Regular Languages

Lecture #2

Introduction to Natural Language Processing

CMPSCI 585, Fall 2007

University of Massachusetts Amherst

Andrew McCallum

Today’s Main Points

- A brief history
- What are regular languages, finite state automata and regular expressions?
- Writing regular expressions (in Python)
- Examples on several large natural language corpora
- Finite-state transducers, and morphology
- Homework assignment #1

Some brief history: 1950s

- Early CL on machines less powerful than pocket calculators.
- Foundational work on automata, formal languages, probabilities and information theory.
- First speech systems (Davis et al, Bell Labs).
- MT heavily funded by military, but basically just word substitution programs.
- Little understanding of natural language syntax, semantics, pragmatics.

Some brief history: 1960s

- Alvey report (1966) ends funding for MT in America - the lack of real results realized
- ELIZA (MIT): Fraudulent NLP in a simple pattern matcher psychotherapist
  - It's true, I am unhappy.
  - *Do you think coming here will make you not to be unhappy?*
  - I need some help; that much is certain.
  - *What would it mean to you if you got some help?*
  - Perhaps I could earn to get along with my mother.
  - *Tell me more about your family.*
- Early corpora: Brown Corpus (Kudera and Francis)

Some brief history: 1970s

- Winograd's SHRDLU (1971): existence proof of NLP (in tangled LISP code).
- Could interpret questions, statements commands.
  - Which cube is sitting on the table?
  - *The large green one which supports the red pyramid.*
  - Is there a large block behind the pyramid?
  - *Yes, three of them. A large red one, a large green cube, and the blue one.*
  - Put a small one onto the green cube with supports a pyramid.
  - *OK.*

Some brief history: 1980s

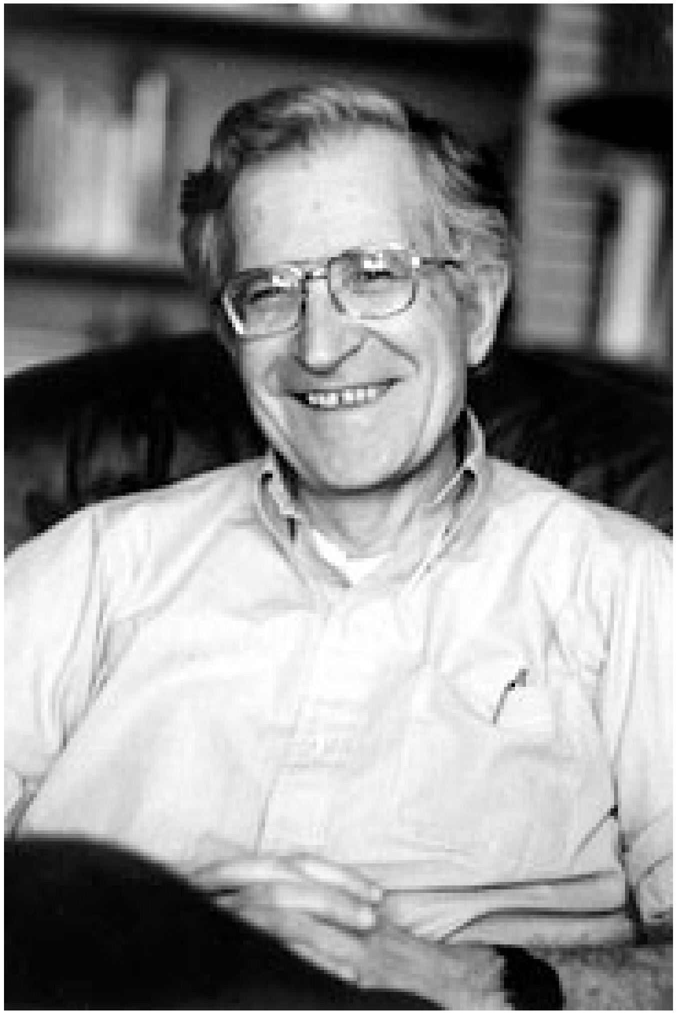
- Procedural --> Declarative (including logic programming)
- Separation of processing (parser) from description of linguistic knowledge.
- Representations of meaning: procedural semantics (SHRDLU), semantic nets (Schank), logic (perceived as answer; finally applicable to real languages (Montague)
- Perceived need for KR (Lenat and Cyc)
- Working MT in limited domains (METEO)

Some brief history: 1990s

- Resurgence of finite-state methods for NLP: in practice they are incredibly effective.
- Speech recognition becomes widely usable.
- Large amounts of digital text become widely available and reorient the field. The Web.
- Resurgence of probabilistic / statistical methods, led by a few centers, especially IBM (speech, parsing, Candide MT system), often replacing logic for reasoning.
- Recognition of *ambiguity* as key problem.
- Emphasis on machine learning methods.

