Introduction to Computational Linguistics

Pavlina Ivanova

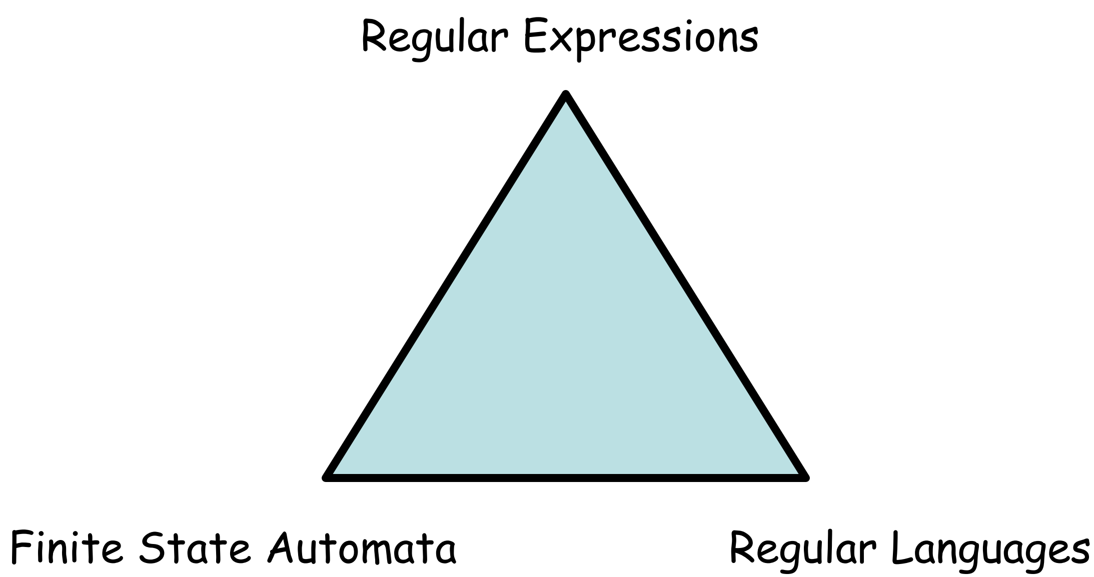
University of Plovdiv, Bulgaria

Lecture 2: Finite-State Automata, Morphology, Morphological Parsing, Transducers, Tokenization

Thanks to Daniel Jurafsky for much of this material

Three equivalent formal constructions

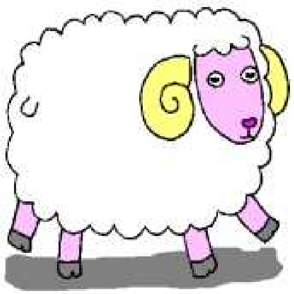
- Any RE (except that use the memory features) can be implemented as a finite-state automation (FSA).
- Any FSA can be described with a RE.
- Both RE and FSA can be used to describe a particular kind of formal language called a regular language.



Finite State Automata

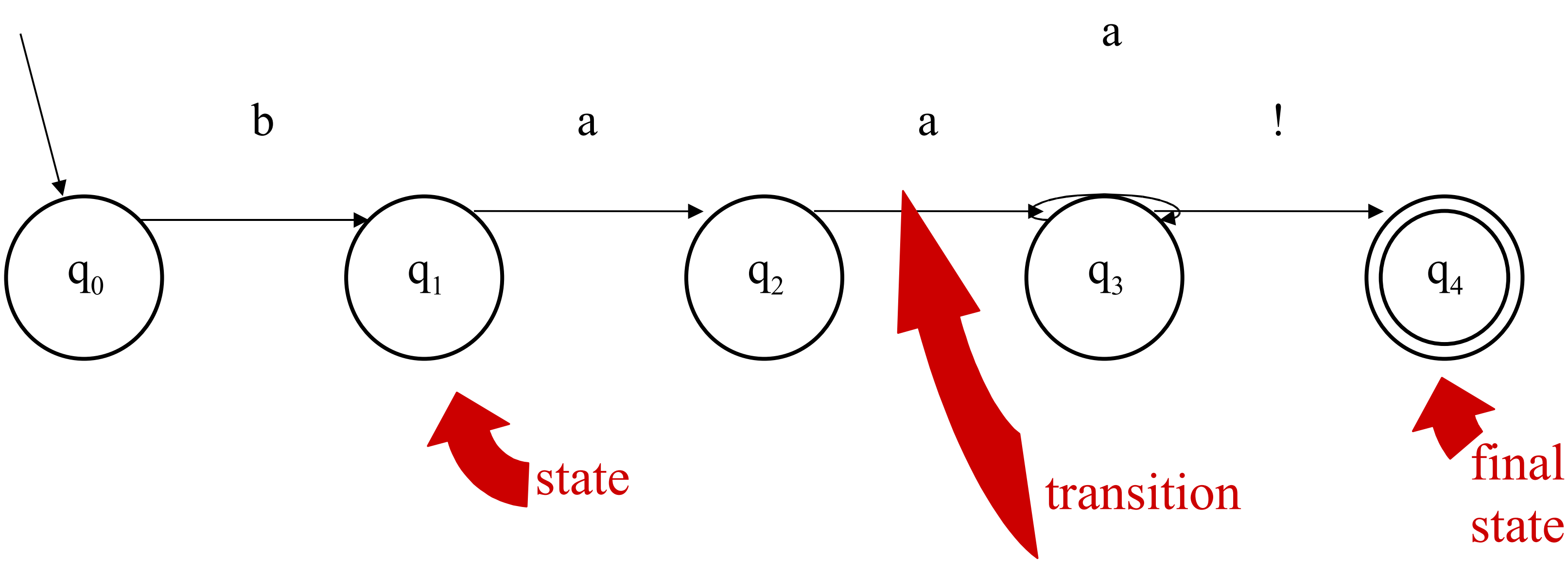
- Terminology: Finite State Automata, Finite State Machines, FSA, Finite Automata
- FSAs and their close relatives are at the core of most algorithms for NLP.
- FSA can be represented as directed graph: a finite set of nodes and labeled directed links between pairs of nodes called arcs.
  - Nodes represent the states
  - Arcs represent the transitions between the states

Finite-state Automata (Machines)



baa!
baaa!
baaaa!
baaaaa!
...

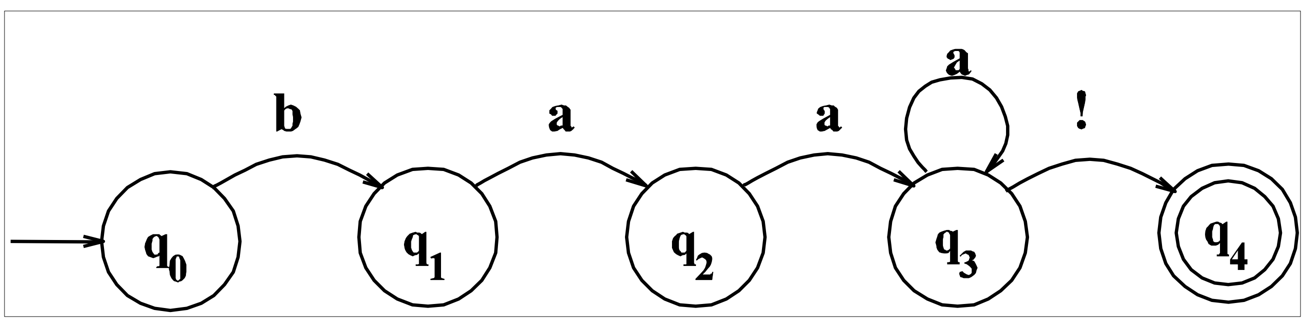
/^baa+!$/



Slide from Dorr/Monz

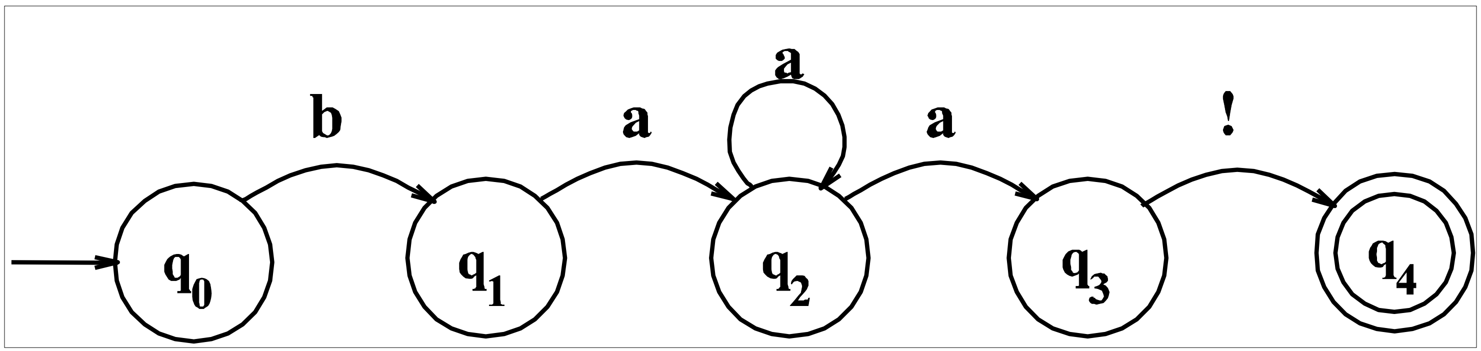
Sheep FSA

- We can say the following things about this machine
  - 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



But note

- There are other machines that correspond to this language



- More on this one later

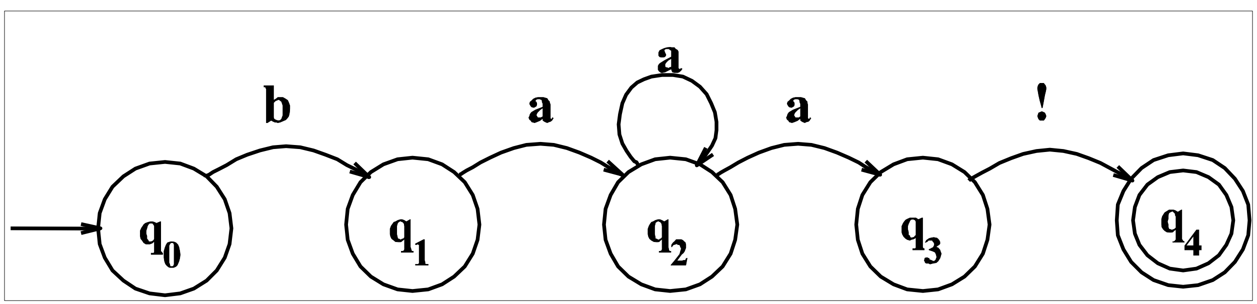
Formal Definition of FSA

- You can specify an FSA by enumerating the following things.
  - The set of states: Q
  - A finite alphabet: Σ
  - A start state $q_0 \in Q$
  - A set F of accepting/final states $F \subseteq Q$
  - A transition function $\delta(q,i)$ that maps $Q x \Sigma$ to Q

