Regular Expressions and Finite State Automata

Mausam

(Based on slides by Jurafsky & Martin,
Julia Hirschberg)

Regular Expressions and Text Searching

- Everybody does it
  - Emacs, vi, perl, grep, etc..
- Regular expressions are a compact textual representation of a set of strings representing a language.

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 Expressions

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"

Regular Expressions

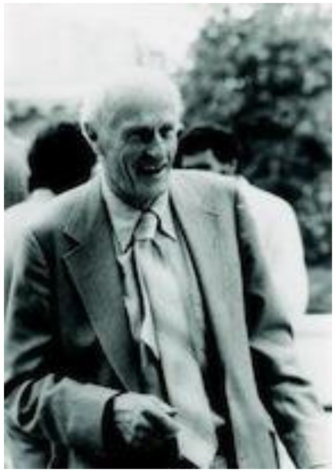
RE	Match	Example Patterns Matched
`/[A-Z]/`	an upper case letter	"we should call it 'Drenched Blossoms' "
`/[a-z]/`	a lower case letter	"my beans were impatient to be hoed!"
`/[0-9]/`	a single digit	"Chapter 1: Down the Rabbit Hole"

Regular Expressions

RE	Match (single characters)	Example Patterns Matched
`[^A-Z]`	not an upper case letter	"Oyfn pripetchik"
`[^Ss]`	neither 'S' nor 's'	"I have no exquisite reason for't"
`[^\.]`	not a period	"our resident Djinn"
`[e^]`	either 'e' or '^'	"look up ^ now"
`a^b`	the pattern 'a^b'	"look up a^ b now"

Regular Expressions: ? * + .

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		begin begun begun beg3n



Stephen C Kleene

Kleene *, Kleene

Regular Expressions: Anchors ^ $

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

Regular Expressions

RE	Expansion	Match	Examples
\d	[0-9]	any digit	Party␣of␣5
\D	[^0-9]	any non-digit	Blue␣moon
\w	[a-zA-Z0-9_]	any alphanumeric/underscore	Daiyu
\W	[^\w]	a non-alphanumeric	!!!!
\s	[␣\r\t\n\f]	whitespace (space, tab)	
\S	[^\s]	Non-whitespace	in␣Concord

Regular Expressions

RE	Match	Example Patterns Matched
`\*`	an asterisk "*"	"K*A*P*L*A*N"
`\.`	a period "."	"Dr. Livingston, I presume"
`\?`	a question mark	"Why don't they come and lend a hand?"
`\n`	a newline	
`\t`	a tab	

Example

- Find all the instances of the word "the" in a text.
  - /the/
  - /[tT]he/
  - /\b[tT]he\b/
  - [^a-zA-Z][tT]he[^a-zA-Z]
  - (^|[^a-zA-Z])[tT]he($|[^a-zA-Z])

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 (Type I)
  - Not matching things that we should have matched (The)
    - False negatives (Type II)

Errors

- We'll be telling the same story for many tasks, all semester. Reducing the error rate for an application often involves two antagonistic efforts:
  - Increasing accuracy, or precision, (minimizing false positives)
  - Increasing coverage, or recall, (minimizing false negatives).

Precision & Recall

	Predicted	
Actual	"P"	"N"
P	TP	FN
N	FP	TN

Precision = TP/(TP+FP)
Recall = TP/(TP+FN)
F-measure = 2pr/(p+r)

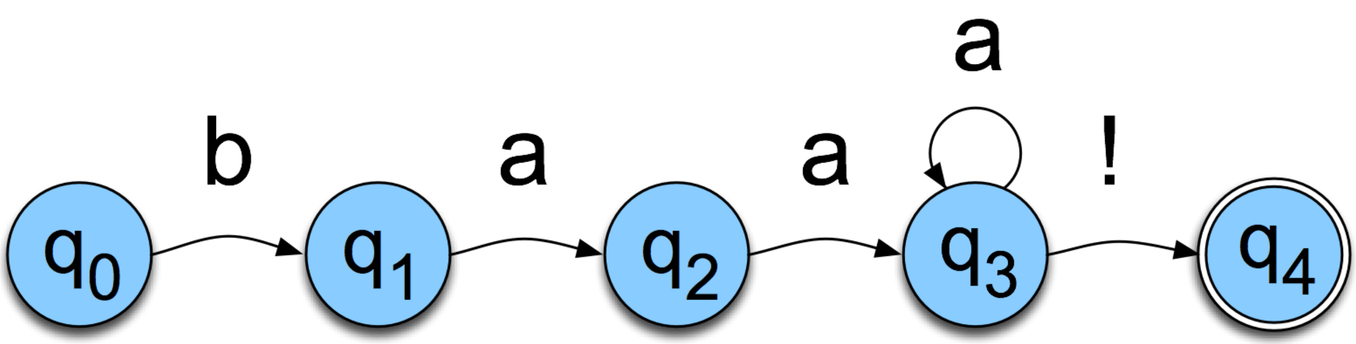
$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}.$$

Finite State Automata

- Regular expressions can be viewed as a textual way of specifying the structure of finite-state automata.

