

CSE528

Natural Language Processing

Venue:ADB-405

Topic: Regular Expressions & Automata

Prof. Tulasi Prasad Sariki,

SCSE, VIT Chennai Campus

www.learnersdesk.weebly.com



Contents

- ❖ NLP Example: Chat with Alice
- ❖ Regular Expressions
- ❖ Regular Expression Patterns
- ❖ Operator precedence
- ❖ Applications
- ❖ Regular Expressions in MS-Word
- ❖ Finite Automata
- ❖ FSA / FST
- ❖ Applications of FSA & FST

NLP Example: Chat with Alice

A.L.I.C.E. (Artificial Linguistic Internet Computer Entity) is an award-winning free natural language artificial intelligence chat robot. The software used to create A.L.I.C.E. is available as free ("open source") Alicebot and AIML software.

<http://www.alicebot.org/about.html>

Regular Expressions

In computer science, RE is a language used for specifying text search string.

A regular expression is a formula in a special language that is used for specifying a simple class of string.

Formally, a regular expression is an algebraic notation for characterizing a set of strings.

RE search requires

- a *pattern* that we want to search for, and
- a *corpus* of texts to search through.

Regular Expressions

A RE search function will search through the corpus returning all texts that contain the pattern.

- In a Web search engine, they might be the entire documents or Web pages.
- In a word-processor, they might be individual words, or lines of a document.
- E.g., the UNIX `grep` command

Regular expressions are case sensitive.

We will use Perl based syntax for representation.

Regular Expressions

Disjunctions **[abc]**

Ranges **[A-Z]**

Negations **[^Ss]**

Optional characters **?** and *****

Wild cards **.**

Anchors **^** and **\$**, also **\b** and **\B**

Disjunction, grouping, and precedence **|**

Regular Expression Patterns

regular expression	example pattern matched
/woodchucks/	“interesting links to <u>woodchucks</u> and lemurs”
/a/	“M <u>a</u> ry Ann stopped by Mona’s”
/Claire says,/	Dagmar, my gift please,” <u>Claire says,</u> ”
/song/	“all our pretty <u>songs</u> ”
/!/	“You’ve left the burglar behind again <u>!</u> ” said Nori

Regular Expression Patterns

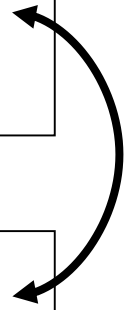
The use of the brackets `[]` to specify a disjunction of characters.

Regular Expression	Match
<code>/[wW]oodchuck/</code>	Woodchuck or woodchuck
<code>/[abc]/</code>	"a", "b", or "c"
<code>/[0123456789]/</code>	Any digit

Regular Expression Patterns

The use of the brackets [] plus the dash - to specify a range.

Regular expression	match	sample pattern
/[A-Z]/	any uppercase letter	this is <u>L</u> inguistics 5981
/[0-9]/	any single digit	this is Linguistics <u>5</u> 981
/[1 2 3 4 5 6 7 8 9 0]/	any single digit	this is Linguistics <u>5</u> 981



Regular Expression Patterns

To search for negation, i.e. a character that I do NOT want to find we use the caret: [^]

Regular expression	match	sample pattern
/[^A-Z]/	not an uppercase letter	<u>t</u> his is Linguistics 5981
/[^L I]/	neither L nor I	<u>t</u> his is Linguistics 5981
/[^.]/	not a period	<u>t</u> his is Linguistics 598

Special characters:

*	an asterisk	"L_I*N*G*U*I*S*T*I*C*S"
\.	a period	"Dr.Doolittle"
\?	a question mark	"Is this Linguistics 5981 <u>?</u> "
\n	a newline	
\t	a tab	

Regular Expression Patterns

To search for optional characters we use the question mark: [?]

Regular expression	match	sample pattern
/colou?r/	colour or color	beautiful <u>colour</u>

To search for any number of a certain character we use the Kleene star: [*]

Regular expression	match
/a*/	any string of zero or more "a"s
/aa*/	at least one a but also any number of "a"s

Regular Expression Patterns

To look for at least one character of a type we use the Kleene “+”:

Regular expression	match
/[0-9]+/	a sequence of digits

Any combination is possible

Regular expression	match
/[ab]*/	zero or more “a”s or “b”s
/[0-9] [0-9]*/	any integer (= a string of digits)

Regular Expression Patterns

The “.” is a very special character -> so-called wildcard

Regular expression	match	sample pattern
/b.l/	any character between b and l	ball, bell, bull, bill

The /. / symbol is called a wildcard : it matches any single character. For example, the regular expression /s.ng/ matches the following English words:

sang, sing, song, sung.

Note that /. / will match and not only alphabetic characters, but also numeric and whitespace characters. Consequently, /s.ng/ will also match non-words such as s3ng.

The pattern /...berry/ finds words like cranberry.

Regular Expression Patterns

Anchors (start of line: “^”, end of line: “\$”)

Regular expression	match	sample pattern
<code>/^Linguistics/</code>	“Linguistics” at the beginning of a line	<u>Linguistics</u> is fun.
<code>/linguistics\.\$/</code>	“linguistics” at the end of a line	We like <u>linguistics</u> .

Anchors (word boundary: “\b”, non-boundary: “\B”)

Regular expression	match	sample pattern
<code>\bthe\b/</code>	“the” alone	This is <u>the</u> place.
<code>\Bthe\B/</code>	“the” included	This is my mo <u>the</u> r.

Regular Expression Patterns

More on alternative characters: the pipe symbol: “|” (disjunction)

Regular expression	match	sample pattern
/colou?r/	colour or color	beautiful <u>colour</u>
/progra(m mme)/	program or programme	linguistics <u>program</u>

Predefined Character class

Character class	Description
<code>\d</code>	A digit. Equivalent to <code>[0-9]</code> .
<code>\D</code>	A non-digit. Equivalent to <code>[^0-9]</code> .
<code>\s</code>	A whitespace character. Equivalent to <code>[\t\n\x0B\f\r]</code> .
<code>\S</code>	A nonwhitespace character. Equivalent to <code>[^\s]</code> .
<code>\w</code>	A word character. Equivalent to <code>[a-zA-Z_0-9]</code> .
<code>\W</code>	A non-word character. Equivalent to <code>[^\w]</code> .

Boundary matchers

Boundary Matcher	Description
<code>^</code>	The beginning of a line
<code>\$</code>	The end of a line
<code>\b</code>	A word boundary
<code>\B</code>	A nonword boundary
<code>\A</code>	The beginning of the text
<code>\G</code>	The end of the previous match
<code>\Z</code>	The end of the text (but for the final line terminator, if any)
<code>\z</code>	The end of the text

Quantifiers

Character	Description
$\{n\}$	n is a nonnegative integer. Matches exactly n times. For example, $'o\{2\}'$ does not match the $'o'$ in "Bob," but matches the two o 's in "food".
$\{n,\}$	n is a nonnegative integer. Matches at least n times. For example, $'o\{2,\}'$ does not match the $'o'$ in "Bob" and matches all the o 's in "fooooood". $'o\{1,\}'$ is equivalent to $'o^+'$. $'o\{0,\}'$ is equivalent to $'o^*'$.
$\{n,m\}$	M and n are nonnegative integers, where $n \leq m$. Matches at least n and at most m times. For example, $'o\{1,3\}'$ matches the first three o 's in "fooooood". $'o\{0,1\}'$ is equivalent to $'o'$

Operator precedence

A regular expression is evaluated from left to right and follows an order of precedence, much like an arithmetic expression.

The following table illustrates, from highest to lowest, the order of precedence of the various regular expression operators:

Operator(s)	Description
\	Escape
(), (?:), (?=), []	Parentheses and Brackets
*, +, ?, {n}, {n,}, {n,m}	Quantifiers
^, \$, \anymetacharacter, anycharacter	Anchors and Sequences
	Alternation

Operator precedence

Characters have higher precedence than the alternation operator, which allows 'm|food' to match "m" or "food". To match "mood" or "food", use parentheses to create a subexpression, which results in '(m|f)ood'.

Applications

Regular Expressions for the Java Programming Language

- `java.util.regex` for enabling the use of regular expressions

Applications

- Simple word replacement
- Email validation
- Removal of control characters from a file
- File searching

Example

write a Perl regular expression to match the English article “the”:

`/the/`

missed ‘The’

`/[tT]he/`

included ‘the’ in ‘others’

`/\b[tT]he\b/`

Missed ‘the25’ ‘the_’

`/[^a-zA-Z][tT]he[^a-zA-Z]/` Missed ‘The’ at the beginning of a line

`/(^|[^a-zA-Z])[tT]he[^a-zA-Z]/`

Example

Write a regular expression that will match “any PC with more than 500MHz and 32 Gb of disk space for less than \$1000”:

Price

- `/$[0-9]+/` # whole dollars
- `/$[0-9]+\.[0-9][0-9]/` # dollars and cents
- `/$[0-9]+(\.[0-9][0-9])?/` #cents optional
- `/\b$[0-9]+(\.[0-9][0-9])?\b/` #word boundaries

Example

Specifications for processor speed

- `/\b[0-9]+ *(MHz | [Mm]egahertz | Ghz | [Gg]igahertz)\b/`

Memory size

- `/\b[0-9]+ *(Mb | [Mm]egabytes?)\b/`
- `/\b[0-9](\.[0-9]+) *(Gb | [Gg]igabytes?)\b/`

Vendors

- `/\b(Win95 | WIN98 | WINNT | WINXP *(NT | 95 | 98 | 2000 | XP?)?)\b/`
- `/\b(Mac | Macintosh | Apple)\b/`

Example

Expression	Matches
<code>/^\s*\$/</code>	Match a blank line.
<code>/\d{2}-\d{5}/</code>	Validate an ID number consisting of 2 digits, a hyphen, and an additional 5 digits.
<code>/<\s*(\S+)(\s[^\>]*)?>[\s\S]*<\s*\1\s*>/</code>	Match an HTML tag.