Some brief history: 2000s

- A bit early to tell! But maybe:
  - Continued surge in probability, Bayesian methods of evidence combination, and joint inference.
  - Emphasis on meaning and knowledge representation.
  - Emphasis on discourse and dialog.
  - Strong integration of techniques, and levels: brining together statistical NLP and sophisticated linguistic representations.
  - Increased emphasis on unsupervised learning.



Noam Chomsky
1928 -

Chomsky Hierarchy
Generative Grammar
Liberatarian-Socialist

The most cited person alive.

A Language

Some sentences in the language

- The man took the book.

From [Chomsky, 1956], his first context-free parse tree.

- The purple giraffe hopped through the clouds.
- This sentence is false.

Some sentences not in the language

- *The girl, the sidewalk, the chalk, drew.
- *Backwards is sentence this.
- *loDvaD tlhIngan Hol ghojmoH be.

Compact description of a language

- Start with some “non-terminal” symbol, ***S***.
- Expand that symbol, using some substitution **rules**.
- ...keep applying rules until all non-terminals are expanded to terminals.
- The string of terminals is in the sentence.

Chomsky Hierarchy

Linguistic example:

- Type 0 languages (Turing-equivalent)
  Rewrite rules a → b
  where a, b are any string of terminals and non-terminals

ATNs

- Context-sensitive languages
  Rewrite rules aXb → acb
  where X is non-terminal and a,b as above

TAGs

- Context-free languages
  Rewrite rules X → a
  where X, a, b as above

PSGs

- Regular languages
  Rewrite rules X → aY
  where X, Y are non-terminals and a is a string of terminals

FSAs

More detail on all this again later.

Regular language example

- Non-terminals:
  - **S, X, Y, Z**
- Terminals:
  - **m, o**
- Rules:

  S → mX
  X → oY
  Y → o
  Y →
- Start symbol:

  S

An expansion:

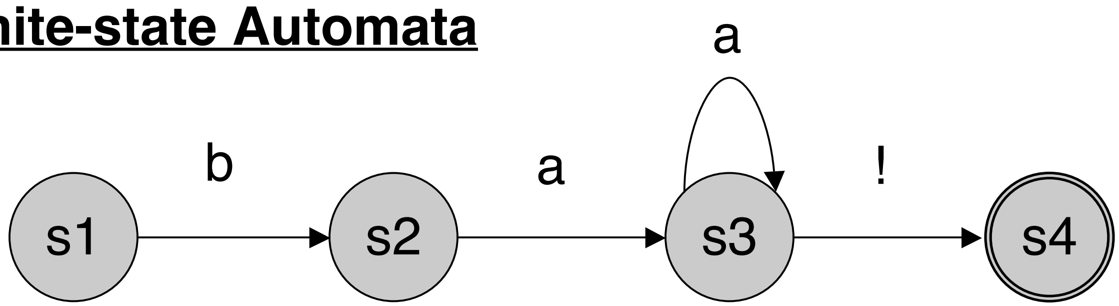
S
mX
moY
mooY
moooY
mooo

Example: Sheep Language

Strings in and out of the example Regular Language:

- In the language:
  "ba!", "baa!", "baaaaa!"
- Not in the language:
- "ba", "b!", "ab!", "bbaaa!", "alibaba!"

Finite-state Automata



double circle
indicates "accept state"

