

# Basic Text Processing

Regular Expressions

# Regular expressions

- A formal language for specifying text strings
- How can we search for any of these?
  - woodchuck
  - woodchucks
  - Woodchuck
  - Woodchucks



# Regular Expressions: Disjunctions

- Letters inside square brackets []

Pattern	Matches
<i>[wW]oodchuck</i>	<i>Woodchuck, woodchuck</i>
<i>[1234567890]</i>	<i>Any digit</i>

- Ranges [A-Z]

Pattern	Matches	
<i>[A-Z]</i>	<i>An upper case letter</i>	<i><u>D</u>renched Blossoms</i>
<i>[a-z]</i>	<i>A lower case letter</i>	<i><u>m</u>y beans were impatient</i>
<i>[0-9]</i>	<i>A single digit</i>	<i>Chapter <u>1</u>: Down the Rabbit Hole</i>

# Regular Expressions: Negation in Disjunction

- Negations `[ ^Ss ]`
  - Carat means negation only when first in []

Pattern	Matches	
<code>[ ^A-Z ]</code>	<i>Not an upper case letter</i>	O <u>y</u> fn pripetchik
<code>[ ^Ss ]</code>	<i>Neither 'S' nor 's'</i>	<u>I</u> have no exquisite reason"
<code>[ ^e^ ]</code>	<i>Neither e nor ^</i>	Look h <u>e</u> re
<code>a^b</code>	<i>The pattern a carat b</i>	Look up <u>a^b</u> now

# Regular Expressions: More Disjunction

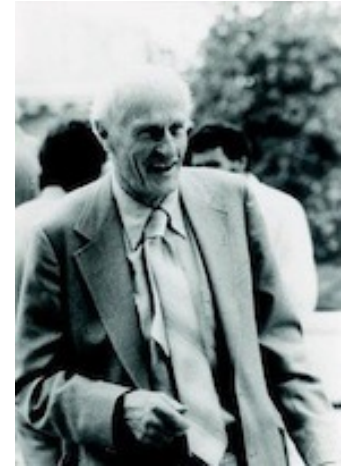
- Woodchucks is another name for groundhog!
- The pipe | for disjunction

Pattern	Matches
<i>groundhog/woodchuck</i>	
<i>yours/mine</i>	<i>yours</i>
<i>a/b/c</i>	<i>= [abc]</i>
<i>[gG]roundhog/[Ww]oodchuck</i>	



# Regular Expressions: ? \* + .

Pattern	Matches	
<b><i>colou?r</i></b>	<i>Optional previous char</i>	<u>color</u> <u>colour</u>
<b><i>oo*h!</i></b>	<i>0 or more of previous char</i>	<u>oh!</u> <u>ooh!</u> <u>oooh!</u> <u>ooooh!</u>
<b><i>o+h!</i></b>	<i>1 or more of previous char</i>	<u>oh!</u> <u>ooh!</u> <u>oooh!</u> <u>ooooh!</u>
<b><i>baa+</i></b>		<u>baa</u> <u>baaa</u> <u>baaaa</u> <u>baaaaa</u>
<b><i>beg.n</i></b>		<u>begin</u> <u>begun</u> <u>begun</u> <u>beg3n</u>



Stephen C Kleene

Kleene \*, Kleene +

# Regular Expressions: Anchors ^ \$

Pattern	Matches
<span style="color: brown;">^</span> <span style="color: black;">[A-Z]</span>	<span style="color: blue;"><u>P</u></span> <i>alo Alto</i>
<span style="color: brown;">^</span> <span style="color: black;">[<span style="color: brown;">^</span>A-Za-z]</span>	<span style="color: blue;"><u>1</u></span> <span style="color: blue;"><u>"Hello"</u></span>
<span style="color: black;">\.</span> <span style="color: brown;">\$</span>	<i>The end</i> <span style="color: blue;"><u>.</u></span>
<span style="color: brown;">.</span> <span style="color: brown;">\$</span>	<i>The end</i> <span style="color: blue;"><u>?</u></span> <i>The end</i> <span style="color: blue;"><u>!</u></span>

**'the' example [in terminal]**



# The Example

- Find me all instances of the word “the” in a text.

the

Misses capitalized examples

[tT]he

Incorrectly returns other or theology

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

# Errors

- The process we just went through was based on fixing two 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 cont.

- In text processing, we are always dealing with these kinds of errors.
- 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).