Regular Expressions in MS-Word

? and *

- ❑ The two most basic wildcard characters are ? and *.
- ❑ ? is used to represent a single character and * represents any number of characters.
- ❑ **s?t** will find sat, set, sit, sat and any other combination of 3 characters beginning with “s” and ending with “t”. Ex: **inset**.
- ❑ **s*t** will find all the above, but will also find “secret”, “serpent”, “sailing boat” and “sign over document”, etc.

@

- ❑ @ is used to find one or more occurrences of the previous character.
- ❑ For example, lo@t will find lot or loot, ful@ will find ful or full etc.

Regular Expressions in MS-Word

< >

- ❑ <s*t> would find “secret” and “serpent” and “sailing boat”, but not “sailing boats” or “sign over documents”. It will also find “set” in “tea-set” , but not “set” in “toolset”.
- ❑ The <> tags can be used in pairs, as above; or individually.
- ❑ ful@> will find “full” and the appropriate part of “wilful”, but will not find “wilfully”.

Regular Expressions in MS-Word

[]

- ❑ Square brackets are always used in pairs and are used to identify specific characters or ranges of characters.
- ❑ [abc] will find any of the letters a, b, or c.
- ❑ [A-Z] will find any upper case letter.
- ❑ [13579] will find any odd digit.

\

- ❑ If you wish to search for a character that has a special meaning in wildcard searches – the obvious example being “?” – then you can do so by putting a backslash in front of it.
- ❑ [\?] will not find “\” followed by any character; but will find “?”

Regular Expressions in MS-Word

[!]

- ❑ [!] is very similar to [] except in this case it finds any character not listed in the box so [!o] would find every character except “o”.
- ❑ You can use ranges of characters in exactly the same way as with [], thus [!A-Z] will find everything except upper case letters.

Regular Expressions in MS-Word

{ }

- ❑ Curly brackets are used for counting occurrences of the previous character or expression.
- ❑ {n} This finds exactly the number “n” of occurrences of the previous character (so for example, a{2} will find “aa”).
- ❑ {n,m} finds text containing between “n” and “m” occurrences of the previous character or expression; so a{2,3} will find “aa” and “aaa”, but only the first 3 characters in “aaaa”).

Regular Expressions in MS-Word

()

- Round brackets have no effect on the search pattern, but are used to divide the pattern into logical sequences where you wish to re-assemble those sequences in a different order during the replace – or to replace only part of that sequence.
- They must be used in pairs and are addressed by number in the replacement.
- Eg: (Tulasi) (Prasad) replaced by \2 \1 (note the spaces in the search and replace strings) – will produce Prasad Tulasi or replaced by \2 alone will give Prasad.

^

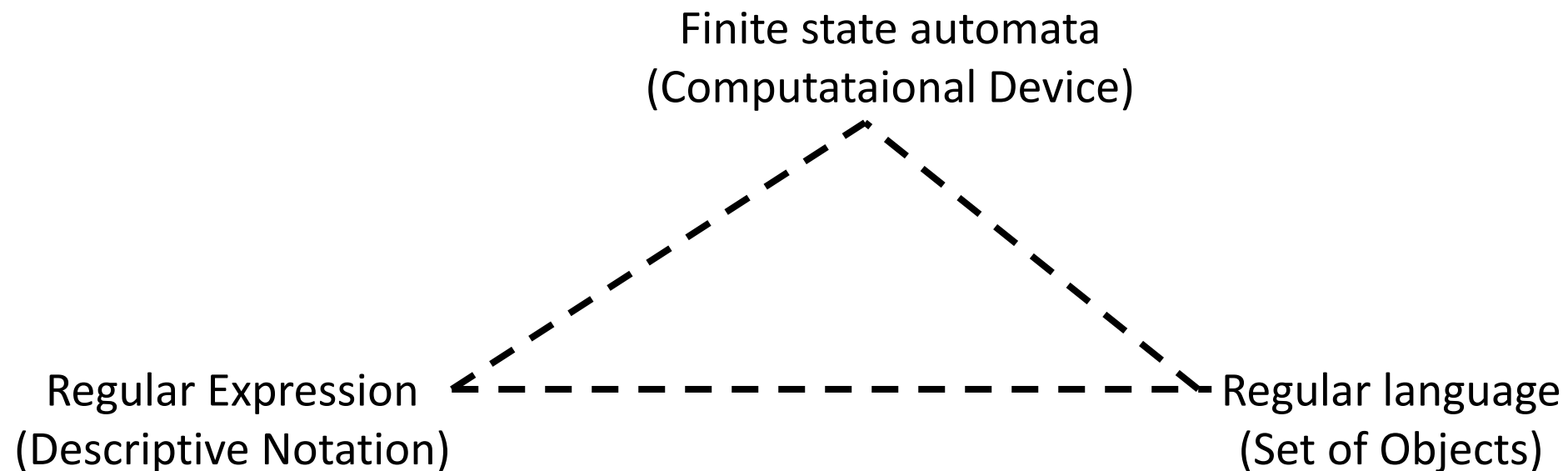
- The ^ (“caret”) character is not specific to wildcard searches but it sometimes has to be used slightly differently from normal, when searching for wildcards.

Finite Automata

- ❑ The **regular expression** is more than just a convenient meta-language for text searching.
- ❑ Any **regular expression** can be implemented as a **finite-state automaton**.
- ❑ Symmetrically, any **finite-state automaton** can be described with a **regular expression**.
- ❑ **Regular expression** is one way of characterizing a particular kind of formal language called a **regular language**.
- ❑ Both **regular expressions** and **finite-state automata** can be used to describe **regular languages**.

Finite Automata

The relationship between finite state automata, regular expression, and regular language



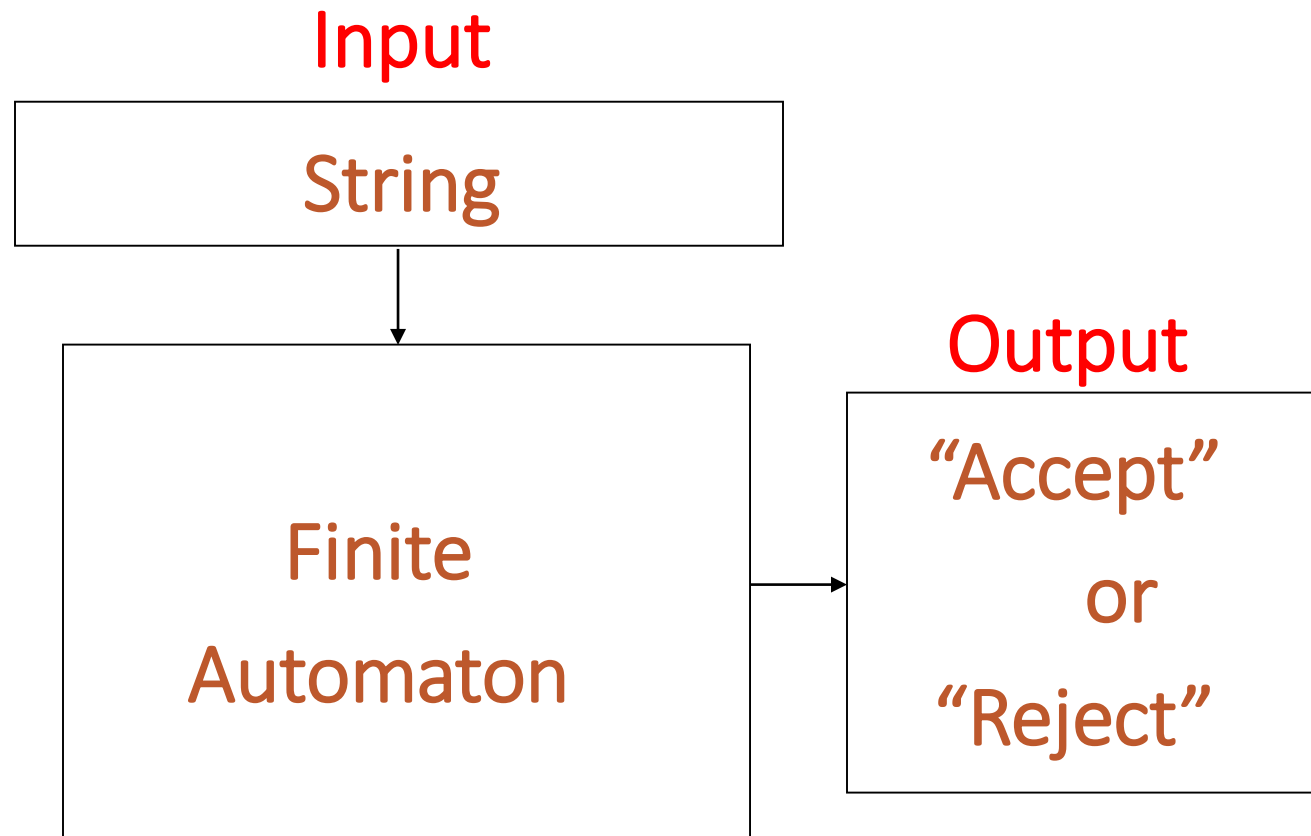
What is a Finite-State Automaton?

