Lecture 2

Rana Salama

INTRODUCTION TO STATISTICAL NLP

CSCI 3907/6907

FALL 2020

Notes

- Assignment 1:
  - Will be posted this week
  - Submission using blackboard
  - All assignment related questions must be directed to the TA
  - No grace period for this one
- Participation

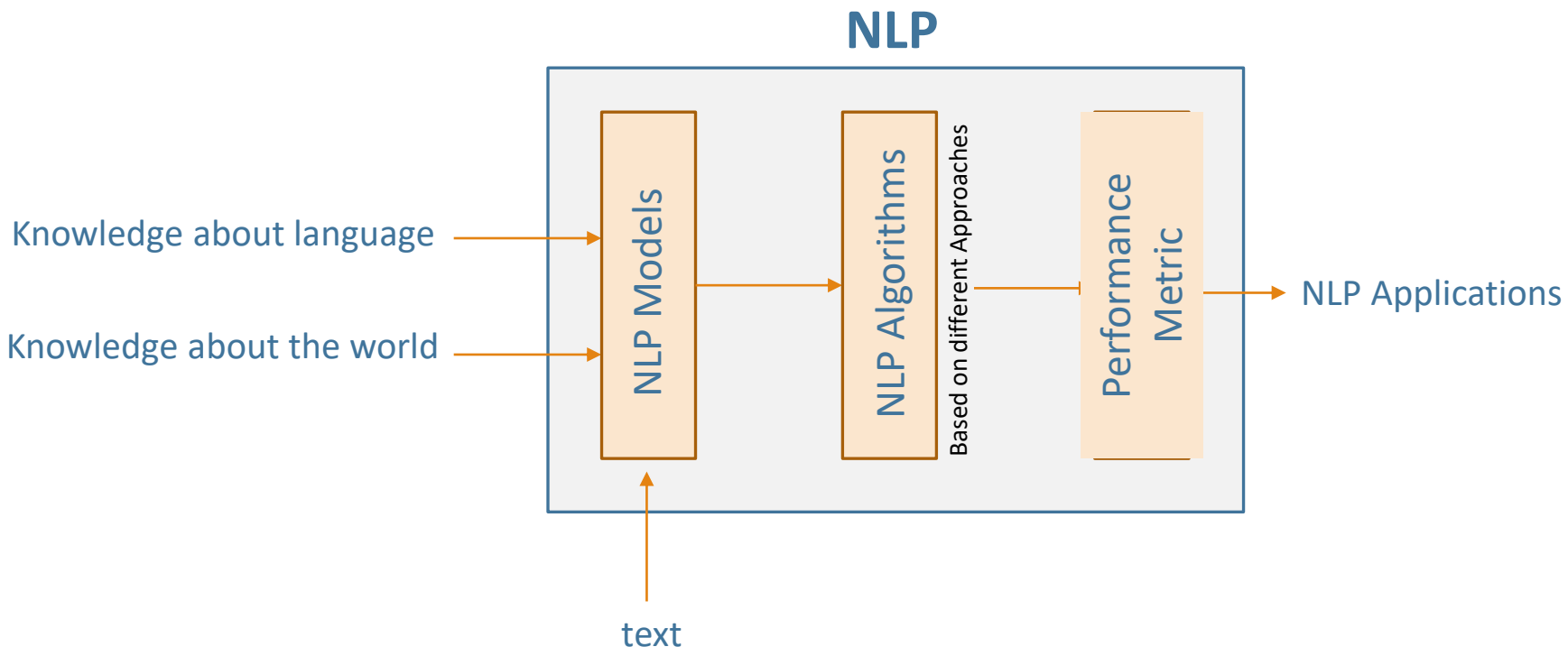
Final Project

- Project Ideas
  - https://nirantk.com/awesome-project-ideas/#forecasting
  - https://www.kaggle.com/
- Groups
  - Topic of interest (9/18)
  - Live discussion (9/25)
  - Project ideas (9/30)
  - Final groups (10/4)
  - Project proposals (10/30)

Review

- What is NLP?
  - Building programs that can recognize, analyze , and generate text and speech
  - Processing unstructured data
- NLP Applications: Machine Translation, Sentiment Analysis, Part of Speech Tagging … etc.
- Ambiguity results from the existence of multiple possibilities for each level
- NLP Approaches: Rule based and statistical based approaches