# Summary

- Regular expressions play a surprisingly large role
  - Sophisticated sequences of regular expressions are often the first model for any text processing text
- For many hard tasks, we use machine learning classifiers
  - But regular expressions are used as features in the classifiers
  - Can be very useful in capturing generalizations

# Basic Text Processing

Regular Expressions

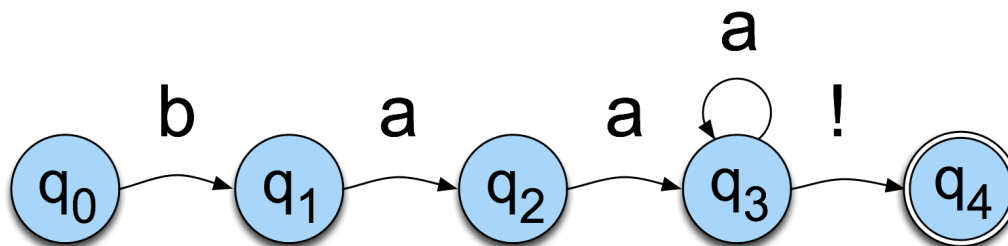
# Finite State Automata

- Regular expressions can be viewed as a textual way of specifying the structure of finite-state automata.
- FSAs and their probabilistic relatives are at the core of much of what we'll be doing all quarter.
- They also capture significant aspects of what linguists say we need for **morphology** and parts of **syntax**.

# FSAs as Graphs

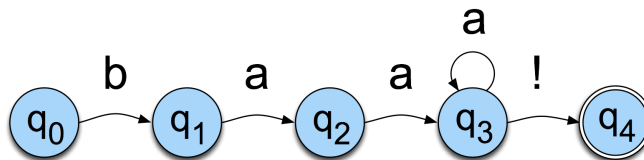
- Let's start with the sheep language

◆ /baa+! /



# Sheep FSA

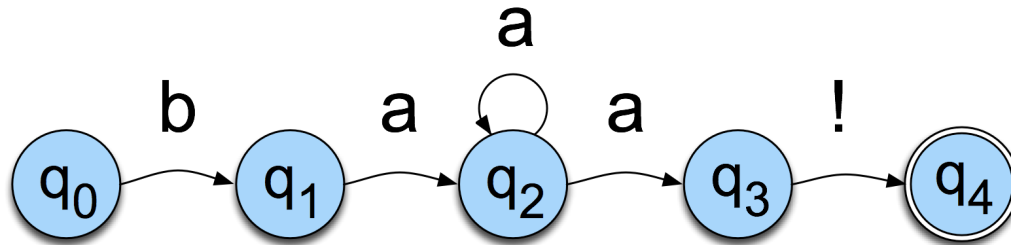
- We can say the following things about this machine
  - ◆ It has 5 states
  - ◆ **b**, **a**, and **!** are in its alphabet
  - ◆  $q_0$  is the start state
  - ◆  $q_4$  is an accept state
  - ◆ It has 5 transitions





# But Note

- There are other machines that correspond to this same language



- More on this one later

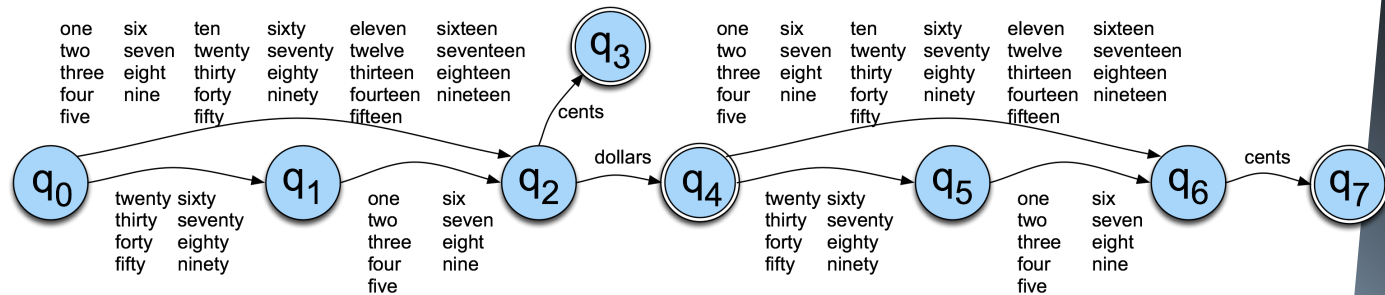
# More Formally

- You can specify an FSA by enumerating the following things.
  - ◆ The set of states:  $Q$
  - ◆ A finite alphabet:  $\Sigma$
  - ◆ A start state
  - ◆ A set of accept/final states
  - ◆ A transition function that maps  $Q \times \Sigma$  to  $Q$

# About Alphabets

- Don't take term **alphabet** word too narrowly; it just means we need a finite set of symbols in the input.
- These symbols can and will stand for bigger objects that can have internal structure.