Another Representation of the FSA

- State-transition table

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

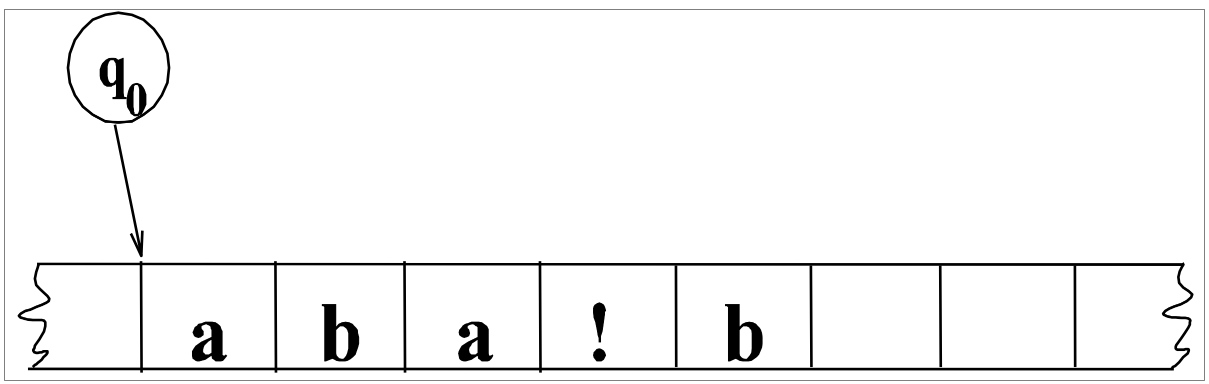


Recognition

- 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

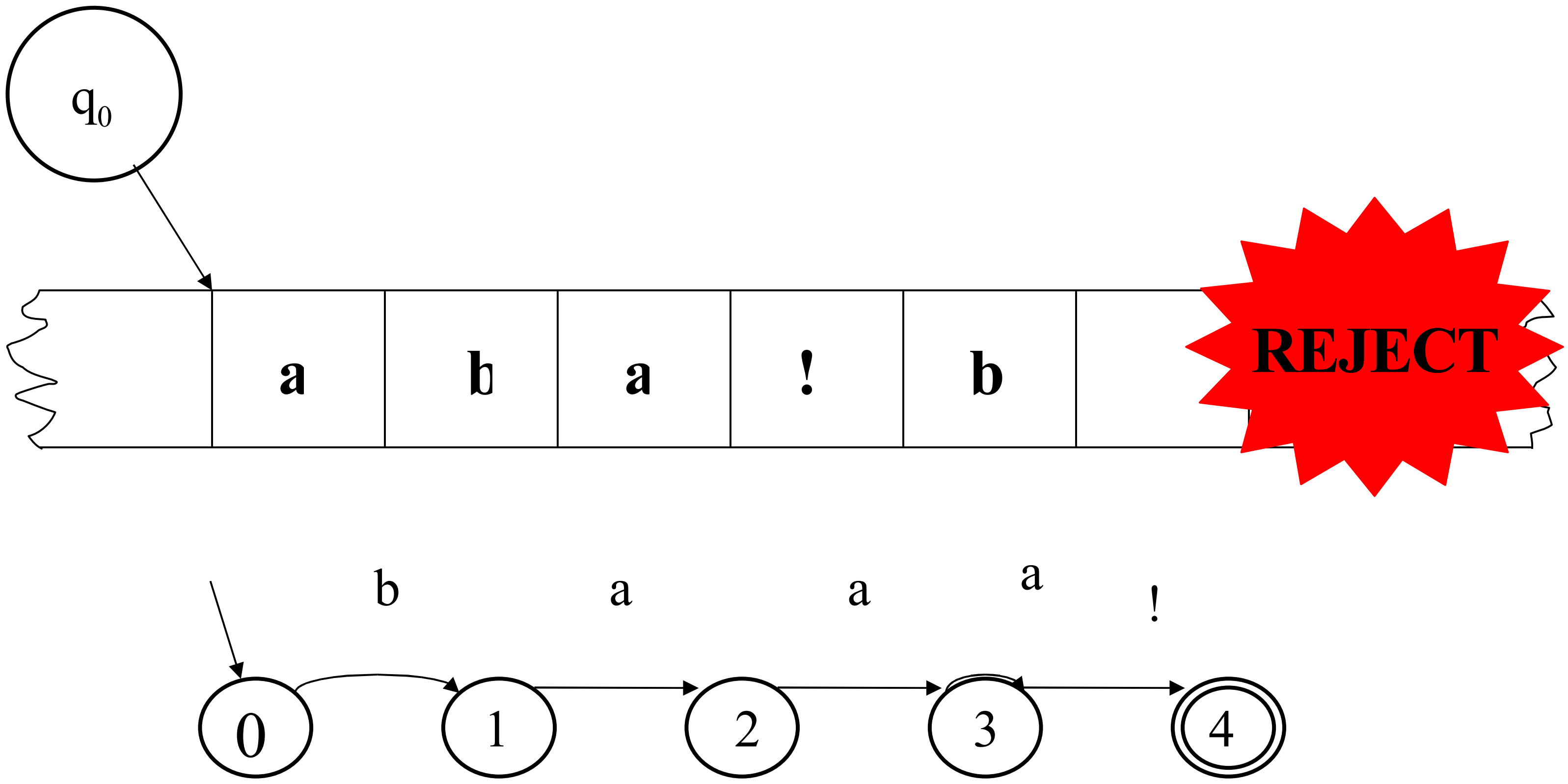
- Traditionally, (Turing's idea) this process is depicted with a long tape broken up into cells, with one symbol written in each cell of the tape.



Recognition

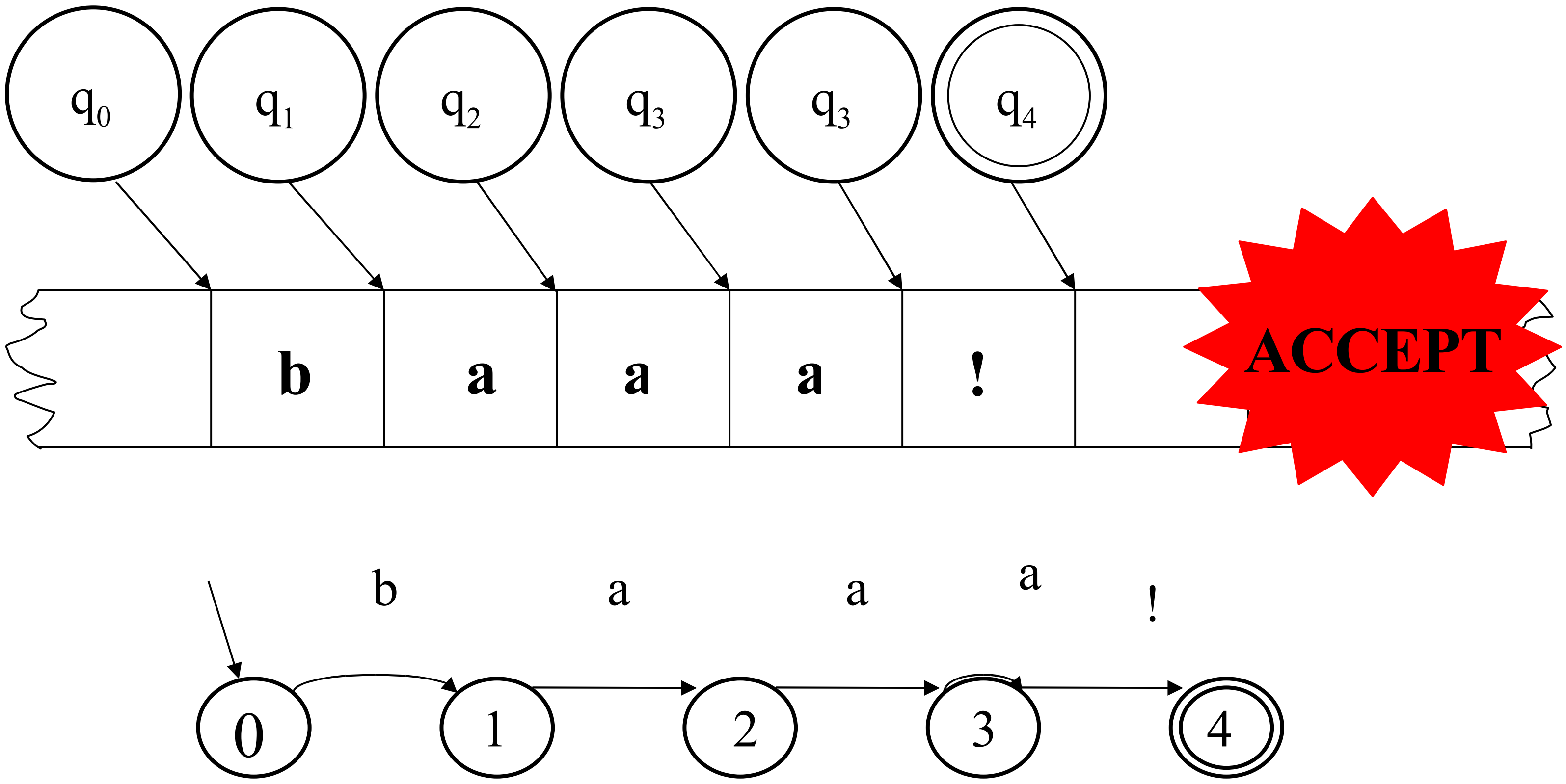
- Start in the start state
- Iterate the following process until you run out of tape
  - Examine the current input
  - Consult the table
  - Go to a new state and update the tape pointer.
- The machine has **successfully** recognized the input if it is in the accepting state when it runs out of input
- The machine **rejects** or fail to accept the input if it never gets to the final state because:
  - It runs out of input
  - Some input doesn´t match an arc

Input Tape



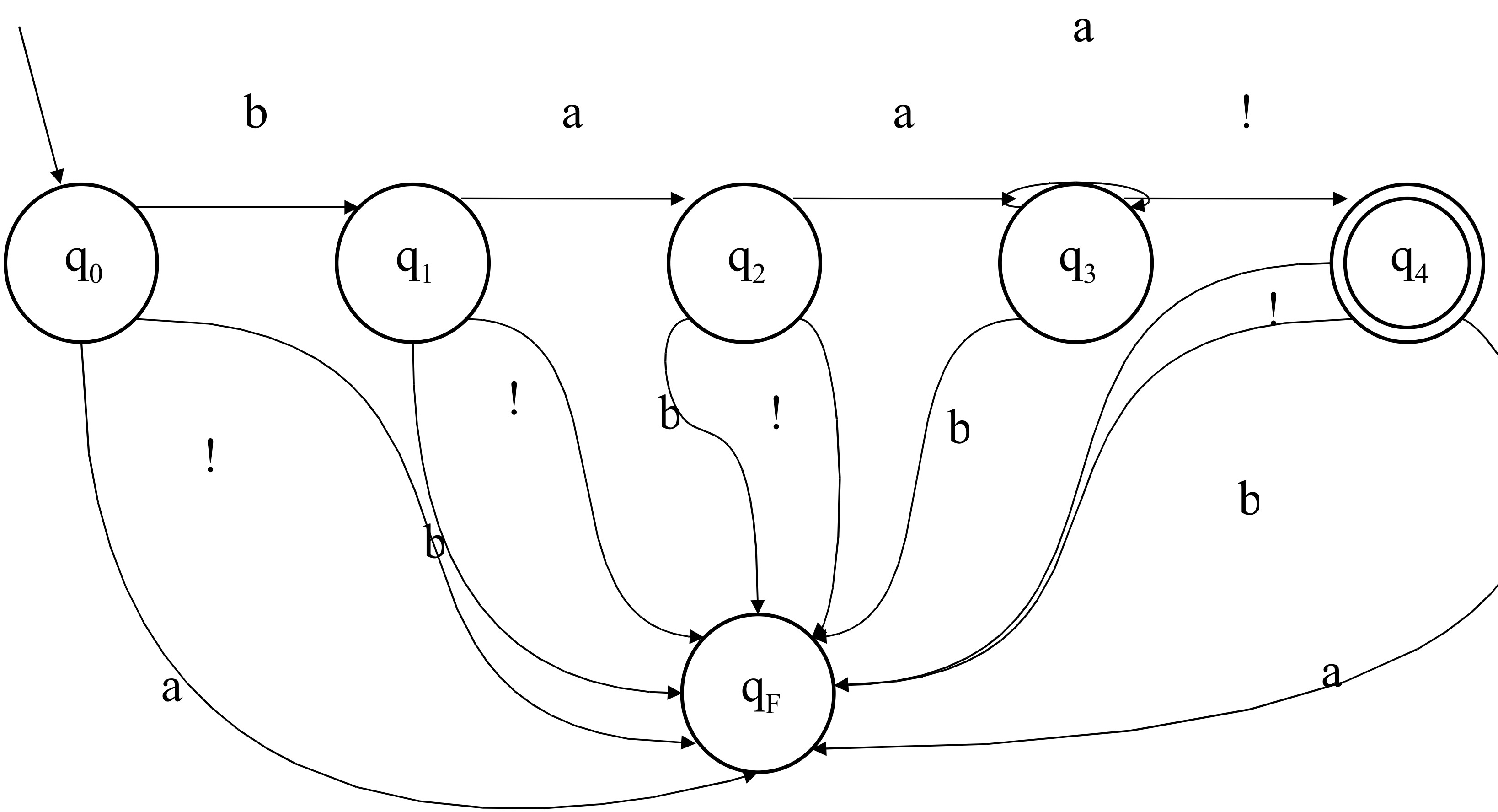
Slide from Dorr/Monz

Input Tape



Slide from Dorr/Monz

Augmented machine with a failing state



Slide from Dorr/Monz

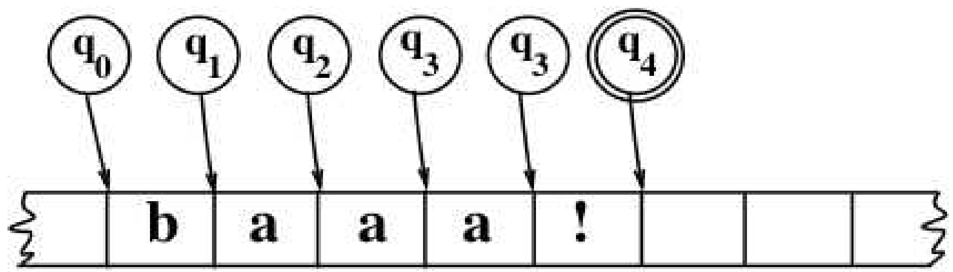
D-RECOGNIZE

```
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
```

Slide from Dorr/Monz

Tracing D-Recognize

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



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 languages.
  - To change the machine, you change the table.

