Regular Expressions and Automata

Outline

- Regular expression– read the book
- How to implement regular expression?
- Finite-state-automata (FSA)
  - Deterministic FSA
  - Non-deterministic FSA

Regular Expressions and Text Searching

- Everybody does it
  - Emacs, vi, grep, etc..
  - Programming language: perl, C#, java....
- Regular expressions are a compact textual representation of a set of strings representing a language
  - In the simplest case, regular expressions describe regular languages

Basic regular expression

RE	Example Patterns Matched
`/woodchucks/`	"interesting links to woodchucks and lemurs"
`/a/`	"Mary Ann stopped by Mona's"
`/Claire␣says,/`	" "Dagmar, my gift please," Claire says,"
`/DOROTHY/`	"SURRENDER DOROTHY"
`/!/`	"You've left the burglar behind again!" said Nori

Regular Expression: simple patterns

```
 $_ = "yabba dabba";
#pattern match return a true or false
  if($_ =~ /abba/)
  {
      print "It matched!\n"
   }
```

A bit more regular expression

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

RE	Match	Example Patterns Matched
/[A-Z]/	an upper case letter	"we should call it 'Drenched Blossoms' "
/[a-z]/	a lower case letter	"my beans were impatient to be hoed!"
/[0-9]/	a single digit	"Chapter 1: Down the Rabbit Hole"

/\bdog\b/	\b: A word boundary	The doggie plays in the yard I lick the dog twice

A bit more more

- Dot (.): matches any single character(but not \n)
  - /3.14159/ cf. /3\.14159/ matches 3.14159 only
- Star (*): matches zero or more of preceding item
  - /fred*/ matches fre, fred, fredddd.
- Plus(+): matches one or more of preceding item
  - /fred+/ matches fred,fredddd but not fre
- Questionmark (?) matches zero or one of preceding item

RE	Match	Example Patterns Matched
`woodchucks?`	woodchuck or woodchucks	"woodchuck"
`colou?r`	color or colour	"colour"

- Ambiguous: /fred*/ String: fredddfff
- Always match the largest string they can

- More …

Example

- Find all the instances of the word "the" in a text.
  - `/the/`
    - The the
  - `/[tT]he/`
    - They they
  - `/\b[tT]he\b/`
    - state-of-the-art

Errors

- The process we just went through was based on two fixing kinds of errors
  - Matching strings that we should not have matched (there, then, other)
    - False positives
  - Not matching things that we should have matched (The)
    - False negatives

Exercise

- Task is to match “the”
- 5 words: “the they theu The teo”
- A solution: `/the/`
- If the matches are “<u>the</u> <u>the</u>y <u>the</u>u The teo”
- What is the false positives?
- What is the false negatives?

Errors

- Reducing the error rate for an application often involves two efforts (often antagonistic):
  - Increasing accuracy, or precision, (minimizing false positives)
  - Increasing coverage, or recall, (minimizing false negatives).
    - "They The the they"
    - /the/ (They The the they) → `/[tT]he/` (They The the they)
- We'll be telling the same story for many tasks, all semester

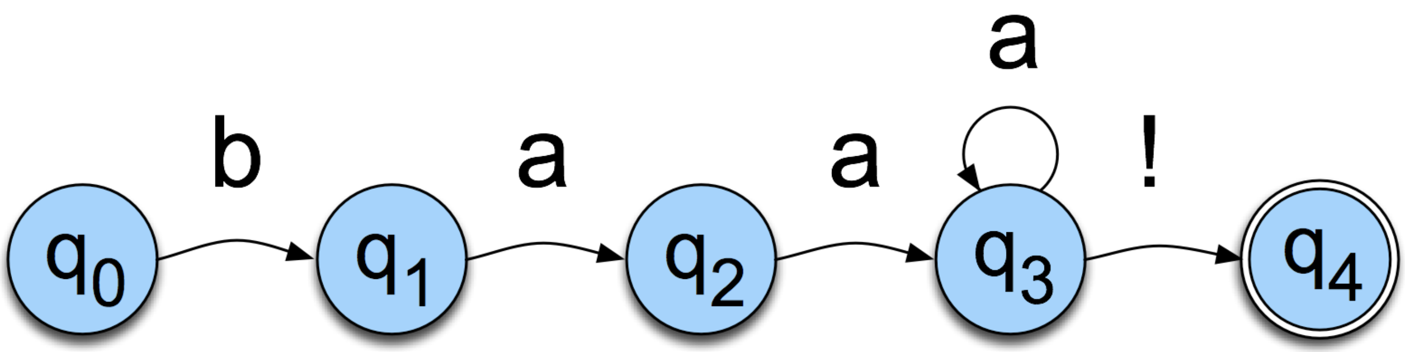
Finite state automata (FSA)

