Regular Expressions and Automata

Berlin Chen 2003

References:

1. Speech and Language Processing, chapter 2

Introduction

- Regular Expressions (REs)
- Finite-State Automata (FSAs)
- Formal Languages
- Deterministic vs. Nondeterministic FSAs
- Concatenation and union of FSAs
- Finite-State Transducers (FSTs)
- FSTs for Morphology Parsing
- Probabilistic FSTs

Regular Expressions (REs)

- First developed by Kleene in 1956
- Definition
  - A formula in *a special (meta-) language* that is used for specifying simple classes of strings
    - A string is any sequence of alphanumeric characters (letters, numbers, spaces, tabs, and punctuation)
    - Are ***case sensitive***
  - *An algebraic notation* for characterizing a set of strings
    - Specify search strings in Web IR systems
    - Define a language in a formal way

Basic Regular Expression Patterns

- Regular expression search requires a pattern that we want to search for, and a corpus of texts to search through
  - Search through the corpus returning all texts (all matches or only the first match) contain the pattern (returning the line of document)

RE	Example Patterns Matched
/woodchucks/	"interesting links to woodchucks and lemurs"
/a/	"Mary Ann stopped by Mona's"
/Chaire␣says,/	"Dagmar, my gift please, Chaire says, "
/song/	"All our pretty songs"
/!/	"You've left the burglar behind again !" said Nori

Basic Regular Expression Patterns

- Square braces [and]
  - The string of characters inside the **braces** 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"

- Dash (-) specifies any one character in 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"

Basic Regular Expression Patterns

- Caret (^) specifies what a single character cannot be in the square braces

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"

- Question-mark (?) specify zero or one instances of the previous character

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

Basic Regular Expression Patterns

- Kleene star (*) means zero or more occurrences of the immediately previous character or regular expression
  - E.g.: the sheep language /baaa*!/
  - Multiple digits /[0-9][0-9]*/

baa!
baaa!
baaaa!
baaaaa!
baaaaaa!
….

- Kleene + (+) means one or more occurrences of the immediately previous character or regular expression
  - E.g.: the sheep language /baa+!/
  - Multiple digits /[0-9]+/

Basic Regular Expression Patterns

- Period (.) is used as a wildcard expression that matches any single character (except a carriage return)

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

  - Often used together with Kleene star (*) to specify any string of characters
    - E.g.: find line in which a particular word appears twice
      /aardvark.* aardvark/

Basic Regular Expression Patterns

- Anchors are special characters that anchor regular expressions to particular places in a string
  - The caret (^) also can be used to match the start of a line
    - Three usages of the caret: to match the start of a line, negation inside of square braces, and just to mean caret
  - The dollar sign ($) match the end of a line
  - (\b) matches a word boundary while (\B) matches a non-boundary
  - E.g. :/^The dog\.$/ matches a line contains only the phrase The dog.

Disjunction

- The pipe symbol (|) specifies the disjunction operation
  - E.g.: match either cat or dog
    /cat|dog/
  - Specify singular and plural nouns
    /gupp(y|ies)/

Precedence

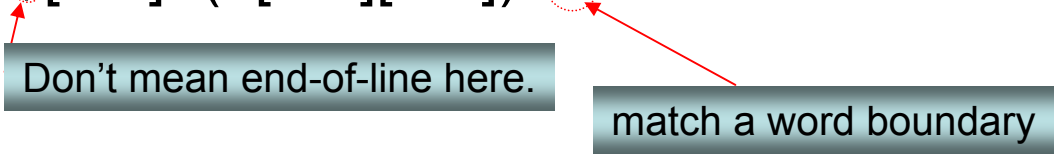
- Operator precedence hierarchy

Parenthesis	( )
Counters	* + ? {}
Sequences and anchors	the ^my end$
Disjunction	|

A More Complex Example

- Example: Deal with prices, $199, $199.99, etc., with decimal point and two digits afterwards

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



- Example: Deal with processor speed (in MHz or GHz), disk space (in Gb) ,or memory size (in Mb or Gb)

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

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

Advanced Operators

- Useful aliases for common ranges

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 expression 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

Characters 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	

Substitution and Memory

- Substitution operator *s/regexp1/regexp2/* allow a string characterized by one regular expression to be replaced by a string characterized by a different one

  s/colour/color/

  - Refer to a particular subpart of the string matching the first pattern, e.g., put angle brackets around all integers in a text (using parenthesis and number operators)

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

  - Specify a certain string or expression occurs twice in the text (the Xer they were, the Xer they will be)

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

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

"register"

The memory feature

Substitution and Memory

- *Substitution using memory* is not part of every regular expression language and is often considered as “extended” feature of regular expressions

- *Substitution using memory* is very useful in implementing simple natural understanding systems

Example: ELIZA

1966

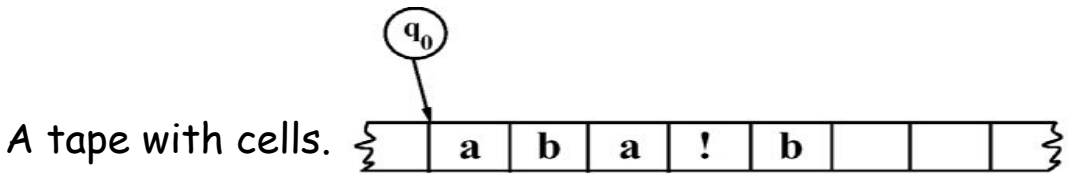
- A simple natural-language understanding program

User1: Men are all alike.
ELIZA1: IN WHAT WAY
User2: They're always bugging us about something or other.
ELIZA2: CAN YOU THINK OF A SPECIAL EXAMPLE
User3: Well, my boyfriend made me come here.
ELIZA3: YOUR BOYFRIEND MADE YOU COME HERE
User4: He says I'm depressed much of the time.
ELIZA4: I AM SORRY TO HERE YOU ARE DEPRESSED

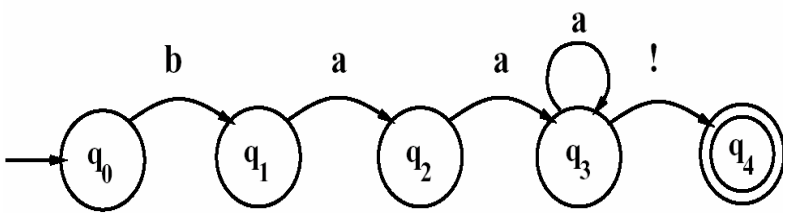
s/.* I'm (depressed|sad) .*/I AM SORRY TO HEAR YOU ARE \1/
s/.* all .*/IN WHAT WAY/
s/.* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE/

Finite-State Automata (FSAs)

- FSA are the theoretical foundation of a good deal of the computational work
  - A directed graph with a finite set of vertices (nodes) as well as arcs (links) between pairs of vertices
  - An FSA can be used for recognizing (accepting) a set of strings (the input written on a long tape)
  - An FSA can be represented with a state-transition-table



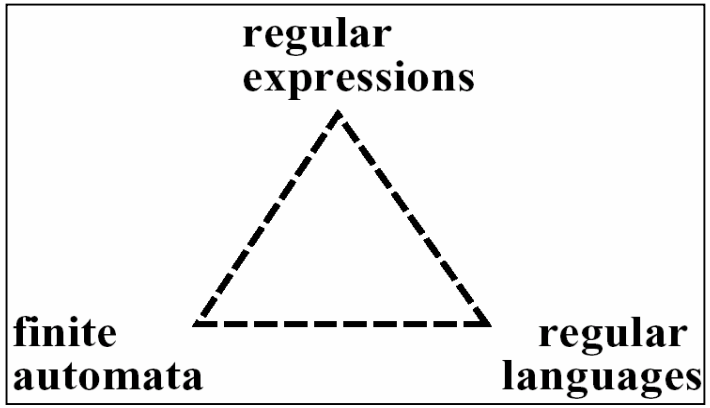
An FSA.