# Dollars and Cents

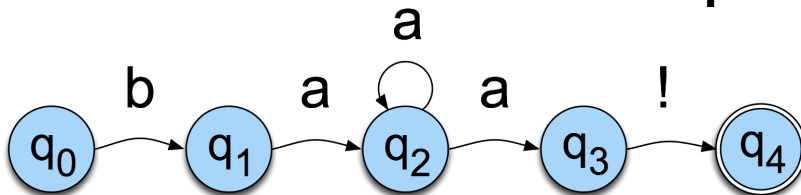


# Yet Another View

- The guts of FSAs can ultimately be represented as tables

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

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

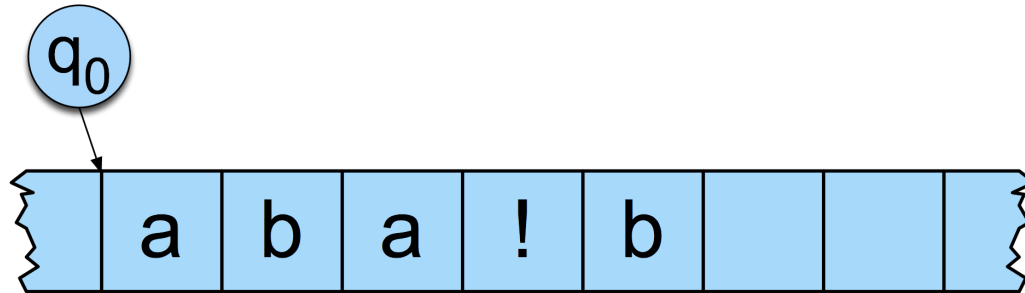


# Recognition

- 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
- 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*  $\leftarrow$  Beginning of tape

*current-state*  $\leftarrow$  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

**elseif** *transition-table*[*current-state*, *tape*[*index*]] is empty **then**

**return** reject

**else**

*current-state*  $\leftarrow$  *transition-table*[*current-state*, *tape*[*index*]]

*index*  $\leftarrow$  *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

# Recognition as Search

- You can view this algorithm as a trivial kind of **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
- It is trivial because?

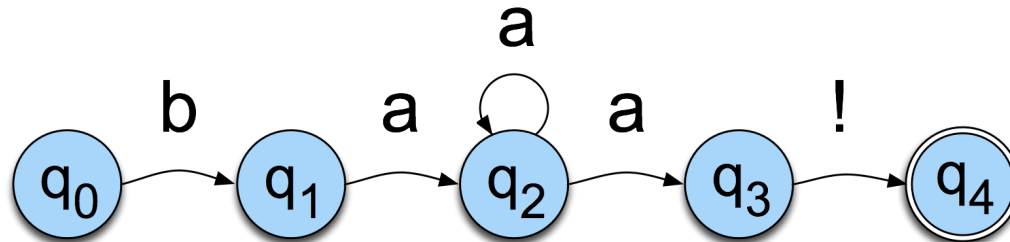
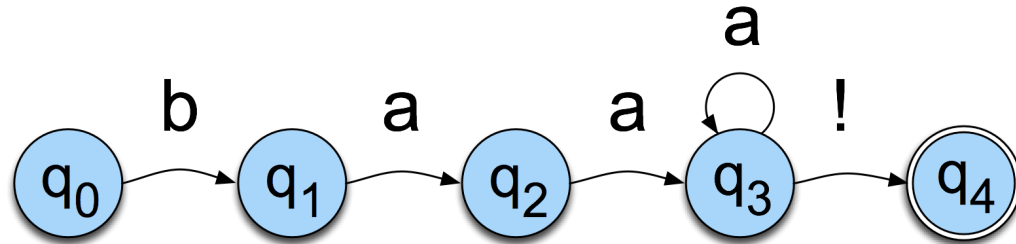
# Generative Formalisms

- **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

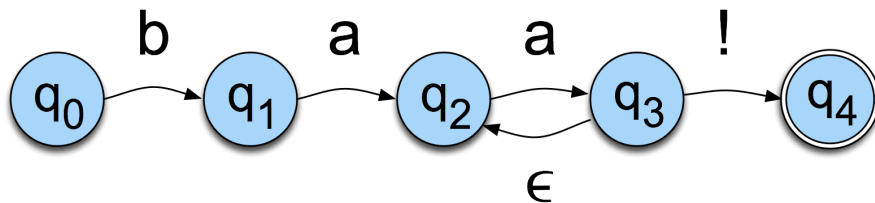
- FSAs can be viewed from two perspectives:
  - ◆ Acceptors that can tell you if a string is in the language
  - ◆ Generators to produce **all and only** the strings in the language

# Non-Determinism



# Non-Determinism cont.

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





# Equivalence

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

# ND Recognition

- Two basic approaches (used in all major implementations of regular expressions, see Friedl 2006)
  1. Either take a ND machine and convert it to a D machine and then do recognition with that.
  2. Or explicitly manage the process of recognition as a state-space search (leaving the machine as is).