Review

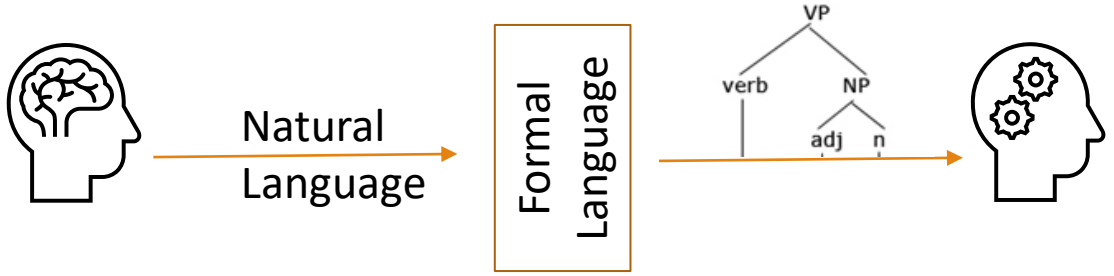

NLP
Knowledge about language
Knowledge about the world
NLP Models
NLP Algorithms
Based on different Approaches
Performance Metric
NLP Applications
text

Basic Text Processing

Regular Expressions & Automata

Formal Languages

- A **formal language** is a set of **strings**, each string composed of symbols from a finite symbol-set called an **alphabet Σ**.
- A model which can both **generate** and **recognize** all and only the strings of a formal language acts as a definition of the formal language.
- A formal language may bear no resemblance at all to a real language (natural language), but can model a subset of expressions:
  - Parts of the phonology, morphology, or syntax.
  - Date and time expressions.



Regular Languages

- A regular language → a formal language that can be fully specified using one of the following:
  - Regular expression
  - Finite state automaton
  - Regular grammar

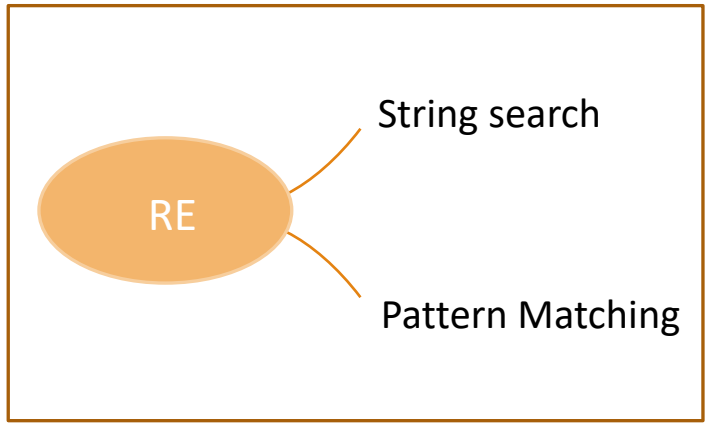
Regular Expressions

Hi,
Thank you for your email.
These are my contact details.
Cell phone: 202-123-4567
Office: 202-123-0000
Home: 202-000-4567
Address: 1234 natural st.

Phone numbers ?

Regular Expressions

- A formal language for specifying text strings.
- First developed in 1956 by Stephen Kleene.
- Used in
  - Unix tools like grep and sed
  - Programming languages: Perl, python, Java, etc.
- Requires a **pattern** and a **corpus** of texts
- Returns all text that matches the pattern

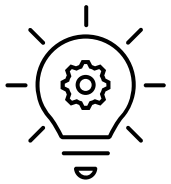


Regular Expressions: Disjunctions

Woodchuck is a rodents that eats wood. So, how much wood would a woodchuck chuck if a woodchuck could chuck wood?