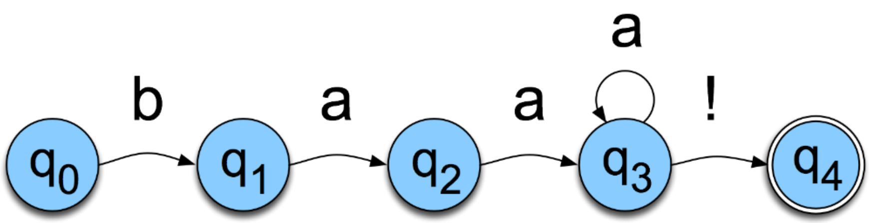
FSAs as Graphs

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



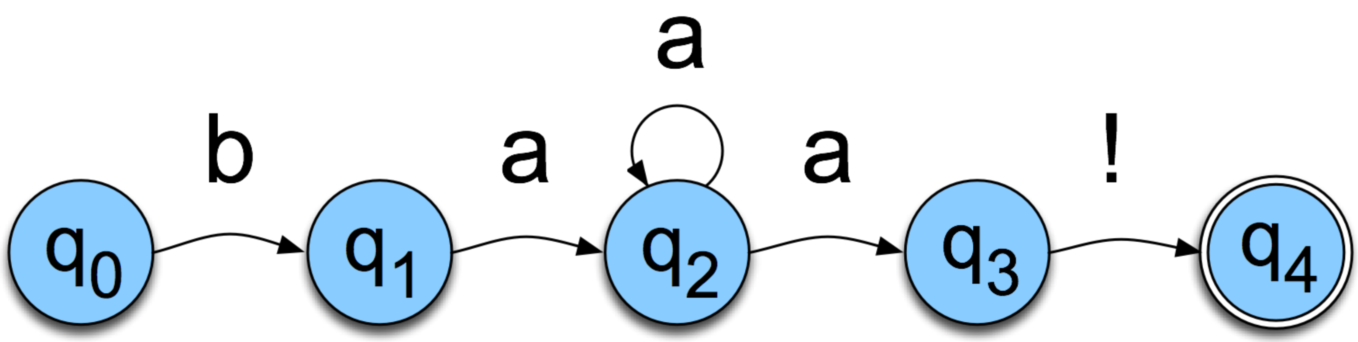
Sheep FSA

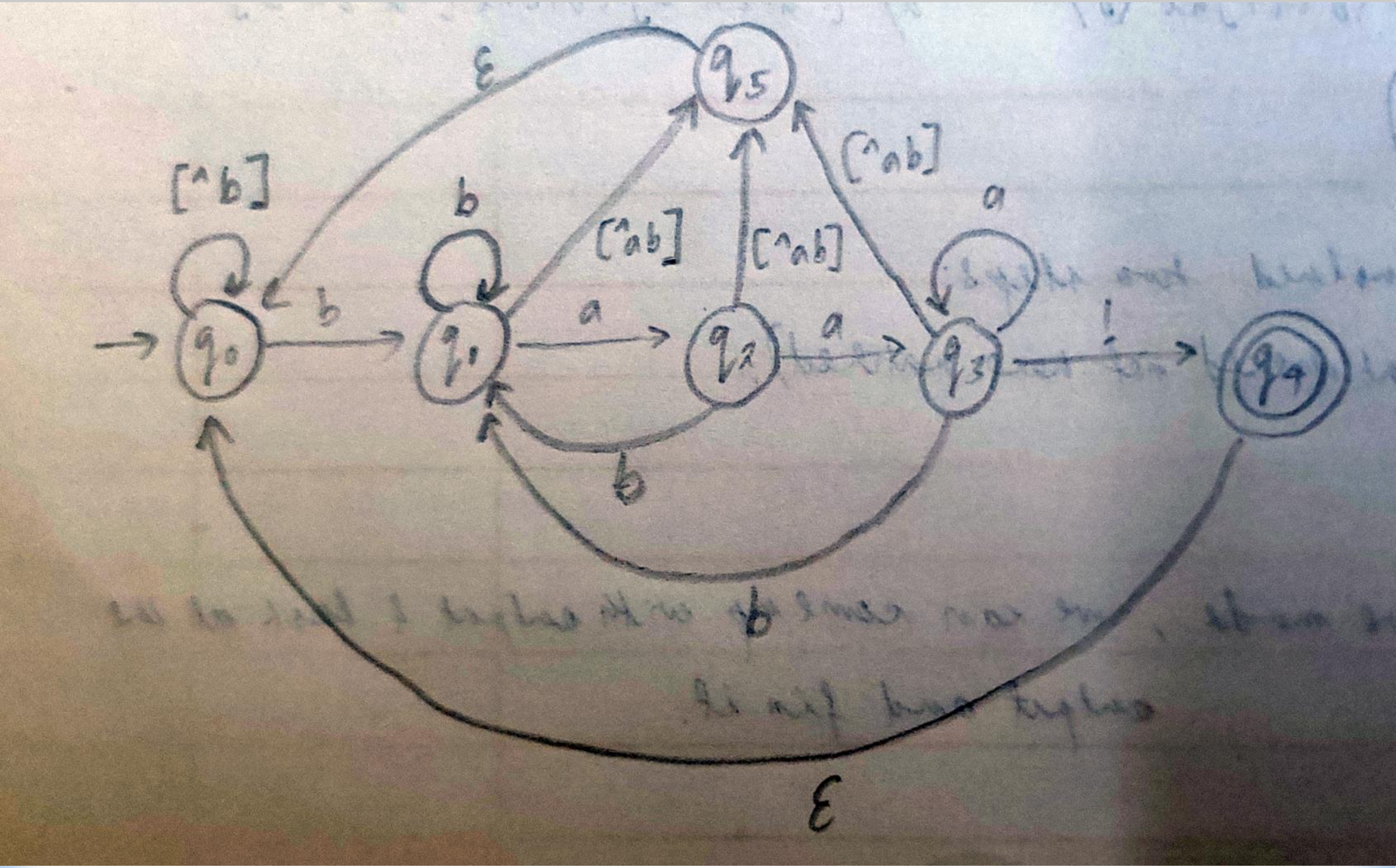
- We can say the following things about this machine
  - It has 5 states
  - b, a, and ! are in its alphabet
  - q_0 is the start state
  - q_4 is an accept state
  - It has 5 transitions