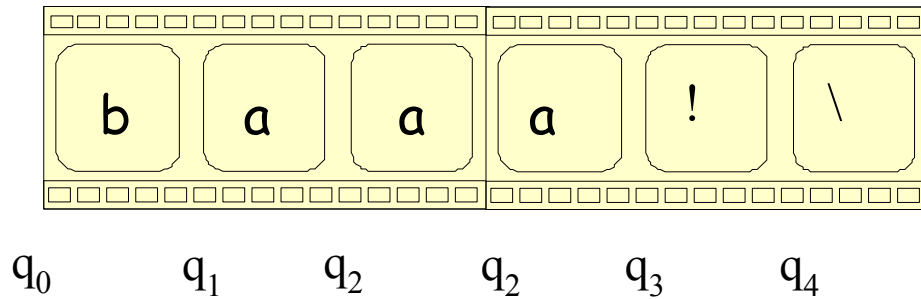
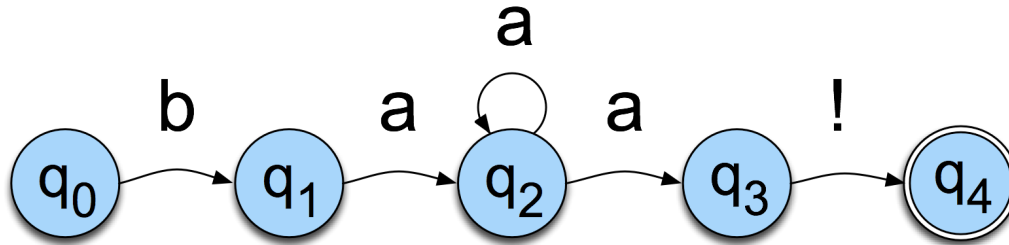
# Non-Deterministic Recognition: Search

- 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

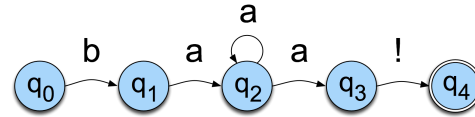
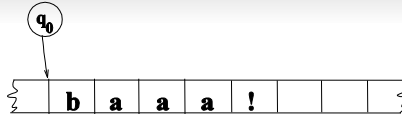
- So **success** in non-deterministic recognition occurs when a path is found through the machine that ends in an accept.
- **Failure** occurs when **all** of the possible paths for a given string lead to failure.

# Example

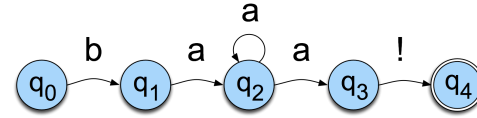
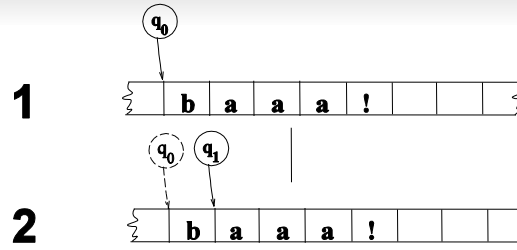


# Example

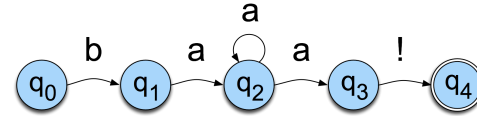
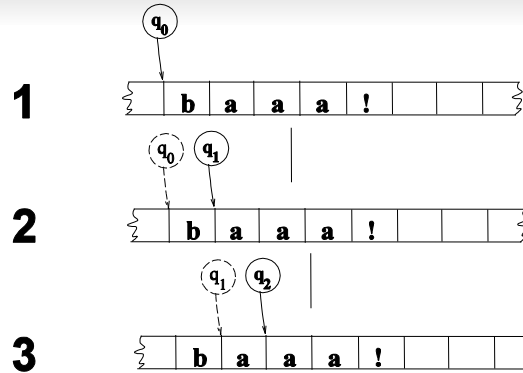
1



# Example

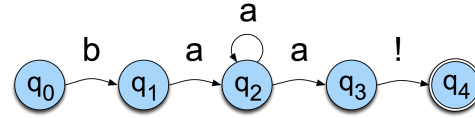
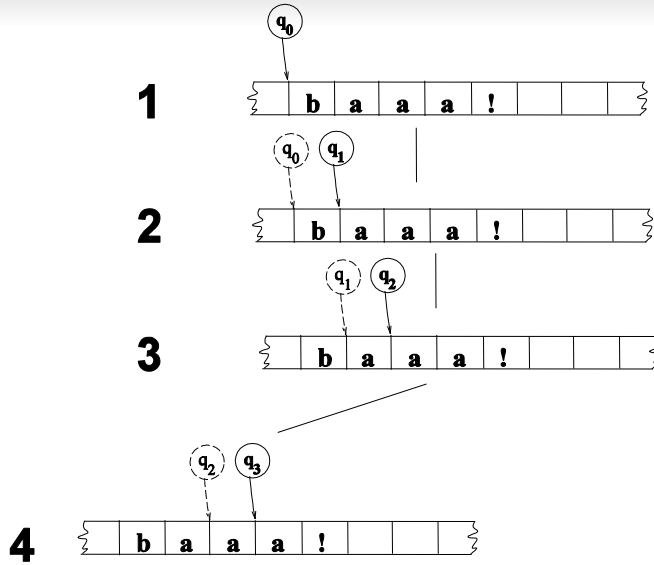