- Regular expressions
  - Compact textual strings (e.g. "/[tT]he/")
    - Perfect for specifying patterns in programs or command-lines
  - Can be implemented as a FSA
- Finite state automata
  - Graphs
  - Can be described with a regular expression (a textual way of specifying the structure of FSA)
  - FSA has a wide range of uses

FSAs as Graphs

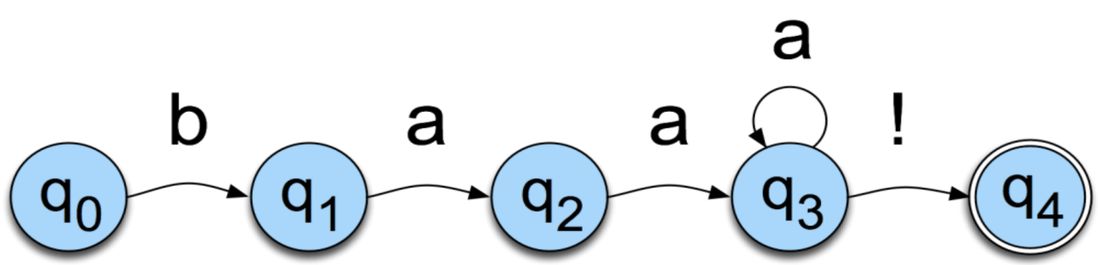
- Let's start with the sheep language from the text
  - /baa+!/

baa! baaa!
baaaa! ...



Sheep FSA

- We can say the following things about this machine
  - It has 5 states
  - b, a, and ! are in its alphabet
  - q_0 is the start state
  - q_4 is an accept state
  - It has 5 transitions



More Formally

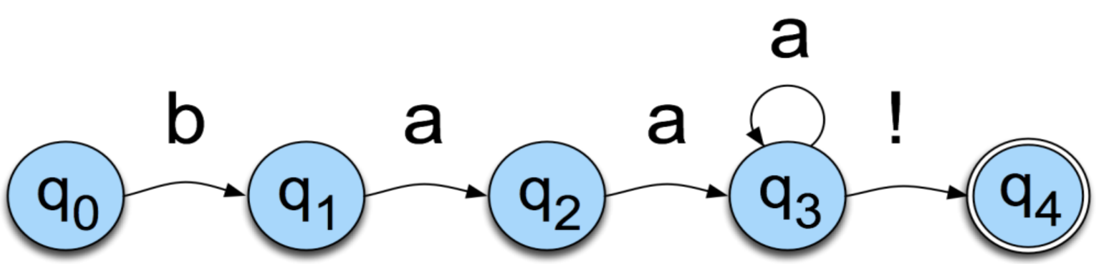
- You can specify an FSA by enumerating the following things.
  - The set of states: Q
  - A finite alphabet: Σ
  - A start state
  - A set of accept/final states
  - A transition function that maps Qx Σ to Q
- Don't take term *alphabet* word too narrowly; it just means we need a finite set of symbols in the input.

