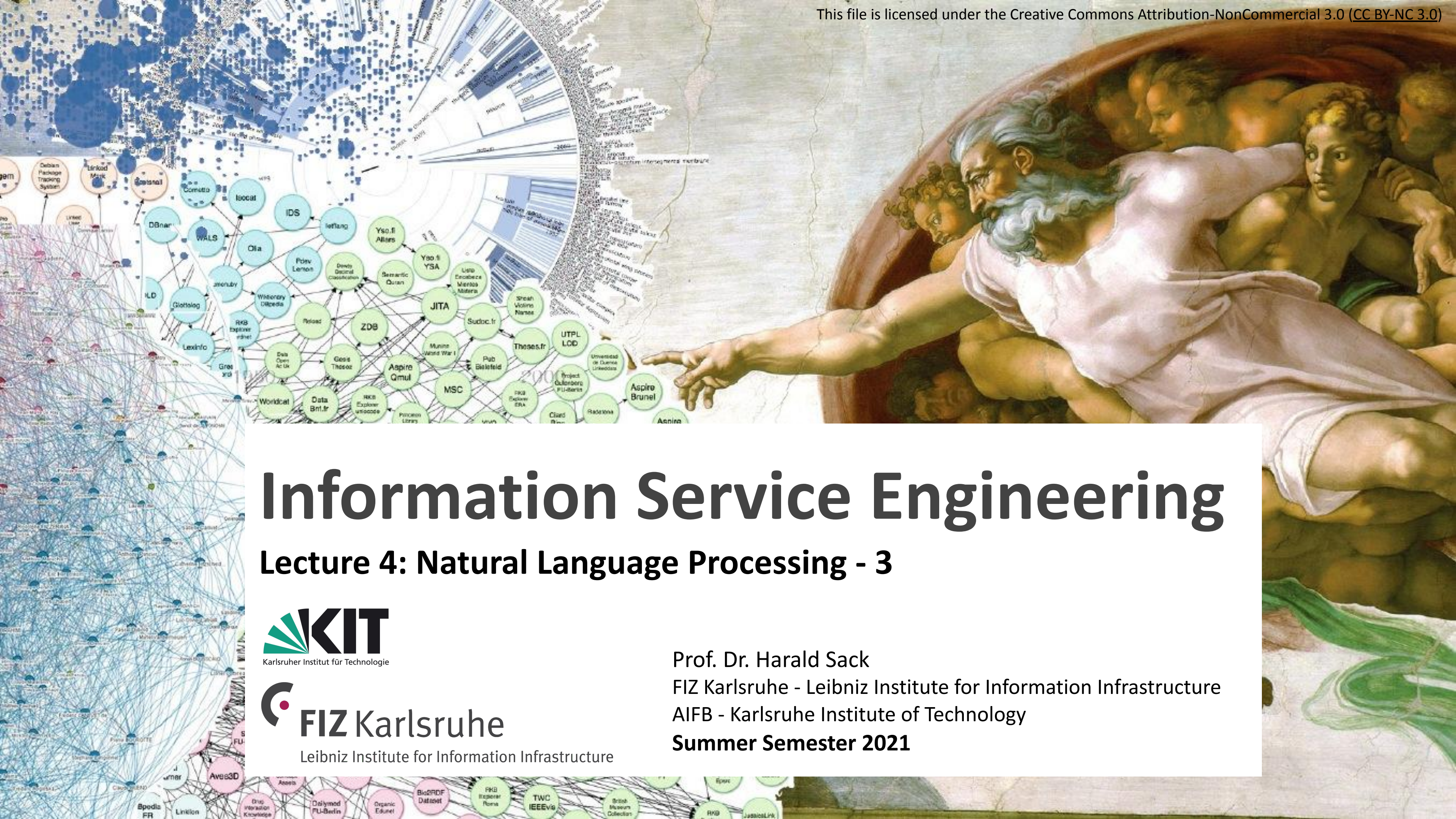


Information Service Engineering

Lecture 4: Natural Language Processing - 3

KIT
Karlsruher Institut für Technologie

FIZ Karlsruhe
Leibniz Institute for Information Infrastructure

Prof. Dr. Harald Sack
FIZ Karlsruhe - Leibniz Institute for Information Infrastructure
AIFB - Karlsruhe Institute of Technology

Summer Semester 2021

Information Service Engineering

Last Lecture: Natural Language Processing (2)

2.0 What is Natural Language Processing?

2.1 NLP and Basic Linguistic Knowledge

2.2 Morphology

2.3 NLP Applications

2.4 NLP Techniques

2.5 NLP Challenges

2.6 Evaluation, Precision and Recall

2.7 Regular Expressions

2.8 Finite State Automata

2.9 Tokenization

2.10 Language Model and N-Grams

2.11 Part-of-Speech Tagging

- Phonetic, lexical, syntactic, semantic Ambiguity and Disambiguation
- Evaluation, Ground Truth, Recall, Precision, F-Measure
- Regular Expressions

Information Service Engineering

Lecture 4: Natural Language Processing (3)

2.0 What is Natural Language Processing?

2.1 Basic Linguistics

2.2 Morphology

2.3 NLP Applications

2.4 NLP Techniques

2.5 NLP Challenges

2.6 Evaluation, Precision and Recall

2.7 Regular Expressions

2.8 Finite State Automata

2.9 Tokenization

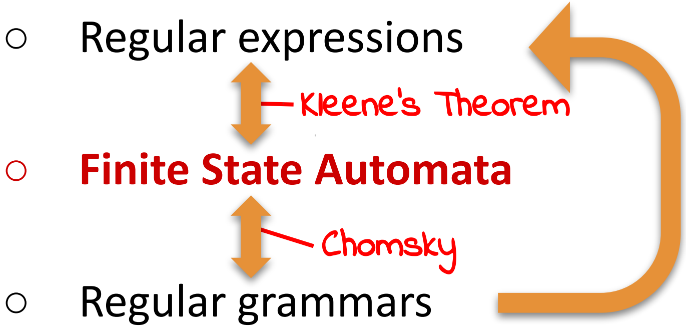
2.10 Language Model and N-Grams

2.11 Part-of-Speech Tagging

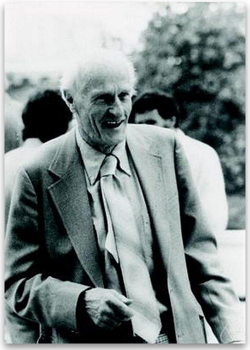
Regular Expressions and Finite State Automata (FSA)

- **Regular Expressions**
  - are one way to characterize a **Regular Language**
    as e.g. /baa+!/

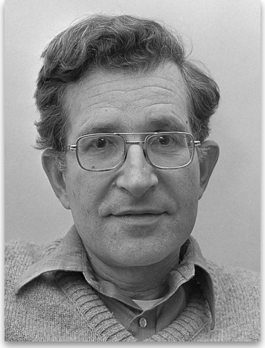
- In general **Regular Languages** can be characterized via



  - Regular expressions
  - Finite State Automata
  - Regular grammars



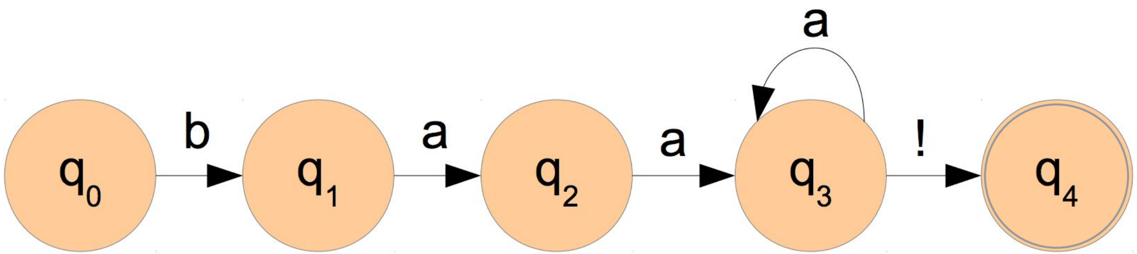
Stephen Cole Kleene
(1909 - 1994)



Noam Chomsky
(*1928)

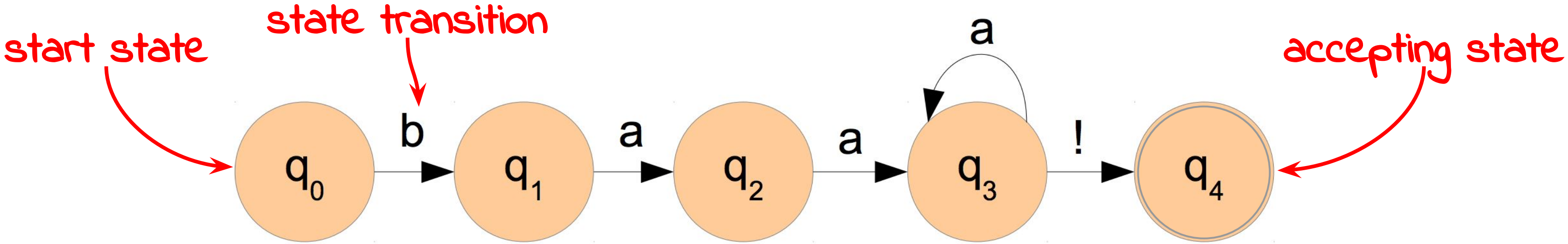
Finite State Automata (FSA)

- A **Finite State Automaton** is an abstract model of a computer which
  - **reads an input string**,
  - and **changes its internal state**
    depending on the current input symbol.
- An FSA can either **accept** or **reject** the input string.
- Every automaton defines a **language**, i.e. the set of strings it accepts.



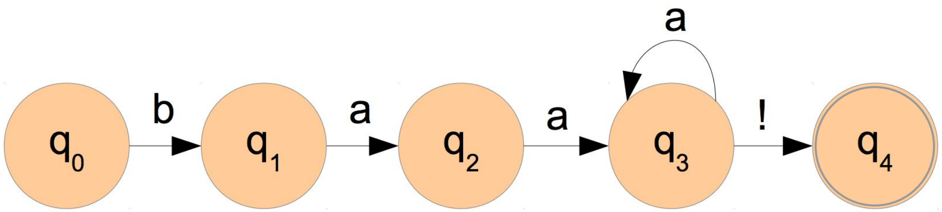
Finite State Automata (FSA)

- **Finite State Automata** are composed of
  - Vertices (nodes)
  - Arcs (links)



- What words (strings) can be recognized by this FSA example?
  - baa! / baaa! / baaaa! / baaaaa! / ..
- Matching RE?
  - **/baa+!/**

Finite State Automata (FSA)



- **Strings accepted:**
    - baa!
    - baaaaa!
- **Strings not accepted:**
    - abc
    - babb
    - !bcd

State Transition Table

	Input		
State	**a**	**b**	**!**
q_0	∅	q_1	∅
q_1	q_2	∅	∅
q_2	q_3	∅	∅
q_3	q_3	∅	q_4
q_4	∅	∅	∅

Formalisation of a Finite State Automaton (FSA)

- **FSA** **$A = (Q, \Sigma, q_0, F, \delta(q,i))$**, with
- **Q**: finite set $\{q_0, q_1, q_2, \ldots, q_{N-1}\}$ of N states
- **Σ**: finite input alphabet of symbols
- **q_0**: the designated start state
- **F**: the set of final states, $F \subseteq Q$
- **$\delta(q,i)$**: the transition function

