Regular Expressions and Automata

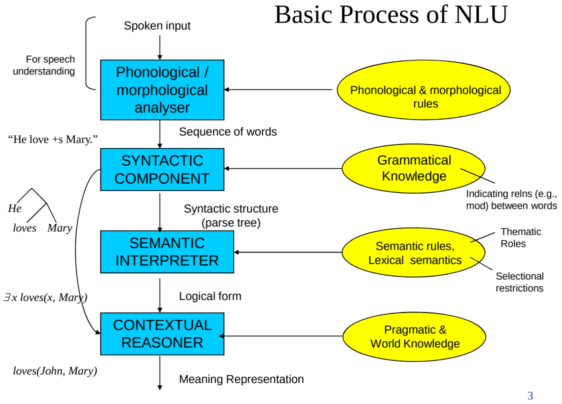
Lecture #2

September 3
2009

Natural Language Understanding

- Associate each input (acoustic signal/character string) with a meaning representation.
- Carried out by a series of components:
  - Each component acts as a translator from one representation to another
  - In general, each component adds successively ‘richer’ information to the output

Basic Process of NLU



Representations and Algorithms for NLP

- Representations: formal models used to capture linguistic knowledge
- Algorithms manipulate representations to analyze or generate linguistic phenomena
- Simplest often produce best performance but….the 80/20 Rule and “low-hanging fruit”

NLP Representations

- State Machines
  - FSAs, FSTs, HMMs, ATNs, RTNs
- Rule Systems
  - CFGs, Unification Grammars, Probabilistic CFGs
- Logic-based Formalisms
  - 1st Order Predicate Calculus, Temporal and other Higher Order Logics
- Models of Uncertainty
  - Bayesian Probability Theory

NLP Algorithms

- Most are parsers or transducers: accept or reject input, and construct new structure from input
  - State space search
    - Pair a partial structure with a part of the input
    - Spaces too big and ‘best’ is hard to define
  - Dynamic programming
    - Avoid recomputing structures that are common to multiple solutions

Today

- Review some of the simple representations and ask ourselves how we might use them to do interesting and useful things
  - Regular Expressions
  - Finite State Automata
- How much can you get out of a simple tool?

Regular Expressions

- Can be viewed as a way to specify:
  - Search patterns over text string
  - Design of a particular kind of machine, called a Finite State Automaton (FSA)
- These are really equivalent

Uses of Regular Expressions in NLP

- As grep, perl: Simple but powerful tools for large corpus analysis and ‘shallow’ processing
  - What word is most likely to begin a sentence?
  - What word is most likely to begin a question?
  - In your own email, are you more or less polite than the people you correspond with?
- With other unix tools, allow us to
  - Obtain word frequency and co-occurrence statistics
  - Build simple interactive applications (e.g., Eliza)
- Regular expressions define regular languages or sets

Regular Expressions

- **Regular Expression**: Formula in algebraic notation for specifying a set of strings
- **String**: Any sequence of alphanumeric characters
  - Letters, numbers, spaces, tabs, punctuation marks
- **Regular Expression Search**
  - Pattern: specifying the set of strings we want to search for
  - Corpus: the texts we want to search through

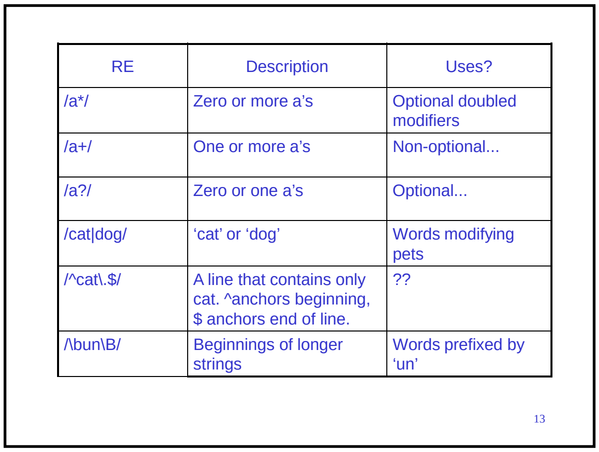
Simple Example

- Sequence of simple characters:

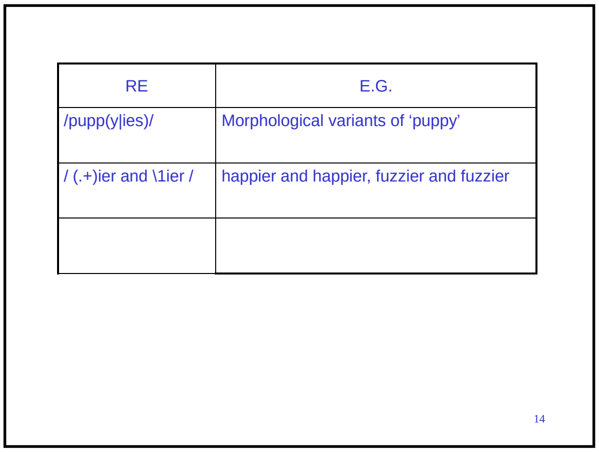
RE	DESCRIPTION	USES
/This/	Matches the string “This”	Finding the word “this” starting a sentence
/this/	Matches the string “this”	Finding the word “this” internal to a sentence.
/[Tt]his/	Matches either “This” or “this” -- disjunction	Finding the word “this” anywhere in a sentence.

Some Examples

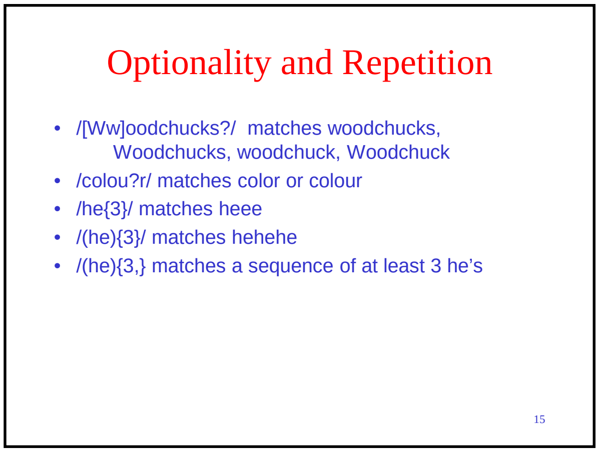
RE	Description	Uses
/./	Wild card; Any char except <cr>	A non-blank line?
/a/	Any ‘a’	Line with words?
/[ab]/	A choice	
/[a-z]/	l.c. char (range)	Common noun?
/[A-Z]/	u.c. char	Proper noun?
/[^?.!]/	Neg of set	Not S-final punc



RE	Description	Uses?
/a*/	Zero or more a's	Optional doubled modifiers
/a+/	One or more a's	Non-optional...
/a?/	Zero or one a's	Optional...
/cat|dog/	'cat' or 'dog'	Words modifying pets
/^cat\.$/	A line that contains only cat. ^anchors beginning, $ anchors end of line.	??
/\bun\B/	Beginnings of longer strings	Words prefixed by 'un'

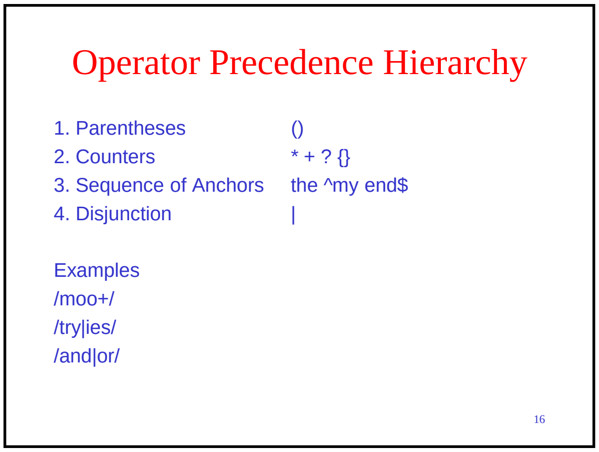


RE	E.G.
/pupp(y|ies)/	Morphological variants of 'puppy'
/ (.+)ier and \1ier /	happier and happier, fuzzier and fuzzier



Optionality and Repetition

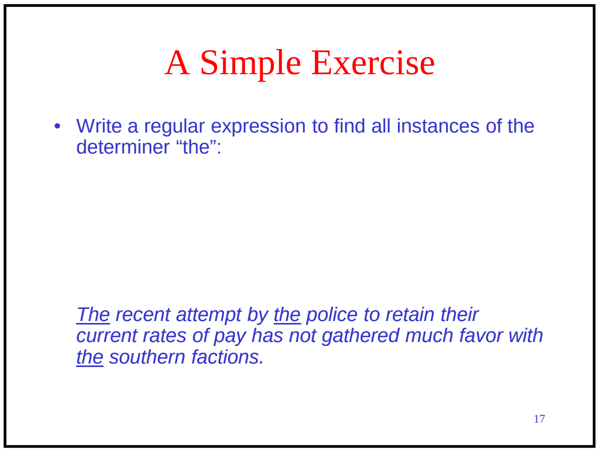
- /[Ww]oodchucks?/ matches woodchucks, Woodchucks, woodchuck, Woodchuck
- /colou?r/ matches color or colour
- /he{3}/ matches heee
- /(he){3}/ matches hehehe
- /(he){3,} matches a sequence of at least 3 he's



