RAMAIAH
Institute of Technology

M.S. Ramaiah Institute of Technology
(Autonomous Institute, Affiliated to VTU)
Department of Computer Science and Engineering

Course Name: Artificial Intelligence(AI)
Course Code: CS53
Credits: 3:0:1
UNIT -4
Term: October 2021 – Feb 2022

RAMAIAH
Institute of Technology

Outline

- Natural Language Processing (NLP) - Introduction
- Language Models
  - What are N-gram models?
- Text Classification
- Information Retrieval
- Information Extraction

RAMAIAH
Institute of Technology

Natural Language Processing - Introduction

- In a recent survey (KDNuggets blog) of data scientists, 62% reported working "mostly or entirely" with data about people. Much of this data is text.
- Natural Languages – Kannada, Hindi, English, …..
- NLP is a central part of mining large datasets.
- Sentiment Analysis – Application of NLP. Along with NLP, it also uses Machine Learning Methods.

Natural Language Processing

- Some basic terms:

Syntax: the allowable structures in the language: sentences, phrases, affixes (-ing, -ed, -ment, etc.).

Semantics: the meaning(s) of texts in the language.

Part-of-Speech (POS): the category of a word (noun, verb, preposition etc.).

Example: "They permitted us."

('They', 'PRP'), ('permitted', 'VBD'), ('us', 'PRP')

Bag-of-words (BoW): a featurization that uses a vector of word counts (or binary) ignoring order.

Example: Positive Verb's BOW=> ['nice', 'enjoy', ' superb', ……..]

N-gram: for a fixed, small N (2-5 is common), an n-gram is a consecutive sequence of words in a text.

Bag of words Featurization

- Assuming we have a dictionary mapping words to a unique integer id, a bag-of-words featurization of a sentence could look like this:

Sentence:	The	cat	sat	on	the	mat
word id's:	1	12	5	3	1	14

- The BoW featurization would be the vector:

Vector	2, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1 ……
position	1 3 5 12 14 ……

- In practice this would be stored as a sparse vector of (id, count)s:

  [(1,2),(3,1),(5,1),(12,1),(14,1)]

- Note that the original word order is lost, replaced by the order of id's.

N-grams

- **Because word order is lost, the sentence meaning is weakened.** This sentence has quite a different meaning but the same BoW vector:

Sentence:	The	mat	sat	on	the	cat
word id s:	1	14	5	3	1	12

- BoW featurization:

Vector 2, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1

- But word order is important, especially the order of nearby words.
- N-grams capture this, by modeling tuples of consecutive words.

N-grams

Sentence: The cat sat on the mat

2-grams: the-cat, cat-sat, sat-on, on-the, the-mat

Notice how even these short n-grams “make sense” as linguistic units. For the other sentence we would have different features:

Sentence: The mat sat on the cat

2-grams: the-mat, mat-sat, sat-on, on-the, the-cat

We can go still further and construct 3-grams:

Sentence: The cat sat on the mat

3-grams: the-cat-sat, cat-sat-on, sat-on-the, on-the-mat

Which capture still more of the meaning:

Sentence: The mat sat on the cat

3-grams: the-mat-sat, mat-sat-on, sat-on-the, on-the-cat

Language model

- Probability distributions over sentences (i.e., word sequences)

$$P(W) = P(w_1 w_2 w_3 w_4 \ldots w_k)$$

- Can use them to generate strings

$$P(w_k \mid w_2 w_3 w_4 \ldots w_{k-1})$$

- Rank possible sentences
  - P("Today is Tuesday") > P("Tuesday Today is")
  - P("Today is Tuesday") > P("Today is Virginia")

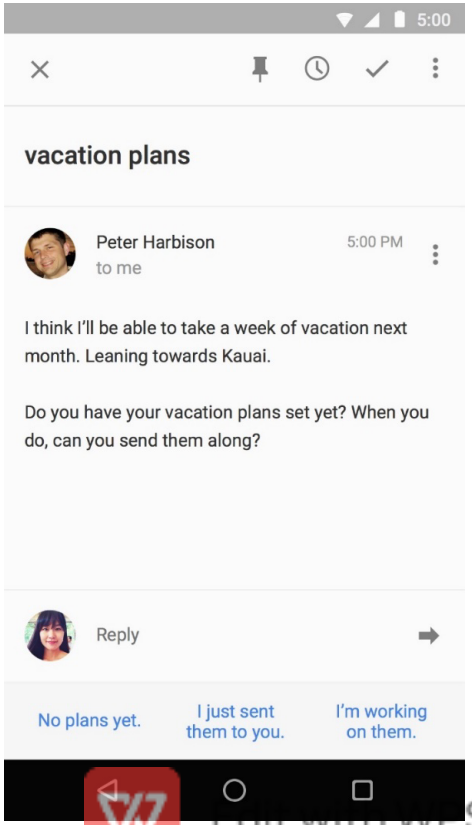
Language model applications

- Autocomplete

RAMAIAH
Institute of Technology

Language model applications

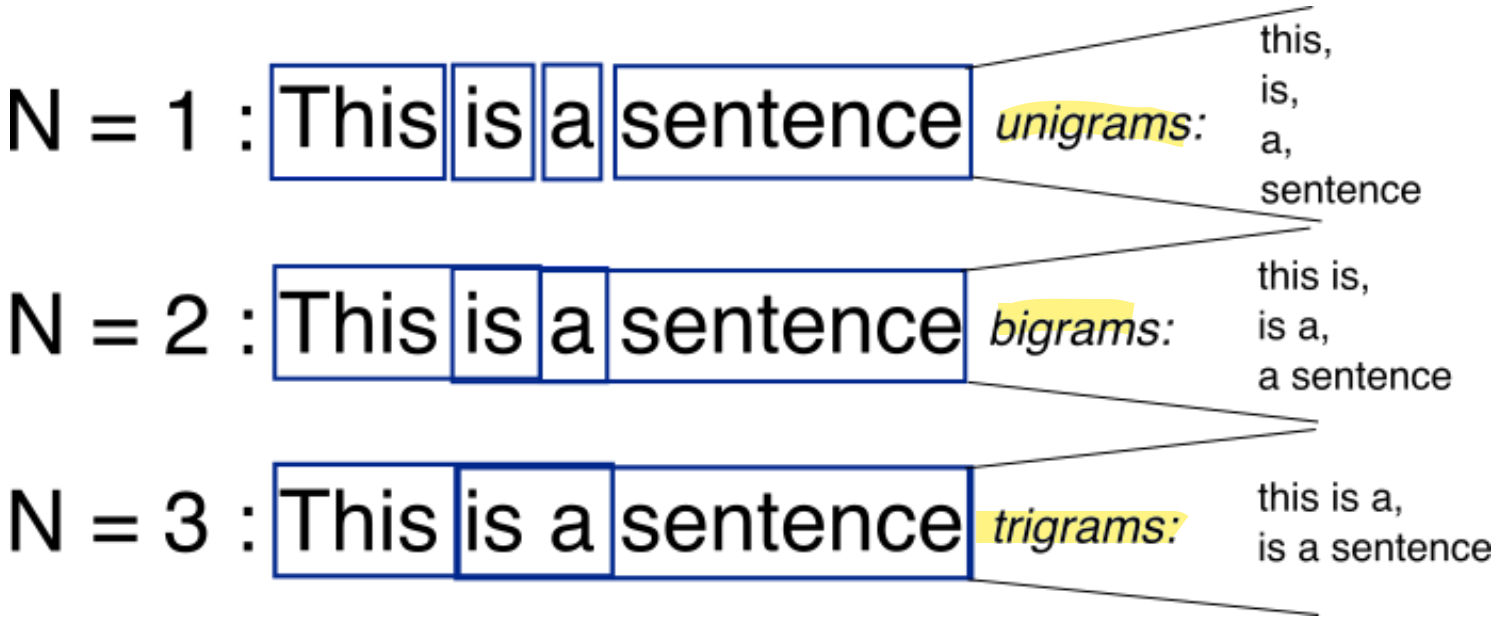
- Smart Reply



RAMAIAH
Institute of Technology

Bag-of-Words with N-grams

- N-grams: a contiguous sequence of n tokens from a given piece of text



N-Gram Models

- N-grams: a contiguous sequence of n tokens from a given piece of text

Unigram model: $P(w_1)P(w_2)P(w_3) \dots P(w_n)$

Bigram model: $P(w_1)P(w_2|w_1)P(w_3|w_2) \dots P(w_n|w_{n-1})$

Trigram model:

$P(w_1)P(w_2|w_1)P(w_3|w_2,w_1) \dots P(w_n|w_{n-1}w_{n-2})$

N-gram model:

$P(w_1)P(w_2|w_1) \dots P(w_n|w_{n-1}w_{n-2} \dots w_{n-N})$

Sampling words with replacement

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, 'and what is the use of a book,' thought Alice 'without pictures or conversation?'

P(of) = 3/66
P(Alice) = 2/66
P(was) = 2/66
P(to) = 2/66
P(her) = 2/66
P(sister) = 2/66
P(,) = 4/66
P(') = 4/66

Conditional on the previous word

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, 'and what is the use of a book,' thought Alice 'without pictures or conversation?'

$P(w_{i+1}= \text{of} \mid w_i=\text{tired}) = 1$
$P(w_{i+1}= \text{of} \mid w_i=\text{use}) = 1$
$P(w_{i+1}= \text{sister} \mid w_i=\text{her}) = 1$
$P(w_{i+1}= \text{beginning} \mid w_i=\text{was}) = 1/2$
$P(w_{i+1}= \text{reading} \mid w_i=\text{was}) = 1/2$

$P(w_{i+1}= \text{bank} \mid w_i=\text{the}) = 1/3$
$P(w_{i+1}= \text{book} \mid w_i=\text{the}) = 1/3$
$P(w_{i+1}= \text{use} \mid w_i=\text{the}) = 1/3$

The Chain Rule

- The joint probability can be expressed in terms of the conditional probability:

$$P(X,Y) = P(X \mid Y)\, P(Y)$$

- With more variables:

$$P(X, Y, Z) = P(X \mid Y, Z)\, P(Y, Z)$$
$$= P(X \mid Y, Z)\, P(Y \mid Z)\, P(Z)$$

$$P(X_1, X_2, \ldots X_n)$$
$$= P(X_1)P(X_2|X_1)P(X_3|X_2, X_1) \ldots P(X_n \mid X_1, \ldots, X_{n-1})$$
$$= P(X_1)\Pi_{i=2}^{n}(X_i \mid X_1, \ldots, X_{i-1})$$

Language model for text

- Probability distribution over sentences
  - $p(w_1\, w_2\, \ldots\, w_n) = p(w_1)p(w_2|w_1)p(w_3|w_1, w_2) \ldots p(w_n|w_1, w_2, \ldots, w_{n-1})$
  - Complexity - $O(V^{n^*})$
    - n^* - maximum sentence length

- Rank possible sentences
  - P("Today is Tuesday") > P("Tuesday Today is")
  - P("Today is Tuesday") > P("Today is Virginia")

Language model with N-gram

- The chain rule:

$$P(X_1, X_2, \ldots X_n) = P(X_1)P(X_2|X_1)P(X_3|X_2, X_1) \ldots P(X_n \mid X_1, \ldots X_{n-1})$$

- N-gram language model assumes each word depends only on the last n-1 words (Markov assumption)
- Example: trigram (3-gram)

$P(\text{"}Today\ is\ a\ sunny\ day\text{"})$
=P("Today")P("is"|"Today")P("a"|"is", "Today") … P("day"|"sunny", "a")

RAMAIAH
Institute of Technology

Approximation (Through Experiments)

Unigram model:

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

Bigram model:

- Condition on the previous word

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

Approximation

N-gram model:

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

Text Classification

- Text classification is also known as categorization: given a text of some kind, decide which of a predefined set of classes it belongs to.
- Language identification and genre classification are examples of text classification, as is **sentiment analysis** (classifying a movie or product review as positive or negative) and **spam detection** (classifying an email message as spam or ham).
- **Sentiment Analysis** is the process of determining whether a piece of writing is positive, negative or neutral.

