Natural Language Processing

Lecture 3

Today's Outline

Administration

Overview of Next 4 Lectures

Regular Expressions and Finite State Automata

Deterministic Recognition

Generative Grammars and Formal Languages

Web Page Changes

New office hours

- M 1:30-3 (214 MIB)
- Th 10-11:30 (741 LRDC)
- F 12-1 (741 LRDC)

Announcements

- Kathy McKeown in CS Lecture Series, October 19

Interesting links

- Eliza, Racter, and friends

Next 4 Lectures: Words

The next four lectures will begin our study of "words" and will cover three related topics

- finite state automata
- finite state transducers
- English morphology

Regular Expressions

Useful ways to view Regular Expressions

- as a practical way to specify textual search strings
- as a way to specify the design of a particular kind of machine

As we will see, these are equivalent

Regular Expressions: Text Searches

Everybody does it

- Perl, emacs, Word, sed, awk, grep, netscape, etc.

Basic regular expression patterns (Perl notation) - J&M pages 23-27

- sequence (slashes)
- disjunction (brackets)
- range (brackets plus dash)
- negation (open bracket plus caret)
- optionality (question-mark)
- Kleene star (asterisk)
- wildcard (period)
- anchors (caret, dollar-sign, \b, \B)

Brackets and Character Disjunction

RE	Match	Example Patterns
`/[wW]oodchuck/`	Woodchuck or woodchuck	"<u>Woodchuck</u>"
`/[abc]/`	'a', 'b', *or* 'c'	"In uomini, in sold<u>a</u>ti"
`/[1234567890]/`	any digit	"plenty of <u>7</u> to 5"

A Simple Example

Finding instances of the determiner "the" (using a pattern, over a corpus)

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

The second RE catches "the" when it begins a sentence, while the third RE does not match "the" when embedded in another word such as "other"

The Two Kinds of Errors

The process we just went through was based on fixing errors in the regular expression

- errors where some of the instances were missed (judged to be not instances when they should have been)
- errors where the instances were included (when they should not have been)

This is pretty much going to be the story of the rest of the course

REs: Text Searches (continued)

Disjunction, grouping, and precedence - J&M pages 27-28

Operators are greedy

Advanced operators - J&M pages 30-31

Substitution and memory - J&M pages 31-33

ELIZA: Substitutions Using Memory

User: Men are all alike.

s/.* all .*/IN WHAT WAY/

ELIZA: IN WHAT WAY

User: They're always bugging us about something or other.

s/.* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE/

ELIZA: CAN YOU THINK OF A SPECIFIC EXAMPLE

User: My boyfriend says I'm depressed.

s/.* I'm (depressed|sad) .*/I AM SORRY TO HEAR YOU ARE \1/

ELIZA: I AM SORRY TO HEAR YOU ARE DEPRESSED

Finite State Automata

Regular Expressions (REs) can be viewed as a way to describe machines called Finite State Automata (FSA, also known as automata, finite automata).

FSAs and their close variants are a theoretical foundation of much of the field of NLP.

FSAs as Directed Graphs

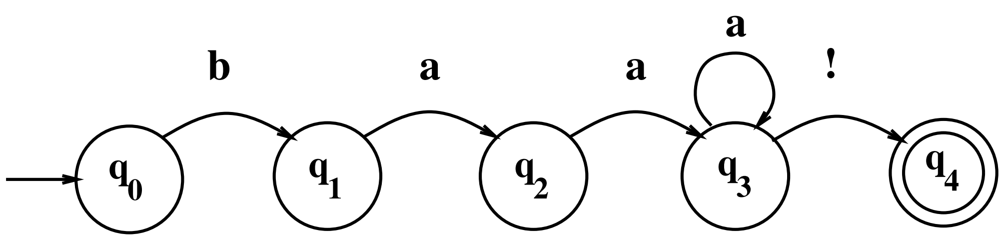
A sheep language

- baa!
- baaa!
- ...