Yet Another View

- an FSA can ultimately be represented as tables

If you're in state 1 and you're looking at an a, go to state 2

	b	a	!	e
0	1			
1		2		
2		3		
3		3	4	
4				

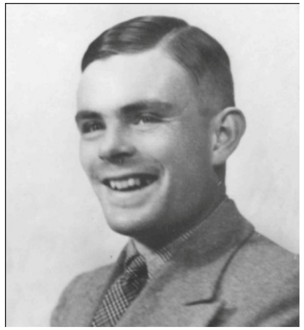


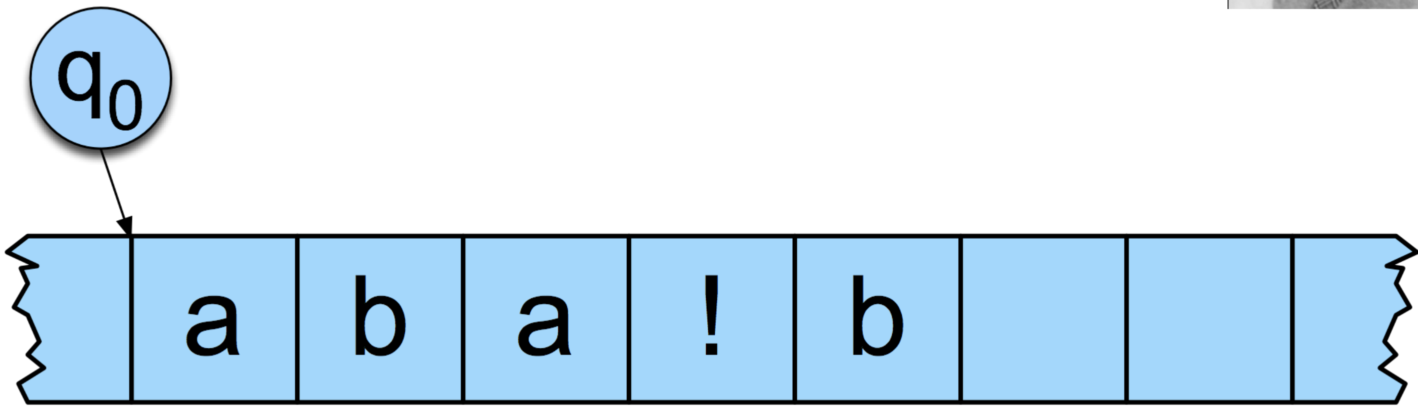
Recognition

- Recognition is the process of determining if a string should be accepted by a machine
- Or… it's the process of determining if a string is in the language we're defining with the machine
- Or… it's the process of determining if a regular expression matches a string

Recognition

- Traditionally, (Turing's notion) this process is depicted with a tape.