  E.g. 1: Amit is happy. E.g. 2: Akshay is frustrated.
- We can treat spam detection as a problem in supervised learning.
- A training set is readily available: the positive (spam) examples are in our spam folder, the negative (ham) examples are in our inbox.

Text Classification

- Here is an examples:

spam: Wholesale Fashion Watches – 57% today. Designer watches for cheap …

spam: WE CAN TREAT ANYTHING YOU SUFFER FROM JUST TRUST US …

ham: Good to see you my friend. Hey Peter, It was good to hear from you …

ham: Abstract: We will motivate the problem of social identity clustering: …

Text Classification Approaches

- There are two ways in text classification
  - Language Modeling Approach
  - Machine Learning Approach

Language Modeling Approach

- We define one n-gram language model for **P(Message|spam)** by training on the spam folder, and one model for **P(Message|ham)** by training in the inbox.
- We can classify a new message with an application of Bayes' rule:

$$\underset{c \in \{spam, ham\}}{\operatorname{argmax}} P(c \mid message) = \underset{c \in \{spam, ham\}}{\operatorname{argmax}} P(message \mid c) P(c) .$$

where P(c) is estimated just by counting the total number of spam and ham messages. This approach works well for spam detection, just as it did for language identification.

Machine Learning Approach

- In the machine-learning approach, we represent the message as a set of feature/value pairs and apply a classification algorithm h to the feature vector X.

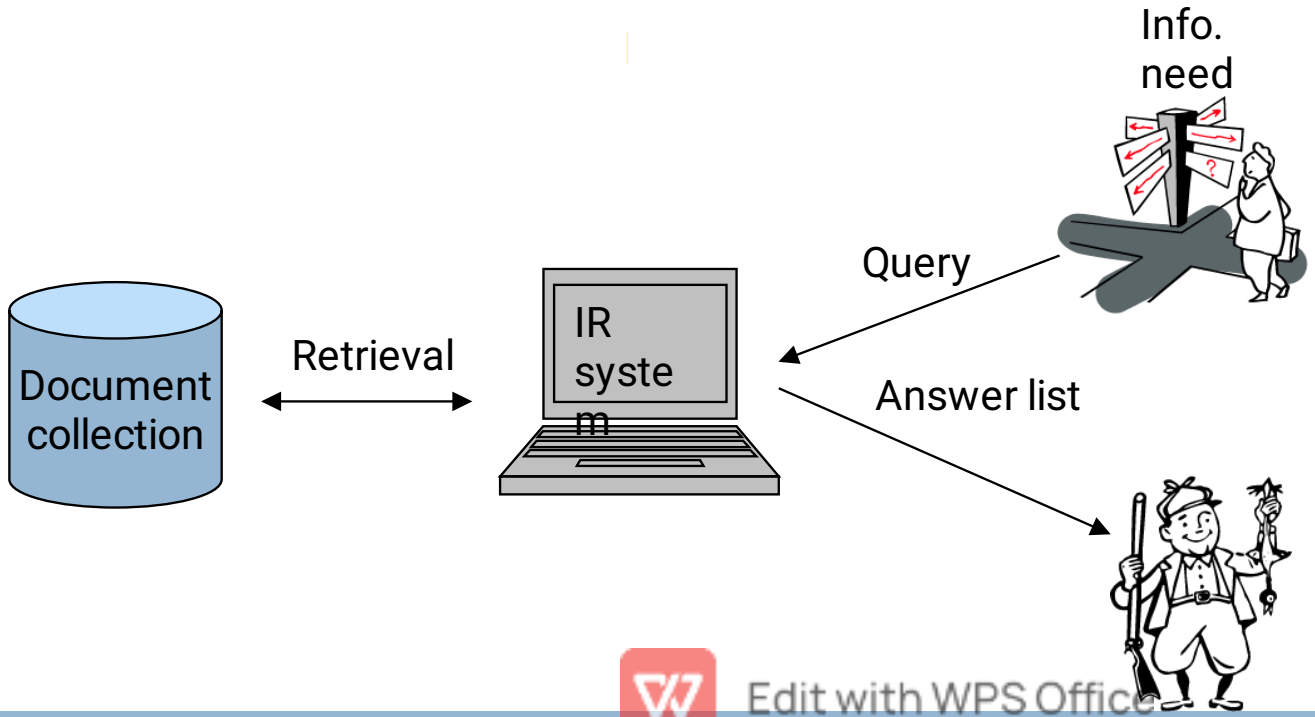
- We can make the language-modeling and machine-learning approaches compatible by thinking of the n-grams as features.

Information Retrieval

- Information retrieval is the task of finding documents that are relevant to a user's need for information.
- Popular examples of information retrieval systems are search engines on the World Wide Web.
- Information Retrieval system can be characterized by:
  - A corpus of documents: Each system must decide what it wants to treat as a document: a paragraph, a page, or a multipage text.
  - Queries posed in a query language
  - A result set
  - A presentation of the result set

The problem of IR

- Goal = find documents *relevant* to an information need from a large document set



Term frequency weight

- The log frequency weight of term t in d is defined as follows

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d} & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

idf weight

- The document frequency dft is defined as the number of documents that t occurs in.
- We define the idf weight of term t as follows:

$$\text{idf}_t = \log_{10} \frac{N}{\text{df}_t}$$

- idf is a measure of the informativeness of the term.

tf-idf weight

- The tf-idf weight of a term is the product of its tf weight and its idf weight.

$$w_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \cdot \log \frac{N}{\mathrm{df}_t}$$

Why is ranking so important?

- Problems with unranked retrieval
  - Users want to look at a few results – not thousands.
  - It's very hard to write queries that produce a few results.
  - Even for expert searchers
  - Ranking is important because it effectively reduces a large set of results to a very small one.

- Actually, in the vast majority of cases people only examine 1, 2, or 3 results.

PageRank Algorithm

- PageRank was invented to solve the problem of the tyranny of TF scores.
- If the query is [IBM], how do we make sure that IBM's home page, ibm.com, is the first result, even if another page mentions the term "IBM" more frequently?
- The idea is that **ibm.com** has many **in-links** (links to the page), so it should be linked higher: each in-link is a **vote for the quality of the linked-to page**.
- But if we only counted in-links, then it would be possible for a web spammer to create a network of pages and have them all point to a page of his choosing, increasing the score of that page.
- Therefore, the PageRank algorithm is designed to weight links from high-quality sites more heavily.
- High-Quality Site: One that is linked to by other high-quality sites.

PageRank Algorithm

- The definition is recursive, but we will see that the recursion bottoms out properly.
- The PageRank for a page p is defined as:

$$PR(p) = \frac{1-d}{N} + d\sum_i \frac{PR(in_i)}{C(in_i)},$$

- PR(p) is the PageRank of page p
- N is the total number of pages in the corpus
- in_i are the pages that link into p
- C(in_i) is the count of the total number of out-links on page in_i.
- The constant d is a **damping factor**. It can be understood through the **random surfer model**: imagine a Web surfer who starts at some random page and begins exploring. With probability d the surfer clicks on one of the links on the page and with probability 1-d he/she gets bored with the page and restarts on a random page anywhere on the web.

PageRank Algorithm

- The PageRank of page p is then the probability that the random surfer will be at p at any point in time.
- PageRank can be computed by an iterative procedure: start with all pages having PR(p)=1, and iterate algorithm, updating ranks until they converge.

HITS Algorithm

- The Hyperlink-Induced Topic Search algorithm, also known as “Hubs and Authorities” or HITS, is another influential link-analysis algorithm.
- HITS differs from PageRank in several ways.
- First, it is query dependent measure: it rates pages with respect to a query.
- Given a query, HITS first finds a set of pages that are relevant to the query.
- Each page in this set is considered an authority on the query to the degree that other pages in the relevant set point to it.
- A page is considered a hub to the degree that it points to other authoritative pages in the relevant set.
- Just as with PageRank, we don’t want to merely count the number of links; we want to give more value to the high-quality hubs and authorities.

HITS Algorithm

- As with PageRank, we iterate a process that updates the authority score of a page to be the sum of the hub scores of the page that point to it, and the hub score to be the sum of the authority scores of the pages it points to.
- If we then normalize the scores and repeat k times, the process will converge.

RAMAIAH
Institute of Technology

HITS Algorithm

```
function HITS(query) returns pages with hub and authority numbers
  pages ← EXPAND-PAGES(RELEVANT-PAGES(query))
  for each p in pages do
    p.AUTHORITY ← 1
    p.HUB ← 1
  repeat until convergence do
    for each p in pages do
      p.AUTHORITY ← Σ_i INLINK_i(p).HUB
      p.HUB ← Σ_i OUTLINK_i(p).AUTHORITY
    NORMALIZE(pages)
  return pages
```

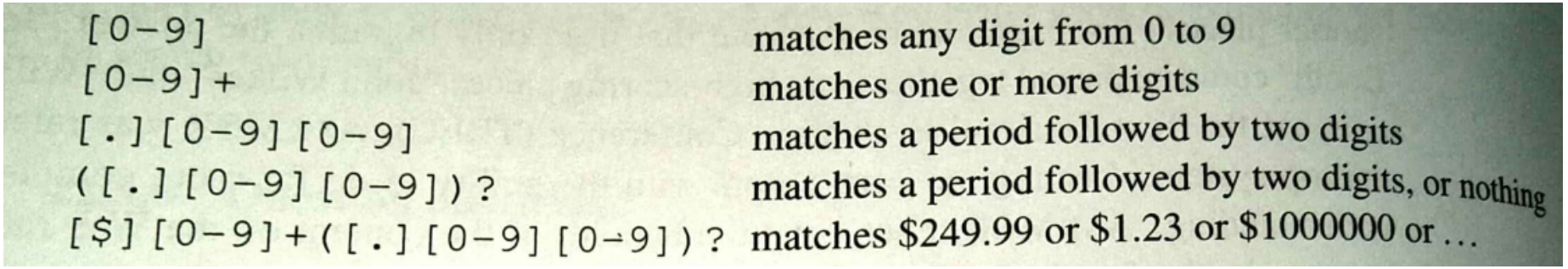
Figure 22.1 The HITS algorithm for computing hubs and authorities with respect to a query. RELEVANT-PAGES fetches the pages that match the query, and EXPAND-PAGES adds in every page that links to or is linked from one of the relevant pages. NORMALIZE divides each page's score by the sum of the squares of all pages' scores (separately for both the authority and hubs scores).