- ❑ An alphabet of *symbols*,
- ❑ A finite set of *states*,
- ❑ A *transition function* from *states and symbols* to *states*,
- ❑ A distinguished member of the set of states called the *start state*, and
- ❑ A distinguished subset of the set of states called *final states*.
- ❑ FSA recognize the regular languages represented by regular expressions
- ❑ Directed graph with labeled nodes and arc transitions

Formally

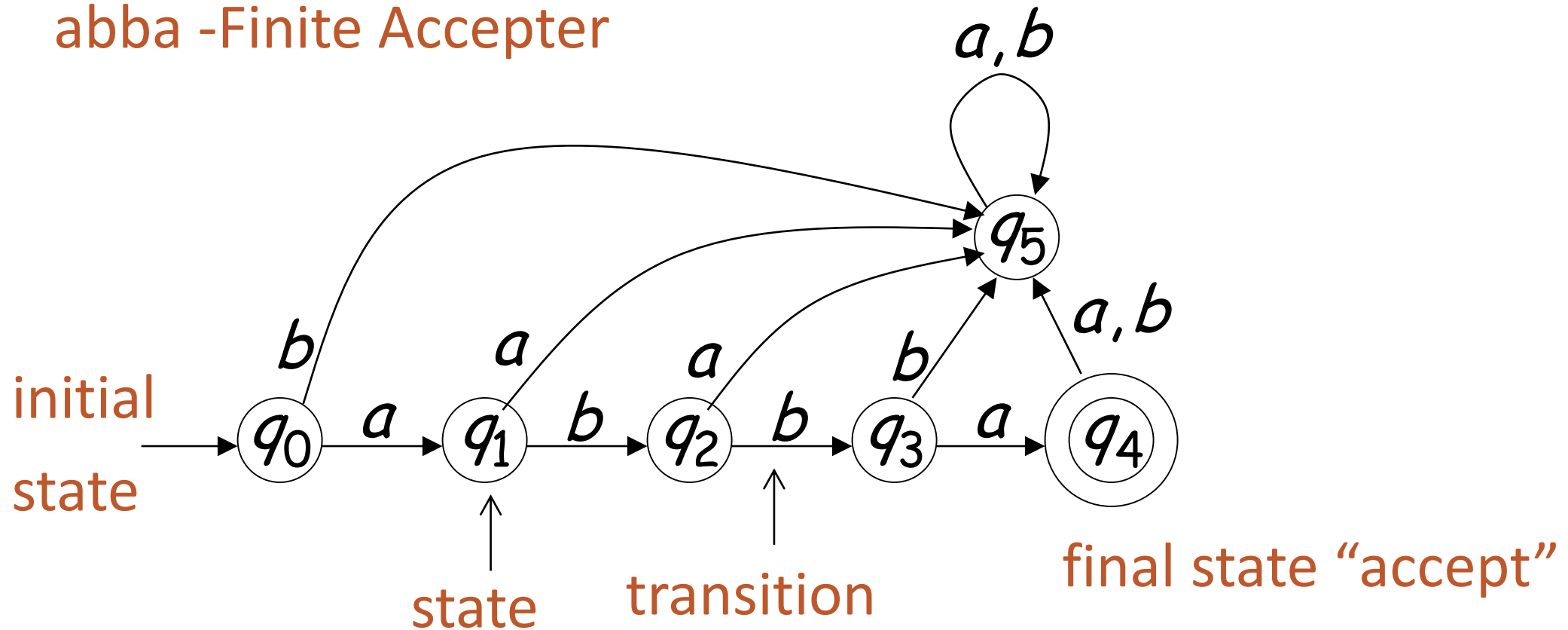
- **FSA** is a 5-tuple consisting of
 - **Q**: a finite set of N states q_0, q_1, \dots, q_N
 - **Σ** : a finite input alphabet of symbols
 - **q_0** : the start state
 - **F**: the set of final states, $F \subseteq Q$
 - **$\delta(q,i)$** : a transition function mapping **$Q \times \Sigma$ to Q**

