Language Processing

COMP3411/9814: Artificial Intelligence

Lecture Overview

- Introduction

- Formal languages and grammars

- Regular expressions

- Minimum edit distance and words

- Natural language modelling: N-gram models

Lecture Overview

- Introduction

- Formal languages and grammars

- Regular expressions

- Minimum edit distance and words

- Natural language modelling: N-gram models

Introduction

- **NLP applications:**

  - Chatbots (customer service), personal assistant (Siri, Alexa), machine translation, social robotics (home).

- **Central problem – Ambiguity:**

  - Ambiguity makes it difficult to interpret meaning.

  - For instance, "The boy saw a girl with a telescope".

Introduction

Jack lost his wallet in his car.
He looked for it for several hours.

Jack forgot his wallet.
Sam did too.

Jack forgot his wallet.
He looked for someone to borrow money from.
Sam did too.

I saw two bears.
Bill saw some too.

Introduction

Discourse Structure

E: So you have the engine assembly finished.
        Now attach the rope to the top of the engine.
          By the way, did you buy petrol today?

A: Yes. I got some when I bought the new lawnmower wheel.
        I forgot to take my can with me, so I bought a new one.

E: Did it cost much?

A: No, and I could use another anyway.

E: OK. Have you got it attached yet?

Tracking focus isn't enough

Introduction

Hierarchical Structure

SEG1

Jack and Sue went to buy a new lawnmower since their old one was stolen.

SEG2

Sue had seen the man who took it and she had chased him down the street, but he'd driven away in a truck.

After looking in the store, they realised they couldn't afford one.

SEG3

By the way, Jack lost his job last month so he's been short of cash recently. He has been looking for a new one, but so far hasn't had any luck.

Anyway, they finally found a used one at a garage sale.

Lecture Overview

- Introduction

- Formal languages and grammars

- Regular expressions

- Minimum edit distance and words

- Natural language modelling: N-gram models

Grammars

- A grammar rule is a formal device for defining sets of sequences of symbols.
- Sequence may represent a statement in a programming language.
- Sequence may be a sentence in a natural language such as English.
- Formally, a grammar is a 4-tuple as: $G = <V, \Sigma, R, S>$:
  - V is a finite set of non-terminal symbols.
  - Σ is a finite set of terminal symbols.
  - R is a set of relations defined in $V \times (V \cup \Sigma)^*$.
  - S is the start symbol.

Notation for Rules

- A grammar specification consists of production rules, such as:

<S> ::= a b

<S> ::= a <S> b

- First rule says that whenever S appears in a string, it can be rewritten with the sequence ab.

- Second rule says that S can be rewritten with **a** followed by S followed by **b**.

- S is a non-terminal symbol, **a** and **b** are terminal symbols.

- A grammar rule can generate a string, e.g.:

S → a S b

a S b → a a S b b

a a S b b → a a a b b b

A Simple Subset of English

sentence --> noun_phrase, verb_phrase

noun_phrase --> determiner, noun

verb_phrase --> verb, noun_phrase

determiner --> [a]

determiner --> [the]

noun --> [cat]

noun --> [mouse]

verb --> [scares]
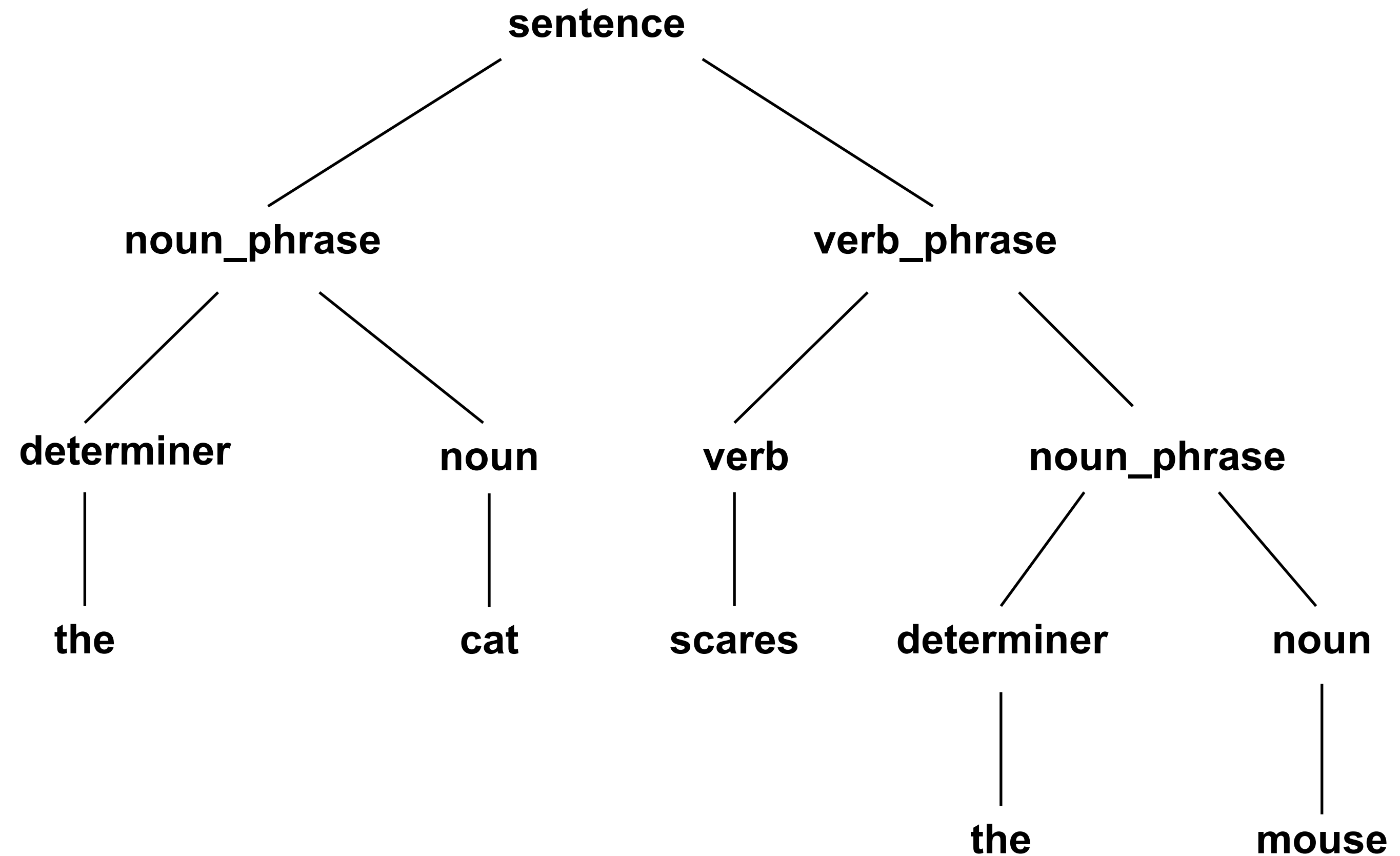
verb --> [hates]