Regular Expression

```
baa*
```

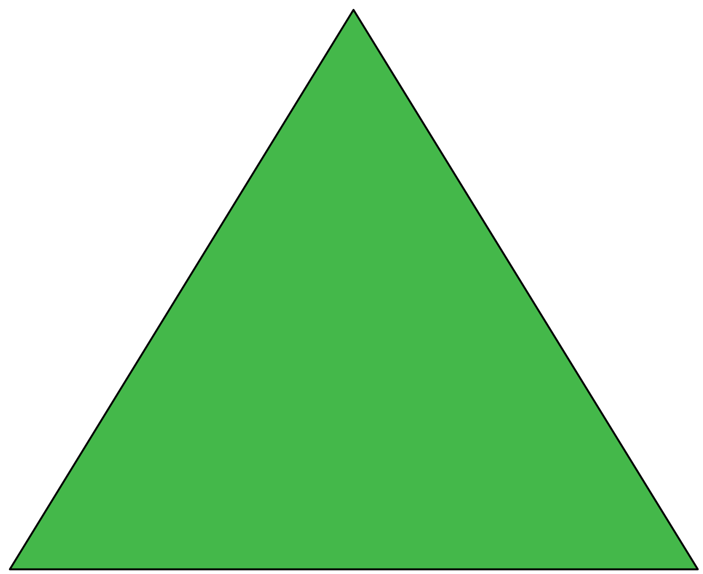
Recognizer

- A recognizer for a language is a program that takes as input a string W and answers "yes" if W is a sentence in the language, and answers "no" otherwise.

- We can think of this as a machine that emits only two possible responses it input.

Regular Languages: related concepts

Regular Languages
the accepted strings

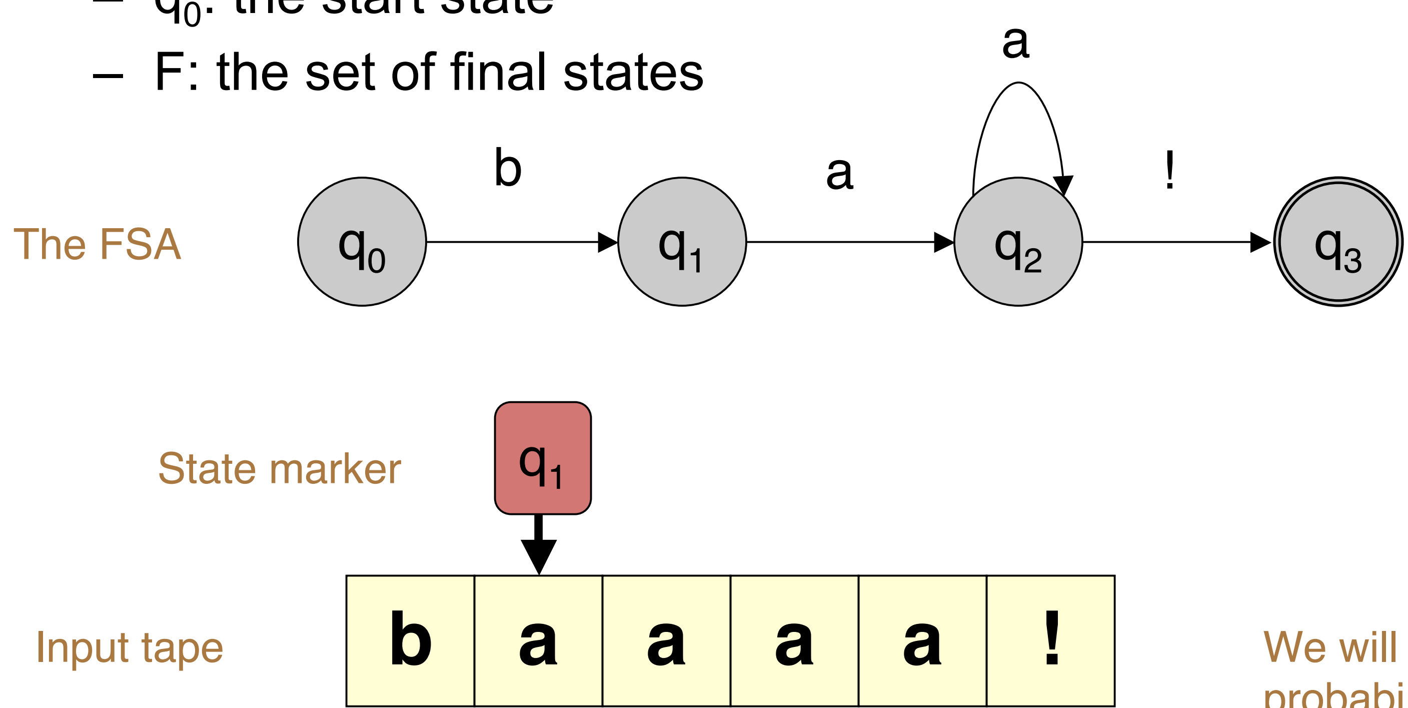


Finite-state Automata
machinery for accepting

Regular Expressions
a way to type the automata

Finite State Automata, more formally

- A finite state automata is a 5-tuple: $(Q, \Sigma, q_0, F, \delta(q,i))$
  - Q : finite set of N states, $q_0, q_1, q_2, \ldots q_N$ (non-terminals)
  - Σ : finite set of (terminals)
  - $\delta(q,i)$: transition function, given state and input, returns next state (production rules)
  - q_0: the start state
  - F: the set of final states



We will later return to a probabilistic version of this with Hidden Markov Models!

