CS-AD 220 – Spring 2016

Natural Language Processing

Session 5: 11-Feb-16

Prof. Nizar Habash

Some slides are adapted from Jurafsky and Martin's course slides on Speech and Language Processing

NYUAD Course CS-AD 220 – Spring 2016
Natural Language Processing

Assignment #1
Unix Tools and Regular Expressions
Assigned Feb 4, 2016

Due Feb 18, 2016 (11:59pm)

I. Grading & Submission

This assignment is about the use of regular expressions (regex) and a set of Unix tools for quick text processing. The assignment accounts for 10% of the full grade. Section III below has a set of questions. The student needs to answer them all. The specific number of points for each question is provided. The student should submit a **PDF** file containing the answers to each question and sub-question in order. The student should also include the commands and the result of applying the commands by copying and pasting from the terminal. Each student must work alone. This is not a group effort.

The assignment is due on Feb 18 before midnight (11:59pm). For late submissions, 10% will be deducted from the homework grade for any portion of each late day. The student should upload the answer to NYU Classes (Assignment #1).

Assignment #1 posted on NYU Classes

Moving Legislative Day Class

- Spring Break is March 18 – 25, 2016
- Sat March 26, 2016 is a Legislative *Thursday*
- Move to

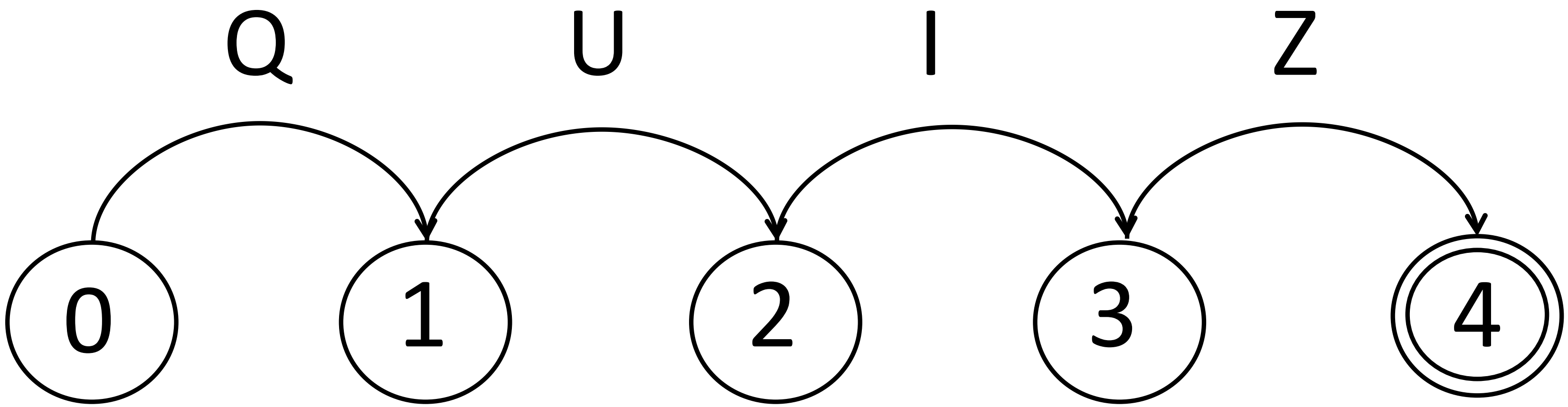
Sat April 2, 2016 at 10am

Same Classroom C2-E049

Syllabus Changes

- Readings
  - 2/16
  - 2/23
- Legislative day move

Date	Topic	Reading	Other
Thu 28th Jan	Introduction to NLP	None	
Tue 2nd Feb	Introduction to NLP	J+M Chap 1	
Thu 4th Feb	Regular Expressions and Basic Text Processing	J+M Chap 2 (intro,2.1)	Assignment 1: due by Session 7 (Feb 18 midnight)
Tue 9th Feb	Regular Expressions and Basic Text Processing	Handout	
Thu 11th Feb	Finite State Automata	J+M Chap 2 (2.2)	
Tue 16th Feb	Finite State Automata	J+M Chap 2 (2.3 to end) + Chap 3 (intro up to 3.2);	
Thu 18th Feb	Morphology and Finite State Transducers	J+M Chap 3 (3.2 up to 3.8);	Assignment 2: due by Session 13 (Mar 10 midnight)
Tue 23rd Feb	Morphology and Finite State Transducers	J+M Chap 3 (3.8 to end); NH Chap 4 (intro and 4.1 only)	
Thu 25th Feb	Language Modeling	J+M Chap 4 (intro up to 4.5)	
Tue 1st Mar	Language Modeling	J+M Chap 4 (4.5 up to 4.9)	
Thu 3rd Mar	Part-of-Speech Tagging	J+M Chap 5 (intro up to 5.5)	
Tue 8th Mar	Part-of-Speech Tagging	J+M Chap 5 (5.5 up to 5.8)	
Thu 10th Mar	Part-of-Speech Tagging	J+M Chap 5 (5.8 to end); handout (Pasha et al., 2014)	
Tue 15th Mar	MIDTERM	All previous readings	
	SPRING BREAK		
Sat 26th Mar	Class moved to April 2nd 10:00am (same room)		
Tue 29th Mar	Syntax and Parsing	J+M Chap 12	Assignment 3: due by Session 21 (Apr 14 midnight)
Thu 31st Mar	Syntax and Parsing	None	
Sat 2nd Apr	Syntax and Parsing	J+M Chap 13	CLASS STARTS @ 10:00 AM
Tue 5th Apr	Syntax and Parsing	None	
Thu 7th Apr	Machine Translation	J+M Chap 25 (intro up to 25.5)	
Tue 12th Apr	Machine Translation	Handout (Papineni et al., 2002)	
Thu 14th Apr	Machine Translation	J+M Chap 25 (25.5 to end);	Assignment 4: due by Session 27 (May 10)
Tue 19th Apr	Machine Translation	Handout (Zens et al., 2002)	


