Regular Expressions and Automata in Natural Language Analysis
CS 4705

Some slides adapted from Hirschberg, Dorr/Monz, Jurafsky
Rule-based vs. Statistical Approaches

- Rule-based = linguistic

For what problems is rule-based better suited and when is statistics better
  - Identifying proper names
  - Distinguishing a biography from a dictionary entry
  - Answering questions

How far can a simple method take us?
  - How much is Google worth?
  - How much is Microsoft worth?

How much knowledge of language do our algorithms need to do useful NLP?
  - 80/20 Rule:
    - Claim: 80% of NLP can be done with simple methods
    - When should we worry about the other 20%?
Rule-based vs. Statistical Approaches

- Rule-based = linguistic

- For what problems is rule-based better suited and when is statistics better
  - Identifying proper names
  - Distinguishing a biography from a dictionary entry
  - Answering questions

- How far can a simple method take us?
  - How much is Google worth?
  - How much is Microsoft worth?
  - How much is IBM worth?

- How much knowledge of language do our algorithms need to do useful NLP?
  - 80/20 Rule:
    - Claim: 80% of NLP can be done with simple methods
    - When should we worry about the other 20%?
Rule-based vs. Statistical Approaches

- Rule-based = linguistic

- For what problems is rule-based better suited and when is statistics better
  - Identifying proper names
  - Distinguishing a biography from a dictionary entry
  - Answering questions

- How far can a simple method take us?
  - How much is Google worth?
  - How much is Microsoft worth?
  - How much is IBM worth?
  - How much is Walmart worth?

- How much knowledge of language do our algorithms need to do useful NLP?
  - 80/20 Rule:
    - Claim: 80% of NLP can be done with simple methods
    - When should we worry about the other 20%?
Rule–based vs. Statistical Approaches

- Rule–based = linguistic

- For what problems is rule–based better suited and when is statistics better
  - Identifying proper names
  - Distinguishing a biography from a dictionary entry
  - Answering questions

- How far can a simple method take us?
  - How much is Google worth?
  - How much is Microsoft worth?
  - How much is a Columbia University education worth?
  - How much is the Statue of Liberty worth?
  - How much is your life worth?

- How much knowledge of language do our algorithms need to do useful NLP?
  - 80/20 Rule:
    - Claim: 80% of NLP can be done with simple methods
    - When should we worry about the other 20%?
Today

- Review some simple representations of language and see how far they will take us
  - Regular Expressions
  - Finite State Automata
- Think about the limits of these simple approaches
  - When are simple methods good enough?
  - When do we need something more?
Regular Expression/Pattern Matching in NLP

- Simple but powerful tools for ‘shallow’ processing of a document or “corpus”
  - What word begins a sentence?
  - What words begin a question?
  - Identify all noun phrases
- Allow us to
  - Build simple interactive applications (e.g. Eliza)
  - Morphological analysis
  - Recognize Named Entities (NE): people names, company names
## Review