Transition Table, δ

State	Input *b*	*a*	*!*
0	1	∅	∅
1	∅	2	∅
2	∅	2	3
3	∅	∅	∅

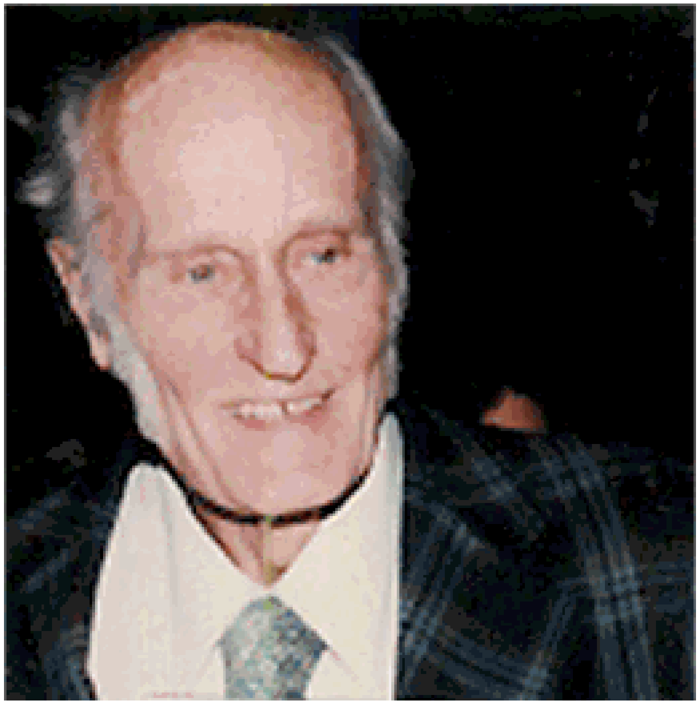
Regular Expressions
The “foundational” operations

	Pattern	Matches
Concatenation	**abc**	*abc*
Disjunction	**a|b**	*a* *b*
	(a|bb)d	*ad* *bbd*
Kleene star	**a***	*ε a aa aaa ...*
	c(a|bb)*	*ca cbba*

The empty string

Regular expressions / Finite-state automata are “closed under these operations”

Stephen Kleene, 1909 - 1994



Attended Amherst College!

Best known for founding the branch of mathematical logic known as recursion theory, together with Alonzo Church, Kurt Godel, Alan Turing and others; and for inventing regular expressions.

*“**Kleene**liness is next to **Godel**iness.”*

Practical Applications of RegEx's

- Web search
- Word processing, find, substitute
- Validate fields in a database (dates, email addr, URLs)
- Searching corpus for linguistic patterns
  – and gathering stats...

- Finite state machines extensively used for
  – acoustic modeling in speech recognition
  – information extraction (e.g. people & company names)
  – morphology
  – ...

Two types of characters in REs

- **Literal**
  - Every normal text character is an RE, and denotes itself.
- **Meta-characters**
  - Special characters that allow you to combine REs in various ways
  - Example:
    a denotes *a*
    a* denotes ε or *a* or *aa* or *aaa* or ...

Basic Regular Expressions

	Pattern	Matches
Character Concat	**went**	went
Alternatives	**(go|went)**	go went
	[aeiou]	a o u
disjunc. negation	**[^aeiou]**	b c d f g
wildcard char	**.**	a z &
Loops & skips	**a***	ε a aa aaa ...
one or more	**a+**	a aa aaa
zero or one	**colou?r**	color colour

Andrew McCallum, UMass Amherst,
including material from Chris Manning and Jason Eisner

