Introduction to Natural Language Processing

Syllabus

Introduction : Natural Language Processing, Why NLP is hard? Programming languages Vs Natural Languages, Are natural languages regular? Finite automata for NLP, Stages of NLP, Challenges and Issues(Open Problems) in NLP

Basics of text processing : Tokenization, Stemming, Lemmatization, Part of Speech Tagging (name entity recognization), (chunking)

1.1 Introduction : Natural Language Processing

- Humans are using language as a primary means of communication and by using this tool they are expressing their emotions and ideas.
- Language is used to shape the thoughts; it has structure and also it carries a meaning. By using our language, we naturally learn the new concepts and hardly realise how we process this natural language.
- **Natural Language Processing** is the process of computer analysis of input provided in a human language, and conversion of this input into a useful form of representation.
- Natural language processing is concerned with the development of computational models of aspects of human language processing. The following are the two main reasons for such developments.
  - o Develop automatic tool for natural language processing.
  - o Gain better understanding of human communication.
- When we build computational models by using human language, we need processing abilities where this processing abilities and incorporates how human collects, Store and process the language. It also needed a knowledge of the world and of language.
- The input and output of the NLP is text or speech.

1.1.1 History of NLP

Machine Translation (1940-1960)

In year 1940 Natural language processing has started. In 1948,in Birkbeck College, London, the first NLP application was developed. Later, in 1950, there were contradictory opinions between linguistics and computer science. Chomsky came up with his first book called syntactic structures and claimed that language is reproductive in nature. Chomsky then in 1957 introduced idea of Generative Grammar that was a rule-based description of syntactic structures.

Flavoured with Artificial Intelligence (1960-1980)

- The year 1960 to 1980 witnessed the developments like Augmented Transition Networks which was a finite state machine that is capable of recognizing regular languages. Then in 1968, Linguist Charles J. Fill more developed Case Grammar and this case grammar utilizes English language to express the association between nouns and verbs by using the preposition.

- Here, Case role is to link certain types of verbs and objects. For example, "Meena broke the table with the stone". In this example case grammar identify Meena as an agent, table as a theme, and stone as an instrument. SHRDLU and LUNAR were the key systems in the year 1960 to 1980.
- Terry Winograd in 1968-70 wrote a program named SHRDLU. This program helps users to communicate with the computer and moving objects. It can handle orders such as *pick up the green ball* and likewise answer the questions like *What is inside the black box*. The SHRDLU's key importance is it shows those syntax, semantics, and reasoning about the world that can be combined to produce a system that understands a natural language.
- Another system is LUNAR. It is the typical example of a Natural Language database interface system. LUNAR used ATNs and Woods' Procedural Semantics. It was proficient of translating extravagant natural language expressions into database queries and handle 78% of requests without mistakes.

1980 - Current

NLP was based on complex sets of hand-written rules till the year 1980. machine learning algorithms were introduced after 1980 for language processing. NLP started growing faster in the beginning of the year 1990s and accomplished good process accuracy, especially in English Grammar. Electronic text was introduced in 1990 that given a decent resource for training and examining natural language programs. Other factors may include the availability of computers with fast CPUs and more memory. The key feature behind the progress of natural language processing was the Internet. In the year1990 Probabilistic and data-driven models had become quite standard. After that, in the year 2000, a huge amount of spoken and textual data becomes available.

1.1.2 Generic NLP System

The Generic NLP systems are ELIZA, SysTran, TAUM METEO, SHRDLU, and LUNAR..

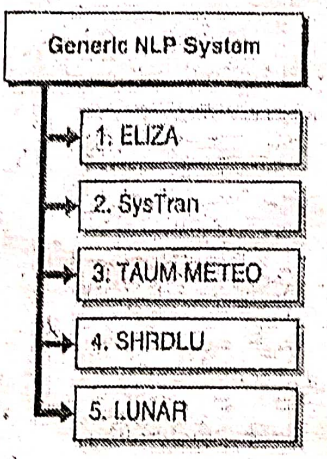


Fig. 1.1.1 : Generic NLP system

1. **ELIZA :** ELIZA an early natural language understanding program created by Joseph Weizenbaum. The human conversation with the user is mimicked by using syntactic patterns. This system demonstrates communication between humans and machines.
2. **SysTran (System Translation) :** In 1969, the SysTran machine translation system was developed. This system was developed for Russian- English translation. This system provides the first online machine translation service named Babel Fish. This Babel Fish was used by Alta Vista Search engine to handle translation request from users.
3. **TAUM METEO :** TAUM METEO, is a natural language generation system. This system was used in Canada for generating weather reports. This system accepts daily weather reports in English and French.