Q
U
I
Z
0
1
2
3
4

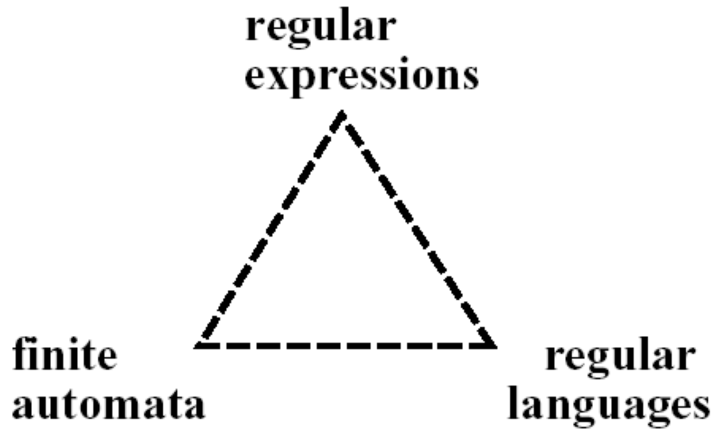
QUIZ

Finite-State Automata
Formal Languages

- A **formal language** is a set of strings, each string composed of symbols from a finite symbol-set call an **alphabet**.
- 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.
- The usefulness of an automaton for defining a language is that it can express an infinite set in a closed form.
- A formal language may bear no resemblance at all to a real language (**natural language**), but
  - We often use a formal language to model part of a natural language, such as parts of the phonology, morphology, or syntax.
- The term **generative grammar** is used in linguistics to mean a grammar of a formal language.

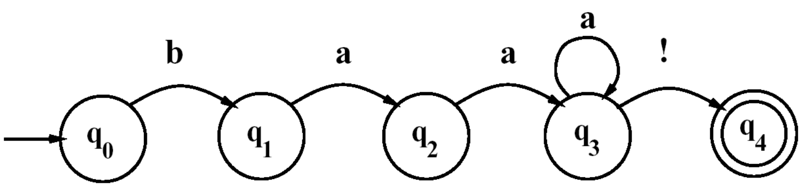
Finite-State Automata
Regular Languages

- A Regular Expression is one way of characterizing a particular kind of **formal language** called a **regular language**.
- A Regular Expression is one way of describing a Finite State Automata (FSA).

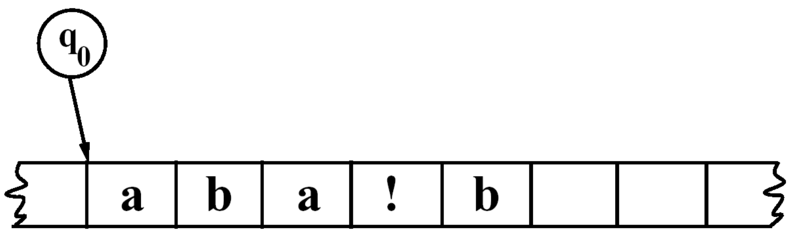


Finite-State Automata

Using an FSA to Recognize Sheeptalk



/baa+!/



A tape with cells

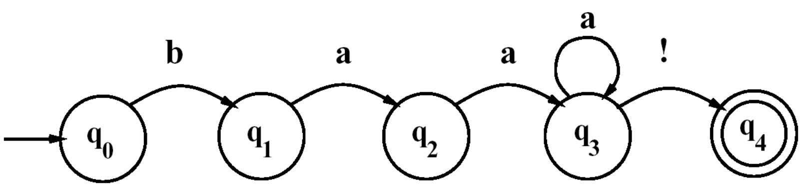
	Input		
State	b	a	!
0	1	∅	∅
1	∅	2	∅
2	∅	3	∅
3	∅	3	4
4:	∅	∅	∅

The transition-state table

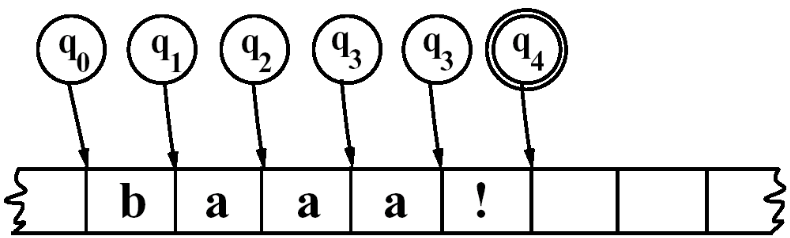
- Automaton (finite automaton, finite-state automaton (FSA))
- State, start state, final state (accepting state)

Finite-State Automata

Using an FSA to Recognize Sheeptalk



/baa+!/



A tape with cells

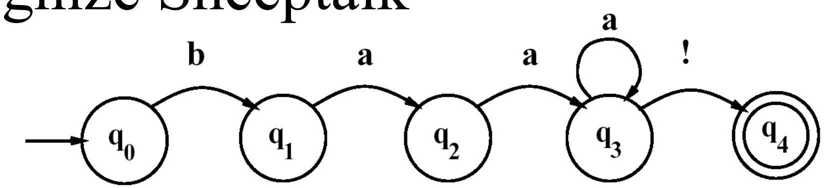
	Input		
State	b	a	!
0	1	∅	∅
1	∅	2	∅
2	∅	3	∅
3	∅	3	4
4:	∅	∅	∅

The transition-state table

- Automaton (finite automaton, finite-state automaton (FSA))
- State, start state, final state (accepting state)

