{n,m}	Matches from n to m occurrences of the previous characters	**ad{2,4}e** **matches** Adde,addde,adddde.
.	Matches one occurrence of any characters of the alphabet or digit	**a.e** **matches** aae, aAe, abe, aBe, a1e, etc.
.*	Matches any string of characters and until it encounters a new line character	

Regular expression (Syntax)

Character Classes [ ]

- [...] matches any character contained in the list
- [abc] means one occurrence of either a, b, or c
- [^...] matches any character not contained in the list (Negated character classes)
- [^abc] means one occurrence of any character that is not an a, b, and c

Regular expression (Syntax)

Character Classes [ ] -Ranges

- [ABCDEFGHIJKLMNOPQRSTUVWXYZ] one upper-case letter ➔ **[A-Z]**
- [abcdefghijklmnopqrstuvwxyz] one lower-case letter ➔ **[a-z]**
- [0123456789] means one digit ➔ **[0-9]**

Pattern	Matches	Examples
[A-Z]	An upper case letter	**D**renched Blossoms
[a-z]	A lower case letter	M**y** beans were impatient
[0-9]	A single digit	Chapter **1**: Down the Rabbit Hole

THINK

Test yourself

1- `[Cc]omputer [Ss]cience` matches

Computer Science,
computer Science,
Computer science,
computer science

2- `[0-9]+\.[0-9]+` matches

decimal numbers

Regular expression (Syntax)

Character Classes [ ]

Pattern	Descriptions	Examples
`\d`	Any digit. Equivalent to **[0-9]**	**A\dC matches** A0C, A1C, A2C, A3C etc.
`\D`	Any non-digit. Equivalent to **[^0-9]**	
`\w`	Any word character: letter, digit,or underscore. Equivalent to **[a-zA-Z0-9_]**	**1\w2 matches** 1a2, 1A2, 1b2, 1B2, etc
`\W`	**[ˆ\w]: non-alphanumeric**	
`\s`	Any white space character: **space, tabulation, new line, form feed, etc**.	
`\S`	Any non-white space character. Equivalent to **[^\s]**	

Regular expression (Syntax)

Union and Boolean Operators (Matching any one of several sub-expressions

- **Union denoted "|":** a|b means either a or b (String is the unit)
- **a|bc matches** **a or bc**
- **(a|b) c matches** **ac or bc**
- **abc* is the set** **ab, abc, abcc, abccc, etc.**
- **(abc)* corresponds to**

  abc, abcabc, abcabcabc, etc.

Regular expression (Syntax)

Word Boundary

- ***A common problem*** is that it may that match exact word or word embedded within a larger one
- *Metasequences:* '**\<**' and '**\>**' match the position at the start and end of a word
- \<cat\> ➔ match if we can find a start-of-word position, followed immediately by cat, followed immediately by an end-of-word position.

  -In the same time **it dose not match the word 'vacation', 'concatenate'**

Regular expression (Syntax)

Line Boundary

- *Metasequences:* ‘**^**‘ and ‘**$**’ match the position at the start and end of a line
- **^cat$ ➔** matches if the line has a beginning-of-line, followed immediately by `cat`, and then followed immediately by the end of the line

THINK

Regular expression Examples

Q1) Let's say you want to search for "grey," but also want to find it if it were spelled "gray“ ➔

gr[ea]y

This means to find **g**, followed by **r**, followed by an **e** or an **a**, all followed by **y**

Q2) Matches HTML headers <H1>, <H2>, <H3>, etc.

You can use **<H[123456]>** ,**<H[1-6]>** or **<[Hh][1-6]>**

Q3) ^(From|Subject |Date): ➔ This matches:

- **start-of-line, followed by `From`, followed by ': '**
- **start-of-line, followed by `Subject`, followed by ': '**
- **start-of-line, followed by `Date` , followed by ': '**

Regular expression Examples (cont.)

- ***Remember***, the list of metacharacters and their meanings are different inside and outside of character classes
  - ^(From|Subject) → Line begins with ‘From’ or ‘Subject’
  - [^1-6] → Matches a character that's not 1 through 6

- [@.$] → dot is not metacharacter because it is within a character class

Regular expression Examples (cont.)

RegEx in Arabic

الم\wا\w\wون (كلمات على وزن المفاعلون)

Matched with:

المساعدون

المسافرون

المكاتبون

يست\w\w\wون (كلمات على وزن يستفعلون)

يستخدمون

يستعملون

يستنبطون

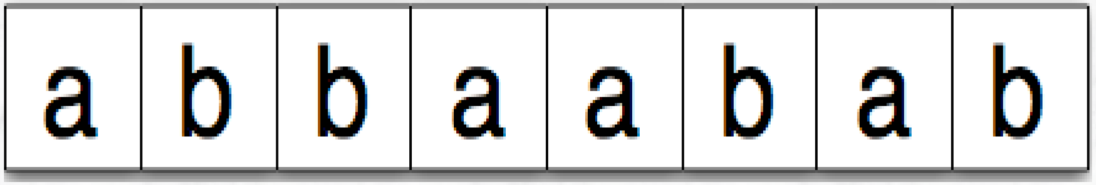
Python: Regular Expressions

```
>>> import re                          %% Import re package
>>> ex = re.compile('a\wc')   %% ‘…’: reg.expression
>>> m = ex.search('ab')       %% Does ‘ab’ contain ex?
>>> print(m)                          %% No.
None
>>> m = ex.search('abc')          %% Does ‘abc’ contain ex?
>>> print(m)                          %% Yes.
<_sre.SRE_Match object; span=(0, 3), match='abc‘>
>>> m = ex.search('a8c')     %% Does ‘a8c’ contain ex?
>>> print(m)                          %% Yes.
<_sre.SRE_Match object; span=(0, 3), match='a8c'>
```