4. **SHRDLU :** Terry Winograd in 1968-70 wrote a program named SHRDLU. This program helps users to communicate with the computer and moving objects. It can handle orders such as pick up the green ball and likewise answer the questions like What is inside the black box. The SHRDLU's key importance is it shows those syntax, semantics, and reasoning about the world that can be combined to produce a system that understands a natural language.
5. **LUNAR :** LUNAR is the typical example of a Natural Language database interface system. It was an early question answer system that answers questions related to moon rock. It was proficient of translating extravagant natural language expressions into database queries and handle 78% of requests without mistakes.

1.1.3 Levels of NLP

There are seven interdependent levels to understand and extract meaning from a text or spoken words. In order to understand natural languages, it's important to differentiate amongst them :

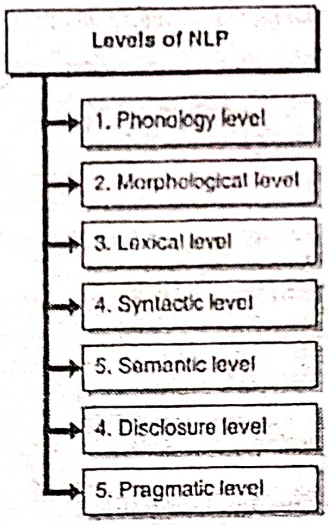


Fig. 1.1.2 : Levels of NLP

1. **Phonology level :** This level basically deals with the pronunciation. As English spelling is especially only partially phonemic, John *inputs the data* does not show these very clearly; for example, the *h* in *John* is silent and the two as in *data* resemble to very unlike sounds.
2. **Morphological level :** Morphology deals with the smallest parts of words that convey meaning, and suffixes and prefixes. Morphemes mean studying how the words are built from smaller meaning. For example, the word 'dog' has single morpheme while the word 'rats' have two morphemes 'rat' and morpheme 's' denotes singular and plural concepts.
3. **Lexical level :** The lexical level deals with the study at the level of words with respect to their lexical meaning and Part-Of-Speech (POS). This level uses lexicon that is a collection of individual lexemes. A lexeme is a basic unit of lexical meaning; which is an abstract unit of morphological analysis that represents the set of forms or "senses" taken by a single morpheme. For example, *"Duck"*, can take the form of a noun or a verb but its POS and lexical meaning can only be derived in context with other words used in the phrase/sentence.
4. **Syntactic level :** Syntactic level deals with grammar and structure of sentences. It studies the proper relationships between words. The POS tagging output of the lexical analysis can be used at the syntactic level of two group words into the phrase and clause brackets. Syntactic Analysis also referred to as *"parsing"*, allows the extraction of phrases which convey more meaning than just the individual words by themselves, such as in a noun phrase.

5. **Semantics level**
   - This level deals with the meaning of words and sentences. There are two approaches of semantic level :
     1. Syntax-driven semantic analysis,
     2. Semantic grammar
   - It is a study of the meaning of words that are associated with grammatical structure. For example, *John inputs the data* from this statement we can understand that John is an *Agent.*
6. **Discourse level**
   - This level deals with the structure of different kinds of text. There are two types of discourse :
     1. Anaphora resolution,
     2. Discourse / text structure recognition.
   - The words are replaced in Anaphora resolution, for example pronouns. Discourse structure recognition determines the purpose of sentences in the text which enhances meaningful illustration of the text.
7. **Pragmatic level**

   This level deals with the use of real world knowledge and understanding of how this influences the meaning of what is being communicated. By analysing the appropriate dimension of the documents and queries, a more detailed representation is derived.

1.2 Why NLP is Hard ?

- NLP is hard because Ambiguity and Uncertainty exist in the language.
- Ambiguity means not having well defined solution. Any sentence in a language with a large-enough grammar can have another interpretation.
- There are various forms of ambiguity related to natural language and they are :

1. Lexical Ambiguity	2. Syntactic Ambiguity
3. Semantic Ambiguity	4. Metonymy Ambiguity

1. **Lexical Ambiguity**
   - When words have multiple assertion then it is known as lexical ambiguity.
   - For example:
     - the word back can be a noun or an adjective.
     - Noun : back stage
     - Adjective : back door
2. **Syntactic Ambiguity**
   - Syntactic ambiguity means sentences are parsed in multiple syntactical forms or A sentence can be parsed in different ways.

- For example :
  - I saw the girl on the beach with my binoculars.
  - In this sentence, confusion in meaning is created. The phrase with my binoculars could modify the verb, saw or the noun, girl.

3. Semantic Ambiguity

- Semantic ambiguity is related to the sentence interpretation.
- For example :
  - I saw the girl on the beach with my binoculars.
  - The sentence means that I saw a girl through my binoculars or the girl had my binoculars with her.

4. Metonymy Ambiguity

- Metonymy is the most difficult ambiguity. It deals with phrases in which the literal meaning is different from the figurative assertion.
- For example :
  - "Nokia us screaming for new management",
  - Here it really doesn't mean that the company is literally screaming.