FSA Acceptor

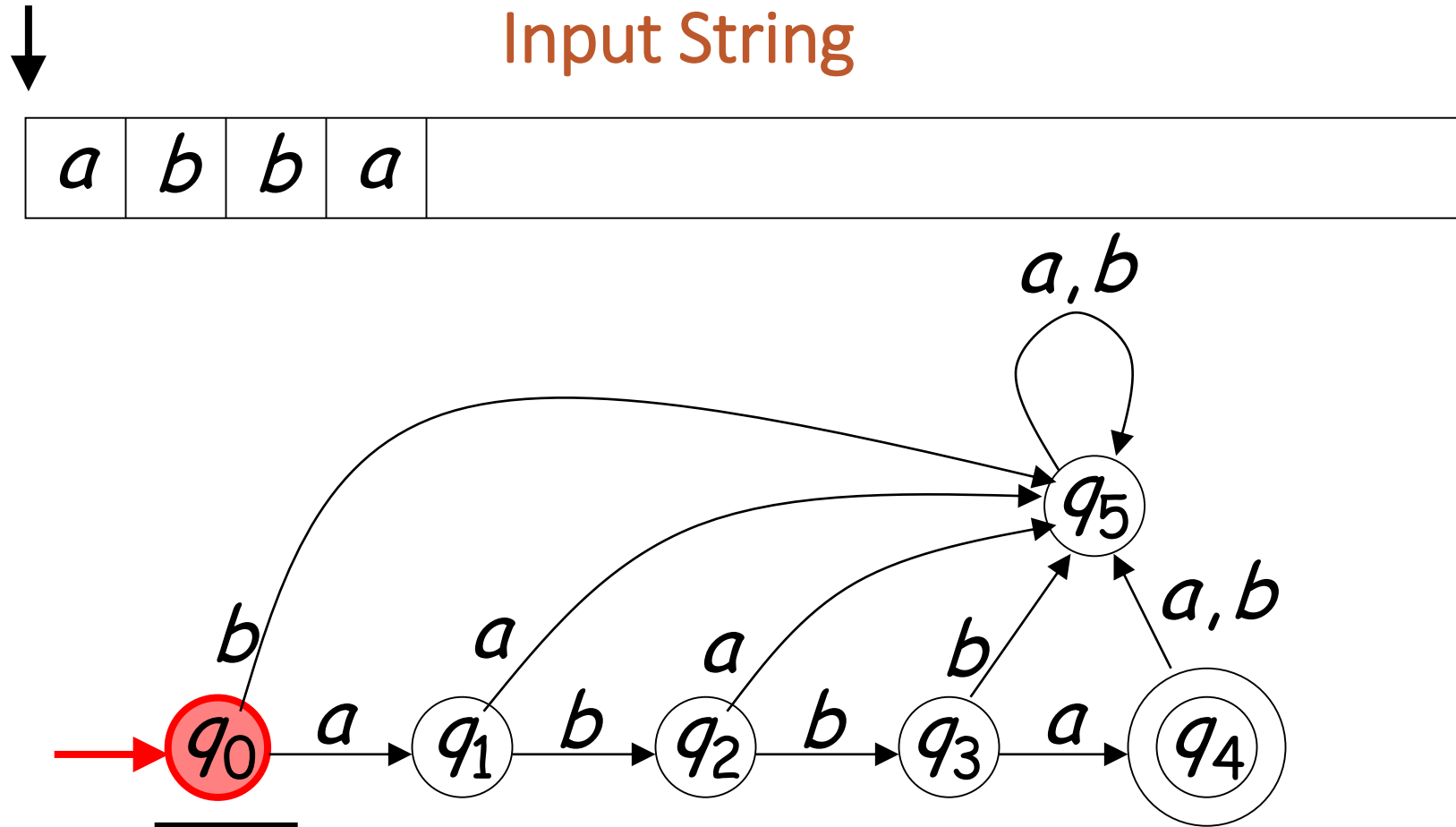


Transition Graph

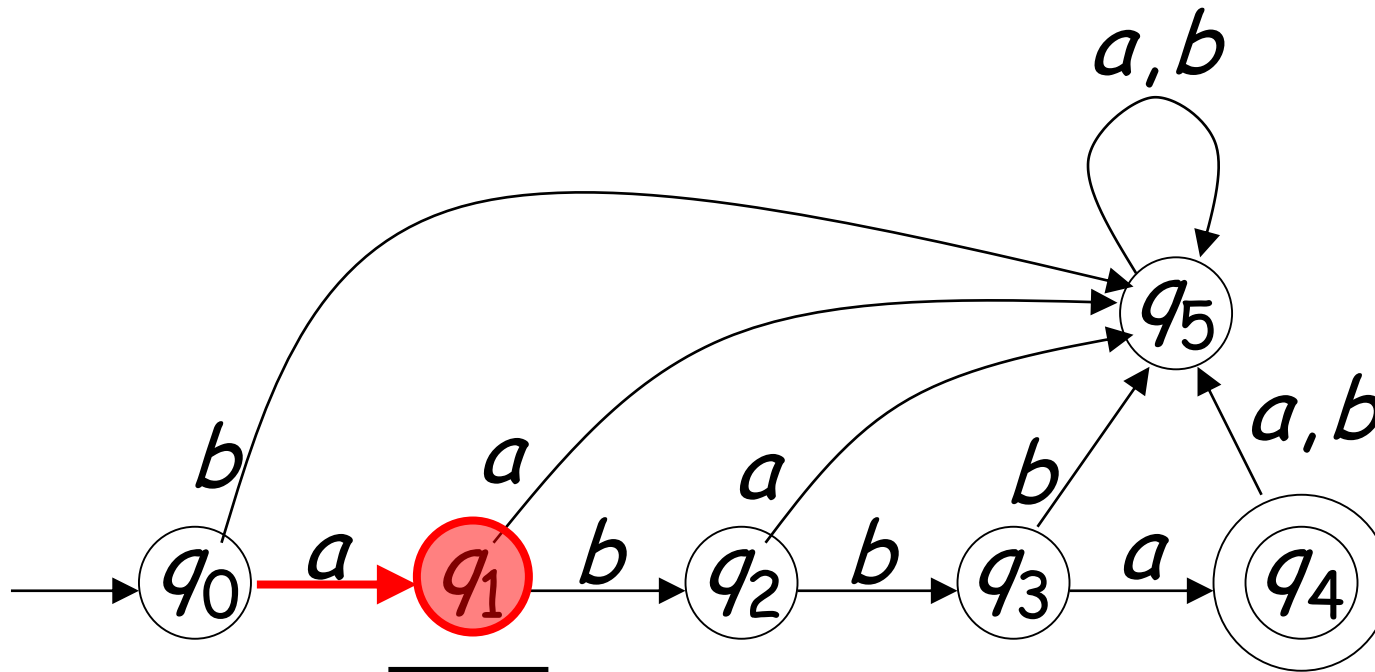
abba -Finite Acceptor



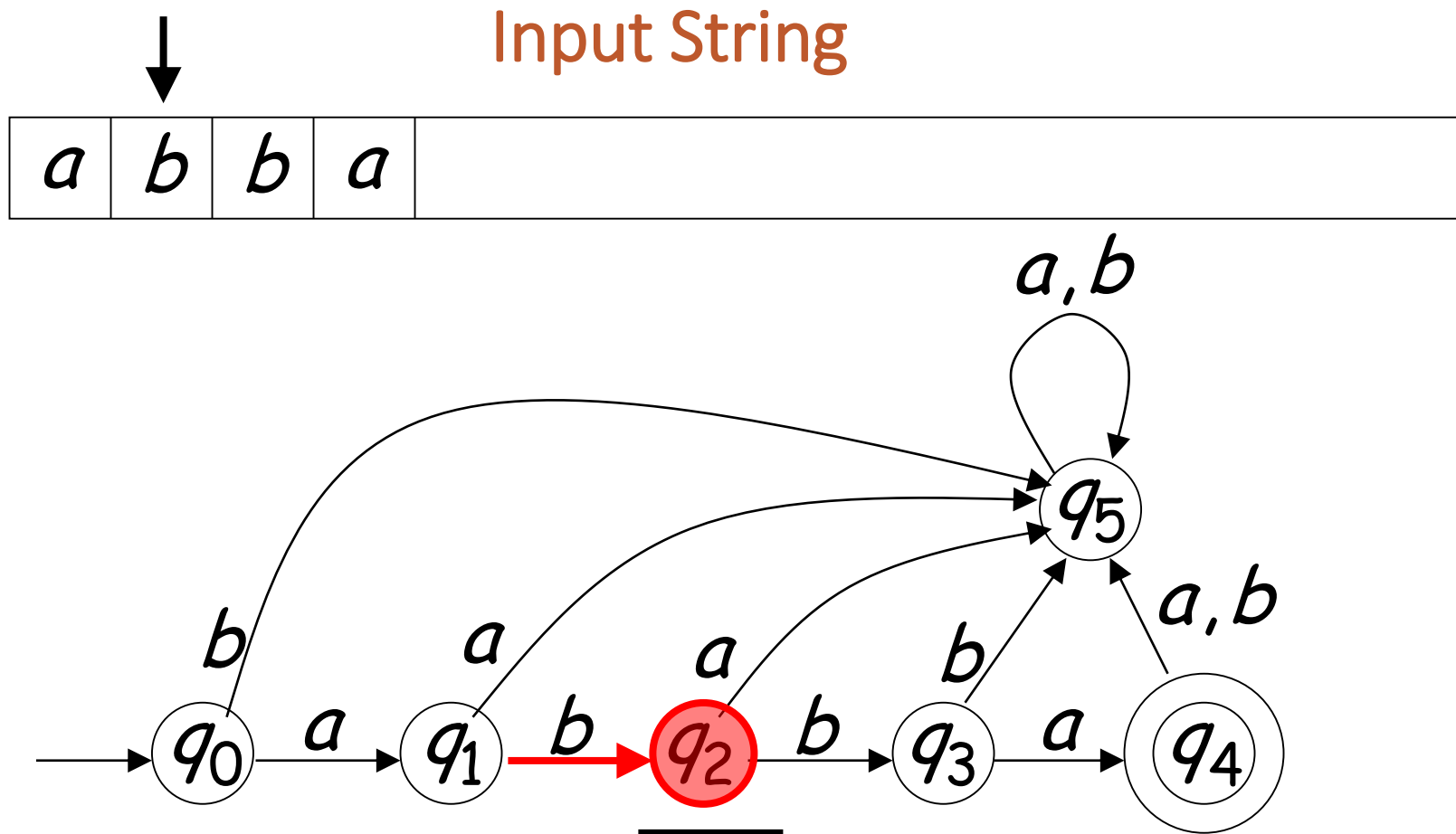
Initial Configuration



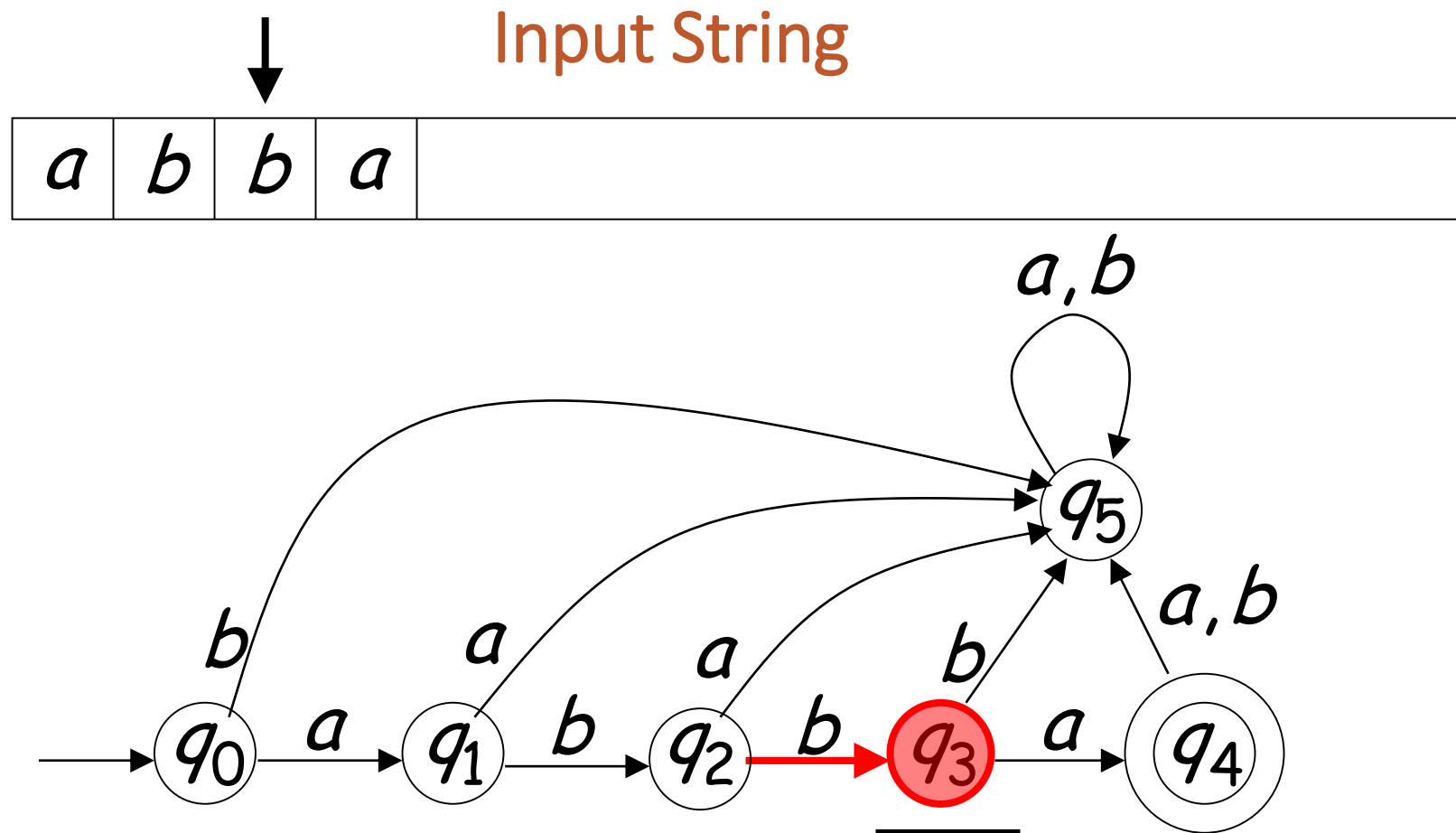
Reading Input



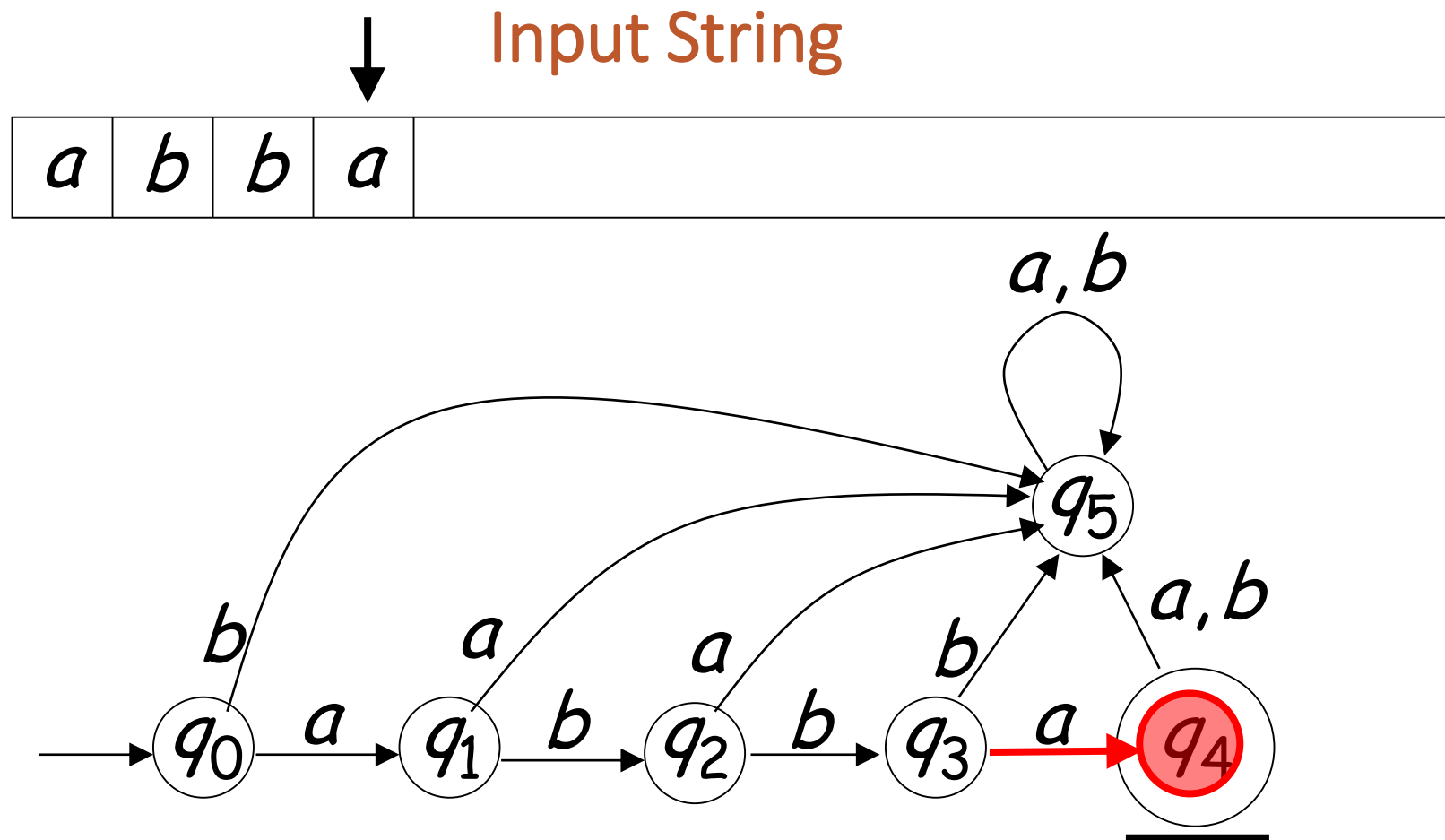
Reading Input



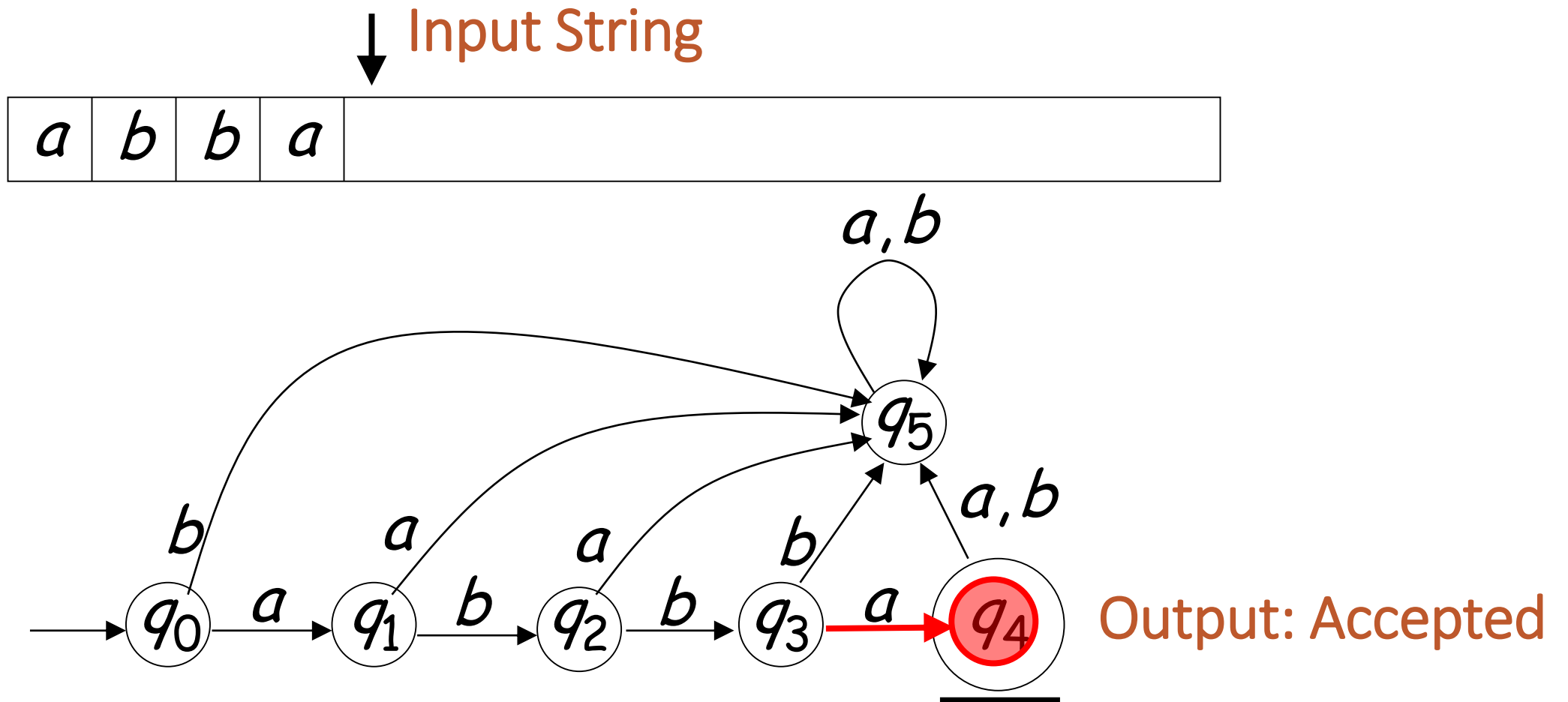
Reading Input



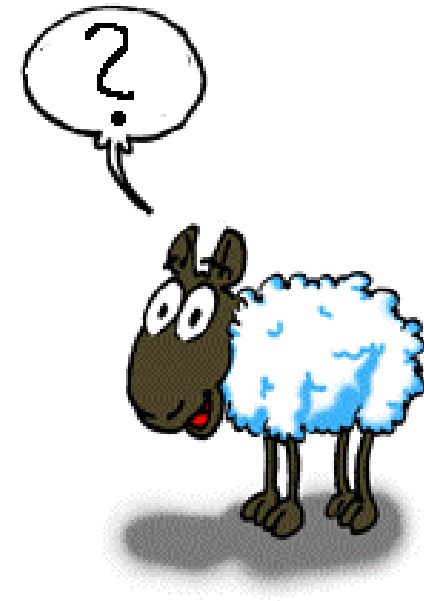
Reading Input



Reading Input



Using an FSA to Recognize Sheep talk



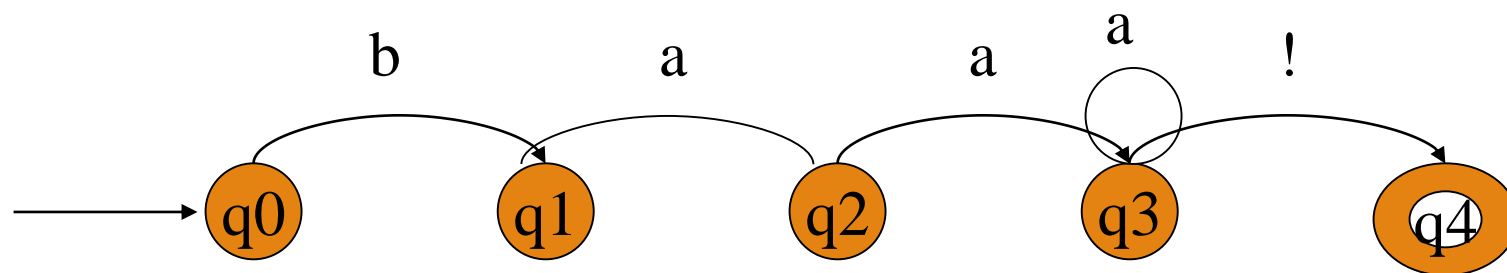