Examples of derivations:

the cat scares the mouse

the mouse hates the cat

the mouse scares the mouse

Parse Trees

- Leaves are labelled by the terminal symbols of the grammar.

- Internal nodes are labelled by non-terminals.

- The parent-child relation is specified by the rules of the grammar.

Typical (Small) Grammar

S → NP VP

NP → [Det] Adj* N [AP | PP | Rel Clause]*

VP → V [NP] [NP] PP*

AP → Adj PP

PP → P NP

Det → a | an | the | . . .

N → John | Mary | park | telescope | . . .

V → saw | likes | believes | . . .

Adj → hot | hotter | . . .

P → in | with | . . .

Extra notation: * is "0 or more"; [. .] is "optional"

Leftmost Derivation Example

S
⇒ NP VP
⇒ N VP
⇒ John VP
⇒ John V NP PP
⇒ John saw NP PP
⇒ John saw N PP
⇒ John saw Mary PP
⇒ John saw Mary P NP
⇒ John saw Mary with NP
⇒ John saw Mary with Det N
⇒ John saw Mary with a N
⇒ John saw Mary with a telescope

⇒ means "rewrites as"

S → NP VP
NP → [Det] Adj* N [AP | PP | Rel Clause]*
VP → V [NP] [NP] PP*
AP → Adj PP
PP → P NP
Det → a | an | the | …
N → John | Mary | park | telescope | . . .
V → saw | likes | believes | . . .
Adj → hot | hotter | . . .
P → in | with | . . .

Rightmost Derivation

S

$\Rightarrow$ NP VP

$\Rightarrow$ NP V NP PP

$\Rightarrow$ NP V NP P NP

$\Rightarrow$ NP V NP P Det N

$\Rightarrow$ NP V NP P Det telescope

$\Rightarrow$ NP V NP P a telescope

$\Rightarrow$ . . .

$\Rightarrow$ . . .

$\Rightarrow$ . . .