More Fancy Regular Expressions

- Special characters
  - **`\t`** tab **`\v`** vertical tab
  - **`\n`** newline **`\r`** carriage return
- Aliases (shorthand)
  - **`\d`** digits [0-9]
  - **`\D`** non-digits [^0-9]
  - **`\w`** alphabetic [a-zA-Z]
  - **`\W`** non-alphabetic [^a-zA-Z]
  - **`\s`** whitespace [\t\n\r\f\v]
  - **`\w`** alphabetic [a-zA-Z]
- Examples
  - **`\d+ dollars`** 3 dollars, 50 dollars, 982 dollars
  - **`\w*oo\w*`** food, boo, oodles
- Escape character
  - **`\`** is the general escape character; e.g. **`\.`** is not a wildcard, but matches a period **`.`**
  - if you want to use **`\`** in a string it has to be escaped **`\\`**

Yet More Fancy Regular Expressions

- Anchors. AKA, "zero width characters".
- They match positions in the text.
  - ^ beginning of line
  - **$** end of line
  - **\b** word boundary, i.e. location with **\w** on one side but not on the other.
  - **\B** negated word boundary, i.e. any location that would not match **\b**
- Examples:
  - **\bthe\b** <u>the</u> together
- Counters {1}, {1,2}, {3,}

Even More Fancy Regular Expressions

- Grouping
  - **`a (good|bad) movie`**
  - **`He said it (again and )*again.`**

- Parens also indicate Registers (saved contents)
  - **`b(\w+)h\1`**
    matches *boohoo* and *baha*, but not *boohaa*
    The digit after the \ indicates which of multiple paren groups, as ordered by when then were opened.

- Grouping without the cost of register saving
  - **`He went (?:this|that) way.`**

Extra Fancy Regular Expressions

- Non-consuming tests
  - `(?=...)` - Positive lookAHEAD
  - `(?!...)` - Negative lookAHEAD
  - `(?<=...)` - Positive lookBEHIND
  - `(?<!...)` - Negative lookBEHIND
- Example
  - **`(?<=[Hh]e) \w+ed (?=\w+ly)`**

Yowza! Regular Expressions

- Multi-line, with comments, etc.

```
FROM = re.compile("""
      ^From:                  # Anchor to start of line
      \s*                     # maybe some spaces
      (?P<user>\w+)           # ‘user’: group of word characters
      @
      (?P<domain>             # the ‘domain’:
      \S+)                    # some non-space characters
      \s                      # finally, a space character
""",re.VERBOSE)
```

Oral Quiz:
Describe the strings these will accept

- `ab+a`
- `(ab)*`
- `([^aeiou][aeiou])\1`
- `\bdis\w+\b...`

Andrew McCallum, UMass Amherst,
including material from Chris Manning and Jason Eisner

Hands-on!
With little strings

- re_show function from NLTK

```
import re
def re_show(regexp, string):
 print re.compile(regexp, re.M).sub("{\g<0>}", string.rstrip())
```

Hands-on!
Investigating corpora

- `regexs.py`
- `regexcount.py`
- You need Python version 2.4 or higher.

- Corpora
  - 200k+ words of Wall Street Journal articles
  - 3 books by Jane Austin
  - Several plays by William Shakespeare
  - Several books by Mark Twain
  - and various other text from Walt Whitman, & others...

These are available on the course Web site

ELIZA (Weizenbaum 1966)

User: *Men are all alike.*

ELIZA: IN WHAT WAY?

User: *They're always bugging us about something or other.*

ELIZA: CAN YOU THINK OF A SPECIFIC EXAMPLE?

User: *Well, my boyfriend made me come here.*

ELIZA: YOUR BOYFRIEND MADE YOU COME HERE.

User: *He say's I'm depressed much of the time.*

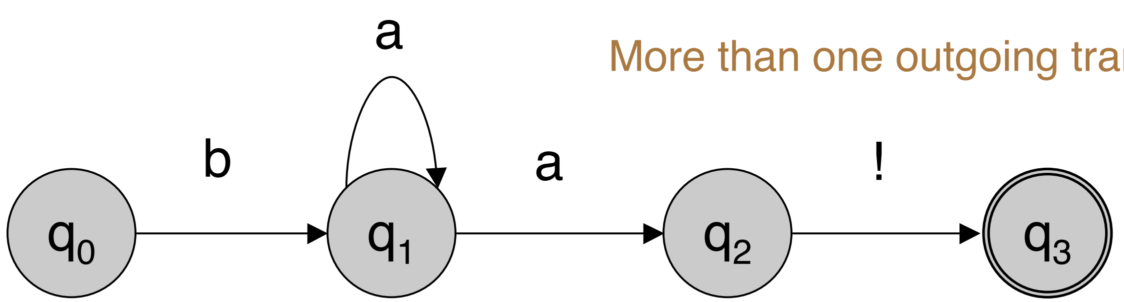
ELIZA: I'M SORRY TO HEAR THAT YOU ARE DEPRESSED.