A regular expression for the sheep language

- /baa+!/

A graphical view of the sheep language

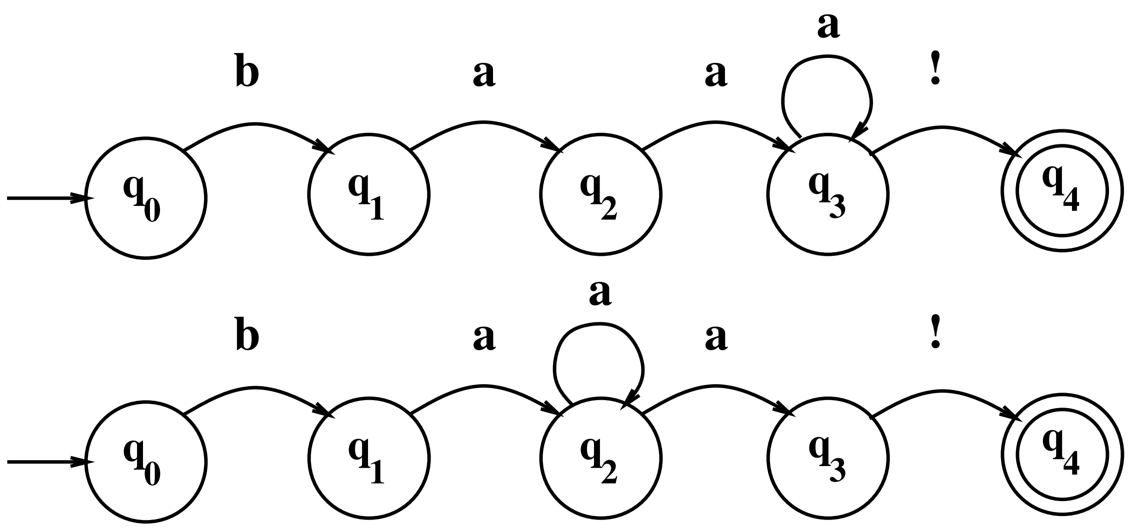


- five states (nodes in the graph)
- q0 is the start state (incoming arrow)
- q4 is the final state (double circle)
- five transitions (arcs in the graph)
- deterministic

Alternatives for our Sheep Language

/baa+!/

/baaa*!/



More on this later...

A More Formal View

FSAs can be formally specied as a 5-tuple

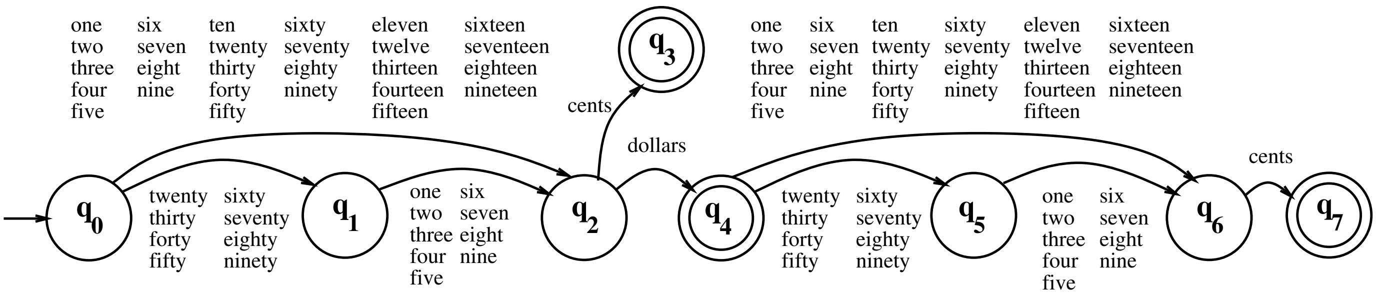
- a finite set of N states $q_0, q_1 \ldots q_n$: Q
- a finite input alphabet of symbols: Σ
- the start state: q_0
- the set of final states: F $\subseteq$ Q
- the transition function that maps Q $\times$ Σ to Q

A Note on Alphabets

You shouldn't view the term alphabet too narrowly. In particular, you don't have to limit it to letters. Any kind of symbol will do.

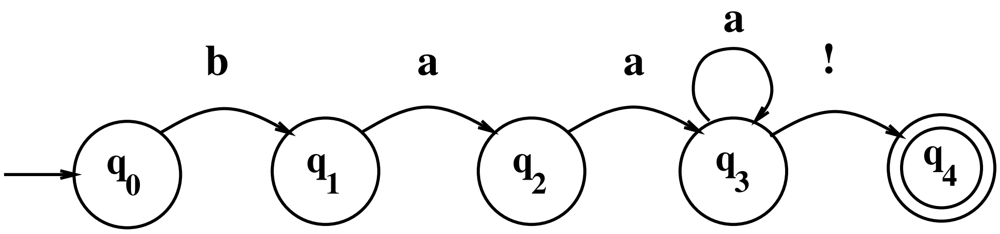
Furthermore, those symbols can stand for objects that themselves have internal structure.