Information Extraction

- Information extraction is the process of acquiring knowledge by skimming a text and looking for occurrences of a particular class of object and for relationships among objects.
- The simplest type of information extraction system is an **attribute-based extraction system** that assumes that the entire text refers to a single object and the task is to extract attributes of that object.
- Consider the example of extracting information from the text “IBM ThinkBook970. Our price: $399.00” . The set of attributes include {Manufacturer=IBM, Model=ThinkBook970, Price=$399.00}.
- We can address this problem by defining a template (also known as pattern) for each attribute we would like to extract.
- The template is defined by a finite state automaton, the simplest example of which is the regular expression.

Information Extraction

- The template is defined by a finite state automaton, the simplest example of which is the regular expression.



`[0-9]`	matches any digit from 0 to 9
`[0-9]+`	matches one or more digits
`[.][0-9][0-9]`	matches a period followed by two digits
`([.][0-9][0-9])?`	matches a period followed by two digits, or nothing
`[$][0-9]+([.][0-9][0-9])?`	matches $249.99 or $1.23 or $1000000 or ...

- Templates are often defined with three parts: a prefix regex, a target regex, and a postfix regex.
- For prices, the **target regex** is as shown above, the prefix would look for strings such as “price:” and the postfix could be empty.

Relational Extraction Systems

- One step up from the attribute-based extraction systems are relational extraction systems, which deal with multiple objects and the relations among them.
- When these systems see the text “$249.99”, they need to determine not just that it is a price, but also which object has that price.
- A typical relational-based extraction system is FASTUS, which handles news stories about corporate mergers and acquisitions.
- It can read the story

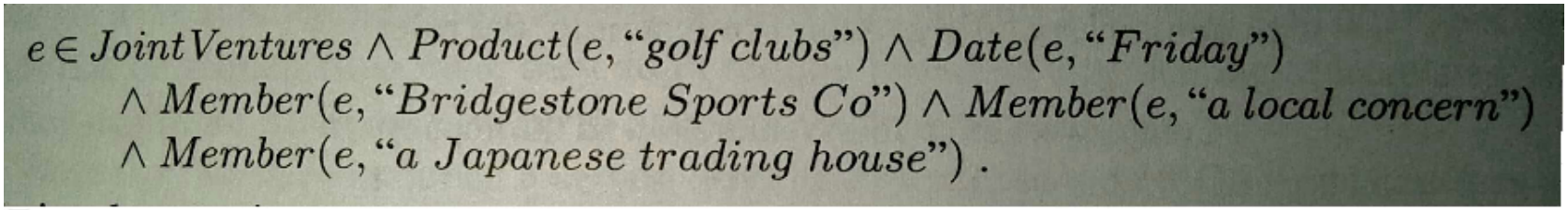
Bridgestone Sports Co. said Friday it has set up a joint venture in Taiwan with a local concern and a Japanese trading house to produce golf clubs to be shipped to Japan.

Relational Extraction Systems

- It can read the story

Bridgestone Sports Co. said Friday it has set up a joint venture in Taiwan with a local concern and a Japanese trading house to produce golf clubs to be shipped to Japan.

And extract the relations:



Relational Extraction Systems

- A relational extraction system can be built as a series of **cascaded finite-state transducers**.
- That is, the system consists of a series of small, efficient finite-state automata (FSAs), where each automaton receives text as input, transduces the text into a different format, and passes it along to the next automaton.
- FASTUS consists of 5 stages:
  1. Tokenization
  2. Complex-word handling
  3. Basic-group handling
  4. Complex-phrase handling
  5. Structure merging

Relational Extraction Systems

- **FASTUS's first stage is tokenization**, which segments the stream of characters into tokens (words, numbers, and punctuation).
- For English, tokenization can be fairly simple; just separating characters at white space or punctuation does a fairly good job.
- Some tokenizers also deal with markup languages such as HTML, SGML, and XML.
- **The second stage handles complex words**, including collocations such as "set up" and "joint venture" as well as proper names such as "Bridgestone Sports Co."
- These are recognized by a combination of lexical entries and finite-state grammar rules. For example, a company name might be recognized by the rule

  CapitalizedWord+("Company" | "Co"| "Inc" | "Ltd")

Relational Extraction Systems

- The third stage handles basic groups, meaning noun groups and verb groups. The idea is to chunk these into units that will be managed by the later stages.
- Example:

Bridgestone Sports Co. said Friday it has set up a joint venture in Taiwan with a local concern and a Japanese trading house to produce golf clubs to be shipped to Japan.

- The above example sentence would emerge from this stage as the following sequence of tagged groups.

1. **NG: Bridgestone Sports Co.**
2. **VG: said**

Relational Extraction Systems

3. NG: Friday
4. NG: it
5. VG: had set up
6. NG: a joint venture
7. PR: in
8. NG: Taiwan
9. PR: with
10. NG: a local concern
11. CJ: and
12. NG: a Japanese trading house
13. VG: to produce
14. NG: golf clubs
15. VG: to be shipped
16. PR: to
17. NG: Japan

Here NG means noun group, VG is verb group, PR is preposition, and CJ is conjunction

Relational Extraction Systems

- The fourth stage combines the basic groups into complex phrases.
- One type of combination of rule deals with domain-specific events.
- For example, the rule

  Company+ SetUp JointVenture (“with” Company+)?

Describes the formation of a Joint Venture.

- This stage is the first one in the cascade where the output is placed into a database template as well as being placed in the output stream.
- The final stage merge structures that were built up in the previous step.

Outline

Natural Language Communication

- Phrase Structure Grammars
- Syntactic Analysis
- Augmented Grammars and Semantic Interpretation
- Machine Translation
- Speech Recognition

RAMAIAH
Institute of Technology

Natural Language for Communication

- Communication is the intentional exchange of information.

Phrase Structure Grammars:

- The n-gram models is based on sequence of words. The big issue for these models is data sparsity – which a vocabulary of, say, 10^5 words, there are 10^15 trigram probabilities to estimate, and so a corpus of even a trillion words will not be able to supply reliable estimates for all of them.
- We can address the problem of sparsity through generalization.
- From the fact that **“black dog”** is more frequent than **“dog black”** and similar observations, we can from the generalization that adjectives tend to come before nouns in English.
- We can combine syntactic categories such as **noun phrase** or **verb phrase**, and combine these categories into **trees** representing the **phrase structure of sentences**.

Phrase Structure Grammars

- There have been many competing language models based on the idea of phrase structure, we describe a popular model called the probabilistic context-free grammar or PCFG.
- A grammar is a collection of rules that defines a language as a set of strings of words.
- Probabilistic means that the grammar assigns a probability to every string.
- Here is a PCFG rule:

  VP → Verb [0.70]

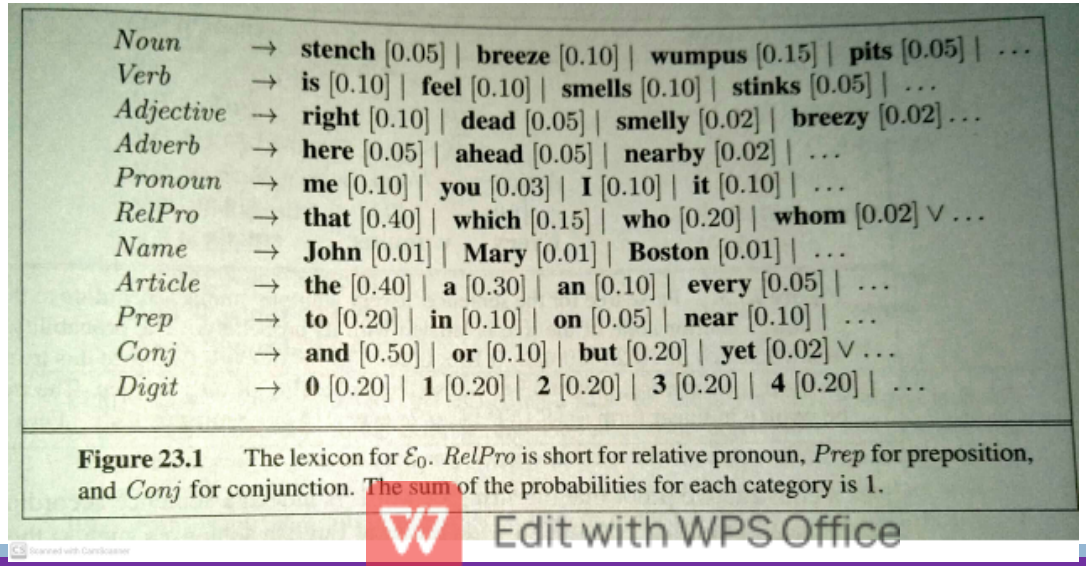
  | VP NP [0.30]

- Here VP (verb phrase) and NP (noun phrase) are non-terminal symbols.
- The grammar also refers to the actual words, which are called terminal symbols.
- This rule is saying that with probability 0.70 a verb phrase consists solely of a verb, and with probability 0.30 it is a VP followed by an NP.

RAMAIAH
Institute of Technology

The lexicon of ε_0

- We now define a grammar for a tiny fragment of English that is suitable for communication between agents exploring the wumpus world. We call this language ε_0
- Later sections improve on ε_0 to make it slightly closer to real English.