S $\rightarrow$ NP VP
NP $\rightarrow$ [Det] Adj* N [AP | PP | Rel Clause]*
VP $\rightarrow$ V [NP] [NP] PP*
AP $\rightarrow$ Adj PP
PP $\rightarrow$ P NP
Det $\rightarrow$ a | an | the | …
N $\rightarrow$ John | Mary | park | telescope | . . .
V $\rightarrow$ saw | likes | believes | . . .
Adj $\rightarrow$ hot | hotter | . . .
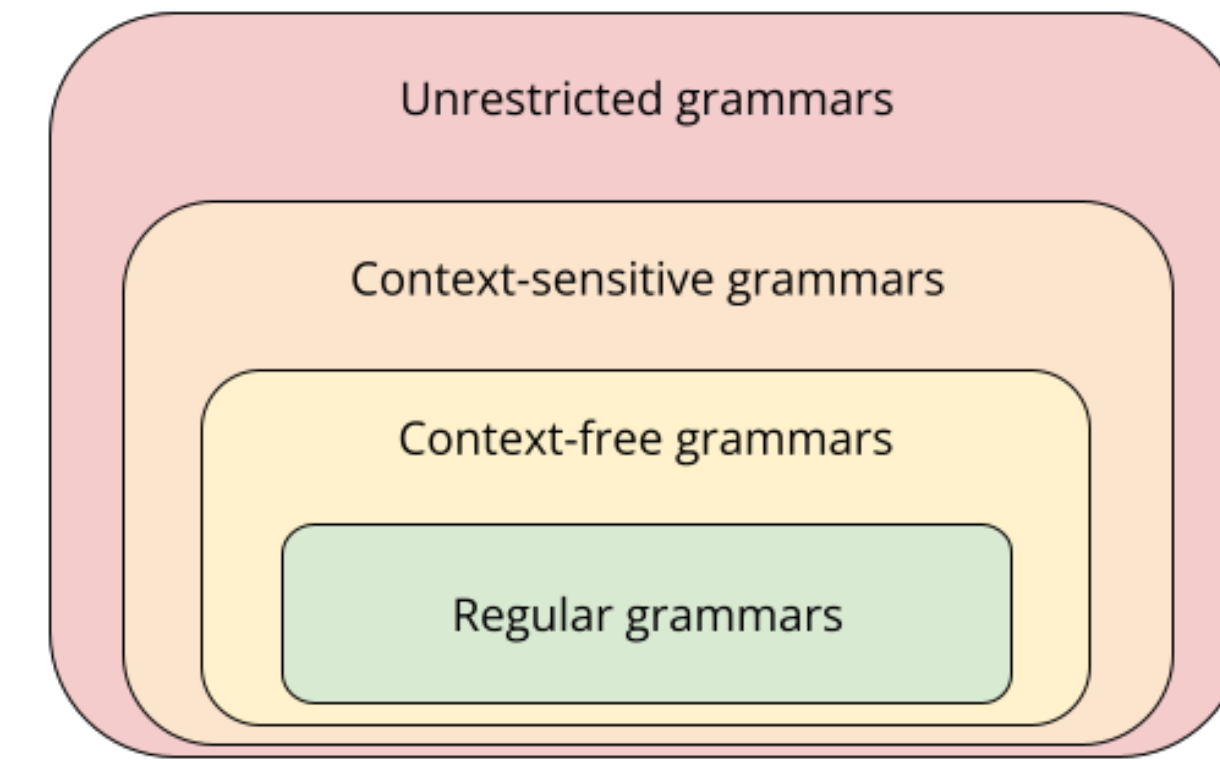P $\rightarrow$ in | with | . . .

Chomsky's Hierarchy

- Grammatical formalisms can be classified by their generative capacity.

- Four classes of grammatical formalisms that differ only in the form of the rewrite rules.

- The classes can be arranged in a hierarchy.

Unrestricted grammars

Context-sensitive grammars

Context-free grammars

Regular grammars
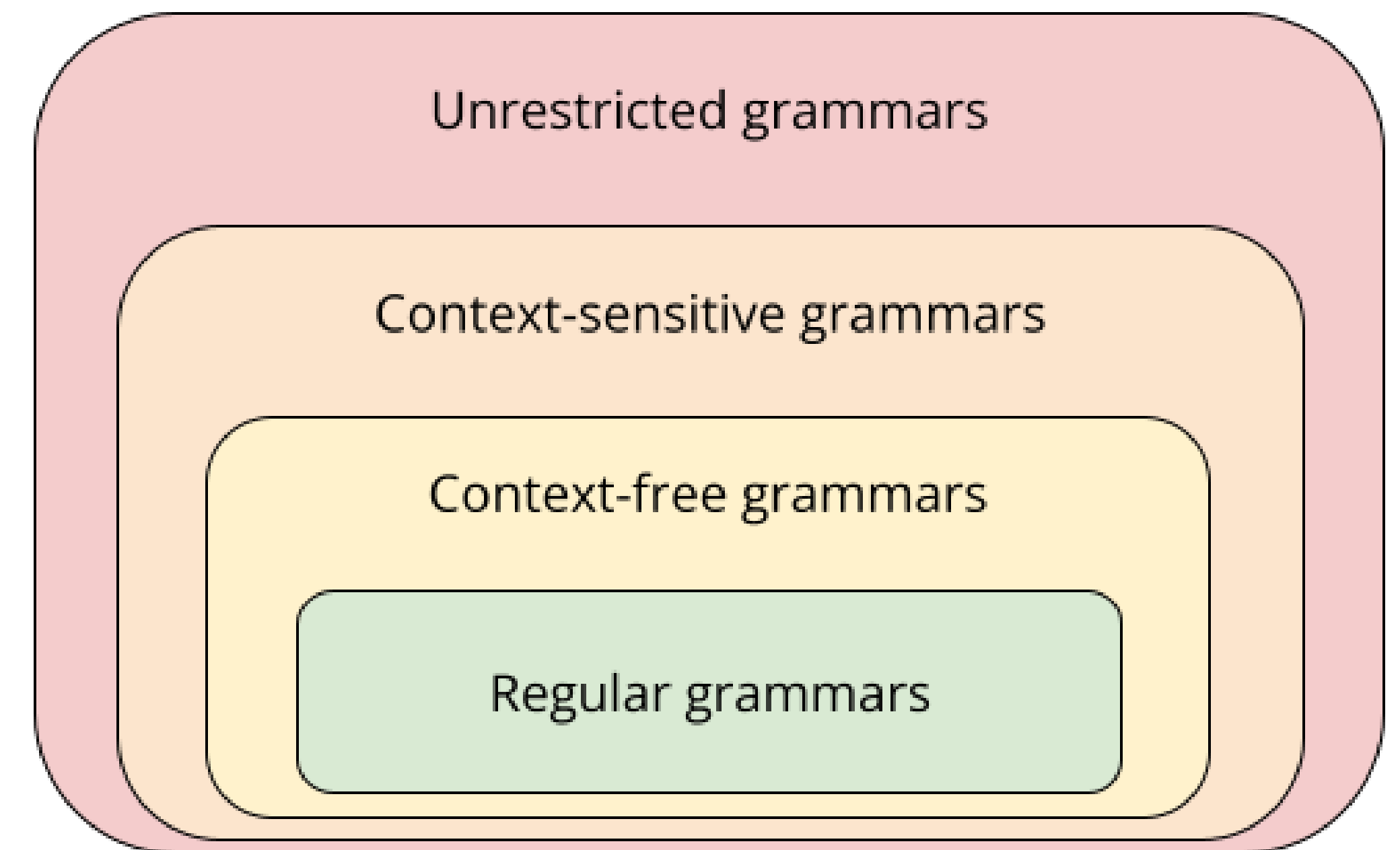
The Chomsky's Hierarchy