Key Points

- Crudely therefore… matching strings with regular expressions (ala Perl) is a matter of
  - translating the expression into a machine (table) and
  - passing the table to an interpreter

Recognition as Search

- You can view this algorithm as state-space search.
- States are pairings of tape positions and state numbers.
- Operators are compiled into the table
- Goal state is a pairing with the end of tape position and a final accept state

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

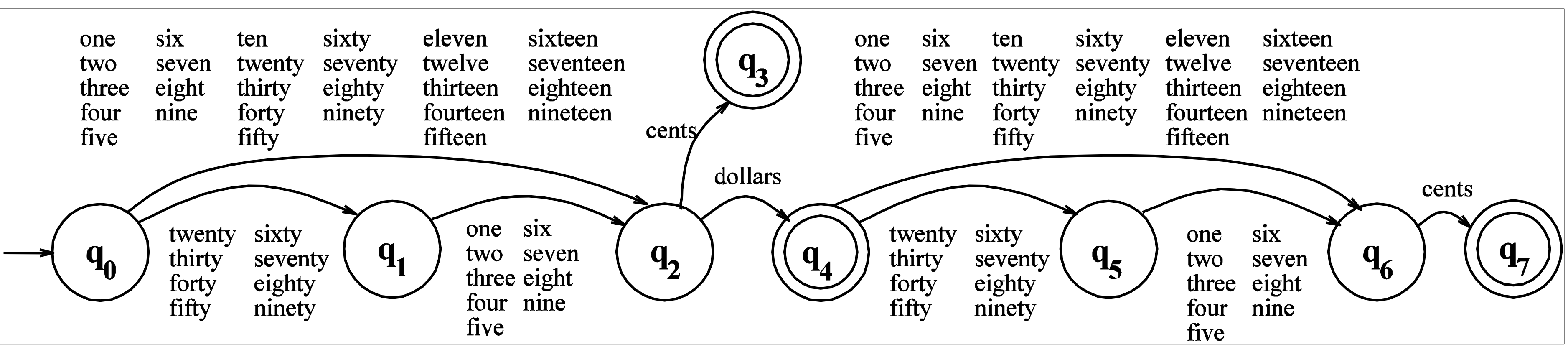
- FSAs can be viewed from two perspectives:
  - Acceptors that can tell you if a string is in the language
  - Generators to produce all and only the strings in the language

Another Example: Dollars and Cents

We can have a higher level alphabet consisting of words.

In this way we can write FSA that models facts about word combinations.

Task: Build an FSA that model the subpart of English dealing with amounts of money.



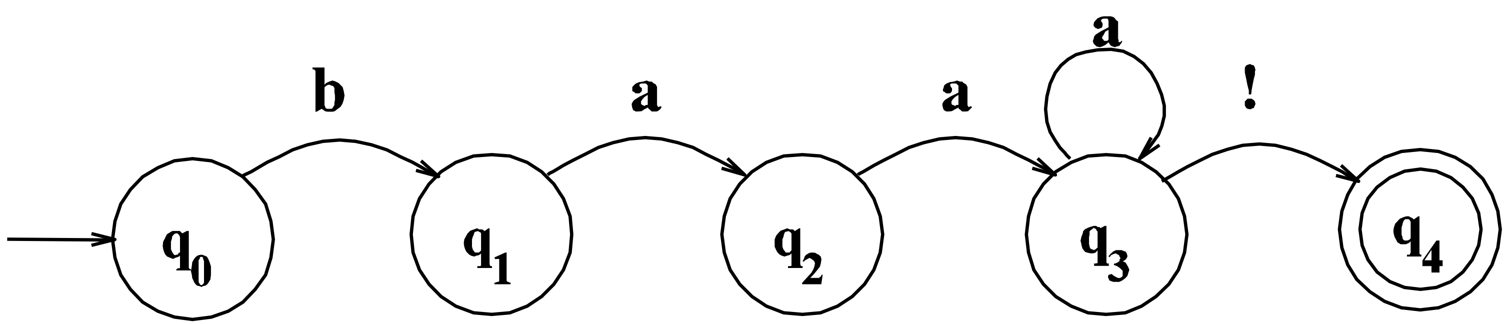
Summary

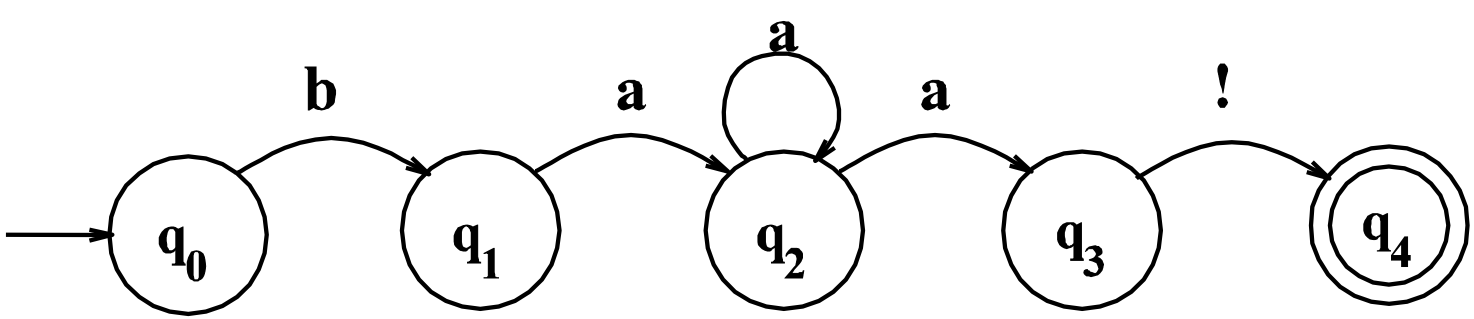
- Regular expressions are just a compact textual representation of FSAs
- Recognition is the process of determining if a string/input is in the language defined by some machine.
  - Recognition is straightforward with deterministic machines.
- FSAs can be used for both generating and recognizing all and only the strings of a formal language

Non-determinism

- A deterministic automaton is one whose behavior during recognition is fully determined by the state it is in and the symbol it is looking at.
- Non-determinism: not fully determined, hence choice

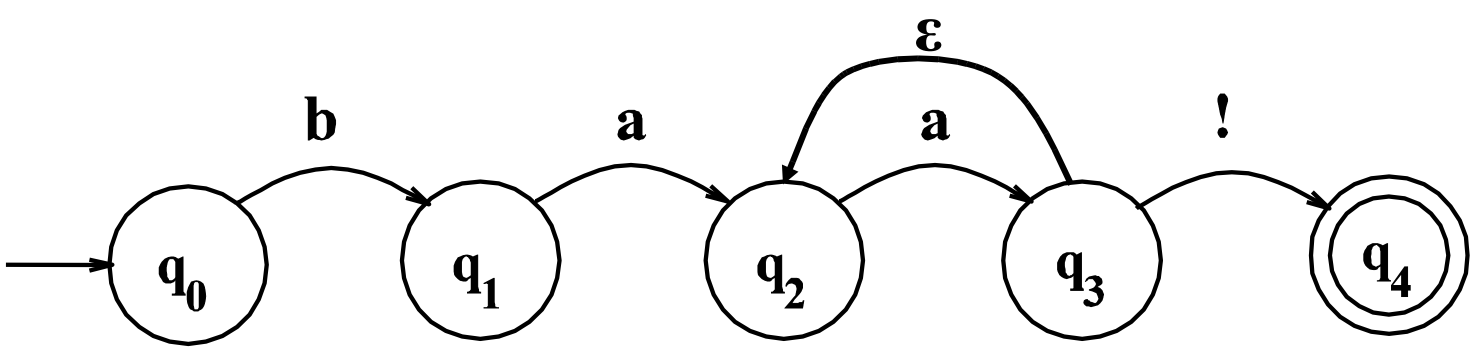
Non-Determinism





Non-Determinism cont.

- Yet another technique
  - Epsilon transitions
  - These transitions do not examine or advance the tape during recognition



NFSA = FSA !!!!

- 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
- It also means that one way to do recognition with a non-deterministic machine is to turn it into a deterministic one.

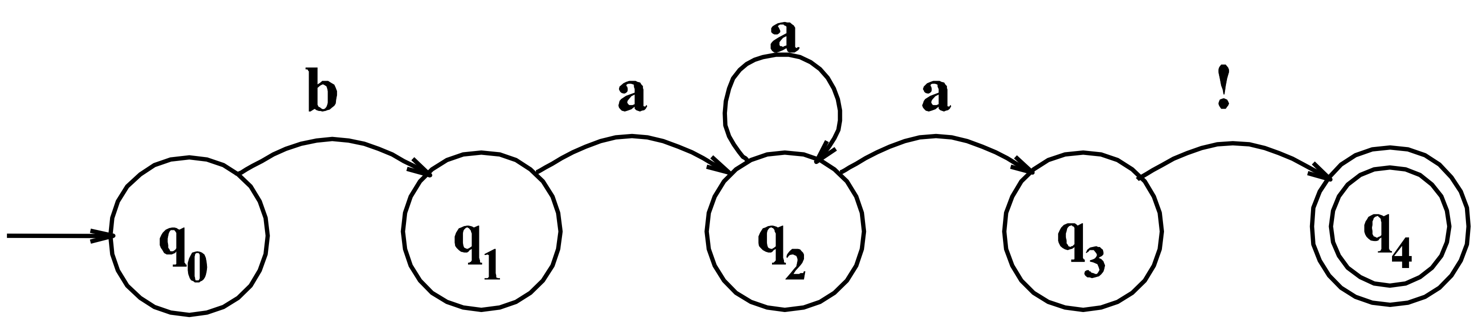
Non-Deterministic Recognition

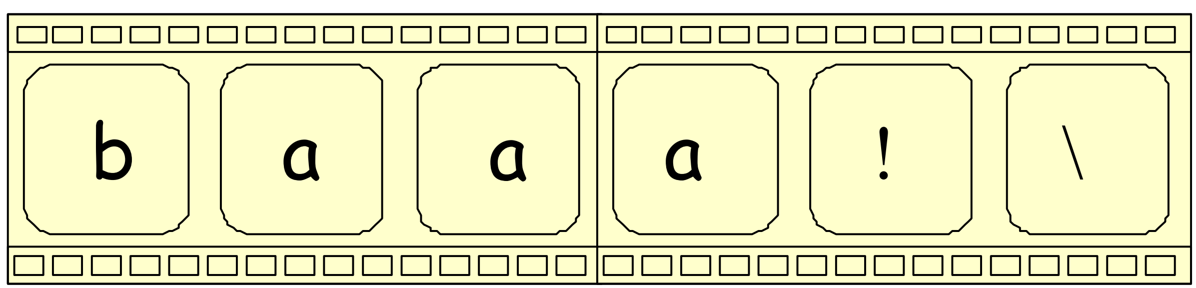
- In a ND FSA there exists at least one path through the machine for a string that is in the language defined by the machine.
- But not all paths directed through the machine for an accept string lead to an accept state.
- No paths through the machine lead to an accept state for a string not in the language.