Implemented with regular expression substitution!

s/.* I'm (depressed|sad) .*/I AM SORRY TO HEAR THAT YOU ARE \1/

s/.* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE?/

Non-deterministic FSAs



More than one outgoing transition on 'a'

State	Input *b*	Input *a*	Input *!*
0	1	∅	∅
1	∅	1,2	∅
2	∅	∅	3
3	∅	∅	∅

Transition *relation*, rather than transition function.

Non-deterministic finite-state automata as "Recognizers"

- The problem:

  When processing a string, we might follow the wrong transition, and reject the string when we should have accepted it!

One solution: turn the NFA into a DFA... (See CMPSCI 250)

- Ubiquitous problem in this course:
  How to efficiently search through various possible "paths" (parses) to find one that works / the most likely one, etc.

How do humans do this?!

Solutions

- **Look-ahead**
  - Peek ahead to help decide which path to take.
- **Parallelism**
  - At each choice, take every path in parallel.
- **Backup**
  - At each choice point, mark the input / state
  - If we fail, go back and try another path
    Need a *stack* (or *queue)* of markers
  - Marker = “Machine state”
  - Collection of current state & markers = “Search state”
  - Depth-first search (or Breadth-first search).

“Smart” heuristic search, “A*”. See CMPSCI 383 (Artificial Intelligence)

RE / FSA equivalence proof

- How would you do it?

Morphology

The study of the sub-word units of meaning.

disconnect

"not" "to attach"

Making a word plural:

		Examples:	
If word is regular,	add s	dog	dogs
If word ends in y,	change y to i, and add s	baby	babies
If word ends in x,	add -es	fox	foxes
. . .			

Recognizing that ***foxes*** breaks down into morphemes ***fox*** and ***-es*** called ***Morphological Parsing***

Parsing = taking an input and producing some sort of structure for it.

Morphology, briefly

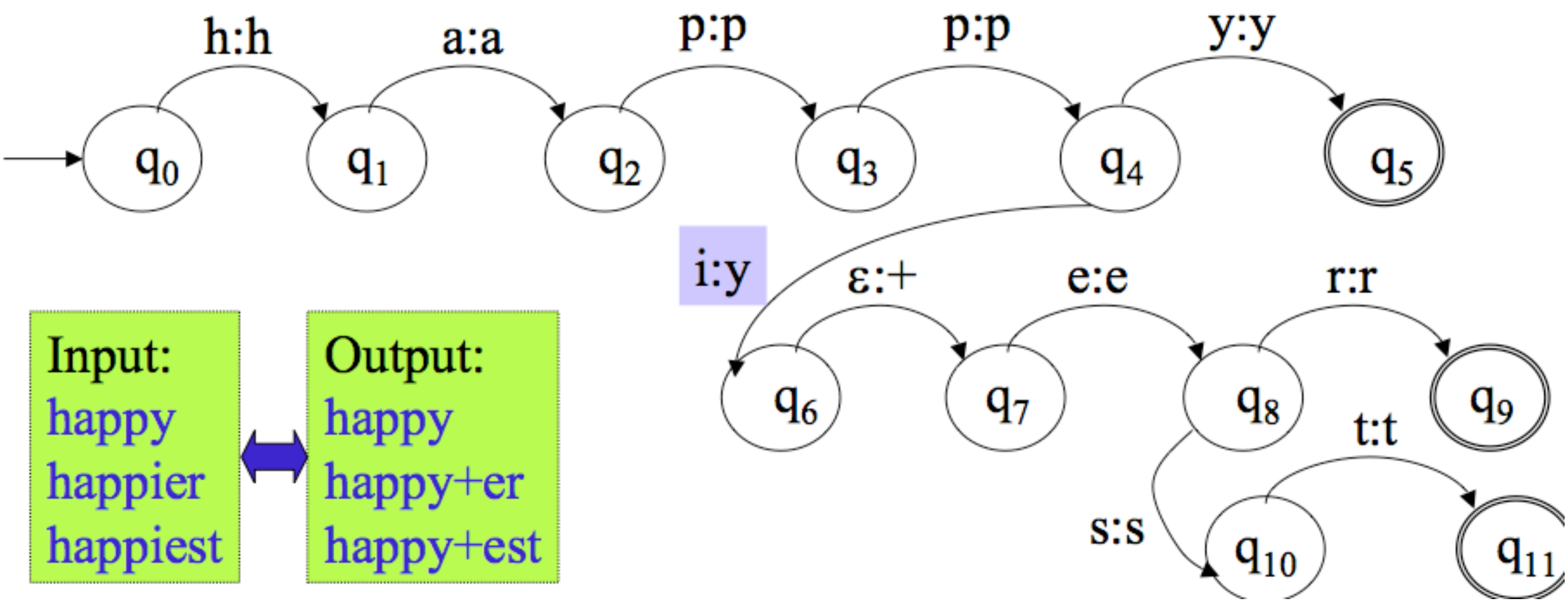
- ***morpheme***: minimal meaning-bearing unit
  - ***stem***: "main" morpheme of a word, e.g. ***fox***
  - ***affixes***: add "additional" meanings, e.g. ***+es***
    includes ***prefixes, suffixes, infixes, circumfixes***,
    e.g. ***un-***, ***-ly***, ...
  - concatenative morphology, non-concatenative
- ***inflection***: stem+morpheme in the same class as stem.
  - e.g. nouns plural ***+s***, possessive ***+'s***
- ***derivation***: stem+morpheme in different class...
  - e.g. ***+ly*** makes and adverb from an adjective