  $\delta: Q \times \Sigma \rightarrow Q$,

  $\delta(q,i) = q'$ for $q, q' \in Q$, $i \in \Sigma$

You denote the transition function δ in terms of a state transition table

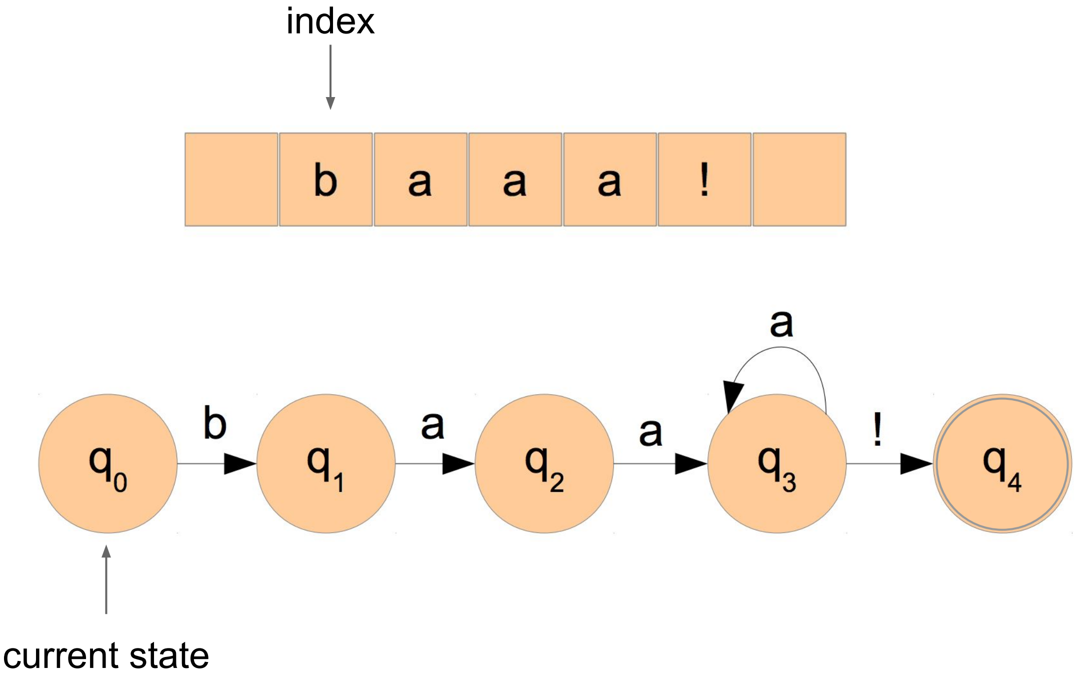
Formalisation of “Sheeptalk”

- $Q = \{q_0, q_1, q_2, q_3, q_4\}$
- $\Sigma = \{a, b, !\}$
- q_0: the start state
- $F = \{q_4\}$
- $\boldsymbol{\delta}(q,i)$: defined by state transition table

State Transition Table

	Input		
State	**a**	**b**	**!**
q_0	∅	q_1	∅
q_1	q_2	∅	∅
q_2	q_3	∅	∅
q_3	q_3	∅	q_4
q_4	∅	∅	∅

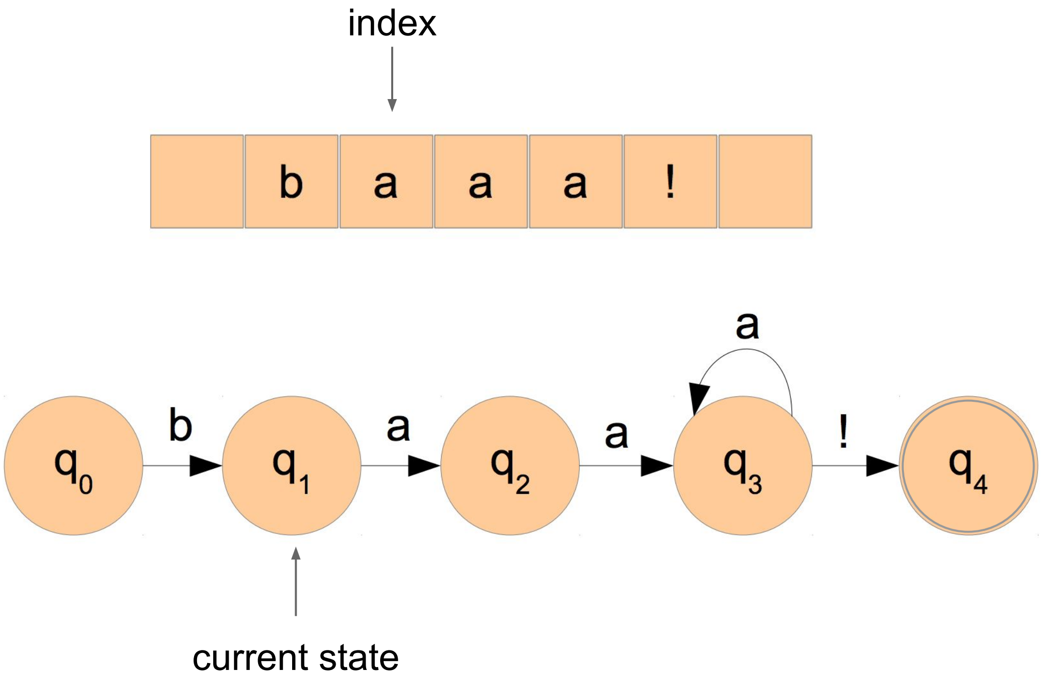
D-RECOGNIZE Algorithm (step1)



State Transition Table

	Input		
State	**a**	**b**	**!**
q_0	∅	q_1	∅
q_1	q_2	∅	∅
q_2	q_3	∅	∅
q_3	q_3	∅	q_4
q_4	∅	∅	∅

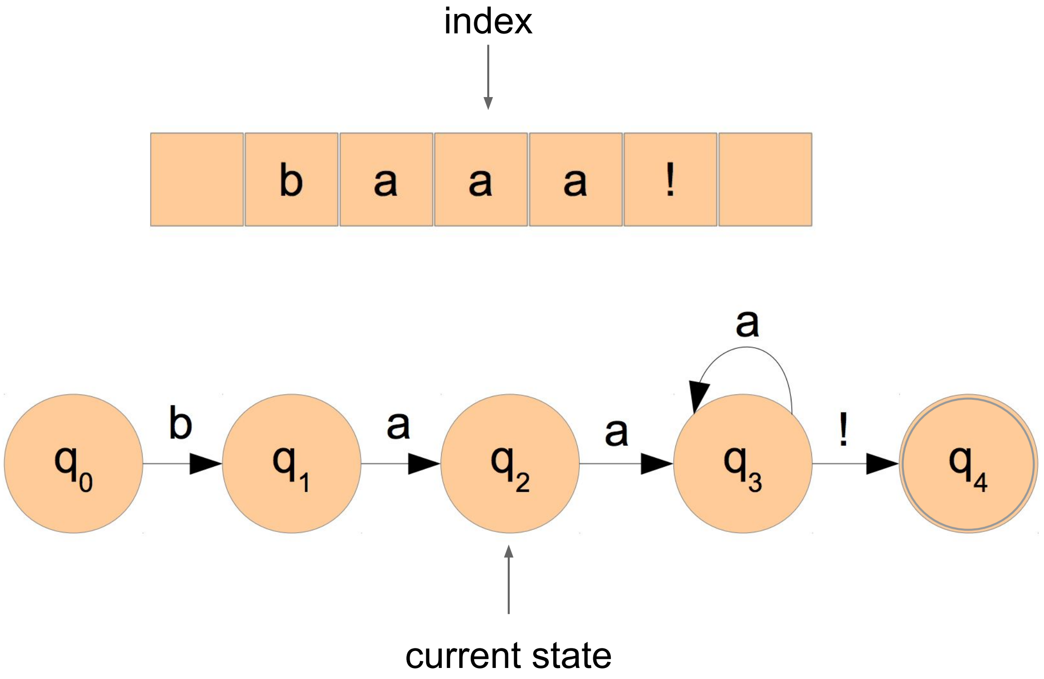
D-RECOGNIZE Algorithm (step2)



State Transition Table

	Input		
State	**a**	**b**	**!**
q_0	∅	q_1	∅
q_1	q_2	∅	∅
q_2	q_3	∅	∅
q_3	q_3	∅	q_4
q_4	∅	∅	∅

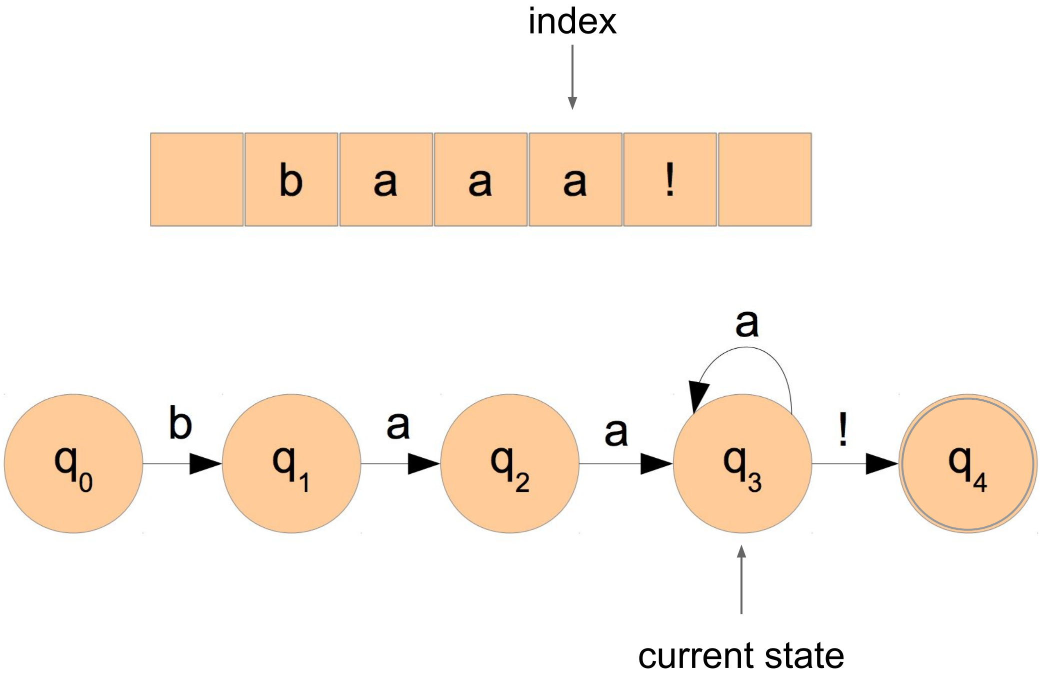
D-RECOGNIZE Algorithm (step3)



State Transition Table

	Input		
State	**a**	**b**	**!**
q_0	∅	q_1	∅
q_1	q_2	∅	∅
q_2	q_3	∅	∅
q_3	q_3	∅	q_4
q_4	∅	∅	∅

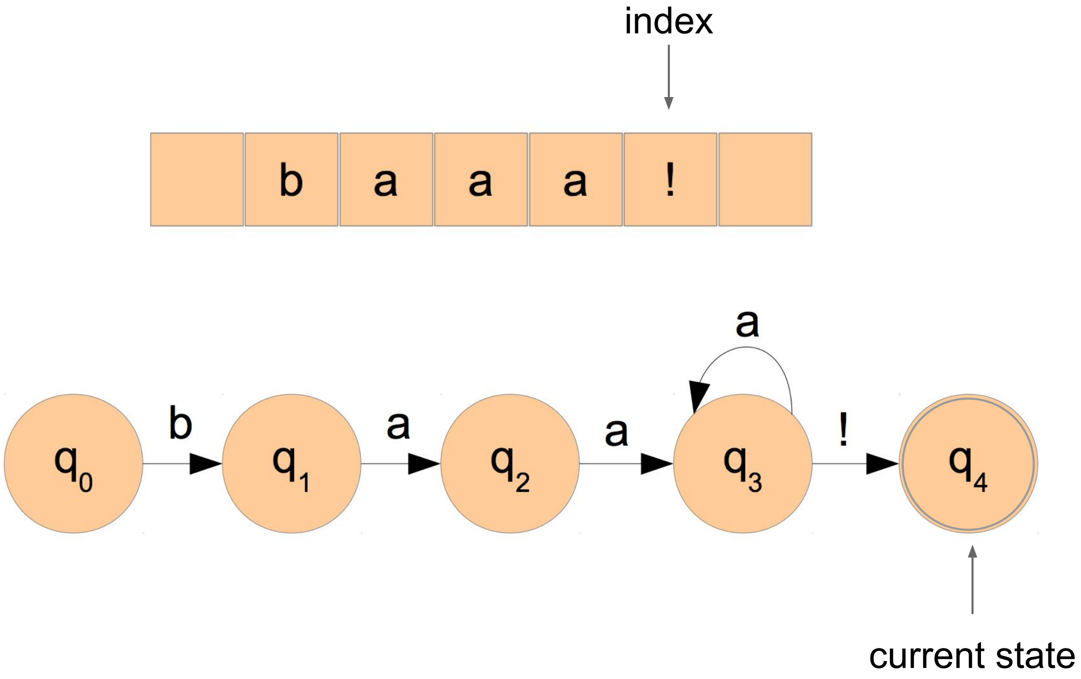
D-RECOGNIZE Algorithm (step4)



State Transition Table

	Input		
State	**a**	**b**	**!**
q_0	∅	q_1	∅
q_1	q_2	∅	∅
q_2	q_3	∅	∅
q_3	q_3	∅	q_4
q_4	∅	∅	∅

D-RECOGNIZE Algorithm (step4)



State Transition Table

	Input		
State	**a**	**b**	**!**
q_0	∅	q_1	∅
q_1	q_2	∅	∅
q_2	q_3	∅	∅
q_3	q_3	∅	q_4
q_4	∅	∅	∅

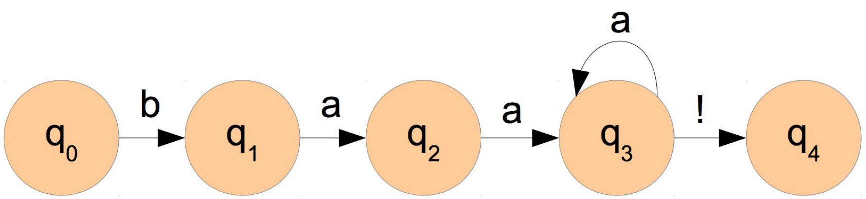
- q_4 is the final state, string is accepted.