Non-Deterministic Recognition

- So success in a non-deterministic recognition occurs when a path is found through the machine that ends in an accept.
- Failure occurs when none of the possible paths lead to an accept state.

Example





q_0 q_1 q_2 q_2 q_3 q_4

Using NFSA to accept strings

- In general, solutions to the problem of choice in non-deterministic models:
  - Backup:
    - When we come to a choice point
    - Put a marker indicating:
      - Where we are in the tape
      - What the state is
  - Look-ahead: We could look ahead in the input to help us decide which path to take.
  - Parallelism: Whenever we come to a choice point, we could look at every alternative path in parallel.

ND-Recognize

```
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
```

Key AI idea: Search

- We model problem-solving as a search for a solution through a space of possible solutions.
- The space consists of states.
- 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.

Two kinds of search

- Depth-first search
  - Explore one path all the way to the end
  - Then backup
  - And try other paths
- Breadth-first search
  - Explore all the paths simultaneously
  - Incrementally extending each tier of the paths

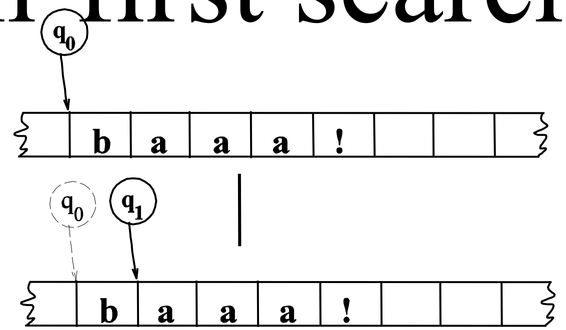
Depth-first search example

1

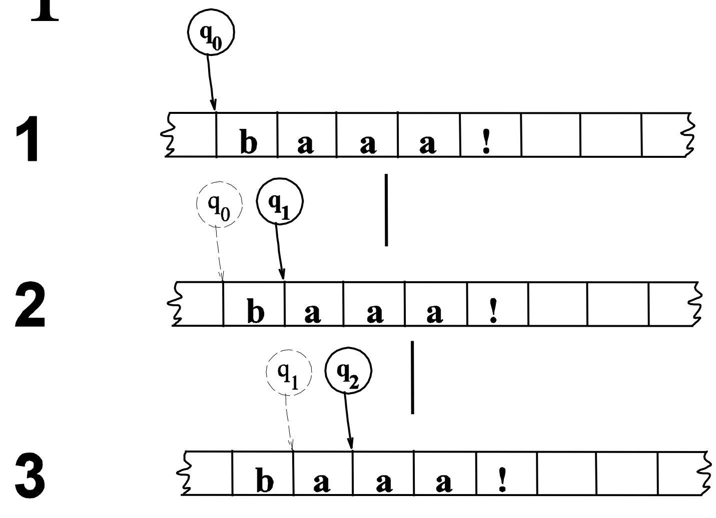
Depth-first search example

1

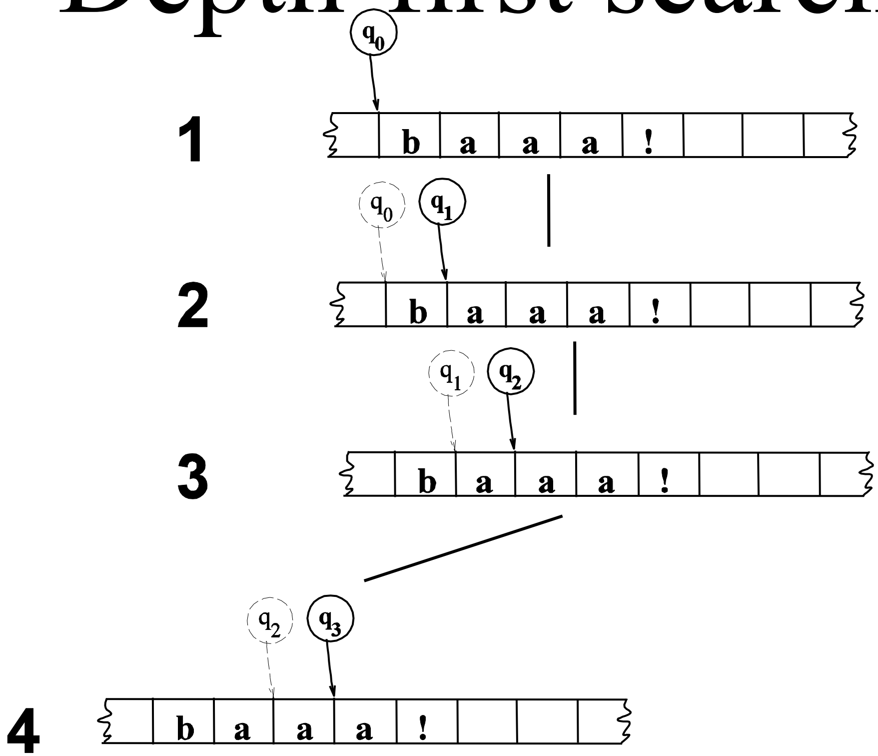
2



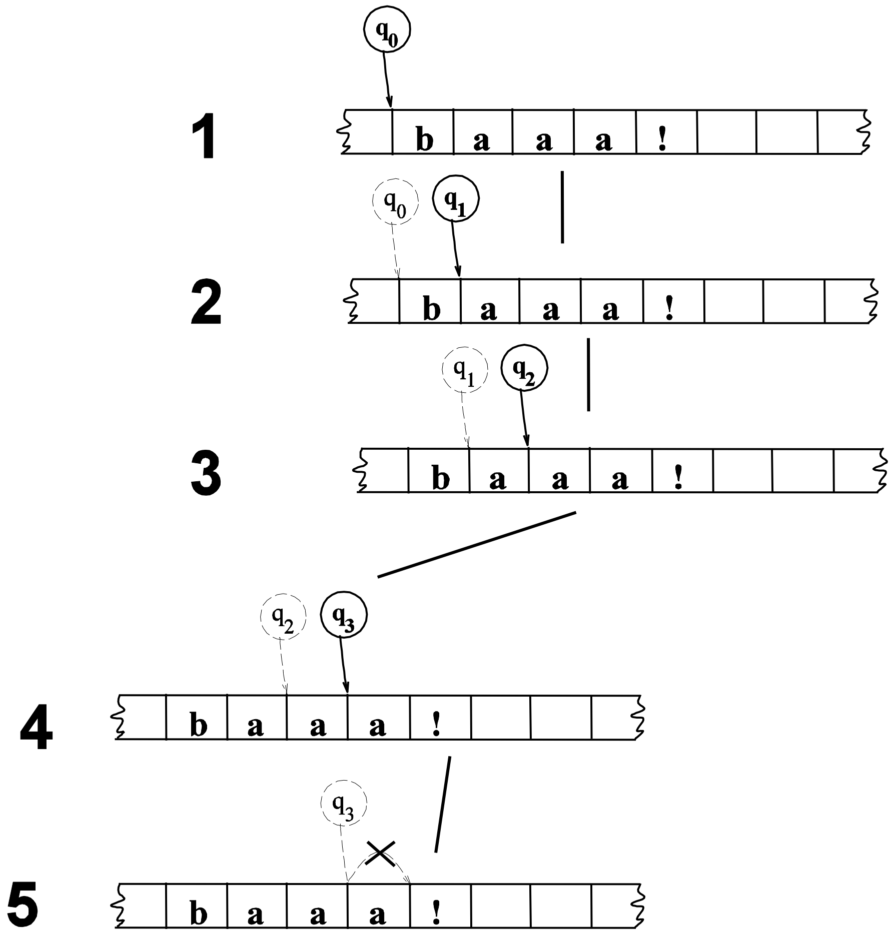
Depth-first search example



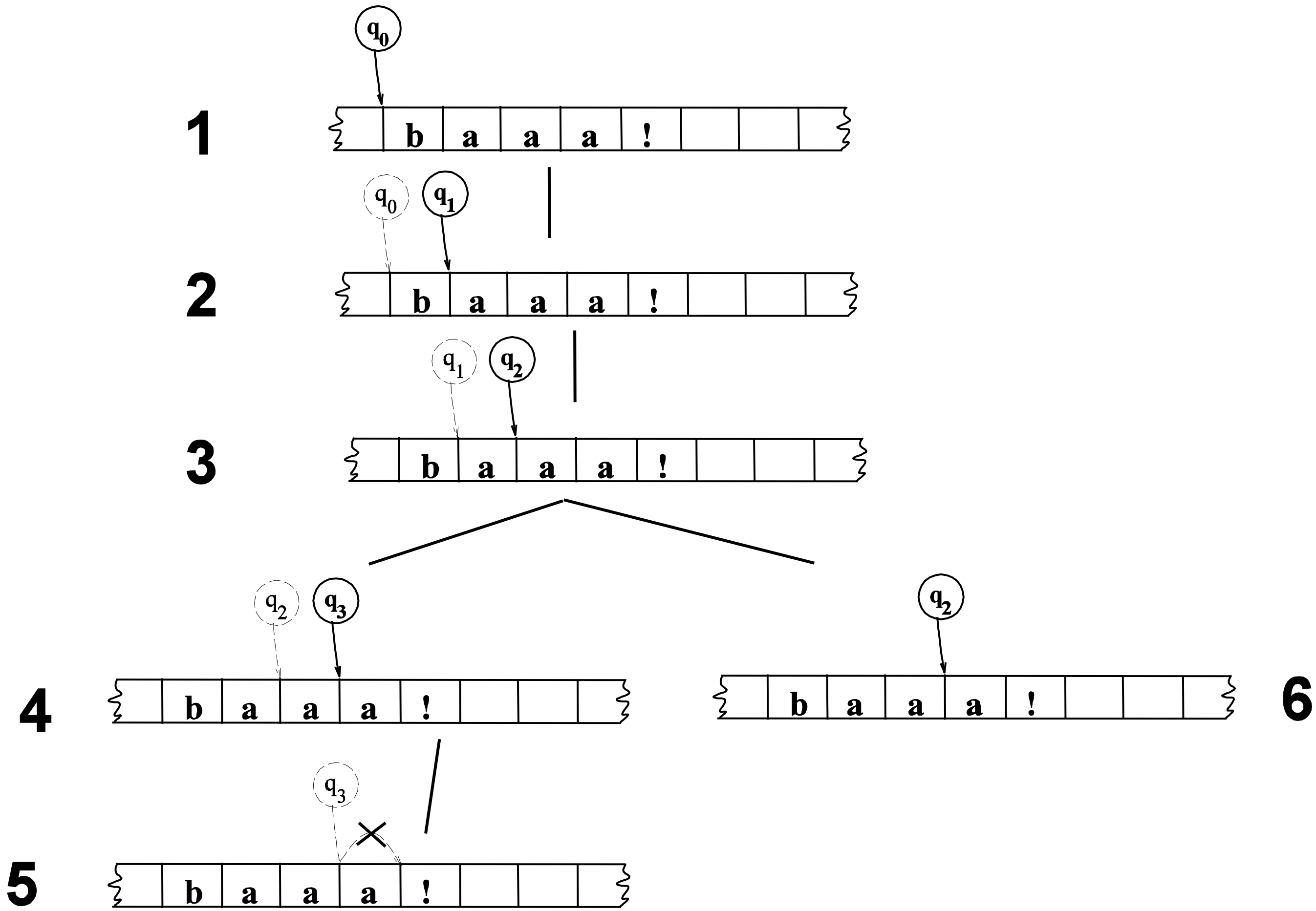
Depth-first search example



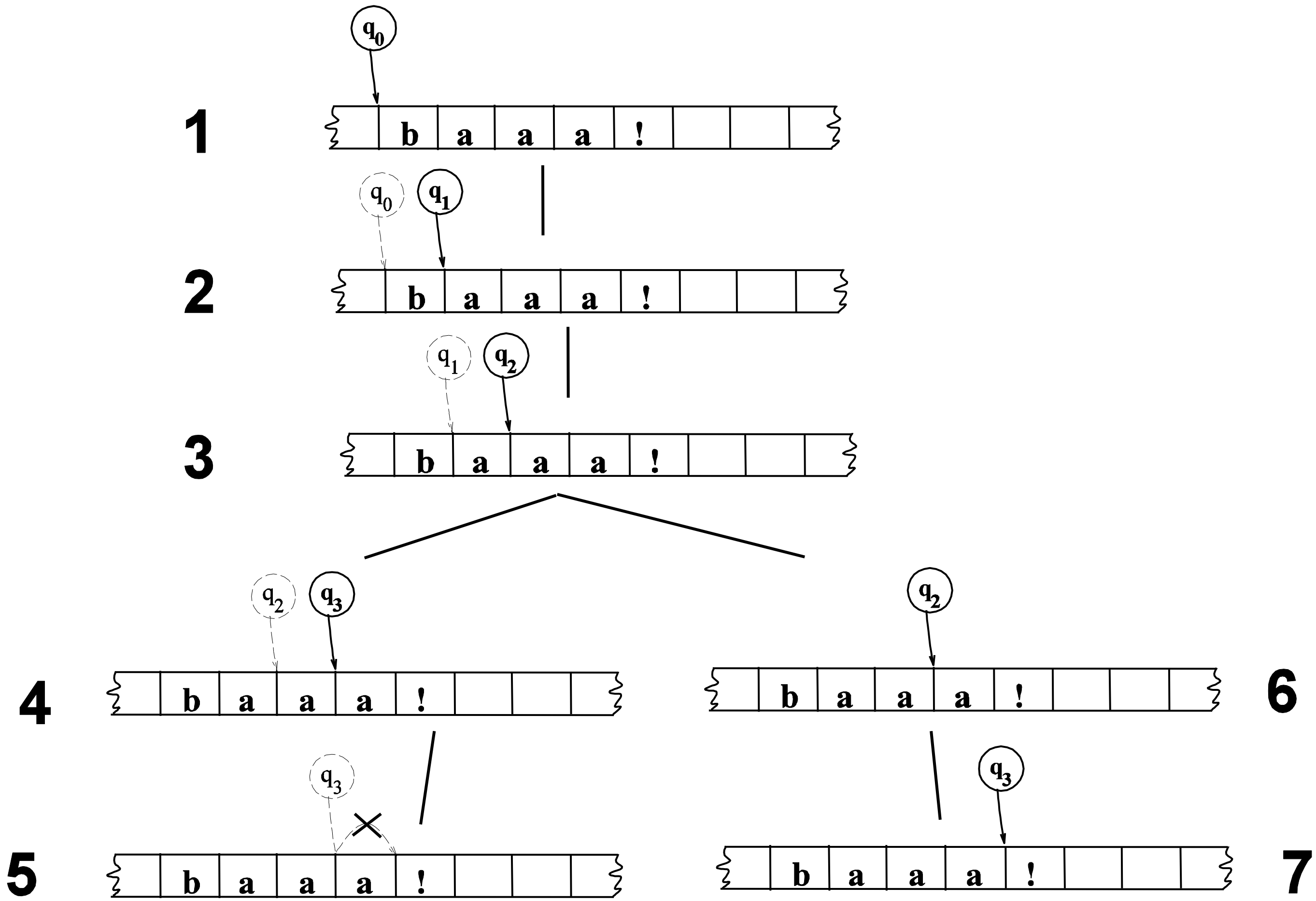
