The Basics of Regular Expressions

COSI 114 – Computational Linguistics

James Pustejovsky

January 13, 2015

Brandeis University

Regular Expressions

- In computer science, RE is a language used for specifying text search strings.
- A *regular expression* is a *formula* in a *special language* that is used for specifying a simple class of *string*.
- Formally, a regular expression is an algebraic notation for characterizing a set of strings.
- RE search requires
  - a *pattern* that we want to search for, and
  - a *corpus* of texts to search through.

Regular Expressions

- A RE search function will search through the corpus returning all texts that contain the pattern.
  - In a Web search engine, they might be the entire documents or Web pages.
  - In a word-processor, they might be individual words, or lines of a document. (We take this paradigm.)
    - E.g., the UNIX `grep` command

Regular Expressions
Basic Regular Expression Patterns

- The use of the brackets `[]` to specify a disjunction of characters.

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"

- The use of the brackets `[]` plus the dash – to specify a range.

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

Regular Expressions
Basic Regular Expression Patterns

- Uses of the caret ^ for negation or just to mean ^

RE	Match (single characters)	Example Patterns Matched
[^A-Z]	not an uppercase 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"

- The question-mark ? marks optionality of the previous expression.

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

- The use of period . to specify any character

RE	Match	Example Patterns
/beg.n/	any character between *beg* and *n*	begin, beg'n, begun

Regular Expressions
Disjunction, Grouping, and Precedence

- Disjunction

```
/cat|dog
```

- Precedence

```
/gupp(y|ies)
```

- Operator precedence hierarchy

```
()
+ ? { }
the  ^my  end$
|
```

Regular Expressions
A Simple Example

- To find the English article *the*

```
/the/

/[tT]he/

/\b[tT]he\b/

/[^a-zA-Z][tT]he[^a-zA-Z]/

/^|[^a-zA-Z][tT]he[^a-zA-Z]/
```

Regular Expressions
A More Complex Example

- "any PC with more than ***500 MHz*** and ***32 Gb*** of disk space for less than ***$1000***"

```
/$[0-9]+/

/$[0-9]+\.[0-9][0-9]/

/\b$[0-9]+(\.[0-9][0-9])?\b/

/\b[0-9]+ *(MHz|[Mm]egahertz|GHz|[Gg]igahertz)\b/

/\b[0-9]+ *(Mb|[Mm]egabytes?)\b/

/\b[0-9](\.[0-9]+)? *(Gb|[Gg]igabytes?)\b/

/\b(Win95|Win98|WinNT|Windows *(NT|95|98|2000)?)\b/

/\b(Mac|Macintosh|Apple)\b/
```

Regular Expressions
Advanced Operators

Aliases for common sets of characters

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

Regular Expressions
Advanced Operators

Regular expression operators for counting

RE	Match
*	zero or more occurrences of the previous char or expression
+	one or more occurrences of the previous char or expression
?	exactly zero or one occurrence of the previous char or expression
{n}	*n* occurrences of the previous char or expression
{n,m}	from *n* to *m* occurrences of the previous char or expression
{n,}	at least *n* occurrences of the previous char or expression

Regular Expressions
Advanced Operators

Some characters that need to be backslashed

RE	Match	Example Patterns Matched
*	an asterisk "*"	"K*A*P*L*A*N"
\.	a period "."	"Dr. Livingston, I presume"
\?	a question mark	"Would you light my candle?"
\n	a newline	
\t	a tab	

Regular Expressions
Regular Expression Substitution, Memory, and ELIZA

```
s/regexp1/regexp2/
```

- E.g. *the 35 boxes → the <35> boxes*

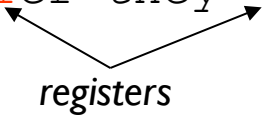
```
s/([0-9]+)/<\1>/
```

- The following pattern matches "*The bigger they were, the bigger they will be*", not *"The bigger they were, the faster they will be"*

```
/the (.*)er they were, the\1er they will be/
```

- The following pattern matches "*The bigger they were, the bigger they were*", not *"The bigger they were, the bigger they will be"*

```
/the (.*)er they (.*), the\1er they \2/
```



Regular Expressions
Regular Expressions Substitution, Memory, and ELIZA

- Eliza worked by having a cascade of regular expression substitutions that each match some part of the input lines and changed them
  - *my → YOUR, I'm → YOU ARE …*

```
s/.* YOU ARE (depressed|sad) .*/I AM SORRY TO HEAR YOU ARE \1/
s/.* YOU ARE (depressed|sad) .*/WHY DO YOU THINK YOU ARE \1/
s/.* all .*/IN WHAT WAY/
s/.* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE/
```

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 BOYBRIEND 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

Operations on strings

- Given two strings $s = a_1 \ldots a_n$ and $t = b_1 \ldots b_m$, we define their concatenation $st = a_1 \ldots a_n b_1 \ldots b_m$

  $s = abb, t = cba$ $\quad st = abbcba$

- We define s^n as the concatenation $ss \ldots s$ n times

  $s = 011$ $\quad s^3 = 011011011$

Operations on languages

- The concatenation of languages L_1 and L_2 is

$$L_1L_2 = \{st: s \in L_1, t \in L_2\}$$

- Similarly, we write L^n for $LL\ldots L$ (n times)
- The union of languages $L_1 \cup L_2$ is the set of all strings that are in L_1 or in L_2

- Example: $L_1 = \{01, 0\}, L_2 = \{\varepsilon, 1, 11, 111, \ldots\}$. What is L_1L_2 and $L_1 \cup L_2$?

Operations on languages

- The star (Kleene closure) of L are all strings made up of zero or more chunks from L:

$$L^* = L^0 \cup L^1 \cup L^2 \cup \ldots$$

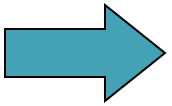
  - This is always infinite, and always contains ε

- Example: $L_1 = \{01, 0\}, L_2 = \{\varepsilon, 1, 11, 111, \ldots\}$.
  What is L_1^* and L_2^*?

Constructing languages with operations

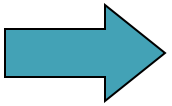
- Let's fix an alphabet, say $\Sigma = \{0, 1\}$
- We can construct languages by starting with simple ones, like $\{0\}$, $\{1\}$ and combining them

$\{0\}(\{0\} \cup \{1\})^*$
all strings that start with 0



$0(0+1)^*$

$(\{0\}\{1\}^*) \cup (\{1\}\{0\}^*)$



01^*+10^*

Regular expressions

- A regular expression over Σ is an expression formed using the following rules:
  - The symbol $\varnothing$ is a regular expression
  - The symbol ε is a regular expression
  - For every $a \in \Sigma$, the symbol a is a regular expression
  - If R and S are regular expressions, so are RS, $R+S$ and R^*.