The state-transition table

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

Finite-State Automata (FSAs)



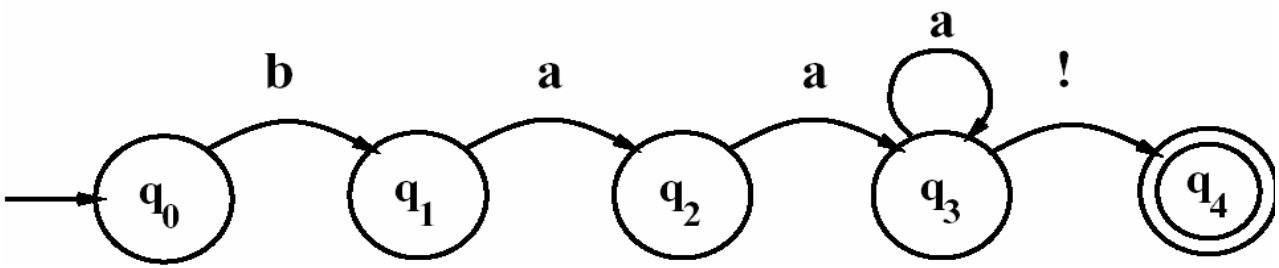
- FSAs and REs
  - Any RE can be implemented as a FSA (except REs with memory feature)
  - Any FSA can be described with a RE (REs can be viewed as a textual way of specifying the structure of FSAs)
  - Both REs and FSAs can be used to describe regular languages

- The main theme in the course
  - Introduce the FSAs for some REs
  - Show how the mapping from REs to FSAs proceeds

Sheep FSA

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

baa!
baaa!
baaaa!
baaaaa!
baaaaaa!
….



Formal Definition of FSAs

- We can specify an FSA by enumerating the following 5 things
  - Q: the set of states, Q={q_0, q_1,... q_N}
  - Σ: a finite alphabet of symobls
  - q_0: a start/initial state
  - F : a set of accept/final states
  - $\delta(q,i)$: a transition function that maps Q×Σ to Q

- Deterministic (FSAs/Recognizers)
  - Has no choice points, the automata/algorithms always know what to do for any input
  - The behavior during recognition is fully determined by the state it is in and the symbol it is looking at

Formal Definition of FSAs

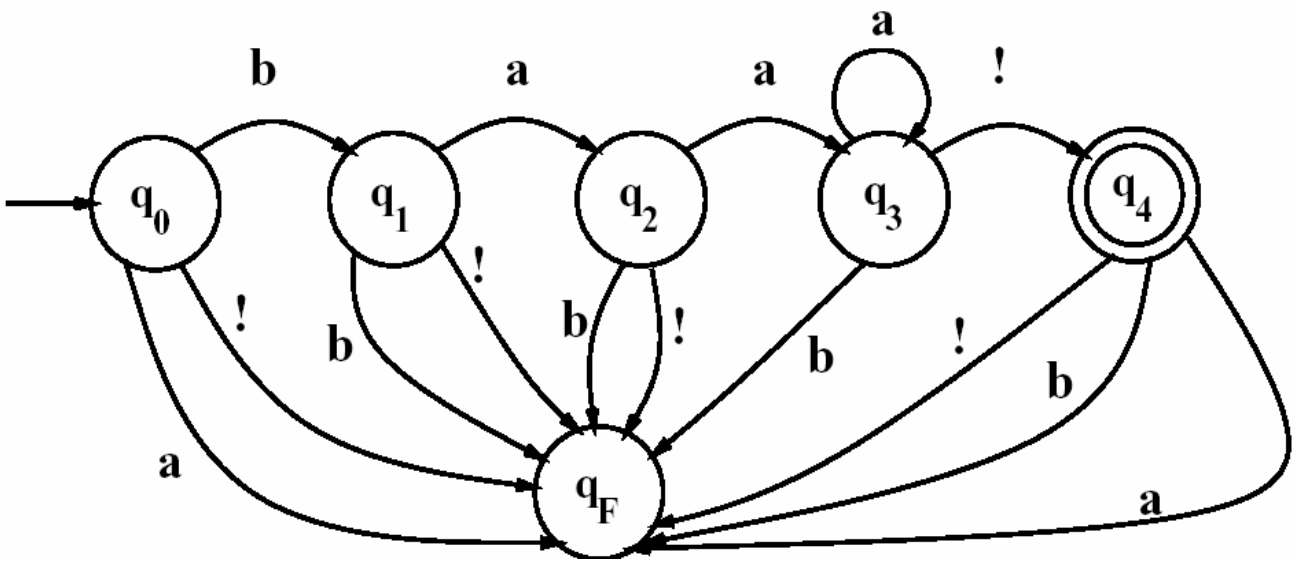
- What is "*recognition*"
  - The process of determining if a string should be accepted by a machine
  - Or, it is the process of determining if a string is in the language defined with the machine
  - Or, it is the process of determining if a regular expression matches a string
- The *recognition* process
  - Simply a process of starting in the start state
  - Examine the current input
  - Consult the table
  - Go to a new state and updating the tape pointer
  - Continue until you run out of tape