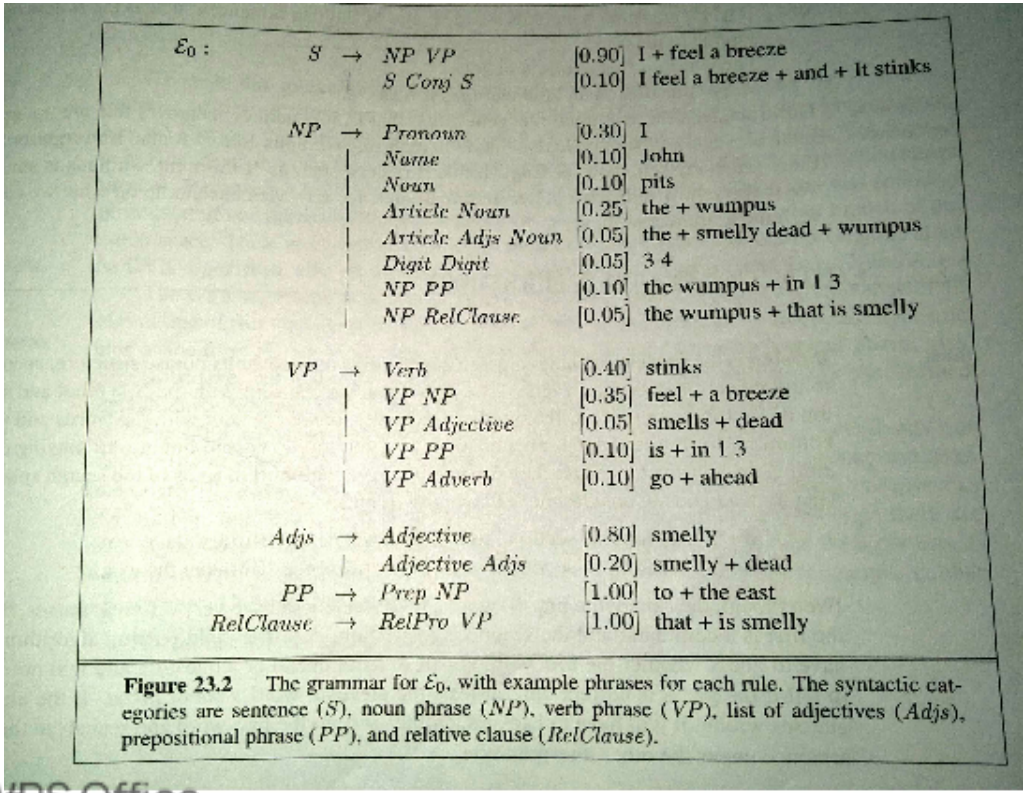
Noun	→	**stench** [0.05] | **breeze** [0.10] | **wumpus** [0.15] | **pits** [0.05] | ...
Verb	→	**is** [0.10] | **feel** [0.10] | **smells** [0.10] | **stinks** [0.05] | ...
Adjective	→	**right** [0.10] | **dead** [0.05] | **smelly** [0.02] | **breezy** [0.02] ...
Adverb	→	**here** [0.05] | **ahead** [0.05] | **nearby** [0.02] | ...
Pronoun	→	**me** [0.10] | **you** [0.03] | **I** [0.10] | **it** [0.10] | ...
RelPro	→	**that** [0.40] | **which** [0.15] | **who** [0.20] | **whom** [0.02] ∨ ...
Name	→	**John** [0.01] | **Mary** [0.01] | **Boston** [0.01] | ...
Article	→	**the** [0.40] | **a** [0.30] | **an** [0.10] | **every** [0.05] | ...
Prep	→	**to** [0.20] | **in** [0.10] | **on** [0.05] | **near** [0.10] | ...
Conj	→	**and** [0.50] | **or** [0.10] | **but** [0.20] | **yet** [0.02] ∨ ...
Digit	→	**0** [0.20] | **1** [0.20] | **2** [0.20] | **3** [0.20] | **4** [0.20] | ...

Figure 23.1 The lexicon for $\mathcal{E}_0$. *RelPro* is short for relative pronoun, *Prep* for preposition, and *Conj* for conjunction. The sum of the probabilities for each category is 1.

RAMAIAH
Institute of Technology

The Grammar of ε_0

- This step is to combine the words into phrases.
- Figure shows the grammar with rules for each of the six syntactic categories and an example for each rewrite rule.



$\mathcal{E}_0$:	S	→ $NP\ VP$	[0.90]	I + feel a breeze
		| $S\ Conj\ S$	[0.10]	I feel a breeze + and + It stinks
	NP	→ $Pronoun$	[0.30]	I
		| $Name$	[0.10]	John
		| $Noun$	[0.10]	pits
		| $Article\ Noun$	[0.25]	the + wumpus
		| $Article\ Adjs\ Noun$	[0.05]	the + smelly dead + wumpus
		| $Digit\ Digit$	[0.05]	3 4
		| $NP\ PP$	[0.10]	the wumpus + in 1 3
		| $NP\ RelClause$	[0.05]	the wumpus + that is smelly
	VP	→ $Verb$	[0.40]	stinks
		| $VP\ NP$	[0.35]	feel + a breeze
		| $VP\ Adjective$	[0.05]	smells + dead
		| $VP\ PP$	[0.10]	is + in 1 3
		| $VP\ Adverb$	[0.10]	go + ahead
	$Adjs$	→ $Adjective$	[0.80]	smelly
		| $Adjective\ Adjs$	[0.20]	smelly + dead
	PP	→ $Prep\ NP$	[1.00]	to + the east
	$RelClause$	→ $RelPro\ VP$	[1.00]	that + is smelly

Figure 23.2 The grammar for $\mathcal{E}_0$, with example phrases for each rule. The syntactic categories are sentence (S), noun phrase (NP), verb phrase (VP), list of adjectives ($Adjs$), prepositional phrase (PP), and relative clause ($RelClause$).

RAMAIAH
Institute of Technology

The Lexicon and Grammar of ε_0

Noun	→	**stench** [0.05] | **breeze** [0.10] | **wumpus** [0.15] | **pits** [0.05] | ...
Verb	→	**is** [0.10] | **feel** [0.10] | **smells** [0.10] | **stinks** [0.05] | ...
Adjective	→	**right** [0.10] | **dead** [0.05] | **smelly** [0.02] | **breezy** [0.02] ...
Adverb	→	**here** [0.05] | **ahead** [0.05] | **nearby** [0.02] | ...
Pronoun	→	**me** [0.10] | **you** [0.03] | **I** [0.10] | **it** [0.10] | ...
RelPro	→	**that** [0.40] | **which** [0.15] | **who** [0.20] | **whom** [0.02] ∨ ...
Name	→	**John** [0.01] | **Mary** [0.01] | **Boston** [0.01] | ...
Article	→	**the** [0.40] | **a** [0.30] | **an** [0.10] | **every** [0.05] | ...
Prep	→	**to** [0.20] | **in** [0.10] | **on** [0.05] | **near** [0.10] | ...
Conj	→	**and** [0.50] | **or** [0.10] | **but** [0.20] | **yet** [0.02] ∨ ...
Digit	→	**0** [0.20] | **1** [0.20] | **2** [0.20] | **3** [0.20] | **4** [0.20] | ...

Figure 23.1 The lexicon for $\mathcal{E}_0$. *RelPro* is short for relative pronoun, *Prep* for preposition, and *Conj* for conjunction. The sum of the probabilities for each category is 1.

$\mathcal{E}_0$:	*S*	→	*NP VP*	[0.90] I + feel a breeze
		|	*S Conj S*	[0.10] I feel a breeze + and + It stinks
	NP	→	*Pronoun*	[0.30] I
		|	*Name*	[0.10] John
		|	*Noun*	[0.10] pits
		|	*Article Noun*	[0.25] the + wumpus
		|	*Article Adjs Noun*	[0.05] the + smelly dead + wumpus
		|	*Digit Digit*	[0.05] 3 4
		|	*NP PP*	[0.10] the wumpus + in 1 3
		|	*NP RelClause*	[0.05] the wumpus + that is smelly
	VP	→	*Verb*	[0.40] stinks
		|	*VP NP*	[0.35] feel + a breeze
		|	*VP Adjective*	[0.05] smells + dead
		|	*VP PP*	[0.10] is + in 1 3
		|	*VP Adverb*	[0.10] go + ahead
	Adjs	→	*Adjective*	[0.80] smelly
		|	*Adjective Adjs*	[0.20] smelly + dead
	PP	→	*Prep NP*	[1.00] to + the east
	RelClause	→	*RelPro VP*	[1.00] that + is smelly

Figure 23.2 The grammar for $\mathcal{E}_0$, with example phrases for each rule. The syntactic categories are sentence (*S*), noun phrase (*NP*), verb phrase (*VP*), list of adjectives (*Adjs*), prepositional phrase (*PP*), and relative clause (*RelClause*).

The Grammar of ε_0

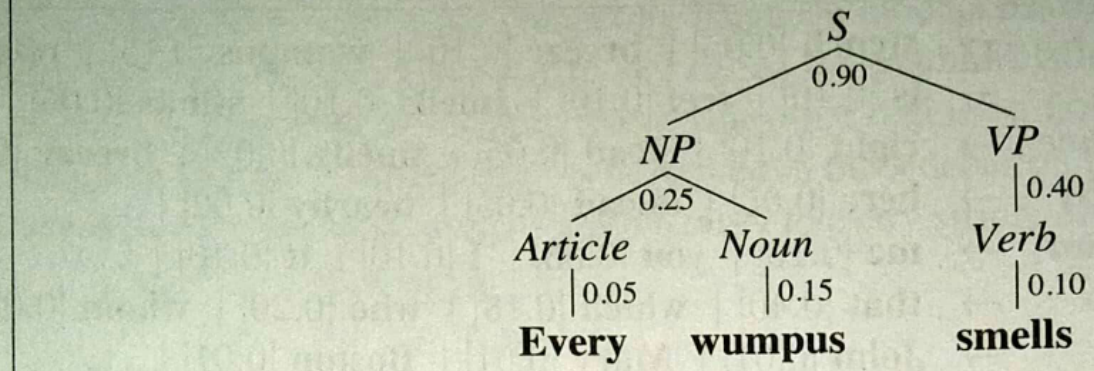


Figure 23.3 Parse tree for the sentence "Every wumpus smells" according to the grammar $\mathcal{E}_0$. Each interior node of the tree is labeled with its probability. The probability of the tree as a whole is $0.9 \times 0.25 \times 0.05 \times 0.15 \times 0.40 \times 0.10 = 0.0000675$. Since this tree is the only parse of the sentence, that number is also the probability of the sentence. The tree can also be written in linear form as [*S* [*NP* [*Article* **every**] [*Noun* **wumpus**]][*VP* [*Verb* **smells**]]].

Syntactic analysis (Parsing)

- Parsing is the process of analyzing a string of words to uncover its phrase structure, according to the rules of a grammar.
- Figure shows that we can start with the S symbol and search top down for a tree that has the words as its leaves, or we can start with the words and search bottom up for a tree that culminates in an S.