Depth-first search example



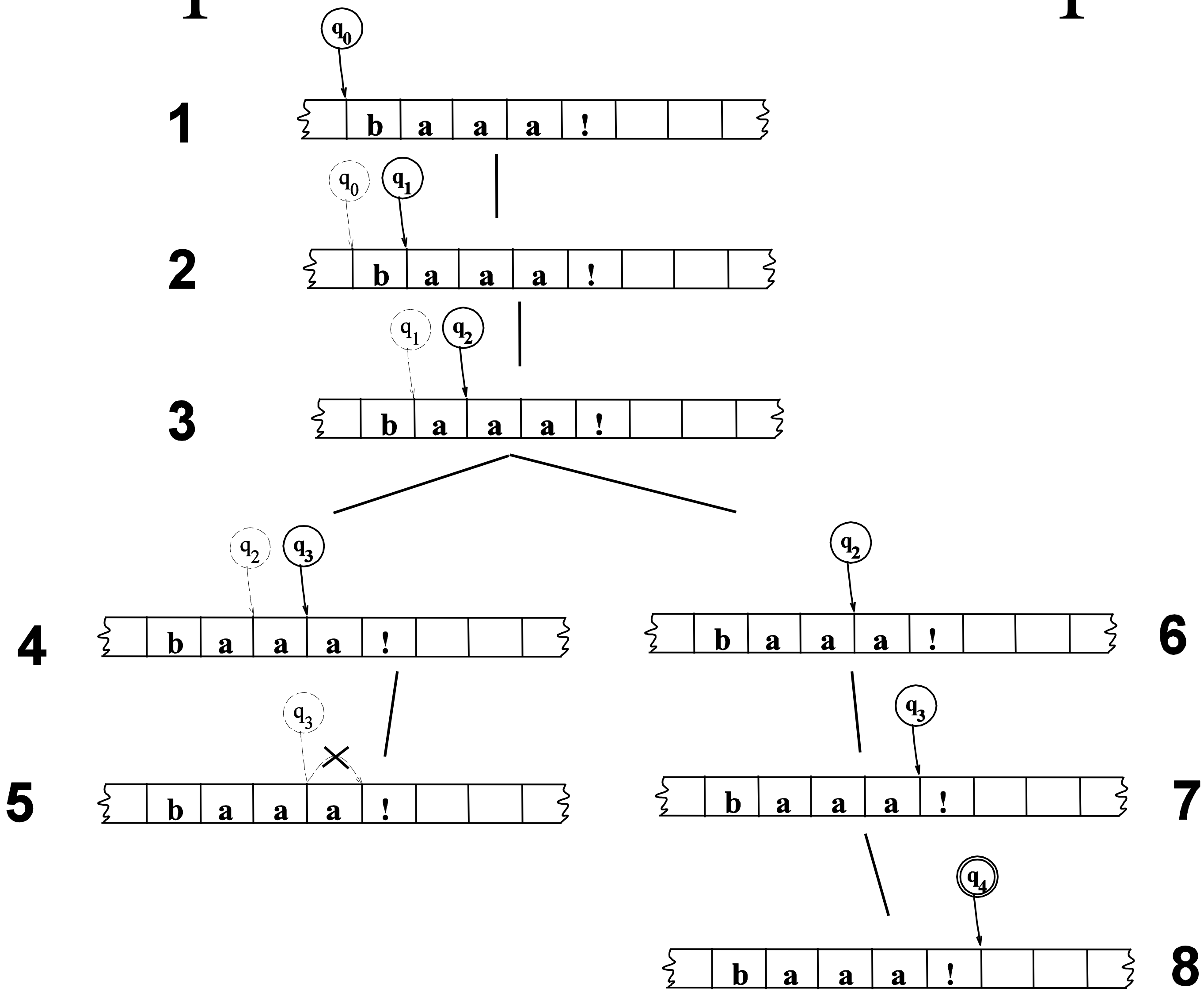
Depth-first search example



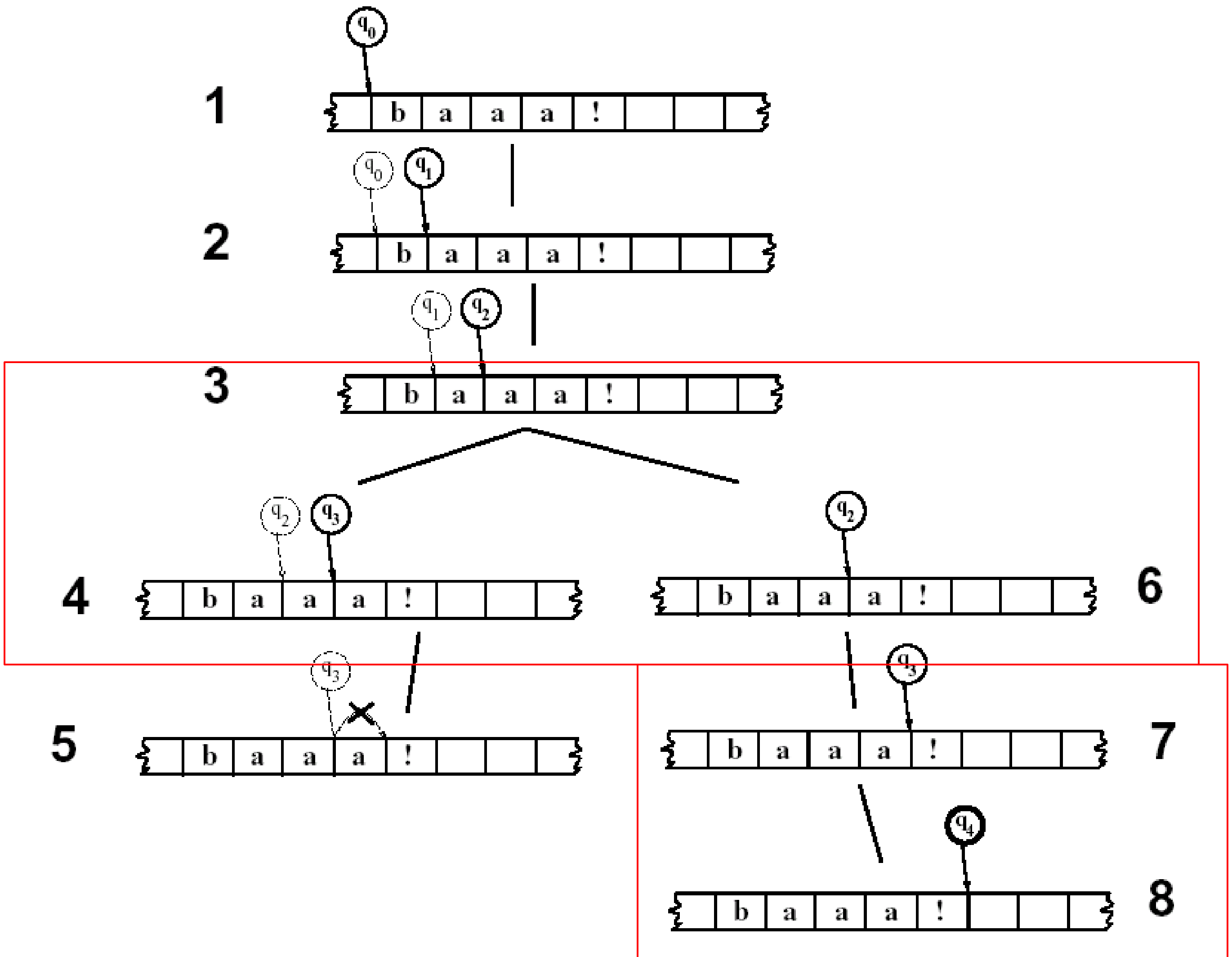
Depth-first search example



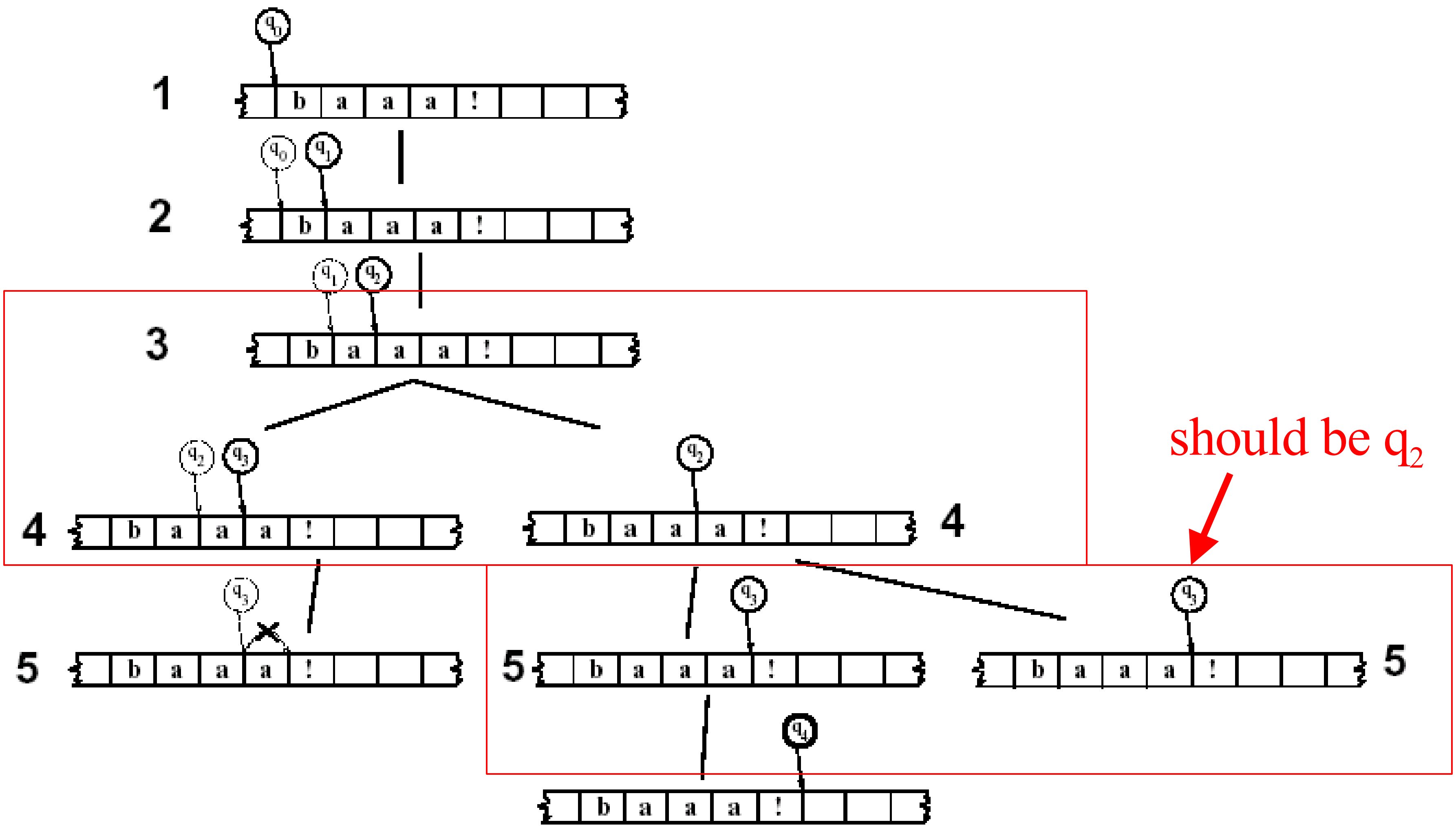
Depth-first search example



NFSA Recognition of "baaa!"

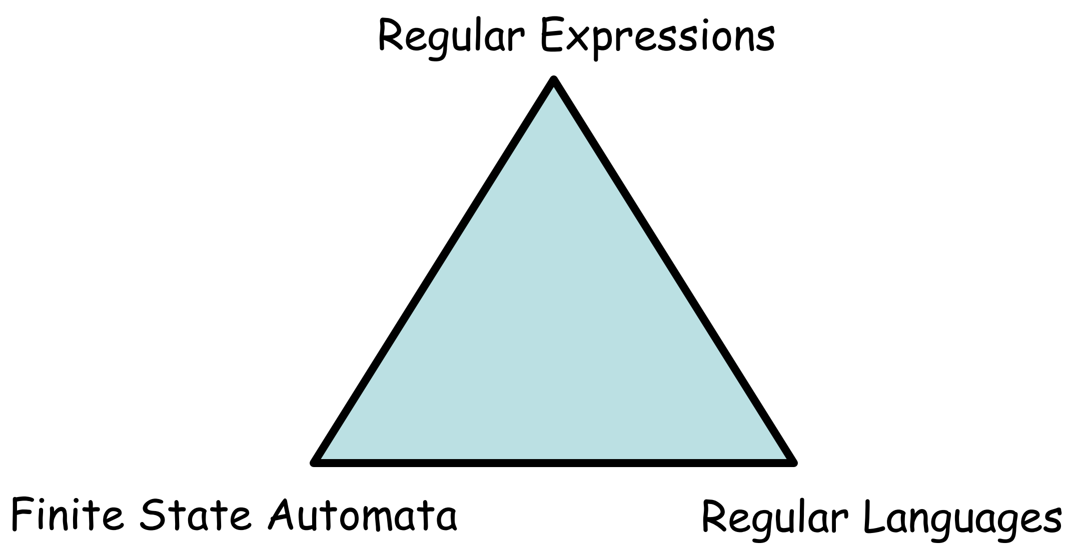


Breadth-first Recognition of “baaa!”



Three Views

- Three equivalent formal ways to look at what we're up to



Regular languages

- Regular languages are characterized by FSAs
- For every NFSA, there is an equivalent DFSA.
- Regular languages are closed under concatenation, Kleene closure, union.

Regular languages

- The class of languages characterizable by regular expressions
- Given alphabet Σ, the regular languages over Σ are:
  - The empty set $\varnothing$ is a regular language

  $\forall a \in \Sigma \cup \varepsilon$, $\{a\}$ is a regular language
  - If L1 and L2 are regular languages, then so are:
    - $L1 \cdot L2 = \{xy \mid x \in L1, y \in L2\}$, concatenation of L1 & L2
    - $L1 \cup L2$, the union of L1 and L2
    - $L1^*$, the Kleene closure of L1

Going from regexp to FSA

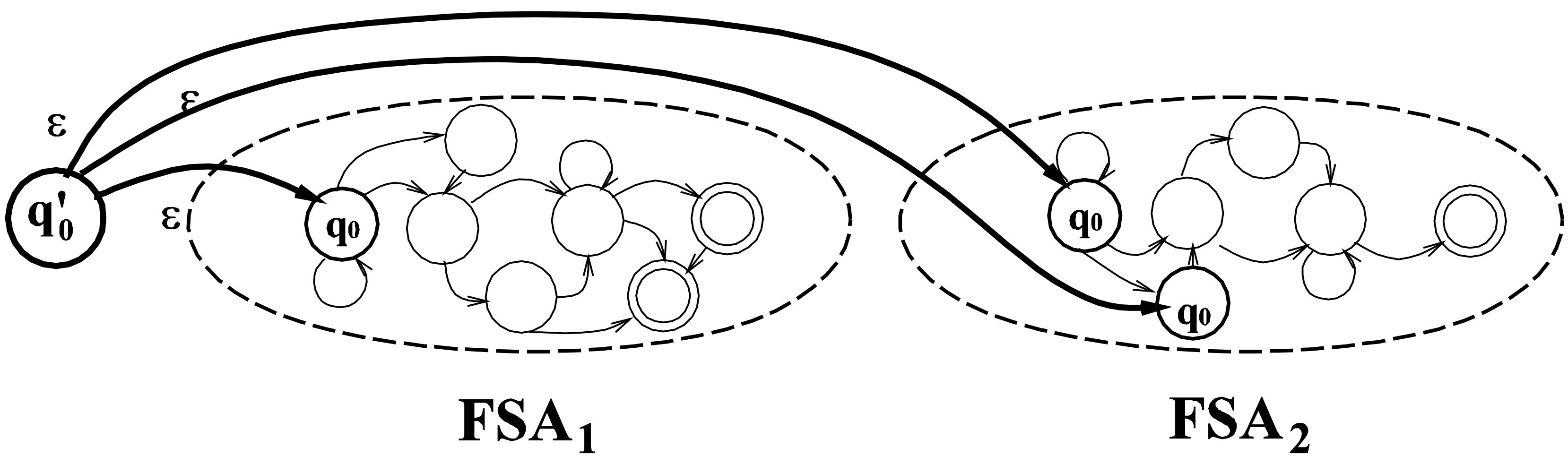
- Since all regular languages meet above properties
- And regular languages are the languages characterizable by regular expressions
- All regular expression operators can be implemented by combinations of union, disjunction, closure
  - Counters (*,+) are repetition plus closure
  - Anchors are individual symbols
  - [] and () and . are kinds of disjunction

Going from regexp to FSA