1.3 Programming Languages Vs. Natural Languages

Sr. No.	Natural Language	Programming Language
1.	The vocabulary of natural language is incredibly extensive	Very few words are used in computer language.
2.	Humans are capable of understanding natural language.	The machines can easily understand computer language.
3.	Natural language is inherently ambiguous.	The language of computers is unambiguous.
4.	Are open and allow combinations without the risk of making mistakes.	Are closed and fixed to avoid confusion and mistakes.
5.	Human beings have the ability to clarify the meaning of an expression. The built-in redundancy of human languages allows some ambiguity to be resolved using context.	Computers are very precise about the instructions they like to receive. Therefore, programming languages have practically no redundancy to prevent ambiguity and issue the correct commands.

1.4 Are Natural Languages Regular ?

- Regular expressions are also called as regexes. It is used for pattern matching standards for string passing and replacement.

- NLP, as an area of computer science, has greatly benefitted from regexps: they are used in phonology, morphology, text analysis, information extraction, & speech recognition. Regular expressions are placed inside the pair of matching.
- Regular expressions are powerful way to find and replace string that take a defined format. For example, regular expressions are used to parse email addresses, URL's, dates, log files, configuration file, command line, programming script or switches.
- Regular expression is a useful tool to design language compilers as well as they are used in natural language processing for tokenization, describing lexicons, morphological analysis, etc. Many of us have used simple form of regular expression for searching file patterns in MS DOS for example dir*.txt.
- The Unix based editor Ed regular expression for popular in computer science. Perl is the first language that provided integrated support for regular expression. It used slash (/) around each regular expression here we are also following the same notation. however, slashes are not a part of regular expression.
- Regular expressions introduced in 1956 by Kleene. It was originally studied as part of theory of computation. Regular expression is an algebraic formula whose value is a pattern consisting of a set of Strings known as the language of expression.
- The simple type of regular expression contains a single symbol.

For example the expression : /a/ - The expression denotes a set containing a string 'a'.

- It also pacifies sequence of characters.
- For example : /avengers/ - It denotes Indian notes that contain screen avengers and nothing else.
- **Brackets :** Characters are group by putting them between square brackets. This way any character in the class will match One character in the input.

  For example,

  /[abcd]/ will match any of a, b, c, and d.

  / [0123456789]/ specifies any single digit.
- **Range :** Sometimes regular expression led to cumbersome notation.

  For example :

  /[abcdefghijklmnopqrstuvwxyz] - It specifies any lowercase letter.
- In such cases a dash is used to specify a range.

  For Example :

  / [3-6]/ specifies any one of the digits 3,4,5, or 6.

  T /[c-f]/ specifies any one of the letter c,d,e, or f.
- **Caret^**: The caret is used at the beginning of the regular expression to specify what a single character cannot be.

  For example :

  /[^x] - matches any single character except x

  /[^A-Z]/ --> not an upper-case letter

  /[^Tt]/ --> neither 'T' nor 't'

/[^.]/ --> not a period

/[p^]/ --> either 'p' or '^'

/x^y/ --> the pattern 'x^y'

- Regular expressions are case sensitive. the pattern /t/ matches *t* but not *T*. It means pattern /Tree/ will not match pattern/tree/. To solve this issue the regular expression for this is written as / [Tt]ree/.
- *** or +** : The use of * or + allows you to add 1 or more of a preceding character.

  For example :

  oo*h! → 0 or more of a previous character (e.g. ooh!,ooooh!)

  o+h! → 1 or more of a previous character (e.g. ooh!,oooooh!)

  paa+ →paa, paaaa, paaaaaa, paaaaaaa
- **?** : The question mark ? letters optionality of the previous expression.

  For example :

  /woodchucks?/ → woodchuck or woodchucks

  /colou?r/ → color or colour
- **Anchor :** These are special characters to accomplish string operations at the start and end of a text input.

  '^' — specifies the start of the string. The character followed by the '^' in the pattern should be the first character of the string in order for a string to match the pattern.

  '$' — specifies the end of the string. The character that precedes the '$' in the pattern should be the last character in the string in order for the string to match the pattern.
- The following are some special characters :

  . → any character except a new line

  \\w → any word character

  \\W → anything but a word character

  \\d → any digit character

  \\D → anything but a digit character

  \\b → a word boundary

  \\B → anything but a word boundary

  \\s → any space character

  \\S → anything but a space character

1.5 Finite Automata for NLP

- An automaton having a finite number of states is named a Finite Automaton (FA) or Finite State automata (FSA). Finite automata are used to identify patterns. It takes the string of symbol as input and changes its state accordingly. When the required symbol is found, then the transition happens.

- When transition takes place, the automata can either move to the succeeding state or stay in the similar state. There are two states in Finite automata: Accept state or Reject state. When the input string is processed successfully, and the automata reached its final state, then it will accept it or reject it.
- Mathematically, an automaton can be represented by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$,

where ,

Q : is a finite set of states.