Finite State Automata (FSA)

▪A **finite state (automata /machine) (FSA/FSM)** is a mathematical abstraction used to design algorithms. In simple terms, a state machine will read a series of inputs. When it reads an input it will switch to a different state. Each state specifies which state to switch for a given input. This sounds complicated but it is really quite simple.

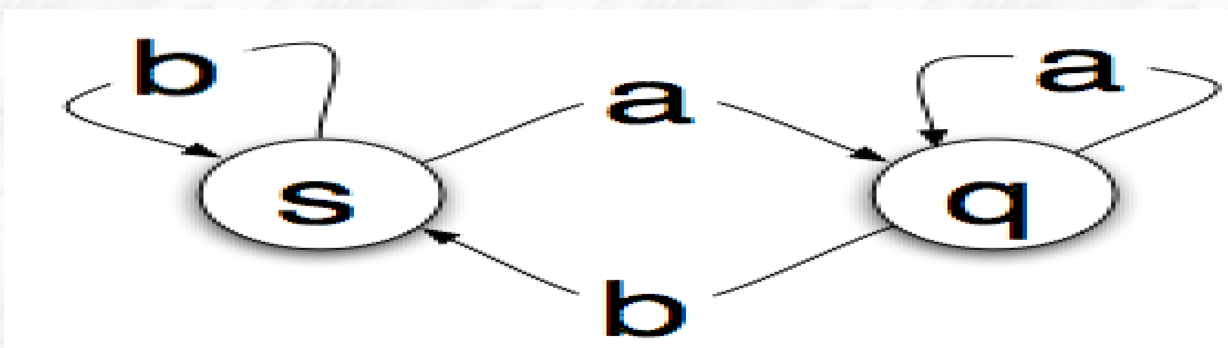
▪Imagine a device that reads a long piece of paper. For every inch of paper there is a single letter printed on it– either the letter **a** or the letter **b** .



Finite State Automata (FSA) cont.

a	b	b	a	a	b	a	b

- As the state machine reads each letter it changes state. Here is a very simple state machine.



○ The circles are “states” that the machine can be in. (S,q)

→ The arrows are the transitions.

Input (a,b)

Finite State Automata (FSA) cont.

- Regular expressions **are equivalent to** FSA and generally **easier to use**
- Regular expressions can always be implemented under the form of automata, and vice versa
- Accepting or rejecting a stream of characters
- Morphological parsing
- Searching for a word or a phrase
  - Search must extend beyond fixed strings
  - We may want to search a word or its plural form, uppercase or lowercase letters, expressions containing numbers, etc.
- Describing grammars for compilers and NLP

Finite State Automata (FSA) cont.

- ***Mathematical Definition*** of Finite-State Automata. An FSA consists of five components **($Q, \Sigma, q_0, F, \delta$)** where:

  1) ***Q*** : is a finite set of states. Information represented by its state.

  2) **Σ** : is a finite set of symbols or characters: the input alphabet. State changes in response to inputs.

  3) $\mathbf{q_0}$: is the start state, $q_0 \in Q$

  4) ***F*** is the set of final states, $F \subseteq Q$

  5) ***δ*** is a relation from states and symbols to states.

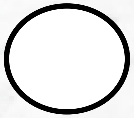
  $\delta: Q \times \Sigma \times Q$. (Transitions)

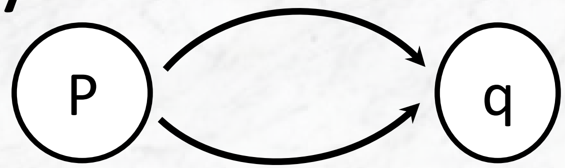
Finite State Automata (FSA) cont.

- Σ is a finite set of symbols or characters
- Σ^* is a set of all strings over Σ
- $\Sigma = \{0,1\}$
- $\Sigma^* = \{1,1111,101,001101010111,\ldots..\}$
- The ***<u>language</u>*** accepted by FSA is a set of all strings it accepts

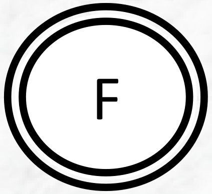
Finite State Automata (FSA) cont.

Graph Representation of FSA’s

- Nodes = states 
- Arcs represent transition function
- Arc from state *p* to state *q* labeled by all those input symbols that have transitions from *p* to *q*



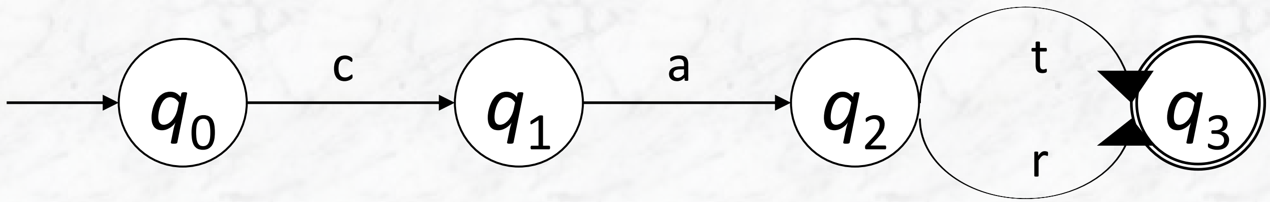
- Arrow labeled “Start” to the start state
- Final states indicated by double circles



Example (1) of FSA cont.