<table>
<thead>
<tr>
<th>RE</th>
<th>Matches</th>
<th>Uses</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>./</code></td>
<td>Any character</td>
<td>A non-blank line</td>
</tr>
<tr>
<td><code>\./, \?</code></td>
<td>A ‘.’, a ‘?’</td>
<td>A statement, a question</td>
</tr>
<tr>
<td><code>/[bckmsr]/</code></td>
<td>Any char in set</td>
<td>Rhyme: <code>/[bckmsr]ite/</code></td>
</tr>
<tr>
<td><code>/[a-z]/</code></td>
<td>Any l.c. letter</td>
<td>Rhyme: <code>/[a-z]ite/</code></td>
</tr>
<tr>
<td><code>/[A-Z]/</code></td>
<td>Capitalized letter</td>
<td>Possible NE</td>
</tr>
<tr>
<td><code>/[^A-Z]/</code></td>
<td>Lower case letter</td>
<td>Not an NE</td>
</tr>
<tr>
<td>RE</td>
<td>Description</td>
<td>Uses?</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>------------------</td>
</tr>
<tr>
<td>/a*/</td>
<td>Zero or more a’s</td>
<td>/(very[ ])*/</td>
</tr>
<tr>
<td>/a+/</td>
<td>One or more a’s</td>
<td>/(very[ ])+/</td>
</tr>
<tr>
<td>/a?/</td>
<td>Optional single a</td>
<td>/(very[ ])?/</td>
</tr>
<tr>
<td>/cat</td>
<td>dog/</td>
<td>‘cat’ or ‘dog’</td>
</tr>
<tr>
<td>/^[Nn]o$/</td>
<td>A line with only ‘No’ or ‘no’ in it</td>
<td></td>
</tr>
<tr>
<td>/\bun\B/</td>
<td>Prefixes</td>
<td>Words prefixed by ‘un’ (nb. union)</td>
</tr>
<tr>
<td>RE plus</td>
<td>E.G.</td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>/kitt(ies</td>
<td>en</td>
<td>ens)/</td>
</tr>
<tr>
<td>/ (.+ier) and \1 /</td>
<td>Patterns: happier and happier, fuzzier and fuzzier, classifier and classifier</td>
<td></td>
</tr>
</tbody>
</table>
Question Answering

- Eliza
Men are all alike.

IN WHAT WAY?

They're always bugging us about something or other.

CAN YOU THINK OF A SPECIFIC EXAMPLE?

Well, my boyfriend made me come here.

YOUR BOYFRIEND MADE YOU COME HERE

He says I'm depressed much of the time.

I AM SORRY TO HEAR YOU ARE DEPRESSED

It's true. I am unhappy

DO YOU THINK COMING HERE WILL HELP YOU NOT TO BE UNHAPPY

I need some help, that much seems certain.

WHAT WOULD IT MEAN TO YOU IF YOU GOT SOME HELP

Perhaps I could learn to get along with my mother.

TELL ME MORE ABOUT YOUR FAMILY

My mother takes care of me.

WHO ELSE IN YOUR FAMILY TAKES CARE OF YOU

My father.

YOUR FATHER

You are like my father in some ways.
Eliza-style regular expressions

Step 1: replace first person with second person references

```
s/\bI('m| am)\b /YOU ARE/g
s/\bmy\b /YOUR/g
S/\bmine\b /YOURS/g
```

Step 2: use additional regular expressions to generate replies

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

Step 3: use scores to rank possible transformations
How far does this allow you to go? How much of a question answering system?

Advantages?

Disadvantages?
Three Views

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

Regular Expressions

Regular Languages

Finite State Automata

Regular Grammars
Finite-state Automata (Machines)

The diagram illustrates a finite-state automaton with states $q_0, q_1, q_2, q_3, q_4$. The automaton transitions between states based on input symbols. The goal is to recognize strings that start with the pattern $/^\text{baa+}!$/.

The automaton starts at state $q_0$ and transitions to $q_1$ on input symbol $b$, to $q_2$ on input symbol $a$, to $q_3$ on input symbol $a$ again, and finally to $q_4$ on the input symbol $!$. The final state is denoted by $q_4$, indicating the automaton's acceptance of the pattern.
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 in Q
- **F**: a set of final states in Q \{q4\}
- **δ(q, i)**: a transition function mapping Q x Σ to Q
Yet Another View

- State-transition table

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th>0</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>b</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>a</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
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
Traditionally, (Turing’s idea) this process is depicted with a tape.
Recognition

- Start in the start state
- Examine the current input
- Consult the table
- Go to a new state and update the tape pointer.
- Until you run out of tape.
Input Tape

Slide from Dorr/Monz
Input Tape

Slide from Dorr/Monz
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.
Non–Deterministic FSAs for SheepTalk
Problems of Non-Determinism

- 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 (e.g. center embedding: The mouse the cat chased died.)
FSAs as Grammars for Natural Language: Names
Recognizing Person Names

- If we want to extract all the proper names in the news, will this work?
  - What will it miss?
  - Will it accept something that is not a proper name?
  - How would you change it to accept all proper names without false positives?
  - Precision vs. recall....
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
- **Affixes**: Bits and pieces that adhere to stems to change their meanings and grammatical functions
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*
  - Goose, *geese*
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
Why care about morphology?