List of items	*Rule*
S	
NP VP	*S* → *NP VP*
NP VP Adjective	*VP* → *VP Adjective*
NP Verb Adjective	*VP* → *Verb*
NP Verb **dead**	*Adjective* → **dead**
NP **is dead**	*Verb* → **is**
Article Noun **is dead**	*NP* → *Article Noun*
Article **wumpus is dead**	*Noun* → **wumpus**
the wumpus is dead	*Article* → **the**

Figure 23.4 Trace of the process of finding a parse for the string "The wumpus is dead" as a sentence, according to the grammar $\mathcal{E}_0$. Viewed as a top-down parse, we start with the list of items being S and, on each step, match an item X with a rule of the form $(X \rightarrow \ldots)$ and replace X in the list of items with $(\ldots)$. Viewed as a bottom-up parse, we start with the list of items being the words of the sentence, and, on each step, match a string of tokens $(\ldots)$ in the list against a rule of the form $(X \rightarrow \ldots)$ and replace $(\ldots)$ with X.

RAMAIAH
Institute of Technology

Find out the sentence probabilities of the below sentences and write the tree in Linear Form

1. John is in the pit.
2. The wumpus that stinks is in 2 2
3. Mary is in Boston and the wumpus is near 3 2

RAMAIAH
Institute of Technology

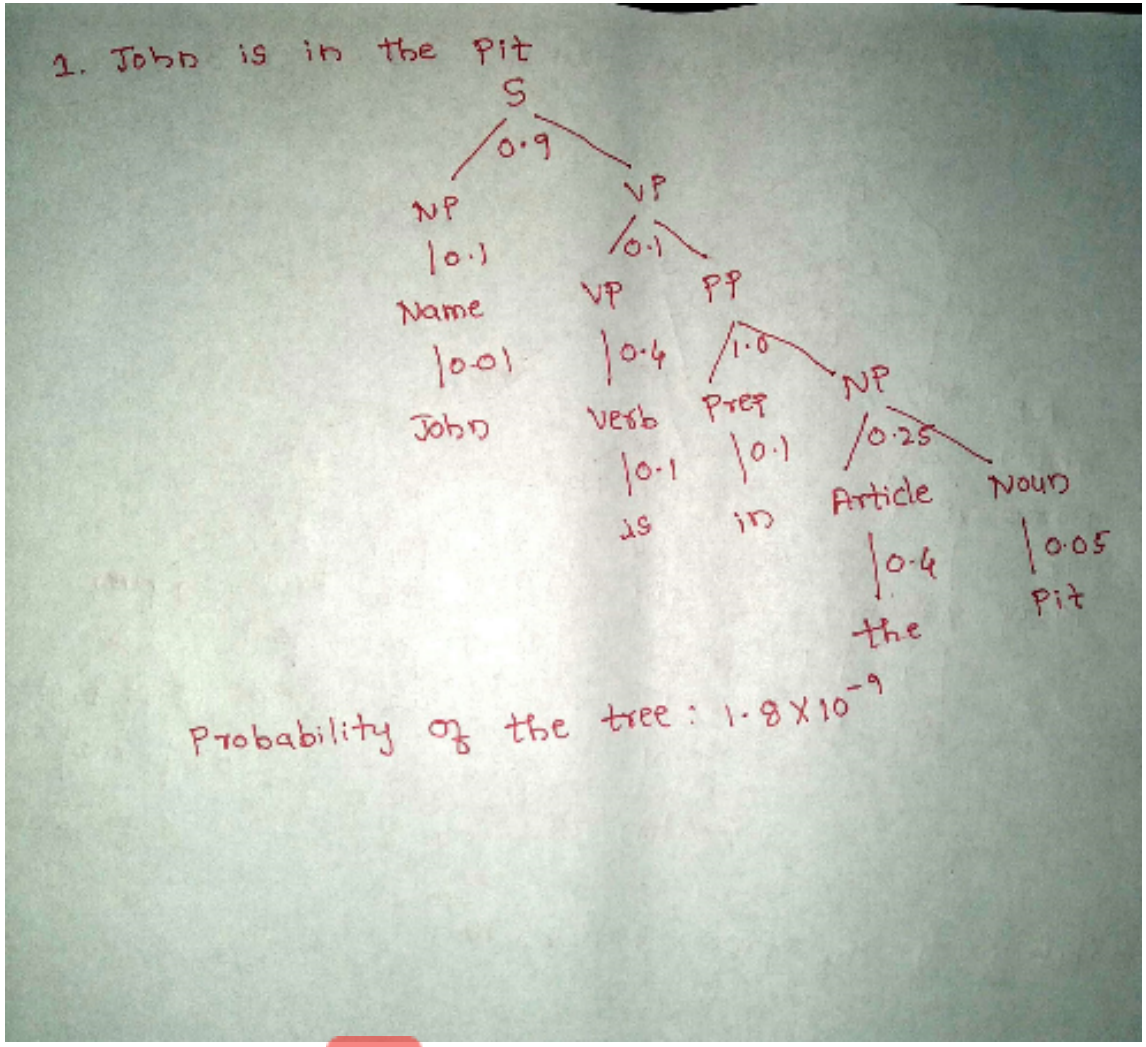
The Grammar of ε_0

Noun	→	**stench** [0.05] | **breeze** [0.10] | **wumpus** [0.15] | **pits** [0.05] | ...
Verb	→	**is** [0.10] | **feel** [0.10] | **smells** [0.10] | **stinks** [0.05] | ...
Adjective	→	**right** [0.10] | **dead** [0.05] | **smelly** [0.02] | **breezy** [0.02] ...
Adverb	→	**here** [0.05] | **ahead** [0.05] | **nearby** [0.02] | ...
Pronoun	→	**me** [0.10] | **you** [0.03] | **I** [0.10] | **it** [0.10] | ...
RelPro	→	**that** [0.40] | **which** [0.15] | **who** [0.20] | **whom** [0.02] ∨ ...
Name	→	**John** [0.01] | **Mary** [0.01] | **Boston** [0.01] | ...
Article	→	**the** [0.40] | **a** [0.30] | **an** [0.10] | **every** [0.05] | ...
Prep	→	**to** [0.20] | **in** [0.10] | **on** [0.05] | **near** [0.10] | ...
Conj	→	**and** [0.50] | **or** [0.10] | **but** [0.20] | **yet** [0.02] ∨ ...
Digit	→	**0** [0.20] | **1** [0.20] | **2** [0.20] | **3** [0.20] | **4** [0.20] | ...

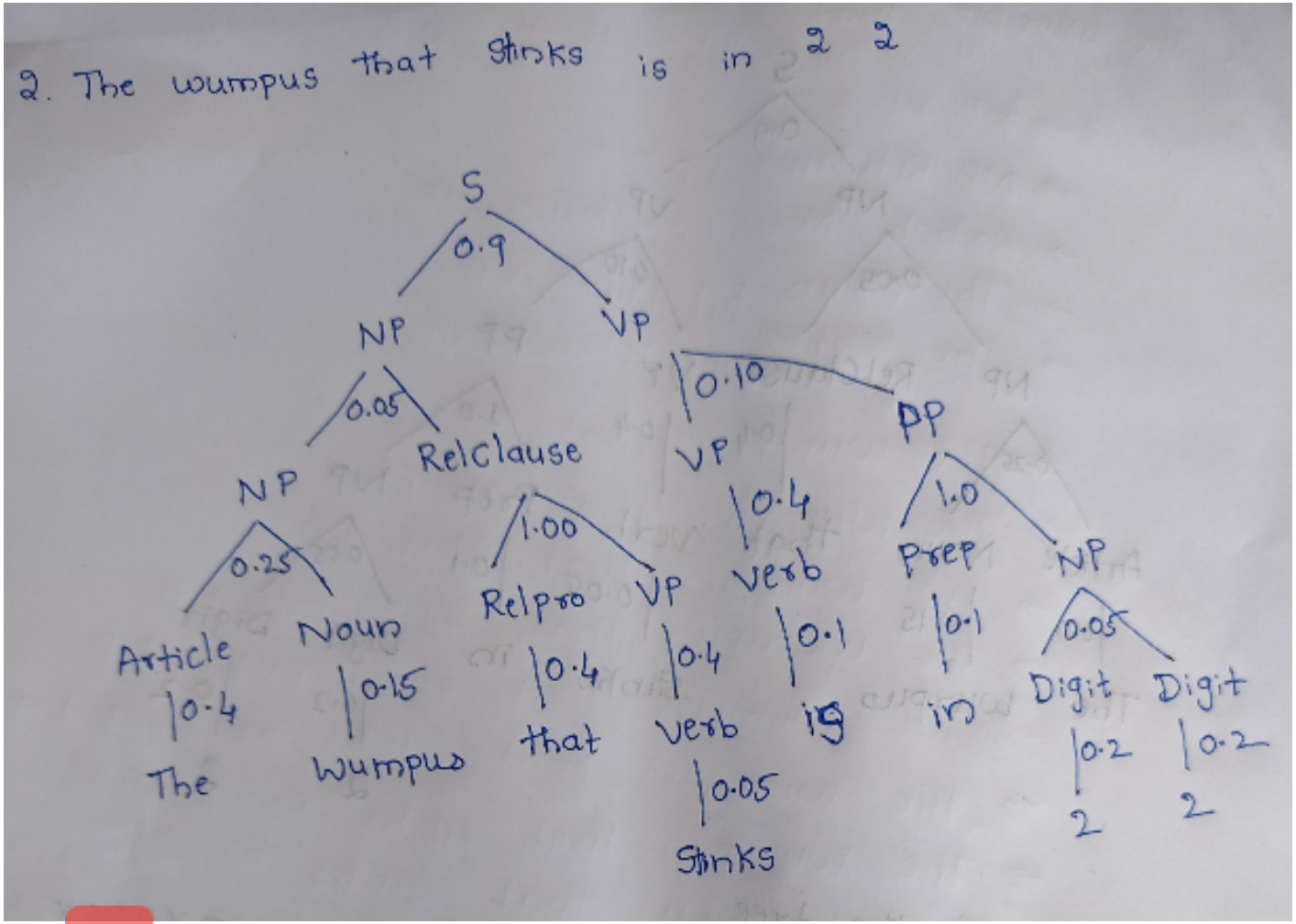
Figure 23.1 The lexicon for $\mathcal{E}_0$. *RelPro* is short for relative pronoun, *Prep* for preposition, and *Conj* for conjunction. The sum of the probabilities for each category is 1.

