CSCI 3907/6907

Fall 2019

Lecture 2 – Part I

Regular Expressions & Automata

Notes

- Syllabus: *Check Often!*

https://sites.google.com/view/csci-39076907-fall2019/syllabus

 - four main assignments 40%, Midterm 30%, Final Project 30%

- Register for Piazza

 https://piazza.com/gwu/fall2019/csci39076907nlp/home

- Start forming a group for final project (max 4 students) and choosing your topic

- The first assignment will be released this week.
  - *All assignment related questions must be directed to the TA.*
  - *You have 7 free grace days for the whole semester.*

Review

- *What is NLP?*
  - *Building programs that can recognize, analyze , and generate text and speech*
  - *Processing unstructured data*

- NLP Applications: Machine Translation, Sentiment Analysis, Part of Speech Tagging … etc

- Ambiguity results from the existence of multiple possibilities for each level

- NLP Approaches: Rule based and statistical based approaches

Outline: Basic Text Processing

- Part 1: Formal languages: regular Expressions and automata
- Part 2: Words and Transducers

Formal Languages

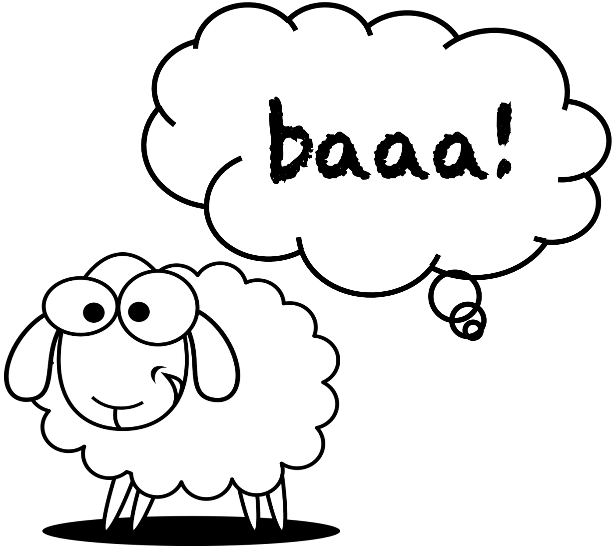
- A **formal language** is a set of **strings**, each string composed of symbols from a finite symbol-set called an **alphabet Σ**.
- A model which can both **generate** and **recognize** all and only the strings of a formal language acts as a definition of the formal language.

Formal Languages – Cont.

- A formal language may bear no resemblance at all to a real language (natural language)

- but can model a subset of expressions:
  - Parts of the phonology, morphology, or syntax.
  - Date and time expressions.

- Example: Sheeptalk
  - Any string from the following (infinite) set:
    Baa!
    Baaa!
    Baaaa!
    Baaaaa!
    …..



Regular Languages

- A regular language $\rightarrow$ a formal language that can be fully specified using one of the following:
  - Regular expression
  - Finite state automaton
  - Regular grammar

Regular Expressions

- A formal language for specifying text search strings
- Used in
  - Unix tools like grep and sed
  - Programming languages: Perl, python, Java, etc.

Regular Expressions

- Requires:
  - a pattern (e.g. all words that start with *ca*)
  - a corpus of texts to search through

- Returns all text that matches the pattern or only the first match

Regular Expressions: Disjunctions

- Letters inside square brackets []

Pattern	Matches
[wW]oodchuck	Woodchuck, woodchuck
[1234567890]	Any digit

- Ranges [A-Z]

Pattern	Matches	Examples
[A-Z]	An upper case letter	Drenched Blossoms
[a-z]	A lower case letter	my beans were impatient
[0-9]	A single digit	Chapter 1: Down the Rabbit Hole

Negation in Disjunction

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

Pattern	Matches	Examples
[^A-Z]	Not an upper case letter	Oyfn pripetchik
[^Ss]	Neither 'S' nor 's'	I have no exquisite reason"
[^e^]	Neither e nor ^	Look here
a^b	The pattern a carat b	Look up a^b now

More Disjunction

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

<table>
<tr><th>Pattern</th><th>Matches</th></tr>
<tr><td>groundhog|woodchuck</td><td>groundhog, woodchuck</td></tr>
<tr><td>yours|mine</td><td>yours , mine</td></tr>
<tr><td>a|b|c</td><td>a, b, c</td></tr>
<tr><td>[gG]roundhog|[Ww]oodchuck</td><td>groundhog, Groundhog, woodchuck, Woodchuck</td></tr>
</table>

?   *   +   .