/woodchuck/	woodchuck
/Woodchuck/	Woodchuck
[wW]oodchuck	Woodchuck, woodchuck

123456789



[1234567890]	Any single digit
[0-9]	Any single digit

Ranges

Regular Expressions: Disjunctions

Letters inside square brackets []

Pattern	Matches
[wW]oodchuck	Woodchuck, woodchuck
[1234567890]	Any digit

Pattern	Matches	Examples
[A-Z]	An upper case letter	Drenched Blossoms
[a-z]	A lower case letter	My BEANS WERE IMPATIENT
[0-9]	A single digit	Chapter 1: Down the Rabbit Hole

Negation in Disjunction

Negations `[^Ss]`

- Caret means negation only when first in []

Pattern	Matches	Examples
[^A-Z]	Not an upper case letter	My BEANS WERE IMPATIENT Happy 2 learn!!
[^Ss]	Neither ‘S’ nor ‘s’	The seven sailors were seasick
[^e^]	Neither e nor ^	here^-^
a^b	The pattern a carat b	Look up a^b now

More Disjunction

Woodchuck is a rodents that eats wood. So, how much wood would a woodchuck chuck if a woodchuck could chuck wood? Groundhog is another name for woodchuck. In fact, woodchucks/groundhogs only eat tree bark if they arise from hibernation before vegetation has sprung. So to answer the riddle – not much.

groundhog|woodchuck → groundhog, woodchuck

The pipe | for disjunction

More Disjunction

Pattern	Matches
groundhog|woodchuck	groundhog, woodchuck
yours|mine	yours , mine
a|b|c = [abc]	a, b, c
[gG]roundhog|[Ww]oodchuck	groundhog, Groundhog, woodchuck, Woodchuck

? * + .

Pattern	Matches	
colou?r	Optional previous char	color colour
oo*h!	0 or more of previous char	oh! ooh! oooh! ooooh!
o+h!	1 or more of previous char	oh! ooh! oooh! ooooh!
baa+		baa baaa baaaa baaaaa
beg.n	any character between beg and n	begin begun begun beg3n

Repetition

Example

- Describing a sheep language
  - Set of alphabet= {a,b}
- Language any strings that could be formed
- Any string from the following (infinite) set:

  Baa!

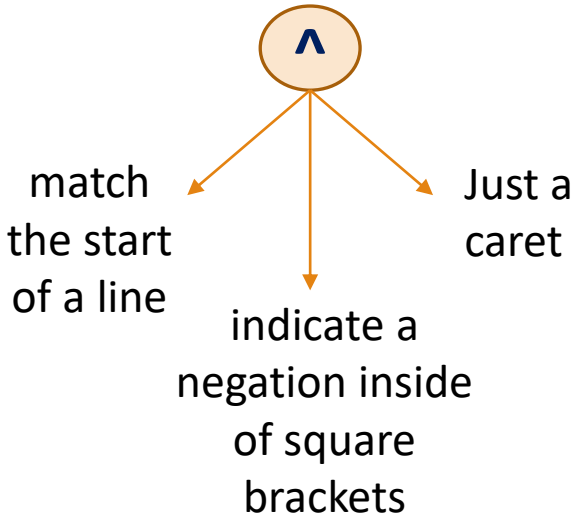
  Baaa!

  Baaaa!

  Baaaaa!
- Regular expression: baa+!

Anchors ^ $

Anchors are special characters that anchor regular expressions to particular places in a string



Pattern	Matches
^[A-Z]	Palo Alto
^[^A-Za-z]	1 “Hello”
\.$	The end.
.$	The end? The end!

Other Operators & Special Characters

Pattern	matches
\d	[0-9], any digit
\D	Any non digit
\w	Any alphanumeric or space
\W	[^\w] any non-alphanumeric
\s	whitespace
\S	non whitespace
*, \?, \.	*, ?, . (Characters that need to be escaped)
\n	end of line
\t	tab
{n}	n occurrences of the previous char or expression
{n,m}	n to m occurrences of the previous char or expression
{n, }	At least n occurrences of the previous char or expression

Resources

https://www.regexpal.com/

https://regexr.com/

https://callumacrae.github.io/regex-tuesday/

https://regexone.com/

https://cheatography.com/davechild/cheat-sheets/regular-expressions/

Hands on!!

https://www.regexpal.com/

We looked!

Then we saw him step in on the mat.

We looked!

And we saw him!

The cat in the Hat!

We loooked!

and found 50 dollars under the mat.

we called 202-123-4567 and told them we are ready to buy a new hat!!

