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 <u>woodchucks</u> and lemurs"
`/a/`	"M<u>a</u>ry Ann stopped by Mona's"
`/Claire␣says,/`	" "Dagmar, my gift please," <u>Claire says,</u>"
`/DOROTHY/`	"SURRENDER <u>DOROTHY</u>"
`/!/`	"You've left the burglar behind again<u>!</u>" 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

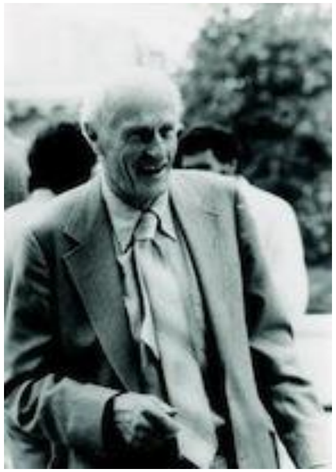
RE	Match	Example Patterns Matched
`/[A-Z]/`	an upper case letter	"we should call it 'Drenched Blossoms' "
`/[a-z]/`	a lower case letter	"my beans were impatient to be hoed!"
`/[0-9]/`	a single digit	"Chapter 1: Down the Rabbit Hole"

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

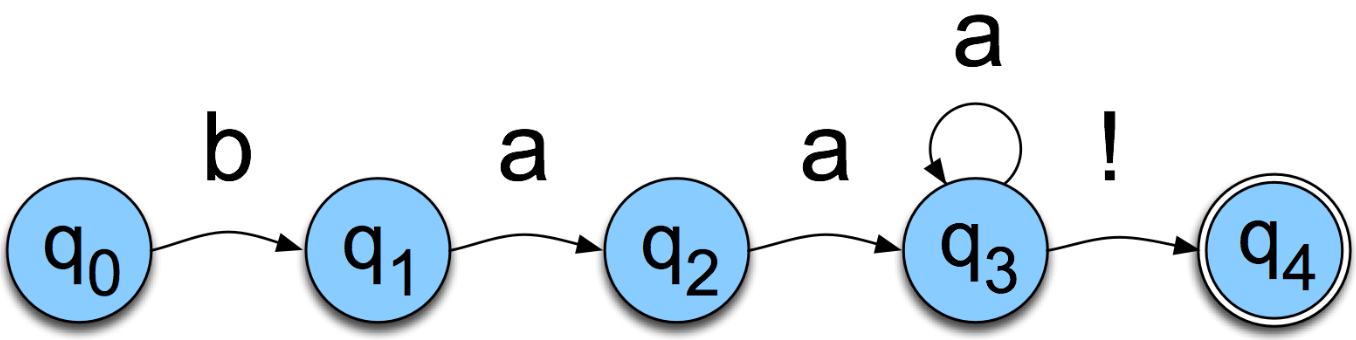
- We'll be telling the same story for many tasks, all semester. Reducing the error rate for an application often involves two antagonistic efforts:
  - Increasing accuracy, or precision, (minimizing false positives)
  - Increasing coverage, or recall, (minimizing false negatives).

Finite State Automata

- Regular expressions can be viewed as a textual way of specifying the structure of finite-state automata.
- FSAs capture significant aspects of what linguists say we need for morphology and parts of syntax.