EX1) Design FSA for a language that accept the set {car, cat}

Regex : ca[rt]



Σ = {c,a,r,t}

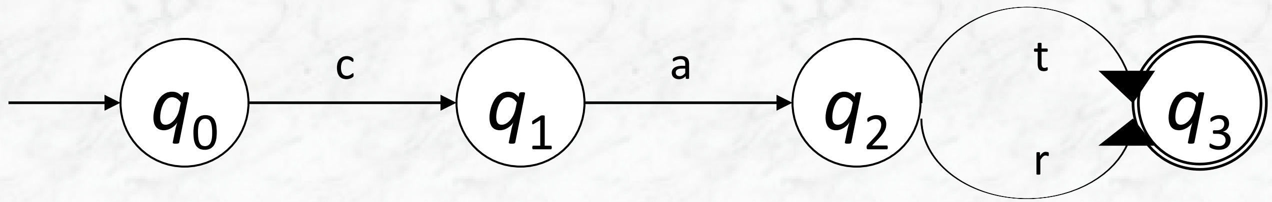
Q ={q0 , q1 , q2 , q3 }

q0= q0

F = {q3}

δ ={(q0 ,c, q1),(q1 ,a, q2),(q2 ,t, q3),(q2 ,r, q3)}

Example (1) of FSA cont.

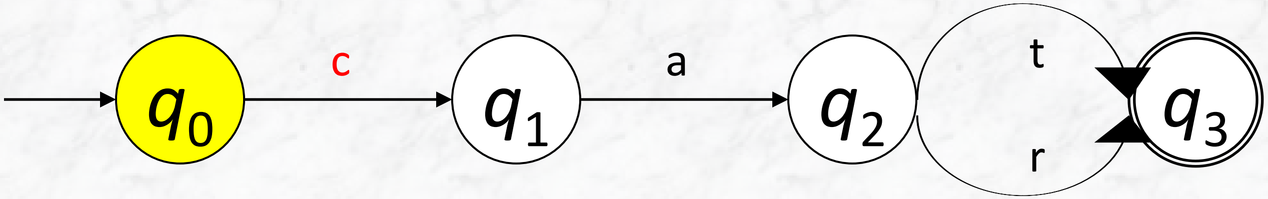


A state-transition table where ∅ denotes impossible transitions

State\Input	c	a	r	t
q_0	q_1	∅	∅	∅
q_1	∅	q_2	∅	∅
q_2	∅	∅	q_3	q_3
q_3	∅	∅	∅	∅

$\delta(q_0, c) = q_1$

Example (1) of FSA cont.

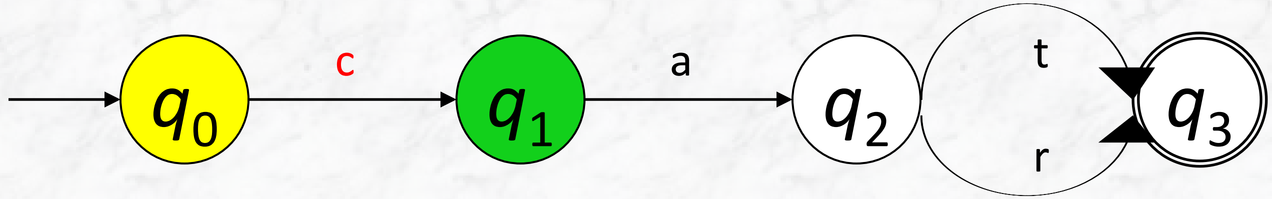


A state-transition table where Ø denotes impossible transitions

State\Input	c	a	r	t
q_0	q_1	Ø	Ø	Ø
q_1	Ø	q_2	Ø	Ø
q_2	Ø	Ø	q_3	q_3
q_3	Ø	Ø	Ø	Ø

$\delta(q_0, c) = q_1$

Example (1) of FSA cont.

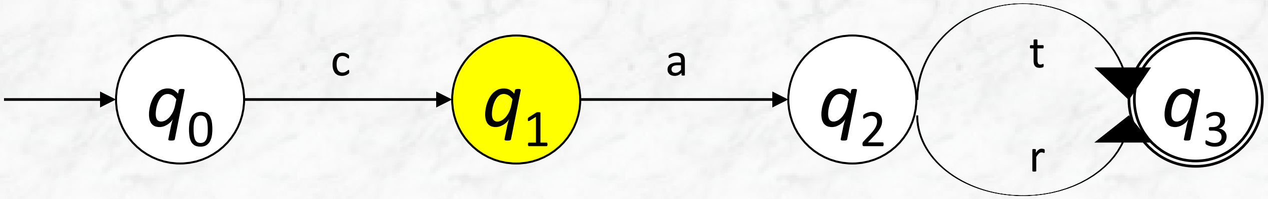


A state-transition table where Ø denotes impossible transitions

State\Input	c	a	r	t
q_0	q_1	Ø	Ø	Ø
q_1	Ø	q_2	Ø	Ø
q_2	Ø	Ø	q_3	q_3
q_3	Ø	Ø	Ø	Ø

$\delta(q_0, c) = q_1$

Example (1) of FSA cont.