Algorithm for Deterministic 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
```

Adding a Fail State to the FSA



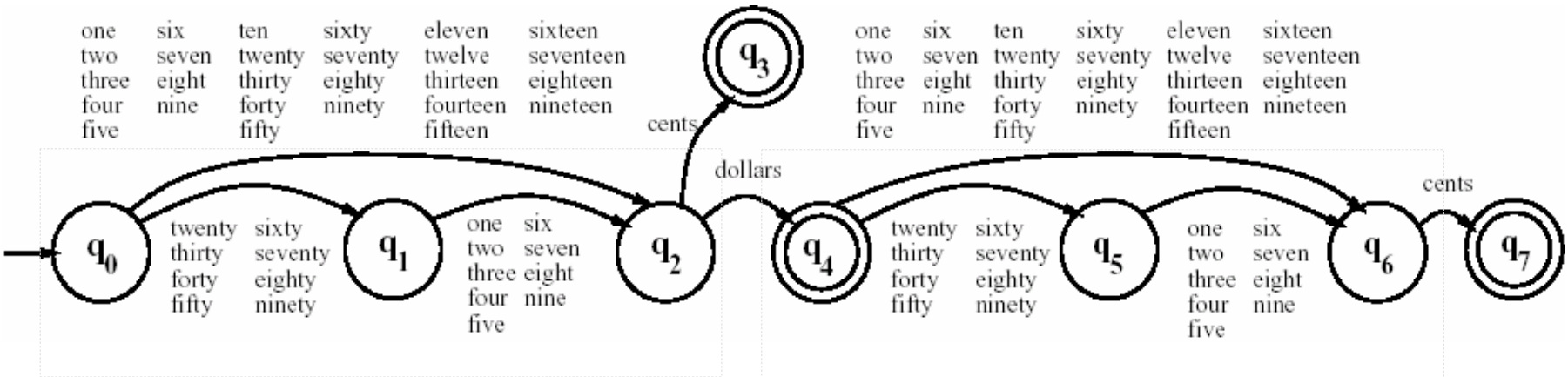
The fail/sink state.

Formal Languages

- Sets of strings composed of symbols from a finite-set (alphabet) and permitted by the rules of formation
- A model (e.g. FSA) which can both *generate* and *recognize* (accept) all and only the strings of a formal language
  - A definition of the formation language (without having to enumerating all strings in the language)
  - Given a model m, we can use $L(m)$ to mean "the formal language characterized by *m*"
  - The formal language defined by the sheeptalk FSA *m*
    $L(m)$={*baa!, baaa!,baaaa!, baaaaa!,….*}
- Often use formal languages to model phonology, morphology, or syntax, …

FSA Dealing with Dollars and Cents

- Such a formal language would model the subset of English



Account for number from 1 to 99.

Account for number from 1 to 99.

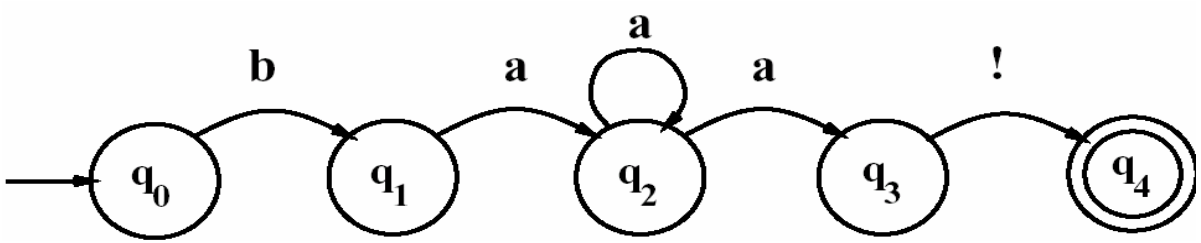
Two Perspectives for FSAs

- FSAs are *acceptors* that can tell you if a string is in the language
  - **Parsing**: find the structure in the string
- FSAs are *generators* to produce all and only the strings in the language
  - **Production/generation**: produce a surface form

Non-Deterministic FSAs

- Non-Deterministic FSAs: NFSAs
- **Recall**
  - "Deterministic" means the behavior during **recognition** is fully determined by the state it is in and the symbol it is looking at

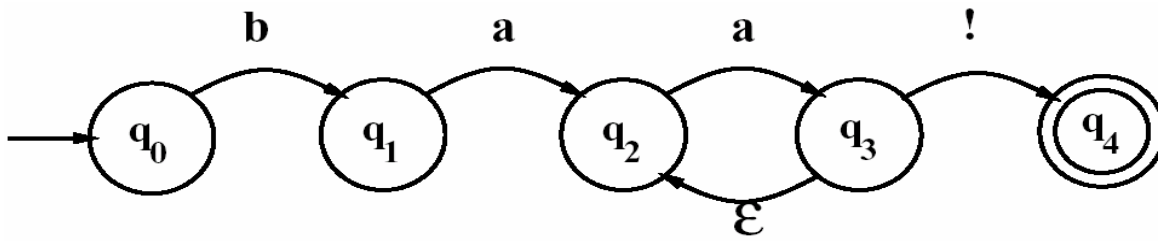