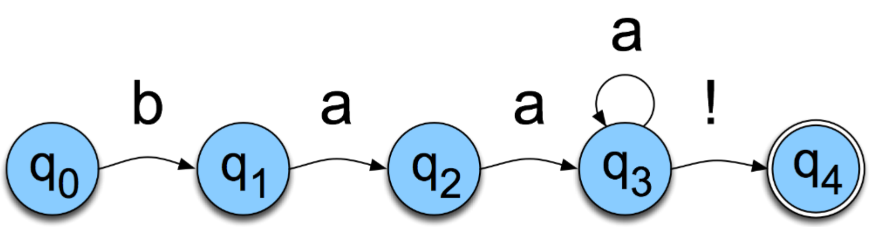
FSAs as Graphs

- Let’s start with the sheep language from Chapter 2
  - `/baa+!/`



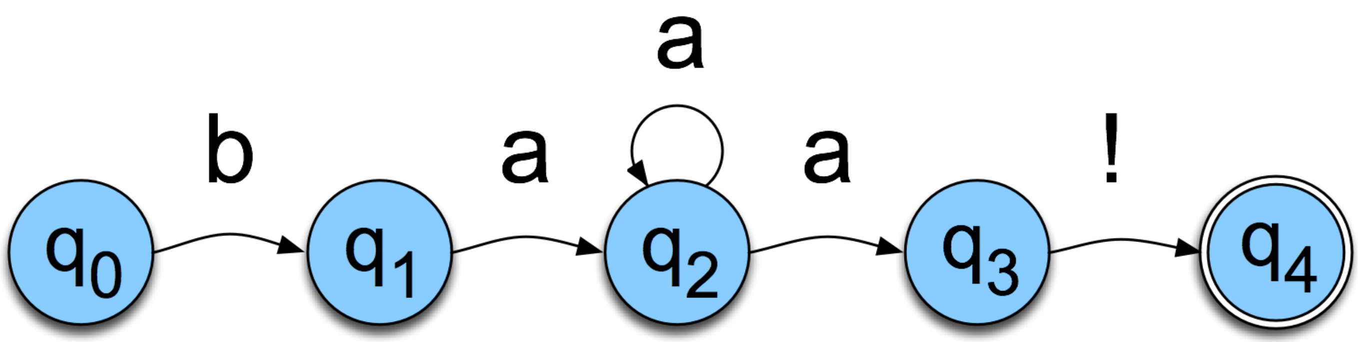
Sheep FSA

- 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

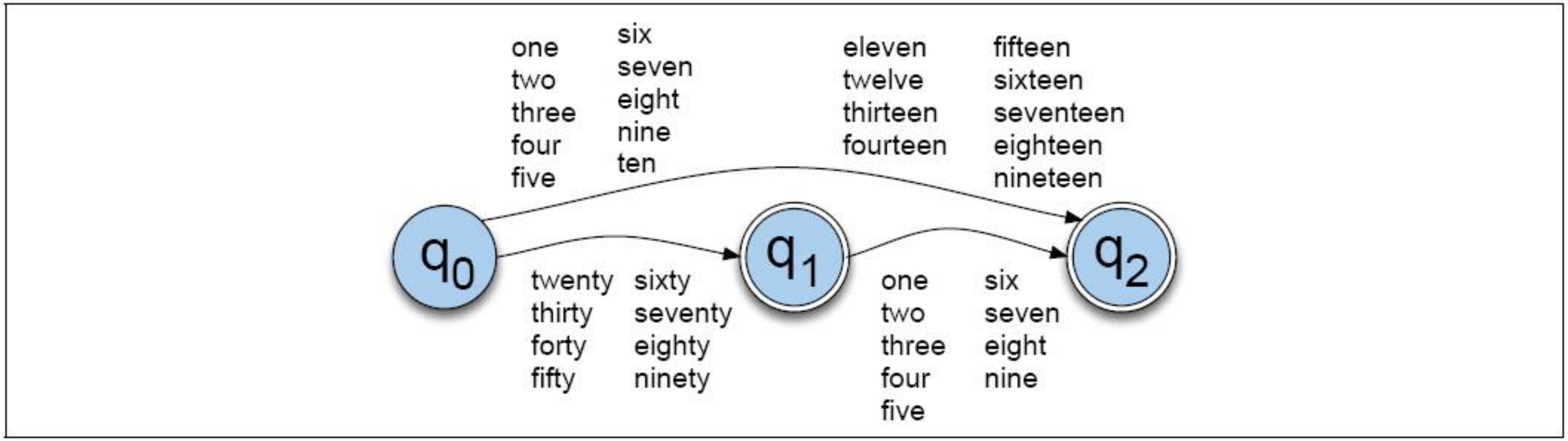
- There are other machines that correspond to this same language



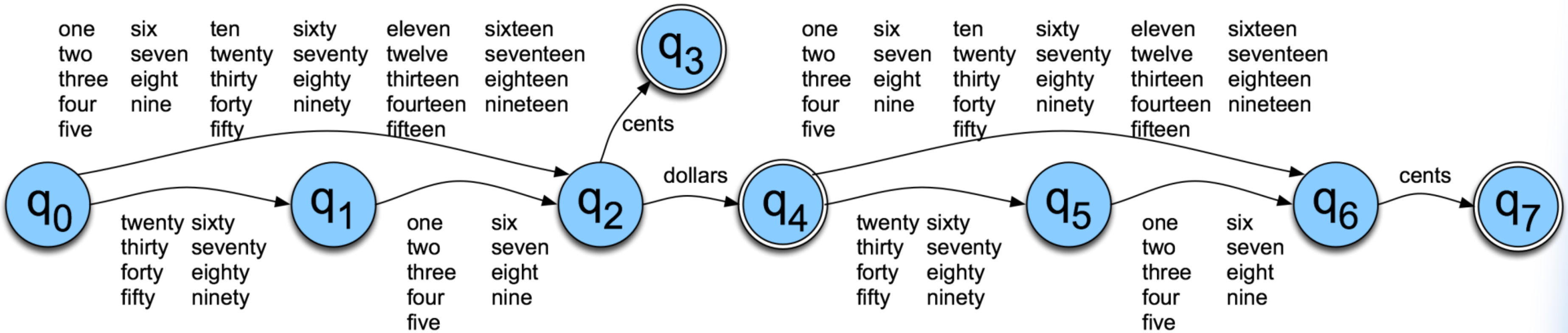
More Formally

- 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 QxΣ to Q

Dollars and Cents



Dollars and Cents

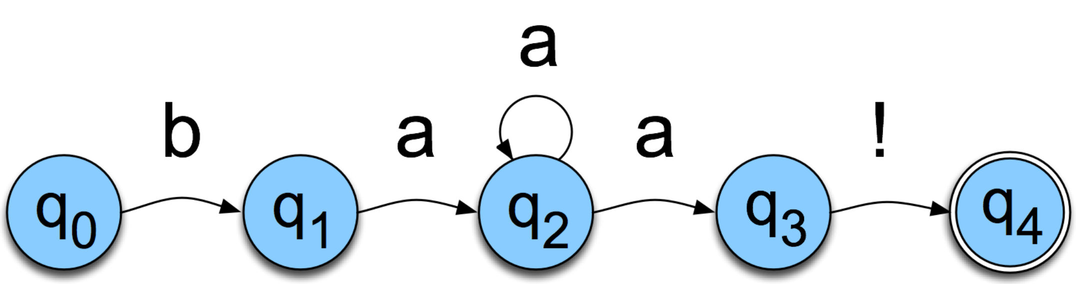


Yet Another View

- 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

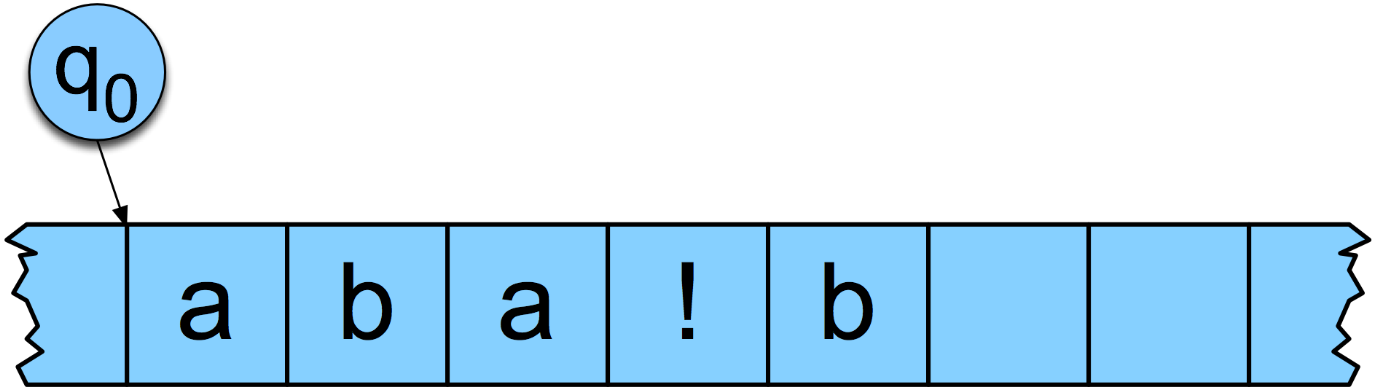


Recognition

- 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

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



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 tape pointer.
- Until you run out of tape.