- Definition of regular language

> A language is regular if it is represented by a regular expression

Examples

1. $01^* = \{0, 01, 011, 0111, \ldots..\}$
2. $(01^*)(01) = \{001, 0101, 01101, 011101, \ldots..\}$
3. $(0+1)^*$
4. $(0+1)^*01(0+1)^*$
5. $((0+1)(0+1)+(0+1)(0+1)(0+1))^*$
6. $((0+1)(0+1))^*+((0+1)(0+1)(0+1))^*$
7. $(1+01+001)^*(\varepsilon+0+00)$

Examples

- Construct a RE over $\Sigma = \{0,1\}$ that represents
  - All strings that have two consecutive 0s.

    $$(0+1)^*00(0+1)^*$$

  - All strings except those with two consecutive 0s.

    $$(1^*01)^*1^* + (1^*01)^*1^*0$$

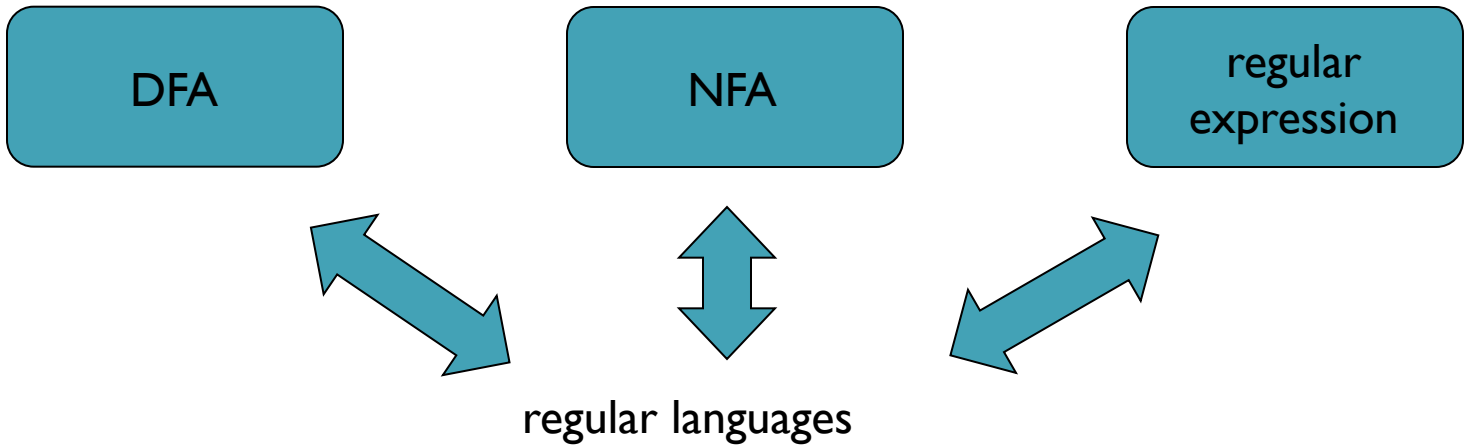
  - All strings with an even number of 0s.

    $$(1^*01^*01^*)^*$$

Main theorem for regular languages

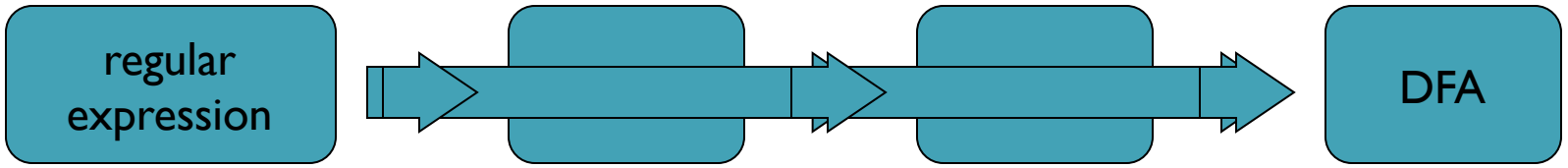
- Theorem

A language is regular if and only if it is the language of some DFA



Proof plan

- For every regular expression, we have to give a DFA for the same language

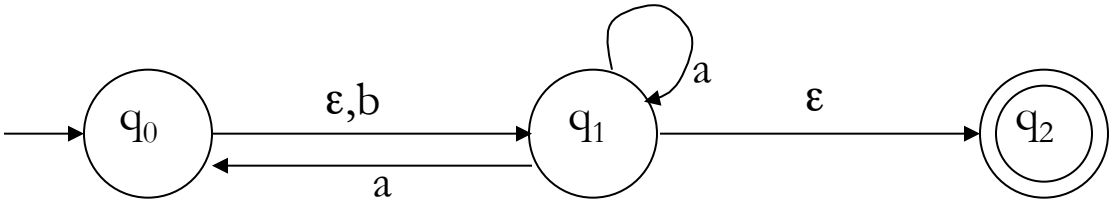


- For every DFA, we give a regular expression for the same language

What is an εNFA?

- An εNFA is an extension of NFA where some transitions can be labeled by ε
  - Formally, the transition function of an εNFA is a function
    $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow$ subsets of Q
- The automaton is allowed to follow ε-transitions without consuming an input symbol

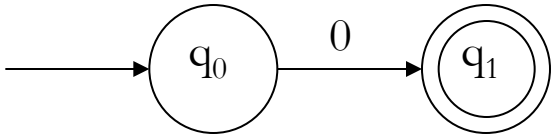
Example of εNFA



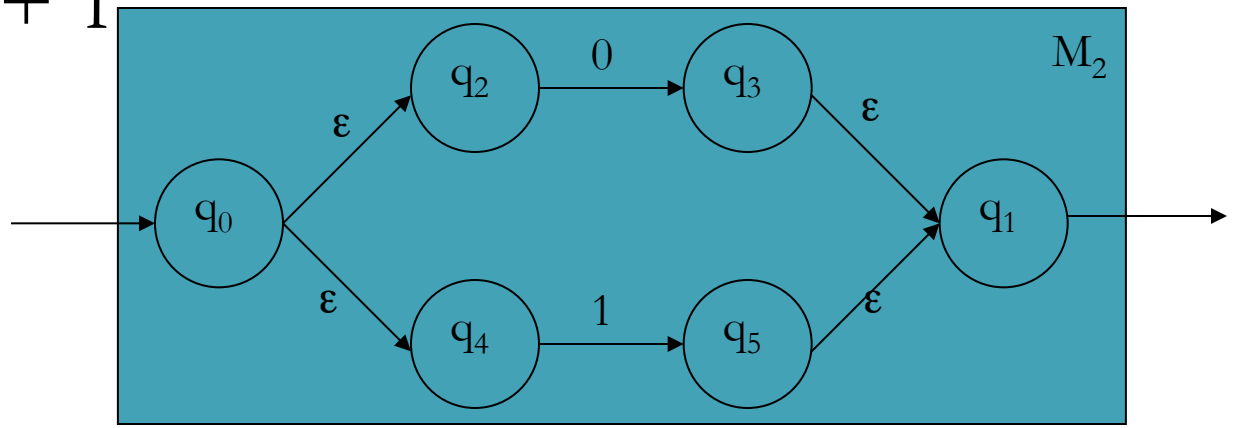
- Which of the following is accepted by this εNFA:
  - aab, bab, ab, bb, a, ε