But Note

- There are other machines that correspond to this same language




q0
q1
q2
q3
q4
q5
[^b]
b
b
a
a
a
!
[^ab]
[^ab]
[^ab]
b
b
ε
ε

Finite State Automata (FSAs)

A finite-state automaton $M = \langle Q, \Sigma, q_0, F, \delta \rangle$ consists of:

- A finite set of states $Q = \{q_0, q_1, .., q_n\}$
- A finite alphabet Σ of input symbols (e.g. $\Sigma = \{a, b, c, ...\}$)
- A designated start state $q_0 \in Q$
- A set of final states $F \subseteq Q$
- A transition function δ:
  - The transition function for a deterministic (D)FSA: $Q \, x \, \Sigma \rightarrow Q$

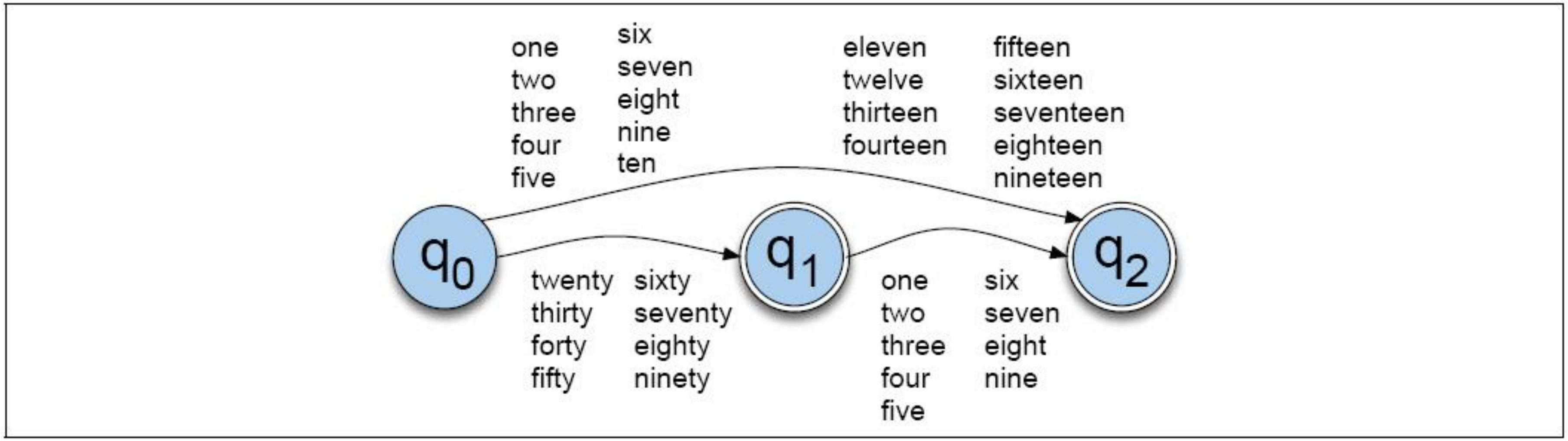
    $\delta(q,w) = q'$ for $q, q' \in Q, w \in \Sigma$

    If the current state is q and the current input is w, go to q'
  - The transition function for a nondeterministic (N)FSA: $Q \, x \, \Sigma \rightarrow 2^Q$

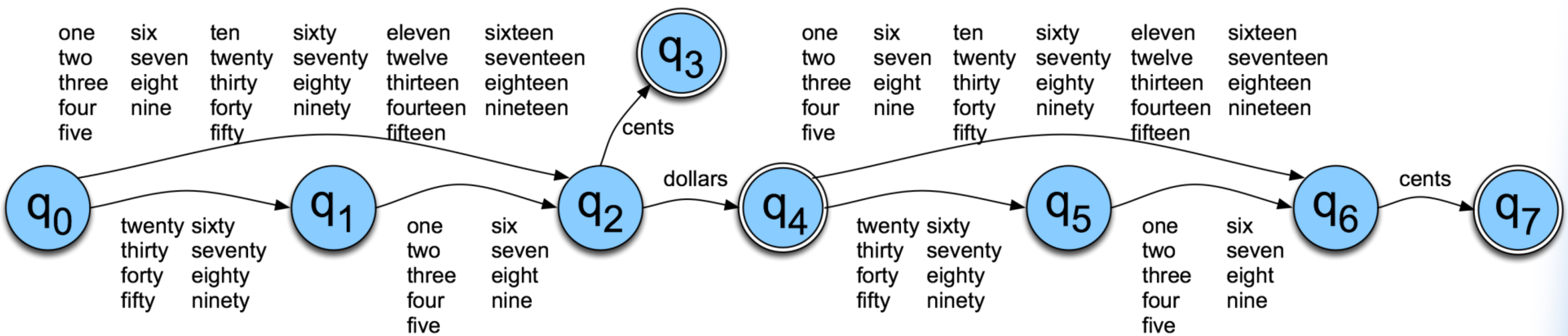
    $\delta(q,w) = Q'$ for $q \in Q, Q' \subseteq Q, w \in \Sigma$