Finite State Automata (FSA) for “Sheeptalk”

- **FSA** **A = (Q, Σ, q_0, F, δ(q,i))**, with
  - **Q** = $\{q_0, q_1, q_2, q_3, q_4\}$
  - **Σ** = {a,b,!}
  - **q_0**: the start state
  - **F** = $\{q_4\}$
  - **δ(q,i)**: defined by state transition table →

State Transition Table

	Input		
State	**a**	**b**	**!**
q_0	∅	q_1	∅
q_1	q_2	∅	∅
q_2	q_3	∅	∅
q_3	q_3	∅	q_4
q_4	∅	∅	∅



Formal Language

- A **model** that can both **generate** and **recognize** all and only those strings given by its definition.
- An automaton can describe an infinite set with a closed form.

- **“Sheeptalk” model “m”:**
  - $\sum = \{b,a,!\}$ - Alphabet
  - L(m) = formal language characterized by „m“
  - L(m) = {baa!,baaa!,baaaa!,....}
  - Usually it holds that $L \subset \sum^*$

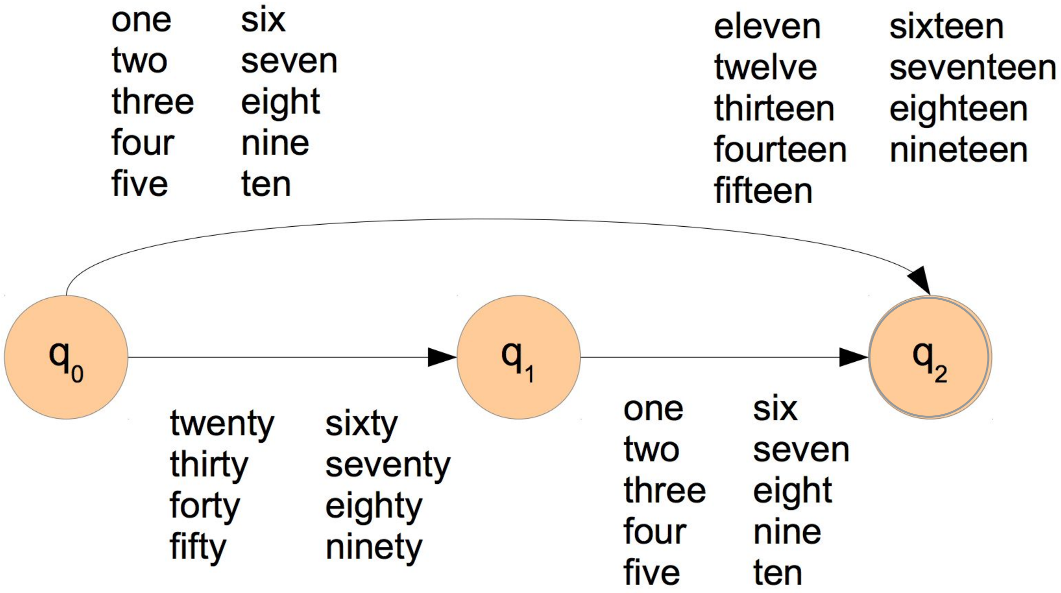
The **Kleene closure** $\sum^*$ ('sigma star') is the (infinite) set of all strings that can be formed from $\sum$.

Formal Language vs. Natural Language

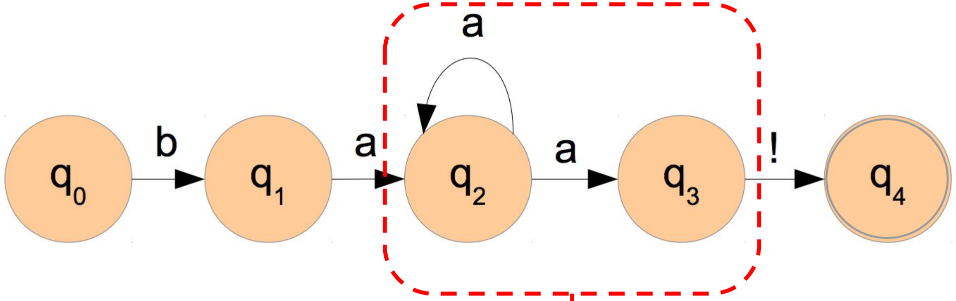
- A **formal language** is not a **natural language**.
- But we can use formal languages to **model parts of natural languages**,
  - such as e.g. **phonology, morphology or syntax**.

Another (simple) FSA Example

- Modelling amounts of money (e.g. 0-99 cent):



Non Deterministic FSAs



State Transition Table

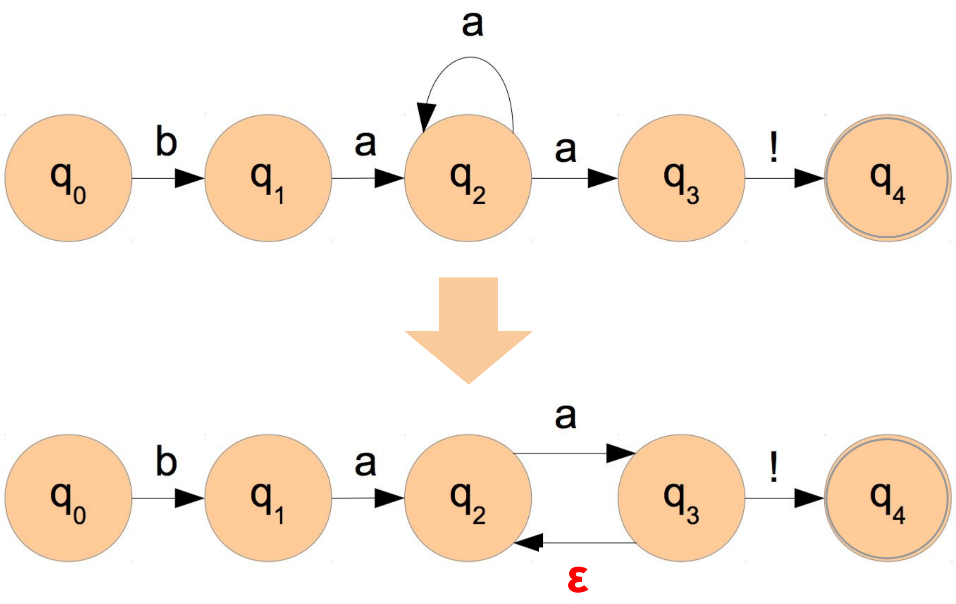
	Input		
State	**a**	**b**	**!**
$\mathbf{q_0}$	∅	q_1	∅
$\mathbf{q_1}$	q_2	∅	∅
$\mathbf{q_2}$	q_2,q_3	∅	∅
$\mathbf{q_3}$	∅	∅	q_4
$\mathbf{q_4}$	∅	∅	∅

- $\boldsymbol{\delta}$**(q,i)**: the **transition function for NFAs**

  $\boldsymbol{\delta}: Q \times \Sigma \rightarrow 2^Q$

  $\boldsymbol{\delta}(q,i) = Q'$ for $q \in Q$, $Q' \subseteq Q$, $i \in \Sigma$

Non Deterministic FSAs with ε-Transitions



- $\boldsymbol{\delta}(q,i)$: the **transition function for NFAs**

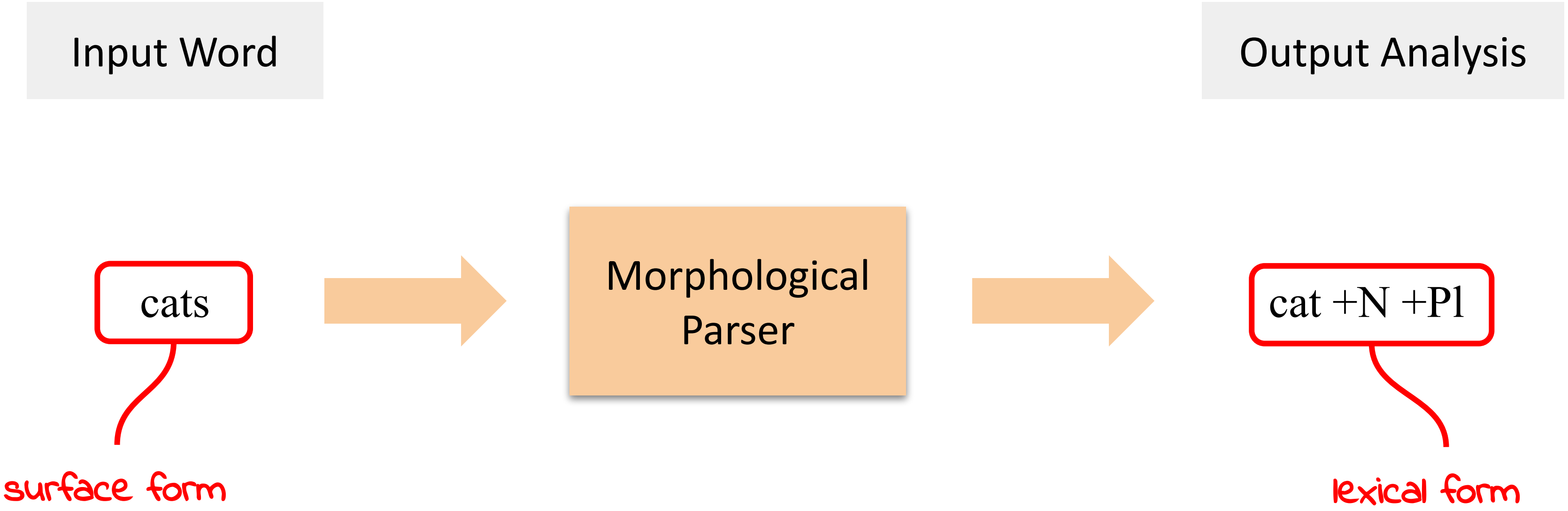
  $\boldsymbol{\delta}: Q \times \Sigma_\varepsilon \rightarrow 2^Q$, $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}, \Sigma \cap \{\varepsilon\} = \varnothing$

  $\boldsymbol{\delta}(q,i) = Q'$ for $q \in Q$, $Q' \subseteq Q$, $i \in \Sigma_\varepsilon$

State Transition Table

	Input			
State	**a**	**b**	**!**	**ε**
q_0	∅	q_1	∅	∅
q_1	q_2	∅	∅	∅
q_2	q_3	∅	∅	∅
q_3	∅	∅	q_4	q_2
q_4	∅	∅	∅	∅

Morphological Parsing



Finite State Morphological Parser

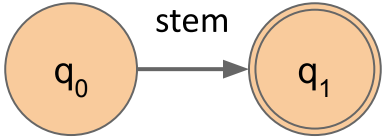
- To construct a morphological par

= part-of-speech,
as e.g. noun, verb, adjective, etc.

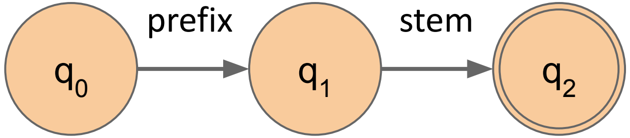
1. **Lexicon**: list of **stems** + **type** and **affixes** .
2. **Morphotactic rules**: model of **morpheme ordering**, e.g. *plural affix -s follows noun.*
3. **Orthographic rules**: spelling rules for morpheme combinations, e.g. ***y* → *ie*** *that changes* ***city* + *-s*** *into* ***cities***.