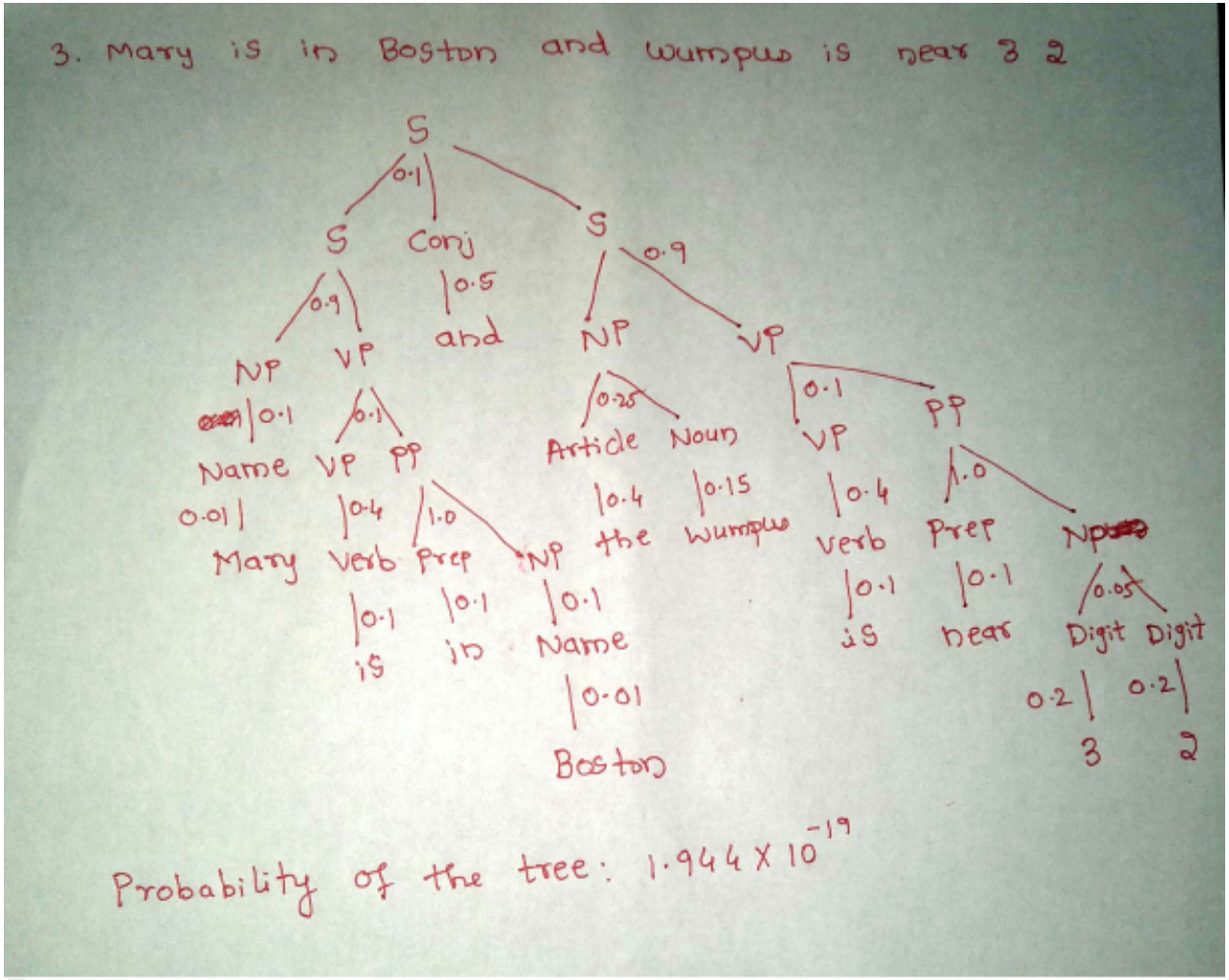
$\mathcal{E}_0$:	*S*	→	*NP VP*	[0.90] I + feel a breeze
		|	*S Conj S*	[0.10] I feel a breeze + and + It stinks
	NP	→	*Pronoun*	[0.30] I
		|	*Name*	[0.10] John
		|	*Noun*	[0.10] pits
		|	*Article Noun*	[0.25] the + wumpus
		|	*Article Adjs Noun*	[0.05] the + smelly dead + wumpus
		|	*Digit Digit*	[0.05] 3 4
		|	*NP PP*	[0.10] the wumpus + in 1 3
		|	*NP RelClause*	[0.05] the wumpus + that is smelly
	VP	→	*Verb*	[0.40] stinks
		|	*VP NP*	[0.35] feel + a breeze
		|	*VP Adjective*	[0.05] smells + dead
		|	*VP PP*	[0.10] is + in 1 3
		|	*VP Adverb*	[0.10] go + ahead
	Adjs	→	*Adjective*	[0.80] smelly
		|	*Adjective Adjs*	[0.20] smelly + dead
	PP	→	*Prep NP*	[1.00] to + the east
	RelClause	→	*RelPro VP*	[1.00] that + is smelly

Figure 23.2 The grammar for $\mathcal{E}_0$, with example phrases for each rule. The syntactic categories are sentence (*S*), noun phrase (*NP*), verb phrase (*VP*), list of adjectives (*Adjs*), prepositional phrase (*PP*), and relative clause (*RelClause*).