# Example

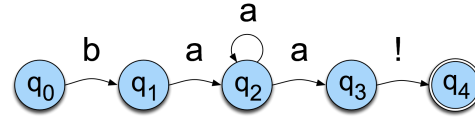
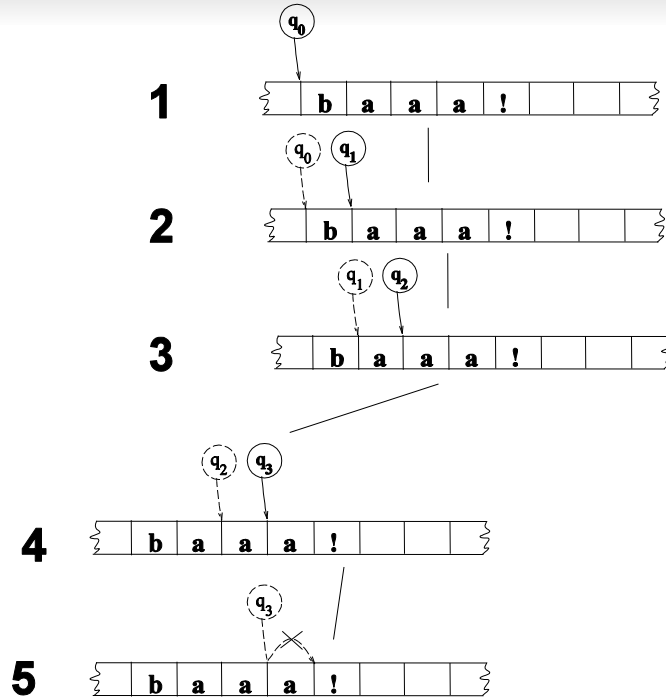