Σ : is a finite set of symbols, called the alphabet of the automaton.

δ : is the transition function

q_0 : is the initial state from where any input is processed ($q_0 \in Q$).

F : is a set of final state/states of Q ($F \subseteq Q$).

- There are two types of Finite Automata .
  1. Deterministic Finite automation (DFA)
  2. Non-deterministic Finite Automation (NDFA)

1. Deterministic Finite Automation (DFA)

Definition : It may be defined as the type of finite automation wherein, for every input symbol we can determine the state to which the machine will move. It has a finite number of states that is why the machine is called Deterministic Finite Automaton (DFA).

- Mathematically, a DFA can be represented by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$,

Where,

Q : is a finite set of states.

Σ : is a finite set of symbols, called the alphabet of the automaton.

δ : is the transition function where $\delta: Q \times \Sigma \rightarrow Q$.

q_0 : is the initial state from where any input is processed ($q_0 \in Q$).

F : is a set of final state/states of Q ($F \subseteq Q$).

- However, graphically a DFA can be represented by diagraphs called state diagrams:

where ,

Vertices	It represents states
Labeled arcs	It represents transitions
Empty incoming arc	It represents initial state
Double circle	It represents final state

Example :

$$\Sigma = \{0, 1\},$$
$$Q = \{q_0, q_1, q_2, q_3, q_4\},$$

start state is q_0

and final state is q_4.

- The following are the rules for transition.
  1. from state q_0 and with input a go to state q_1
  2. from state q_1 and with input b go to state q_2
  3. from state q_1 and with input c go to state q_3
  4. from state q_2 and with input b go to state q_4
  5. from state q_3 and with input b go to state q_4
- State transition diagram is as shown in Fig. 1.5.1.

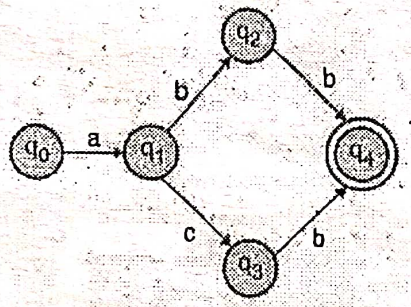


Fig. 1.5.1

2. Non-deterministic Finite Automation (NDFA)

Definition : *Non-deterministic Finite Automation is defined as the type of finite automation where for each input symbol we cannot determine the state to which the machine will move i.e., the machine can move to any combination of the states. It means for each state there can be more than one transition on a given symbol, each lead to a different state. It has a finite number of states due to this the machine is called Non-deterministic Finite Automation.*

- Mathematically, NDFA can be given by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$.

  Where,

  Q : is a finite set of states.

  $\sum$: is a finite set of symbols, called the alphabet of the automaton.

  δ : is the transition function where $\delta : Q \times \Sigma \rightarrow 2^Q$.

  q_0 : is the initial state from where any input is processed ($q_0 \in Q$).

  F : is a set of final state/states of Q ($F \subseteq Q$).

- Graphically the NDFA is represented same as DFA. Consider the same example of DFA. There are two possible transitions from state q_0 input symbol *a*. The transition diagram for NDFA is as shown in Fig. 1.5.2.

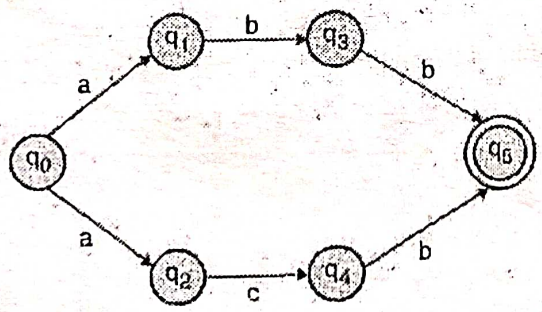


Fig. 1.5.2

1.6 Stages of NLP

There are five stages in natural language processing. The Fig. 1.6.1 shows the stages lexical analysis, syntactic analysis, semantic analysis, disclosure integration, and pragmatic analysis.