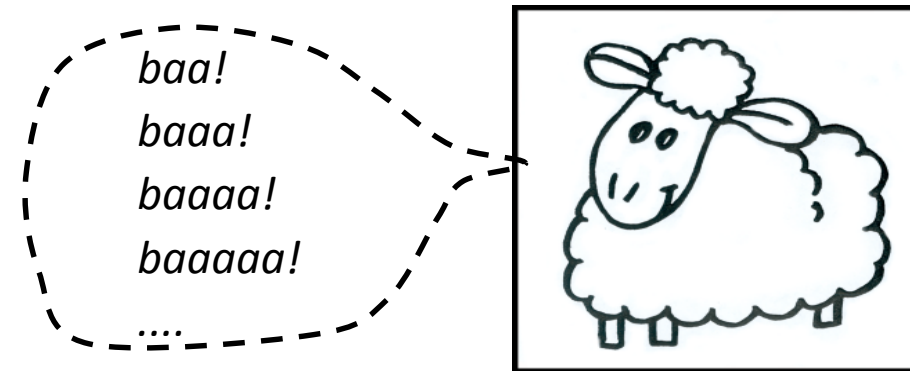
Using an FSA to Recognize Sheep Talk

Sheep language can be defined as any string from the following (infinite) set:

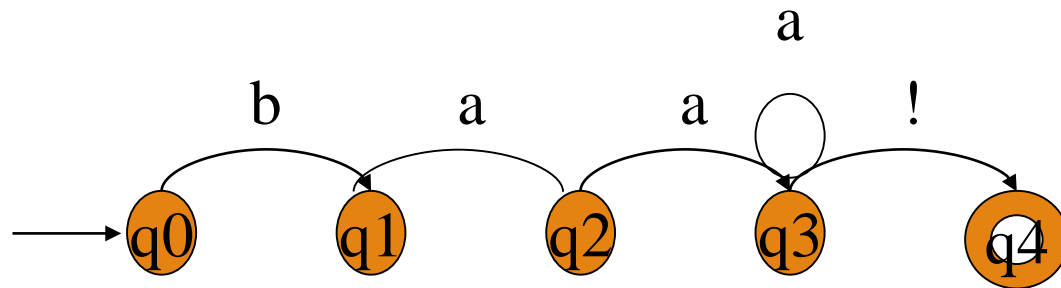
The regular expression for this kind of sheeptalk is

`/baa+!/`

All RE can be represented as FSA



State Transition Table for Sheep Talk



State	Input		
	b	a	!
0(null)	1	∅	∅
1	∅	2	∅
2	∅	3	∅
3	∅	3	4
4:	∅	∅	∅

Algorithm

```
function D-RECOGNIZE(tape,machine) returns accept or reject
index <- Beginning of tape
current-state <- Initial state of machine
loop