- Spelling correction: reference
- Morphology in machine translation
  - Spanish words **quiero** and **quieres** are both related to **querer** ‘want’
- Hyphenation algorithms: refer–ence
- Part–of–speech analysis: google, googler
- Text–to–speech: grapheme–to–phoneme conversion
  - **hothouse** (/T/ or /D/)
- Allows us to guess at meaning
  - ‘Twas brillig and the slithy toves…
  - Muggles moogled migwiches
We’d like to use the machinery provided by FSAs to capture facts about morphology
  - Ie. Accept strings that are in the language
  - And reject strings that are not
  - And do it in a way that doesn’t require us to in effect list all the words in the language
What do we need to build a morphological parser?

- **Lexicon**: list of stems and affixes (w/ corresponding part of speech (p.o.s.))
- **Morphotactics** of the language: model of how and which morphemes can be affixed to a stem
- **Orthographic rules**: spelling modifications that may occur when affixation occurs
  - \( \text{in} \rightarrow \text{il} \) in context of \( \text{l} \) (\text{in–} + \text{legal})
- Most morphological phenomena can be described with **regular expressions** – so finite state techniques often used to represent morphological processes
Regular singular nouns are ok
Regular plural nouns have an –s on the end
Irregulars are ok as is
Simple Rules

$q_0$  reg-noun  $q_1$  plural -s  $q_2$

irreg-pl-noun

irreg-sg-noun
Now Add in the Words
Derivational morphology: adjective fragment

- **Adj-root\(_1\):** clear, happi, real
- **Adj-root\(_2\):** big, red (*bigly*)
Parsing/Generation vs. Recognition

- We can now run strings through these machines to recognize strings in the language
  - **Accept** words that are ok
  - **Reject** words that are not

- But recognition is usually not quite what we need
  - Often if we find some string in the language we might like 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”
The kind of parsing we’re talking about is normally called **morphological analysis**. It can either be:

- An important stand-alone component of an application (spelling correction, information retrieval)
- Or simply a link in a chain of processing
Generativity

- Nothing really privileged about the directions.
- We can write from one and read from the other or vice-versa.
- One way is generation, the other way is analysis
Kimmo Koskenniemi’s two-level morphology idea: word is a relationship between lexical level (its morphemes) and surface level (its orthography)

**Lexical**

```
cat +N +Pl
```

**Surface**

```
cats
```
Transitions

- **c:c** means read a c on one tape and write a c on the other
- **+N:ɛ** means read a +N symbol on one tape and write nothing on the other
- **+PL:s** means read +PL and write an s
The Gory Details

- Of course, it's not as easy as
  - “cat +N +PL” <-> “cats”
- As we saw earlier there are geese, mice and oxen
- But there are also a whole host of spelling/pronunciation changes that go along with inflectional changes
  - Cats vs Dogs
  - Fox and Foxes
To deal with this we can simply add more tapes and use the output of one tape machine as the input to the next.

So to handle irregular spelling changes we’ll add intermediate tapes with intermediate symbols.
Multi-Level Tape Machines

- **Lexical**: 
  - Intermediate: 
    - Surface: 

  We use one machine to transduce between the lexical and the intermediate level, and another to handle the spelling changes to the surface tape.
Lexical to Intermediate Level
The add an “e” rule as in \( \text{fox}^s\# \leftrightarrow \text{foxes}\# \)
Foxes

Lexical: f o x +N +Pl

Intermediate: f o x ^ s #

Surface: f o x e s

$T_{\text{lex}}$: 0 1 2 5 6 7

$T_{\text{e-insert}}$: 0 0 0 1 2 3 4 0
Regular expressions and FSAs can represent subsets of natural language as well as regular languages
- Both representations may be difficult for humans to understand for any real subset of a language
- Can be hard to scale up: e.g., when many choices at any point (e.g. surnames)
- But quick, powerful and easy to use for small problems

Next class:
- Read Ch 4