Simple Dollars and Cents Example



- alphabet of words
- accepts phrases like *ten cents, three dollars, twenty dollars thirty five cents*
- but...
  - accepts *one dollars*
  - doesn't accept *hundred dollars*

Yet Another Representation



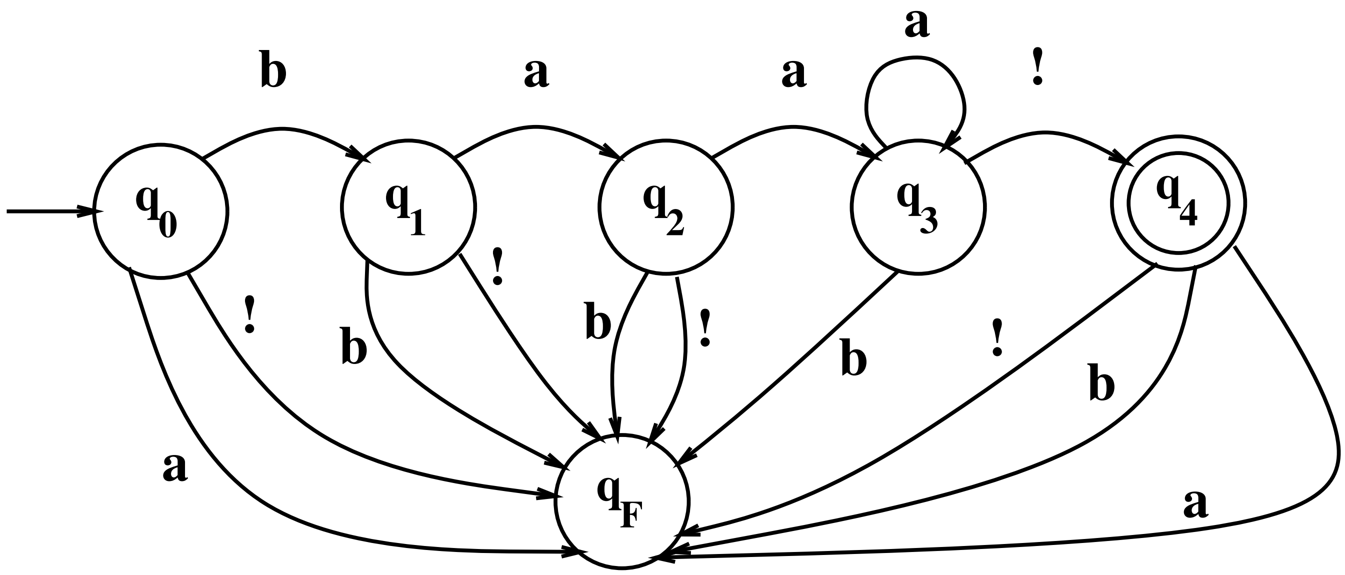
FSAs can also be encoded as *State Transition Tables*

- Q $= \{q_0, q_1, q_2, q_3, q_4\}$
- $\Sigma = \{$a,b,!$\}$
- F=$\{q_4\}$
- transition function =

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

Adding a Failing State

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

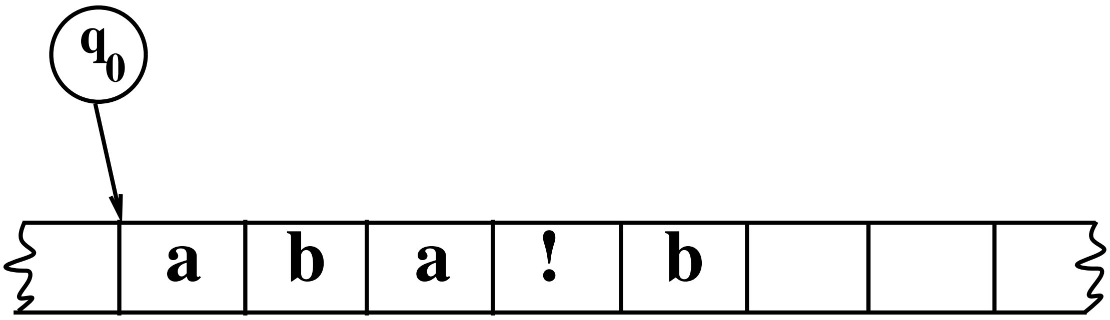


Recognition

Recognition (or acceptance) is the process of determining whether or not a given input should be accepted by a given machine

In terms of REs, its the process of determining whether or not a given input matches a particular regular expression.