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

- Crudely therefore… 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

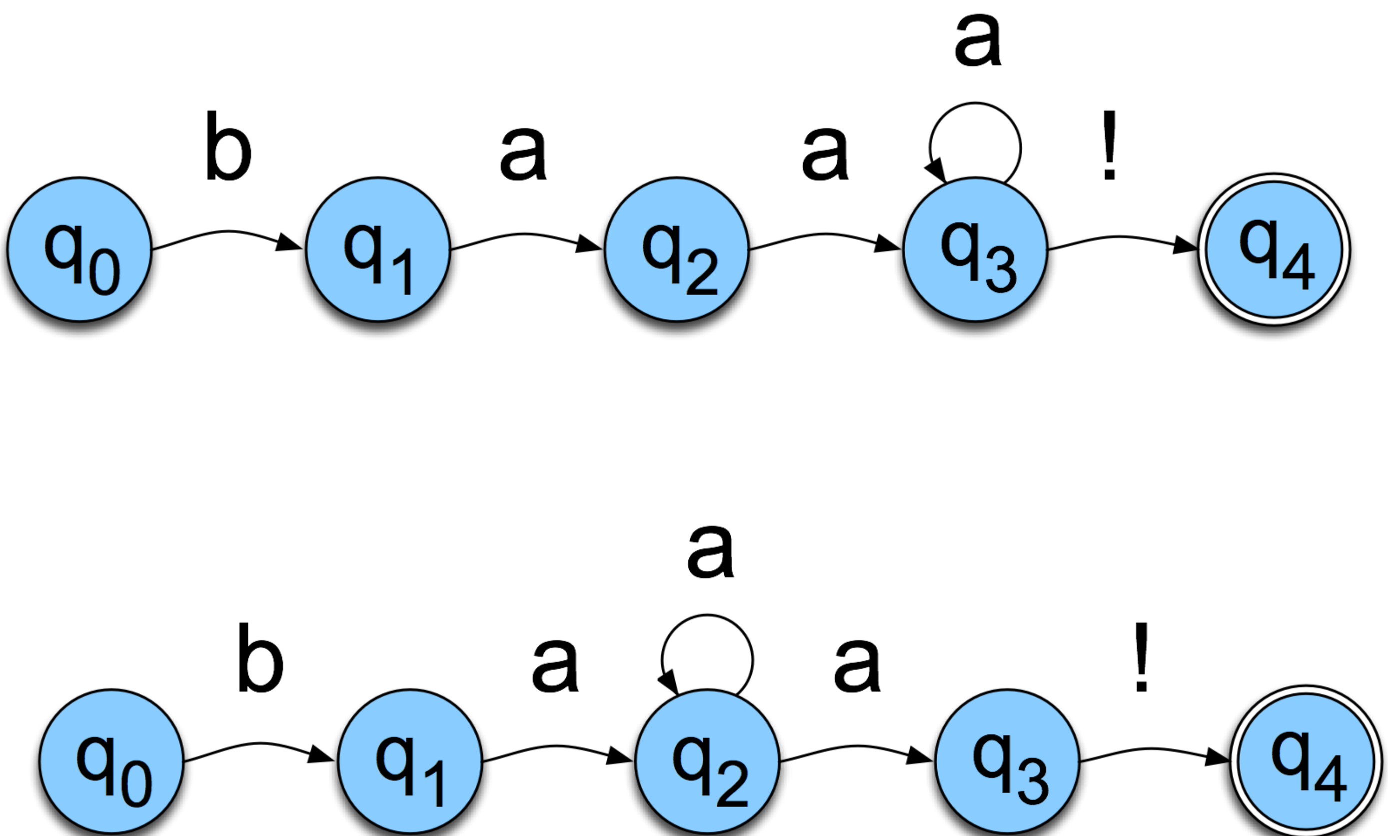
Generative Formalisms

- *Formal Languages* are sets of strings composed of symbols from a finite set of symbols.
- 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



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).

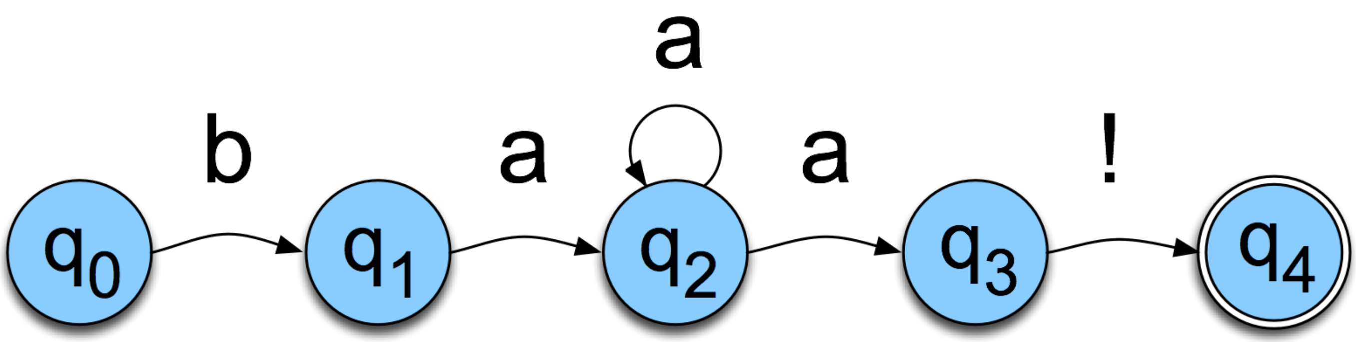
Non-Deterministic Recognition: Search

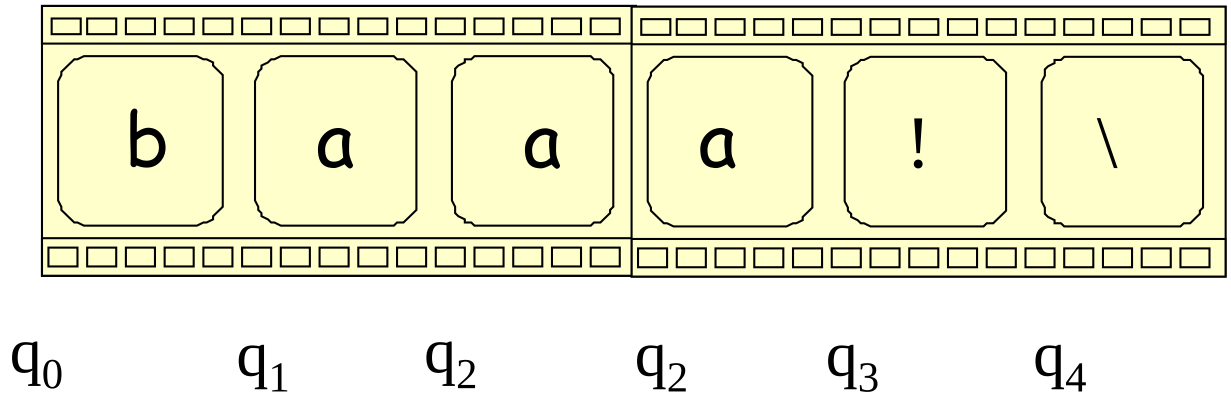
- In a ND FSA there exists at least one path through the machine for a string that is in the language defined by the machine.
- But not all paths directed through the machine for an accept string lead to an accept state.
- No paths through the machine lead to an accept state for a string not in the language.