- So if we could just show how to turn closure/union/concatenation from regexps to FSAs, this would give an idea of how FSA compilation works.
- The actual proof that regular languages = FSAs has 2 parts
  - An FSA can be built for each regular language
  - A regular language can be built for each automaton
- So I'll give the intuition of the first part:
  - Take any regular expression and build an automaton
  - Intuition: induction
    - Base case: build an automaton for single symbol (say 'a')
    - Inductive step: Show how to imitate the 3 regexp operations in automata

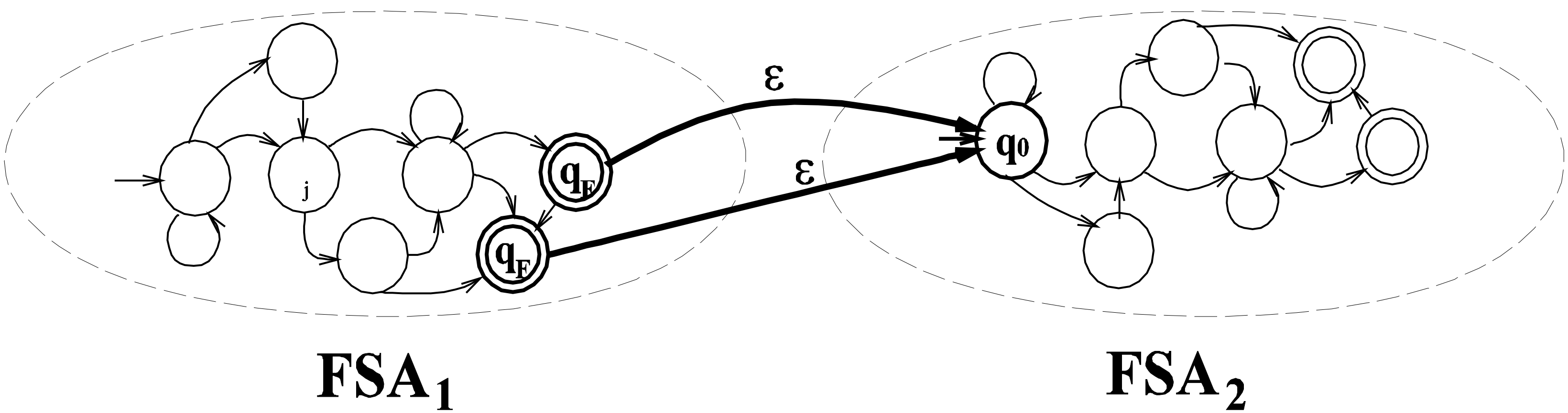
Union

- Accept a string in either of two languages



Concatenation

- Accept a string consisting of a string from language L1 followed by a string from language L2.



Summary so far

- Finite State Automata
  - Deterministic Recognition of FSAs
  - Non-Determinism (NFSAs)
  - Recognition of NFSAs
  - (sketch of) Proof that regular expressions = FSAs

FSAs and Computational Morphology

- An important use of FSAs is for morphology, the study of word parts.

English Morphology

- Morphology is the study of the ways that words are built up from smaller meaningful units called morphemes
- We can usefully divide morphemes into two classes
  - Stems: The core meaning bearing units (the main morphemes of the words)
  - Affixes: Bits and pieces that adhere to stems to change their meanings and grammatical functions

    Affixes are further divided into prefixes (precede the stem), suffixes (follow the stem), circumfixes (do both), and infixes (are inserted inside the stem).