Example

Find me all instances of the word “the” in a text.

```
The
[tT]he
[^a-zA-Z][tT]he[^a-zA-Z]
```

Errors

- Two kinds of errors:
  - False positive (Type I)
    - Matching strings that we should not have matched (there, then, other)
  - False negative (Type II)
    - Not matching things that we should have matched (The)

Errors – Cont.

- We always deal with these types of errors in NLP
- Reduce the error rate for an application involve:
  - Increasing accuracy or precision *(minimizing false positives)*
  - Increasing coverage or recall *(minimizing false negative)*

Substitution And Memory

Find a pattern in text, then replace it with another pattern.

```
s/pattern/replacement/
```

Capture groups: store a matched pattern in memory, then reuse it in substitution or later matches:

- The resulting match is stored in a numbered register (e.g \1 and \2)
  - Using memory in search:

```
/the (.*)er they were, the \1er they will be/
/the (.*)er they (.*), the \1er we \2/
```

  - Using memory in substitution: `s/([0-9]+)/<\1>/`

ELIZA

Use regular expressions and capture groups to generate responses:

http://psych.fullerton.edu/mbirnbaum/psych101/Eliza.htm

```
User:   You are like my father in some ways.
ELIZA:  WHAT RESEMBLANCE DO YOU SEE
User:   You are not very aggressive but I think you don't want me to
        notice that.
ELIZA:  WHAT MAKES YOU THINK I AM NOT VERY AGGRESSIVE
User:   You don't argue with me.
ELIZA:  WHY DO YOU THINK I DON'T ARGUE WITH YOU
User:   You are afraid of me.
ELIZA:  DOES IT PLEASE YOU TO BELIEVE I AM AFRAID OF YOU
```

Weizenbaum (1966)

```
s/.* I'M (depressed|sad) .*/I AM SORRY TO HEAR YOU ARE \1/
s/.* I AM (depressed|sad) .*/WHY DO YOU THINK YOU ARE \1/
s/.* all .*/IN WHAT WAY/
s/.* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE/
```

Simple ELIZA Implementation

```
#Upper case all characters

tr/a-z/A-Z/;

#Remove all non-words except for spaces

s/[^\w\s]//;

#Switch first person to second person

s/\bMY\b/YOUR/g;

s/\b(I|ME)\b/YOU/g;

s/\bAM\b/ARE/g;

#Responses

s/.*\b(HI|HELLO|HOWDY)\b.*/$1 THERE. HOW ARE YOU DOING?/;

s/.*\bYOU ARE (DEPRESSED|SAD|UNHAPPY)\b.*/I AM SORRY TO HEAR YOU ARE $1./;

s/.*\bALL\b.*/IN WHAT WAY?/;

s/.*\bALWAYS\b.*/CAN YOU THINK OF A SPECIFIC EXAMPLE?/;

s/.*\bNOT\b.*/WHY NOT?/;

s/.*\bYES\b.*/I SEE/;

#Add prompt

s/^/>> /;
```

Web Scraping

https://blog.apify.com/what-is-web-scraping-1b548f8d6ac1

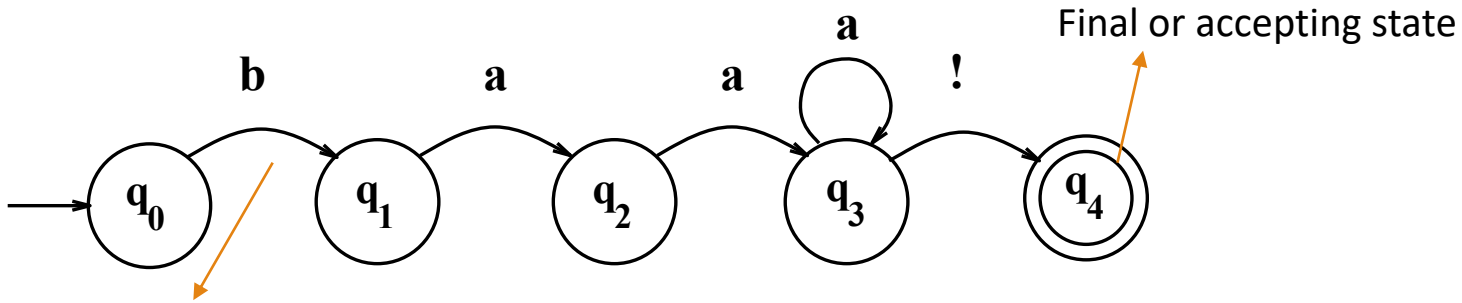
Summary

- Regular expressions play a surprisingly large role
  - Sophisticated sequences of regular expressions are often the first model for any text processing text
  - Early version of chatbots (ELIZA), text normalization, tokenization, stemming, sentence segmentation, string similarities
- For many hard tasks, we use machine learning classifiers
  - But regular expressions are used as features in the classifiers
  - Can be very useful in capturing generalizations