Traditionally , recognition is viewed as processing an input written on a tape consisting of cells containing elements from the alphabet



D-Recognize

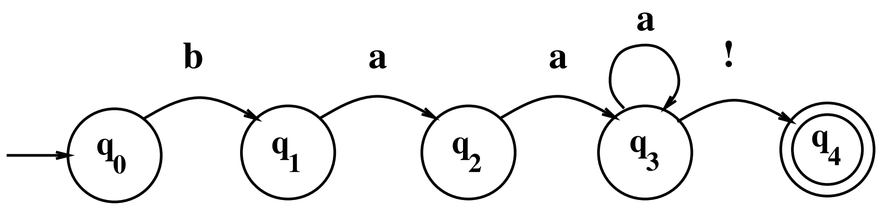
```
function D-RECOGNIZE(tape, machine) returns accept or reject

  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
```

D-Recognize (continued)

Key points about d-recognize

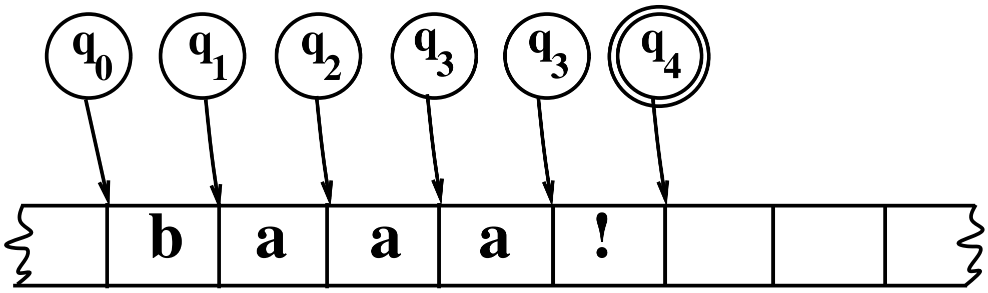
- Deterministic means that the code always knows what to do at each point in the process.
- d-recognize is essentially a crude table-driven interpreter.
- the recognition code is universal for all FSAs. To change to a new formal language, you merely change the alphabet and the table. The recognition code stays the same.
- thus, searching for a string using a RE involves compiling the RE into a table and passing the table to an interpreter.

An Example Trace

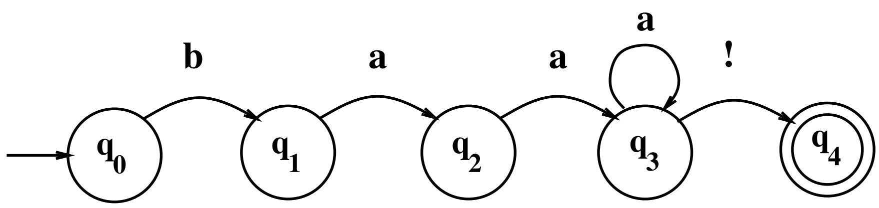


function D-RECOGNIZE(*tape, machine*) **returns** accept or reject

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



Another Example Trace



What about abc?

Formal Languages

A <u>Formal Language</u> is a set of strings composed of symbols from a finite set of symbols (the alphabet).

FSAs (and regular expressions) define formal languages (without having to explicitly enumerate the set).

Given a machine m (such as a particular FSA) L(m) means the formal language characterized by m.

- L(sheeptalk FSA) = {baa!, baaa!, baaaa!, ...} (an infinite set)

Generative Grammars

The term <u>Generative</u> refers to the idea that FSAs can be viewed as generators of formal languages as well as acceptors. This dual view will pop up again and again.

To generate, you traverse the machine and write transition symbols on the tape rather than reading them.

For Next Time

Finish Chapter 2 (page 40 on)

Acknowledgements

Some of this material was taken from the course web pages of James Martin and Drago Radev.