English Morphology

- Four clasess of ways to combine morphemes to create words that play important role in NLP:
  - Inflection - the combination of a word stem with a grammatical morpheme, resulting in a word of the *same* class as the original stem (with the same meaning), and usually filling some syntactic function like agreement. Ex.: bird, birds; want, wants, wanted
  - Derivation - the combination of a word stem with a grammatical morpheme, usually resulting in a word of a *different* class, often with a meaning hard to predict exactly. Ex.: computerize, computerization; bad, badly; constant, inconstant

English Morphology

- Compounding - the combination of multiple word stems together. Ex.: doghouse
- cliticization - the combination of a word stem with a clitic. A clitic is a morpheme that acts syntactically like a word, but is reduced in form and attached (phonologically and sometimes orthographically) to another word. Ex.: I've

Inflectional Morphology

- English has a relatively simple inflectional system; only nouns, verbs, and sometimes adjectives can be inflected, and the number of possible inflectional affixes is quite small.
- English **nouns** have only two kinds of inflection: an affix that marks plural and an affix that marks possessive.
- English **verbal** inflection is slightly more complex.

  English has three kinds of verbs; main verbs – regular and irregular (*eat, sleep, walk*), modal verbs (*can, will, should*), and primary verbs (*be, have, do*).

  They have afixes appropriate to the tense of the verb.

Regulars and Irregulars

- It gets a little complicated by the fact that some words misbehave (refuse to follow the rules)
  - Mouse/mice, goose/geese, ox/oxen
  - Go/went, fly/flew
- The terms regular and irregular will be used to refer to words that follow the rules and those that don't.

Regular and Irregular Nouns and Verbs

- Regulars…
  - Walk, walks, walking, walked, walked
  - Table, tables
- Irregulars
  - Eat, eats, eating, ate, eaten
  - Catch, catches, catching, caught, caught
  - Cut, cuts, cutting, cut, cut
  - Leaf, leaves

Nouns and Verbs

	Regular Nouns		Irregular Nouns	
Singular	cat	thrush	mouse	ox
Plural	cats	thrushes	mice	oxen

Morphological Form Classes	Regularly Inflected Verbs			
stem	walk	merge	try	map
-s form	walks	merges	tries	maps
-ing participle	walking	merging	trying	mapping
Past form or *-ed* participle	walked	merged	tried	mapped

Morphological Form Classes	Irregularly Inflected Verbs		
stem	eat	catch	cut
-s form	eats	catches	cuts
-ing participle	eating	catching	cutting
Past form	ate	caught	cut
-ed participle	eaten	caught	cut

Derivational Morphology

- Derivation in English is quite complex.
- Start with compute
  - Computer -> computerize -> computerization
  - Computation -> computational
  - Computer -> computerize -> computerizable
  - Compute -> computee

Why care about morphology?

- ´Stemming´ in information retrieval
  - Might want to search for "going home" and find pages with both "went home" and "will go home"
- Morphology in machine translation
  - Need to know that the Spanish words quiero and quieres are both related to querer 'want'
- Morphology in spell checking
  - Need to know that misclam and antiundoggingly are not words despite being made up of word parts

Can’t just list all words

- Turkish
- Uygarlastiramadiklarimizdanmissinizcasina
- `(behaving) as if you are among those whom we could not civilize’
- Uygar `civilized’ + las `become’ + tir `cause’ + ama `not able’ + dik `past’ + lar ‘plural’+ imiz ‘p1pl’ + dan ‘abl’ + mis ‘past’ + siniz ‘2pl’ + casina ‘as if’

What we want

- Something to automatically do the following kinds of mappings:
- Cats cat +N +PL
- Cat cat +N +SG
- Cities city +N +PL
- Merging merge +V +Present-participle
- Caught catch +V +past-participle

Morphological Parsing

- **Parsing** means taking an input and producing some sort of linguistic structure for it (morphological, syntactic, semantic).
- Forms of linguistic structures:
  - String
  - Tree
  - Network
- The problem of recognizing that a word (like *foxes*) breaks down into component morphemes (*fox* and *-es*) and building a structured representation of this fact is called **morphological parsing**.

Morphological Parsing: Goal

English		Spanish		
Input	Morphologically Parsed Output	Input	Morphologically Parsed Output	Gloss
cats	cat +N +PL	pavos	pavo +N +Masc +Pl	‘ducks’
cat	cat +N +SG	pavo	pavo +N +Masc +Sg	‘duck’
cities	city +N +Pl	bebo	beber +V +PInd +1P +Sg	‘I drink’
geese	goose +N +Pl	canto	cantar +V +PInd +1P +Sg	‘I sing’
goose	goose +N +Sg	canto	canto +N +Masc +Sg	‘song’
goose	goose +V	puse	poner +V +Perf +1P +Sg	‘I was able’
gooses	goose +V +1P +Sg	vino	venir +V +Perf +3P +Sg	‘he/she came’
merging	merge +V +PresPart	vino	vino +N +Masc +Sg	‘wine’
caught	catch +V +PastPart	lugar	lugar +N +Masc +Sg	‘place’
caught	catch +V +Past			

Figure 3.2 Output of a morphological parse for some English and Spanish words. Spanish output modified from the Xerox XRCE finite-state language tools.

FSAs and the Lexicon

- This will actual require a kind of FSA called the Finite State Transducer (FST)
- First we'll capture the morphotactics
  - The rules governing the ordering of affixes in a language.
- Then we'll add in the actual words

Building a Morphological Parser

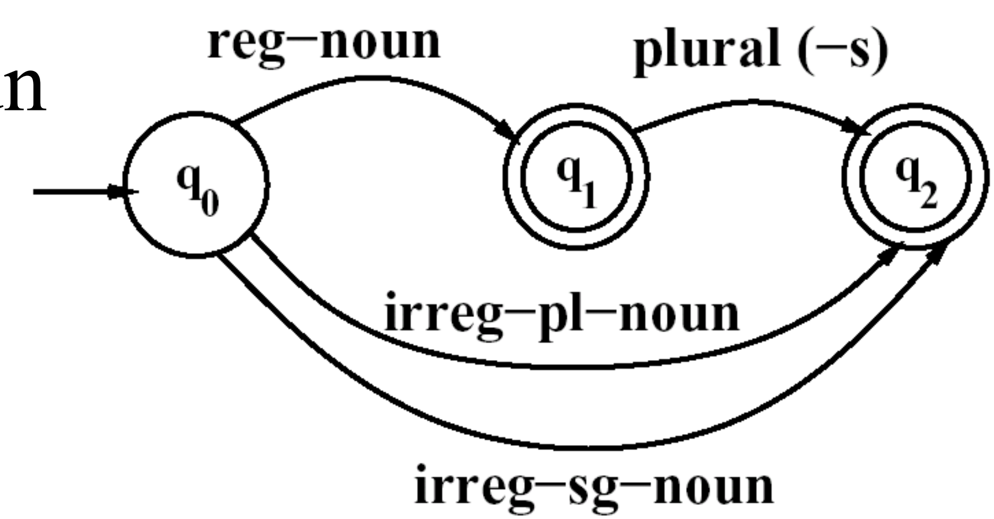
- Three components:
  - Lexicon - the list of stems and affixes, together with basic information about them (whether a stem is a Noun stem or a Verb stem, etc.).
  - Morphotactics - the model of morpheme ordering that explains which classes of morphemes can follow other classes of morphemes inside a word.
  - Orthographic or Phonological Rules - these spelling rules are used to model the changes that occur in a word, usually when two morphemes combine (e.g., the y->ie spelling rule that changes city + -s to cities rather than citys).

Lexicon: FSA Inflectional Noun Morphology

- English Noun Lexicon

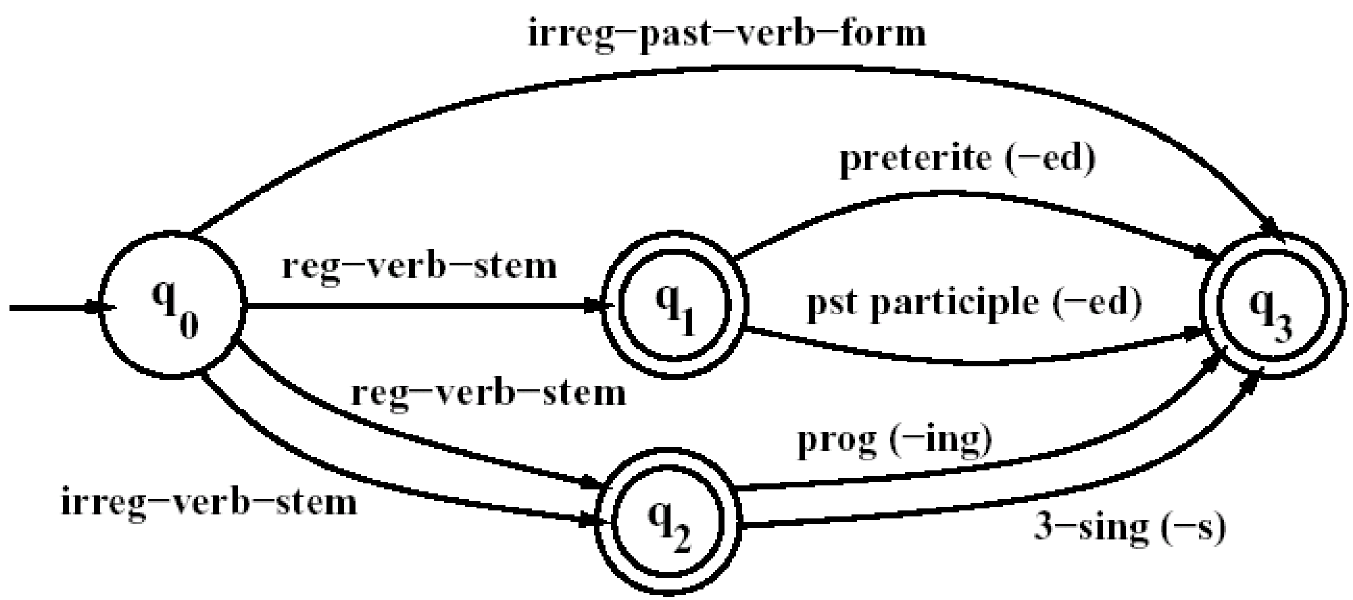
reg-noun	Irreg-pl-noun	Irreg-sg-noun	plural
fox cat dog	geese sheep mice	goose sheep mouse	-s