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.

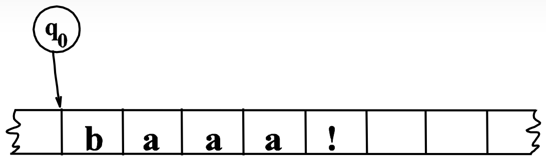
Example

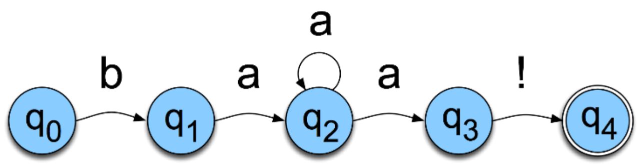




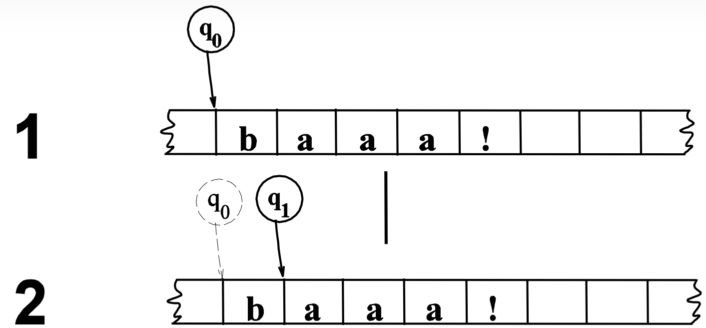
Example