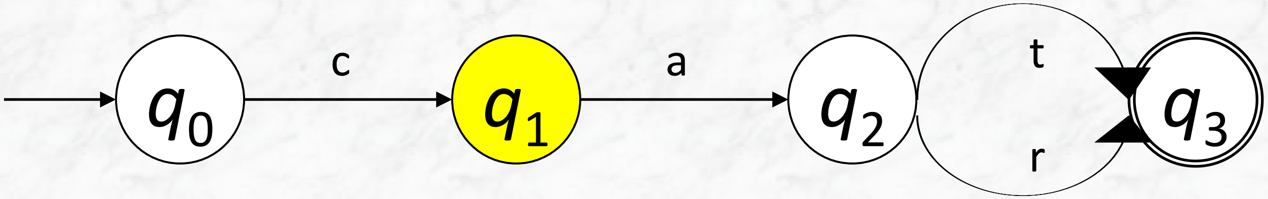


A state-transition table where Ø denotes impossible transitions

State\Input	c	a	r	t
q_0	q_1	Ø	Ø	Ø
q_1	Ø	q_2	Ø	Ø
q_2	Ø	Ø	q_3	q_3
q_3	Ø	Ø	Ø	Ø

$\delta(q_1, r) = \emptyset$

Example (1) of FSA cont.

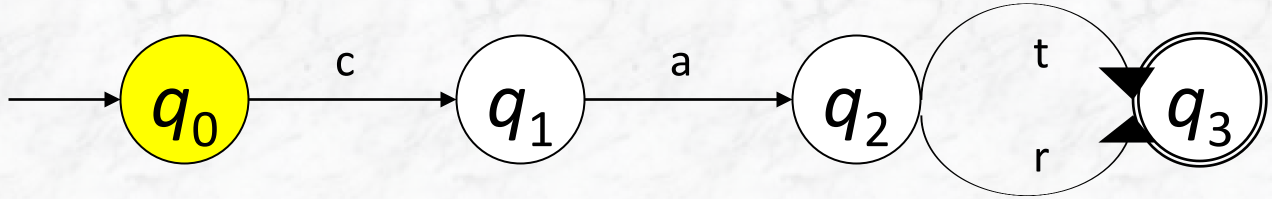


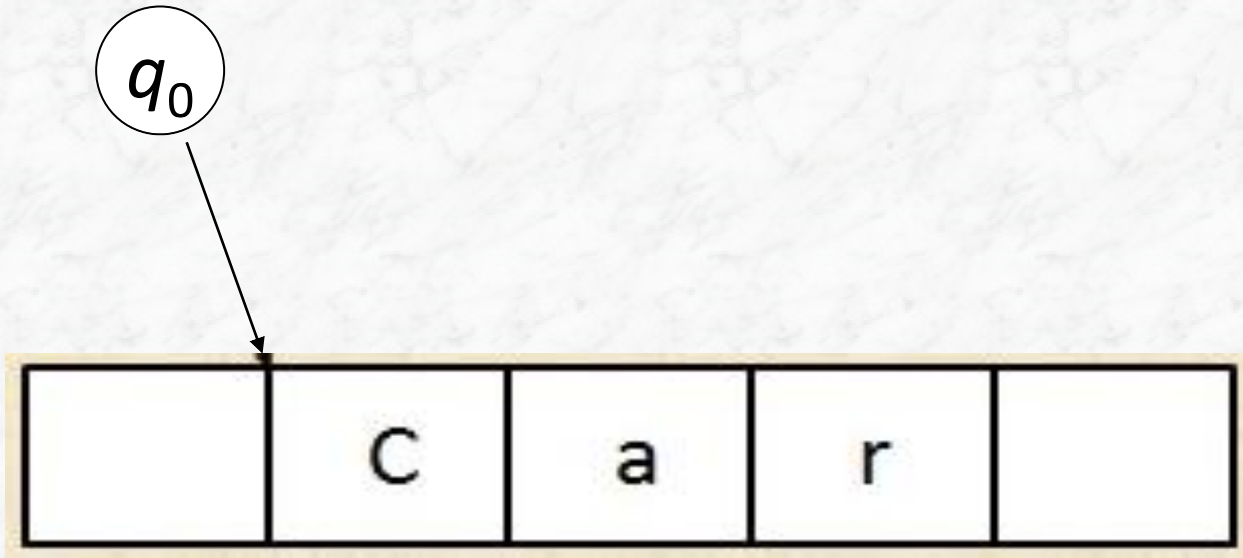
A state-transition table where **Ø** denotes impossible transitions