Finite-State Automata

Using an FSA to Recognize Sheeptalk

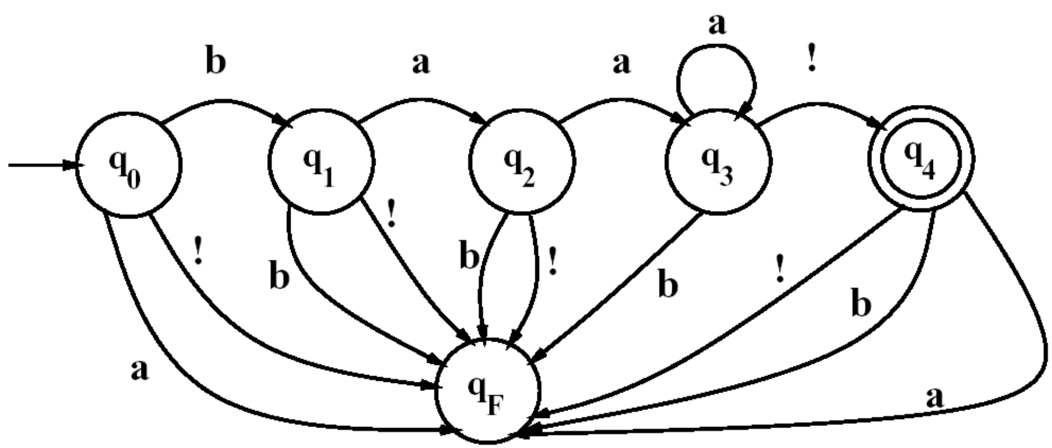


- A finite automaton is formally defined by the following five parameters:
  - Q: a finite set of N states $q_0, q_1, \ldots, q_N$
  - Σ: a finite input alphabet of symbols
  - q_0: the start state
  - F: the set of final states, $F \subseteq Q$
  - $\delta(q,i)$: the transition function or transition matrix between states. Given a state $q \in Q$ and input symbol $i \in \Sigma$, $\delta(q,i)$ returns a new state $q' \in Q$. δ is thus a relation from $Q \times \Sigma$ to Q;

	Input		
State	b	a	!
0	1	$\emptyset$	$\emptyset$
1	$\emptyset$	2	$\emptyset$
2	$\emptyset$	3	$\emptyset$
3	$\emptyset$	3	4
4:	$\emptyset$	$\emptyset$	$\emptyset$

Finite-State Automata
Using an FSA to Recognize Sheeptalk

- Adding a fail state

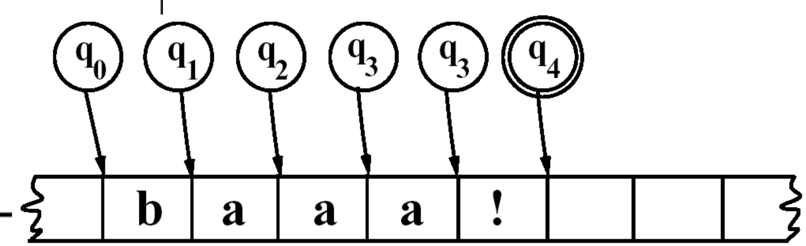


Finite-State Automata
Using an FSA to Recognize Sheeptalk

- An algorithm for deterministic recognition of FSAs.

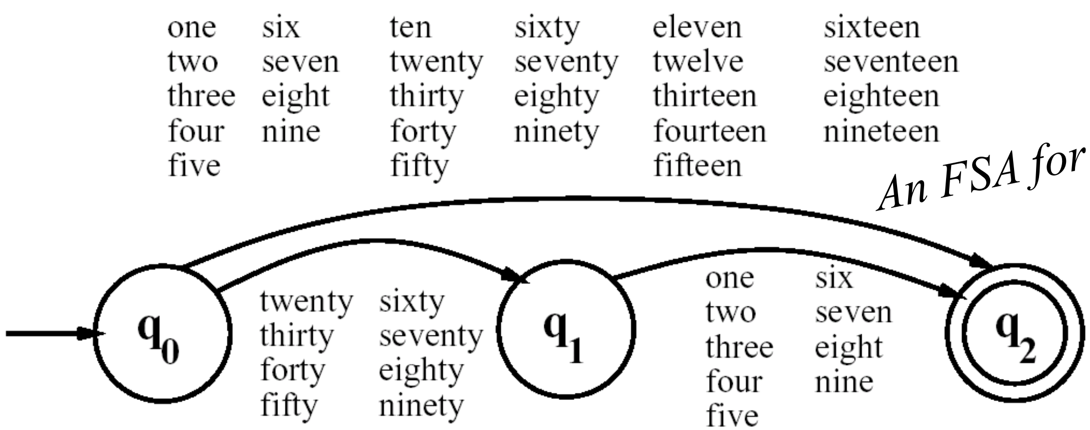
```
function D-RECOGNIZE(tape, machine) returns accept or reject

  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
```