Operator Precedence Hierarchy

1. Parentheses	()
2. Counters	* + ? {}
3. Sequence of Anchors	the ^my end$
4. Disjunction	|

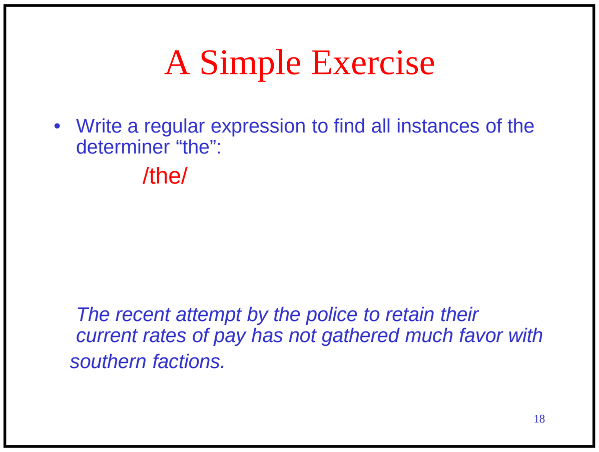
Examples
/moo+/
/try|ies/
/and|or/



A Simple Exercise

- Write a regular expression to find all instances of the determiner "the":

The recent attempt by the police to retain their current rates of pay has not gathered much favor with the southern factions.



A Simple Exercise

- Write a regular expression to find all instances of the determiner "the":

  /the/

The recent attempt by the police to retain their current rates of pay has not gathered much favor with southern factions.

A Simple Exercise

- Write a regular expression to find all instances of the determiner “the”:

  /the/

  /[tT]he/

The recent attempt by the police to retain their current rates of pay has not gathered much favor with the southern factions.

A Simple Exercise

- Write a regular expression to find all instances of the determiner “the”:

  /the/

  /[tT]he/

*The recent attempt by the police to retain **the**ir current rates of pay has not ga**the**red much favor with the sou**the**rn factions.*

A Simple Exercise

- Write a regular expression to find all instances of the determiner “the”:

  /the/

  /[tT]he/

  /\b[tT]he\b/

The recent attempt by the police to retain their current rates of pay has not gathered much favor with the southern factions.

A Simple Exercise

- Write a regular expression to find all instances of the determiner “the”:

  /the/

  /[tT]he/

  /\b[tT]he\b/

  /(^|[^a-zA-Z])[tT]he[^a-zA-Z]/

The recent attempt by the police to retain their current rates of pay has not gathered much favor with the southern factions.

The Two Kinds of Errors

- The process we just went through was based on fixing errors in the regular expression
  - Errors where some of the instances were missed (judged to not be instances when they should have been) – False negatives
  - Errors where the instances were included (when they should not have been) – False positives
- This is pretty much going to be the story of the rest of the course!

Substitutions (Transductions)

- Sed or ‘s’ operator in Perl
  - s/regexp1/pattern/
  - s/I am feeling (.++)/You are feeling \1?/
  - s/I gave (.+) to (.+)/Why would you give \2 \1?/

ELIZA: Substitutions Using Memory

User: Men are all alike.

ELIZA: IN WHAT WAY

s/.* all .*/IN WHAT WAY/

They're always bugging us about something or other.

ELIZA: CAN YOU THINK OF A SPECIFIC EXAMPLE

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

User: My boyfriend says I'm depressed.

ELIZA: I AM SORRY TO HEAR YOU ARE DEPRESSED

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

Using RE's: Examples

- Predictions from a news corpus:
  - Which candidate for Governor of California is mentioned most often in the news? Is going to win?
  - What stock should you buy?
  - Which White House advisers have the most power?
- Language use:
  - Which form of comparative is more frequent: 'oftener' or 'more often'?
  - Which pronouns are conjoined most often?
  - How often do sentences end with infinitival 'to'?
  - What words most often begin and end sentences?
  - What's the most common word in your email? Is it different from your neighbor?

- Personality profiling:
  - Are you more or less polite than the people you correspond with?
  - With labeled data, which words signal friendly messages vs. unfriendly ones?