FSA for Morphology

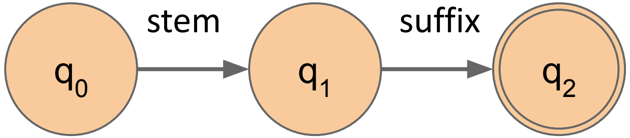
- grace:



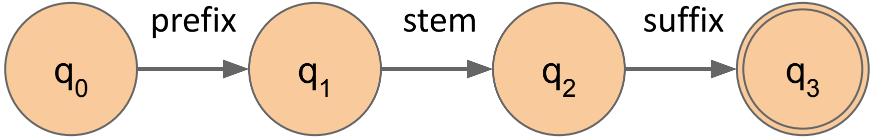
- dis-grace:



- grace-ful:

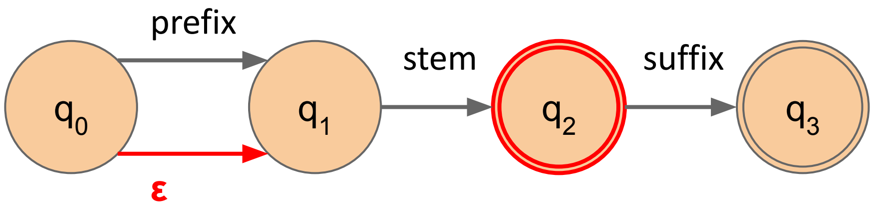


- dis-grace-ful:

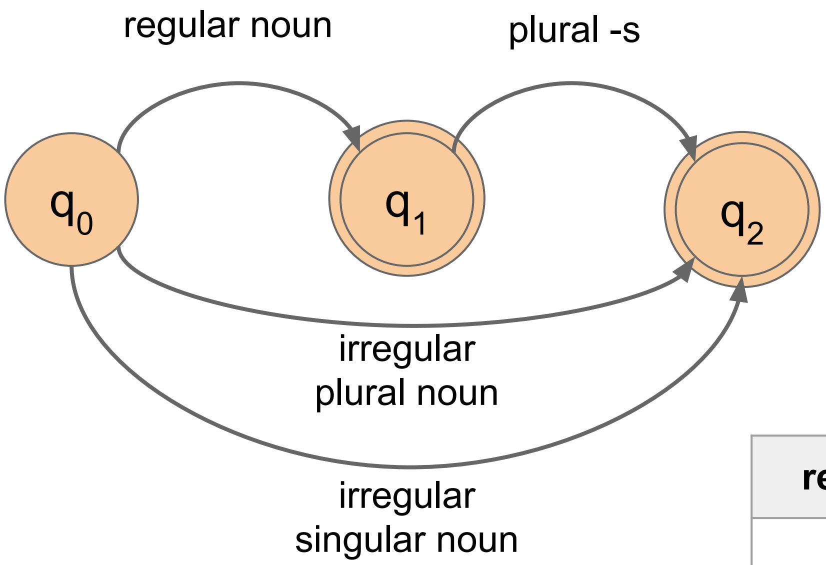


Union: Merging Automata

- grace,
  dis-grace,
  grace-ful,
  dis-grace-ful



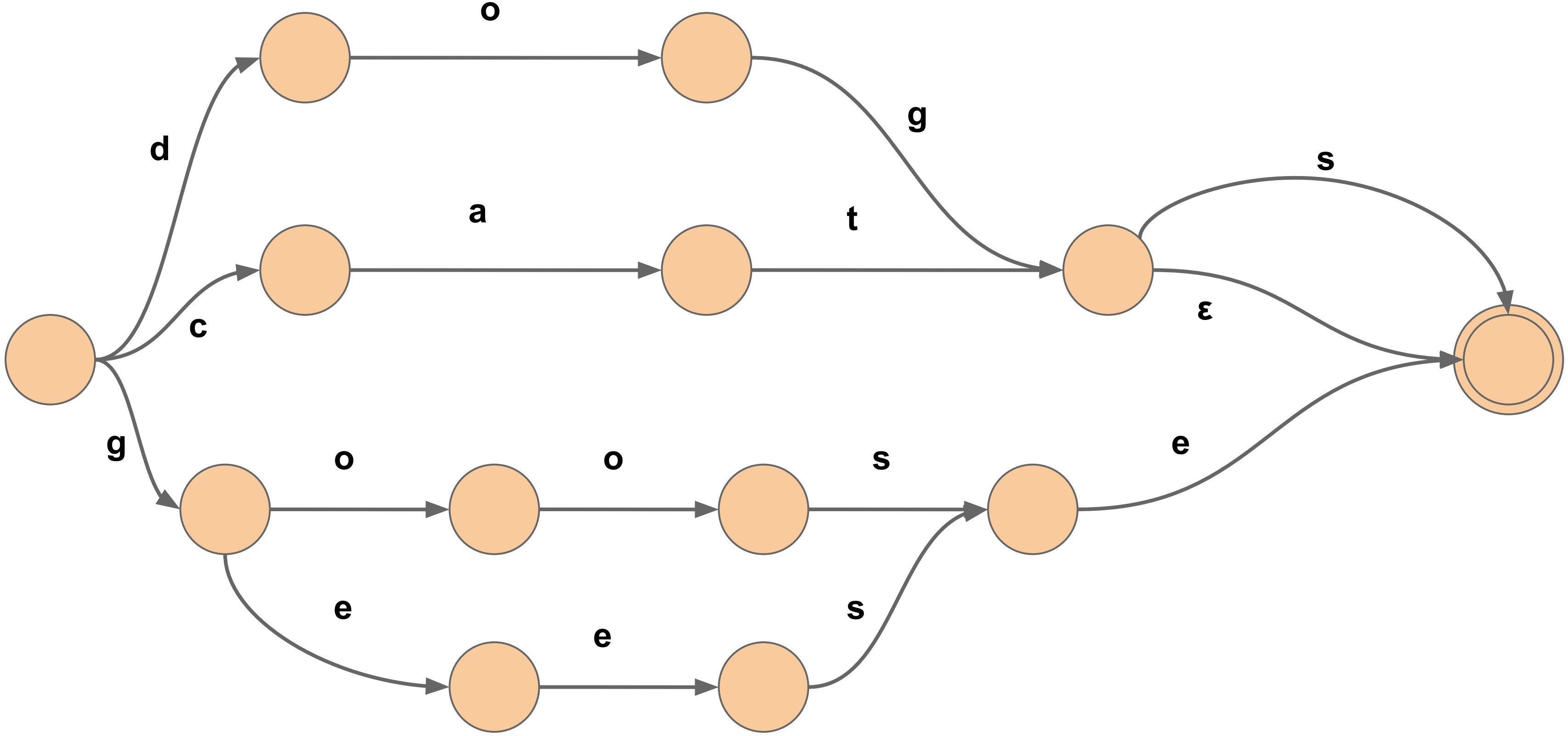
Simple FSA for English Nominal Inflection



- Some irregular words require stem changes, e.g. *goose - geese.*

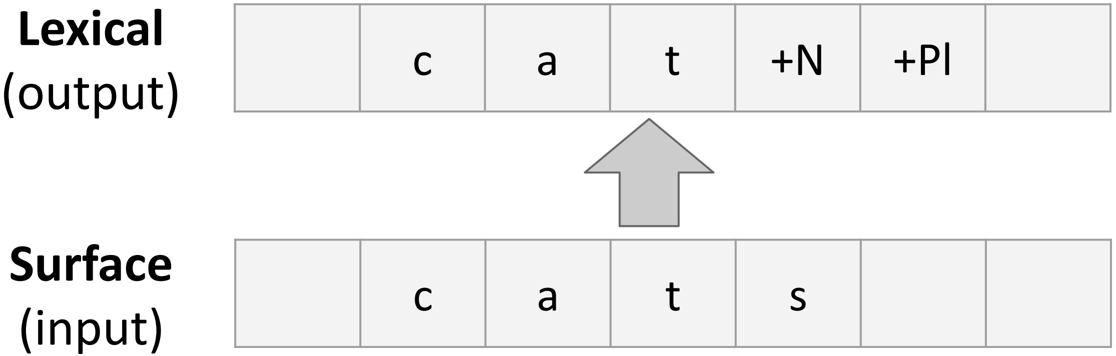
reg-noun	irreg-pl-noun	irreg-sg-noun	plural
dog	geese	goose	-s
cat	sheep	sheep	
aardvark	mice	mouse	

Expanded FSA for a Few English Nouns



Recognition vs. Analysis

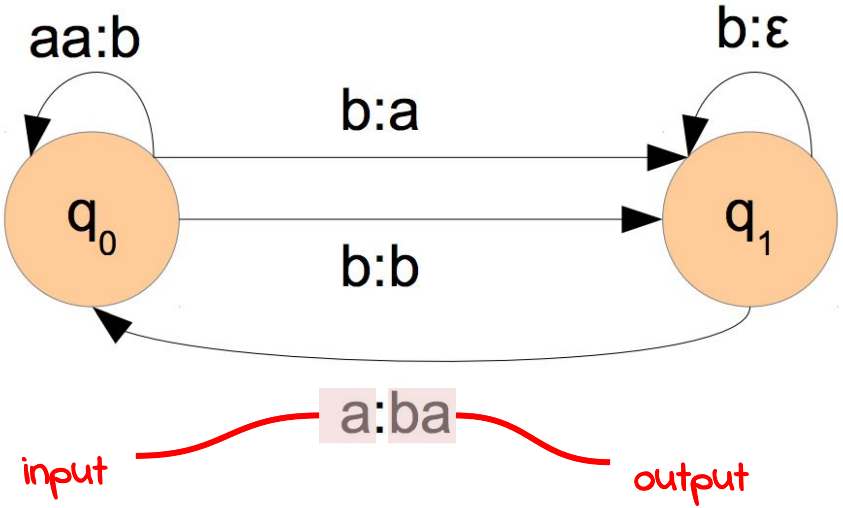
- FSAs can recognize (**accept**) a string, but they don’t tell us its internal structure.
- We need a machine that maps (**transduces**) the input string into an output string that encodes its structure:



Lexical (output)		c	a	t	+N	+Pl	
Surface (input)		c	a	t	s		

Finite State Transducer (FST)

- A Finite State Transducer maps between **two sets of symbols**.
- **2-tape FSA** that **recognizes** or **generates** pairs of strings.
- A FST defines relations between sets of strings.