Morphological Parsing with Finite State Transducers

- We want a system that given ***foxes*** will output a parse: ***fox+es*** or ***fox +PL***
- FSAs will take input, but not produce output (other than “accept”/”reject”)

- Solution: **Finite State Transducers** (FST):
  – A FST is a two-tape automaton that recognizes or generates **pairs** of strings.

Example Finite-state Transducer



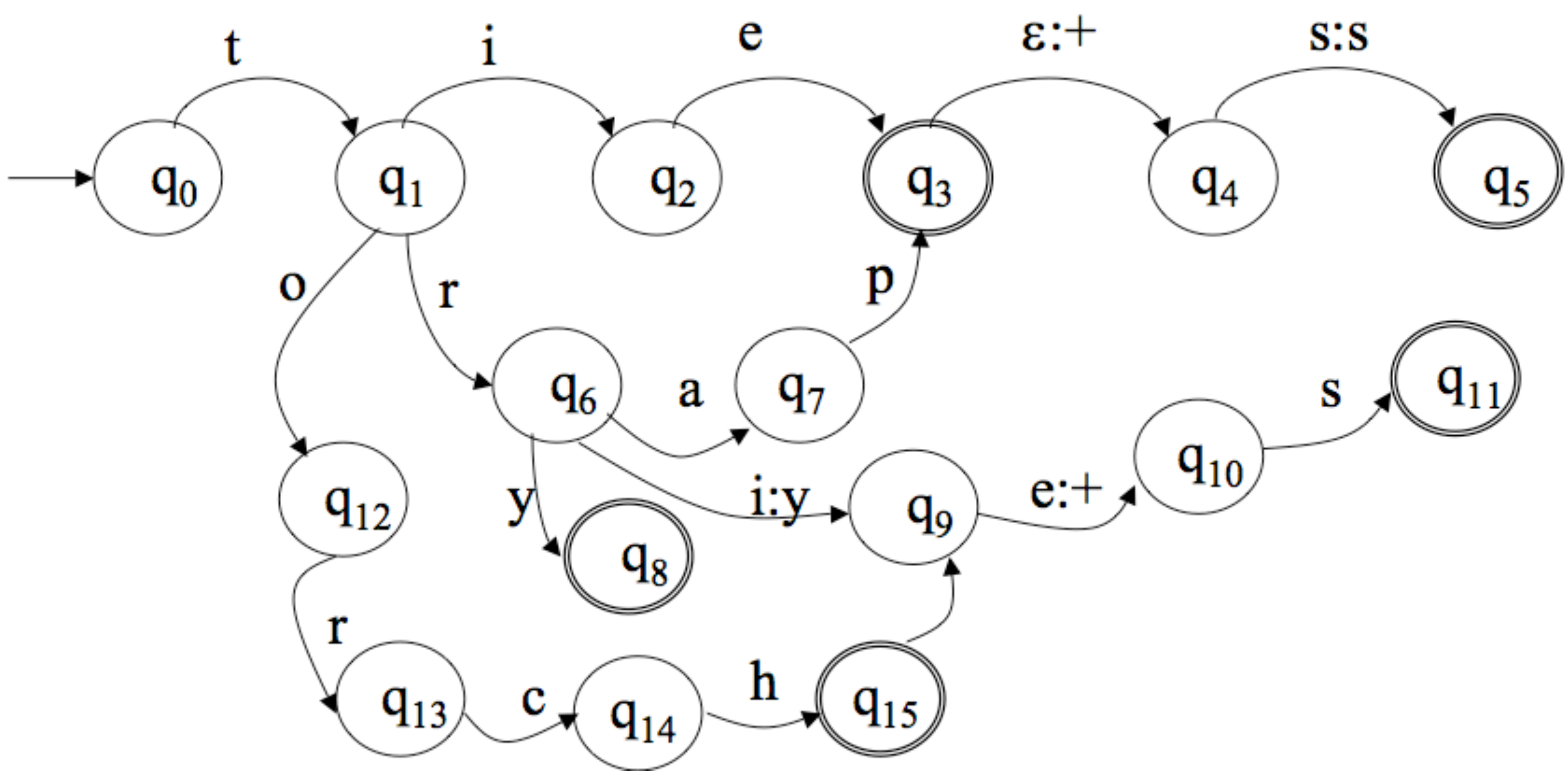
FSTs can be used to transform a word surface form into morphemes (or vice-versa!)