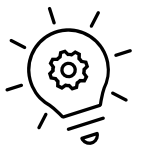
Finite State Automata

Finite State Automata (FSA)

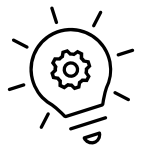
- An abstract model of a computer which can:
  - Read an input string
  - and change its internal state depending on the current input
- Every automaton
  - Defines a language (the set of strings it accepts)
  - Consists of N **states** and a **transition** matrix.
  - Has a **start** state, and a set of final "**accept**" states



When b is received as an input, a transition from q0 to q1 is performed



What strings does this automata accept?

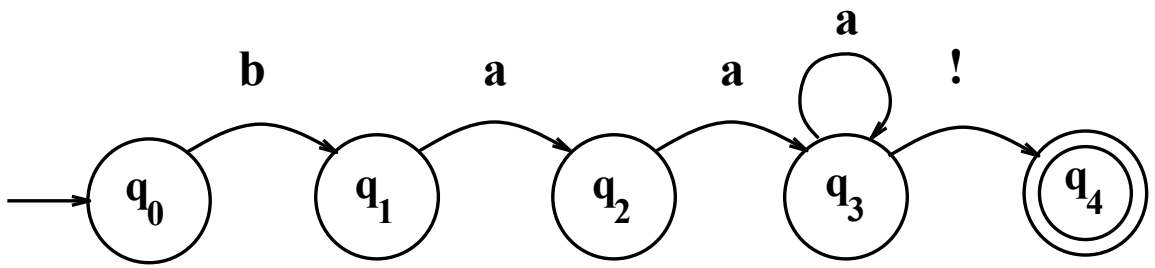


What is the matching RE?

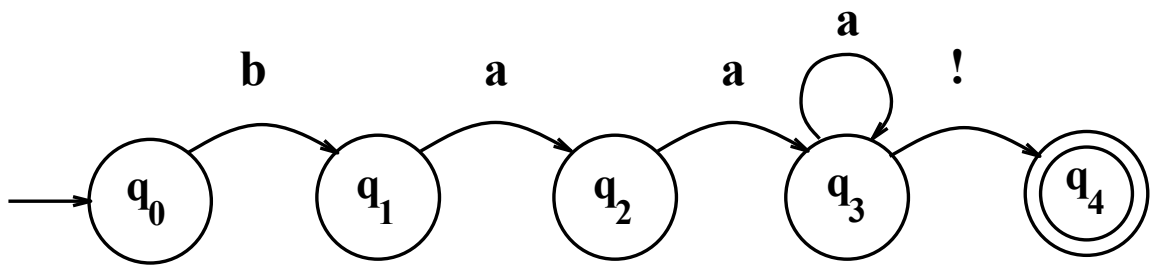
Formally

A regular language is a 5-tuple consisting of

- Q: set of states {q0,q1,q2,q3,q4}
- Σ: an alphabet of symbols {a,b,!}
- q0: a start state in Q
- F: a set of final states in Q {q4}
- δ(q,i): a transition function mapping Q x Σ to Q



Finite State Automata (FSA)



- **Strings accepted:**
  - baa!
  - baaaaa!
- **Strings not accepted:**
  - ba!
  - abc
  - Babb
  - !aab