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

Stephen C Kleene

Anchors ^ $

Pattern	Matches
^[A-Z]	Palo Alto
^[^A-Za-z]	1 “Hello”
\.$	The end.
.$	The end? The end!

Other Operators & Special Characters

Pattern	matches
\d	[0-9], any digit
\D	Any non digit
\w	Any alphanumeric or space
\W	[^\w] any non-alphanumeric
\s	whitespace
\S	non whitespace
*, \?, \.	*, ?, . (Characters that need to be escaped)
\n	end of line
\t	tab
{n}	n occurrences of the previous char or expression
{n,m}	n to m occurrences of the previous char or expression
{n, }	At least n occurrences of the previous char or expression

Substitution And Memory

- Find a pattern in text, then replace it with another pattern.

  `s/pattern/replacement/`

- Capture groups: store a matched pattern in memory, then reuse it in substitution or later matches:
  - The resulting match is stored in a numbered register (e.g \1 and \2)
    - Using memory in search:

      ```
      /the (.*)er they were, the \1er they will be/
      /the (.*)er they (.*), the \1er we \2/
      ```

    - Using memory in substitution: `s/([0-9]+)/<\1>/`

ELIZA

- Use regular expressions and capture groups to generate responses:

User: You are like my father in some ways.
ELIZA: WHAT RESEMBLANCE DO YOU SEE
User: You are not very aggressive but I think you don't want me to notice that.
ELIZA: WHAT MAKES YOU THINK I AM NOT VERY AGGRESSIVE
User: You don't argue with me.
ELIZA: WHY DO YOU THINK I DON'T ARGUE WITH YOU
User: You are afraid of me.
ELIZA: DOES IT PLEASE YOU TO BELIEVE I AM AFRAID OF YOU

Weizenbaum (1966)

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

Simple ELIZA Implementation

```
#Upper case all characters
tr/a-z/A-Z/;
#Remove all non-words except for spaces
s/[^\w\s]//;
#Switch first person to second person
s/\bMY\b/YOUR/g;
s/\b(I|ME)\b/YOU/g;
s/\bAM\b/ARE/g;
#Responses
s/.*\b(HI|HELLO|HOWDY)\b.*/$1 THERE. HOW ARE YOU DOING?/;
s/.*\bYOU ARE (DEPRESSED|SAD|UNHAPPY)\b.*/I AM SORRY TO HEAR YOU ARE $1./;
s/.*\bALL\b.*/IN WHAT WAY?/;
s/.*\bALWAYS\b.*/CAN YOU THINK OF A SPECIFIC EXAMPLE?/;
s/.*\bNOT\b.*/WHY NOT?/;
s/.*\bYES\b.*/I SEE/;
#Add prompt
s/^/>> /;
```

Example

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

the

e.g. misses capitalized examples

[tT]he

e.g. incorrectly returns *other* or *theology*

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

Errors

- Two kinds of errors:
  - False positive (Type I)
    - Matching strings that we should not have matched (there, then, other)
  - False negative (Type II)
    - Not matching things that we should have matched (The)

Errors – Cont.

- We always deal with these types of errors in NLP
- Reduce the error rate for an application involve:
  - Increasing accuracy or precision *(minimizing false positives)*
  - Increasing coverage or recall *(minimizing false negative)*

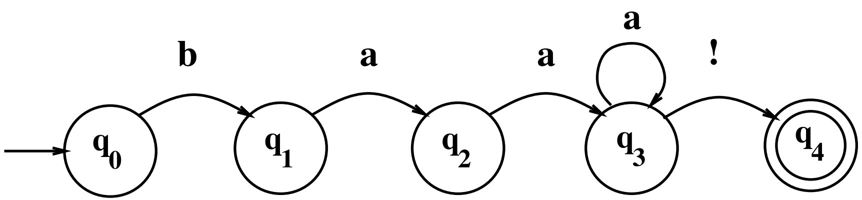
Summary

- Regular expressions play a surprisingly large role
  - Sophisticated sequences of regular expressions are often the first model for any text processing text
  - Early version of chatbots (ELIZA), text normalization, tokenization, stemming, sentence segmentation, string similarities