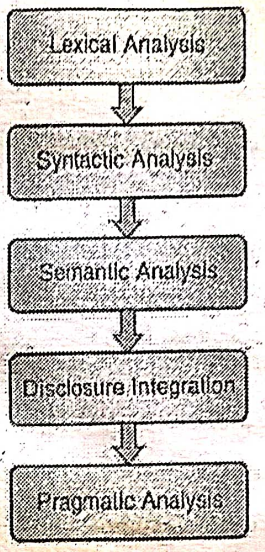


Fig. 1.6.1 : Stages of NLP

1. **Lexical Analysis**
   - Lexical Analysis is the first stage in NLP. It is also known as morphological analysis.
   - At this stage the structure of the words is identified and analysed.
   - Lexicon of a language means the collection of words and phrases in a language.
   - Lexical analysis is dividing the whole portion of text into paragraphs, sentences, and words.
2. **Syntactic Analysis (Parsing)**
   - It involves analysis of words in the sentence for grammar and ordering words in a way that shows the relationship among the words.
   - The sentence such as *The school goes to girl* is rejected by English syntactic analyser.
3. **Semantic Analysis**
   - Semantic analysis draws the exact meaning or the dictionary meaning from the text.
   - The text is checked for meaningfulness. It is done by mapping syntactic structures and objects in the task domain.
   - The semantic analyser neglects sentence such as *"hot ice-cream"*.
4. **Discourse Integration**
   - The meaning of any sentence depends upon the meaning of the sentence just before it. Furthermore, it also brings about the meaning of immediately following sentence.
   - For example : *Meena is a girl, she goes to school* here "she" is a dependency pointing to Meena.
5. **Pragmatic Analysis**
   - During this, what was said is re-interpreted on what it truly meant. It contains deriving those aspects of language which necessitate real world knowledge.
   - For example, John saw *Mary in a garden with a cat,* here we can't say that John is with cat or mary is with cat

1.7 Challenges and Issues (Open Problems) in NLP

NLP is a powerful tool with enormous benefits; nonetheless there are still numerous Natural Language Processing challenges.:

1. Contextual words and phrases and homonyms

- The same words and phrases can have diverse meanings according the context of a sentence and many words have the exact same pronunciation but completely different meanings.
- For example :
  - I ran to the store because we ran out of milk.
  - Can I run something past you really quick?
  - The house is looking really run down.
- In the above three sentences the meaning of the *run* is different according to the context
- Homonyms means the pronunciation of two or more words is same but have different meaning. For example, their and there, right and write. This will create problem in question answering and speech-to-text applications.

2. Synonyms

- Synonyms can cause issues like contextual understanding since we use many different words to express the identical idea.
- Additionally, some of these words may convey exactly the same meaning, while some may be levels of complexity and different people use synonyms to denote slightly different meanings within their personal vocabulary.
- For example, small, little, tiny, minute have same meaning.

3. Irony and sarcasm

- Irony and sarcasm present problems for machine learning models since they usually use words and phrases that, strictly by definition, may be positive or negative, but truly mean the opposite.
- Models can be trained with certain indications that frequently accompany ironic or sarcastic phrases, like yeah right, whatever, etc., and word embeddings (where words that have the same meaning have a similar representation), but it's still a complicated process.

4. Ambiguity

- Ambiguity in NLP refers to sentences and phrases that potentially have two or more possible interpretations.
- There is lexical, syntactic and semantic ambiguity.

5. Errors in text or speech

- Misspelled or misused words can generate problems for text analysis. Autocorrect and grammar correction applications can handle common mistakes, but do not at all times understand the writer's intention.
- With spoken language it is difficult for the machine to understand mispronunciations, different accents, stammers, etc.

6. **Idioms and slang**
   - Informal phrases, expressions, idioms, and culture-specific lingo present a number of problems for NLP, especially for models intended for comprehensive use.
   - Because as formal language, idioms may have no dictionary definition at all, and these expressions may even have different meanings in different geographic areas.
   - Furthermore, cultural slang is continuously morphing and increasing, so new words arise every day.
7. **Domain specific language**
   - Different businesses and industries often use very different language.
   - An NLP processing model needed for healthcare would be very different than one used to process legal documents.
8. **Low-resource languages**
   - Artificial Intelligence, machine learning NLP applications have been mostly built for the most common, widely used languages.
   - It is absolutely incredible at how precise translation systems have become. However, many languages, especially those spoken by people with less access to technology often go overlooked and under processed.
   - For example, there are over 3,000 languages in Africa, alone. There simply isn't ample data on many of these languages.

Review Questions

Q. 1 What is NLP? What are the applications of NLP ?

Q. 2 Explain generic NLP system.

Q. 3 What are the levels of NL P?

Q. 4 Explain the stages of NLP ?

Q. 5 Explain the challenges in NLP ?

Q. 6 List and explain applications of NLP.

Q. 7 Explain the knowledge level ?

Q. 8 Why NLP Is hard ?.

Q. 9 Write short note Finite Automata in NLP ?

Q.10 Write difference between Programming languages Vs Natural Languages.

2

Unit II

Language Syntax and Semantics

Syllabus

Morphological Analysis : What is Morphology? Types of Morphemes, Inflectional morphology & Derivational morphology, Morphological parsing with Finite State Transducers (FST)

Syntactic Analysis : Syntactic Representations of Natural Language, Parsing Algorithms, Probabilistic context-free grammars, and Statistical parsing

Semantic Analysis : Lexical Semantic, Relations among lexemes & their senses –Homonymy, Polysemy, Synonymy, Hyponymy, WordNet, Word Sense Disambiguation (WSD), Dictionary based approach, Latent Semantic Analysis

2.1 Morphological Analysis

2.1.1 What is Morphology ?

- Morphology is nothing but studying the way of words formation from minor meaning-bearing units, morphemes.
- The smallest meaningful units in any language are called morphemes. In a language, morphemes are the building blocks of a word. Examples of morphemes in English include words, plural morphemes ('-s' and '-es'), and grammatical morphemes ('-ing' and '-ed'), among others.