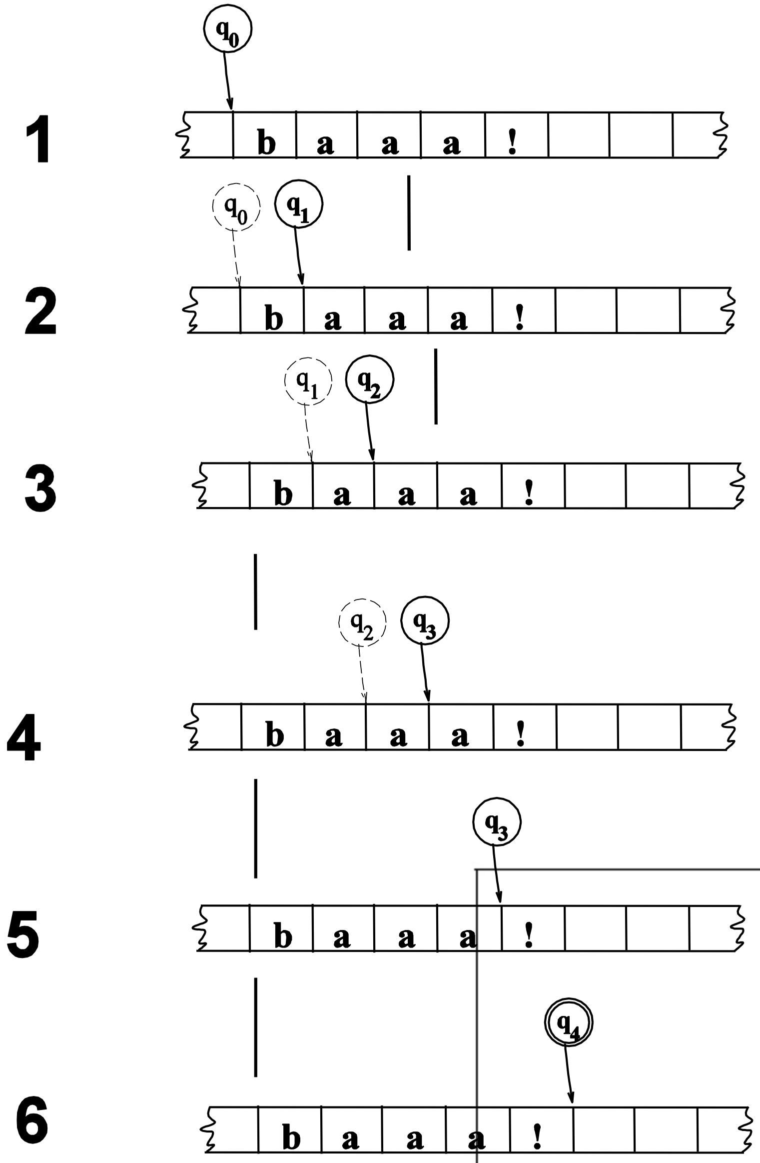


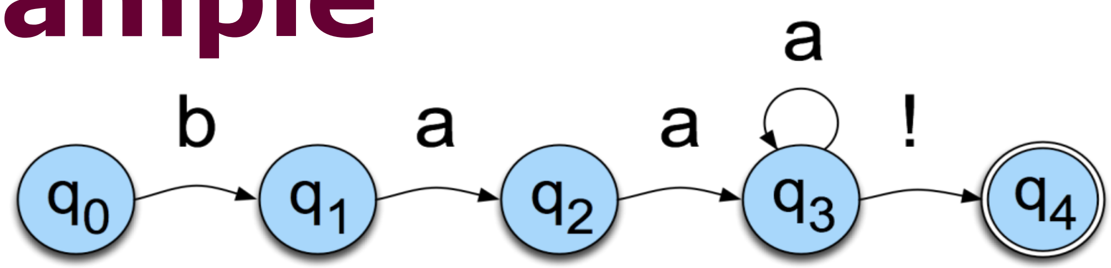


Recognition (D-Recognize)

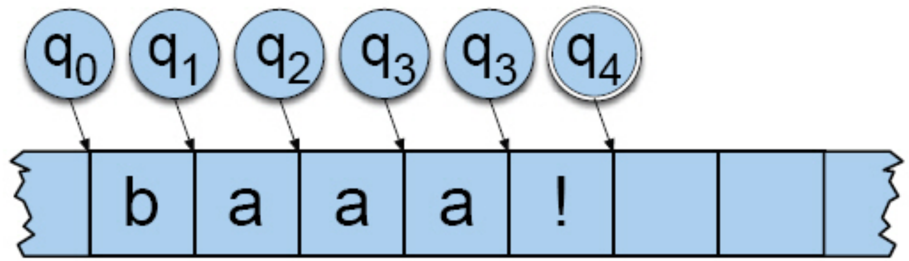
- Simply a process of starting in the start state
- Examining the current input
- Consulting the table
- Going to a new state and updating the tape pointer.
- Until you run out of tape.
- ??