- E.g.: non-deterministic FSAs for the sheeptalk



	Input			
State	b	a	!	ε
0	1	Ø	Ø	Ø
1	Ø	2	Ø	Ø
2	Ø	2,3	Ø	Ø
3	Ø	Ø	4	Ø
4:	Ø	Ø	Ø	Ø

Non-Deterministic FSAs

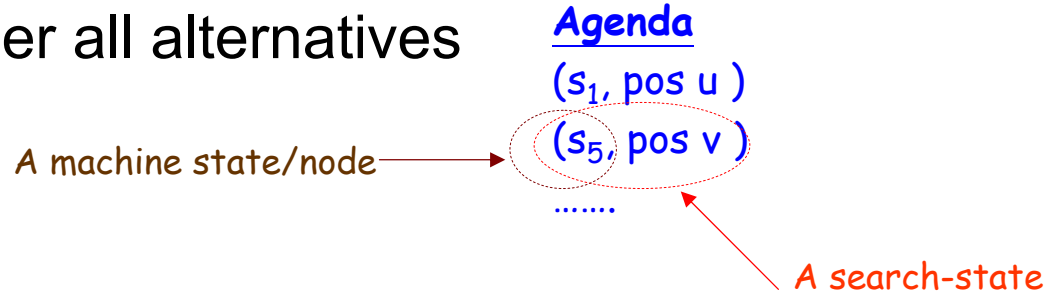
- With ε transitions
  - Arcs that have no symbols on them
    - Move without looking at the input



- When NFSAs take a wrong choice
  - Follow the wrong arc and reject the input when we should have accepted it
    - E.g. when input is "baa!"

Solutions for Wrong Choices

- **Backup**
  - When at a choice point, put a marker (current state, current position at the input tape) and unexplored choices on the agenda
    - Remember all alternatives

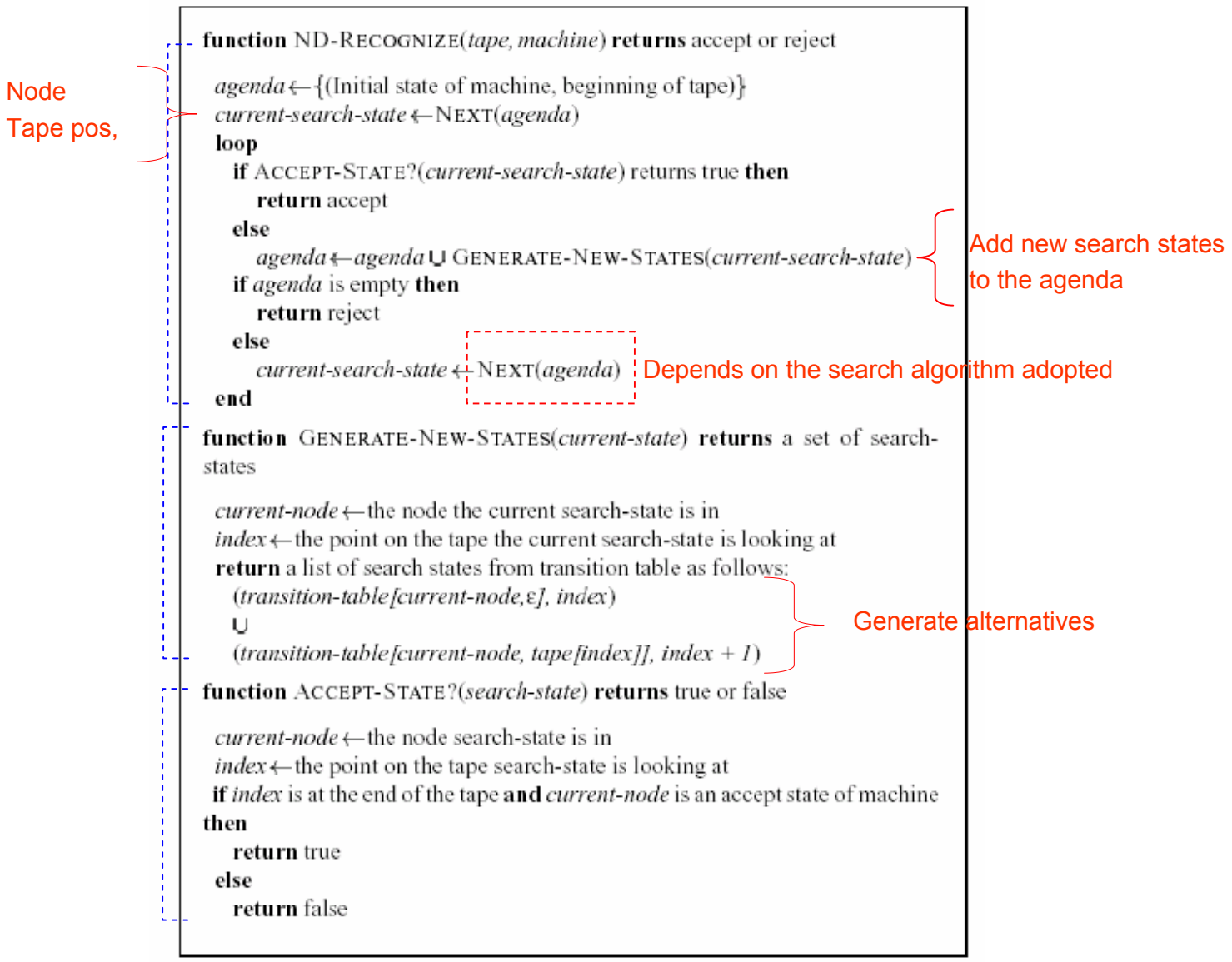


- **Look-ahead**
  - We could look ahead in the input to help us decide which path to take

Discussed later

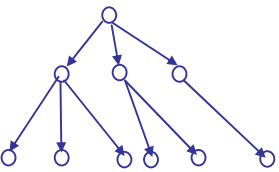
- **Parallelism**
  - When at a choice point, we could look at every alternative path in parallel

Algorithm for Non-Deterministic FSAs



Algorithm for Non-Deterministic FSAs

- Implementation of the NEXT function
  - **Depth-first search or Last In First Out** (LIFO)
    - Place the newly created states at the front of the agenda
    - The NEXT returns the state at the front of the agenda



Infinite loop ?

  - **Breadth-first search or First In First Out** (FIFO)
    - Place the newly created states at the back of the agenda
  - **Dynamic programming or A***

Infinite loop ?

Time-synchronous

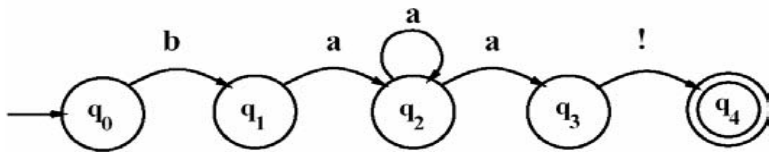
Viterbi/Breadth-first search

Time-asynchronous

Best-first search

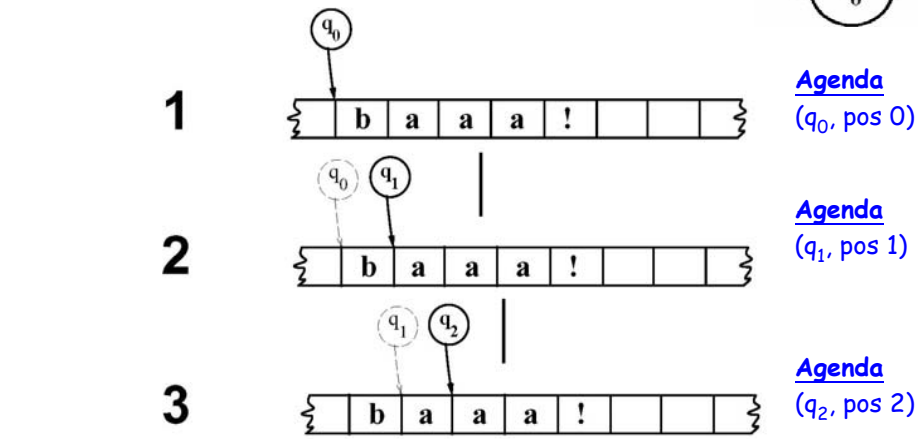