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



Finite-State Automata

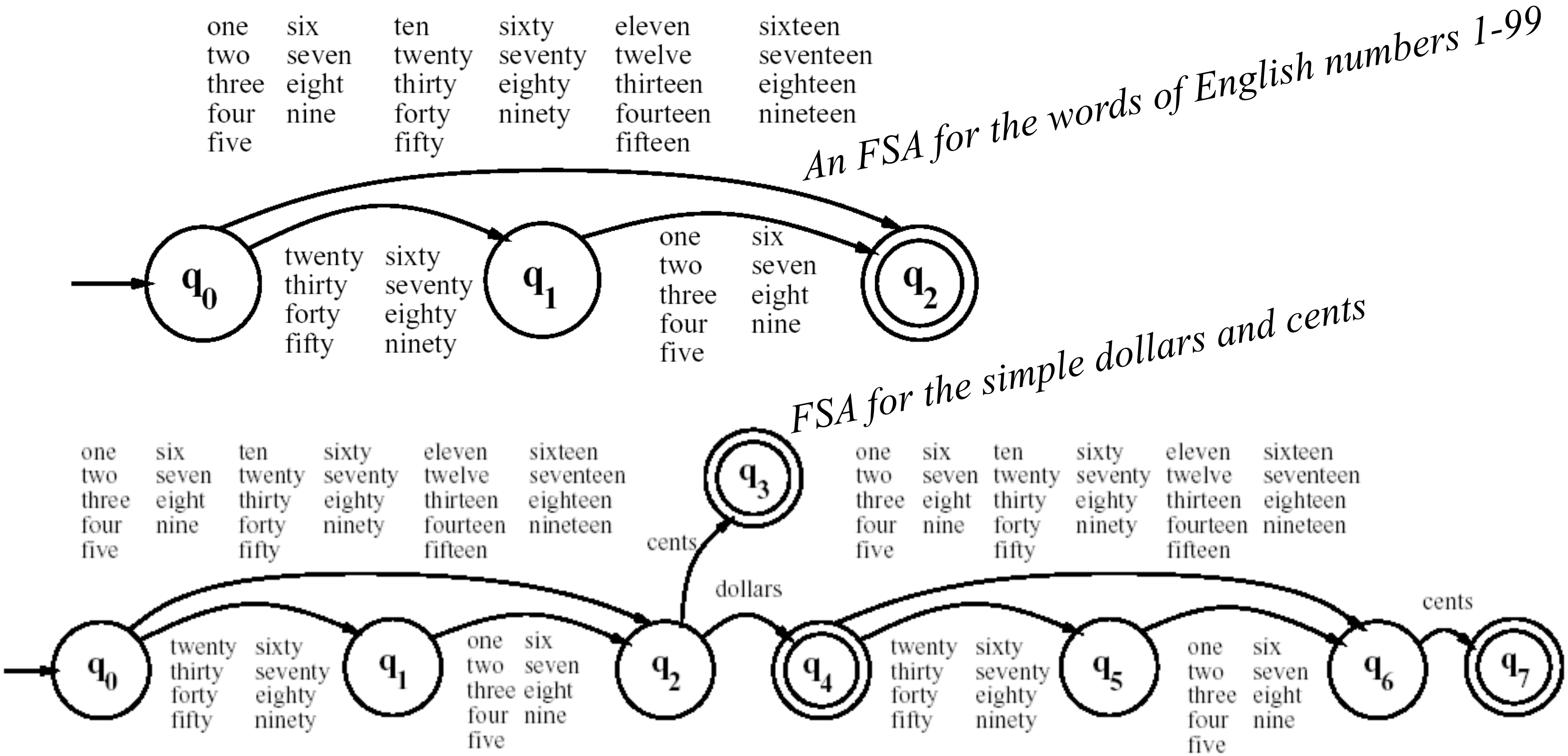
Another Example



An FSA for the words of English numbers 1-99

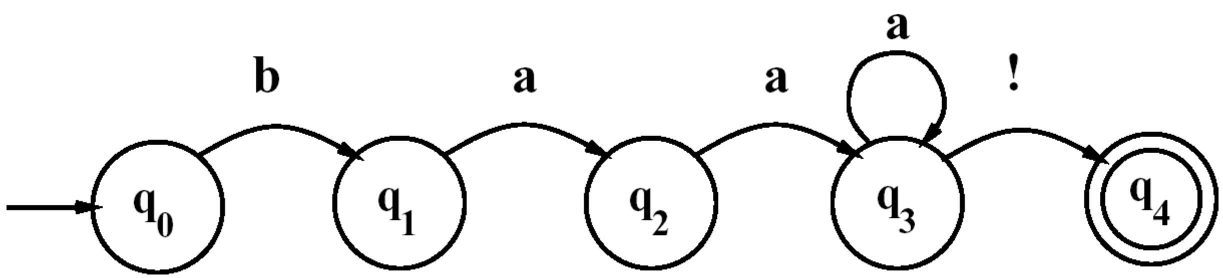
Finite-State Automata

Another Example

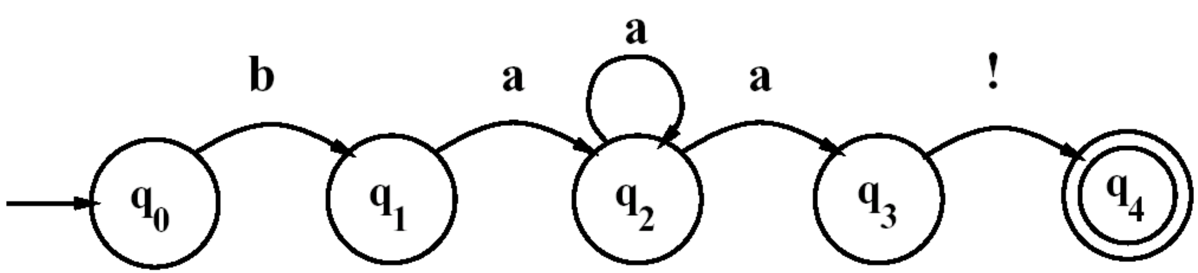


Finite-State Automata

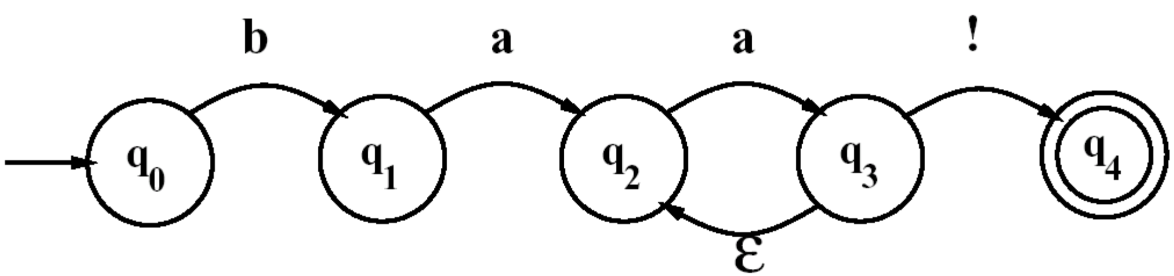
Non-Deterministic FSAs



Deterministic FSA



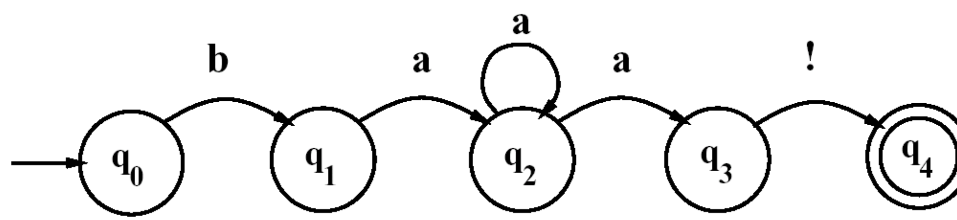
Non-Deterministic FSA



Non-Deterministic FSA

Finite-State Automata
Using an NFSA to Accept Strings

- Solutions to the problem of multiple choices in an NFSA
  - Backup
  - Look-ahead
  - Parallelism



	Input			
State	b	a	!	ε
0	1	∅	∅	∅
1	∅	2	∅	∅
2	∅	2,3	∅	∅
3	∅	∅	4	∅
4:	∅	∅	∅	∅

Finite-State Automata

Using an NFSA to Accept Strings