Examples: regular expression → εNFA

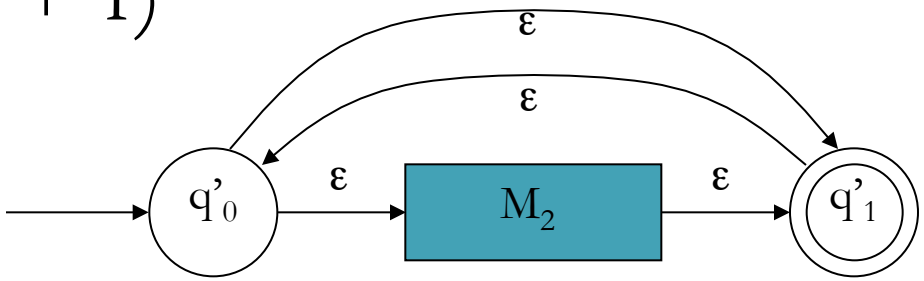
- $R_1 = 0$



- $R_2 = 0 + 1$



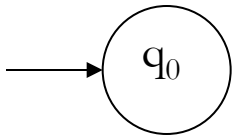
- $R_3 = (0 + 1)^*$



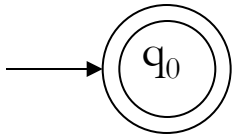
General method

regular expr ⇒ εNFA

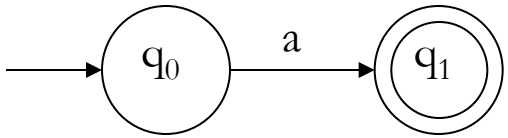
$\emptyset$



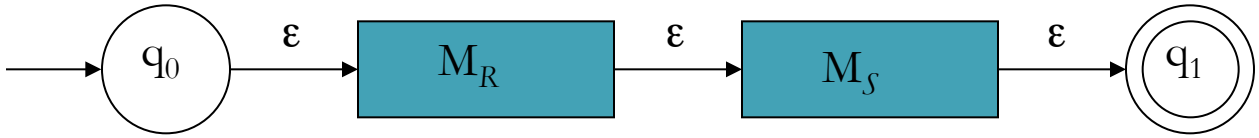
ε



symbol a



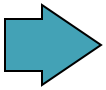
RS



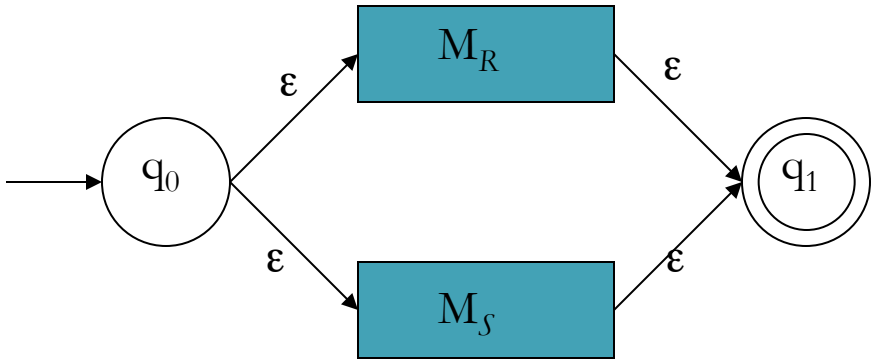
Convention

- When we draw a box around an εNFA:
  - The arrow going in points to the start state
  - The arrow going out represents all transitions going out of accepting states
  - None of the states inside the box is accepting
  - The labels of the states inside the box are distinct from all other states in the diagram