- 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

Finite State Automata (FSA)

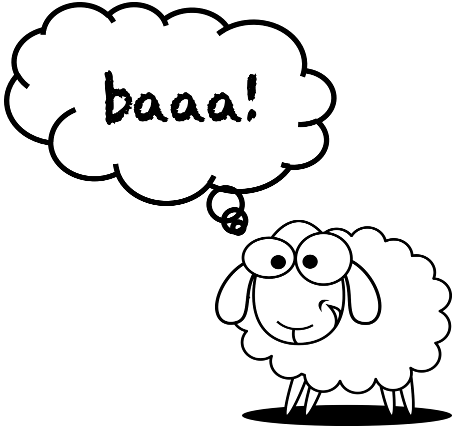
- Abstract model of computation
- Consists of N **states** and a **transition** matrix.
- A **start** state, and a set of final "**accept**" states
- Can be deterministic or non deterministic

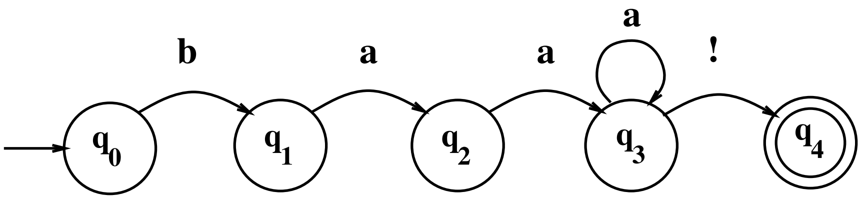


	Input		
State	b	a	!
0	1	∅	∅
1	∅	2	∅
2	∅	3	∅
3	∅	3	4
4:	∅	∅	∅

Formally

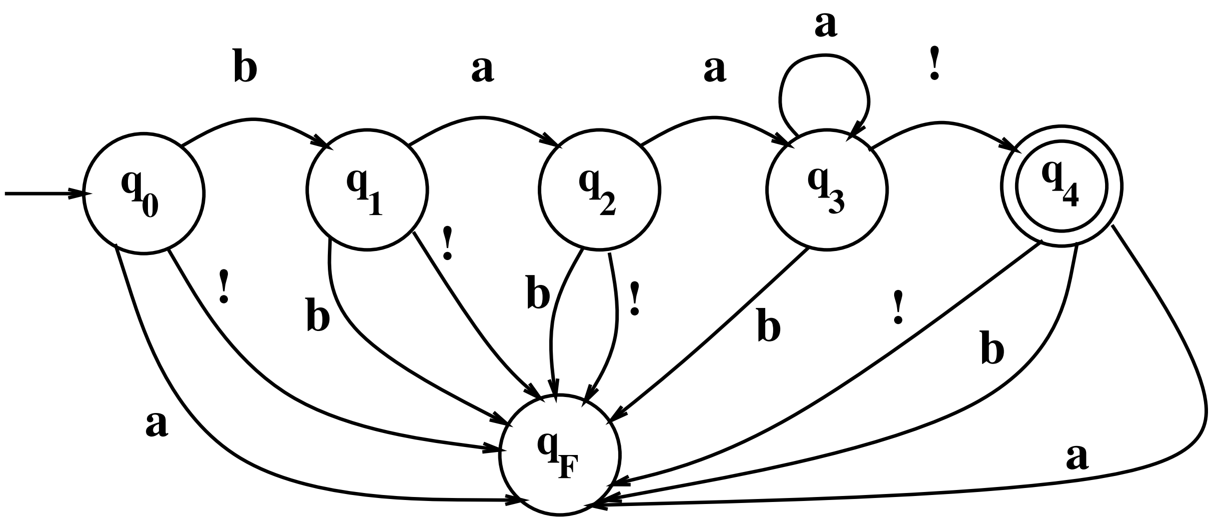
- A regular language is a 5-tuple consisting of
  - Q: set of states {q0,q1,q2,q3,q4}
  - Σ: an alphabet of symbols {a,b,!}
  - q0: a start state in Q
  - F: a set of final states in Q {q4}
  - δ(q,i): a transition function mapping Q x Σ to Q