Example





	b	a	!	e
0	1			
1		2		
2		3		
3		3	4	
4				



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
```

Key Points

- Deterministic means that 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 regular languages.
  - To change the machine, you simply change the table.

Key Points

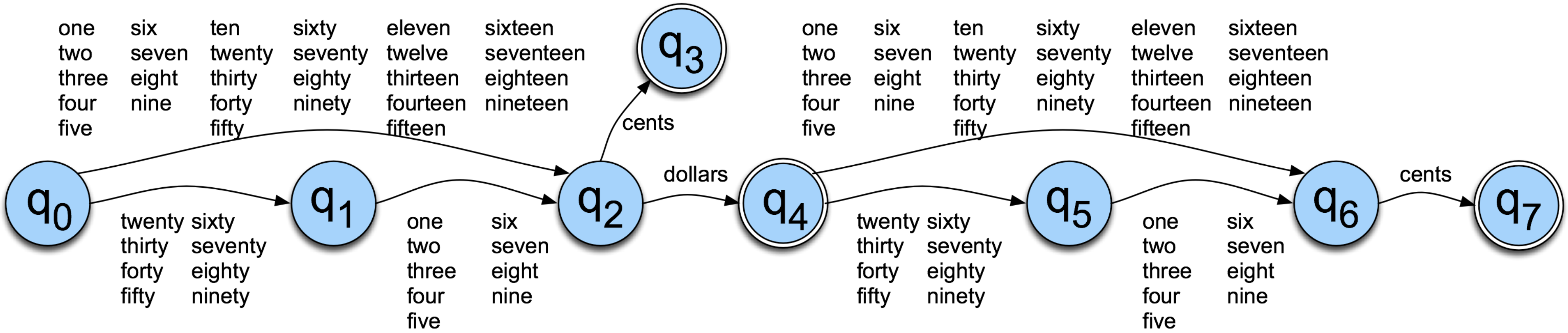
- matching strings with regular expressions (ala Perl, grep, etc.) is a matter of
  - translating the regular expression into a machine (a table) and
  - passing the table and the string to an interpreter that implement D-recognize

A short summary

- Regular expression
  - Very basic one /the/
  - More notations [], ?, *, .
- Two types of errors
  - false positives
  - false negatives
- Finite state automata
  - Representation
  - D-recognize algorithm to implement regular expression

Dollars and Cents

- Don't take term *alphabet* word too narrowly; it just means we need a finite set of symbols in the input.



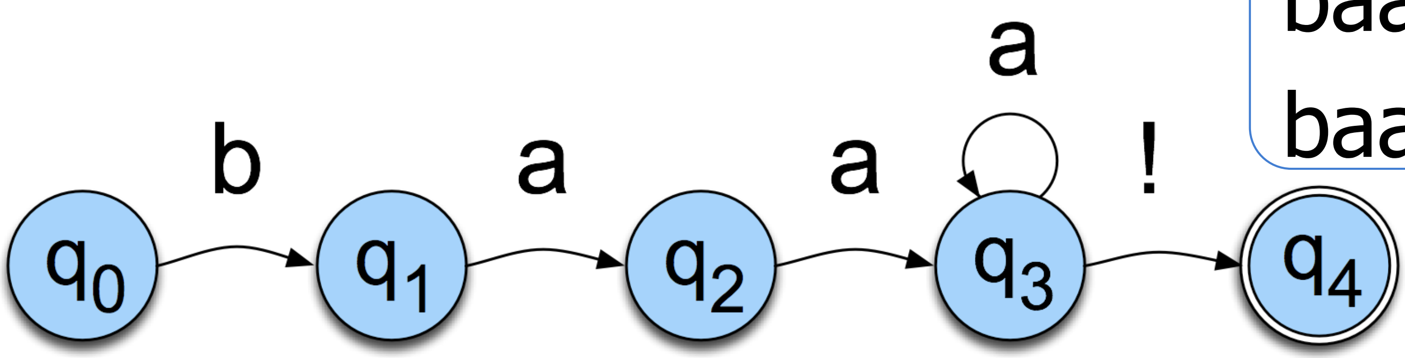
Exercise

- How to represent the words for English numbers 1-99 in FSA?