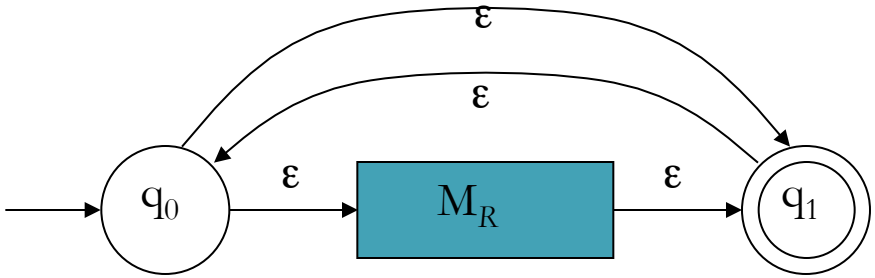
General method continued

regular expr  εNFA

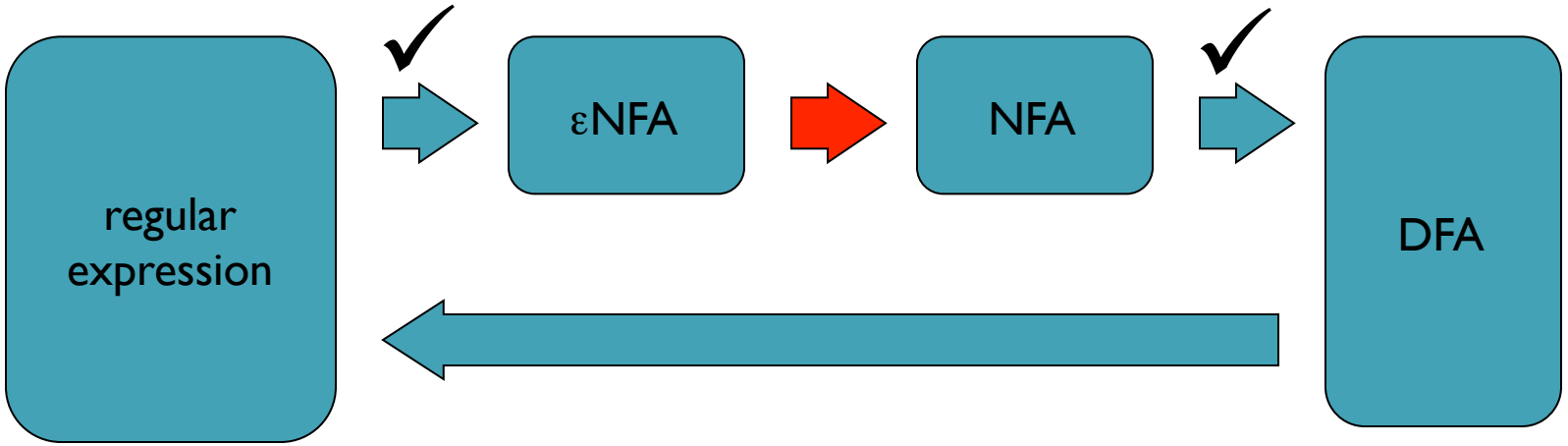
$R + S$



R^*

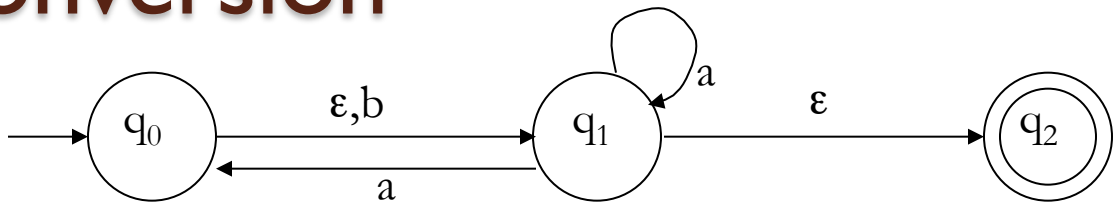


Road map



Example of εNFA to NFA conversion

εNFA:



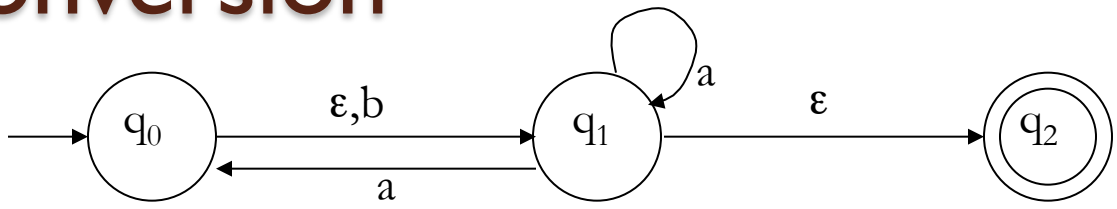
Transition table of corresponding NFA:

states	inputs: a	inputs: b
q_0	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$
q_1	$\{q_0, q_1, q_2\}$	$\varnothing$
q_2	$\varnothing$	$\varnothing$

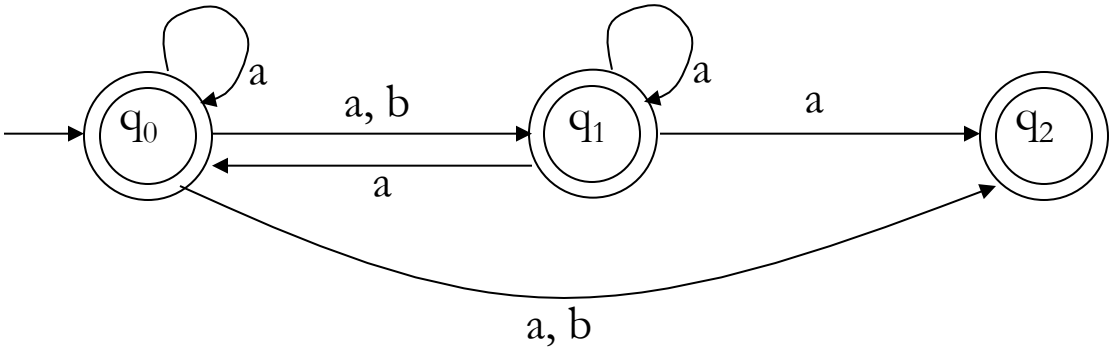
Accepting states of NFA: $\{q_0, q_1, q_2\}$

Example of εNFA to NFA conversion

εNFA:



NFA:



General method

- To convert an εNFA to an NFA:
  - States stay the same
  - Start state stays the same
  - The NFA has a transition from q_i to q_j labeled a iff the εNFA has a path from q_i to q_j that contains one transition labeled a and all other transitions labeled ε
  - The accepting states of the NFA are all states that can reach some accepting state of εNFA using only ε-transitions

Why the conversion works

In the original ε-NFA, when given input $a_1 a_2 \ldots a_n$ the automaton goes through a sequence of states:

$$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow \ldots \rightarrow q_m$$

Some ε-transitions may be in the sequence:

$$q_0 \xrightarrow[\varepsilon]{} \ldots \xrightarrow[a_1]{} \xrightarrow[\varepsilon]{} q_{i_1} \xrightarrow[\varepsilon]{} \ldots \xrightarrow[a_2]{} \xrightarrow[\varepsilon]{} q_{i_2} \xrightarrow[\varepsilon]{} \ldots \xrightarrow[\varepsilon]{} q_{i_n}$$

In the new NFA, each sequence of states of the form:

$$q_{i_k} \xrightarrow{\varepsilon} \ldots \xrightarrow{a_{k+1}} \xrightarrow{\varepsilon} q_{i_{k+1}}$$

will be represented by a single transition

$q_{i_k} \xrightarrow{a_{k+1}} q_{i_{k+1}}$ because of the way we construct the NFA.

Proof that the conversion works

- More formally, we have the following invariant for any $k \geq 1$:

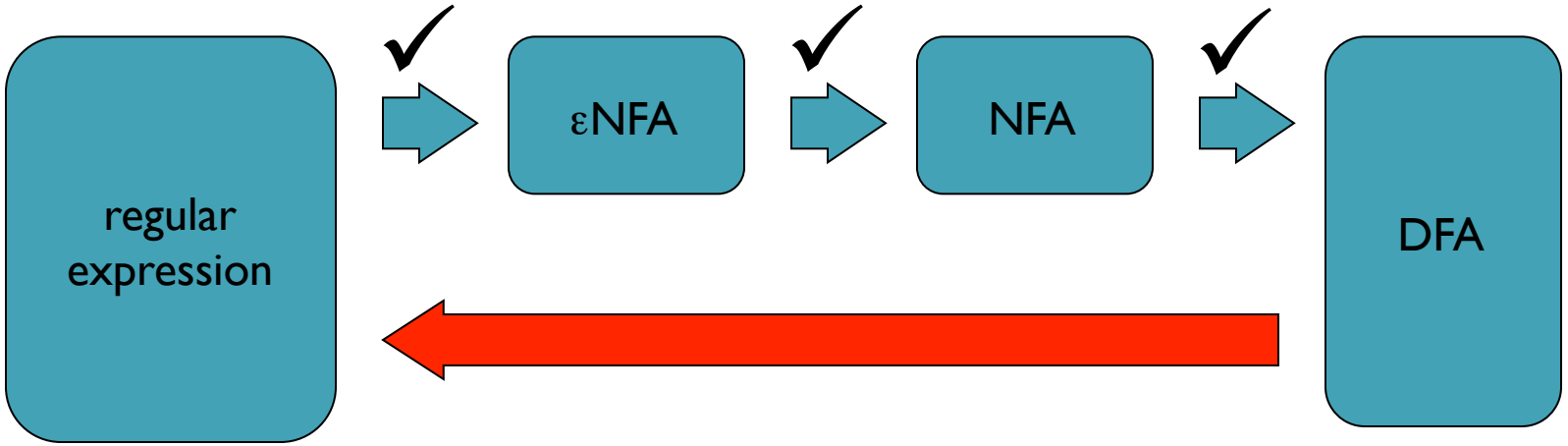
After reading k input symbols, the set of states that the εNFA and NFA can be in are exactly the same