    if End of input has been reached then
        if current-state is an accept state then
            return accept
        else
            return reject
    elseif transition-table[current-state,tape[index]] is empty then
        return reject
    else
        current-state <- transition-table[current-state,tape[index]]
        index <- index +1
```

Using an FSA to Recognize Sheep Talk

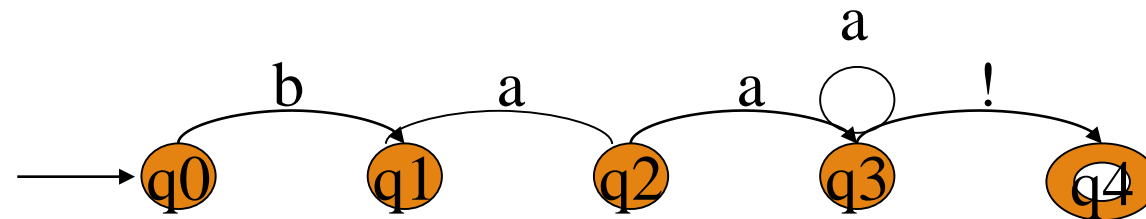
□ FSA recognizes (**accepts**) strings of a regular language

□ baa!

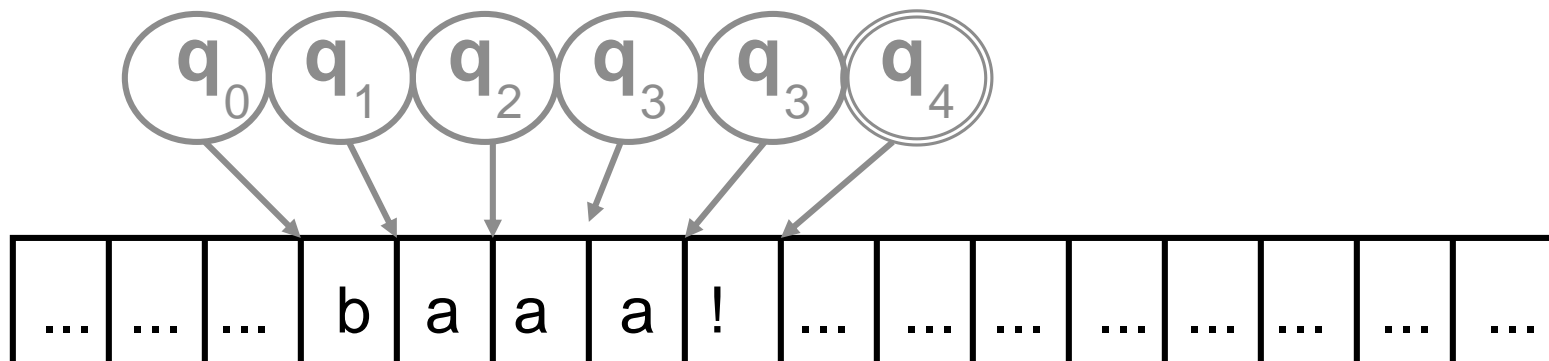
□ baaa!

□ baaaa!