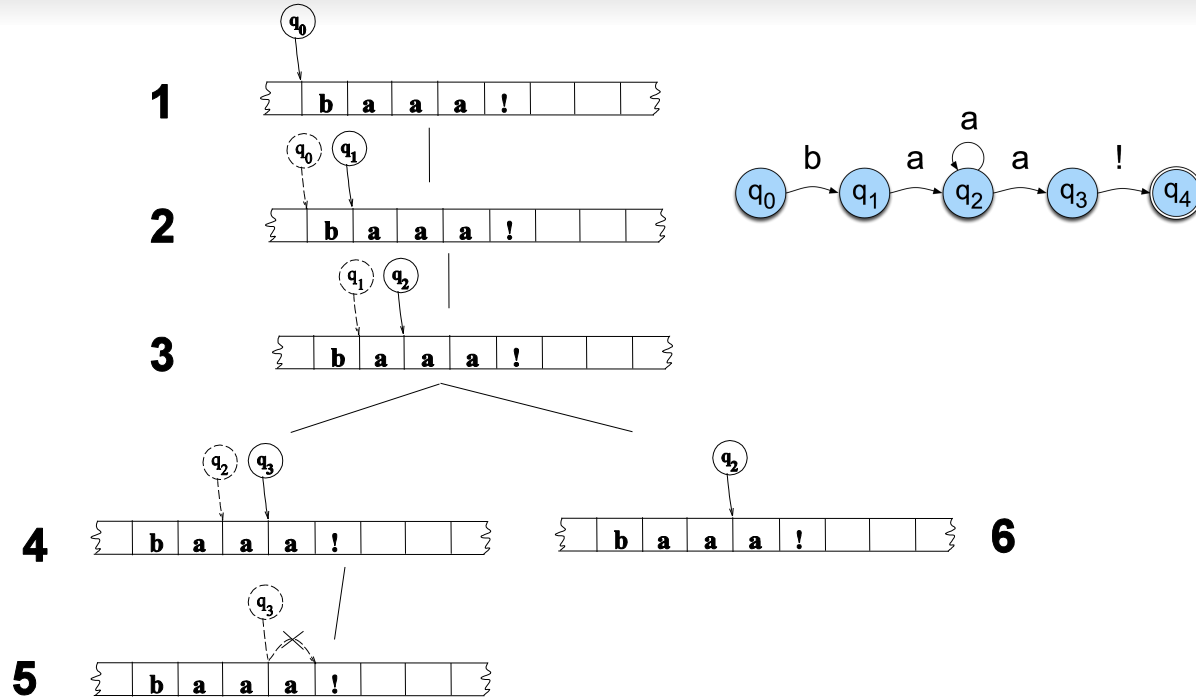
# Example



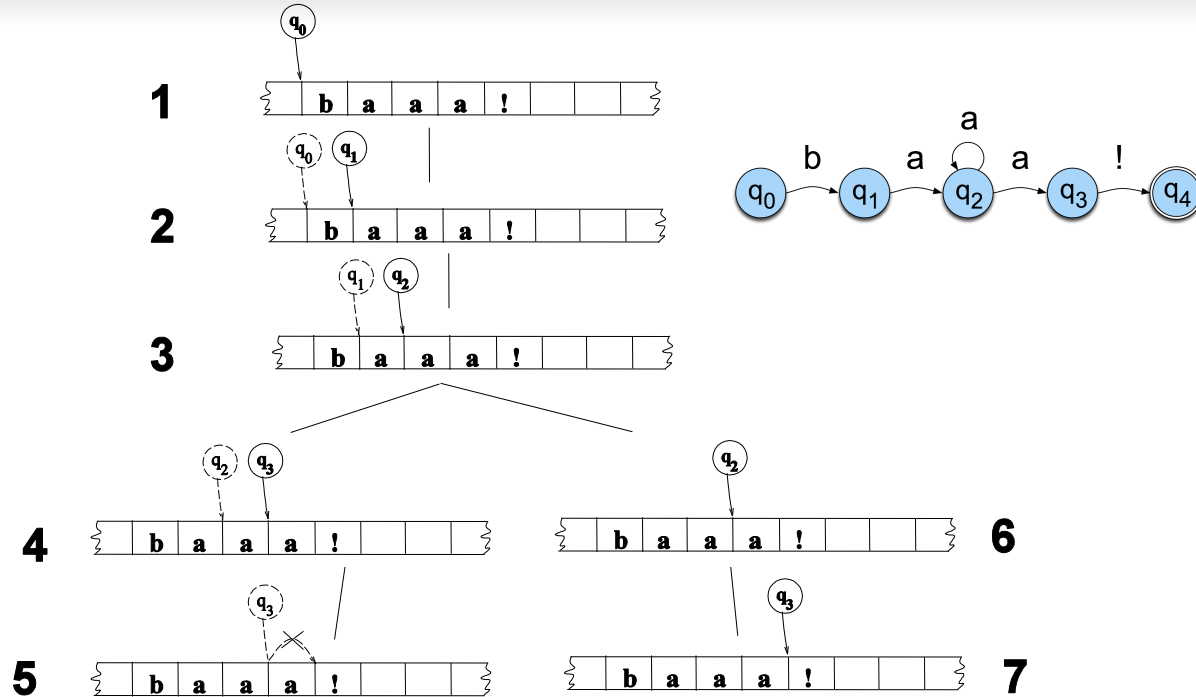
# Example



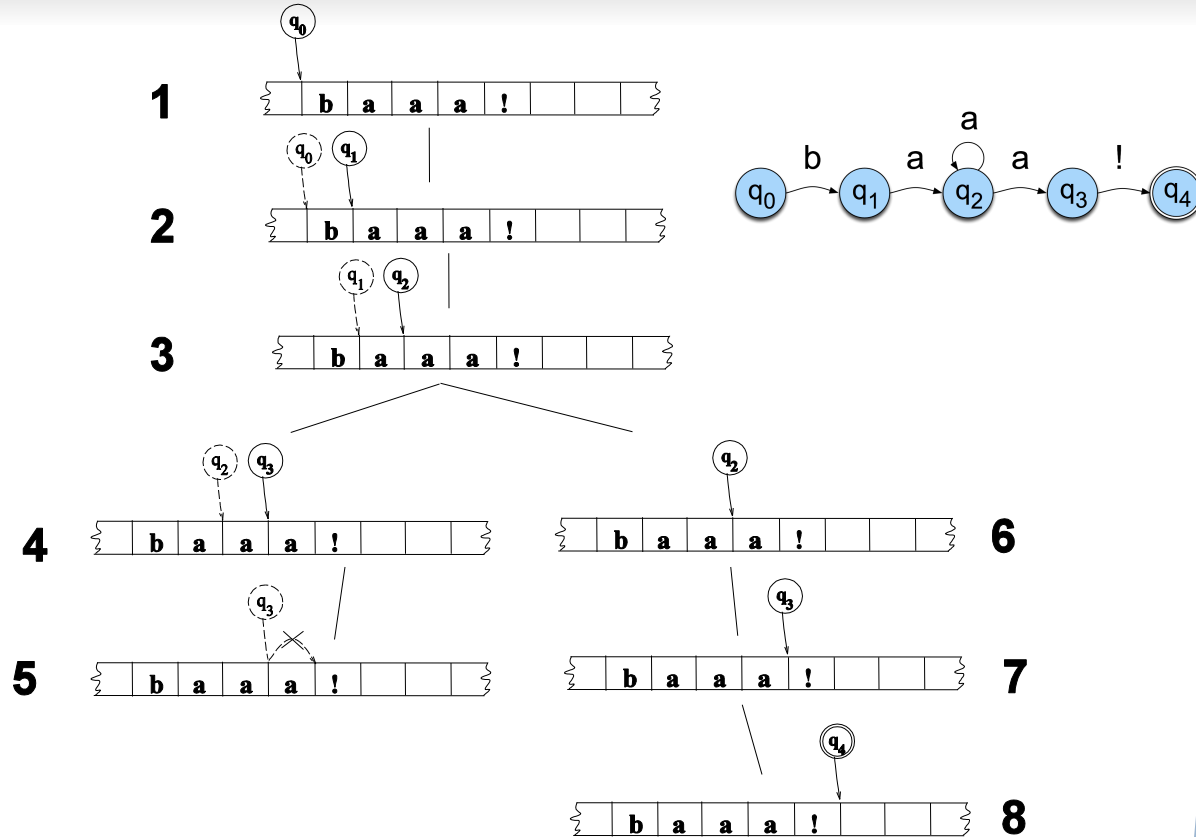
# Example



# 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.

# Why Bother?

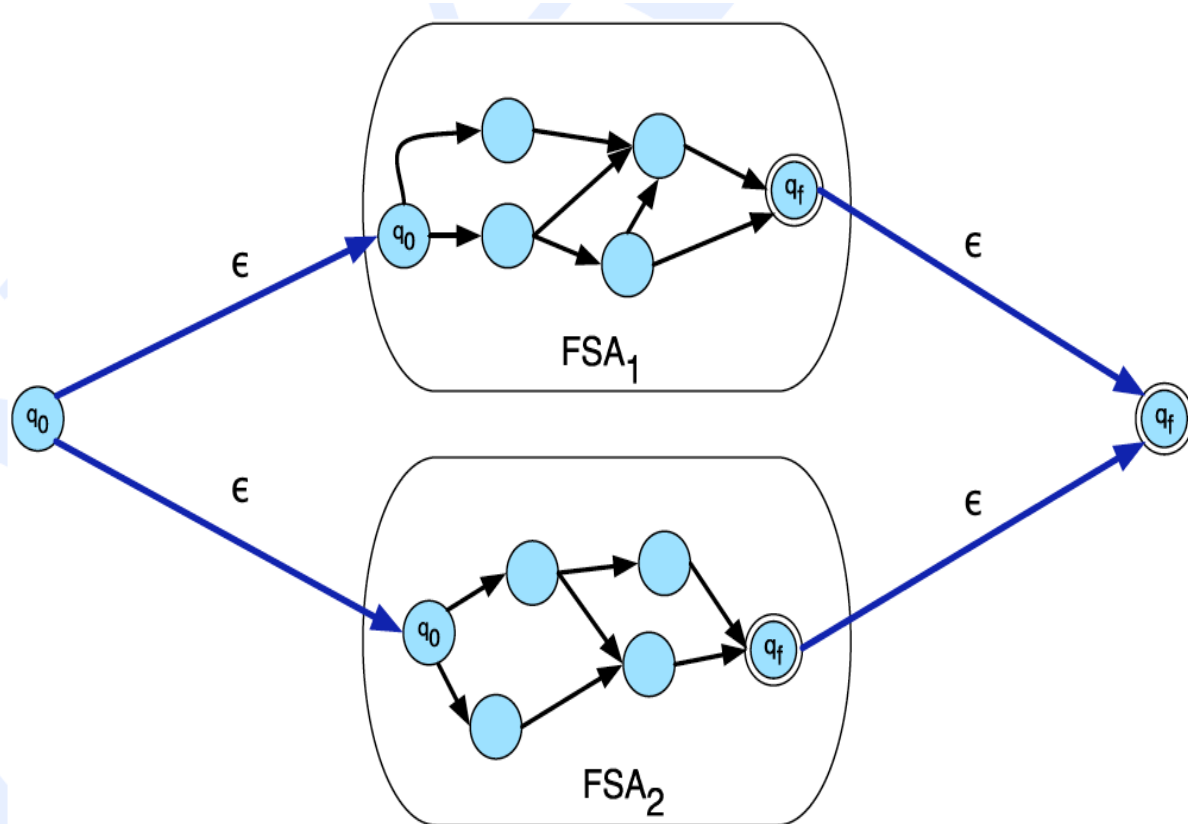
- Non-determinism doesn't get us more formal power and it causes headaches so why bother?