	Input		
State	b	a	!
0	1	Ø	Ø
1	Ø	2	Ø
2	Ø	3	Ø
3	Ø	3	4
4:	Ø	Ø	Ø

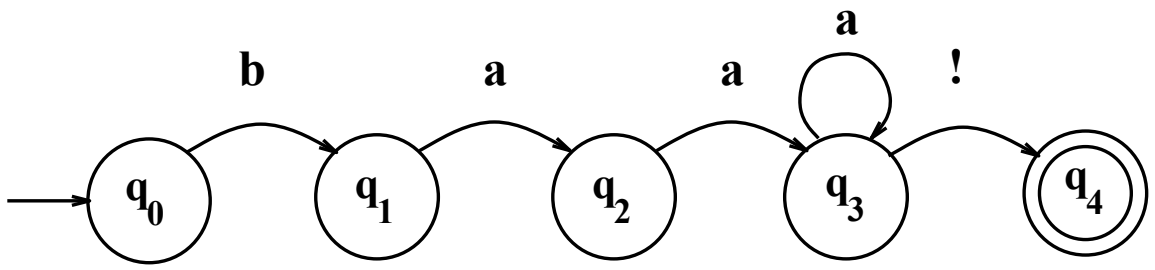
Transition Table

Formally

FSA=(Q, Σ, q_0,F, δ(q,i))

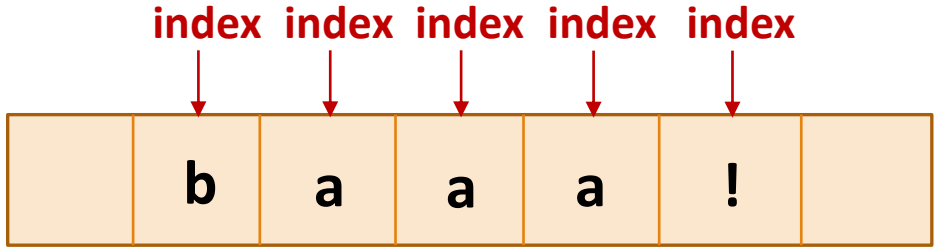
A regular language is a 5-tuple consisting of

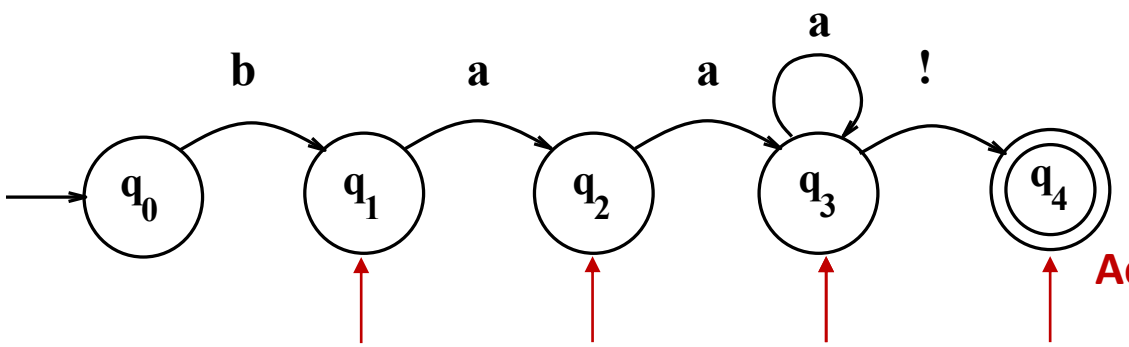
- Q: set of states {q0,q1,q2,q3,q4}
- Σ: an alphabet of symbols {a,b,!}
- q0: a start state in Q
- F: a set of final states in Q {q4}, F ⊆ Q
- δ(q,i): a transition function mapping Q x Σ to Q



How it works

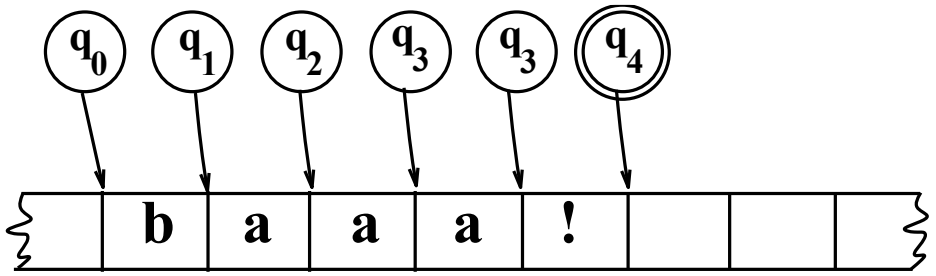
Imagine the input being written on a long tape broken into cells