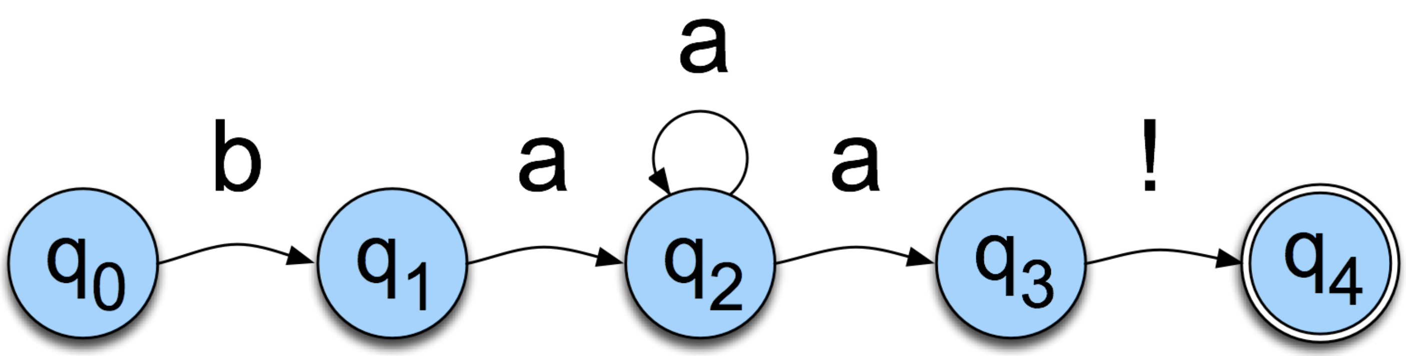
Non-Deterministic (NFSA)

- the sheep language from the text
  - /baa+!/

baa! baaa!
baaaa! ...

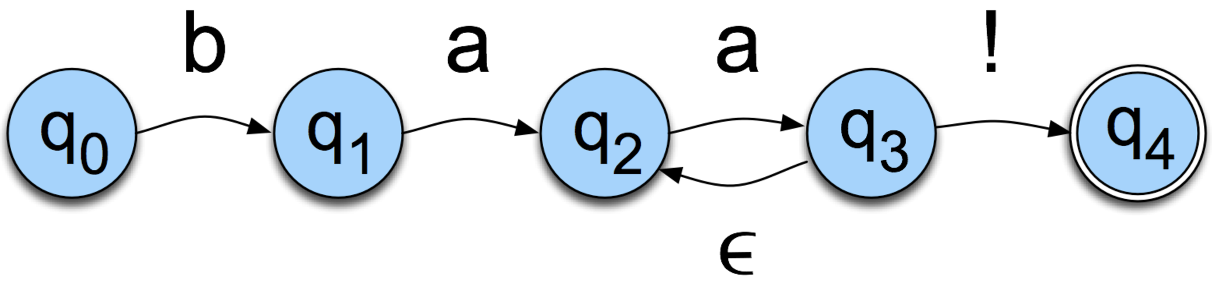


- There are other machines that correspond to this same language



Non-Deterministic cont.

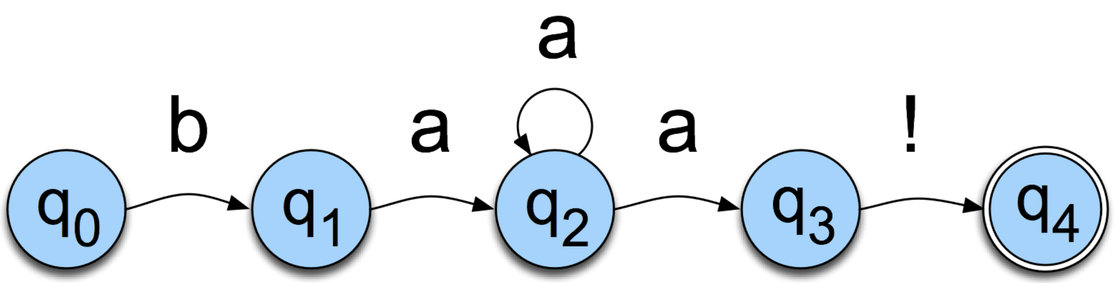
- Yet another technique
  - Epsilon transitions
  - Key point: these transitions do not examine or advance the tape during recognition



Equivalence

- Non-deterministic machines can be converted to deterministic ones with a fairly simple construction
- That means that they have the same power;
  - non-deterministic machines are not more powerful than deterministic ones in terms of the languages they can and can't characterize

Yet Another View for another



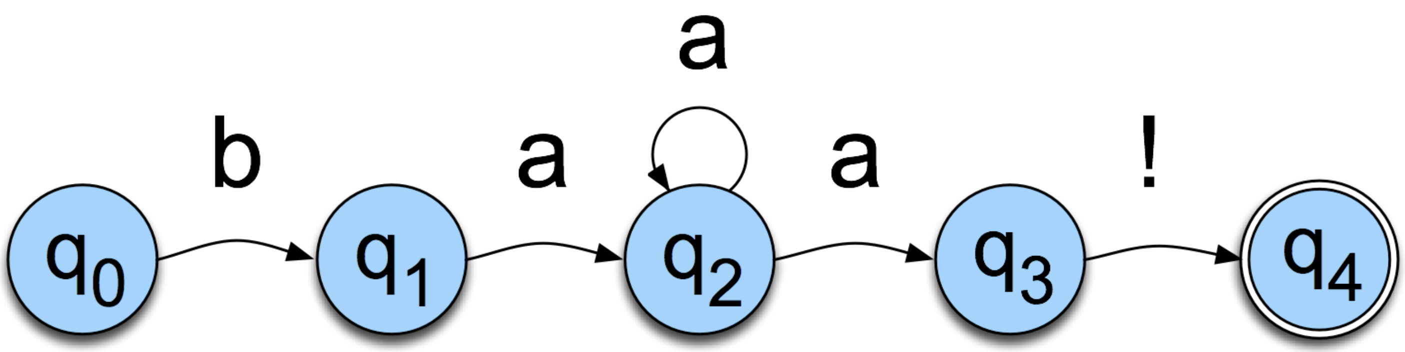
	b	a	!	e
0	1			
1		2		
2		2,3		
3			4	
4				

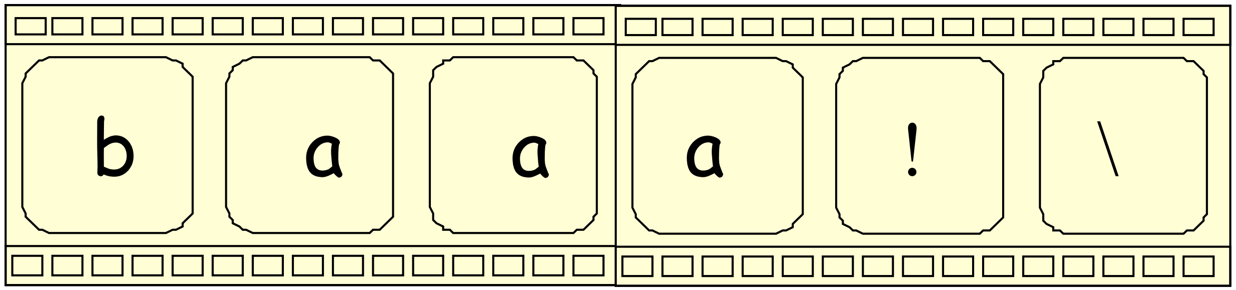
If you're in state 2 and you're looking at an a, go to state 2 or 3

ND Recognition

- Two basic approaches (used in all major implementations of regular expressions)