Chomsky's Hierarchy

- **Unrestricted grammars**: both sides of the rewrite rules can have any number of terminal and nonterminal symbols, as in the rule A B C → D E.

- **Context-sensitive grammars:** the right-hand side must contain at least as many symbols as the left-hand side. The name "context-sensitive" comes from the fact that a rule such as A X B → A Y B says that an X can be rewritten as a Y in the context of a preceding A and a following B. Context-sensitive grammars can represent languages such as $a^nb^nc^n$.

- **Context-free grammars:** the left-hand side consists of a single non-terminal symbol. Thus, each rule licenses rewriting the nonterminal as the right-hand side in any context. Context-free grammars can represent a^nb^n, but not $a^nb^nc^n$.

- **Regular grammars:** every rule has a single non-terminal on the left-hand side and a terminal symbol optionally followed by a non-terminal on the right-hand side. They cannot represent a^nb^n. The closest they can come is representing a^*b^*, a sequence of any number of a's followed by any number of b's.

Lecture Overview

- Introduction

- Formal languages and grammars

- Regular expressions

- Minimum edit distance

- Natural language modelling: N-gram models

Regular expressions

A formal language for specifying text strings

How can we search for any of these?
- ◦ woodchuck
- ◦ woodchucks
- ◦ Woodchuck
- ◦ Woodchucks

- ◦ Sophisticated sequences of regular expressions are often the first model for any text processing text.

Regular Expressions: Disjunctions

Letters inside square brackets []

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

Ranges [A-Z]

Pattern	Matches	
[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

Regular Expressions: Negation in Disjunction

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

Pattern	Matches	
`[^A-Z]`	Not an upper case letter	`O`<u>`y`</u>`fn pripetchik`
`[^Ss]`	Neither 'S' nor 's'	<u>`I`</u>` have no exquisite reason"`
`[e^]`	Either e or ^	`Look h`<u>`e`</u>`re`
`a^b`	The pattern a carat b	`Look up `<u>`a^b`</u>` now`

Regular Expressions: More Disjunction

Woodchuck is another name for groundhog!

The pipe | for disjunction

Pattern	Matches
groundhog|woodchuck	woodchuck
yours|mine	yours
a|b|c	= [abc]
[gG]roundhog|[Ww]oodchuck	Woodchuck

Regular Expressions: ? *+.

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	
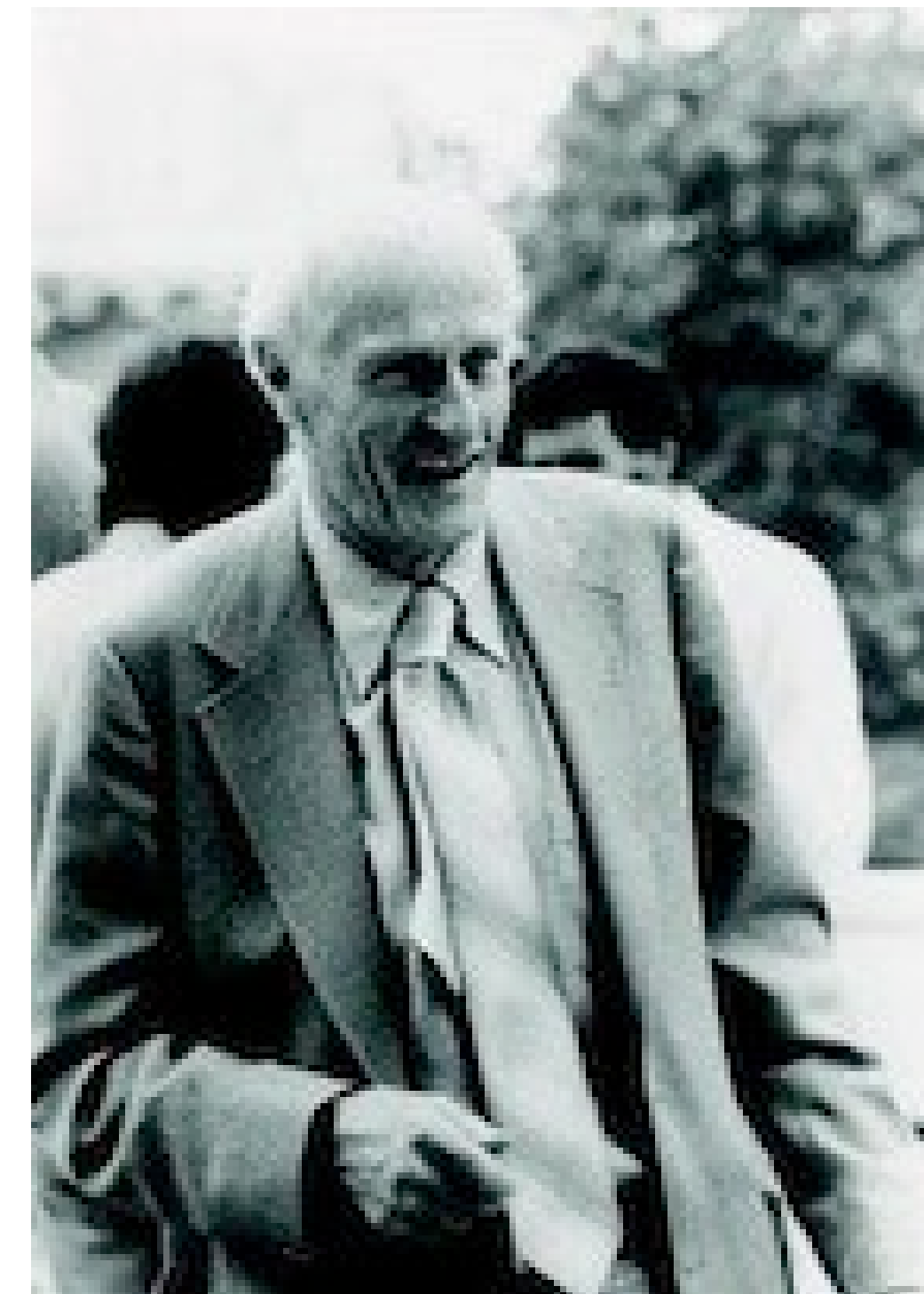
Stephen C Kleene