Accepted string

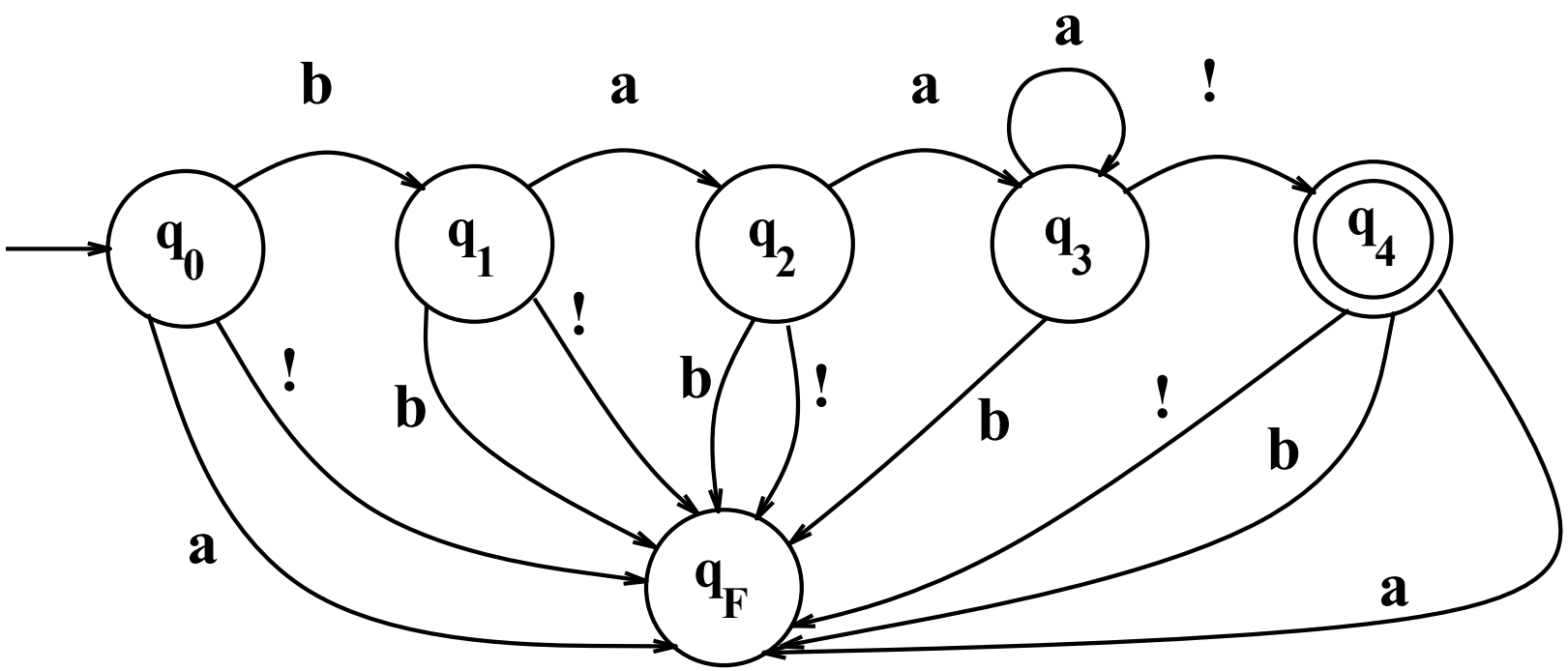
	Input		
State	b	a	!
0	1	Ø	Ø
1	Ø	2	Ø
2	Ø	3	Ø
3	Ø	3	4
4:	Ø	Ø	Ø



The Tape Metaphor

Finite State Automata (FSA)