function ND-RECOGNIZE(*tape, machine*) **returns** accept or reject

```
agenda ← {(Initial state of machine, beginning of tape)}
current-search-state ← NEXT(agenda)
loop
  if ACCEPT-STATE?(current-search-state) returns true then
    return accept
  else
    agenda ← agenda ∪ GENERATE-NEW-STATES(current-search-state)
  if agenda is empty then
    return reject
  else
    current-search-state ← NEXT(agenda)
end
```

function GENERATE-NEW-STATES(*current-state*) **returns** a set of search-states

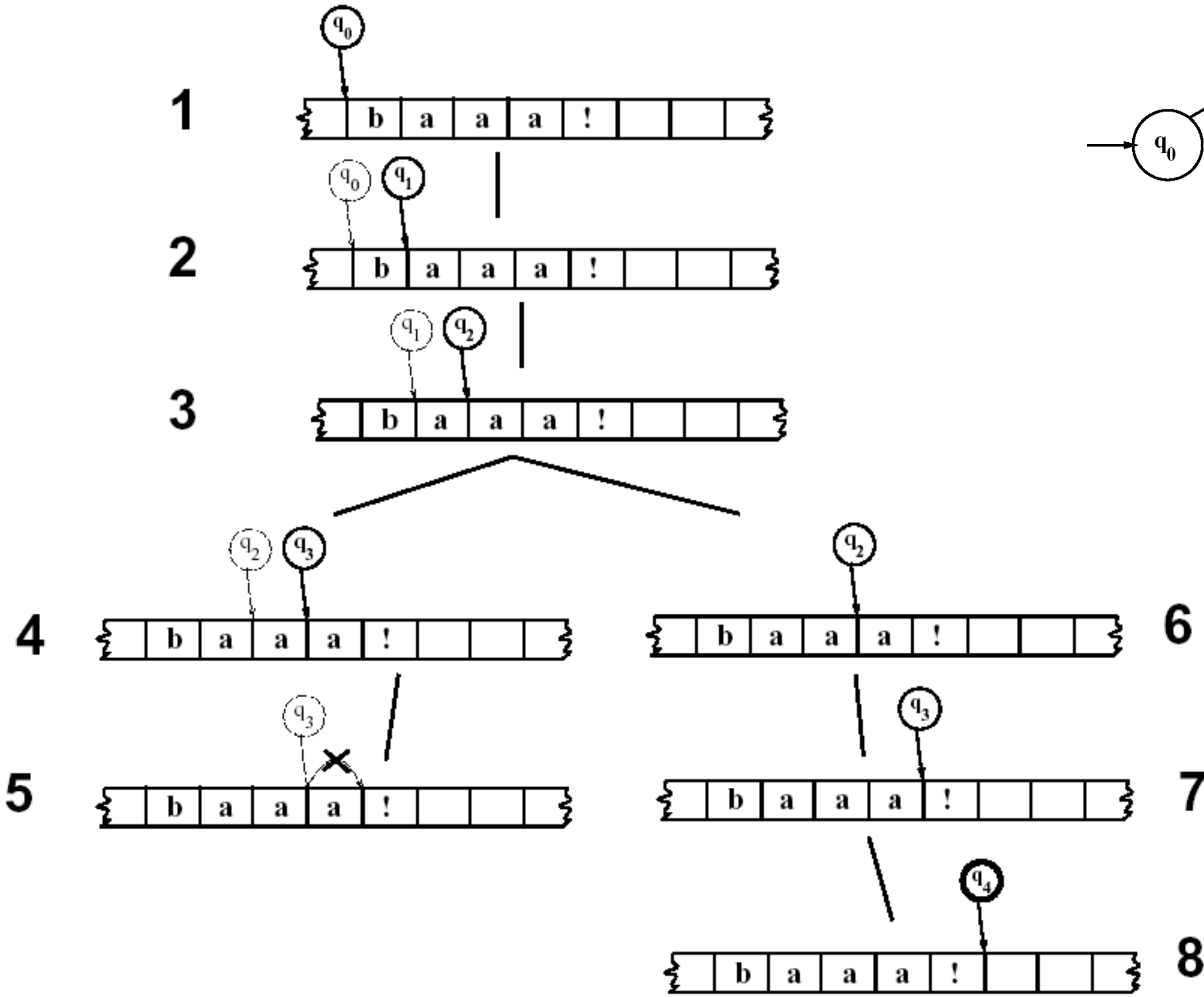
```
current-node ← the node the current search-state is in
index ← the point on the tape the current search-state is looking at
return a list of search states from transition table as follows:
  (transition-table[current-node,ε], index)
  ∪
  (transition-table[current-node, tape[index]], index + 1)
```