- We prove this by induction on k
- When $k = 0$, the εNFA can be in more states, while the NFA must be in q_0

Proof that the conversion works

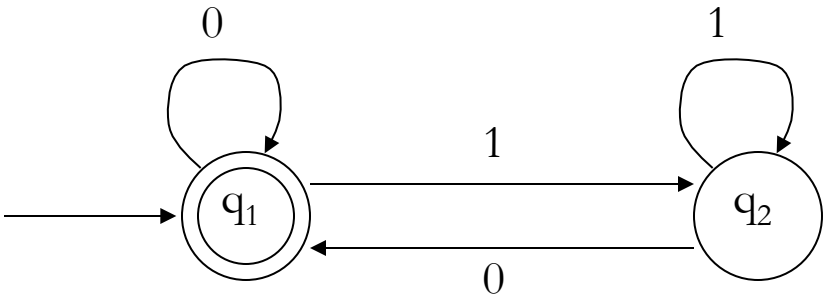
- When $k \geq 1$ (input is not the empty string)
  - If εNFA is in an accepting state, so is NFA
  - Conversely, if NFA is an accepting state q_i, then some accepting state of εNFA is reachable from q_i, so εNFA accepts also
- When $k = 0$ (input is the empty string)
  - The εNFA accepts iff one of its accepting states is reachable from q_0
  - This is true iff q_0 is an accepting state of the NFA

From DFA to regular expressions



Example

- Construct a regular expression for this DFA:



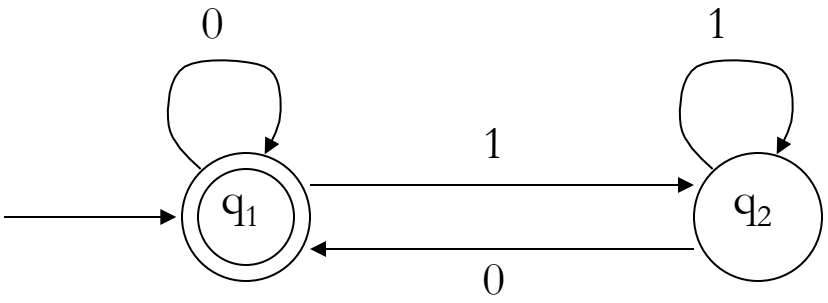
$$(0 + 1)^*0 + \varepsilon$$

General method

- We have a DFA M with states $q_1, q_2, \ldots q_n$
- We will inductively define regular expressions R_{ij}^{k}

R_{ij}^{k} will be the set of all strings that take M from q_i to q_j with intermediate states going through $q_1, q_2, \ldots$ or q_k only.

Example



$$R_{11}^{0} = \{\varepsilon, 0\} = \varepsilon + 0$$

$$R_{12}^{0} = \{1\} = 1$$

$$R_{22}^{0} = \{\varepsilon, 1\} = \varepsilon + 1$$

$$R_{11}^{1} = \{\varepsilon, 0, 00, 000, \ldots\} = 0^*$$

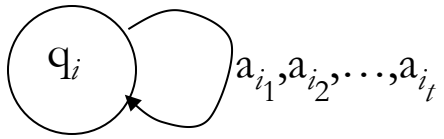
$$R_{12}^{1} = \{1, 01, 001, 0001, \ldots\} = 0^*1$$

General construction

- We inductively define R_{ij}^{k} as:

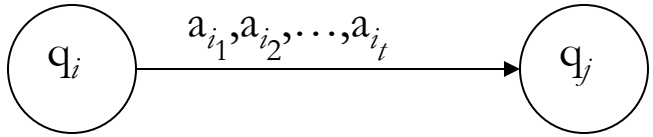
$R_{ii}^{0} = a_{i_1} + a_{i_2} + \ldots + a_{i_t} + \varepsilon$

(all loops around q_i and ε)



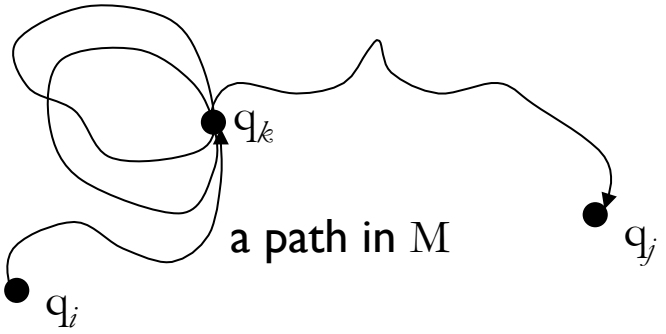
$R_{ij}^{0} = a_{i_1} + a_{i_2} + \ldots + a_{i_t}$ if $i \neq j$

(all $q_i \rightarrow q_j$)