Kleene *,   Kleene +

Regular Expressions: Anchors ^ $

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

Example

Find me all instances of the word "the" in a text.

`the`

Misses capitalized examples

`[tT]he`

Incorrectly returns `other` **or** `theology`

`[^a-zA-Z][tT]he[^a-zA-Z]`

Incorrectly when it begins or finishes a line

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

Errors

The process we just went through was based on fixing two kinds of errors:

1. Matching strings that we should not have matched (there, then, other)
   False positives (Type I errors)

2. Not matching things that we should have matched (The)
   False negatives (Type II errors)

Errors cont.

In NLP we are always dealing with these kinds of errors.

Reducing the error rate for an application often involves two antagonistic efforts:

- ◦ Increasing accuracy or precision (minimizing false positives)
- ◦ Increasing coverage or recall (minimizing false negatives).

Substitutions

Substitution in Python and UNIX commands:

```
s/regexp1/pattern/
```

e.g.:

```
s/colour/color/
```

Capture Groups

- Say we want to put angles around all numbers:
  the 35 boxes → *the <35> boxes*

- Use parentheses () to "capture" a pattern into a numbered register (1, 2, 3…)

- Use \1 to refer to the contents of the register
  `s/([0-9]+)/<\1>/`

Capture groups: multiple registers

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

Matches

  the faster they ran, the faster we ran

But not

  the faster they ran, the faster we ate

But suppose we don't want to capture?

Parentheses have a double function: grouping terms, and capturing

Non-capturing groups: add a **?:** after the first parenthesis:

```
/(?:some|a few) (people|cats) like some \1/
```

matches
◦ `some cats like some cats`

but not
◦ `some cats like some some`

Simple Application: ELIZA

Early NLP system that imitated a Rogerian psychotherapist
- Joseph Weizenbaum, 1966.

Uses pattern matching to match, e.g.,:
- `"I need X"`

and translates them into, e.g.
- `"What would it mean to you if you got X?`

Simple Application: 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

How ELIZA works

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?/

How many words in a sentence?

"I do uh main- mainly business data processing"
- Fragments, filled pauses

"Seuss's cat in the hat is different from other cats!"
- **Lemma**: same stem, part of speech, rough word sense
  - cat and cats = same lemma
- **Wordform**: the full inflected surface form
  - cat and cats = different wordforms

How many words in a sentence?

they lay back on the San Francisco grass and looked at the stars and their

Type : an element of the vocabulary.

Token : an instance of that type in running text.

How many?
- 15 tokens (or 14)
- 13 types (or 12) (or 11)

Corpora

Words don't appear out of nowhere!

A text is produced by

- a specific writer(s),

- at a specific time,

- in a specific variety,

- of a specific language,

- for a specific function.

Text Normalization

Every NLP task requires text normalization:

1. Tokenizing (segmenting) words
2. Normalizing word formats
3. Segmenting sentences

Space-based tokenization

A very simple way to tokenize
- For languages that use space characters between words
  - Arabic, Cyrillic, Greek, Latin, etc., based writing systems
- Segment off a token between instances of spaces

Unix tools for space-based tokenization
- The "tr" command
- Given a text file, output the word tokens and their frequencies
- Remove all the numbers and punctuation.

Issues in Tokenization

Can't just blindly remove punctuation:

- m.p.h., Ph.D., AT&T, cap'n
- prices ($45.55)
- dates (01/02/06)
- URLs (http://www.stanford.edu)
- hashtags (#nlproc)
- email addresses (someone@cs.colorado.edu)

Clitic contraction: a word that doesn't stand on its own

- "are" in we're, French "je" in j'ai, "le" in l'honneur

When should multiword expressions (MWE) be words?