State\Input	c	a	r	t
q_0	q_1	Ø	Ø	Ø
q_1	Ø	q_2	Ø	Ø
q_2	Ø	Ø	q_3	q_3
q_3	Ø	Ø	Ø	Ø

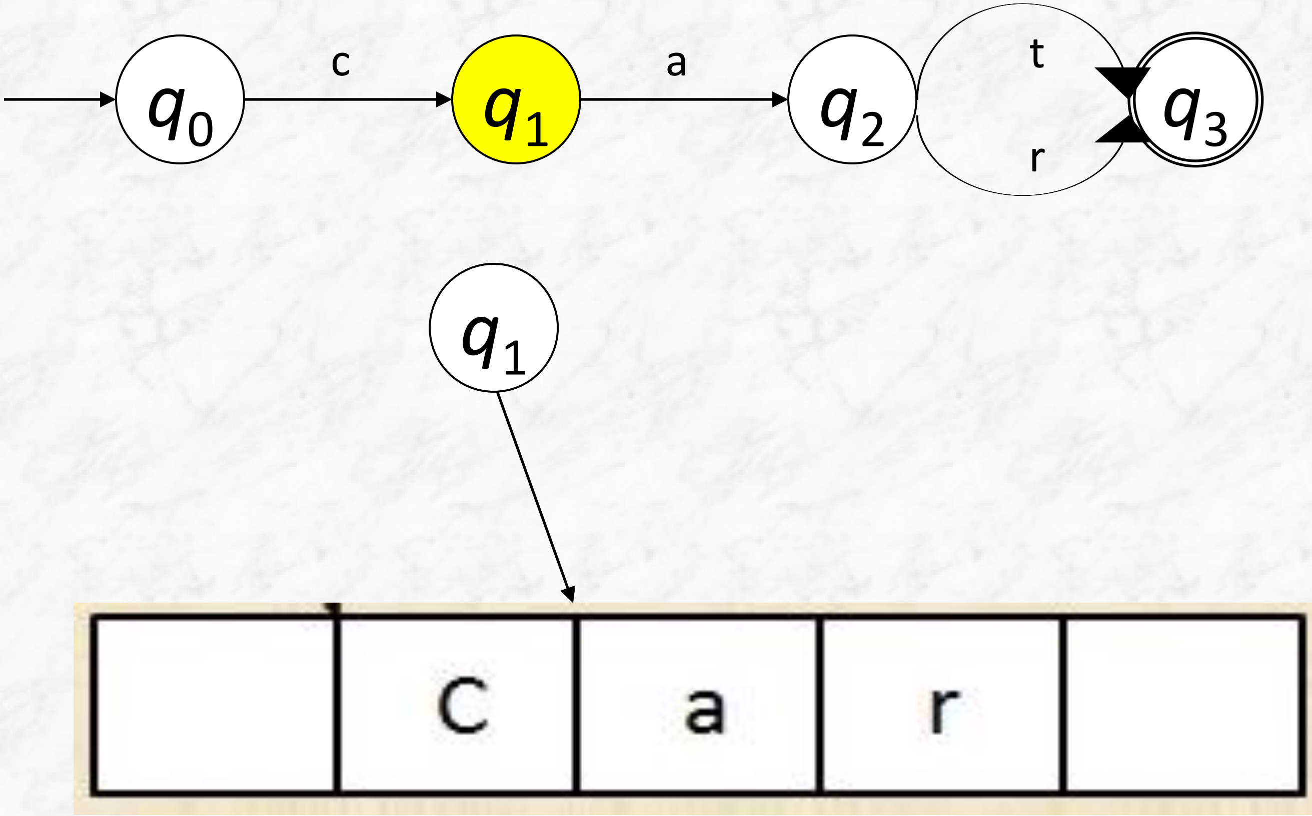
$\delta(q_1, r) = \emptyset$

Example (1) of FSA [test] cont.