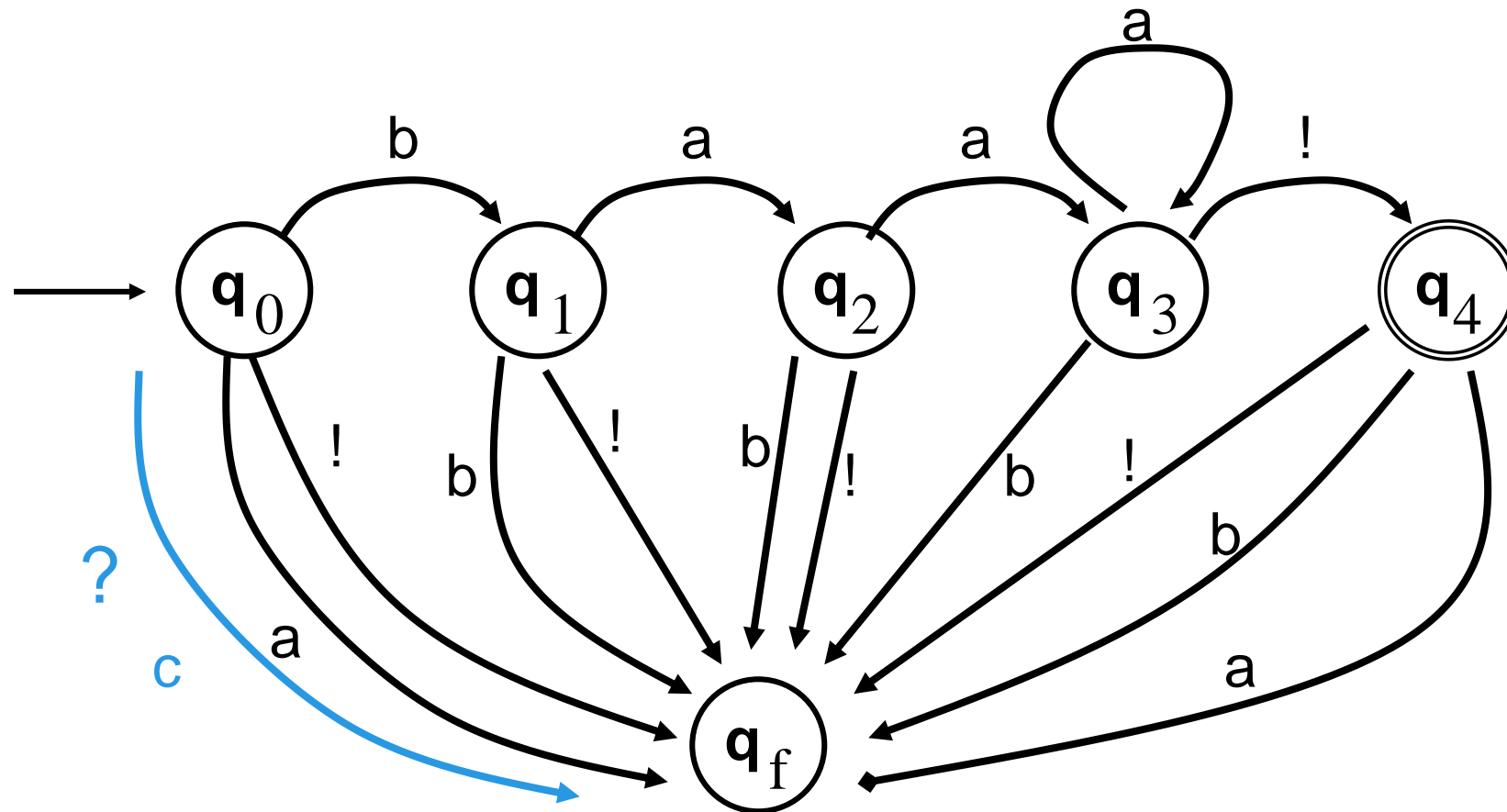
□ ...



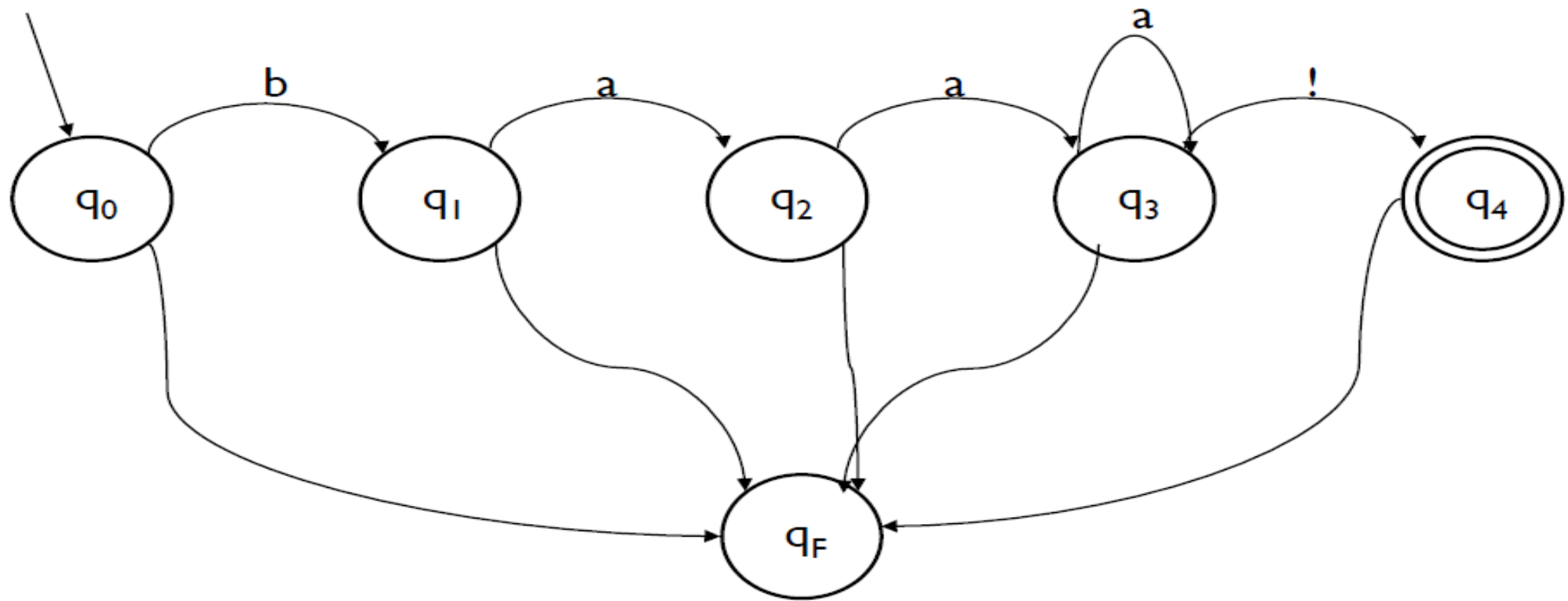
□ Tracing the execution of FSA on some sheep talk