Formalisation of a Finite State Transducer (FST)

- A **finite state transducer T = (Q, Σ, Δ, q0, F, $\boldsymbol{\delta}$, σ)** consists of:
    - **Q**: finite set $\{q_0, q_1, q_2, \ldots, q_{N-1}\}$ of N states
    - **Σ**: finite set of input symbols
    - **Δ: finite set of output symbols**
    - $\mathbf{q_0}$: the designated start state
    - **F**: the set of final states, $F \subseteq Q$
    - $\boldsymbol{\delta}$**(q,i)**: the transition function

      $\boldsymbol{\delta}: Q \times \Sigma \rightarrow 2^Q$, $\boldsymbol{\delta}(q,i) = Q'$ for $q \in Q$, $Q' \subseteq Q$, $i \in \Sigma$
    - **σ(q,i):** the output function

      $\sigma: Q \times \Sigma \rightarrow \Delta^*$, $\sigma(q,i) = \omega$ for $q \in Q$, $i \in \Sigma$, $\omega \in \Delta^*$

Formalisation of a Finite State Transducer (FST)

- A **finite state transducer T = L_{in} × L_{out}** defines a relation between two regular languages L_{in} and L_{out}:
- L_{in} = {cat, cats, fox, foxes, ...}
- L_{out} = {cat+N+Sg, cat+N+Pl, fox+N+Sg, fox+N+Pl ...}
- **T** = { <cat, cat+N+Sg>, <cats, cat+N+Pl>, <fox, fox+N+Sg>, <foxes, fox+N+Pl> }

FST as "translator":

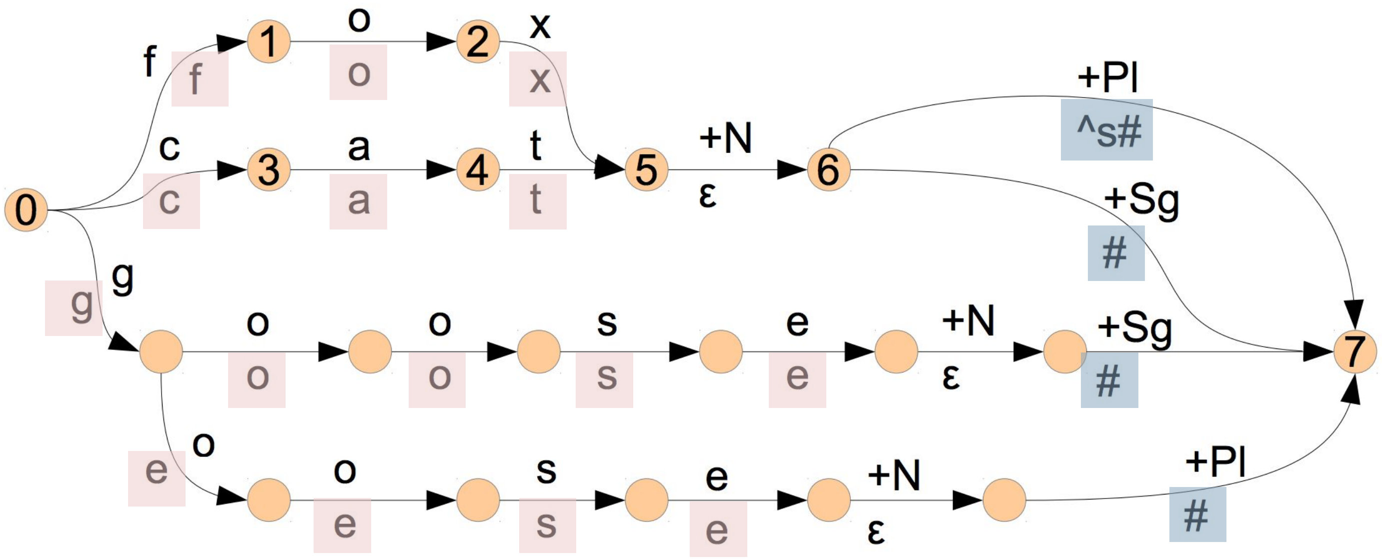
- Reads a string (L_{in}) and outputs another string (L_{out})
- **Morphological parsing**: Surface form (input); Lexical form (output)

Finite State Transducers for Morphological Parsing

- Two tapes
  - Upper (**lexical**) tape: output alphabet Δ
    - cat +N +Pl
  - Lower (**surface**) tape: input alphabet Σ
    - cats

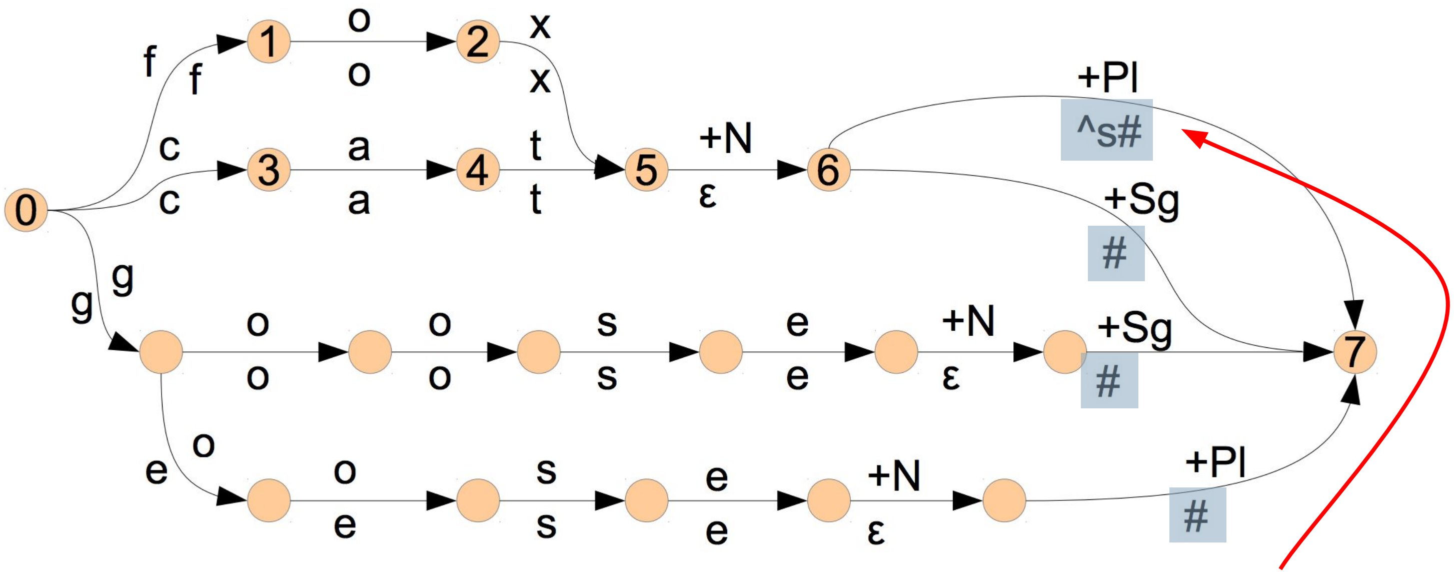
Lexical		c	a	t	+N	+Pl	
Surface		c	a	t	s		

Finite State Transducers for Morphological Parsing



- Lexical form : surface form ► goose:geese ► g:g o:e o:e s:s e:e
  - ○ **Default pairs** (g:g) vs. **Feasible pairs** (e.g., o:e)

Finite State Transducers for Morphological Parsing



- We indicate **morpheme boundaries** (^) and **word boundaries** (#).

FST and Orthographic Rules

- English often requires spelling changes at morpheme boundaries.
- Introduction of **orthographic rules**, as e.g.

Name	Orthographic Rule	Example
Consonant doubling	Consonant doubled before -ing/-ed	beg / begging
E deletion	Silent e dropped before -ing and -ed	make / making
E insertion	E added after -s, -z, -x, -cg. -sh before -s	fox / foxes
Y replacement	-y changes to -ie before -s, -i before -ed	try / tries
K insertion	Verbs ending with vowel + -c ad -k	panic / panicked

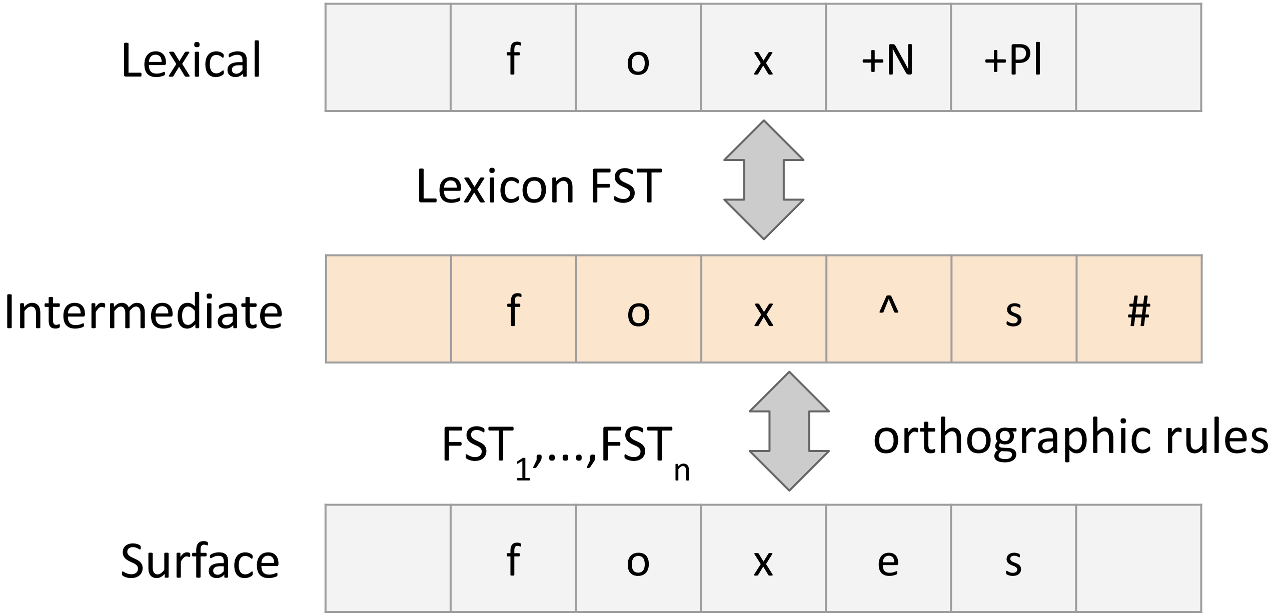
Intermediate Representations

- English plural -s:
  - cat ⇒ cats, dog ⇒ dogs
  - but: fox ⇒ foxes, buzz ⇒ buzzes
- Idea: We define an **intermediate representation** which captures **morpheme boundaries** (^) and **word boundaries** (#):

○	Lexical form:	cat+N+Pl	fox+N+Pl
○	Intermediate representation:	cat^s#	fox^s#
○	Surface form:	cats	foxes

- **Intermediate-to-Surface Spelling Rule:**
  *If **plural ‘s’** follows a morpheme ending in **‘x’,‘z’** or **‘s’**, **insert ‘e’**.*

FST and Orthographic Rules



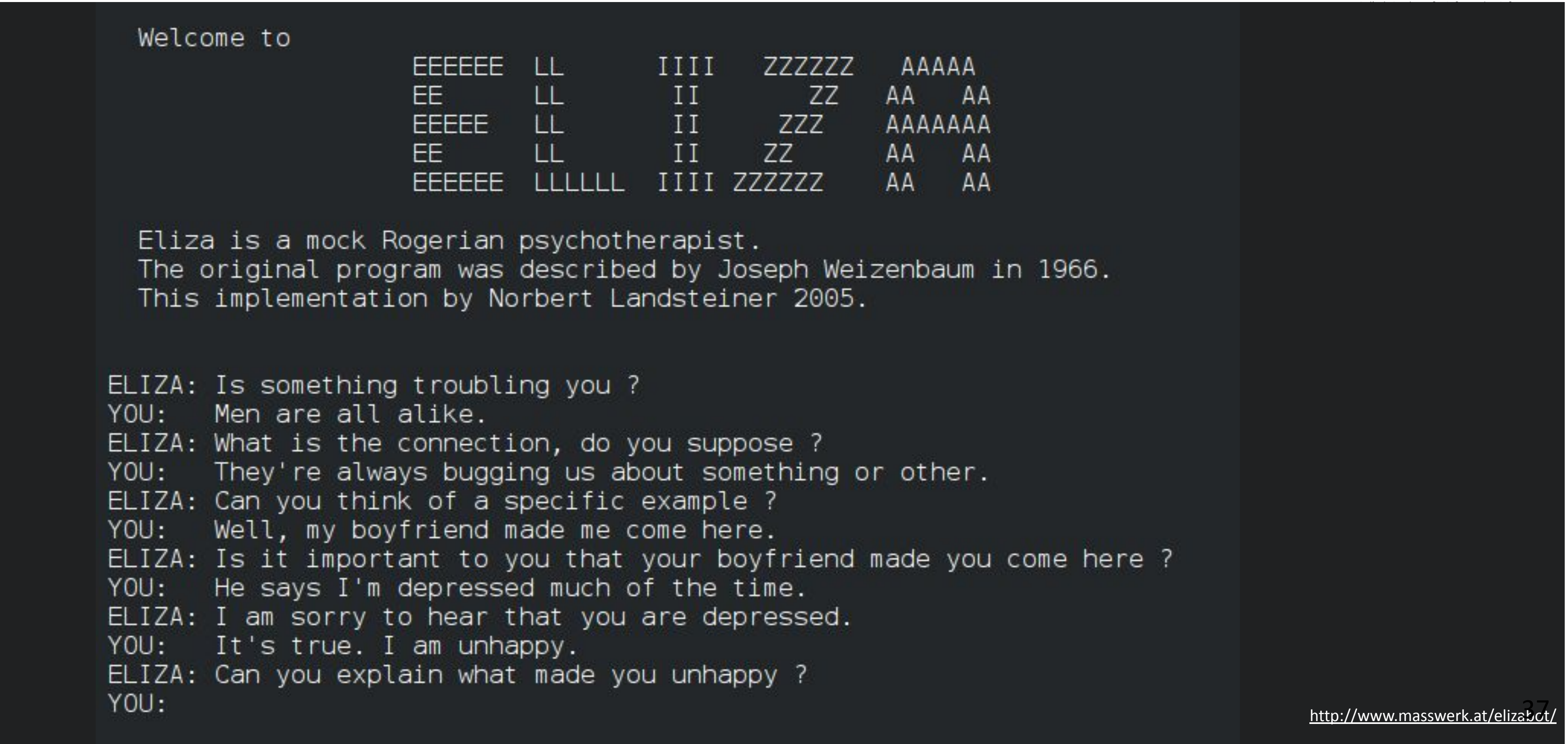
Lexical		f	o	x	+N	+Pl	
Intermediate		f	o	x	^	s	#
Surface		f	o	x	e	s	