- New York, rock 'n' roll

Tokenization in NLTK

Tokenization needs to be run before any other language processing. A standard method is to use deterministic algorithms based on regular expressions.

```
>>> text = 'That U.S.A. poster-print costs $12.40...'
>>> pattern = r'''(?x)       # set flag to allow verbose regexps
...      ([A-Z]\.)+          # abbreviations, e.g. U.S.A.
...    | \w+(-\w+)*          # words with optional internal hyphens
...    | \$?\d+(\.\d+)?%?    # currency and percentages, e.g. $12.40, 82%
...    | \.\.\.              # ellipsis
...    | [][.,;"'?():-_`]    # these are separate tokens; includes ], [
... '''
>>> nltk.regexp_tokenize(text, pattern)
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']
```

Word Normalization

Putting words/tokens in a standard format
- ◦ U.S.A. or USA
- ◦ uhhuh or uh-huh
- ◦ Fed or fed
- ◦ am, is, be, are

Applications like information retrieval or speech recognition: reduce all letters to lower case
- ◦ Since users tend to use lower case
- ◦ Possible exception: upper case in mid-sentence?
  - ◦ e.g., **General Motors**

For sentiment analysis or machine translation
- ◦ Case is helpful (**US** versus **us** is important)

Lemmatization

Represent all words as their lemma, their shared root
= dictionary headword form:

- *am, are, is → be*

- *car, cars, car's, cars' → car*

- *He is reading detective stories*

    → He be read detective story

Lemmatization is done by Morphological Parsing

Morphemes:
- The small meaningful units that make up words
- **Stems**: The core meaning-bearing units
- **Affixes**: Parts that adhere to stems, often with grammatical functions

Morphological Parsers:
- Parse *cats* into two morphemes *cat* and *s*

Porter Stemmer:
- Based on a series of rewrite rules run in series. Some sample rules:

$$\text{ATIONAL} \rightarrow \text{ATE} \quad (\text{e.g., relational} \rightarrow \text{relate})$$

$$\text{ING} \rightarrow \epsilon \quad \text{if stem contains vowel (e.g., motoring} \rightarrow \text{motor})$$

$$\text{SSES} \rightarrow \text{SS} \quad (\text{e.g., grasses} \rightarrow \text{grass})$$

Stemming

Reduce terms to stems, chopping off affixes crudely

This was not the map we found in Billy Bones's chest, but an accurate copy, complete in all things-names and heights and soundings-with the single exception of the red crosses and the written notes.

→

Thi wa not the map we found in Billi Bone s chest but an accur copi complet in all thing name and height and sound with the singl except of the red cross and the written note .

Sentence Segmentation

!, ? mostly unambiguous but **period** "." is very ambiguous
- ◦ Sentence boundary
- ◦ Abbreviations like Inc. or Dr.
- ◦ Numbers like .02% or 4.3

Common algorithm: Tokenize first: use rules or ML to classify a period as either (a) part of the word or (b) a sentence-boundary.
- ◦ An abbreviation dictionary can help

Sentence segmentation can then often be done by rules based on this tokenization.

Lecture Overview

- Introduction

- Formal languages and grammars

- Regular expressions

- Minimum edit distance

- Natural language modelling: N-gram models

How similar are two strings?

Spell correction
- The user typed "graffe"

  Which is closest?
  - graf
  - graft
  - grail
  - giraffe

- Also for Machine Translation, Information Extraction, Speech Recognition

Edit Distance

The minimum edit distance between two strings is the minimum number of editing operations:

- Insertion
- Deletion
- Substitution

Needed to transform one into the other

Minimum Edit Distance

Two strings and their **alignment**:

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
```

Minimum Edit Distance

If each operation has cost of 1
◦ Distance between these is 5

If substitutions cost 2 (Levenshtein)
◦ Distance between them is 8

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s     i s
```

How to find the Min Edit Distance?

Searching for a path (sequence of edits) from the start string to the final string:

- ◦ **Initial state**: the word we're transforming
- ◦ **Operators**: insert, delete, substitute
- ◦ **Goal state**: the word we're trying to get to
- ◦ **Path cost**: what we want to minimize: the number of edits

intention

Del   Ins   Sub

ntention   eintention   entention
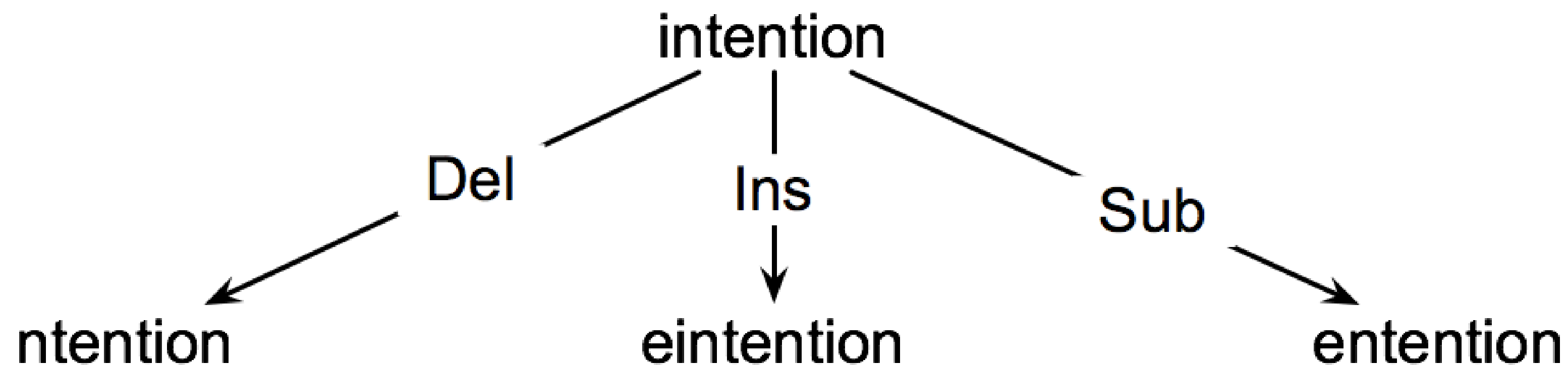
Minimum Edit as Search