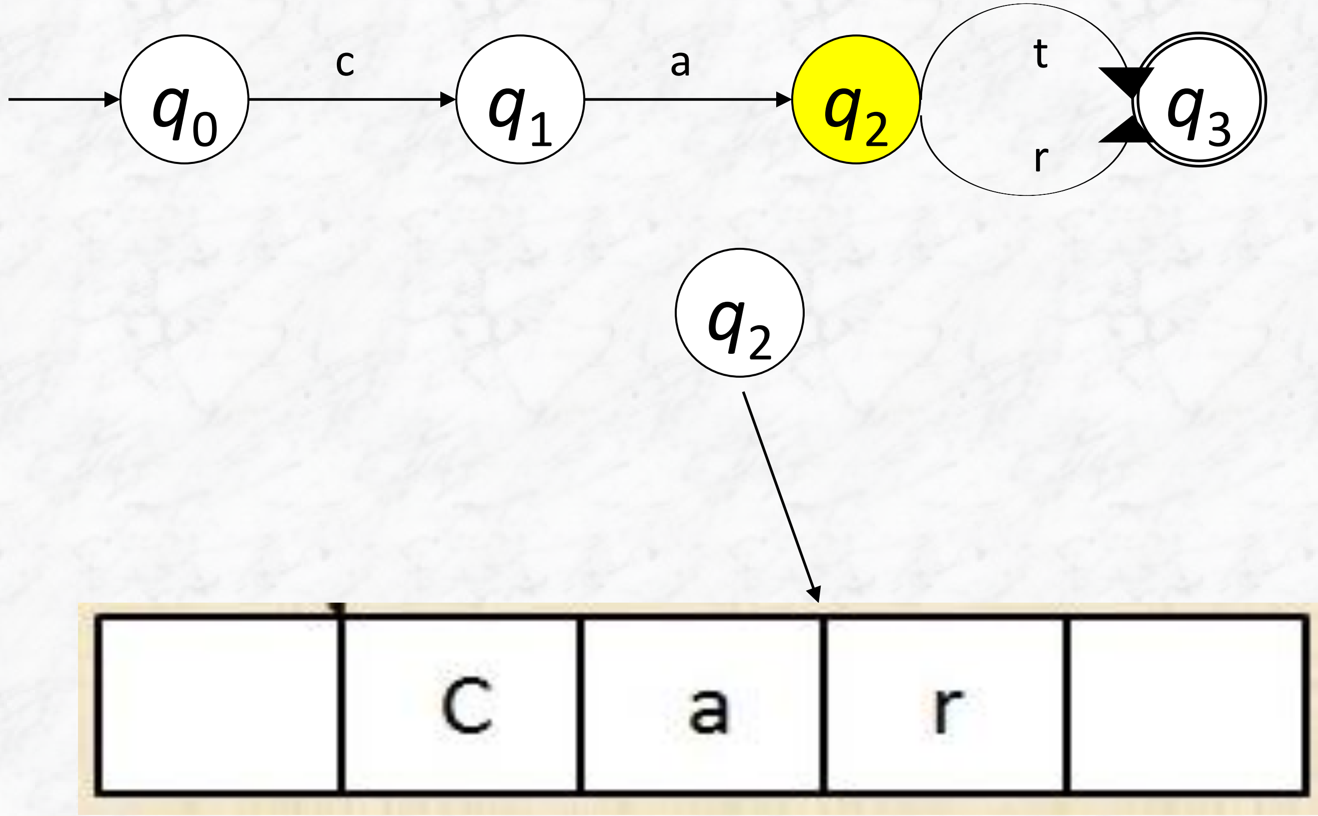




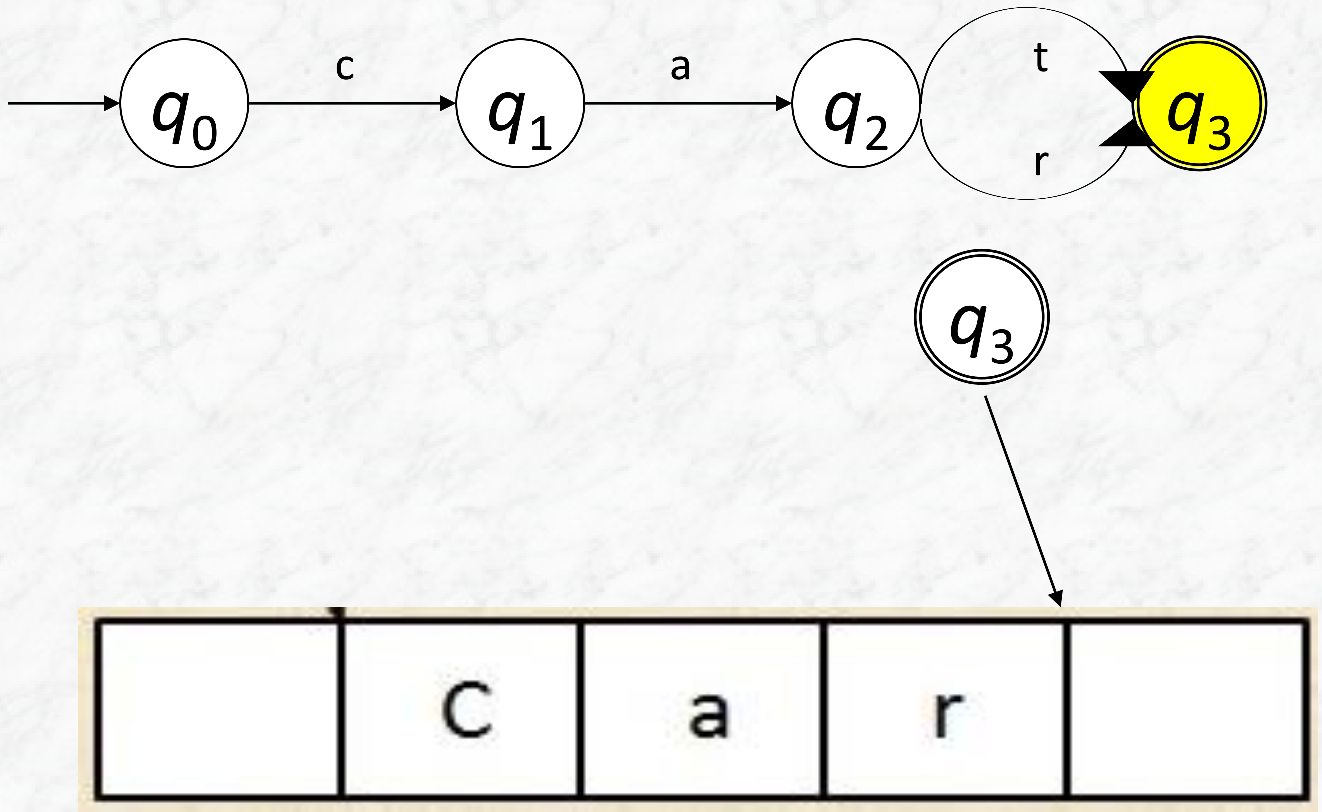
Example (1) of FSA [test] cont.