Lexicon FST

$FST_1,...,FST_n$

orthographic rules

More FST Applications



```
  Welcome to
                    EEEEEE  LL      IIII   ZZZZZZ   AAAAA
                    EE      LL       II       ZZ   AA   AA
                    EEEEE   LL       II      ZZZ   AAAAAAA
                    EE      LL       II    ZZ      AA   AA
                    EEEEEE  LLLLLL  IIII ZZZZZZ    AA   AA

  Eliza is a mock Rogerian psychotherapist.
  The original program was described by Joseph Weizenbaum in 1966.
  This implementation by Norbert Landsteiner 2005.


ELIZA: Is something troubling you ?
YOU:   Men are all alike.
ELIZA: What is the connection, do you suppose ?
YOU:   They're always bugging us about something or other.
ELIZA: Can you think of a specific example ?
YOU:   Well, my boyfriend made me come here.
ELIZA: Is it important to you that your boyfriend made you come here ?
YOU:   He says I'm depressed much of the time.
ELIZA: I am sorry to hear that you are depressed.
YOU:   It's true. I am unhappy.
ELIZA: Can you explain what made you unhappy ?
YOU:
```

http://www.masswerk.at/elizabot/

ELIZA as an FST Cascade

- Human: You don't argue with me.
- Computer: WHY DO YOU THINK I DON'T ARGUE WITH YOU

- 1. Replace **you** with **I** and **me** with **you**:
  - "**You** don't argue with **me.**"
  - **I** DON'T ARGUE WITH **YOU.**
- 2. Replace <*string*> with **Why do you think <*string*>**:
  - **WHY DO YOU THINK** I DON'T ARGUE WITH YOU

Information Service Engineering

Lecture 4: Natural Language Processing (3)

2.0 What is Natural Language Processing?

2.1 NLP and Basic Linguistic Knowledge

2.2 Morphology

2.3 NLP Applications

2.4 NLP Techniques

2.5 NLP Challenges

2.6 Evaluation, Precision and Recall

2.7 Regular Expressions

2.8 Finite State Automata

2.9 Tokenization

2.10 Language Model and N-Grams

2.11 Part-of-Speech Tagging

Tokenization

- **Tokenization** is the process of breaking a stream of text up into words, phrases, symbols, or other meaningful elements.
- Distinguish
  - Word tokenization
  - Sentence tokenization
- At first glance, English word tokenization might seem simple, but...

Word Tokenization

- **English word tokenization** might simply make use of white spaces…

> Latest figures from the US government show the trade deficit with China reached an all-time high of $365.7bn (£250.1bn) last year. By February this year it had already reached $57bn.

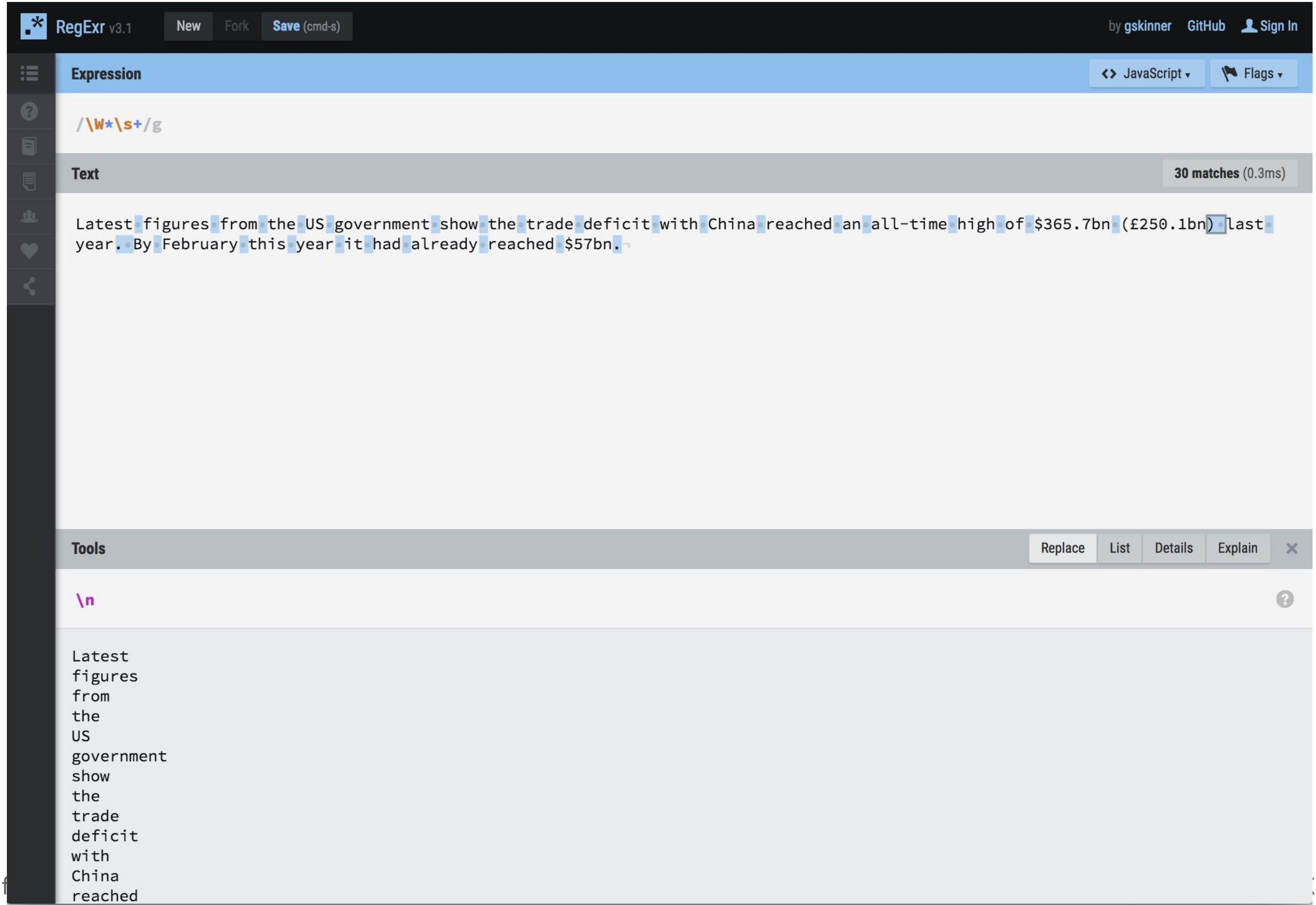
- Tokenization can easily be implemented via **regular expressions**:
  - `\W*\s+`

Anything non-alpha-numeric

Followed by white spaces

http://regexr.com/

Word Tokenization



```
/\W*\s+/g
```

30 matches (0.3ms)

```
Latest figures from the US government show the trade deficit with China reached an all-time high of $365.7bn (£250.1bn) last
year. By February this year it had already reached $57bn.
```

```
\n
```

```
Latest
figures
from
the
US
government
show
the
trade
deficit
with
China
reached
```

http://regexr.com/

Word Tokenization

Latest figures from the US government show the trade deficit with China reached an all-time high of $365.7bn (£250.1bn) last year. By February this year it had already reached $57bn.

- Intended result:

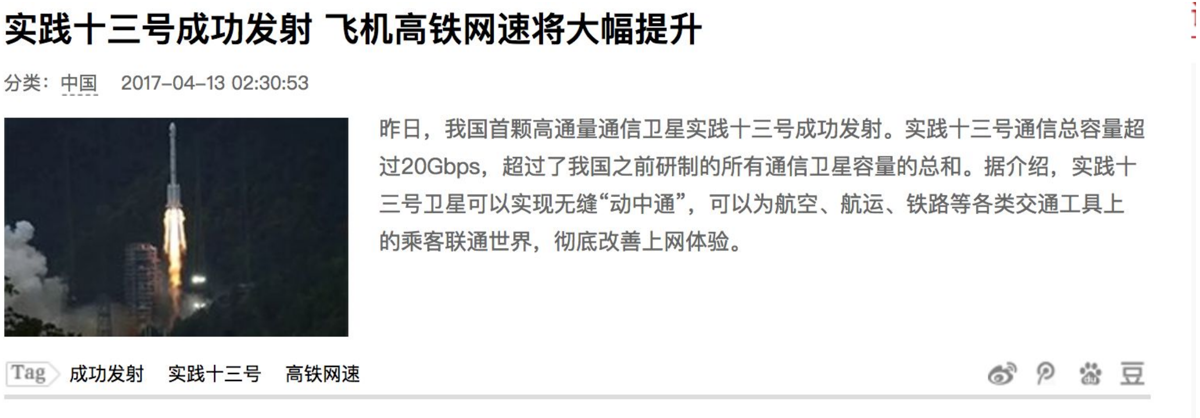
Latest figures from the US government show the trade deficit with China reached an all time high of $ 365.7 bn (£ 250.1 bn) last year . By February this year it had already reached $ 57 bn .

Word Tokenization

- Issues related to tokenization:
  - Separators: punctuations
  - Exceptions: „m.p.h“, „Ph.D“
    - Expansions: „we're“ = „we are“
    - Multi-words expressions: „New York“
  - Numbers:
    - Dates: *3/20/91*
    - More Dates: *55 B.C.*
    - IP addresses: *192.168.0.1*
    - Phone numbers: *(800) 234-2333*
    - ...

Segmentation = Tokenization

- **Word segmentation**: separation of the morphemes
  but also tokenization for languages without 'space' character.



实践十三号成功发射 飞机高铁网速将大幅提升

分类：中国 2017–04–13 02:30:53

昨日，我国首颗高通量通信卫星实践十三号成功发射。实践十三号通信总容量超过20Gbps，超过了我国之前研制的所有通信卫星容量的总和。据介绍，实践十三号卫星可以实现无缝“动中通”，可以为航空、航运、铁路等各类交通工具上的乘客联通世界，彻底改善上网体验。

Tag 成功发射 实践十三号 高铁网速

http://www.bjnews.com.cn/

- *Chinese, Japanese*: sentences but not words are delimited.
- *Thai and Lao*: phrases and sentences but not words are delimited.
- *Vietnamese*: syllables but not words are delimited.

Sentence Splitting

- Dividing a string of written language into its component sentences.
- In **English** and some other languages, using **punctuation**, particularly the *full stop/period character (.?!)* is a reasonable approximation.
- **Non trivial problem**, since in English the full stop character also is used for abbreviations or numbers.
    - Examples: „Mr.“, „4.5“

Sentence Splitting

- **“Vanilla” Approach**
  - If it's a **period**, it ends a sentence.
  - If the **preceding token** is in the **hand-compiled list of abbreviations**, then it doesn't end a sentence.
  - If the **next token** is **capitalized**, then it ends a sentence.

- Capable of detecting **ca. 95% of sentence boundaries**.
- Alternative Approaches:
  - Based on **regular expressions**
  - Based on **rules** or **machine learning**, e.g. binary classifiers that decide whether a certain punctuation is part of a word or not.