2.1.2 Types of Morphemes

There are two classes of morphemes :

1. Stem or free morphemes
2. Affixes or bound morphemes

1. Stem or free morphemes

- The morpheme that can appear in isolation and be meaningful is known as a free morpheme. The stem (root) of a word is referred to as a free morpheme. Because a word's root form can reveal its meaning.
- For instance, dog, carry, good, and so on.

2. Affixes or bound morphemes

- A bound morpheme is a morpheme that is usually attached to any other free morpheme to provide additional meaning of various kinds, including plural and grammatical variations. Affixes are sometimes used to refer to bound morphemes.
- Let take the example of a word unhappiness. The word unhappiness consists of 3 parts as shown in the Fig. 2.1.1

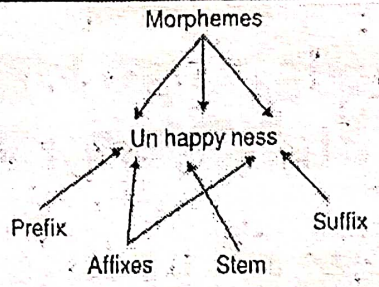


Fig. 2.1.1 : Example of word unhappiness

- The main morpheme of the word is stem. It also states that stem are the morphemes with the central meaning.
- Once the affixes are applied to the stem it modifies its meaning. There are 4 types of affixes :
  - **Prefix :** These are the morphemes that come before the stem.
  - **Suffix :** These are the morphemes that comes at the end of the stem
  - **Infix :** These are the morphemes that come to the any end of the stem
  - **Circumfix :** These are the morphemes that come inside the stem
- For Example :

Table 2.1.1

Affixes	Original word	Affix applied	Word after applying affix
suffix	rat	s	rats
prefix	happy	un	unhappy
circumfix	sag	Ge and t	Ge-sag-t (in German it means said)
inflix	hingi	um	Humingi (in Philippine language)

2.1.3 Inflectional Morphology & Derivational Morphology

There are three types of word formation

1. Inflection
2. Derivation
3. Compounding

1. Inflection

- When the word stem is fused with the grammatical morpheme then it generally results in a word of the similar class as the original stem.
- If we consider the English language then the things that get inflected are nouns, verbs, and sometimes adjectives. So, the affixes are quite small In number.
- Nouns have simple inflectional morphology. Examples of the Inflection of noun in English are given below, here an affix is marking plural.
  - mat (-s), cat(-s)

- ibis(-es)
- thrush(-es)
- waltz(-es), finch(-es), box(-es)
- butterfly(-lies)
- ox (oxen), mouse (mice) [irregular nouns]

- A possessive affix is a suffix or prefix attached to a noun to indicate its possessor. the following are some affixes marking possessive
  - Regular singular noun- llama's
  - Plural noun not ending in 's '–children's
  - Regular plural noun –llamas'
  - Names ending in 's' or 'z' - Euripides' comedies
- Verbs have slightly more complex inflectional, but still relatively, simple inflectional morphology. There are three types of verbs in English
  - Main verbs- eat, sleep, impeach
  - Modal verbs- can will, should
  - Primary verbs-be, have, do
- Regular verb is a large class are regular. In this class all the verbs have the same endings marking the same functions. Regular verbs have four morphological form. Just by knowing the stem we can predict the other forms. Just add one of the three predictable endings and make some regular spelling changes. These regular verbs and forms are significant in the morphology of English because of their majority and being productive.
- The Table 2.1.2 shows the morphological forms for regular verbs.

Table 2.1.2

Stem	Talk	Urge	Cry	tap
-s form	Talks	urges	cries	Taps
-ing form	Talking	urging	Crying	Tapping
Past form or -ed participle	Talked	urged	Cried	tapped

- The morphological form for the irregular verbs is shown in the Table 2.1.3

Table 2.1.3

Stem	Eat	Think	put
-s form	Eats	Thinks	puts
-ing form	eating	Thinking	putting
Past form	Ate	Thought	Put
-ed participle	Eaten	Thought	put

The simple from : be

The -ing participle form : being

The past participle : been

The first person singular present tense from : am

The third person present tense (-s) from : is

The plural present tense from : are

The singular past tense form : was

The plural past tense from : were

2. Derivation

- The combination of a word stem with a grammatical morpheme usually resulting in a word of a different class, often with a meaning hard to predict exactly.
- Nominalizations are nothing but the nouns which are generated from adjectives in English. The Table 2.1.4 shows the formation of new nouns, often from verbs or adjectives.