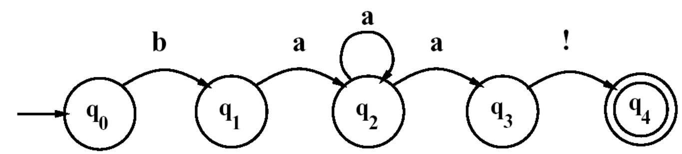
function ACCEPT-STATE?(*search-state*) **returns** true or false

```
current-node ← the node search-state is in
index ← the point on the tape search-state is looking at
if index is at the end of the tape and current-node is an accept state of machine
then
  return true
else
  return false
```

Finite-State Automata

Using an NFSA to Accept Strings



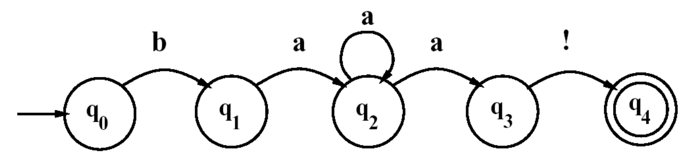


Finite-State Automata
Recognition as Search

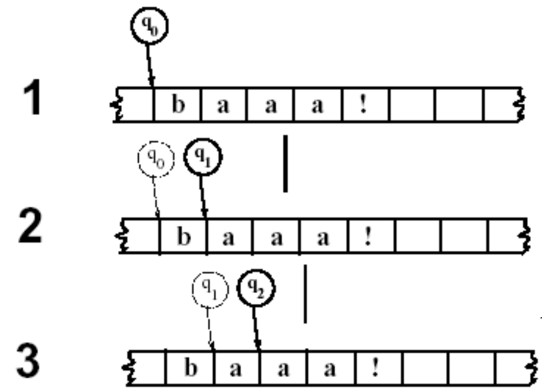
- Algorithms such as ND-RECOGNIZE are known as **state-space search**
- **Depth-first search** or **Last In First Out** (**LIFO**) strategy
- **Breadth-first search** or **First In First Out** (**FIFO**) strategy
- More complex search techniques such as **dynamic programming** or **A***