    If the current state is q and the current input is w, go to any $q' \in Q'$

Dollars and Cents



Dollars and Cents

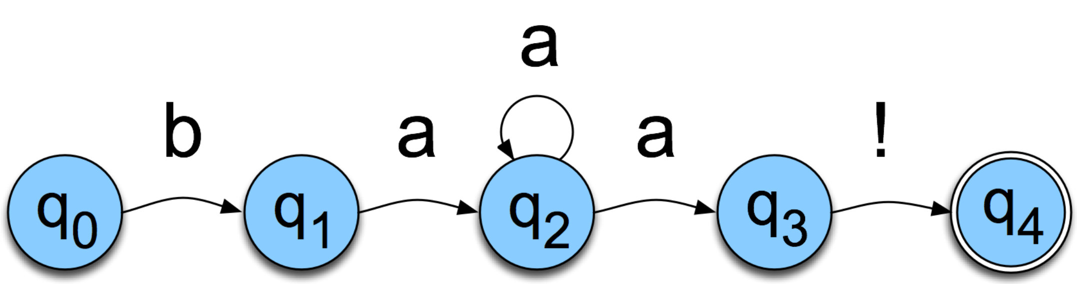


Yet Another View

- The guts of FSAs can ultimately be represented as tables

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

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



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
- Those all amount the same thing in the end

Recognition

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

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.

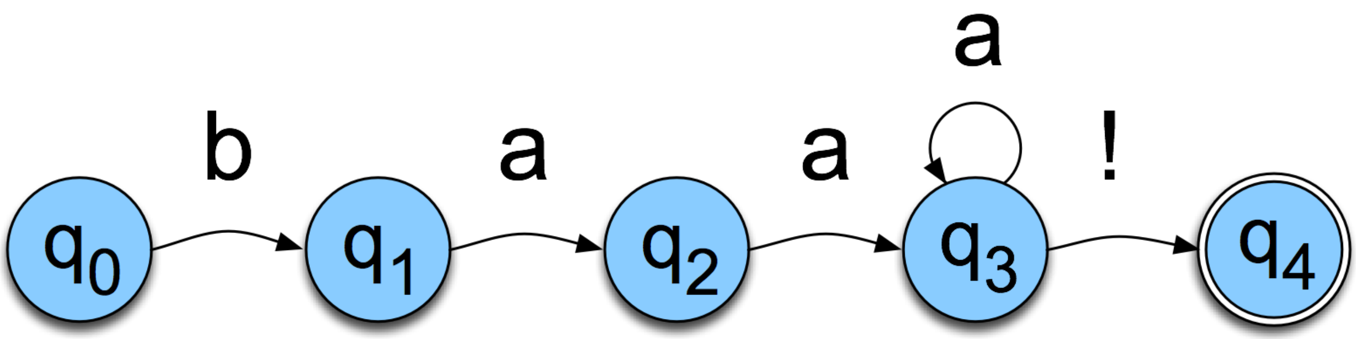
Generative Formalisms

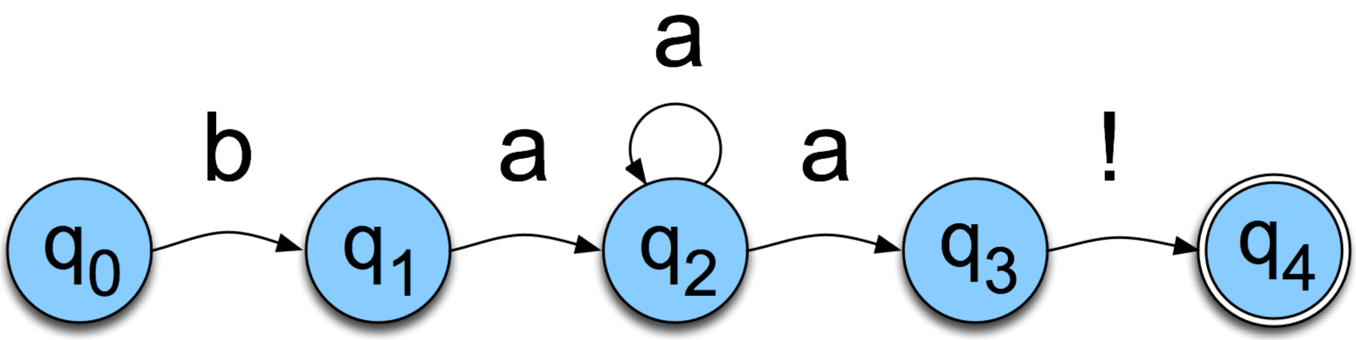
- *Formal Languages* consist of words whose letters are taken from an alphabet and are well-formed according to a specific set of rules
- Finite-state automata define formal languages (without having to enumerate all the strings in the language)
- The term *Generative* is based on the view that you can run the machine as a generator to get strings from the language.

Generative Formalisms

- FSAs can be viewed from two perspectives:
  - Acceptors that can tell you if a string is in the language
  - Generators to produce *all and only* the strings in the language

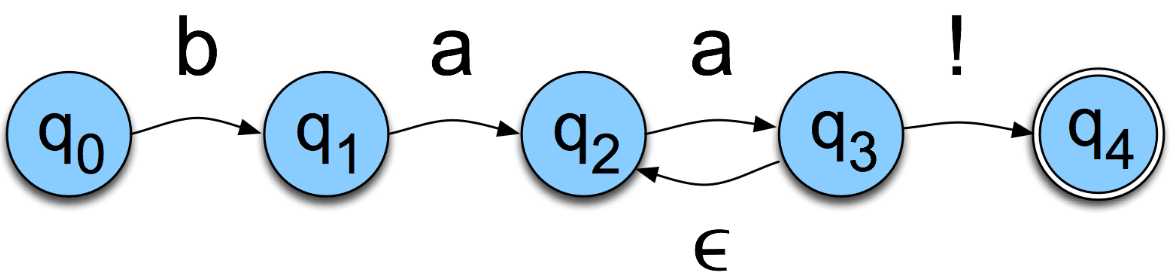
Non-Determinism





Non-Determinism cont.