But the space of all edit sequences is huge!
- We can't afford to navigate naïvely
- Lots of distinct paths wind up at the same state.
  - We don't have to keep track of all of them
  - Just the shortest path to each of those revisited states.

Defining Min Edit Distance

For two strings
- X of length n
- Y of length m

We define D(i,j)
- the edit distance between X[1..i] and Y[1..j]
  - i.e., the first i characters of X and the first j characters of Y
- The edit distance between X and Y is thus D(n,m)

Dynamic Programming for Minimum Edit Distance

Dynamic programming: A tabular computation of D(n,m)

Solving problems by combining solutions to subproblems.

Bottom-up
- ◦ We compute D(i,j) for small i,j
- ◦ And compute larger D(i,j) based on previously computed smaller values
- ◦ i.e., compute D(i,j) for all i (0 < i < n) and j (0 < j < m)

Defining Min Edit Distance (Levenshtein)

Initialization

```
D(i,0) = i
D(0,j) = j
```

Recurrence Relation:

```
For each  i = 1…M
      For each  j = 1…N
```

$$
D(i,j) = \min \begin{cases} D(i-1,j) + 1 & \text{(deletion)} \\ D(i,j-1) + 1 & \text{(insertion)} \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } X(i) \neq Y(j) \\ 0; & \text{if } X(i) = Y(j) \end{cases} \end{cases}
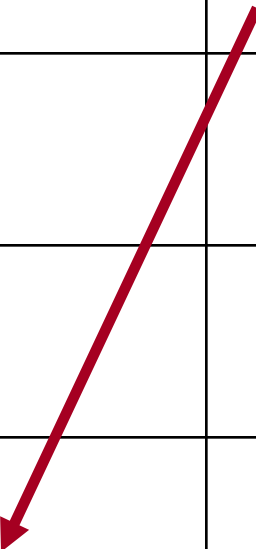$$

Termination:

```
D(N,M) is distance
```

The Edit Distance Table

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

The Edit Distance Table

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

The Edit Distance Table

N	9	8	9	10	11	12	11	10	9	**8**
O	8	7	8	9	10	11	10	9	8	9
I	7	6	7	8	9	10	9	8	9	10
T	6	5	6	7	8	9	8	9	10	11
N	5	4	5	6	7	8	9	10	11	10
E	4	3	4	5	6	7	8	9	10	9
T	3	4	5	6	7	8	7	8	9	8
N	2	3	4	5	6	7	8	7	8	7
I	1	2	3	4	5	6	7	6	7	8
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

Lecture Overview

- Introduction

- Formal languages and grammars

- Regular expressions

- Minimum edit distance

- Natural language modelling: N-gram models

Probabilistic Language Models

Goal: assign a probability to a sentence

- ◦ Machine Translation:
  - ◦ P(**high** winds tonite) > P(**large** winds tonite)
- ◦ Spell Correction
  - ◦ The office is about fifteen **minuets** from my house
    - ◦ P(about fifteen **minutes** from) > P(about fifteen **minuets** from)
- ◦ Speech Recognition
  - ◦ P(I saw a van) >> P(eyes awe of an)

Why?

Probabilistic Language Modeling

Goal: compute the probability of a sentence or sequence of words:

$$P(W) = P(w_1,w_2,w_3,w_4,w_5...w_n)$$

Related task: probability of an upcoming word:

$$P(w_5|w_1,w_2,w_3,w_4)$$

A model that computes either of these:

$P(W)$ or $P(w_n|w_1,w_2...w_{n-1})$ is called a **language model**.

Better: **the grammar** But **language model** or **LM** is standard

How to compute P(W)

How to compute this joint probability:
- ◦ P(its, water, is, so, transparent, that)

Using conditional probabilities:
p(B|A) = P(A,B)/P(A)   Rewriting: **P(A,B) = P(A)P(B|A)**

More variables:
$$P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)$$

The Chain Rule in General

$$P(x_1,x_2,x_3,...,x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1,x_2)...P(x_n|x_1,...,x_{n-1})$$

The Chain Rule applied to compute joint probability of words in sentence

$$P(w_1 w_2 \cdots w_n) = \prod_i P(w_i \mid w_1 w_2 \cdots w_{i-1})$$

P("its water is so transparent") =

P(its) × P(water|its) × P(is|its water)

   × P(so|its water is) × P(transparent|its water is so)

How to estimate these probabilities

Could we just count and divide?

$$P(\text{the} \mid \text{its water is so transparent that}) =$$

$$\frac{Count(\text{its water is so transparent that the})}{Count(\text{its water is so transparent that})}$$

No! Too many possible sentences!