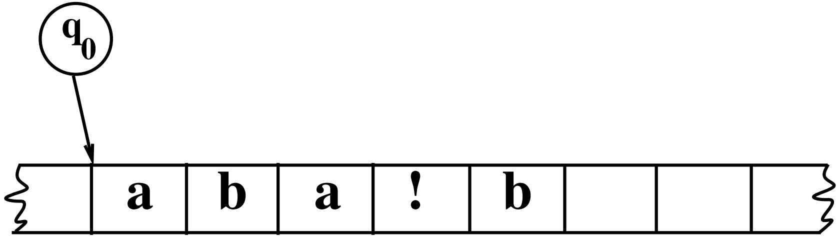
Finite State Automata (FSA)

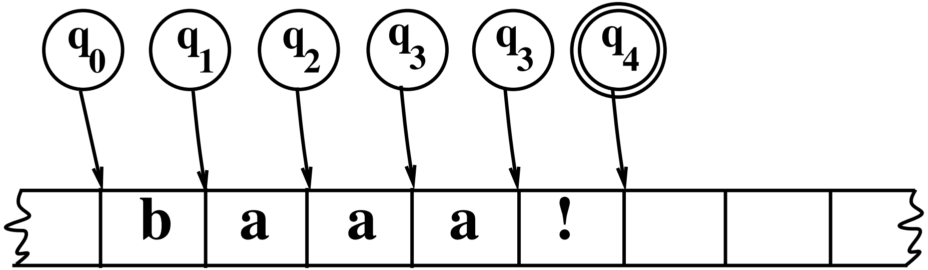
- Adding a **fail** state



The Tape Metaphor

- Imagine the input being written on a long tape broken into cells:





Deterministic FSAs

- Deterministic: 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.

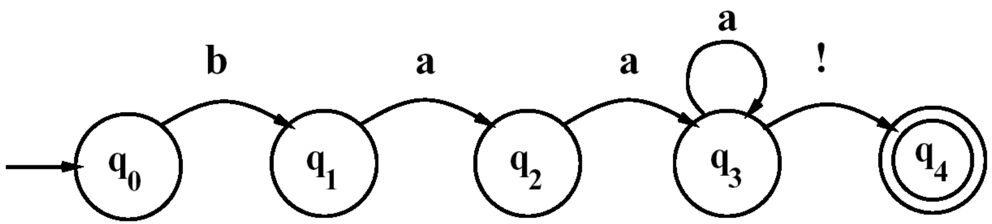
- FSAs can be useful tools for recognizing – and generating – subsets of natural language

Deterministic Recognition Algorithm

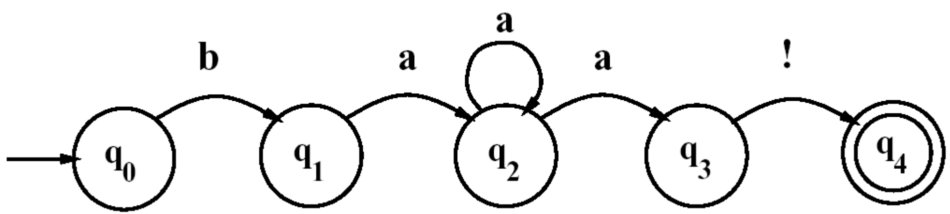
```
function D-RECOGNIZE(tape, machine) returns accept or reject

  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
```

Non-Deterministic FSAs

- Deterministic sheep language FSA



- Non-deterministic sheep language FSA



Dealing with 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

Equivalence

- For every NFSA there exists an equivalent DFSA
  - (i.e. that accepts exactly the same set of strings).
- The resulting DFSA, may have many more states than the original NFSA
  - (up to 2^N states for a NFSA with N states).