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



Equivalence

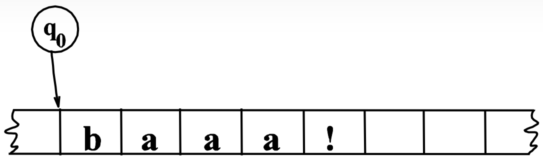
- Non-deterministic machines can be converted to deterministic ones with a fairly simple construction
- That means that they have the same power; non-deterministic machines are not more powerful than deterministic ones in terms of the languages they can accept

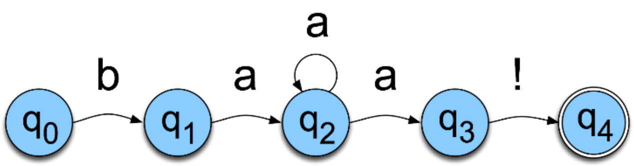
ND Recognition

- Two basic approaches (used in all major implementations of regular expressions, see Friedl 2006)
  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).

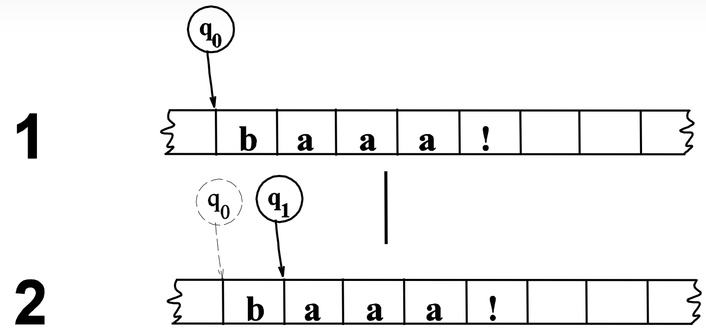
Example

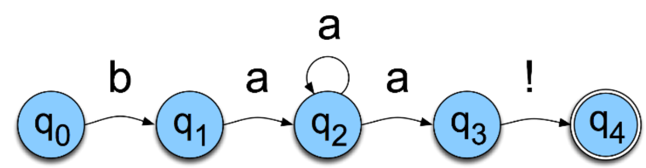
1



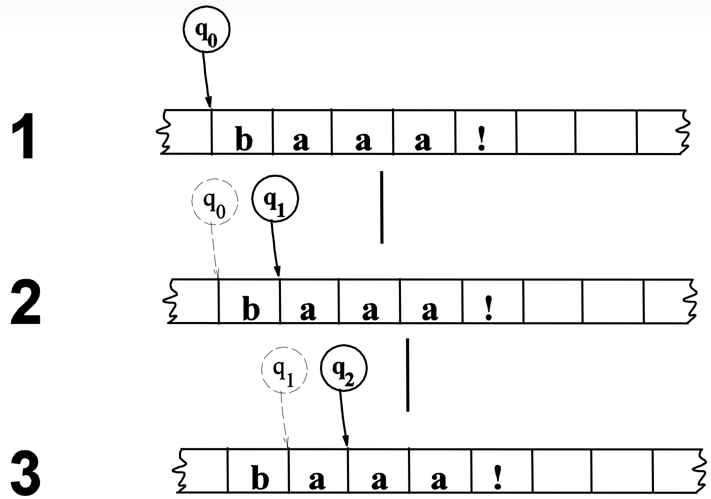


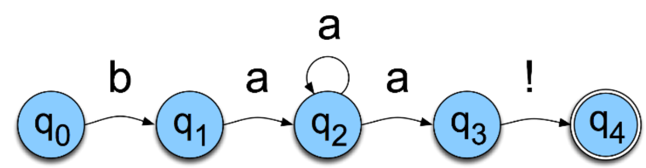
Example



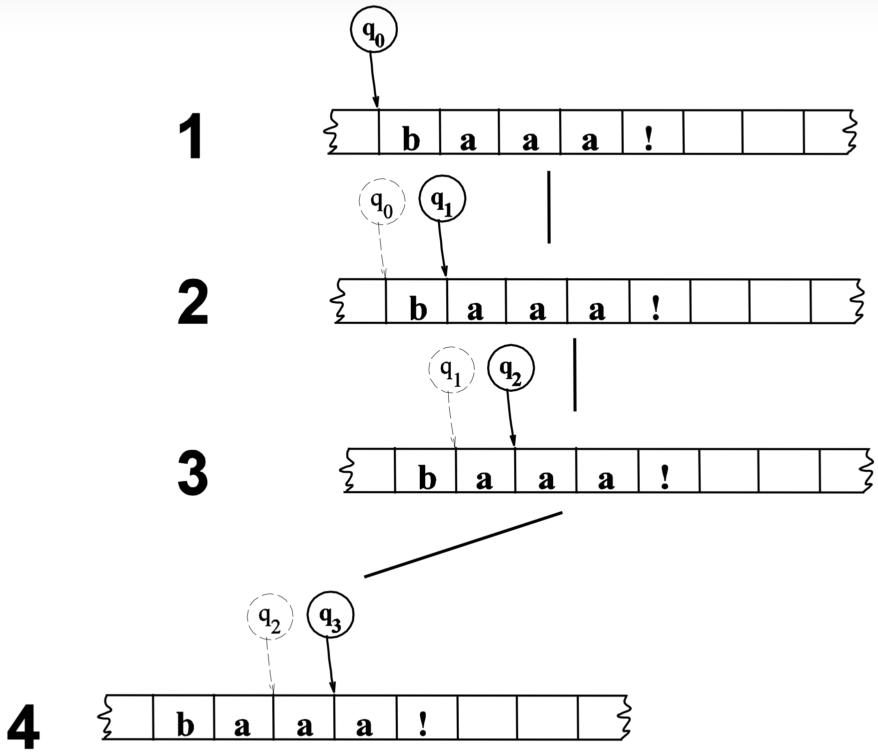


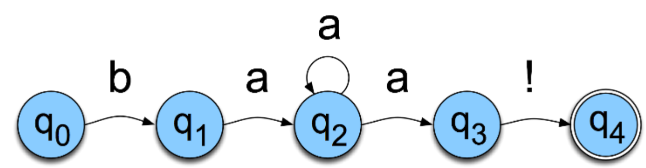
Example



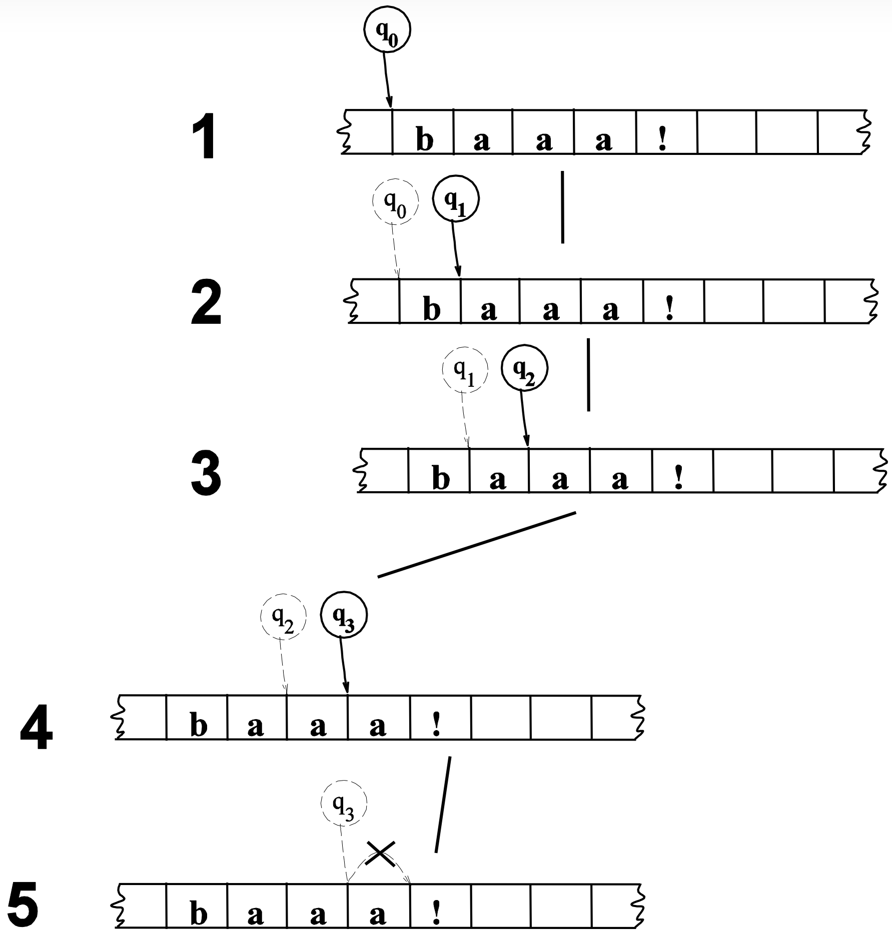


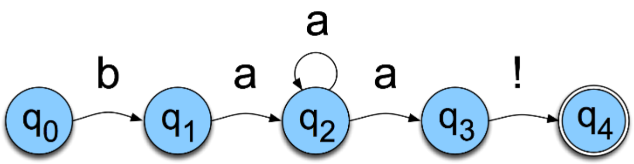