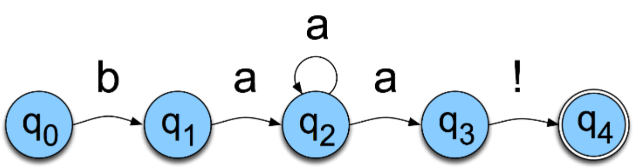
1



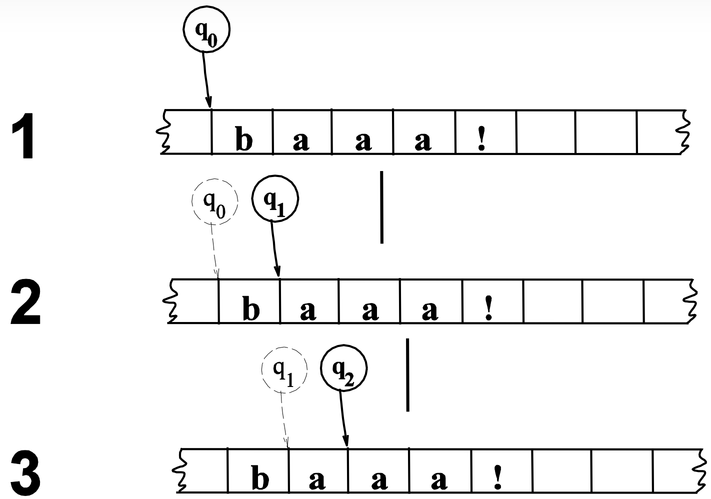


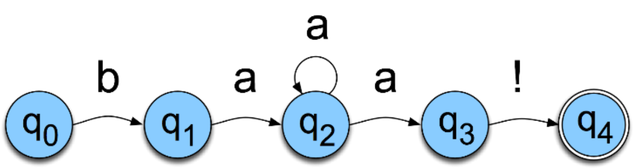
Example



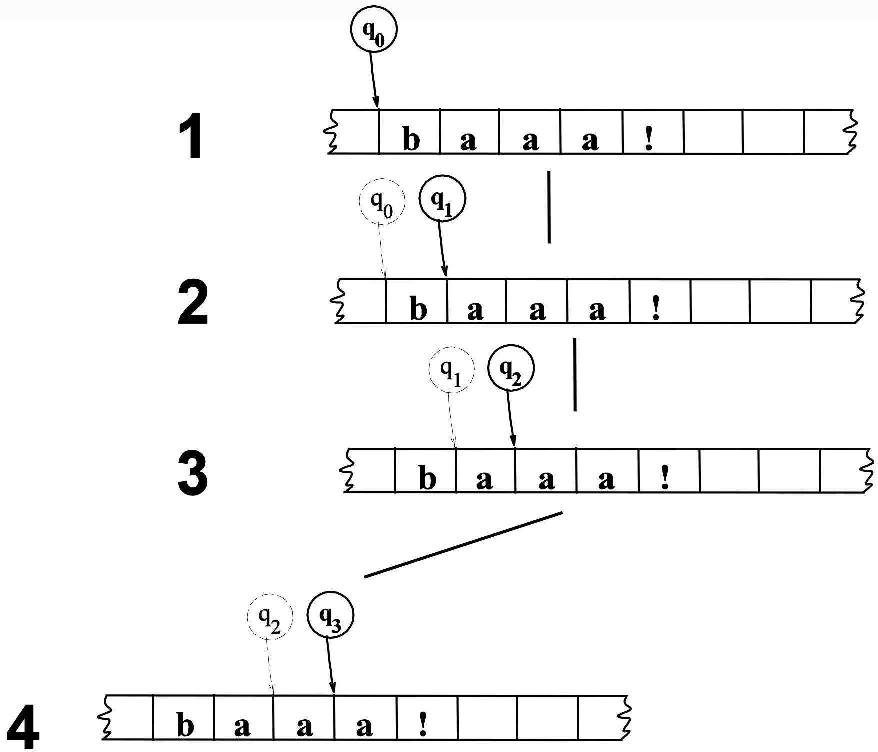


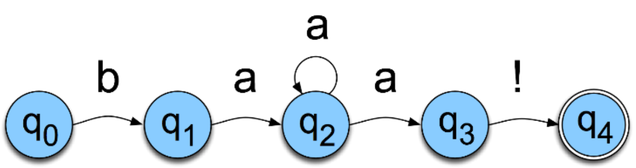