  - ◆ More natural (understandable) solutions
  - ◆ Compactness (sometimes)

# Compositional Machines

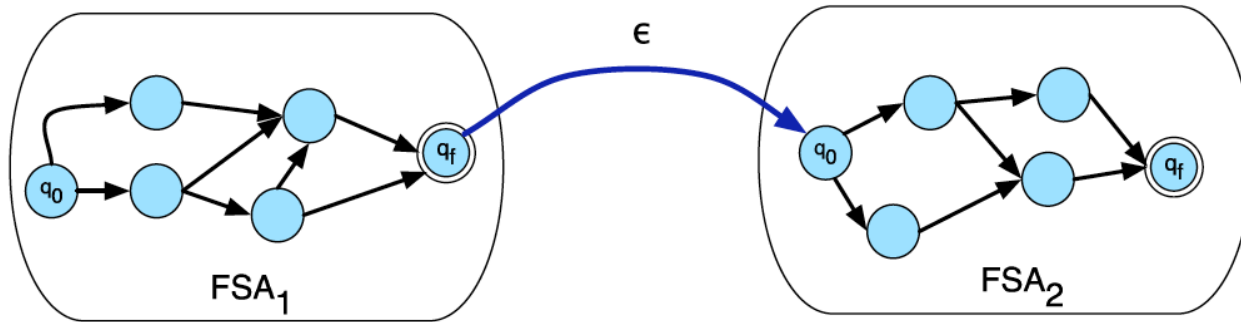
- Formal languages are just **sets** of strings
- Therefore, we can talk about various **set operations** (intersection, union, concatenation)
- This turns out to be a useful exercise