Example



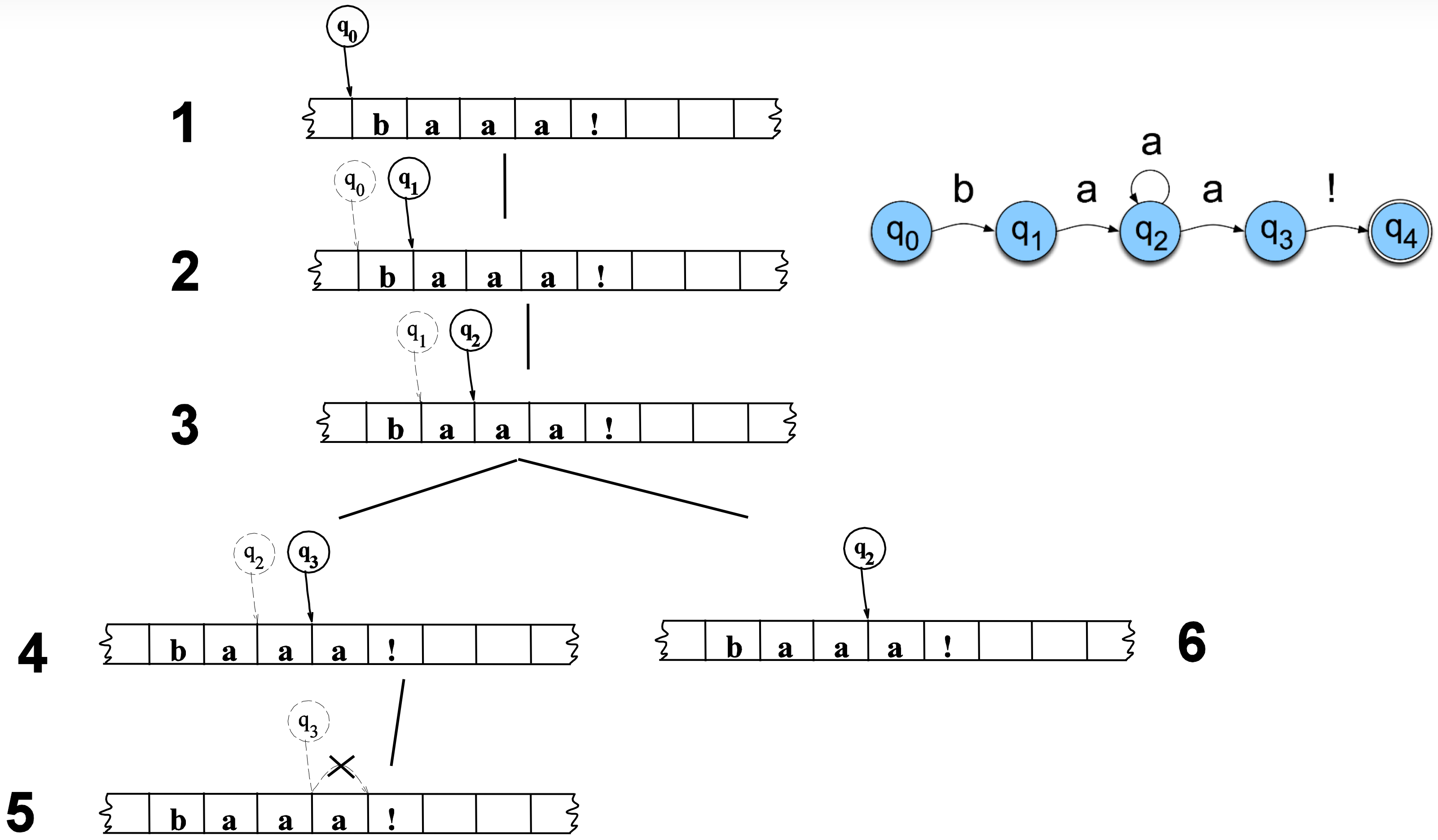


Example

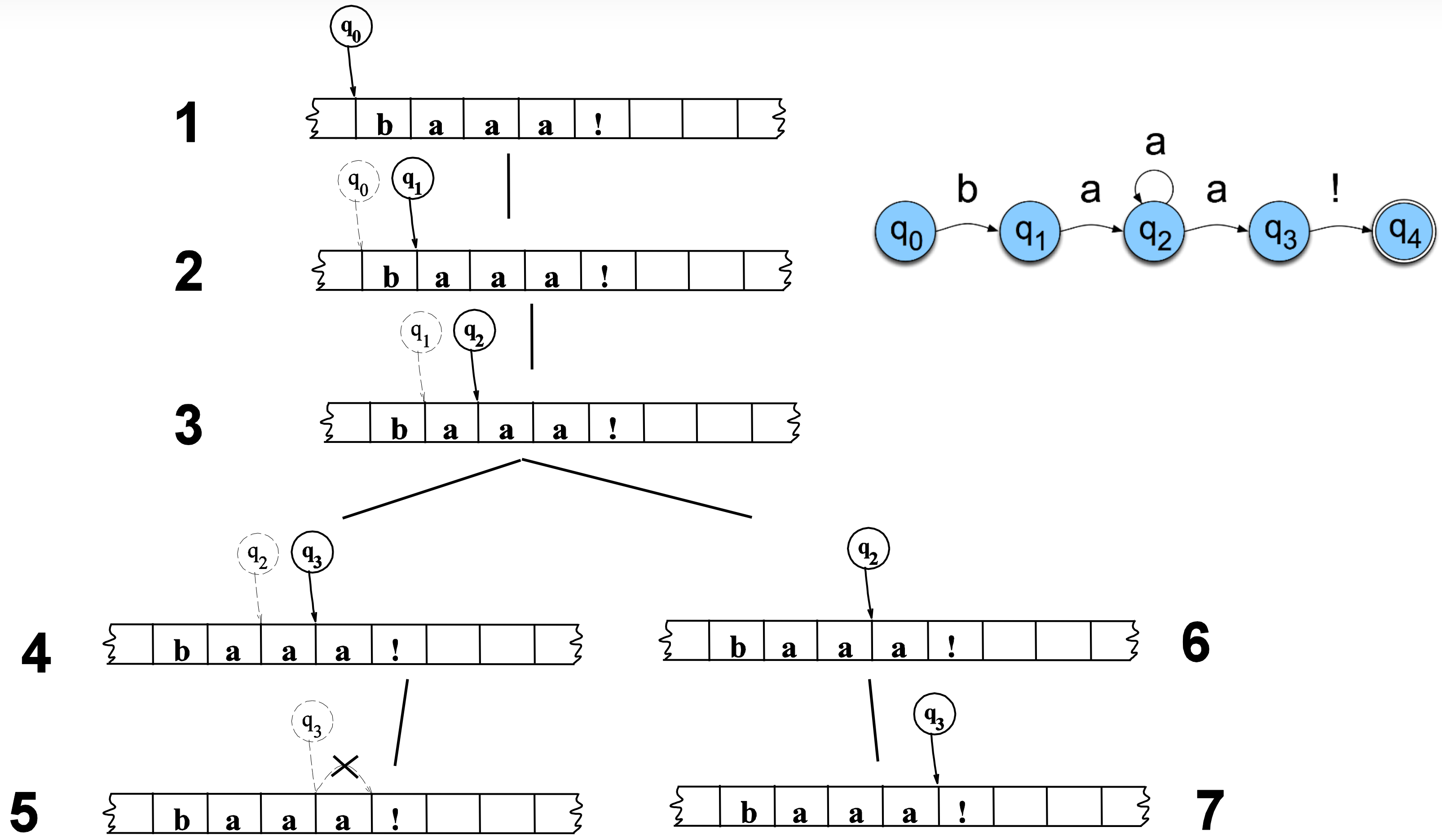




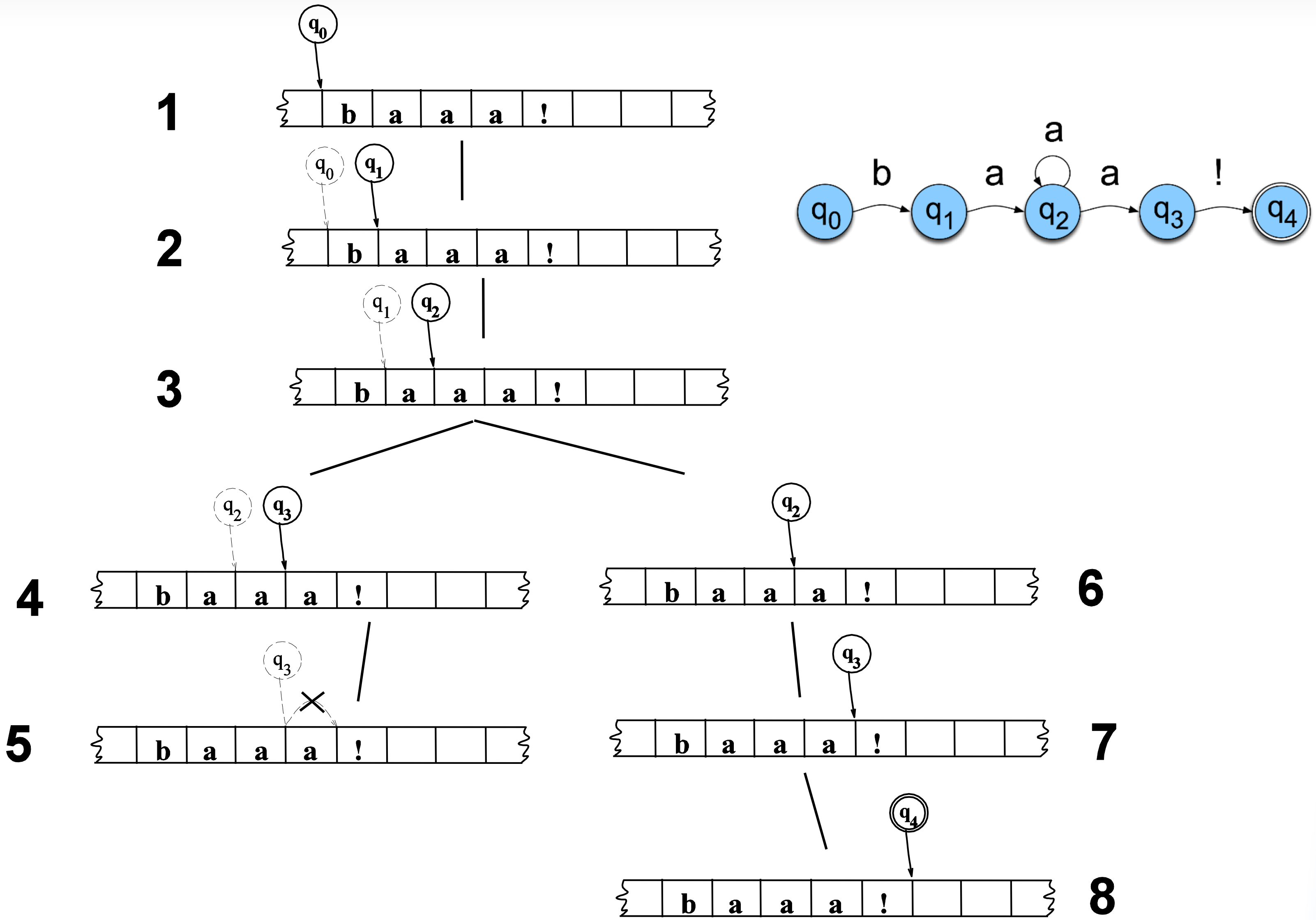
Example



Example



Example



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.

Uses of Regexes

- Observing simple subcomponents
  - Dollars and cents
  - Date, Time
  - Chemical compounds
  - Mathematical formulas
  - Word search in crossword puzzles
  - Noun compounds, Lexico-POS patterns

- Use regexes in low-data setting
- Use regexes as features in ML

Summing Up

- Regular expressions and FSAs can represent subsets of natural language as well as regular languages
  - Both representations may be difficult for humans to use for any real subset of a language
  - But quick, powerful and easy to use for small problems

- Finite state transducers and rules are common ways to incorporate linguistic ideas in NLP for small applications

- Particularly useful for no data setting