Algorithm for Non-Deterministic FSAs

- Depth-first search



b a a a ! NIL

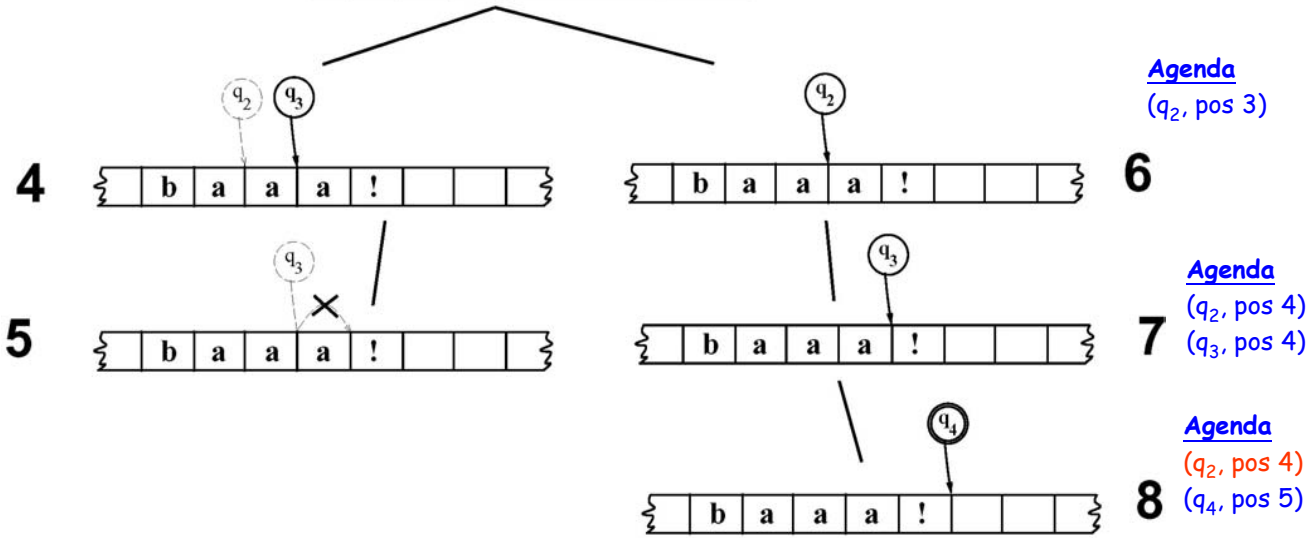
0 1 2 3 4 5



Agenda
(q_0, pos 0)

Agenda
(q_1, pos 1)

Agenda
(q_2, pos 2)



Agenda
(q_2, pos 3)
(q_3, pos 3)

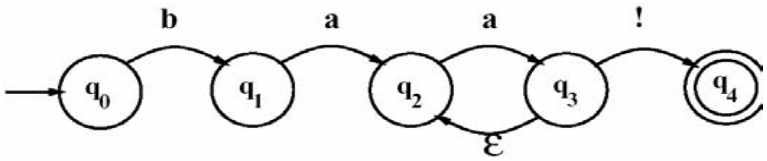
Agenda
(q_2, pos 3)

Agenda
(q_2, pos 4)
(q_3, pos 4)

Agenda
(q_2, pos 4)
(q_4, pos 5)

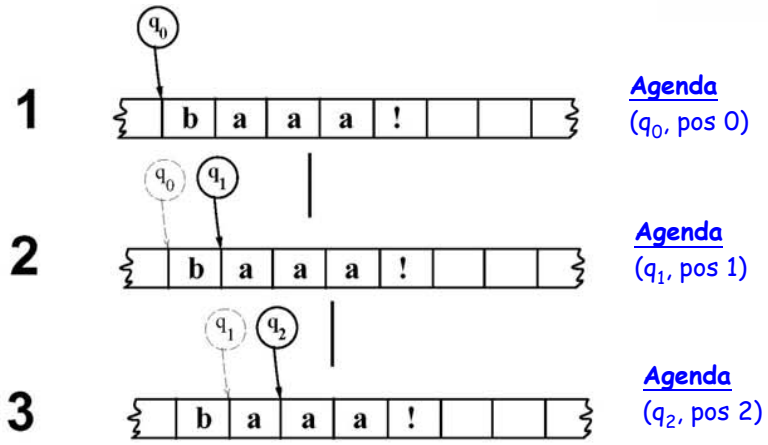
Algorithm for Non-Deterministic FSAs

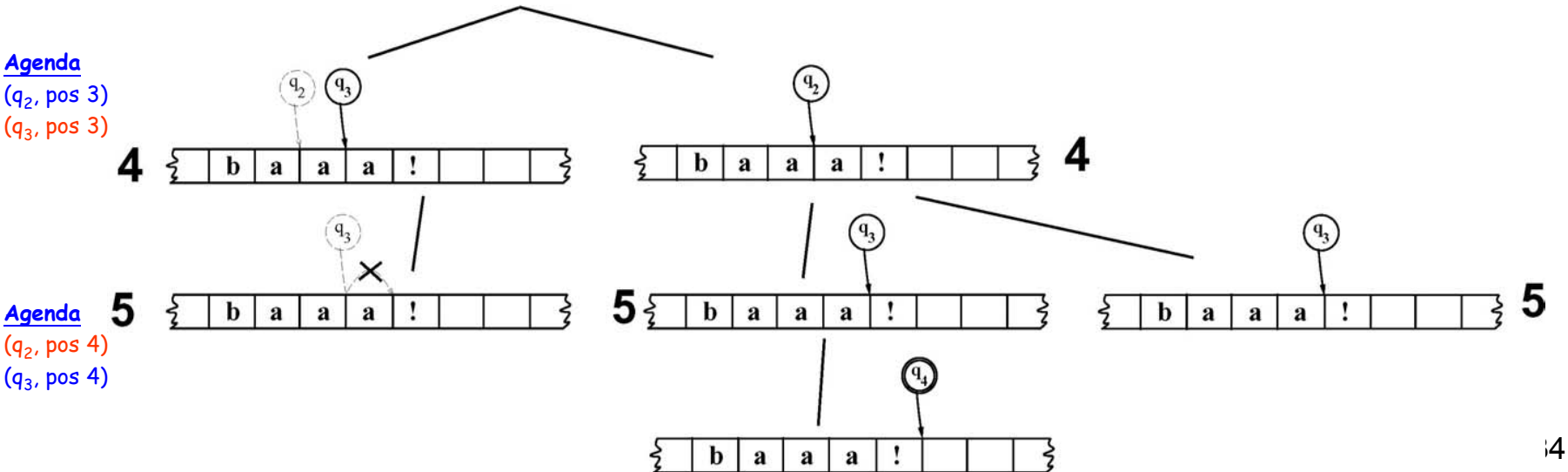
- Breadth-first search



b a a a ! NIL

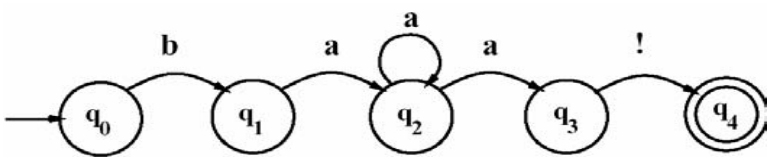
0 1 2 3 4 5



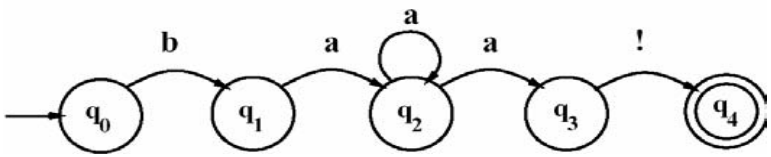


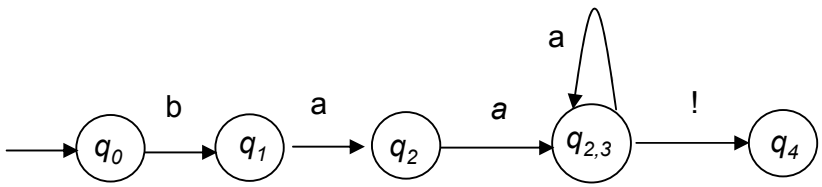
Relating DFSA and NDFSA

- For any NFSA, there is an exactly equivalent DFSA (which has the same power)
  - A simple algorithm for converting an NFSA to an equivalent DFSA
    - E.g. a parallel algorithm traverses the NFSA and groups the states we reach on the same input symbol into an equivalent class and give a new state label to this new equivalent class state
  - The number of states in the equivalent deterministic automaton may be much larger



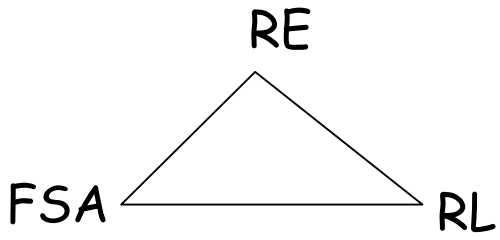
Relating DFSA and NDFSA





Regular Languages and FSAs

- Regular languages
  - The class of languages that are **definable** by regular expressions
  - Or the class of languages that are **characterized** by finite-state automata



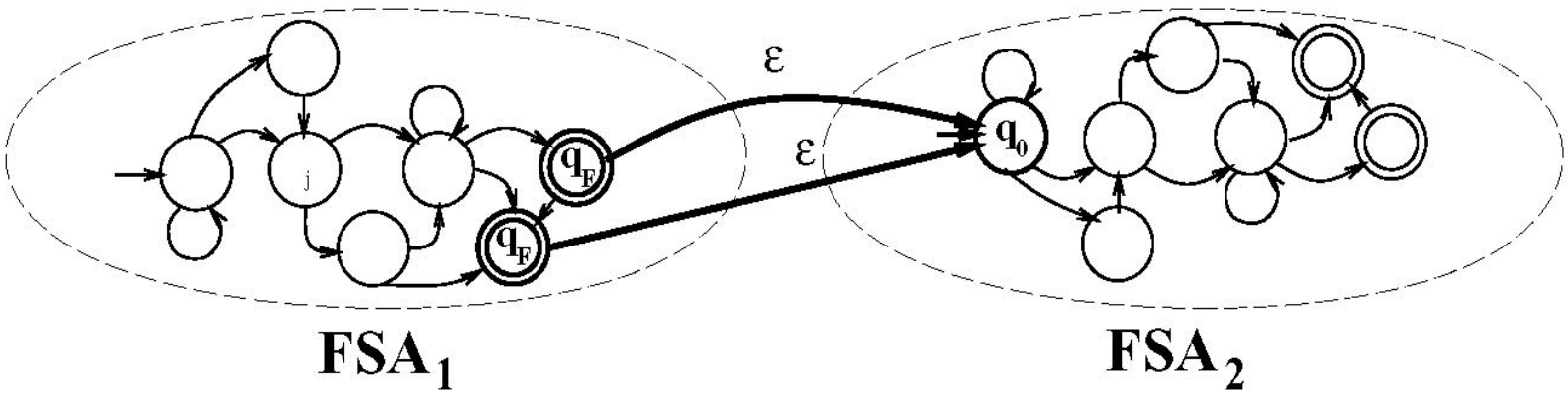
- **Definition of regular language**
  - $\emptyset$ is a primitive regular language