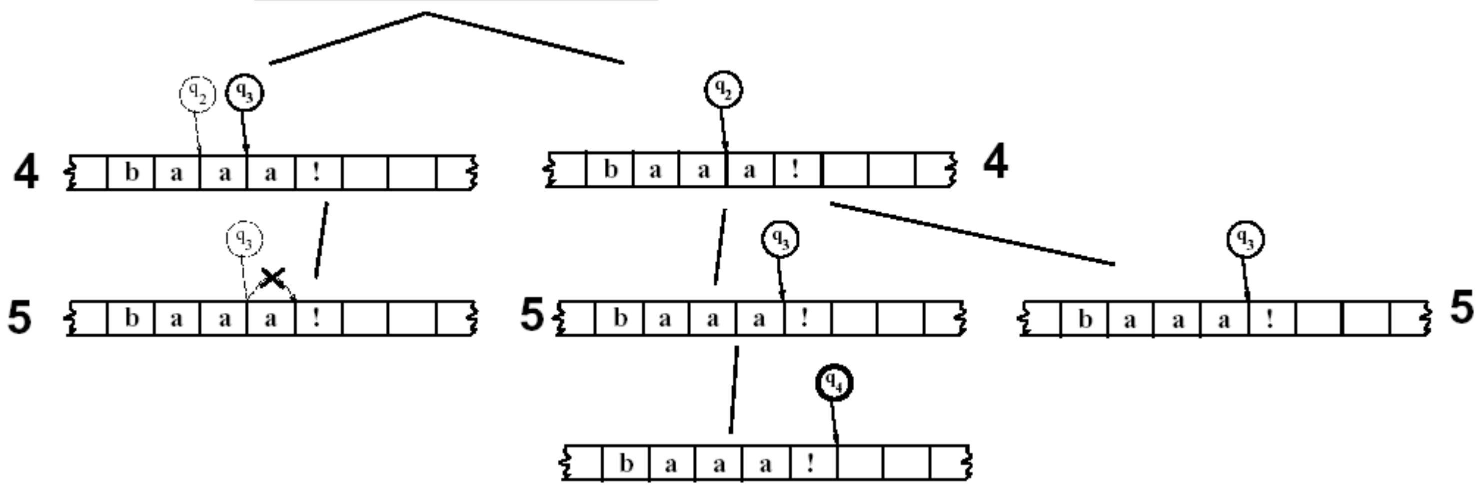
Finite-State Automata

Recognition as Search



A breadth-first trace of FSA #1 on some sheeptalk

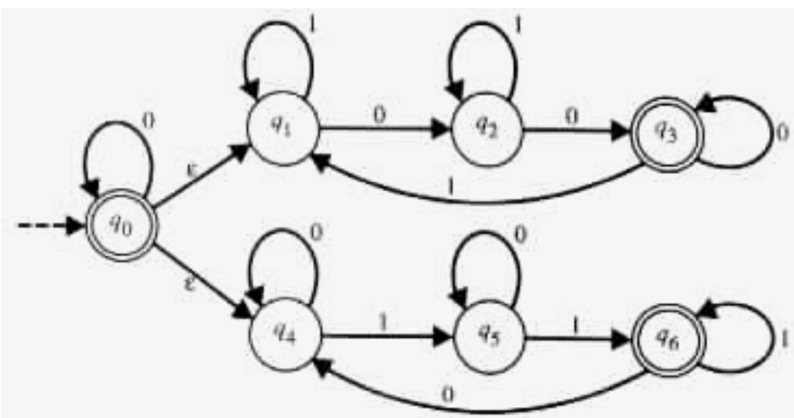


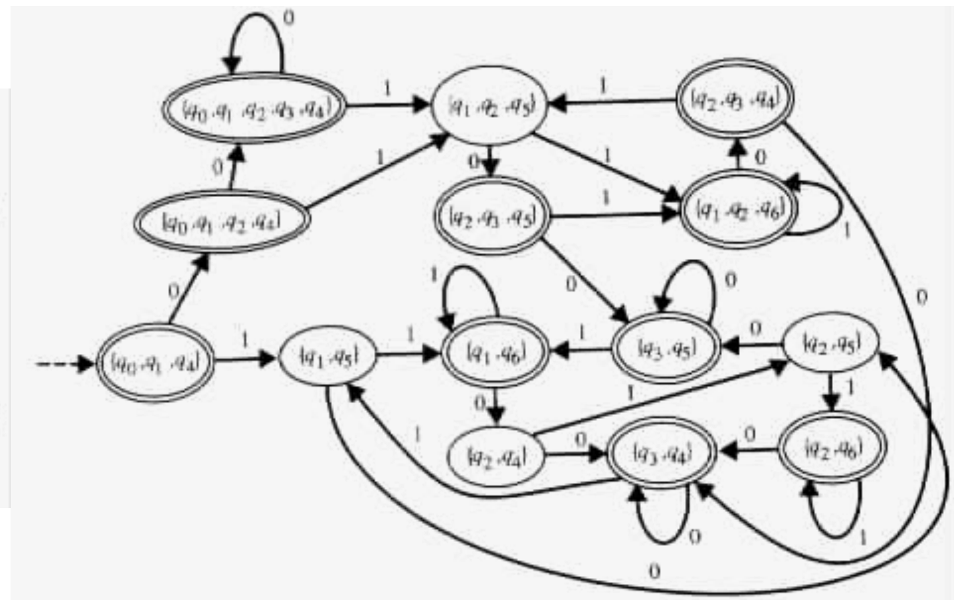


Relating DFSA and NFSA

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

NFSA → DFSA

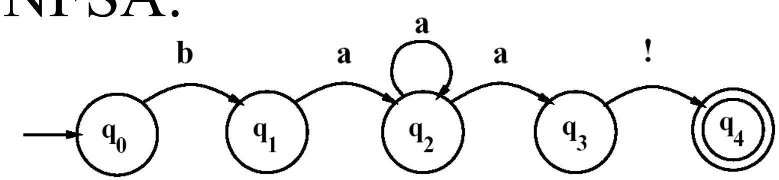




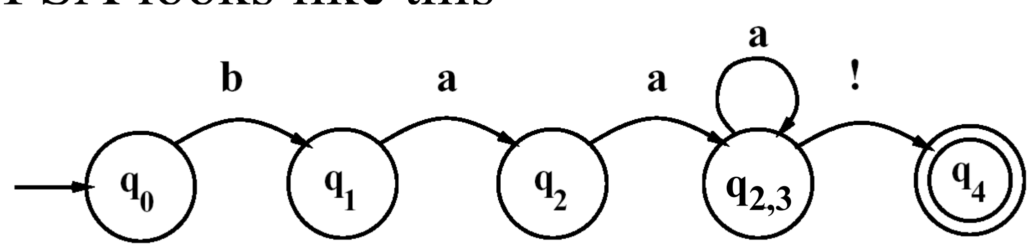
- *Eitan Gurari, Ohio State University*

Example of NFSA → DFSA

- Trace the states of the following NFSA:
  - q0 / b → q1
  - q1 /a → q2
  - q2 /a → q2,3 (an ambiguous state: q2 or q3)
  - q2,3 /a → q2,3 (here we trace the union of q2/a and q3/a)
  - q2,3 /! → q4 (again, trace the union of q2/! and q3/!)



- The DFSA states are q0, q1, q2, q2,3, q4
  - The DFSA looks like this



2.3 Regular Languages and FSAs

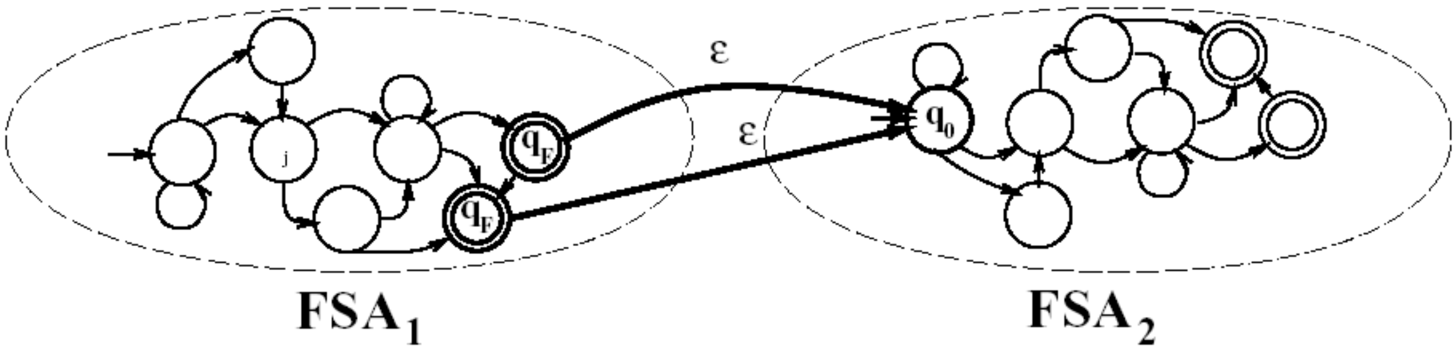
- The class of languages that are definable by regular expressions is exactly the same as the class of languages that are characterizable by FSA (D or ND).
  - These languages are called **regular languages**.
- The regular languages over Σ is formally defined as:
  1. ϕ is an RL
  2. $\forall a \in \Sigma$, $\{a\}$ is an RL
  3. If L_1 and L_2 are RLs, then so are:

     a) $L_1 \cdot L_2 = \{xy \mid x \in L_1 \text{ and } y \in L_2\}$, the **concatenation** of L_1 and L_2

     b) $L_1 \cup L_2$, the **union** of L_1 and L_2

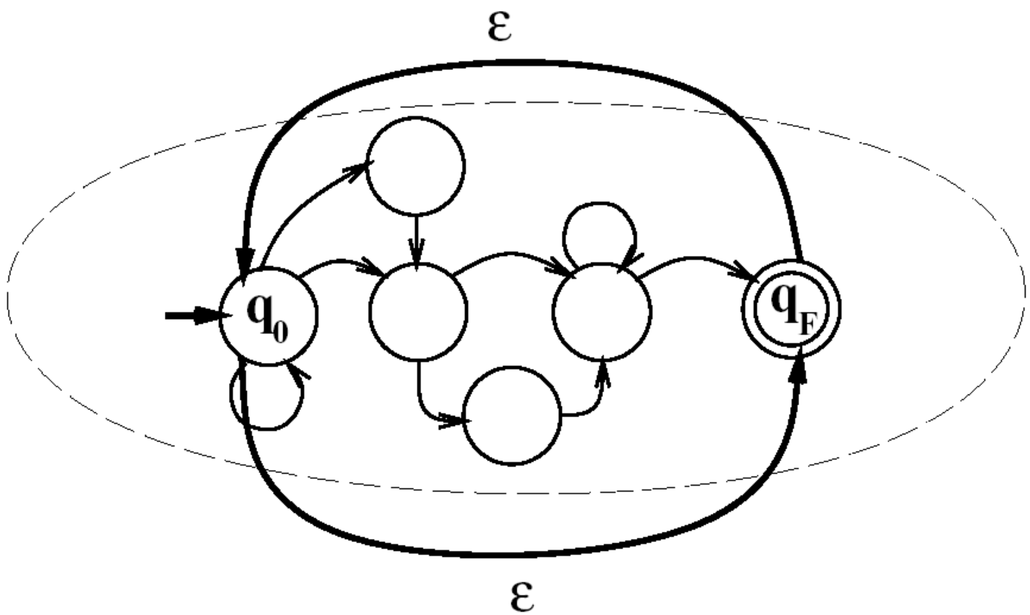
     c) $L_1{}^*$, the **Kleene closure** of L_1

2.3 Regular Languages and FSAs



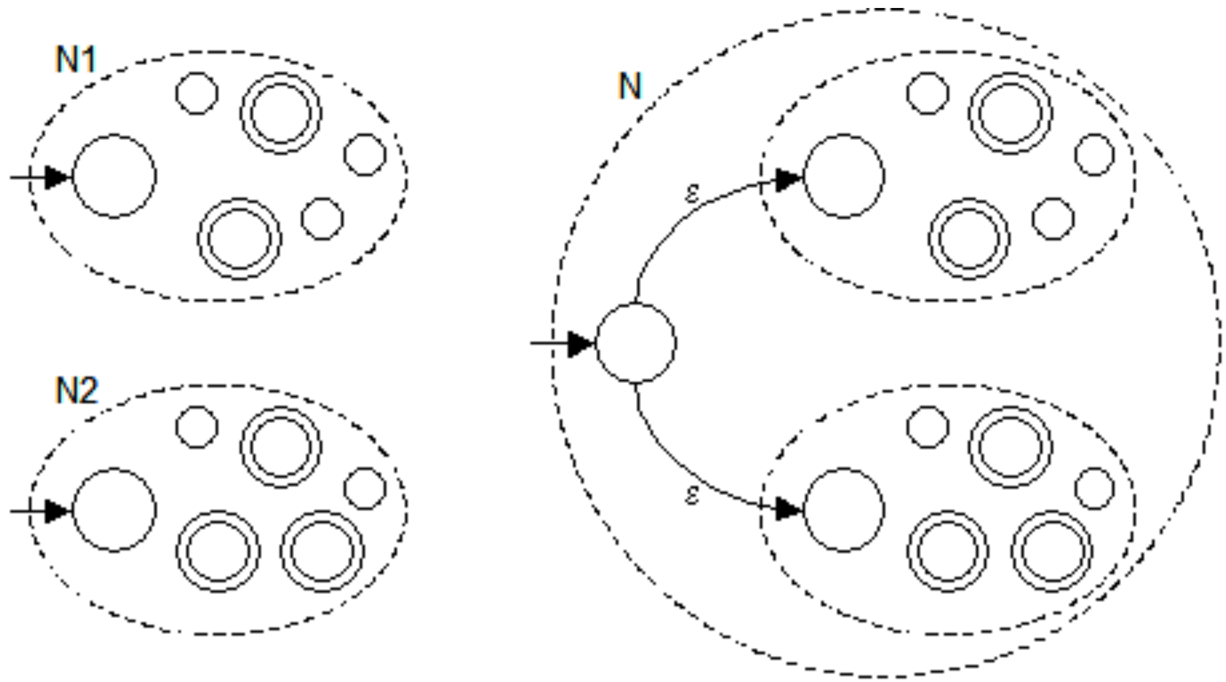
The concatenation of two FSAs

2.3 Regular Languages and FSAs



The closure (*Kleene* *) *of an FSAs*

2.3 Regular Languages and FSAs



The union (|) *of two FSAs*

http://csclab.murraystate.edu/bob.pilgrim/302/regular_languages.html

Next Time

- Read J+M Chapter 2 (2.3 to end)
  and J+M Chapter 3 (intro up to 3.2)