We'll never see enough data for estimating these

Markov Assumption

Andrei Markov

Simplifying assumption:

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{that})$$

Or maybe

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{transparent that})$$

Markov Assumption

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i \mid w_{i-k} \ldots w_{i-1})$$

In other words, we approximate each component in the product

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-k} \ldots w_{i-1})$$

Simplest case: Unigram model

$$P(w_1 w_2 \cdots w_n) \approx \prod_i P(w_i)$$

Some automatically generated sentences from a unigram model

```
fifth, an, of, futures, the, an, incorporated, a, a,
the, inflation, most, dollars, quarter, in, is, mass

thrift, did, eighty, said, hard, 'm, july, bullish

that, or, limited, the
```

Bigram model

- Condition on the previous word:

$$P(w_i \mid w_1 w_2 \square \ w_{i-1}) \approx P(w_i \mid w_{i-1})$$

```
texaco, rose, one, in, this, issue, is, pursuing, growth, in,
a, boiler, house, said, mr., gurria, mexico, 's, motion,
control, proposal, without, permission, from, five, hundred,
fifty, five, yen
```

```
outside, new, car, parking, lot, of, the, agreement, reached
```

```
this, would, be, a, record, november
```

N-gram models

We can extend to trigrams, 4-grams, 5-grams

In general this is an insufficient model of language
  ◦ because language has **long-distance dependencies**:

"The computer which I had just put into the machine room on the fifth floor crashed."

But we can often get away with N-gram models

Approximating Shakespeare

|	
1 gram	–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
2 gram	–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.
3 gram	–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.

References

- Jurafsky, D. & Martin, J. H. Speech and Language Processing. Stanford. 2023. Chapters 2 and 3.

- Russell, S.J. & Norvig, P. Artificial Intelligence: A Modern Approach. Fourth Edition, Pearson Education, Hoboken, NJ, 2021. Chapters 22 and 23.
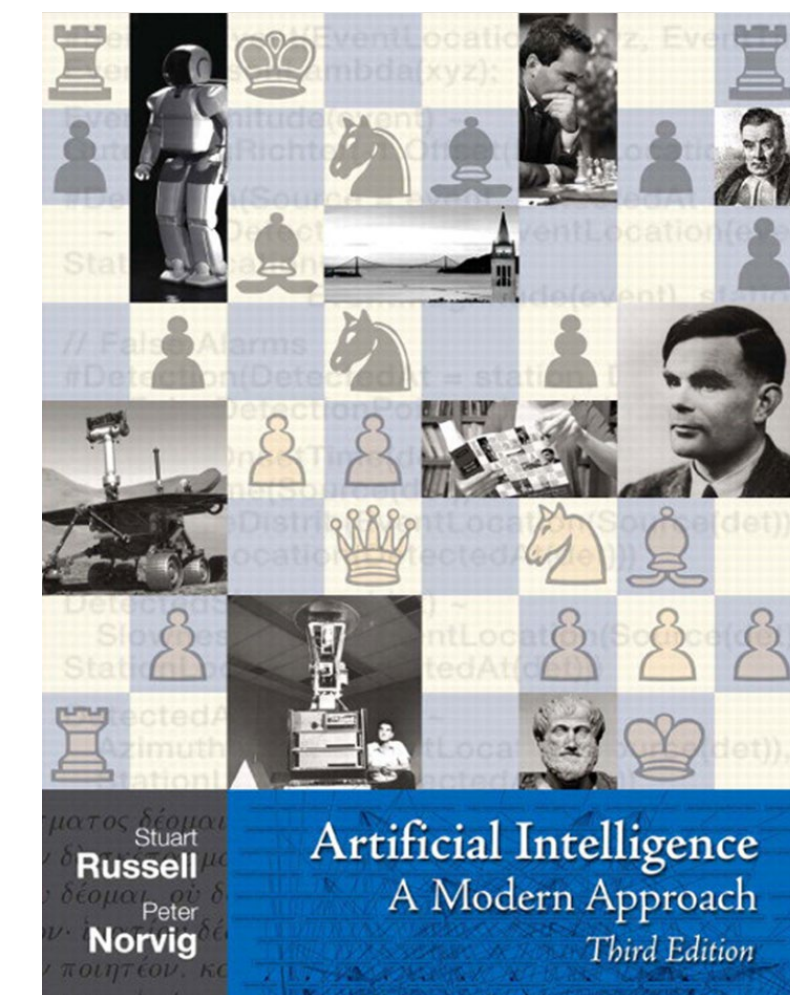
Speech and Language Processing

An Introduction to Natural Language Processing,
Computational Linguistics, and Speech Recognition

Third Edition draft

Daniel Jurafsky
Stanford University

James H. Martin
University of Colorado at Boulder

Copyright ©2023. All rights reserved.

Draft of January 7, 2023. Comments and typos welcome!

Artificial Intelligence
A Modern Approach
Third Edition

Stuart **Russell**
Peter **Norvig**

Feedback

- In case you want to provide anonymous feedback on these lectures, please visit:

- https://forms.gle/KBkN744QuffuAZLF8

Muchas gracias!

AI Lecture Feedback

This is a short form to provide early feedback for lectures

franciscocruzhh@gmail.com Switch account

Not shared

* Indicates required question

In case you want a reply, provide your zID. Otherwise your answer is anonymous.

Your answer

how did you participate? *

○ In the classroom

○ Watch the class from automatic recording

If you have any comments, feedback, or question about the lectures, this is the place.

Your answer

Submit   Clear form