Adding a **fail** state



Deterministic Recognition Algorithm

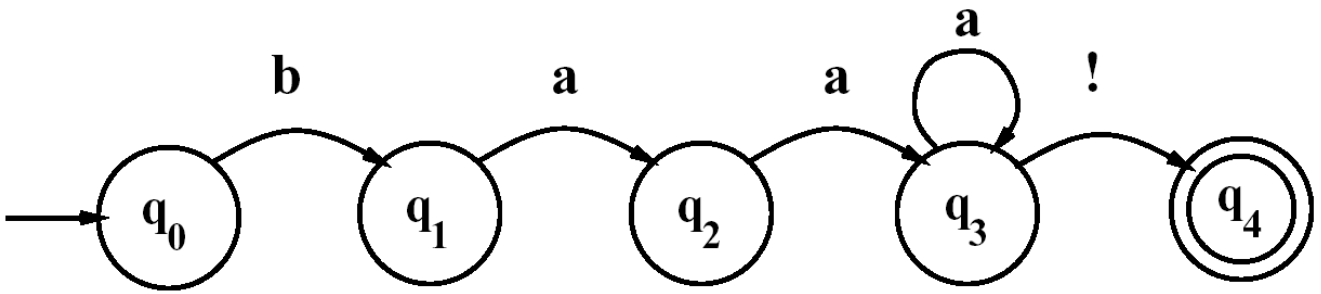
```
function D-RECOGNIZE(tape, machine) returns accept or reject

  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
    elsif transition-table[current-state,tape[index]] is empty then
      return reject
    else
      current-state ← transition-table[current-state,tape[index]]
      index ← index + 1
  end
```

Deterministic FSAs

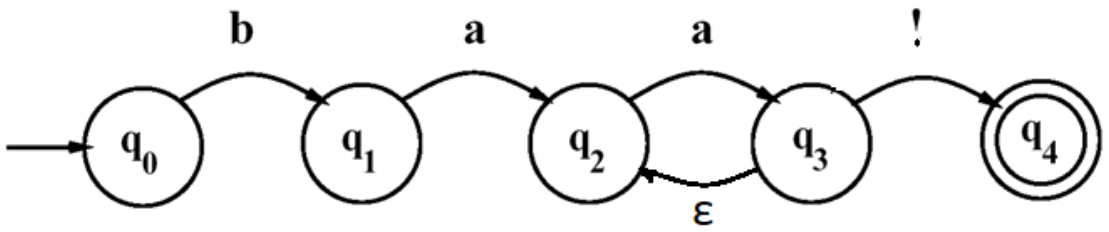
- These FSAs are deterministic
  - At each point in processing there is always one unique thing to do (no choices).
- D-RECOGNIZE is a simple table-driven interpreter
  - The algorithm is universal for all unambiguous languages.
  - To change the machine, you change the table.
- FSAs can be useful tools for recognizing – and generating – subsets of natural language
  - But they cannot represent all NL phenomena (e.g. $a^n b^n$)

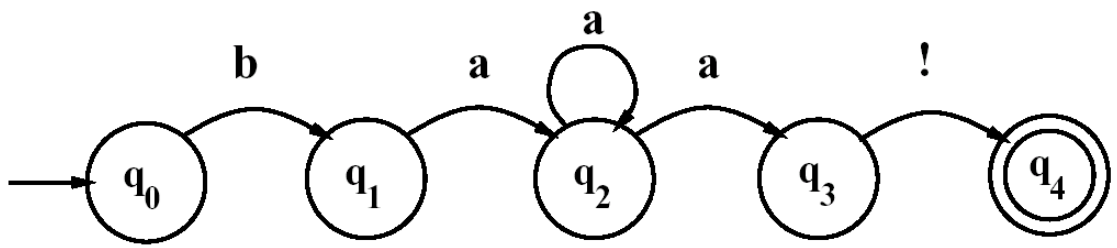
Non-Deterministic FSAs

Deterministic sheep language FSA



Non-deterministic sheep language FSA





Dealing with non-determinism

- At any choice point, we may follow the wrong arc
- Potential solutions:
  - Save backup states at each choice point
  - Look-ahead in the input before making choice
  - Pursue alternatives in parallel

Equivalence

- For every NFSA there exists an equivalent DFSA
  - (i.e. that accepts exactly the same set of strings).
- The resulting DFSA, may have many more states than the original NFSA
  - (up to 2^N states for a NFSA with N states).

FSAs and Regular Expressions: Examples

Strings of the form: [ab]+
i.e., L={a,b,ab,ba,aab,bab,aba,bba,...}

Set of zero or more a's

Strings of the form $91 or $99.00

Set of all lowercase alphabetic strings ending in b

[ab]* with exactly 2 a's