  - $\forall a \in \Sigma \cup \varepsilon,\ \{a\}$ is a primitive regular language
  - If L_1 and L_2 are regular languages, then so are
    - $L_1 \cdot L_2 = \{xy \mid x \in L_1, y \in L_2\}$ the **concatenation** of L_1 and L_2
    - $L_1 \cup L_2$ the **union** or **disjunction** of L_1 and L_2
    - L_1^* the **Kleene closure** of L_1

The Closure of Regular Languages

- Regular languages are closed under the following operations
  - **Interaction**: if L_1 and L_2 are regular languages then so is $L_1 \cap L_2$
  - **Difference**: if L_1 and L_2 are regular languages then so is $L_1 - L_2$
  - **Complementation**: if L_1 is a regular language then so is $\Sigma^* - L_1$
  - **Reversal**: if L_1 is a regular language then so is L_1^R

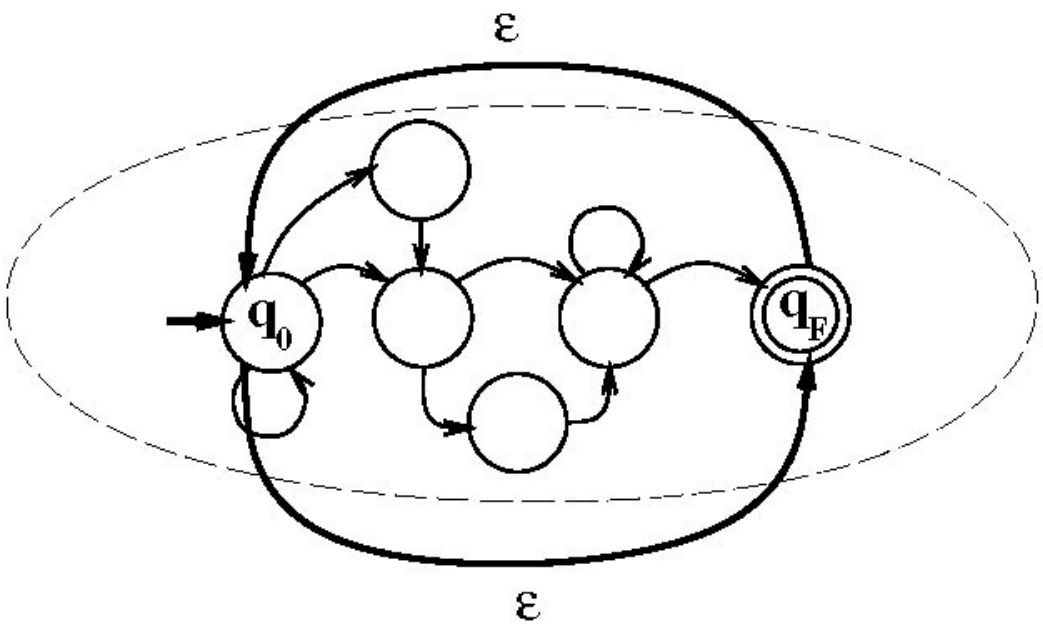
The Concatenation of Two FSAs

- Accept a string consisting of a string from language L_1 followed by a string from language L_2



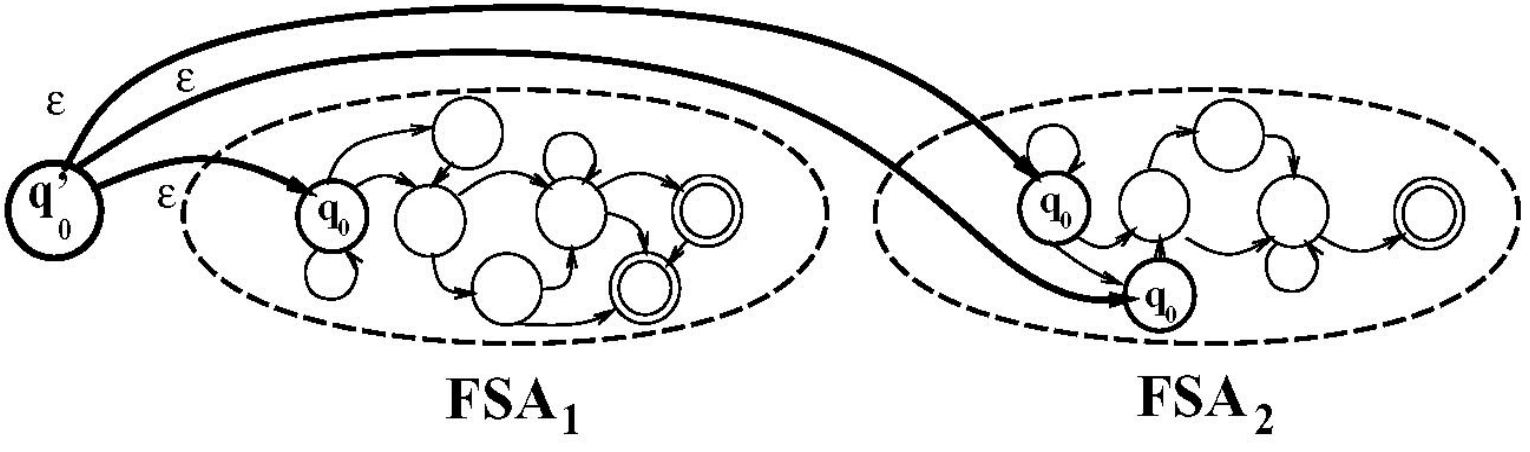
The Kleene * Closure of an FSA

- All final states of the FSA back to the initial states by ε-transitions



The Union of Two FSAs

- Accept a string in either of two languages



Review: English Morphology

- Morphology is the study of the ways that words are built up from smaller meaningful units called morphemes
- Morphemes are divided into two classes
  - Stems: The core meaning bearing units
  - Affixes: Bits and pieces that adhere to stems to change their meanings and grammatical functions
- Two classes of ways to form words from morphemes
  - Inflectional morphology
  - Derivational morphology