Information Service Engineering

Lecture 4: Natural Language Processing (3)

2.0 What is Natural Language Processing?

2.1 NLP and Basic Linguistic Knowledge

2.2 Morphology

2.3 NLP Applications

2.4 NLP Techniques

2.5 NLP Challenges

2.6 Evaluation, Precision and Recall

2.7 Regular Expressions

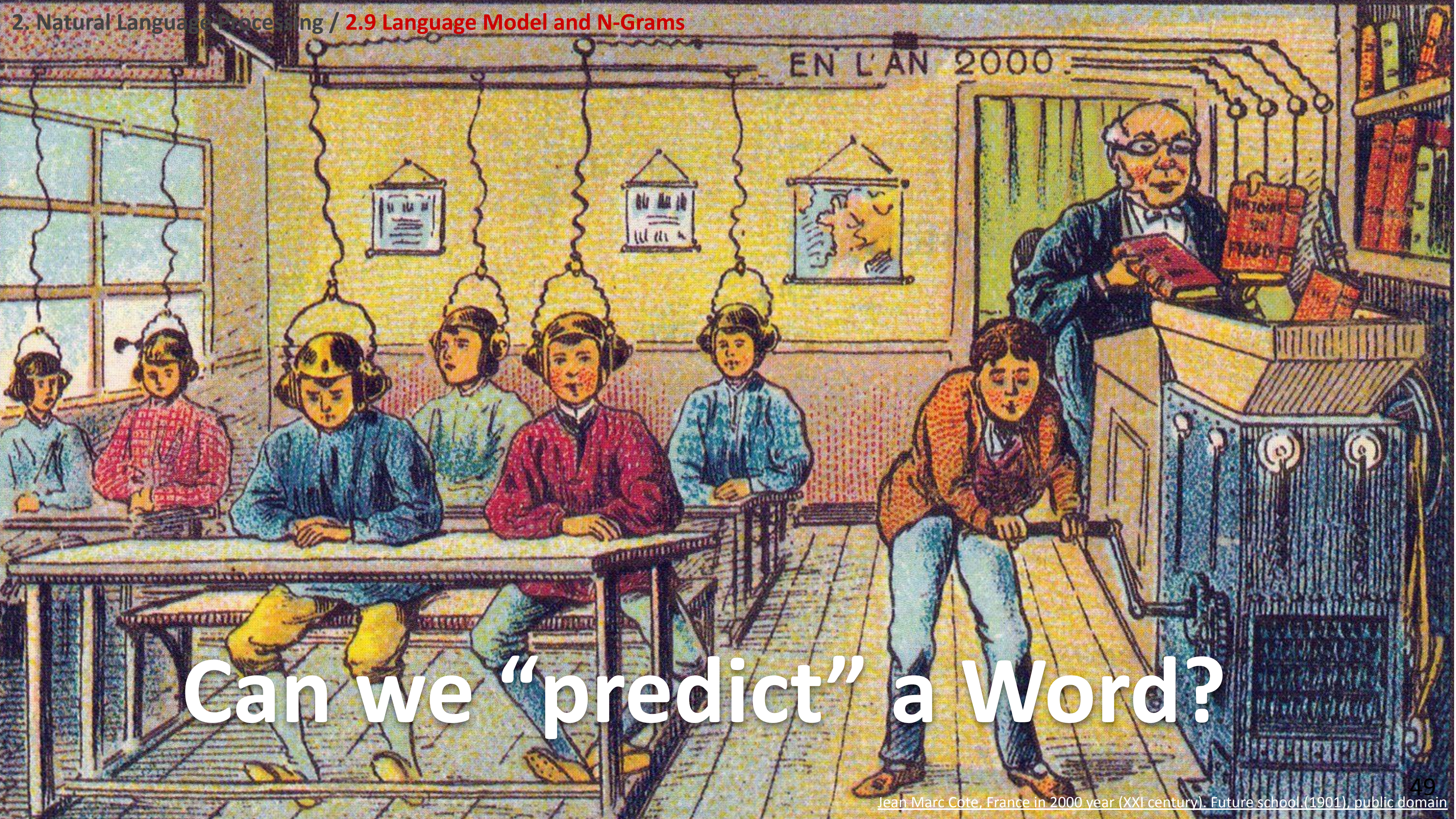
2.8 Finite State Automata

2.9 Tokenization

2.10 Language Model and N-Grams

2.11 Part-of-Speech Tagging

Can we “predict” a Word?



Jean Marc Cote, France in 2000 year (XXI century). Future school (1901), public domain

Word Prediction

- “To be or not to…”
- “The pen is mightier than the…”
- “You can’t judge a book by…”

- “Es irrt der Mensch, solang’ er ….”
- “Die Botschaft hör ich wohl, allein mir fehlt …”

Human Prediction

- How do humans predict words?
  - ○ **Domain knowledge**, as e.g.
    red blood vs. hat
  - ○ **Syntactic knowledge**, as e.g.
    The … <adjective|noun>
  - ○ **Lexical knowledge**, as e.g.
    Baked potato vs. steak

- **Claim**: A useful part of the knowledge needed to allow *Word Prediction* can be captured using **simple statistical techniques**.

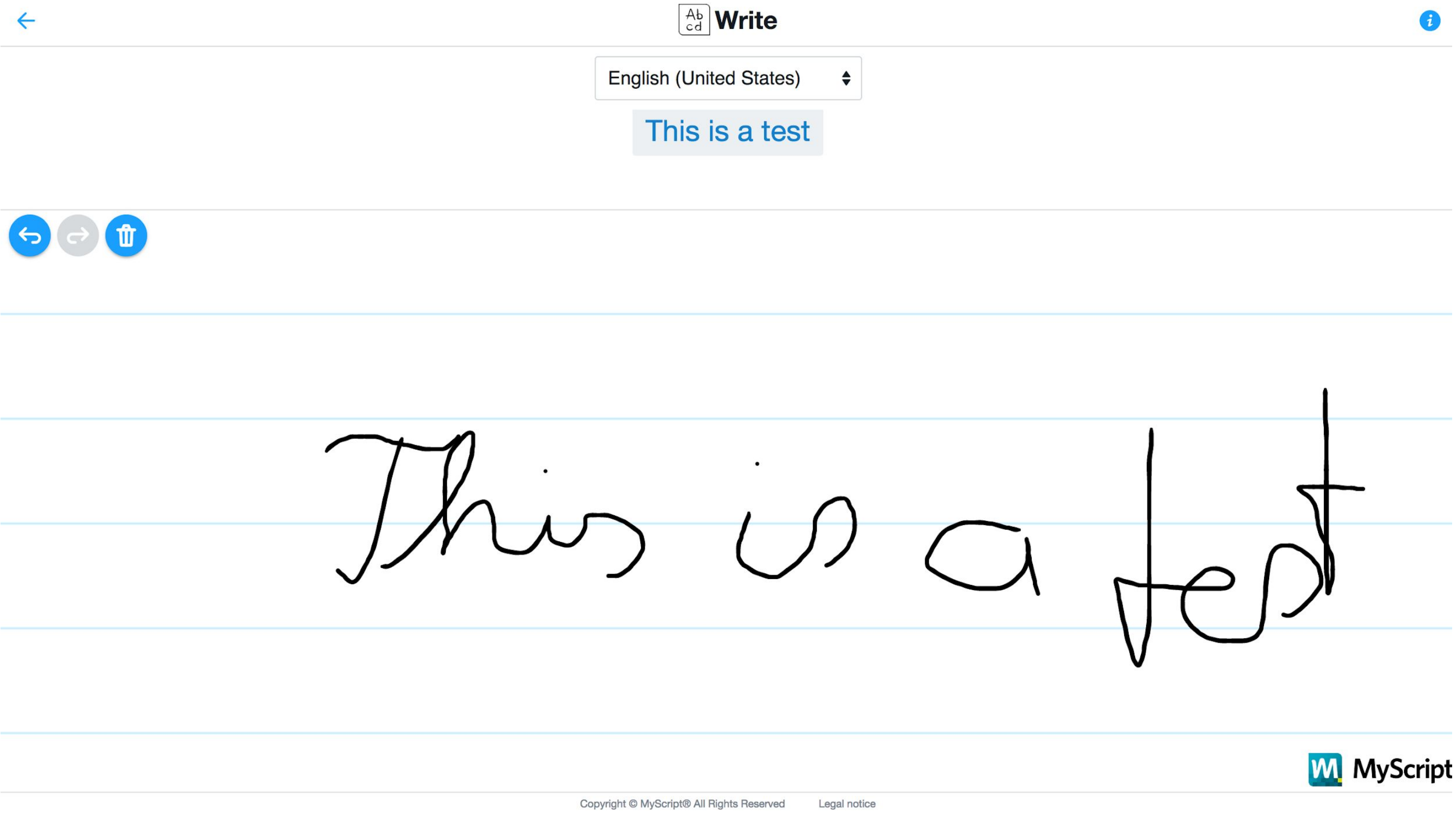
N-gram Models

- **Word Prediction** can be formalized with probabilistic **N-gram models**:
  - **2-gram (bigram):** (to, be), (be, or), (or, not), (not, to)
  - **3-gram (trigram)**: (to, be, or), (be, or, not), (or, not, to)
- An **N-gram** is an N-Token of words.
- In an N-gram model, the **last word w_n** depends only on the **previous n-1 words (w_1,...w_{n-1})** (**Markov assumption**)
- and thus, the **last word w_n** will be computed **from the previous n-1 words (w_1,...w_{n-1})**.
- Statistical models of word sequences are called **Language Models (LM)**.

Speech Recognition

- *„Computers can recognize speech.“*
  „Computers can wreck a nice peach.”
- *„Give peace a chance.“*
  „Give peas a chance.“
- *„ice cream.“*
  „I scream.“
- *“Two birds are flying.”*
  “Two beards are flying.”

Handwriting Recognition



Language: English (United States)

Recognized text: This is a test

Handwritten input: This is a test

http://webdemo.myscript.com/views/text.html

Basic Probability Theory

- **Trial:**
  - Throwing a dice, predicting a word.
- **Sample space** Ω:
  - The set of all possible outcomes
    (all numbers in a lottery; all words in Shakespeare’s plays).
- **Event** ω ⊆ Ω:
  - An actual outcome (a subset of Ω)
    (predicting ‘the’, throwing a “3”,...).

The Probability of Events

- **Kolmogorov Axioms:**
  1. Each event has a probability between 0 and 1.

     $$0 \leq P(\omega \subseteq \Omega) \leq 1$$

  2. The null event has probability 0.

     The probability that any event happens is 1.

     $$P(\emptyset) = 0 \text{ and } P(\Omega) = 1$$

  3. The probability of all disjoint events sums to 1.

     $$\sum_{\omega_i \subseteq \Omega} P(\omega_i) = 1 \quad \forall j \neq i : \omega_i \cap \omega_j = \emptyset, \bigcup_i \omega_i = \Omega$$

Statistical Language Model

- Finding the probability of a sentence or a sequence of words:

$$P(S) = P(w_1, w_2, \cdots, w_n)$$

- Example:
  - *„Computers can recognize speech.“*
  - *P(Computer, can, recognize, speech)*

- Rank possible sentences:
  - *P(“Today is Wednesday”) > P(“Wednesday today is”)*
  - *P(“Today is Wednesday”) > P(“Today is book”)*

Conditional Probability

Thomas Bayes
(1700-1761)

$$P(B|A) = \frac{P(A \cap B)}{P(A)}$$

Conditional Probability ***P(B|A)***
that **event B** occurs under the assumption that **event A** has already occurred.

$$P(A,B) = P(A) \cdot P(B|A)$$

Bayes Theorem
The probability that **event A occurs followed by event B** equals the probability that event A occurs and event B occurs under the assumption that event A has occurred.

$$P(A,B,C,D) = P(A) \cdot P(B|A) \cdot P(C|A,B) \cdot P(D|A,B,C)$$

Extension to multiple events via **chain rule**

Conditional Probability

$$P(S) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_1, w_2) \cdots P(w_n|w_1, \cdots, w_{n-1})$$

$$P(S) = \prod_{i=1}^{n} P(w_i|w_1, \cdots, w_{i-1})$$

Generalization of the Bayes Theorem for modelling a sequence of words in a (natural) language.

- P(*to be or not*) = P(*to*) · P(*be*|*to*) · P(*or*|*to,be*) · P(*not*|*to,be,or*)
- But how do we determine the **probability of the occurrence of words**?

Conditional Probability

$$P(S) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_1, w_2) \cdots P(w_n|w_1, \cdots, w_{n-1})$$

$$P(S) = \prod_{i=1}^{n} P(w_i|w_1, \cdots, w_{i-1})$$

Generalization of the Bayes Theorem for modelling a sequence of words in a (natural) language.

- P(*to be or not*) = P(*to*) ·
  P(*be*|*to*) ·
  P(*or*|*to,be*) ·
  P(*not*|*to,be,or*)

- But how do we determine the **probability of the occurrence of words**?