Table 2.1.4

Suffix	Base Verb/Adjective	Derived Noun
-ation	Computerize (V)	Computerization
-ee	Appoint (V)	Appointee
-er	Kill (V)	Killer
-ness	Fuzzy (A)	Fuzziness

- Adjectives can also be derived from nouns or verbs. The Table 2.1.5 shows the objectives derived from the verbs/noun.

Table 2.1.5

Suffix	Base Verb/Adjective	Derived Noun
-al	Computation (N)	Computational
-able	embrace (V)	Embraceable
-less	clue (A)	Clueless
-ness	Fuzzy (A)	Fuzziness

3. Compounding

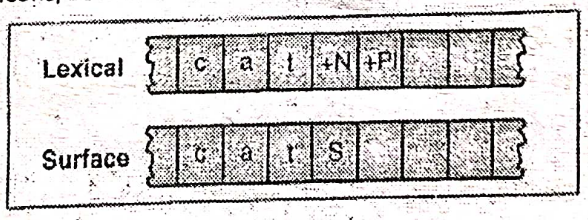
- A word is composed of a number of morphemes concatenated together.
- In English, compounds can be written together, for example, notebook, desktop, overlook, bookstore, fireman etc.
- Separately the compounds are written as Living room, dinner table, full moon etc.

2.1.4 Morphological Parsing with Finite State Transducers (FST)

- The objective of the morphological parsing is to produce output lexicons for a single input lexicon. let's consider, parsing just the productive nominal plural (-s) and the verbal progressive (-ing). Our aim is to take input forms like those in the first column below and produce output forms like those in the second column. The second column contains the stem of each word as well as mixed morphological features. These features specify additional information about the stem. For example: +SG - singular, +PL – plural.

Cats	cat + N + PLU
Cat	cat + N + SG
Goose	goose + N + SG or goose + V
Geese	goose + N + PLU
Gooses	goose + V + 3SG
Catch	catch + V
Caught	catch + V + PAST or catch + V + PP

- The column contains the stem of the corresponding word (lexicon) in first column, along with its morphological features, like, +N means word is noun, +SG means it is singular, +PL means it is plural, +V for verb, and pres-part for present participle. There can be more than one lexical level representation for a given word
- We achieve it through two level morphology, which represents a word as a correspondence between lexical level - a simple concatenation of lexicons, as shown in column 2 of table.



- For a morphological processor, we need at least followings :
  - **Lexicon :** The list of stems and affixes together with basic information about them such as their main categories (noun, verb, adjective, ...) and their sub-categories (regular noun, irregular noun, ...).
  - **Morphotactics :** The model of morpheme ordering that explains which classes of morphemes can follow other classes of morphemes inside a word.
  - **Orthographic Rules (Spelling Rules) :** These spelling rules are used to model changes that occur in a word (normally when two morphemes combine).

2.1.4(A) Lexicon and Morphotactics

- A lexicon is a repository for words. They are grouped according to their main categories: noun, verb, adjective, adverb, ...
- They may be also divided into sub-categories : regular-nouns, irregular-singular nouns, irregular-plural nouns, ...

- The simplest possible lexicon would consist of an explicit list of every word of the language (every word, i.e., including abbreviations ('AAA') and proper names ('Jane' or 'Beijing') as follows : a, AAA, AA, Aachen, aardvark, aardwolf, aba, abaca, aback . . .
- The simplest way to create a morphological parser, put all possible worlds (together with its inflections) into a lexicon.
- Since it will often be inconvenient or impossible, for the various reasons, to list every word in the language, computational lexicons are usually structured with a list of each of the stems and affixes of the language together with a representation of the morphotactics that tells us how they can fit together. There are many ways to model morphotactics; one of the most common is the finite-state automaton.
- The following FSA assumes that the lexicon includes regular nouns (reg-noun) that take the regular -s plural (e.g., cat, dog, fox, aardvark), also includes irregular noun forms that don't take -s, both singular irreg-sg-noun (goose, mouse) and plural irreg-pl-noun (geese, mice).

Lexicon :

regular-noun	irregular-pl-noun	irreg-sg-noun	plural
fox	geese	goose	-s
cat	sheep	Sheep	
dog	mice	mouse	

- Simple English Nominal Inflection (Morphotactic Rules)

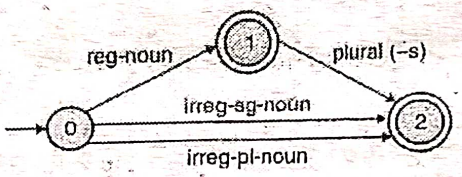


Fig. 2.1.2

- A similar model for English verbal inflection is given below. This lexicon has three stem classes (reg-verb-stem, irreg-verb-stem, and irreg- past-verb-form), plus 4 more affix classes (-ed past, -ed participle, -ing participle, and 3rd singular – s).