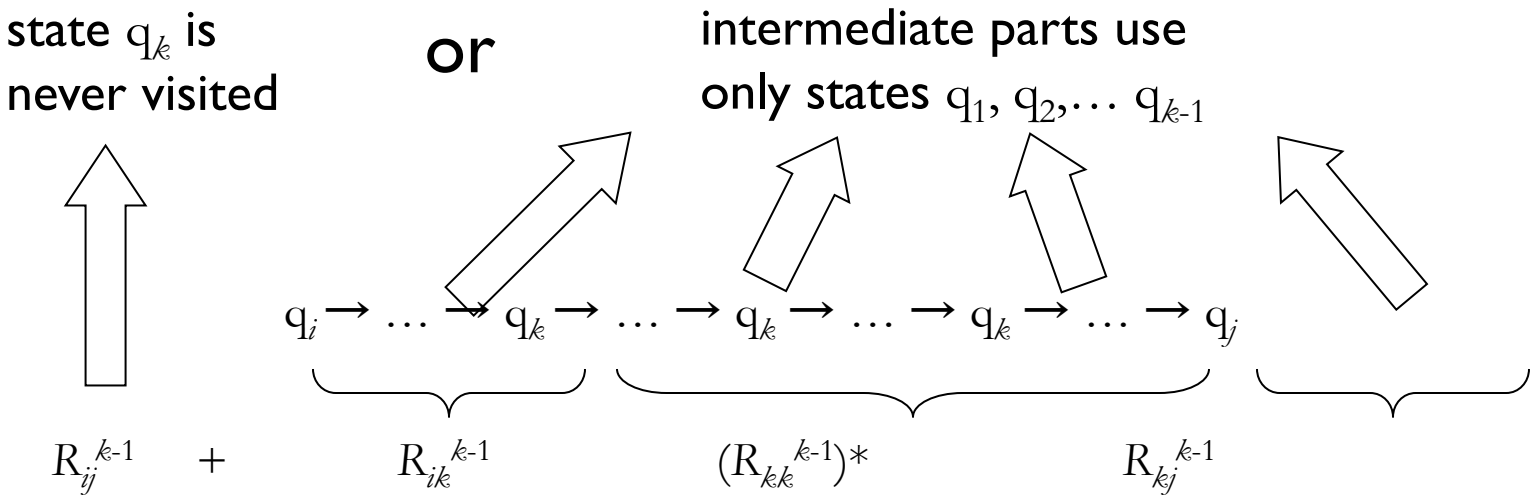
$R_{ij}^{k} = R_{ij}^{k-1} + R_{ik}^{k-1}(R_{kk}^{k-1})^{*}R_{kj}^{k-1}$

(for $k > 0$)



Informal proof of correctness

- Each execution of the DFA using states q_1, q_2, … q_k will look like this:



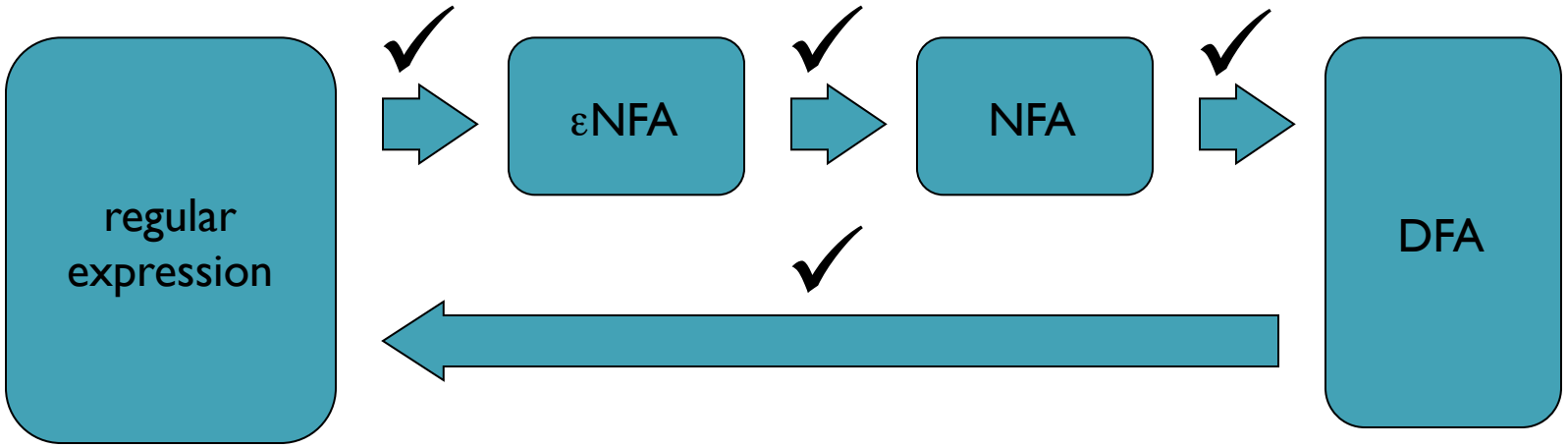
Final step

- Suppose the DFA start state is q_1, and the accepting states are $F = \{q_{j_1} \cup q_{j_2} \ldots \cup q_{j_t}\}$

- Then the regular expression for this DFA is

$$R_{1j_1}{}^{n} + R_{1j_2}{}^{n} + \ldots\ldots + R_{1j_t}{}^{n}$$

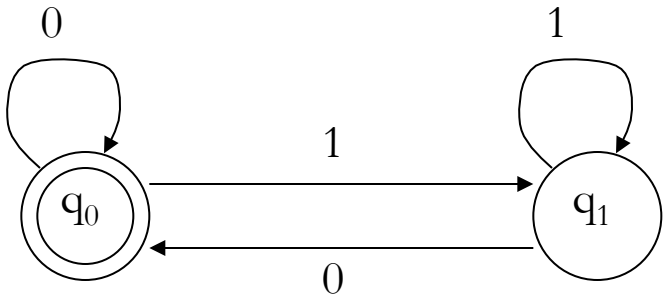
All models are equivalent



A language is regular iff it is accepted by a DFA, NFA, εNFA, or regular expression

Example

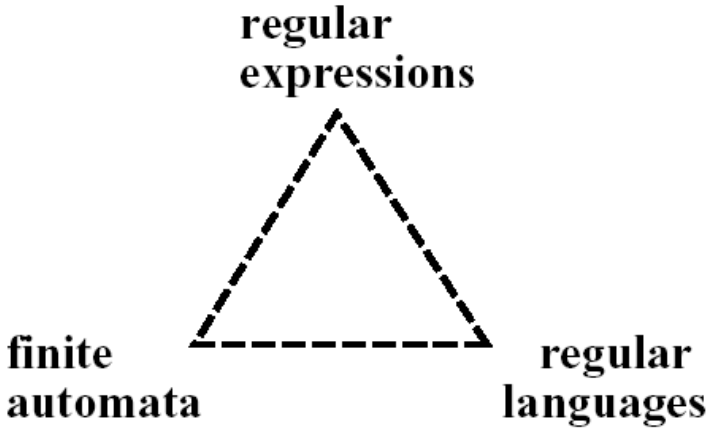
- Give a RE for the following DFA using this method:



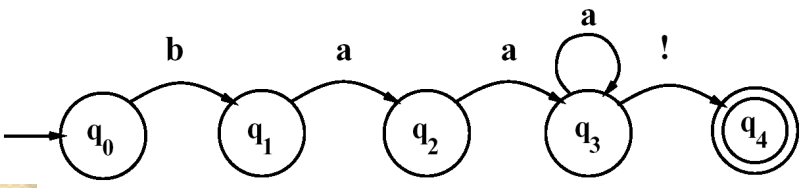
$$(0 + 1)^*0 + \varepsilon$$

2.2 Finite-State Automata

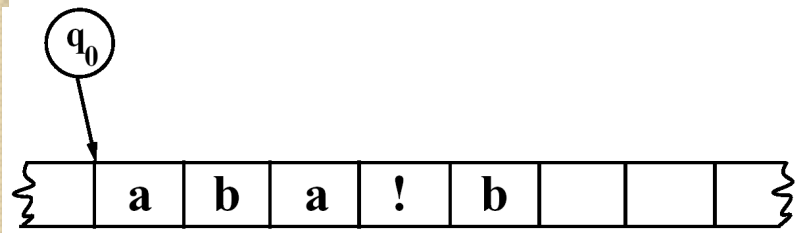
- An RE is one way of describing a FSA.
- An RE is one way of characterizing a particular kind of formal language called a **regular language**.