Example



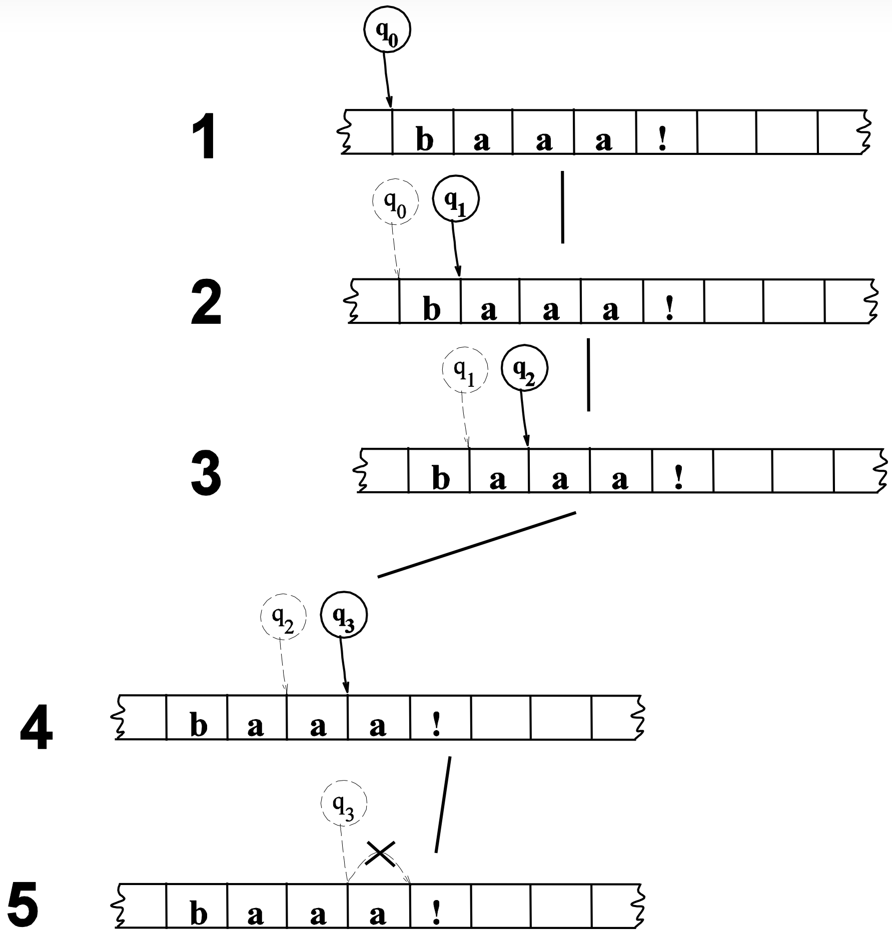


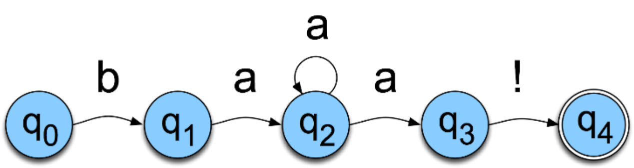
Example



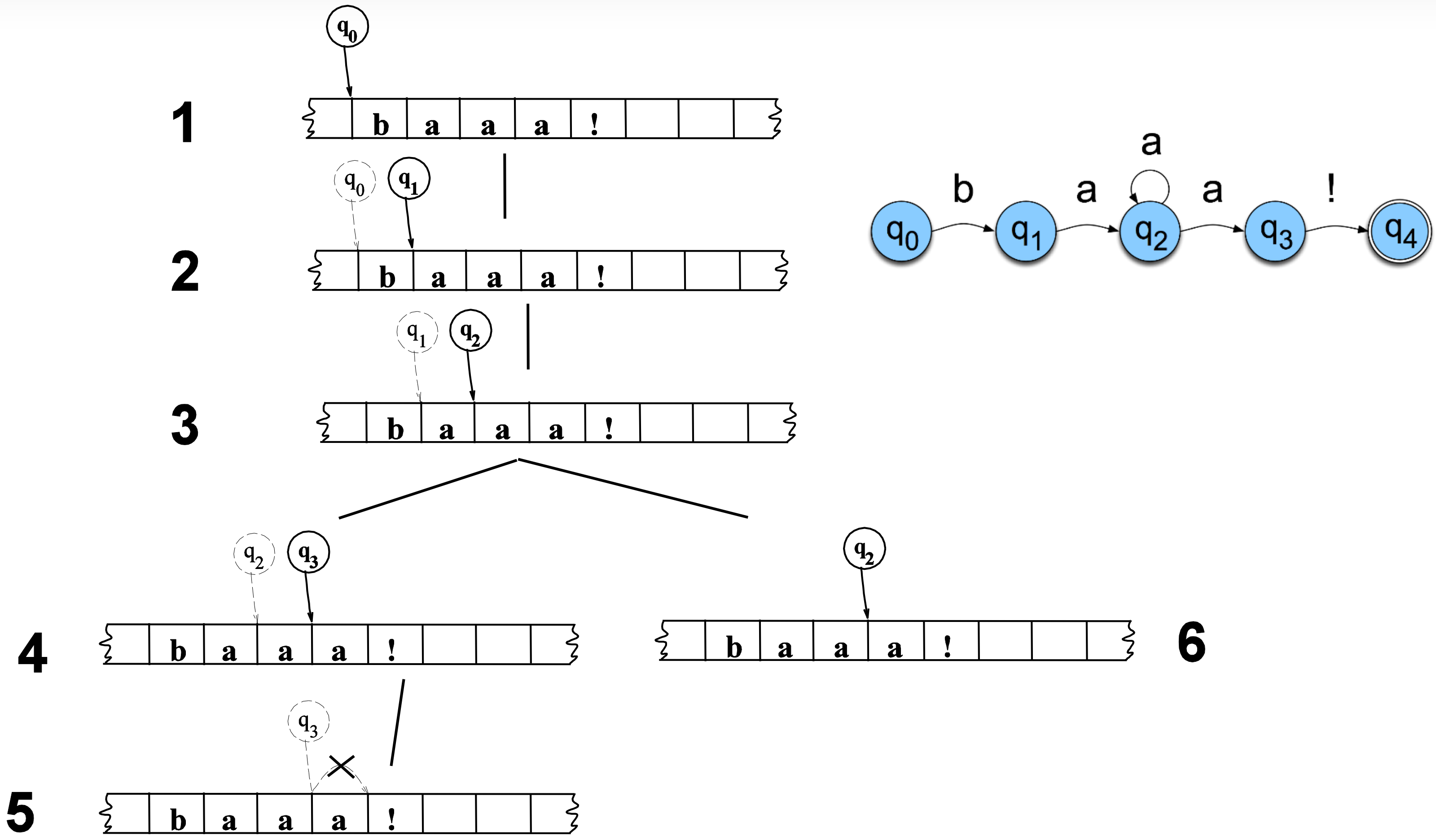


Example

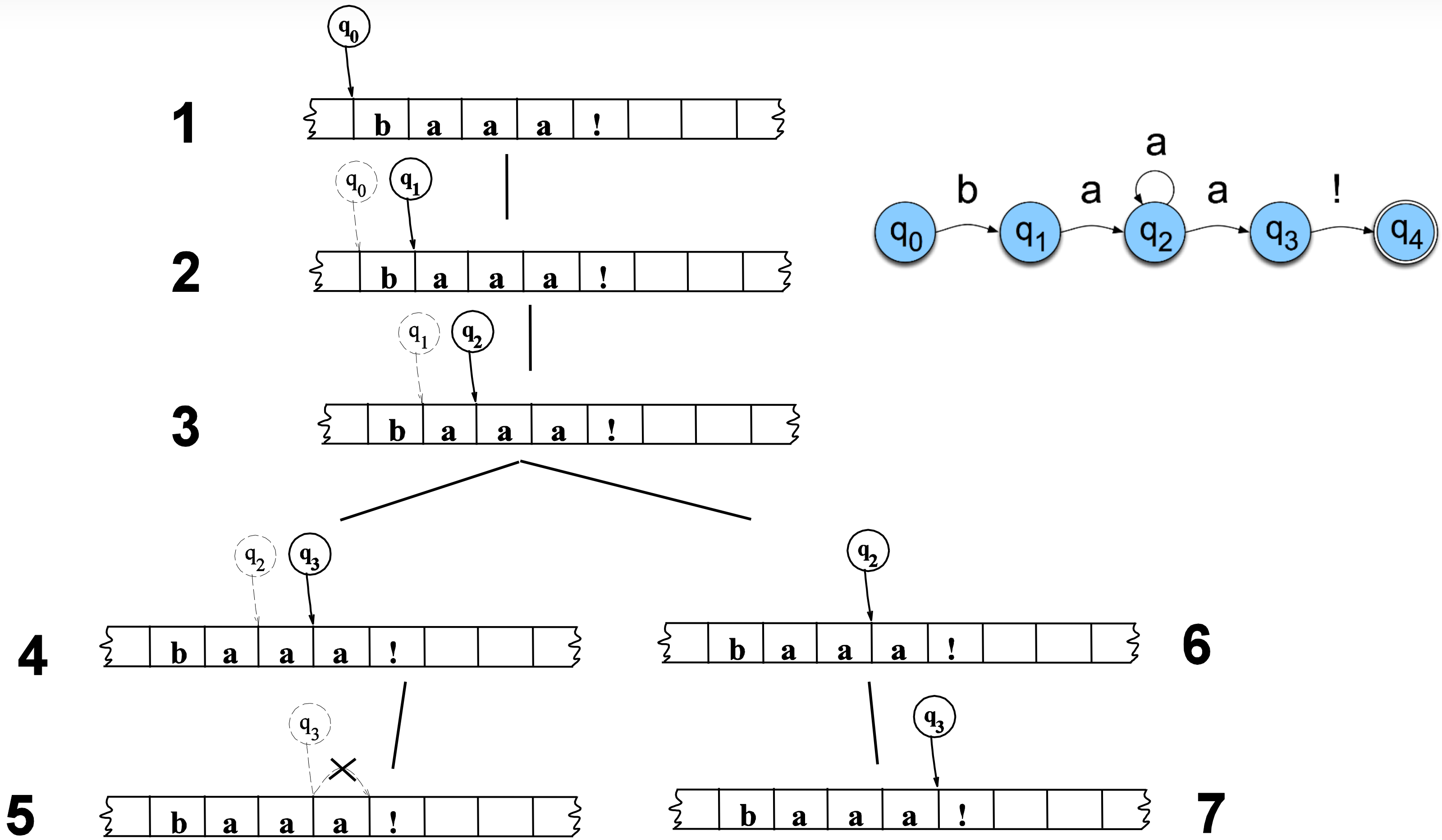