- English Noun Rule

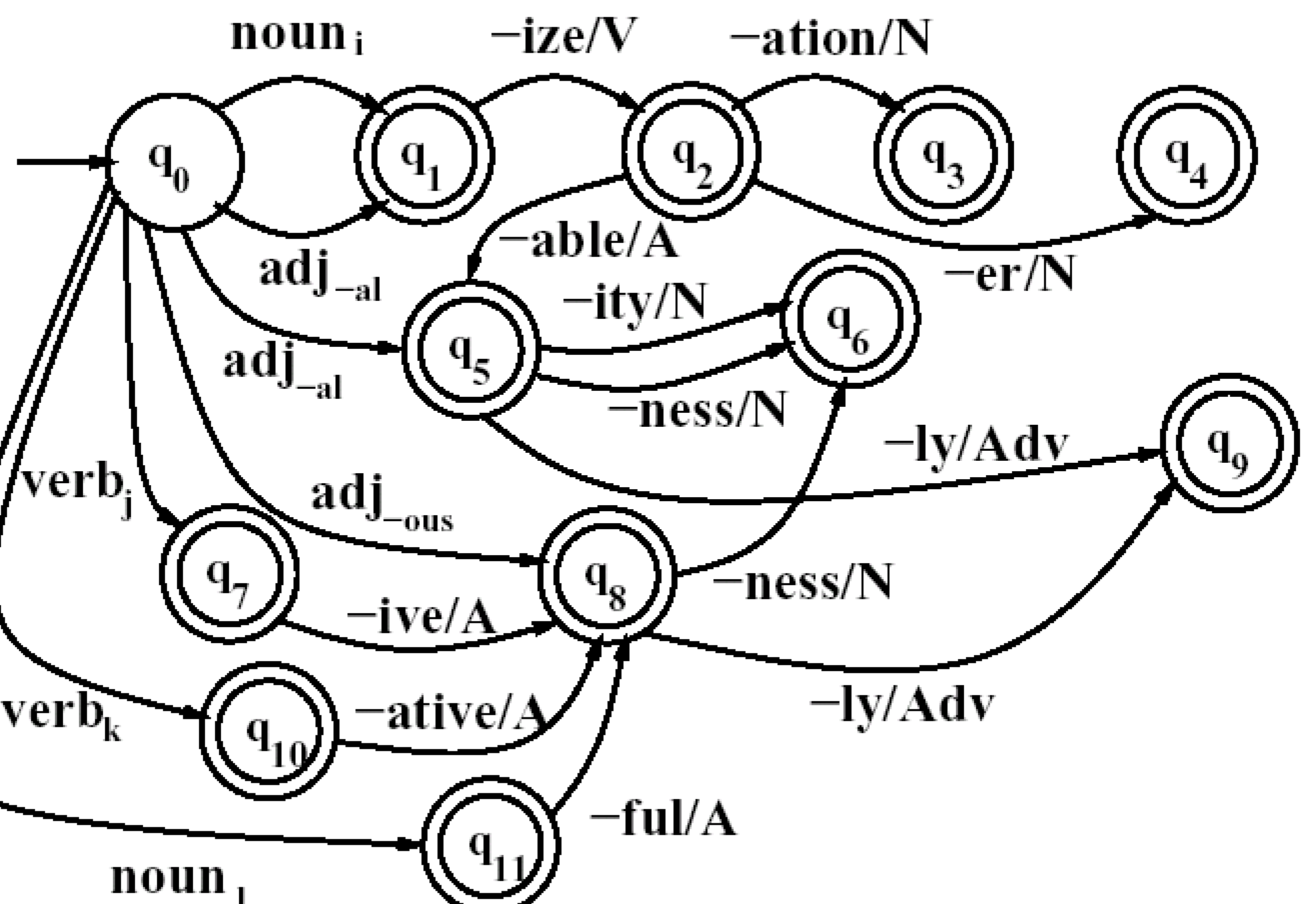


Lexicon and Rules: FSA English Verb Inflectional Morphology

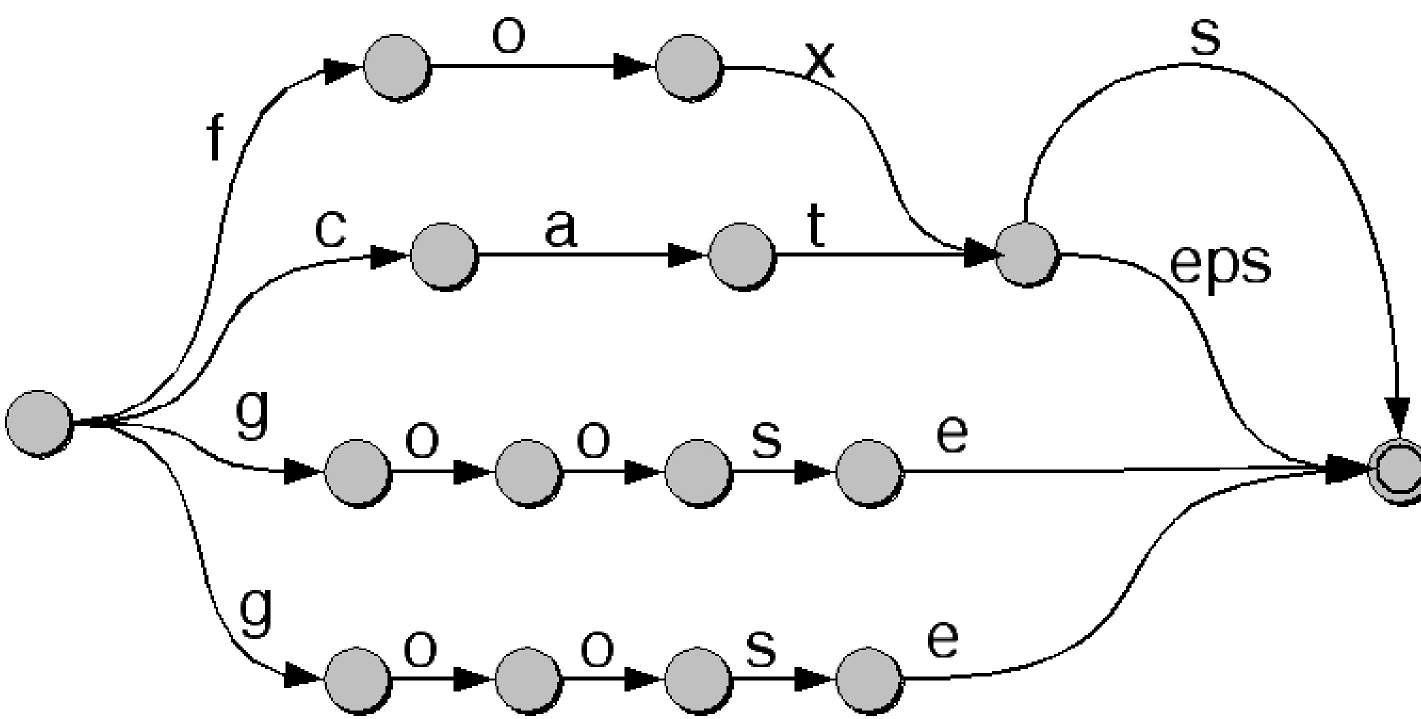
reg-verb-stem	**irreg-verb-stem**	**irreg-past-verb**	**past**	**past-part**	**pres-part**	**3sg**
walk fry talk impeach	cut speak spoken sing sang	caught ate eaten	-ed	-ed	-ing	-s



More Complex Derivational Morphology



Using FSAs for Recognition: English Nouns and Inflection



Parsing/Generation vs. Recognition

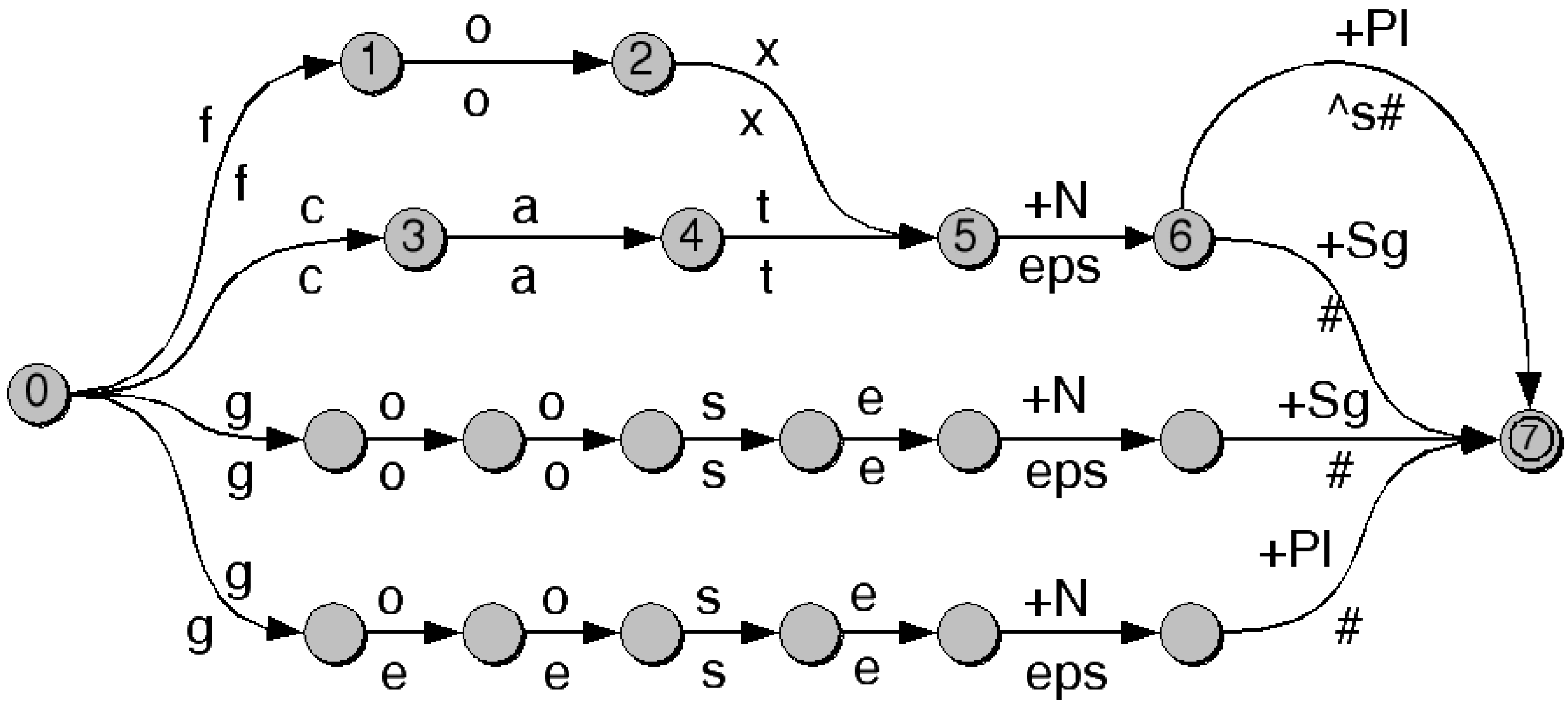
- We can only recognize words
- But this isn't the same as parsing
  - Parsing: building structure
  - Usually if we find some string in the language we need to find the structure in it (parsing)
  - Or we have some structure and we want to produce a surface form (production/generation)
- Example
  - From "cats" to "cat +N +PL"

Finite State Transducers

- The simple story
  - Add another tape
  - Add extra symbols to the transitions

  - On one tape we read "cats", on the other we write "cat +N +PL"

Nominal Inflection FST



For more on morphology and full definition of FSTs

- Read Chapter 3 of J&M book

Tokenization

- Segmenting words in running text
- Segmenting sentences in running text
- Why not just periods and white-space?
  - **Mr. Sherwood said reaction to Sea Containers' proposal has been "very positive." In New York Stock Exchange composite trading yesterday, Sea Containers closed at $62.625, up 62.5 cents.**
  - **"I said, 'what're you? Crazy?' " said Sadowsky. "I can't afford to do that.''**
- Words like:
  - **cents. said, positive." Crazy?**

Can’t just segment on punctuation

- Word-internal punctuation
  - M.p.h
  - Ph.D.
  - AT&T
  - 01/02/06
  - Google.com
  - 555,500.50
- Expanding clitics
  - What’re -> what are
  - I’m -> I am
- Multi-token words
  - New York
  - Rock ‘n’ roll

Sentence Segmentation

- !, ? relatively unambiguous
- Period "." is quite ambiguous
  - Sentence boundary
  - Abbreviations like Inc. or Dr.
- General idea:
  - Build a binary classifier:
    - Looks at a "."
    - Decides EndOfSentence/NotEOS
    - Could be hand-written rules, or machine-learning

Word Segmentation in Chinese

- Some languages don’t have spaces
  - Chinese, Japanese, Thai, Khmer
- Chinese:
  - Words composed of characters
  - Characters are generally 1 syllable and 1 morpheme.
  - Average word is 2.4 characters long.
  - Standard segmentation algorithm:
    - Maximum Matching (also called Greedy)

Maximum Matching Word Segmentation

- Given a wordlist of Chinese, and a string.
- Start a pointer at the beginning of the string
- Find the longest word in dictionary that matches the string starting at pointer
- Move the pointer over the word in string
- Go to 2

English example (Palmer 00)

- the table down there
- thetabledownthere
- Theta bled own there

- Words astonishingly well in Chinese
- Far better than this English example suggests
- Modern algorithms better still:
  - probabilistic segmentation

Summary

- Finite State Automata
- Deterministic Recognition of FSAs
- Non-Determinism (NFSAs)
- Recognition of NFSAs
- Proof that regular expressions = FSAs
- Very brief sketch: Morphology, FSAs, FSTs
- Very brief sketch: Tokenization