2.2 Finite-State Automata
Using an FSA to Recognize Sheeptalk



/baa+!/



A tape with cells

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

The transition-state table

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

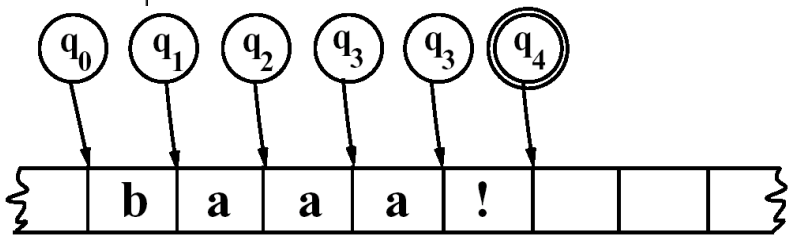
2.2 Finite-State Automata
Using an FSA to Recognize Sheeptalk

- 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;

2.2 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
```

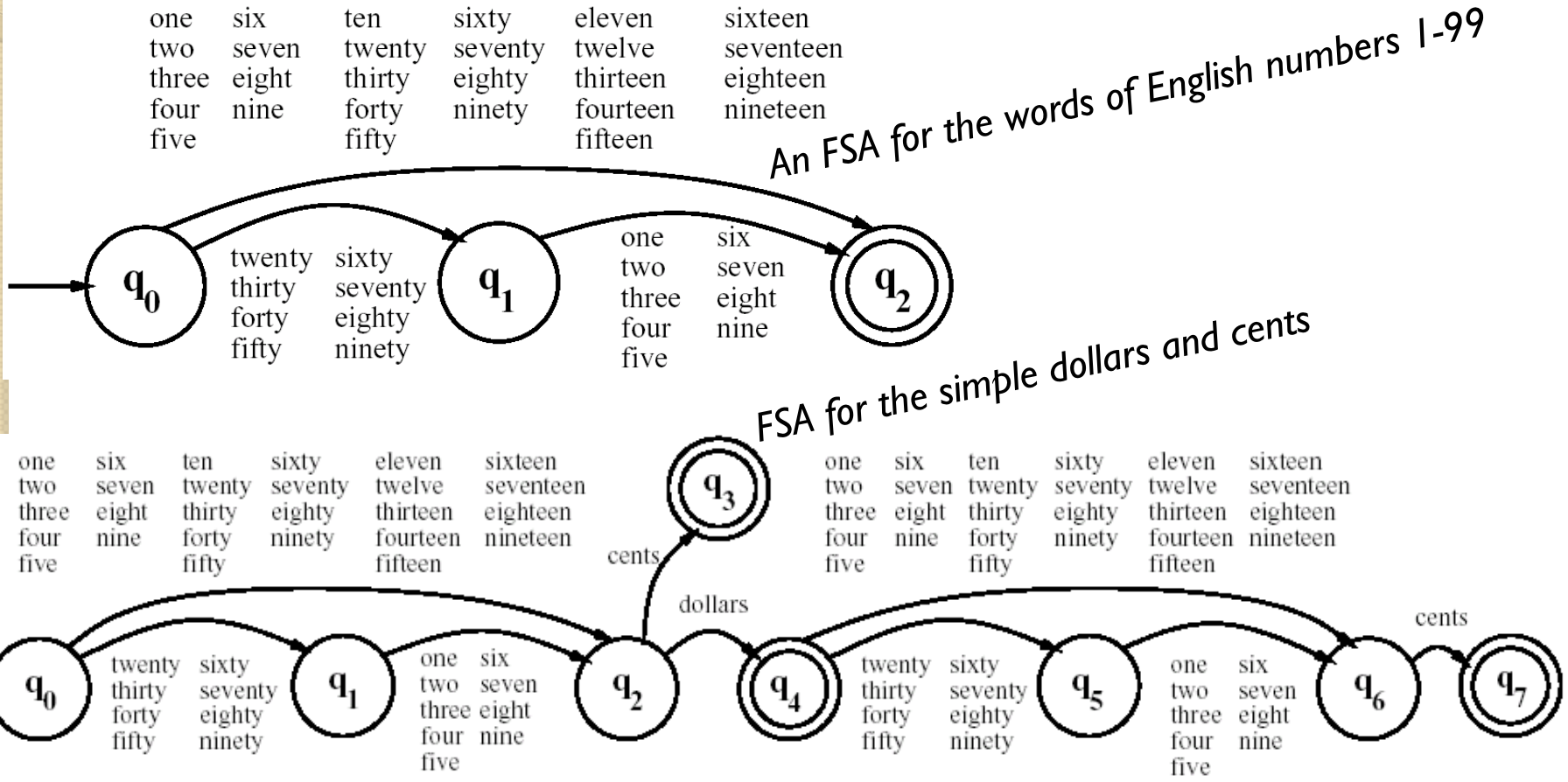


2.2 Finite-State Automata

Formal Languages

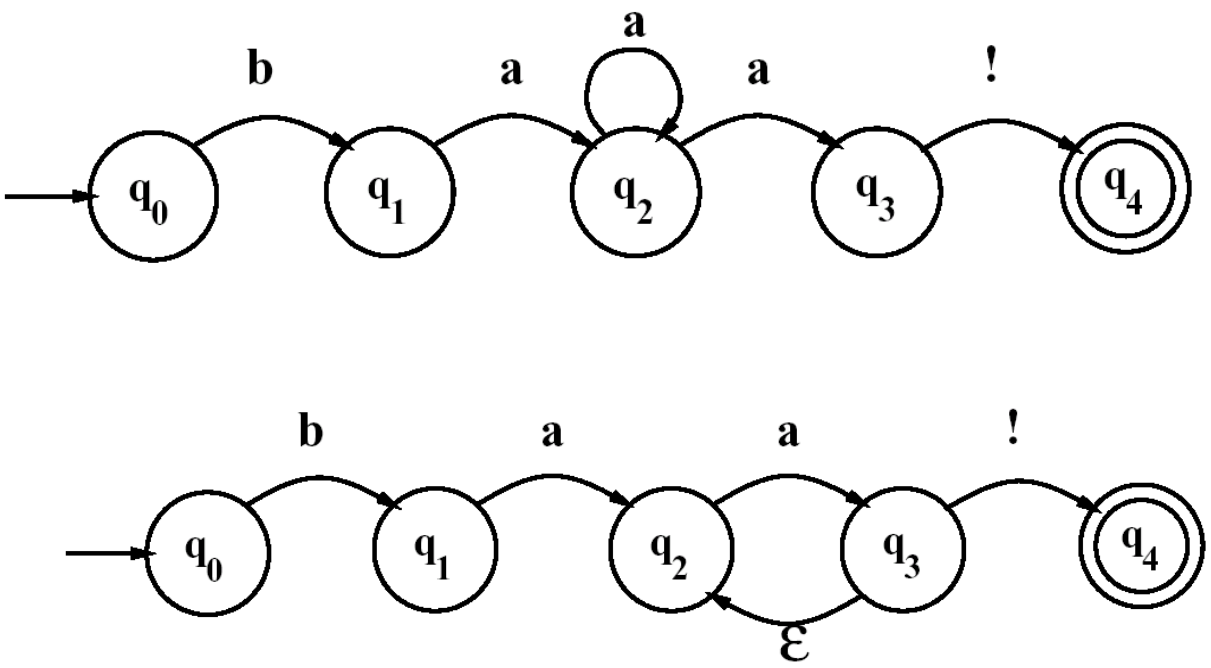
- **Key concept #1: Formal Language:** A model which can both generate and recognize all and only the strings of a formal language acts as a *definition* of the formal language.