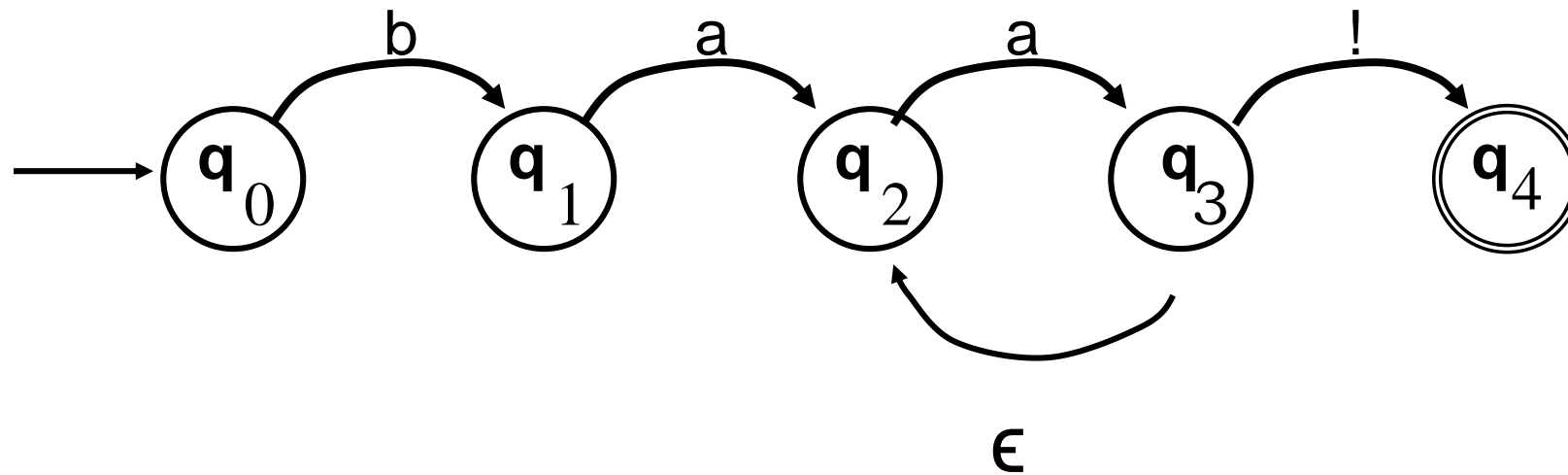
Adding a fail state to FSA



Adding an else arch

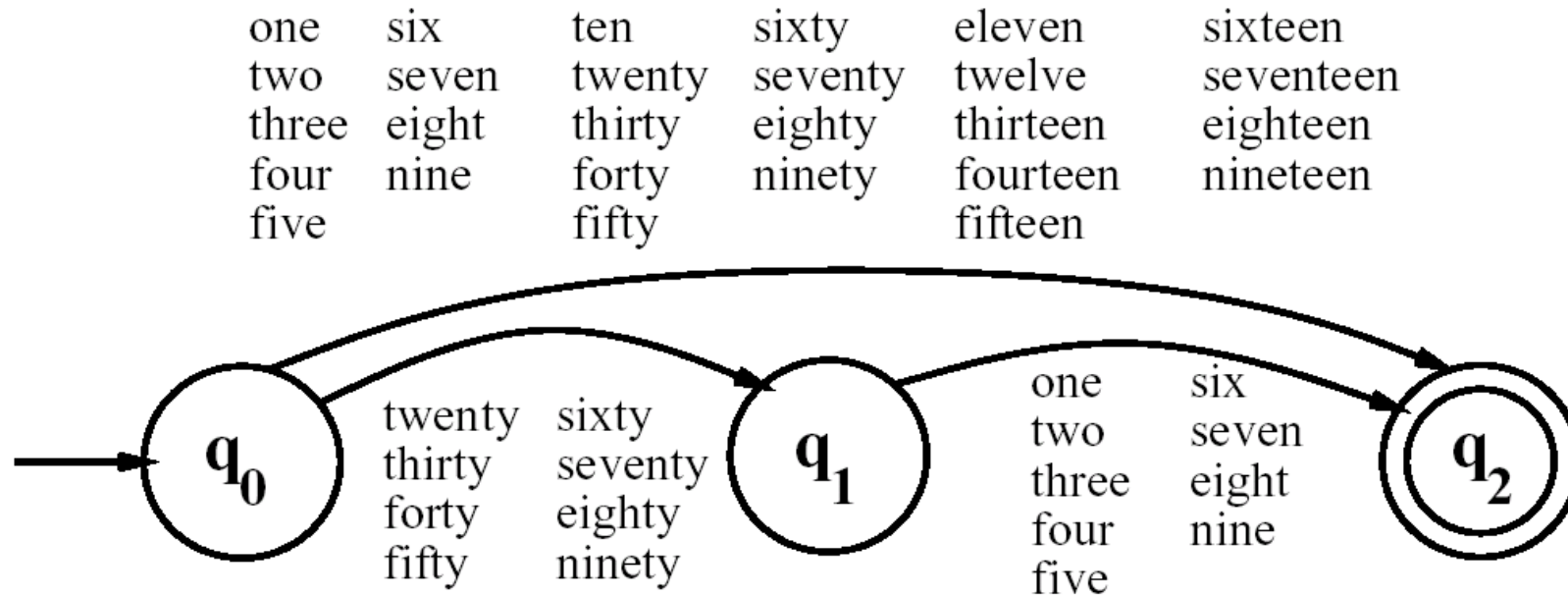


Adding ϵ Transition



Example FSA

An FSA for the words of English numbers 1-99



FSA for NLP

- **Word Recognition**
- **Dictionary Lookup**
- **Spelling Conventions**

Word Recognition

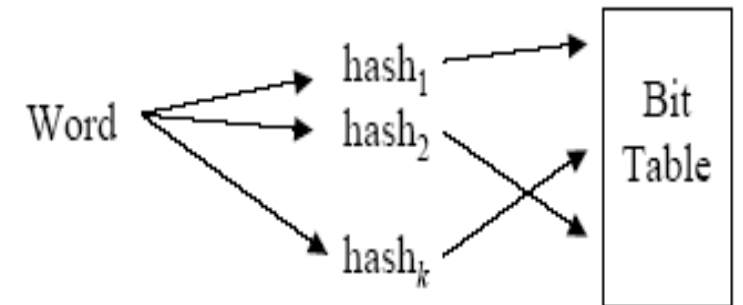
A word recognizer takes a string of characters as input and returns “yes” or “no” according as the word is or is not in a given set.

Solves the *membership problem*.

- e.g. Spell Checking, Scrabble(Un-ordered Concatenation)

Approximate methods

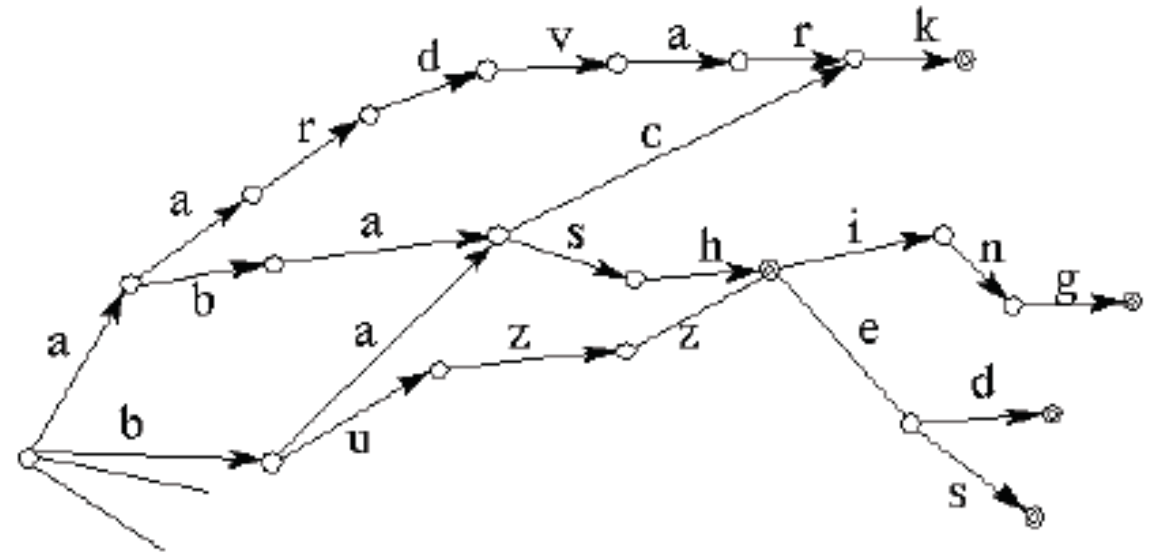
- Has right set of letters (any order).
- Has right sounds (Soundex).
- Random (suprimposed) coding (Unix Spell)



Word Recognition

Exact Methods

- Hashing
- Search (linear, binary ...)
- Digital search (“Tries”)
- Finite-state automata



Dictionary Lookup

Dictionary lookup takes a string of characters as input and returns “yes” or “no” according as the word is or is not in a given set *and returns information about the word.*

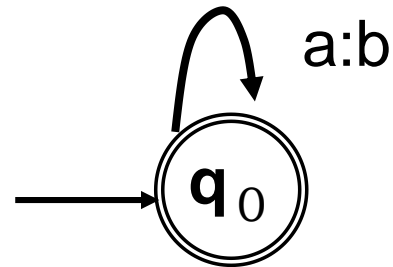
Lookup Methods

Approximate — guess the information

- If it ends in “ed”, it’s a past-tense verb.
- Exact — store the information for finitely many words
- Table Lookup
 - Hash
 - Search

Finite State Transducers

A finite state transducer essentially is a finite state automaton that works on two (or more) tapes. The most common way to think about transducers is as a kind of "translating machine". They read from one of the tapes and write onto the other.



$a:b$ at the arc means that in this transition the transducer reads a from the first tape and writes b onto the second.

Finite State Transducers

Transducer behaves as follows in the different modes.

- generation mode: It writes a string of a s on one tape and a string of b s on the other tape. Both strings have the same length.
- recognition mode: It accepts when the word on the first tape consists of exactly as many a s as the word on the second tape consists of b s.
- translation mode (left to right): It reads a s from the first tape and writes an b for every a that it reads onto the second tape.
- translation mode (right to left): It reads b s from the second tape and writes an a for every b that it reads onto the first tape.
- relator mode: Computes relations between sets

FST vs FSA

FSA can act as a

- Recognizer
- Generator
- 5 tuple Representation
- Equivalent to regular languages

FST can act as a

- Recognizer
- Generator
- Translator
- Set relator
- 7 tuple Representation
- Equivalent to regular relations

Lexical

	c	a	t	+N	+PL			
--	----------	----------	----------	-----------	------------	--	--	--

Surface

	c	a	t	s				
--	----------	----------	----------	----------	--	--	--	--

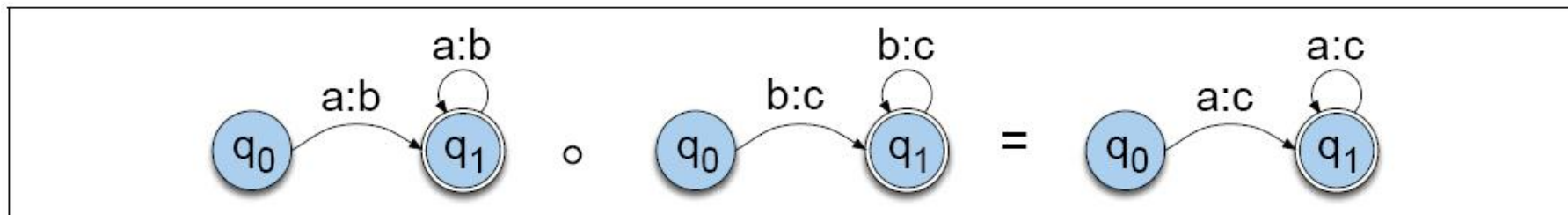
FST Operations

- Inversion: Switching input and output labels

- If T maps from I to O , T^{-1} maps from O to I

- Composition:

- If T_1 is a transducer from I_1 to O_1 and T_2 is a transducer from I_2 to O_2 , then $T_1 \circ T_2$ is a transducer from I_1 to O_2 .



FST for NLP

- ❑ Tokenization
- ❑ Morphological analysis
- ❑ Transliteration
- ❑ Parsing
- ❑ Translation
- ❑ Speech recognition
- ❑ Spoken language understanding

END