Example (1) of FSA [test] cont.



Example (1) of FSA [test] cont.



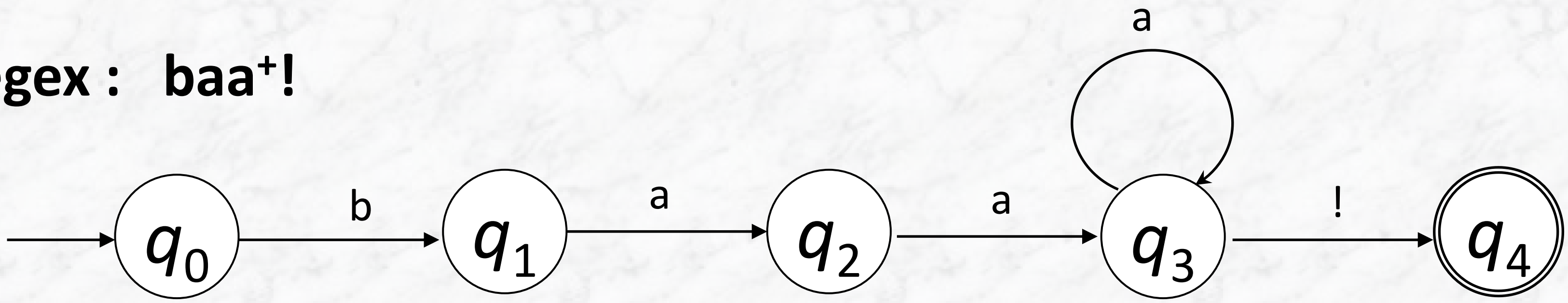
Accepted

Example (2) of FSA.

EX2) Design FSA for a ‘sheep language’ that can accept any of the following strings (infinite set)

baa!,baaa!,baaaa!,baaaaa!,............

Regex : baa$^+$!



Σ = {b,a,!}

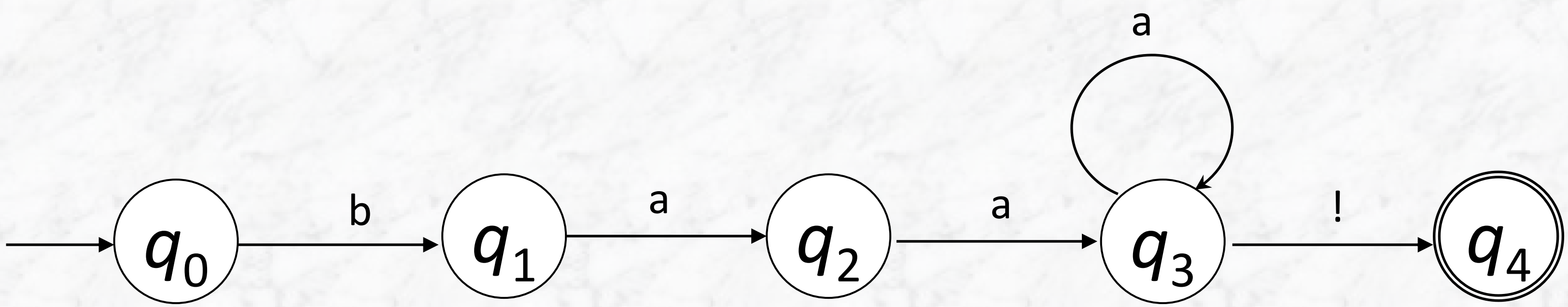
Q ={q0 , q1 , q2 , q3, q4}

q0= q0

F = {q4}

δ ={(q0 ,b, q1),(q1 ,a, q2),(q2 ,a, q3),(q3 ,a, q3) ,(q3 ,!, q4)}

Example (2) of FSA. (cont)



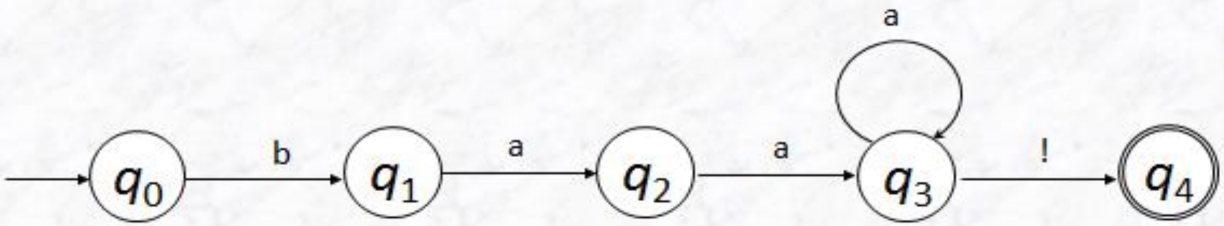
State\Input	b	a	!
q_0	q_1	∅	∅
q_1	∅	q_2	∅
q_2	∅	q_3	∅
q_3	∅	q_3	q_4
q_4	∅	∅	∅

Nondeterministic Finite State Automata(NFSA)