Corpora

- To understand and model how language works, we need empirical evidence.

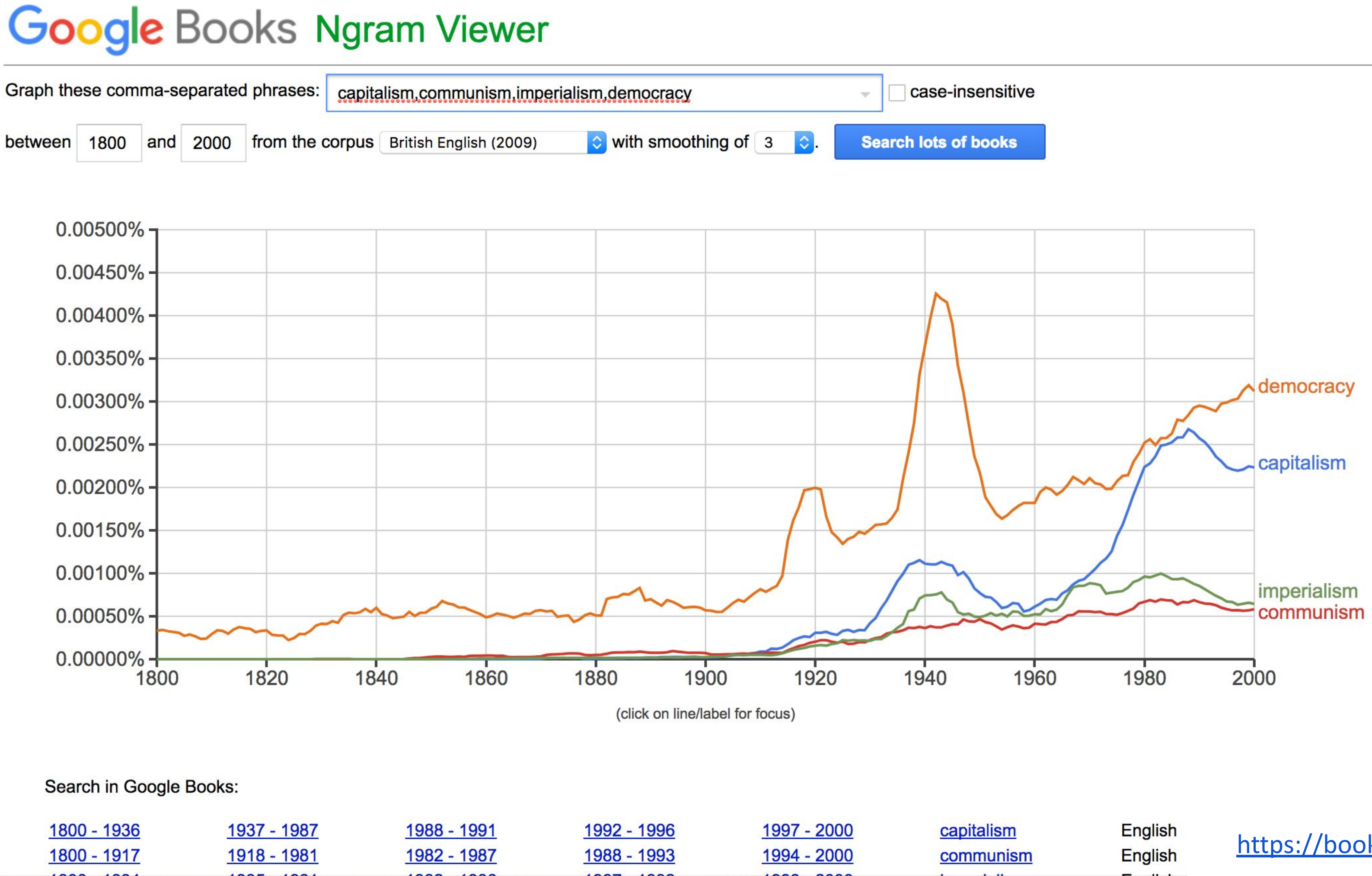
- **Probabilities** are based on **counting things**.
- **Idea:** Count the occurrence of words **in large collections of texts (=corpora)**.

- **A corpus** is a computer-readable collection of text or speech.
  Ideally, naturally-occurring corpora serve as realistic samples of a language.

○ Corpus of Contemporary **American English:**	520m words, US, 1990-2015
○ The **British** National Corpus:	100m words, UK, 1991-1994
○ The **International** Corpus of English:	23 local corpora, 1m words each

- ○ **The Google N-gram Corpus**
  - ■ N-grams from printed sources, 1500-2008, in English, Chinese, French, German, Hebrew, Italian, Russian, or Spanish,
    1,024,908,267,229 words.

http://www.corpusdata.org/
http://www.natcorp.ox.ac.uk/
https://books.google.com/ngrams
https://research.googleblog.com/2006/08/all-our-n-gram-are-belong-to-you.html



Graph these comma-separated phrases: capitalism,communism,imperialism,democracy □ case-insensitive

between 1800 and 2000 from the corpus British English (2009) with smoothing of 3.

(click on line/label for focus)

Search in Google Books:

1800 - 1936	1937 - 1987	1988 - 1991	1992 - 1996	1997 - 2000	capitalism	English
1800 - 1917	1918 - 1981	1982 - 1987	1988 - 1993	1994 - 2000	communism	English

https://books.google.com/ngrams

Complexity of a Statistical Language Model

$$P(S) = P(w_1)\cdot P(w_2|w_1)\cdot P(w_3|w_1, w_2)\cdots P(w_n|w_1,\cdots,w_{n-1})$$

$$P(S) = \prod_{i=1}^{n} P(w_i|w_1,\cdots,w_{i-1})$$

- **Complexity**: $O(|V|^{n^*})$ V...Vocabulary, n^*...maximum sentence length
  - 475,000 main headwords in *Webster's Third New International Dictionary*
  - Average English sentence length: 14.3 words
  - A rough estimate: $O(475{,}000^{14}) \sim 3.38 \cdot 10^{66}$ TB
- By applying an **N-gram model**, we make the model more compact: $O(475{,}000^{N})$.

N-gram Models

- The intuition of the **N-gram model** is that
  - instead of computing the probability of a word given its entire history,
  - we can **approximate** the history **by just the last few words**.

- For the **bigram model** we approximate the probability of a word given all its previous words by using **only the conditional probability of its preceding word** (**Markov assumption**):

$$P(w_n|w_1, \cdots, w_{n-1}) \approx P(w_n|w_{n-1})$$

Markov Assumption

- Using the **Markov Assumption** to compute the probability of a text sequence for the bigram model:

$$P(S) = \prod_{i=1}^{n} P(w_i | w_1, \cdots, w_{i-1})$$

$$\Downarrow$$

$$P(S) = \prod_{i=1}^{n} P(w_i | w_{i-1})$$

N-gram Model

- Unigram: $$P(S) = \prod_{i=1}^{n} P(w_i)$$
- Bigram: $$P(S) = \prod_{i=1}^{n} P(w_i|w_{i-1})$$
- Trigram: $$P(S) = \prod_{i=1}^{n} P(w_i|w_{i-1}, w_{i-2})$$
- N-gram: $$P(S) = \prod_{i=1}^{n} P(w_i|w_1, \cdots, w_{i-1})$$

Maximum Likelihood Estimation

- How to estimate N-gram probabilities?
- **Maximum Likelihood Estimation (MLE)**
  - A method of estimating the parameters of a statistical model given **observations**.
  - By finding the parameter values that maximize the likelihood of making the observations given the parameters.
- The **MLE for the parameters of an N-gram model** is computed by **normalizing counts from a corpus**.

$$P(w_n|w_{n-1}) = \frac{\#(w_{n-1}w_n)}{\sum_w \#(w_{n-1}w)} = \frac{\#(w_{n-1}w_n)}{\#w_{n-1}}$$

Markov Assumption and Maximum Likelihood Estimation

$$\begin{aligned}\mathrm{P(to\ be\ or\ not)} \quad = \quad & \mathrm{P(not|to\ be\ or)} \cdot \\ & \mathrm{P(or|to\ be)} \cdot \\ & \mathrm{P(be|to)} \cdot \\ & \mathrm{P(to)}\end{aligned}$$

Markov Assumption

$$\begin{aligned}\mathrm{P(to\ be\ or\ not)} \quad = \quad & \mathrm{P(to)} \cdot \\ & \mathrm{P(be|to)} \cdot \\ & \mathrm{P(or|be)} \cdot \\ & \mathrm{P(not|or)}\end{aligned}$$

Maximum Likelihood Estimation

$$P(not|or) = \frac{\#or\ not}{\#or}$$

N-gram Model - Generating Shakespeare

- Unigram: To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have
  Hill he late speaks; or! a more to leg less first you enter.
- Bigram: Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow.
  What means, sir. I confess she? then all sorts, he is trim, captain
- Trigram: Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, ’tis done.
  This shall forbid it should be branded, if renown made it empty.
- 4-gram: King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv’d in;
  It cannot be but so.

How to generate “plausible” Text from N-grams

- Example for 2-grams (for n-grams simply adapt).

- From your corpus:
  1. Choose a random 2-gram with (<s>,w_1)
  2. Next choose another random n-gram (w_1,w_2)
  3. Continue choosing (w_i,w_{i+1}), until you choose (w_n,</s>) as the last word.
  4. Then tie all new words (<s>,w_1,...,w_n,</s>) together in a sentence.

- Why does it work?
  - |Shakespeare Corpus|=884,647 tokens, |V|=29,066
  - Shakespeare produced only 300,000 2-gram types out of $|V|^2 = 844 \cdot 10^6$ possible 2-grams.
  - So, 99.96% of the possible bigrams were never used.
  - 4-grams: The output looks like Shakespeare because it is fragments of Shakespeare...

Information Service Engineering

Lecture 4: Natural Language Processing (3)

2.0 What is Natural Language Processing?

2.1 NLP and Basic Linguistic Knowledge

2.2 Morphology

2.3 NLP Applications

2.4 NLP Techniques

2.5 NLP Challenges

2.6 Evaluation, Precision and Recall

2.7 Regular Expressions

2.8 Finite State Automata

2.9 Tokenization

2.10 Language Model and N-Grams

2.11 Part-of-Speech Tagging

2. Natural Language Processing - 3

Bibliography

- D. Jurafsky, J. H. Martin, *Speech and Language Processing, 2nd ed (draft)*, 2007,
  - Section 2.2, *Finite State Automata*
  - Section 3.2-3.8, *Finite State Transducers*

    (please note that this refers to the 2nd ed.)

- D. Jurafsky, J. H. Martin, *Speech and Language Processing, 3rd ed (draft).*, 2019,
  - Section 3.1, *N-grams*

    (please note that this refers to the 3rd ed.)

2. Natural Language Processing - 3

Syllabus Questions

- Define a Finite State Automaton.
- What is a Finite State Automaton used for in NLP?
- What is the difference between a Finite State Automaton and a Finite State Transducer?
- How (in principle) is morphological parsing implemented with an FST?
- What is tokenization?
- What are the challenges for sentence tokenization and word tokenization?
- Sketch a simple approach (vanilla approach) for sentence tokenization that achieves better results than only looking for sentence delimiters.
- What is a language model?
- What is the purpose of a language model?
- Why are we using N-grams to approximate a language model?

2. Natural Language Processing - 3

Images

[1] Stephen Cole Kleene 1978, photo: Konrad Jacobs, Erlangen, Copyright is MFO, [CC-SA 2.0], via Wikimedia Commons, https://commons.wikimedia.org/wiki/File:Kleene.jpg

[2] Noam Chomsky, 8. Dec 1977, [CC0 1.0], via Wikimedia Commons, https://commons.wikimedia.org/wiki/File:Noam_Chomsky_(1977).jpg

[3] A conversation with the ELIZA chatbot, public domain, https://commons.wikimedia.org/wiki/File:ELIZA_conversation.png

[4] Jean Marc Cote, France in 2000 year (XXI century). Future school.(1901), public domain, https://commons.wikimedia.org/wiki/File:France_in_XXI_Century._School.jpg

[5] Thomas Bayes (d. 1761) in Terence O'Donnell, History of Life Insurance in Its Formative Years (Chicago: American Conservation Co., 1936), p. 335, public domain, https://commons.wikimedia.org/wiki/File:Thomas_Bayes.gif