  1. Either take a ND machine and convert it to a D machine and then do recognition with that.
  2. Or explicitly manage the process of recognition as a state-space search (leaving the machine as is).

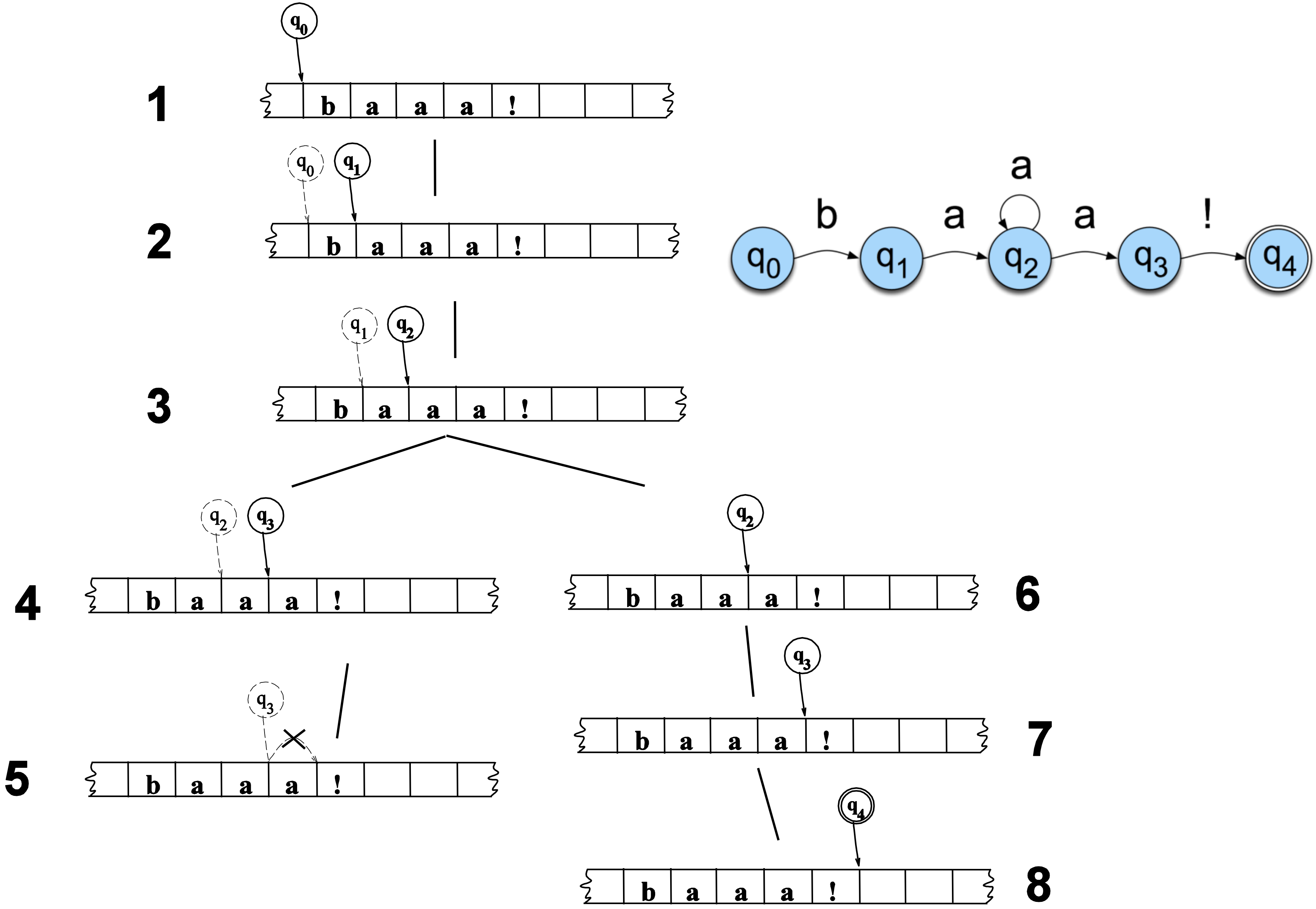
Example





q_0 q_1 q_2 q_2 q_3 q_4

Example


1
2
3
4
5
6
7
8
b a a a !
b a a a !
b a a a !
b a a a !
b a a a !
b a a a !
b a a a !
b a a a !
q0
q1
q2
q3
q4
b
a
a
a
!

Key Points

- States in the search space are pairings of tape positions and states in the machine.
- By keeping track of as yet unexplored states, a recognizer can systematically explore all the paths through the machine given an input.

Non-Deterministic Recognition: Search

- In a ND FSA there exists at least one path through the machine for a string that is in the language defined by the machine.
- But not all paths directed through the machine for an accept string lead to an accept state.
- No paths through the machine lead to an accept state for a string not in the language.

Non-Deterministic Recognition

- So success in non-deterministic recognition occurs when a path is found through the machine that ends in an accept.
- Failure occurs when all of the possible paths for a given string lead to failure.

Why Bother?

- Non-determinism doesn't get us more formal power and it causes headaches so why bother?
  - More natural (understandable) solutions
  - Not always obvious to users whether or not the regex that they've produced is non-deterministic or not
    - Better to not make them worry about it

A summary

- Regular expression
  - Very basic one /the/
  - More notations [], ?, *, .
- Two types of errors
- Finite state automata
  - Deterministic (NFSA)
    - D-recognize algorithm to implement regular expression
  - Non-Deterministic (NFSA)
    - Two approaches to implementing regular expression

Readings: Quick Introduction to Regular Expressions in Java

- Java.util.regex API

http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/package-summary.html

- Java regexps tutorial

http://docs.oracle.com/javase/tutorial/essential/regex/