Probability of the parse tree: 4.32 e-12





4. Develop a parse tree for the sentence "Jack slept on the table" using the following rules.

S → NP VP

NP → N

NP → DET N

VP → V PP

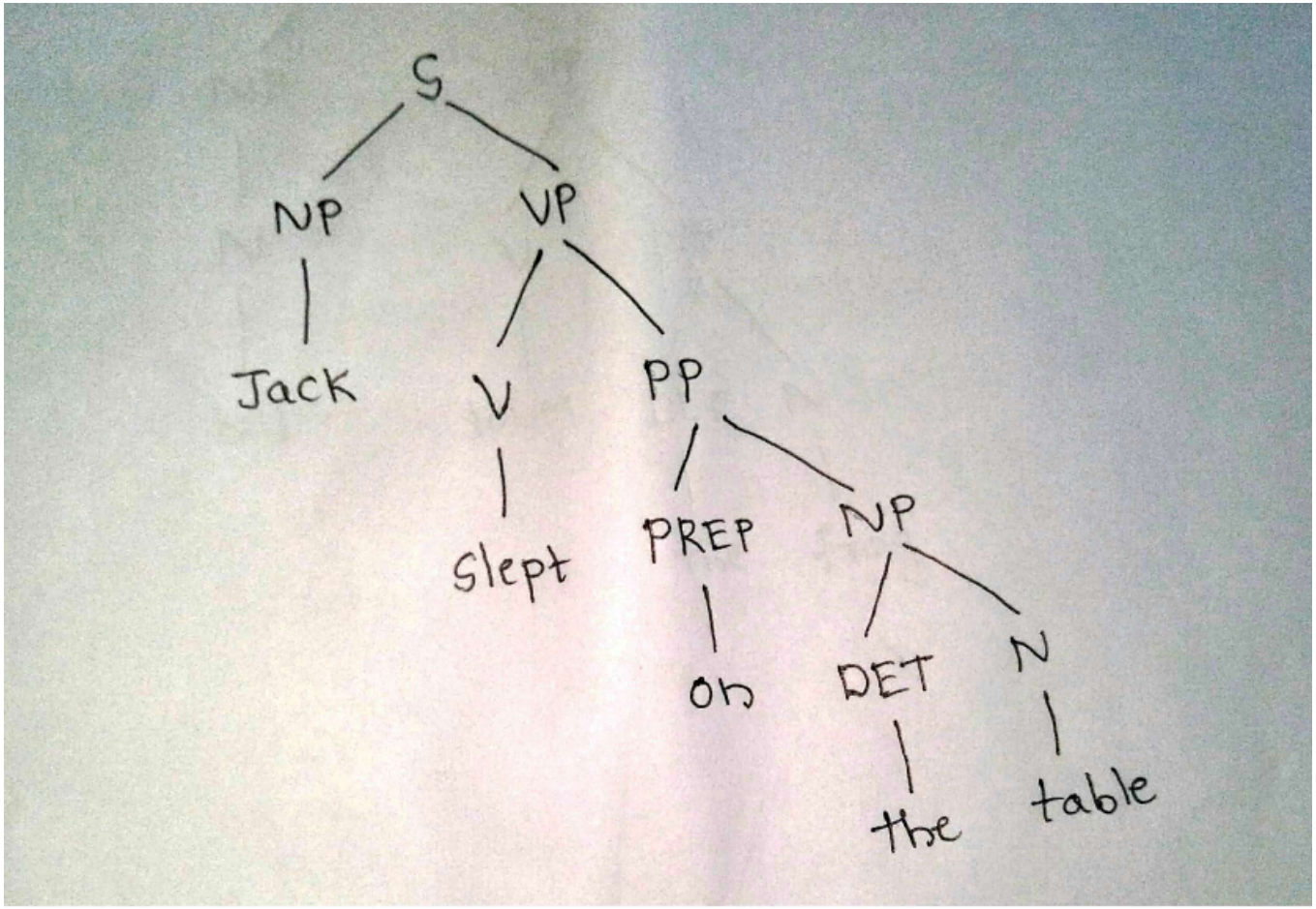
PP → PREP NP

N → jack | table

V → slept

DET → the

PREP → on



5. Derive a parse tree for the Sentence "Bill loves the frog", where the following rewrite rules are used.

S → NP VP

NP → N

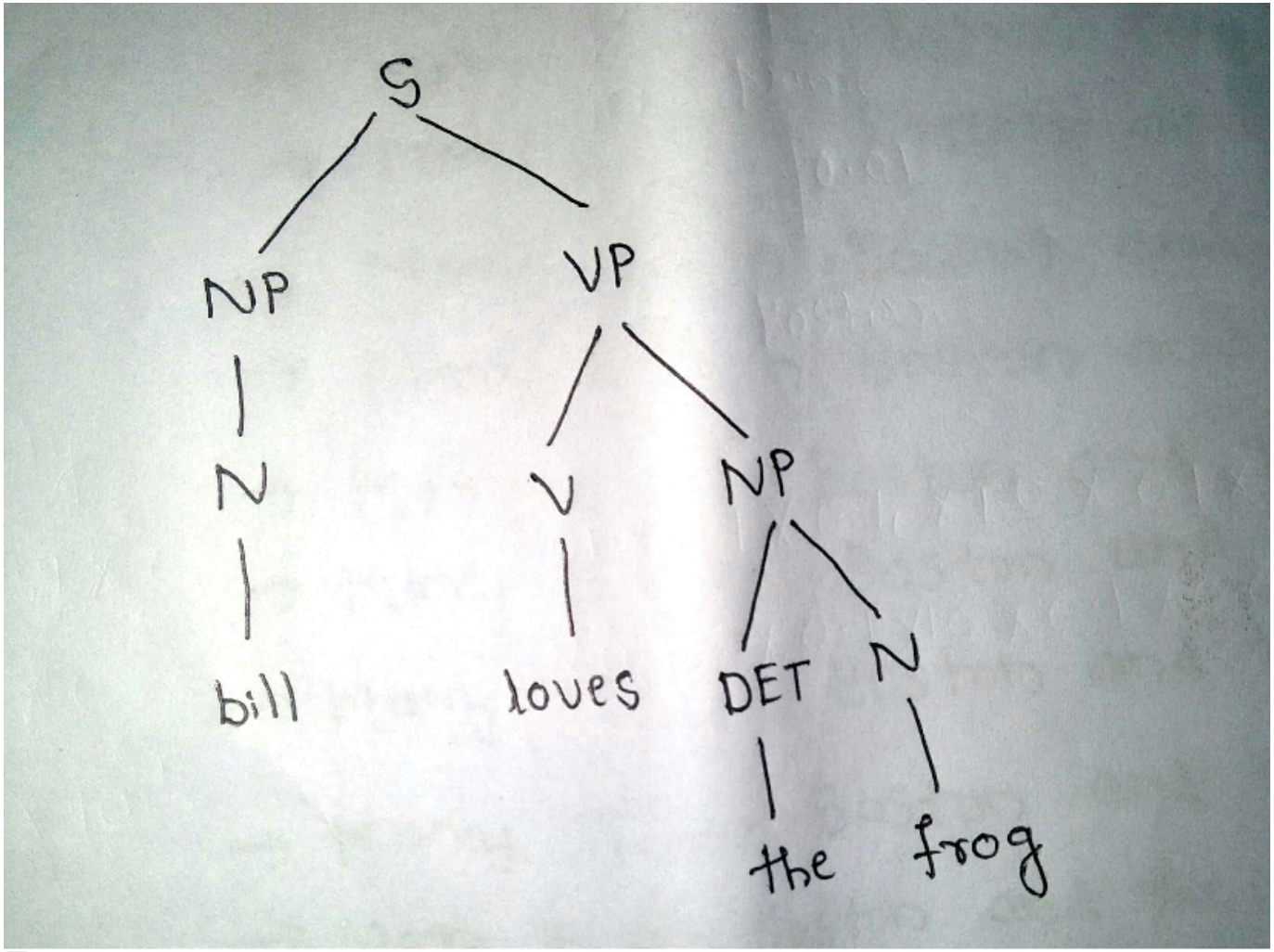
NP → DET N

VP → V NP

DET → the

V → loves

N → bill | frog



Syntactic analysis (Parsing)

- Both top-down and bottom-up parsing can be inefficient, however, because they can end up repeating effort in areas of the search space that lead to dead ends.
- Consider the following two sentences:

1. Have the students in section 2 of Computer Science 101 take the exam.
2. Have the students in section 2 of Computer Science 101 taken the exam?

- Even though they share the first 10 words, these sentences have **very different parses**, because the **first is command** and **second is a question**.
- A left to right parsing algorithm would have to guess whether the first word is part of the command or a question and will not be able to tell if the guess is correct until at least the eleventh word, take or taken.
- **If the algorithm guesses wrong, it will have to backtrack all the way to the first word** and reanalyze the whole sentence under the other interpretation.

Syntactic analysis (Parsing)

- To avoid this source of inefficiency we can use dynamic programming: **every time we analyze a substring, store the results so we won't have to reanalyze it later**.
- For example, once we discover that **"the students in section 2 of Computer Science 101"** is an NP, we record that result in a data structure known as a chart.
- Algorithms that do this are called **chart parsers**.
- There are many types of chart parsers; we describe a **bottom-up version** called the **CYK algorithm**, after its inventors, **John Cocke**, **Daniel Younger**, and **Tadeo Kasami**.

CYK Algorithms

- The CYK Algorithm is shown in Figure.
- It requires a grammar with all rules in one of two very specific formats: lexical rules of the form **X -> word**, and syntactic rules of the form **X -> Y Z**.
- This grammar format, called Chomsky Normal Form.

```
function CYK-PARSE(words, grammar) returns P, a table of probabilities
  N ← LENGTH(words)
  M ← the number of nonterminal symbols in grammar
  P ← an array of size [M, N, N], initially all 0
  /* Insert lexical rules for each word */
  for i = 1 to N do
    for each rule of form (X → words_i [p]) do
      P[X, i, 1] ← p
  /* Combine first and second parts of right-hand sides of rules, from short to long */
  for length = 2 to N do
    for start = 1 to N − length + 1 do
      for len1 = 1 to N − 1 do
        len2 ← length − len1
        for each rule of the form (X → Y Z [p]) do
          P[X, start, length] ← MAX(P[X, start, length],
                                    P[Y, start, len1] × P[Z, start + len1, len2] × p)
  return P
```

Figure 23.5 The CYK algorithm for parsing. Given a sequence of words, it finds the most probable derivation for the whole sequence and for each subsequence. It returns the whole table, P, in which an entry $P[X, start, len]$ is the probability of the most probable X of length len starting at position $start$. If there is no X of that size at that location, the probability is 0.

CYK Algorithms

- Consider the sentence

  "Fall leaves fall and spring leaves spring"

- It is ambiguous because each word (except "and") can be either a noun or a verb, and "fall" and "spring" can be adjectives as well. With ε_0 the sentence has four parses:

  [S [S [NP Fall leaves] fall] and [S [NP spring leaves] spring]]

  [S [S [NP Fall leaves] fall] and [S spring [VP leaves spring]]

  [S [S Fall [VP leaves fall]] and [S [NP spring leaves] spring]]

  [S [S Fall [VP leaves fall]] and [S spring [VP leaves spring]]]

- If we had c two-ways-ambiguous conjoined subsentences, we would have 2^c ways of choosing parses for the subsentences.

CYK Algorithms

- CYK algorithm uses space of **$O(n^2m)$** for the P table
- n is the number of words in the sentence
- m is the number of nonterminal symbols in the grammar
- time complexity: **$O(n^3m)$**.
- Since m is constant for a particular grammars, this is commonly described as $O(n^3)$.

Augmented Grammars

- How to extend the context free grammars.
- Example, not every NP is independent of context, but rather, certain NPs are more likely to appear in one context, and others in another context.

Lexicalized PCFGs (Probabilistic Context Free Grammars):

- To get the relationship between the verb **"eat"** and the nouns **"banana"** versus **"bandanna"** we can use a lexicalized PCFG, in which the probabilities for a rule depend on the relationship between words in the parse tree, not just on the adjacency of words in a sentence.
- We can't have the probability depend on every word in the tree, because **we won't have enough training data** to estimate all those probabilities.
- It is useful to introduce the notion of the **head of the phrase** – the most important word. Thus, **"eat"** is the head of the VP **"eat a banana"** and **"banana"** is the head of the NP **"a banana"**.

Augmented Grammars

- We use the notation VP(v) to denote a phrase with category VP whose head word is v.
- We say that the category VP is augmented with the head variable v.
- Here is an augmented grammar that describes the verb-object relation:

  VP(v) -> Verb(v) NP(n) [P1(v, n)]

  VP(v) -> Verb(v) [P2(v)]

  NP(n) -> Article(a) Adjs(j) Noun(n) [P3(n, a)]
- Here the probability P1(v, n) depends on the head words v and n.
- We would set this probability to relatively high when v is “eat” and n is “banana” and low when n is “bandanna”.

Semantic interpretation

- To show how to add semantics to a grammar, we start with an example that is simpler than English: **the semantics of arithmetic expressions**.
- Figure shows a **grammar for arithmetic expressions**, where **each rule is augmented with a variable** indicating the **semantic interpretation of the phrase**.
- The semantics of a digit such a s 3 is the digit itself.
- The semantics of an expression such as “3+4” is the operator “+” applied to the semantics of the phrase “3” and the phrase “4”.
- The rules obey the **principle of compositional semantics – the semantics of a phrase is a function of the semantics of the subphrases**.

$$
\begin{aligned}
Exp(x) &\rightarrow Exp(x_1)\ Operator(op)\ Exp(x_2)\ \{x = Apply(op, x_1, x_2)\}\\
Exp(x) &\rightarrow (\ Exp(x)\ )\\
Exp(x) &\rightarrow Number(x)\\
Number(x) &\rightarrow Digit(x)\\
Number(x) &\rightarrow Number(x_1)\ Digit(x_2)\ \{x = 10 \times x_1 + x_2\}\\
Digit(x) &\rightarrow x\ \{0 \leq x \leq 9\}\\
Operator(x) &\rightarrow x\ \{x \in \{+, -, \div, \times\}\}
\end{aligned}
$$

Figure 23.8 A grammar for arithmetic expressions, augmented with semantics. Each variable x_i represents the semantics of a constituent. Note the use of the $\{test\}$ notation to define logical predicates that must be satisfied, but that are not constituents.

Semantic interpretation

- Consider the below grammar for arithmetic expressions. Draw the parse tree for the expression 3+(4/2) and write the linear form of parse tree.

$$
\begin{aligned}
Exp(x) &\rightarrow Exp(x_1)\ Operator(op)\ Exp(x_2)\ \{x = Apply(op, x_1, x_2)\}\\
Exp(x) &\rightarrow (\ Exp(x)\ )\\
Exp(x) &\rightarrow Number(x)\\
Number(x) &\rightarrow Digit(x)\\
Number(x) &\rightarrow Number(x_1)\ Digit(x_2)\ \{x = 10 \times x_1 + x_2\}\\
Digit(x) &\rightarrow x\ \{0 \leq x \leq 9\}\\
Operator(x) &\rightarrow x\ \{x \in \{+, -, \div, \times\}\}
\end{aligned}
$$

Figure 23.8 A grammar for arithmetic expressions, augmented with semantics. Each variable x_i represents the semantics of a constituent. Note the use of the $\{test\}$ notation to define logical predicates that must be satisfied, but that are not constituents.

Semantic interpretation

- Figures shows the parse tree for 3+(4/2) according to this grammar.
- The root of the parse tree is Exp(5), an expression whose semantic interpretation is 5.