- A **formal language** is a set of strings, each string composed of symbols from a finite symbol-set call an **alphabet**.
- 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.

2.2 Finite-State Automata
Another Example



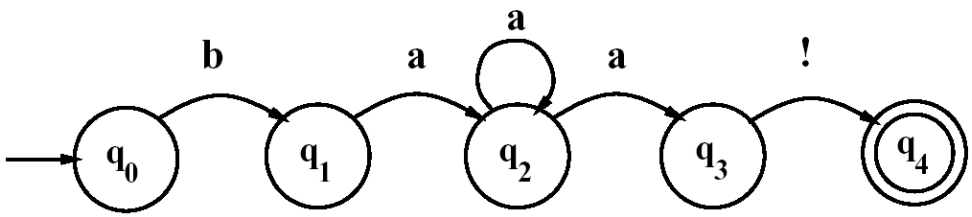
2.2 Finite-State Automata

Non-Deterministic FSAs



2.2 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:	∅	∅	∅	∅

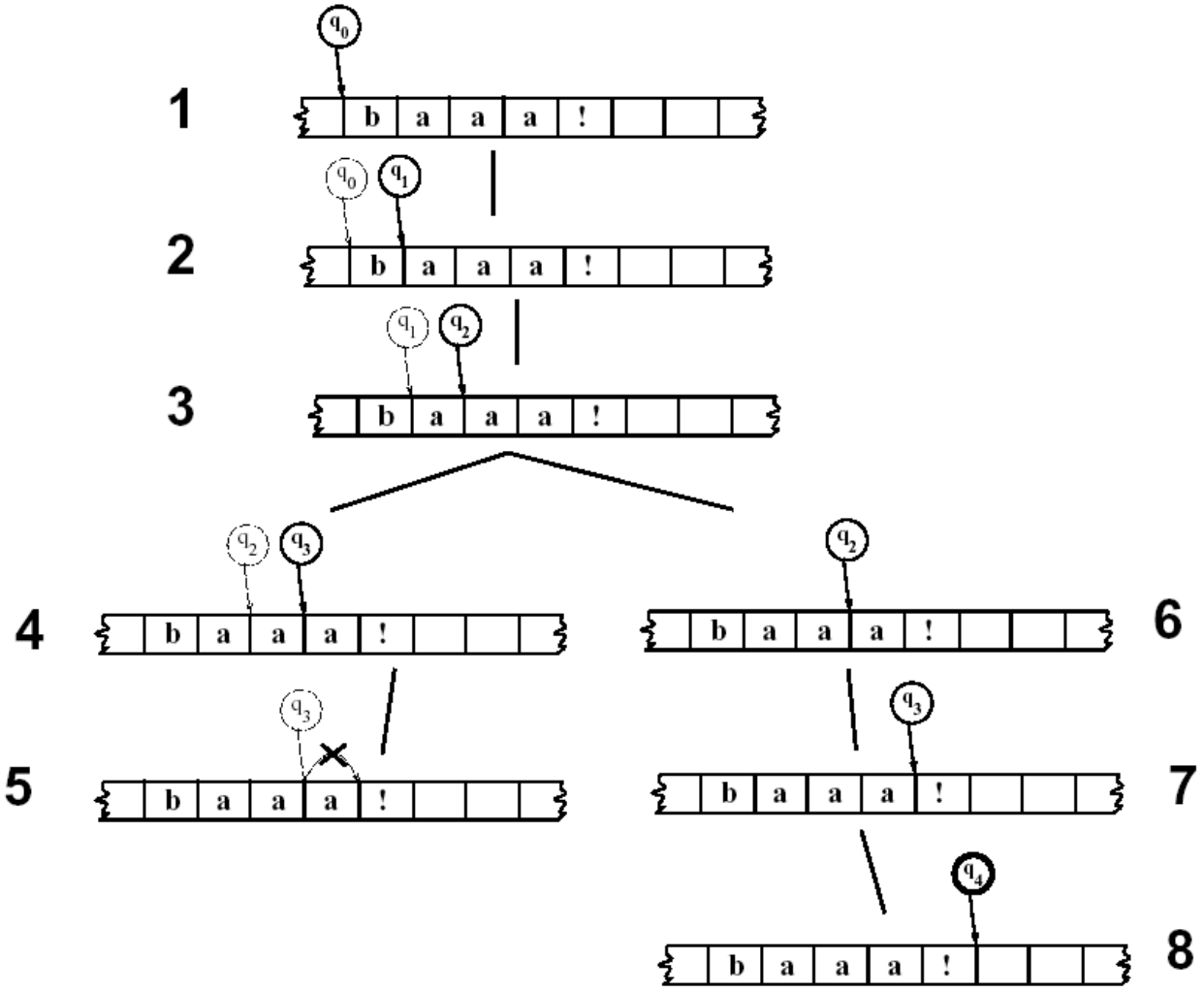
Finite-State Automata

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
```

2.2 Finite-State Automata

Using an NFSA to Accept Strings



2.2 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***