An entire lexicon can be encoded as a FST.

FST transition table

	Input									
State	h:h	a:a	p:p	y:y	i:y	ε:+	e:e	r:r	s:s	t:t
0	**1**	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
1	Ø	2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
2	Ø	Ø	3	Ø	Ø	Ø	Ø	Ø	Ø	Ø
3	Ø	Ø	**4**	Ø	Ø	Ø	Ø	Ø	Ø	Ø
4	Ø	Ø	Ø	**5**	**6**	Ø	Ø	Ø	Ø	Ø
5:	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
6	Ø	Ø	Ø	Ø	Ø	**7**	Ø	Ø	Ø	Ø
7	Ø	Ø	Ø	Ø	Ø	Ø	**8**	Ø	Ø	Ø
8	Ø	Ø	Ø	Ø	Ø	Ø	**10**	**9**	Ø	Ø
9:	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

Fragment of a lexicon in a FST



Further Closure Properties of FSAs

Regular languages are also closed under the following operations

- **Reversal**: If L1 is regular, so is the language consisting of the set of all reversals of strings in L1.
- **Intersection**: if L1 and L2 are regular languages, so is the language consisting of all strings that are in both L1 and L2.
- **Difference**: If L1 and L2 are regular languages, so is the language consisting of all strings in L1 that are not in L2.
- **Complementation**: If L1 is a regular language, so is the set of all possible strings that are not in L1.

Announcement: *Undergraduate* CMPSSCI Meeting

- **“First Friday”**
  - Curriculum Information
  - Spring Events
  - Jobs/Co-ops/Research positions in and out of the Department
  - Library Carrels
  - And More!

- **Friday, September 7, 2007**
  (3pm for new or transfer students)
  3:30 - 5:00 pm
  CMPS 150/151 (Computer Science Building)
  Refreshments will be served.

Next class (Tuesday Feb 7)

- Learning Python
  - Variables, operators, conditionals, iteration, etc.
  - functions, classes, modules

  - Gather statistics from Python-ized Penn Treebank.
  - Calculate statistics from 200k words of WSJ
  - Implement a phrase structure grammar, and generate sentences from it.

- Install Python, and bring your laptop with you!

First Homework, assigned today!

- Essentially:
  - Write some regular expressions
  - Run them on some corpora
  - Write ~1 page about your experience and findings
  - Extra credit for creativity and interesting application!

- Feel free to come do it in office hours!

- Due next Thursday, one week from today.
  (Don't wait until Wednesday to install Python!
- Recommended schedule:
  - Idea by Saturday
  - Coded/tested by Monday
  - Write-up by Wednesday

Office Hours, CS Building, Rm 264

- Monday, ??
- Tuesday, ??
- Wednesday, am and evening
- **Thursday, 4-5pm**
- **Friday, 2-4pm**

- If you can’t make any of these times, let me know.

Aside: Grammar Induction

- Also called “Grammatical Inference”
- “Learning” finite-state automata from many examples of strings in (and out of) the language.
- http://www.info.ucl.ac.be/~pdupont/pdupont/gram.html
- Learning FSA and CFG structure from data!

Thank you!