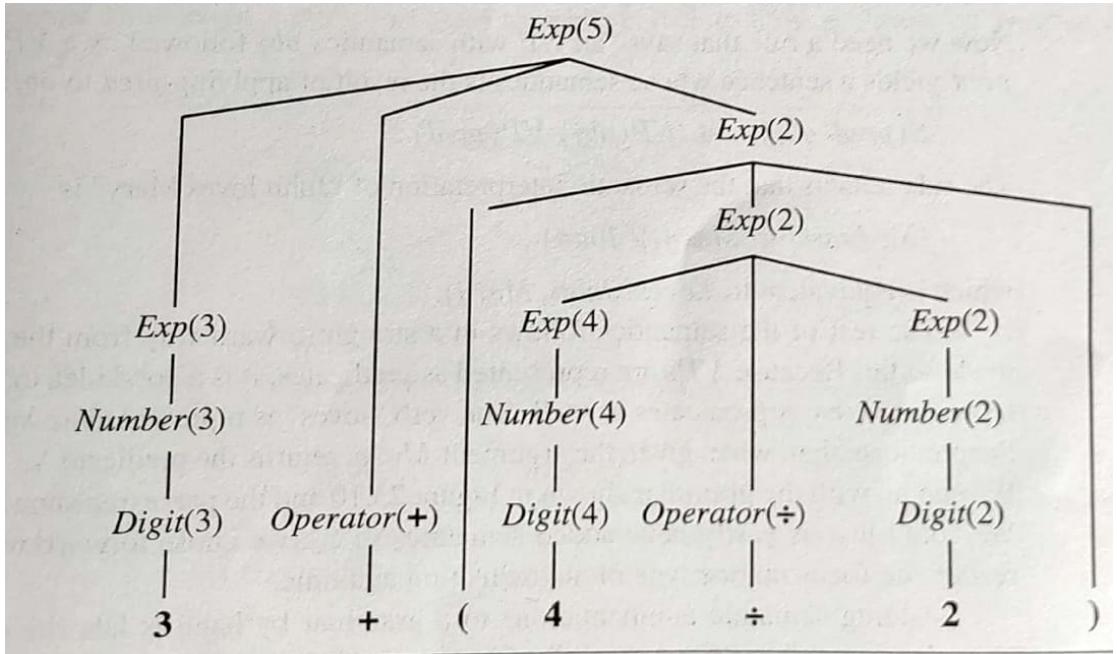


Figure 23.9 Parse tree with semantic interpretations for the string "3 + (4 ÷ 2)".

Semantic interpretation

- Now move on to the semantics of English.
- We start by determining what semantic representations we want to associate with what phrases.
- Consider the simple example sentence **"John loves Mary".**
- **John** is NP, **loves Mary** is VP.
- John is a "logical term".
- Using the λ-notation we can represent "loves Mary" as the predicate

  λx Loves(x, Mary)
- Now we need a rule that says "**an NP with semantics *obj* followed by a VP** with semantics *pred* yields a sentence whose semantics is the result of applying pred to obj."

  S(pred(obj)) -> NP(obj) VP(pred)
- The rule tells us that the semantic interpretation of "John loves Mary" is

  (λx Loves(x, Mary))(John),

  which is equivalent to **Loves(John, Mary)**

Semantic interpretation

- The verb “loves” is represented as

  λy λx Loves(x, y)

  the predicate that, when given the argument Mary, returns the predicate

  λx Loves(x, Mary)

$$S(pred(obj)) \rightarrow NP(obj)\ VP(pred)$$
$$VP(pred(obj)) \rightarrow Verb(pred)\ NP(obj)$$
$$NP(obj) \rightarrow Name(obj)$$

$$Name(John) \rightarrow \textbf{John}$$
$$Name(Mary) \rightarrow \textbf{Mary}$$
$$Verb(\lambda y\ \lambda x\ Loves(x, y)) \rightarrow \textbf{loves}$$

Figure 23.10 A grammar that can derive a parse tree and semantic interpretation for “John loves Mary” (and three other sentences). Each category is augmented with a single argument representing the semantics.

Semantic interpretation

$S(pred(obj)) \rightarrow NP(obj)\ VP(pred)$
$VP(pred(obj)) \rightarrow Verb(pred)\ NP(obj)$
$NP(obj) \rightarrow Name(obj)$

$Name(John) \rightarrow$ **John**
$Name(Mary) \rightarrow$ **Mary**
$Verb(\lambda y\ \lambda x\ Loves(x, y)) \rightarrow$ **loves**

Figure 23.10 A grammar that can derive a parse tree and semantic interpretation for "John loves Mary" (and three other sentences). Each category is augmented with a single argument representing the semantics.

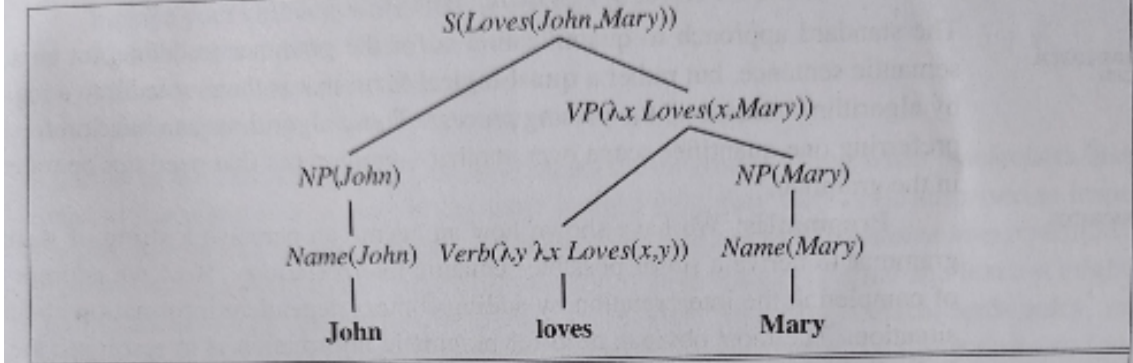


Figure 23.11 A parse tree with semantic interpretations for the string "John loves Mary".

Machine Translation

- Machine translation is the automatic translation of text from one natural language (the source) to another (the target).
- Here is a passage in English:

AI is one of the newest fields in science and engineering. Work started in earnest soon after World War II, and the name itself was coined in 1956. Along with molecular biology, AI is regularly cited as the "field I would most like to be in" by scientists in other disciplines.

- And here it is translated from English to Danish by an online tool, Google translate:

AI er en af de nyeste områder inden for videnskab og teknik. Arbejde startede for alvor lige efter Anden Verdenskrig, og navnet i sig selv var opfundet i 1956. Sammen med molekylær biologi, er AI jævnligt nævnt som "feltet Jeg ville de fleste gerne være i" af forskere i andre discipliner.

Machine Translation – Historical Applications

- **Rough translation** – as provided by free online services, gives the "gist" of a foreign sentence or document, but contains error.
- **Pre-edited translation** – is used by companies to publish their **documentations and sales materials in multiple languages**. The original source text is written in a constrained language that is easier to translate automatically, and the **results are usually edited by a human** to correct any errors.
- **Restricted-source translation** – works fully automatically, but only on **highly stereotypical language**, such as weather report.

Machine Translation Systems

- A representation language that makes all the distinctions necessary for a set of languages is called an **interlingua**.
- All translation systems must model the source and target languages, but systems vary in the type of models they use.
- Some systems attempt to analyze the source language text all the way into an interlingua knowledge representation and then generate sentences in the target language from that representation.
- This is difficult because it involves three unsolved problems: **creating a complete knowledge representation of everything; parsing into that representation**; and **generating sentences from that representation**.

Machine Translation Systems

- Other systems are based on **transfer model**.
- They keep a database of translation rules and whenever the rule matches they translate directly.
- Transfer can occur at the lexical, syntactic, or semantic level.
- For example, a strictly syntactic rule maps English [Adjective Noun] to French [Noun Adjective].
- A mixed syntactic and lexical rule maps French [S1 “et puis” S2] to English [S1 “and then” S2].

Machine Translation Systems

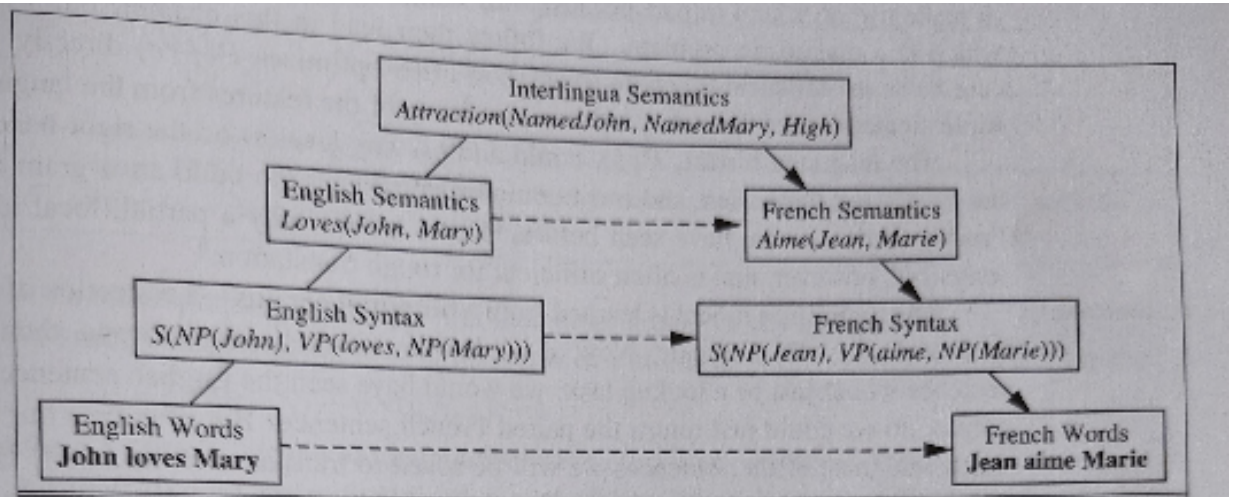


Figure 23.12 The Vauquois triangle: schematic diagram of the choices for a machine translation system (Vauquois, 1968). We start with English text at the top. An interlingua-based system follows the solid lines, parsing English first into a syntactic form, then into a semantic representation and an interlingua representation, and then through generation to a semantic, syntactic, and lexical form in French. A transfer-based system uses the dashed lines as a shortcut. Different systems make the transfer at different points; some make it at multiple points.

Speech Recognition

- Speech recognition is the task of **identifying a sequence of words uttered by a speaker**, given the acoustic signal.
- Application include
  - Millions of people interact with speech recognition everyday to navigate voice mail systems
  - Search the web from mobile phones
- Speech is an attractive option when **hands-free operation is necessary**, as when operating machinery.
- Speech recognition is difficult task because the sounds made by a speaker are **ambiguous** and **noisy**.
- As a well known example, the phrase **“recognize speech”** sounds almost the same as **“wreck a nice beach”** when spoken quickly.

Speech Recognition

- Even this short example shows several of the issues that makes speech problematic.
- First, **Segmentation**: written words in English have spaces between them, but in fast speech there are no pauses in “wreck a nice” that would distinguish it as a multiword phrase as opposed to the single word “recognize”.
- Second, **Coarticulation**: when speaking quickly the “s” sound at the end of “nice” merges with the “b” sound at the beginning of “beach”, yielding something that is close to a “sp”.
- Another problem is **homophones**: words like “to”, “too” and “two” that sound the same but differ in meaning.

RAMAIAH
Institute of Technology

Thank you