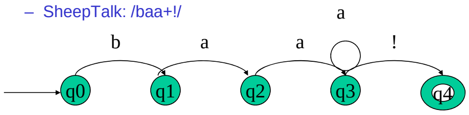
Finite State Automata

- Regular Expressions (REs) can be viewed as a way to describe machines called <u>Finite State Automata</u> (FSA, also known as automata, finite automata).

- FSAs and their close variants are a theoretical foundation of much of the field of NLP.

Finite State Automata

- FSAs recognize the regular languages represented by regular expressions
  - SheepTalk: /baa+!/

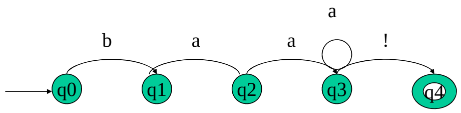


- **Directed graph with labeled nodes and arc transitions**

•**Five states:** q0 the start state, q4 the final state, 5 transitions

Formally

- FSA is a 5-tuple consisting of
  - Q: set of states {q0,q1,q2,q3,q4}
  - Σ: an alphabet of symbols {a,b,!}
  - q0: a start state
  - F: a set of final states in Q {q4}
  - δ(q,i): a transition function mapping Q x Σ to Q



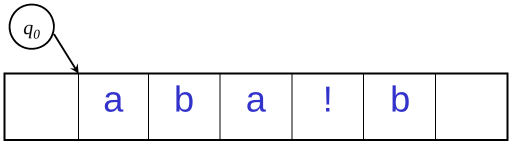
State Transition Table for SheepTalk

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

Recognition

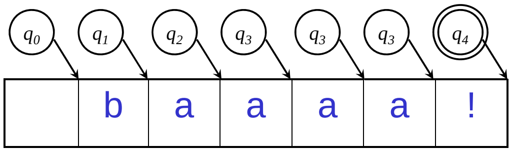
- Recognition (or acceptance) is the process of determining whether or not a given input should be accepted by a given machine.
- In terms of REs, it’s the process of determining whether or not a given input matches a particular regular expression.
- Traditionally, recognition is viewed as processing an input written on a tape consisting of cells containing elements from the alphabet.

- FSA recognizes (**accepts**) strings of a regular language
  - baa!
  - baaa!
  - baaa!
  - …
- Tape metaphor: a rejected input



D-Recognize

```
Function D-Recognize(tape, machine) returns accept or reject
Index ← Beginning of tape
Current-state ← Initial state of the 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 ← transition-table[current-state,tape[index]]
Index ← index + 1
end
```



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

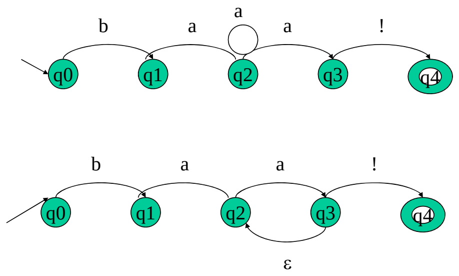
Key Points About D-Recognize

- Deterministic means that the code always knows what to do at each point in the process
- Recognition code is universal for all FSAs. To change to a new formal language:
  - change the alphabet
  - change the transition table
- Searching for a string using a RE involves compiling the RE into a table and passing the table to the interpreter

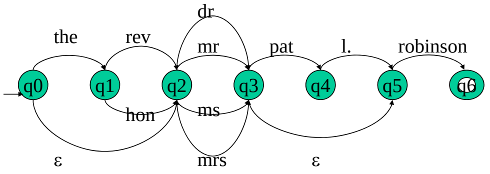
Determinism and Non-Determinism

- **Deterministic**: There is at most one transition that can be taken given a current state and input symbol.
- **Non-deterministic**: There is a choice of several transitions that can be taken given a current state and input symbol. (The machine doesn't specify how to make the choice.)

Non-Deterministic FSAs for SheepTalk



FSAs as Grammars for Natural Language



Can you use a regexpr to capture this too?

Problems of Non-Determinism

- 'Natural'….but at any choice point, we may follow the wrong arc
- Potential solutions:
  - Save backup states at each choice point
  - Look-ahead in the input before making choice
  - Pursue alternatives in parallel
  - Determinize our NFSAs (and then minimize)
- FSAs can be useful tools for recognizing – and generating – subsets of natural language
  - But they cannot represent all NL phenomena (Center Embedding: The mouse the cat ... chased died.)

Summing Up

- Regular expressions and FSAs can represent subsets of natural language as well as regular languages
  - Both representations may be impossible for humans to understand for any real subset of a language
  - But they are very easy to use for smaller subsets
- Next time: Read Ch 3
- For fun:
  - Think of ways you might characterize features of your email using only regular expressions