Morphology Parsing

- Find the morphology structure of an input (surface) form

Inputs	Morphological Parsed Outputs
cats	cat + N +PL
cat	cat + N + SG
cities	city + N +PL
geese	goose + N +PL
goose	(goose +N +SG) or (goose +V)
gooses	gooses +V +3SG
merging	merge + V + PRES-PART
caught	(catch +V +PAST-PART) or (catch +V + PAST)

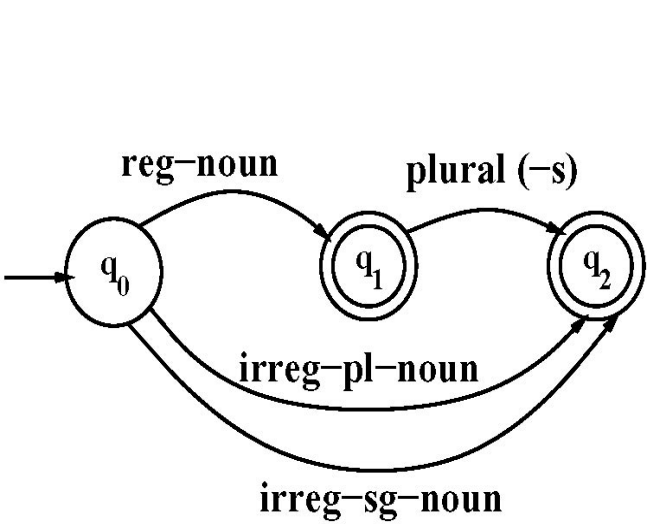
word stems and morphological features

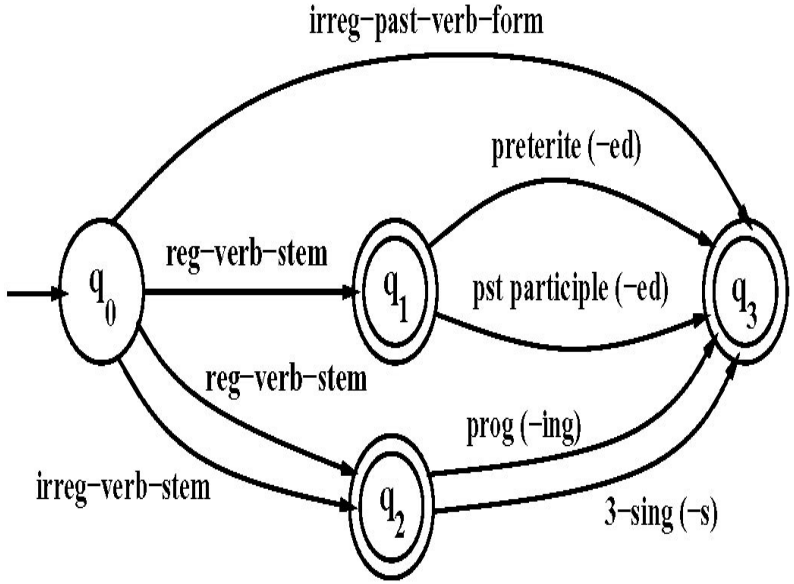
Constituents of Morphology Parser

- Lexicon
  - List of stems and affixes, with basic information about them
  - E.g.: noun/verb stems, etc.
- Morphotactics
  - The model of morpheme ordering
  - E.g.: the rule that English plural morpheme follows the noun rather than preceding it
- Orthographic rules
  - The **spelling rules** used to model the changes that occur in a word, when two morphemes combine
  - E.g: city + -s → cities ("consonant" + "y" → "ie")

FSAs for Morphotatics Knowledge

- An FSA for English nominal/verb inflection

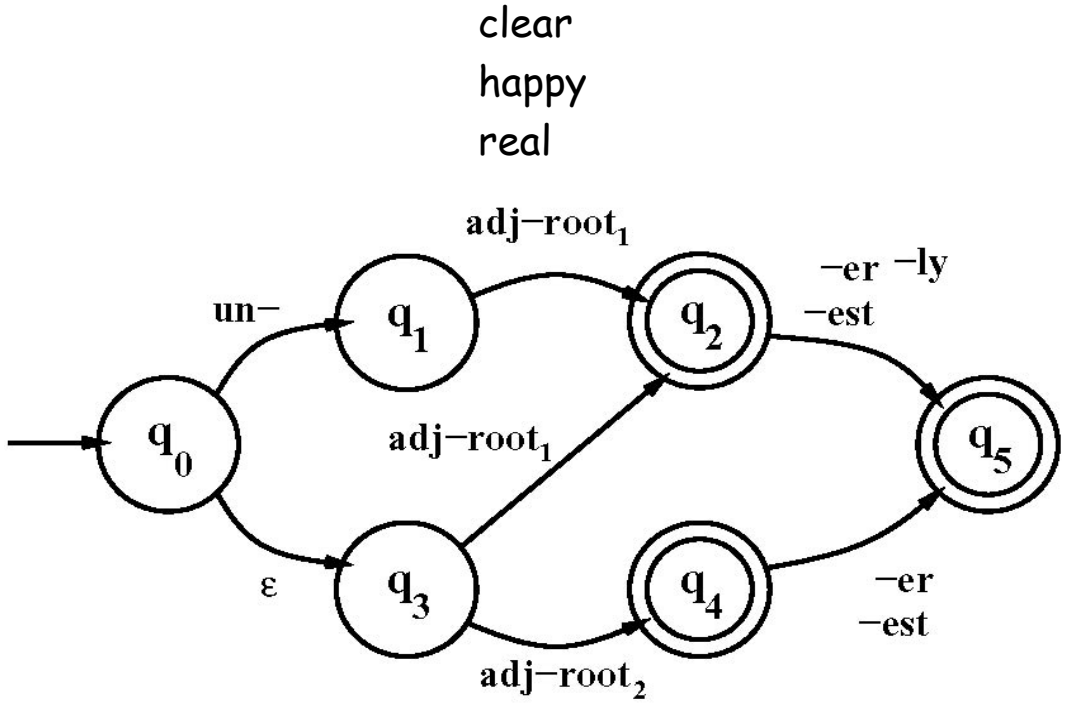




  - Govern the ordering of affixes

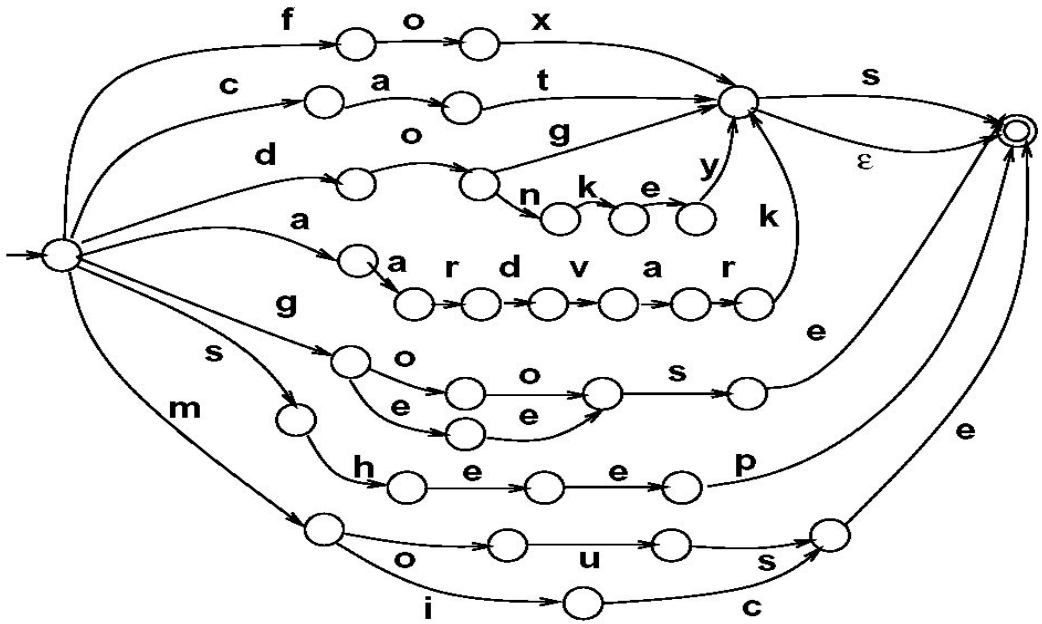
FSAs for Morphotatics Knowledge

- An FSA for English adjective derivation



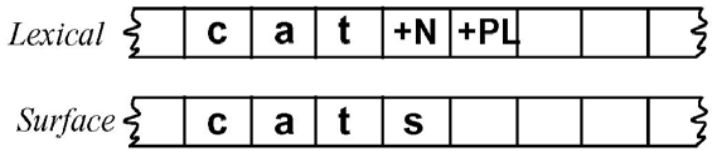
FSAs for Morphological Recognition

- Determine whether an input string of letters makes up a legitimate word
- An FSA for English nominal inflection
  - Plug in "*sub-lexicons*" into the FAS
    - E.g.: the reg-noun-stem, irreg-sg-noun etc.



Finite State Transducer (FST)

- FST has a more general function than FSA