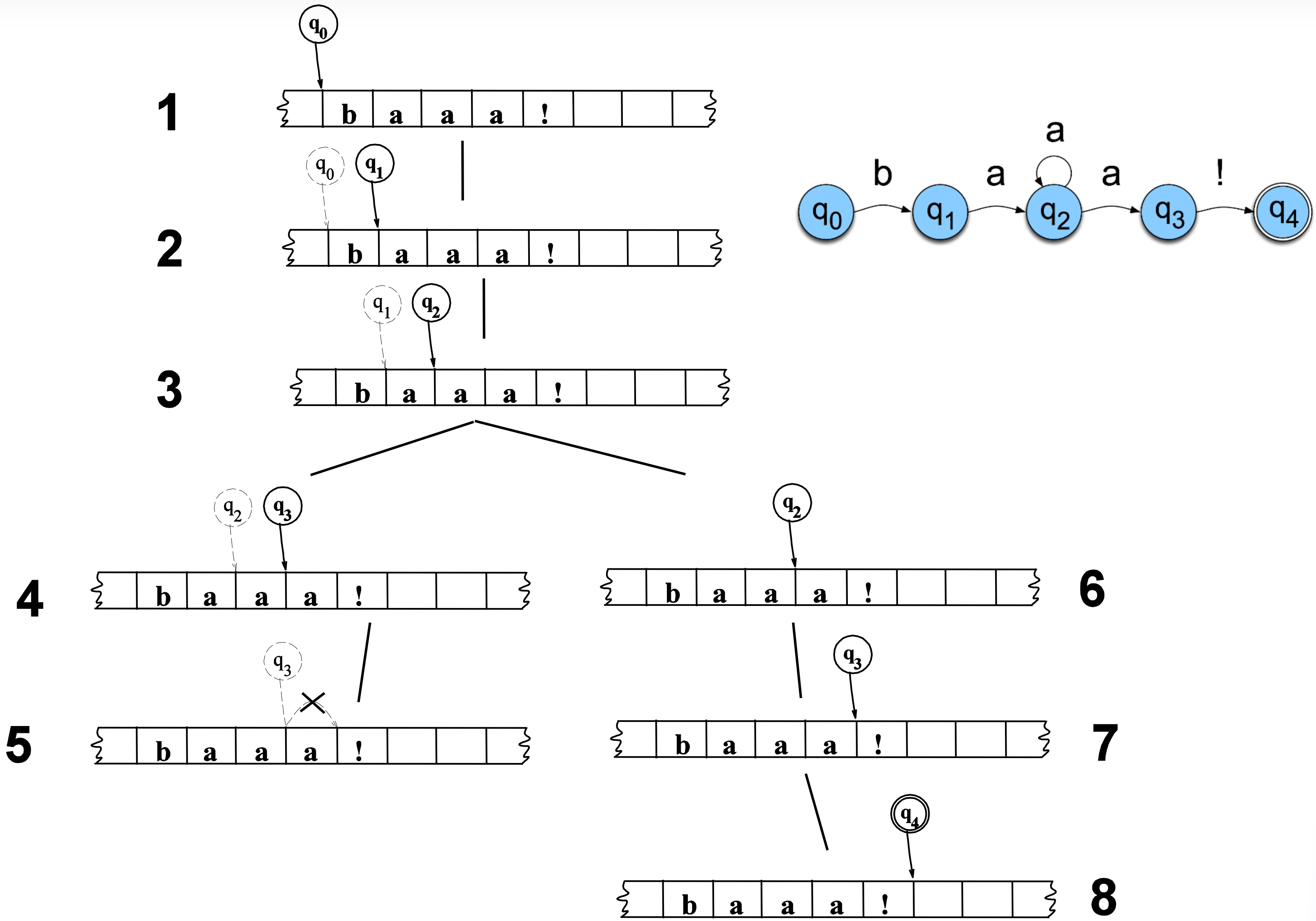
Example



Example



Example



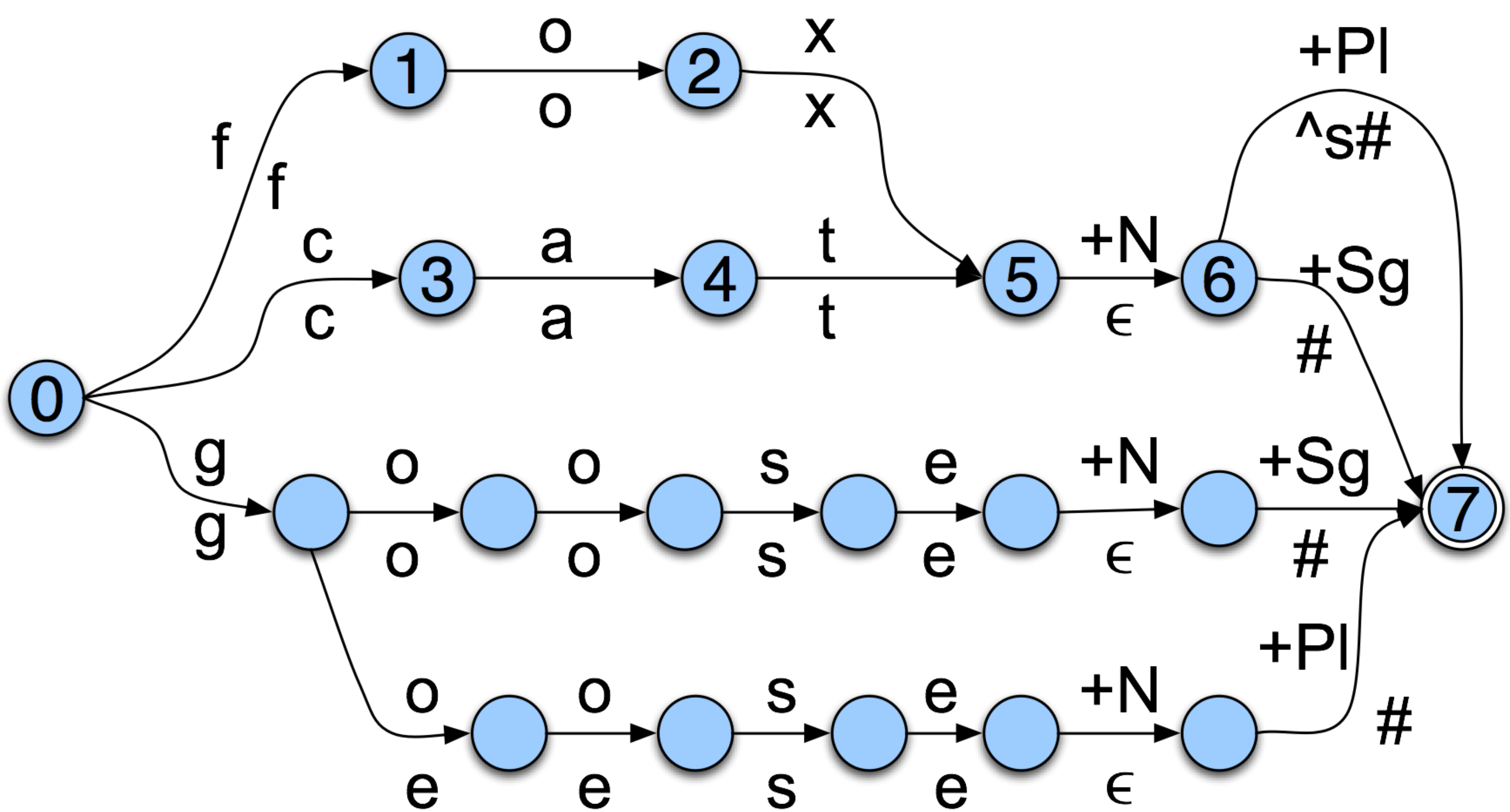
Key Points

- 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.