# Union



# Concatenation



# Basic Text Processing

Word tokenization

# Text Normalization

- Every NLP task needs to do text normalization:
  1. Segmenting/tokenizing words in running text
  2. Normalizing word formats
  3. Segmenting sentences in running text

# How many words?

- I do uh main- mainly business data processing
  - Fragments, filled pauses
- Seuss's **cat** in the hat is different from other **cats**!
  - **Lemma**: same stem, part of speech, rough word sense
    - **cat** and **cats** = same lemma
  - **Wordform**: the full inflected surface form
    - **cat** and **cats** = different wordforms

# How many words?

they lay back on the San Francisco grass and looked at the stars and their

- **Type**: an element of the vocabulary.
- **Token**: an instance of that type in running text.
- How many?
  - 15 tokens (or 14)
  - 13 types (or 12)

# How many words?

**$N$**  = number of tokens

**$V$**  = vocabulary = set of types

$|V|$  is the size of the vocabulary

Church and Gale (1990):  $|V| > O(N^{\frac{1}{2}})$

	Tokens = $N$	Types = $ V $
<i>Switchboard phone</i>	<i>2.4 million</i>	<i>20 thousand</i>
<i>Shakespeare</i>	<i>884,000</i>	<i>31 thousand</i>
<i>Google N-grams</i>	<i>1 trillion</i>	<i>13 million</i>

# Issues in Tokenization

- Finland's capital → Finland Finlands Finland's ?
- what're, I'm, isn't → What are, I am, is not
- Hewlett-Packard → Hewlett Packard ?
- state-of-the-art → state of the art ?
- Lowercase → lower-case lowercase lower case ?
- San Francisco → one token or two?
- m.p.h., PhD. → ??

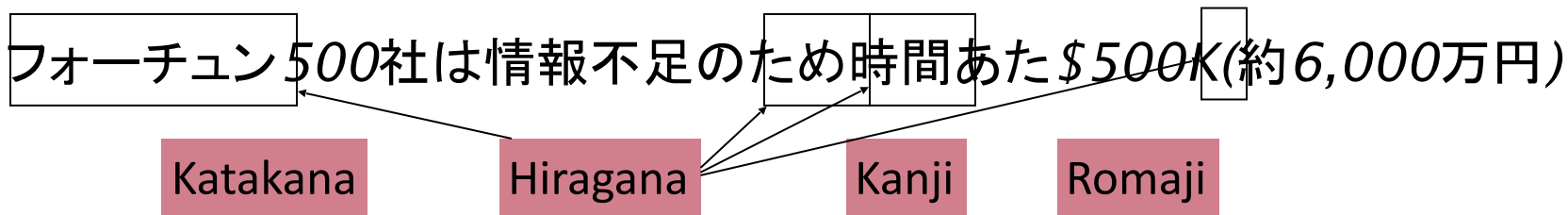


# Tokenization: language issues

- French
  - *L'ensemble* → one token or two?
    - *L ? L' ? Le ?*
    - Want *l'ensemble* to match with *un ensemble*
- German noun compounds are not segmented
  - *Lebensversicherungsgesellschaftsangestellter*
  - 'life insurance company employee'
  - German information retrieval needs **compound splitter**

# Tokenization: language issues

- Chinese and Japanese no spaces between words:
  - 莎拉波娃现在居住在美国东南部的佛罗里达。
  - 莎拉波娃 现在 居住 在 美国 东南部 的 佛罗里达
  - Sharapova now lives in US southeastern Florida
- Further complicated in Japanese, with multiple alphabets intermingled
  - Dates/amounts in multiple formats



End-user can express query entirely in hiragana!

# Word Tokenization in Chinese

- Also called **Word Segmentation**
- Chinese words are composed of characters
  - Characters are generally 1 syllable and 1 morpheme.
  - Average word is 2.4 characters long.
- Standard baseline segmentation algorithm:
  - Maximum Matching (also called Greedy)

## Maximum Matching Word Segmentation Algorithm

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

# Max-match segmentation illustration

- Thecatinthehat                      the cat in the hat
- Thetabledownthere                the table down there  
    theta bled own there
- Doesn't generally work in English!
- But works astonishingly well in Chinese
  - 莎拉波娃现在居住在美国东南部的佛罗里达。
  - 莎拉波娃 现在 居住 在 美国 东南部 的 佛罗里达
- Modern probabilistic segmentation algorithms even better

# Basic Text Processing

Word tokenization