  - FSA defines a formal language by defining a set of strings
  - FST defines **a relation between two set of string**
    - Add another tape
    - Add extra symbols (outputs) to the transitions (the Mealy machine)
    - Read one string and generate another one
      - E.g.: On one tape we read "cats", on the other we write "cat +N +PL (morphology parsing)



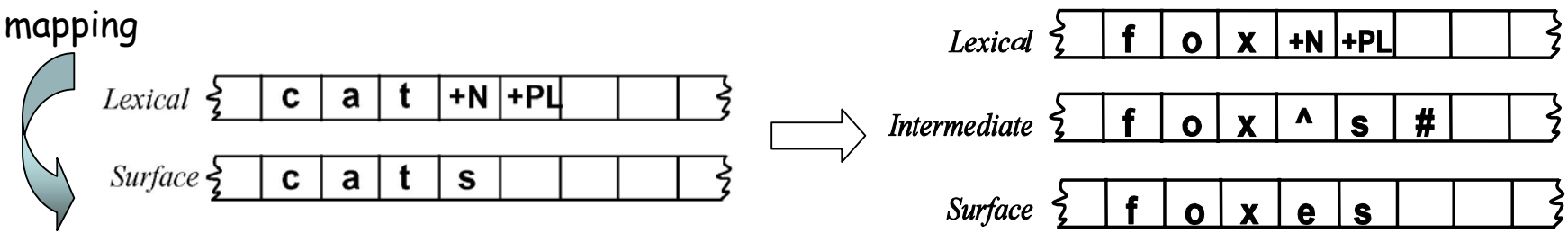
Finite State Transducer

- Formal Definition
  - Q :The set of states, Q={q_0, q_1,... q_N}
  - Σ: a finite alphabet of **complex** symbols, *i*:*o*; *i* from an input alphabet *I*, and o from an output alphabet *O*, both include the epsilon symbol ε
  - q_0: the start state
  - F: the set of accept/final states
  - δ (*q*,*i*:*o*): the transition function that maps Q$\times\Sigma$ to Q

- FST are closed under **union**, but not closed under **difference**, **complement**, and **intersection** (because of epsilon symbol ε , et al.)

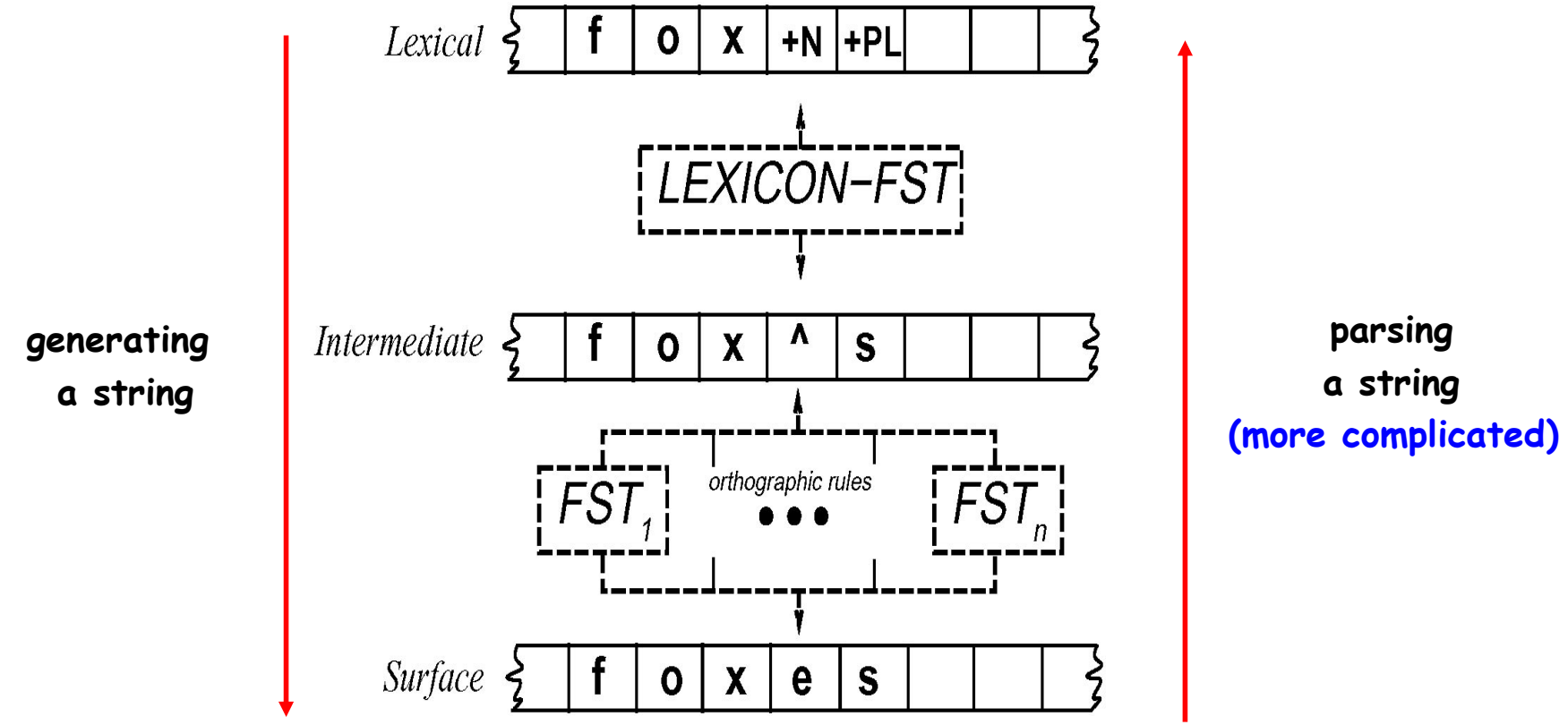
Finite State Transducer

- Two additional closure properties
  - **Inversion**
    - The inversion of a transducer T (T^{-1}) simply switches the input and output labels
    - FST-as-parser $\longleftrightarrow$ FST-as-generator
  - **Composition**
    - If T_1 is a transducer from I_1 to O_1 and T_2 a transducer from I_2 to O_2 then $T_1 \circ T_2$ map from I_1 to O_2



Two-level Morphology System

- Generating and Parsing with FST lexicon and rule



Two-level Morphology System

- Orthographic rules
  - An FST to process a sequence of words
    - #: word boundary

Antworth 1990