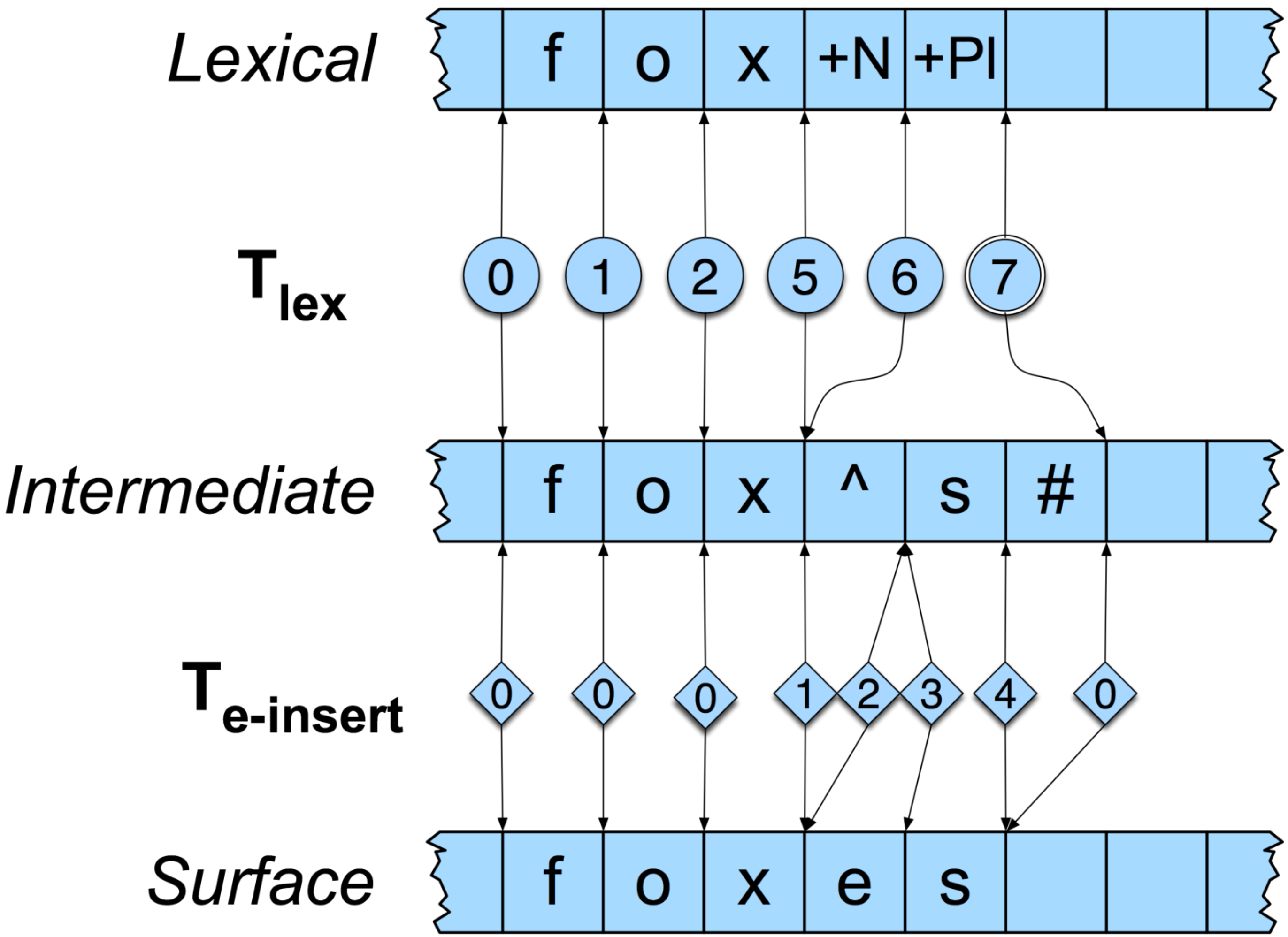
FSTs (Contd)

FST Fragment: Lexical to Intermediate

- ^ is morpheme boundary; # is word boundary



Putting Them Together



Practical Uses

- This kind of parsing is normally called morphological analysis
- Can be
  - An important stand-alone component of an application (spelling correction, information retrieval, part-of-speech tagging,…)
  - Or simply a link in a chain of processing (machine translation, parsing,…)

FST-based Tokenization

```
#!/usr/bin/perl

$letternumber = "[A-Za-z0-9]";
$notletter = "[ˆA-Za-z0-9]";
$alwayssep = "[\\?!()\";/\\|‘]";
$clitic = "(’|:|-|’S|’D|’M|’LL|’RE|’VE|N’T|’s|’d|’m|’ll|’re|’ve|n’t)";

$abbr{"Co."} = 1; $abbr{"Dr."} = 1; $abbr{"Jan."} = 1; $abbr{"Feb."} = 1;

while ($line = <>){ # read the next line from standard input

    # put whitespace around unambiguous separators
    $line =˜ s/$alwayssep/ $& /g;

    # put whitespace around commas that aren’t inside numbers
    $line =˜ s/([ˆ0-9]),/$1 , /g;
    $line =˜ s/,([ˆ0-9])/ , $1/g;

    # distinguish singlequotes from apostrophes by
    # segmenting off single quotes not preceded by letter
    $line =˜ s/ˆ’/$& /g;
    $line =˜ s/($notletter)’/$1 ’/g;

    # segment off unambiguous word-final clitics and punctuation
    $line =˜ s/$clitic$/ $&/g;
    $line =˜ s/$clitic($notletter)/ $1 $2/g;

   # now deal with periods.  For each possible word
   @possiblewords=split(/\s+/,$line);
   foreach $word (@possiblewords) {
      # if it ends in a period,
      if (($word =˜ /$letternumber\./)
             && !($abbr{$word})  # and isn’t on the abbreviation list
                # and isn’t a sequence of letters and periods (U.S.)
                # and doesn’t resemble an abbreviation (no vowels: Inc.)
             && !($word =˜
                 /ˆ([A-Za-z]\.([A-Za-z]\.)+|[A-Z][bcdfghj-nptvxz]+\.)$/)) {
          # then segment off the period
          $word =˜ s/\.$/ \./;
      }
      # expand clitics
      $word =˜s/’ve/have/;
      $word =˜s/’m/am/;
      print $word," ";
   }
 print "\n";
}
```

Porter Stemmer (1980)

- Common algorithm for stemming English
- Conventions + 5 phases of reductions
  - phases applied sequentially
  - each phase consists of a set of commands
  - sample convention: *Of the rules in a compound command, select the one that applies to the longest suffix.*

Porter Stemmer (1980)

- Standard, very popular and usable stemmer (IR, IE) – identify a word's stem
- Sequence of cascaded rewrite rules, e.g.
  - IZE → ε (e.g. unionize → union)
  - CY → T (e.g. frequency → frequent)
  - ING → ε , if stem contains vowel (motoring → motor)
- Can be implemented as a lexicon-free FST (many implementations available on the web)
- http://text-processing.com/demo/stem/

Eliza

User_1:	Men are all alike.
ELIZA_1:	IN WHAT WAY
User_2:	They're always bugging us about something or other.
ELIZA_2:	CAN YOU THINK OF A SPECIFIC EXAMPLE
User_3:	Well, my boyfriend made me come here.
ELIZA_3:	YOUR BOYFRIEND MADE YOU COME HERE
User_4:	He says I'm depressed much of the time.
ELIZA_4:	I AM SORRY TO HEAR YOU ARE DEPRESSED

Eliza FST

```
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/
```

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