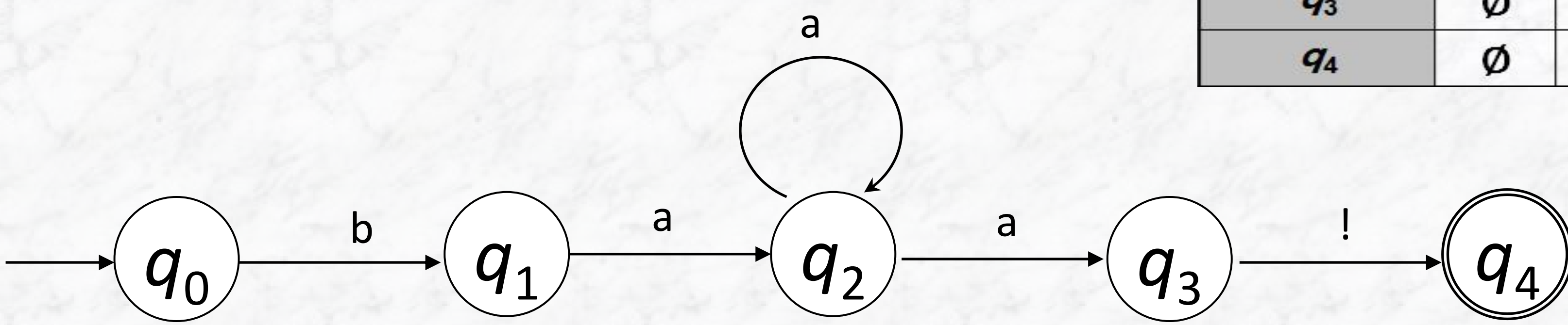
- In the FSAs that we have seen so far, there is exactly one action to be taken on each input symbol.

- In a nondeterministic FSA, a set of choices exist at each step.
  - zero, one or several possible transitions

Example of NFSA (**sheep language: baa+!**)

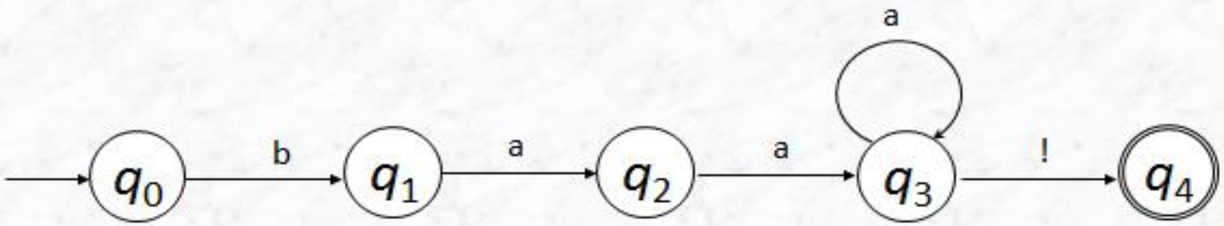


State\Input	b	a	!
q_0	q_1	∅	∅
q_1	∅	q_2	∅
q_2	∅	q_3	∅
q_3	∅	q_3	q_4
q_4	∅	∅	∅

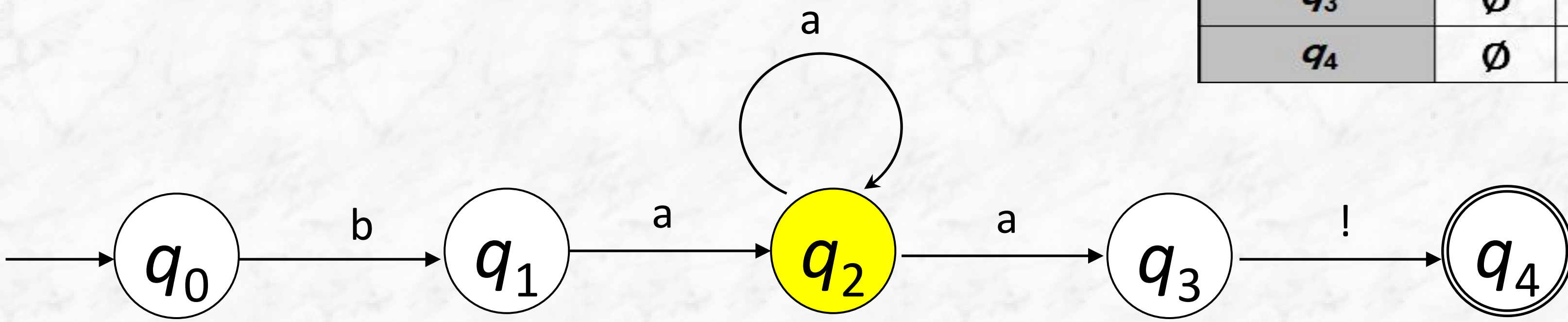


State\Input	b	a	!
q_0	q_1	∅	∅
q_1	∅	q_2	∅
q_2	∅	q_3, q_2	∅
q_3	∅	∅	q_4
q_4	∅	∅	∅

Example of NFSA (sheep language: baa+!)



State\Input	b	a	!
q_0	q_1	∅	∅
q_1	∅	q_2	∅
q_2	∅	q_3	∅
q_3	∅	q_3	q_4
q_4	∅	∅	∅



State\Input	b	a	!
q_0	q_1	∅	∅
q_1	∅	q_2	∅
q_2	∅	q_3, q_2	∅
q_3	∅	∅	q_4
q_4	∅	∅	∅