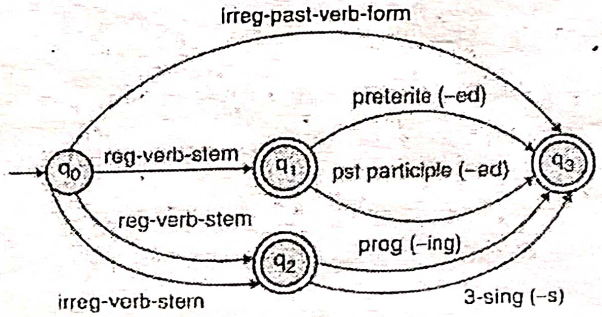


Fig. 2.1.3

reg-verb-stem	irreg-verb-stem	irreg-past-verb	past	past-part	press-part	3sg
walk	cut	caught	-ed	-ed	-ing	-s
fry	speak	ate				
talk	Sing	eaten				
impeach	sang					
	cut					
	spoken					

- Derivational data here we are modelling, so here the adjectives will become opposites, comparatives and adverbs. Small part of the morphotactics of English adjectives, for example big, bigger, biggest. An initial hypothesis might be that adjectives can have an optional prefix (un-), an obligatory root (big, cool, etc) and an optional suffix (-er, -est, or -ly).

Big, bigger, biggest	cool, cooler, coolest, coolly
Happy, happier, happiest, happily	red, redder, reddest
Unhappy, unhappier, unhappiest, unhappily,	real, unreal, really
Clear, clearer, clearest, clearly, unclear, unclearly,	

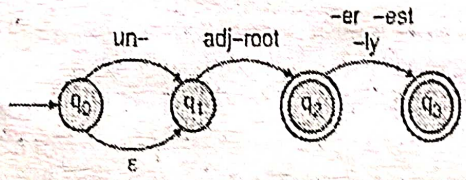


Fig. 2.1.4

- An FSA for another fragment of English derivational morphology.

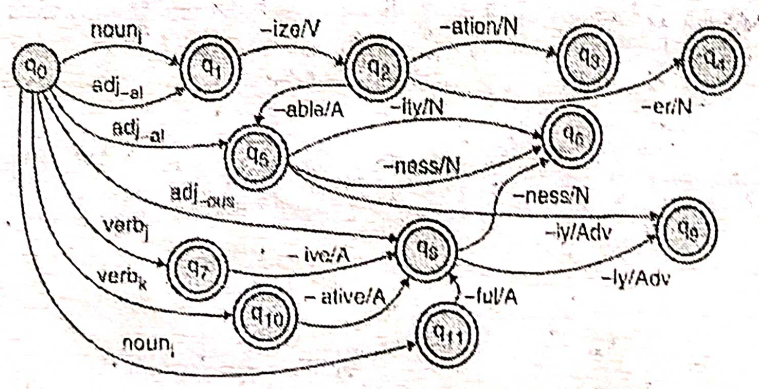


Fig. 2.1.5

- **Compose :** The Fig. 2.1.5 shows the fleshed-out English nominal inflection FST Tlex, expanded from Tnum by replacing the three arcs with individual stem work the rules are demonstrated in the Table 2.1.6.

Table 2.1.6

Name	Description of Rule	Example
Consonant doubling	1 letter 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,-ch, -sh before -s	Watch/watches
Y replacement	-y changes to -ie before -s, -I before -ed	Try/tries
K insertion	verbs ending with vowel + -c add-k	Panic/paniked

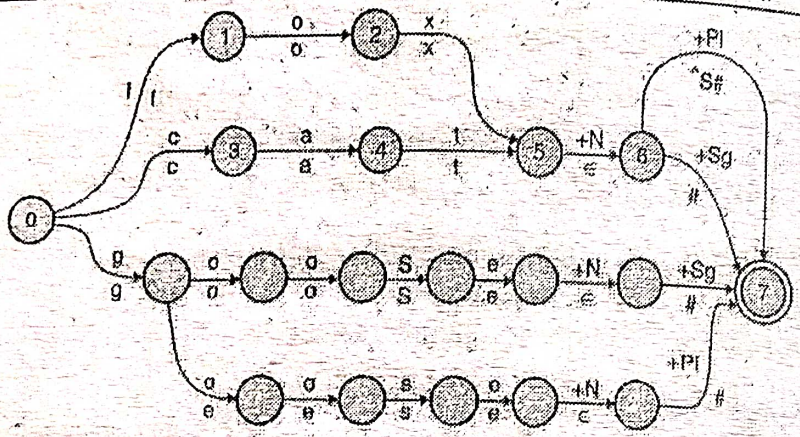


Fig. 2.1.6

2.2 Syntactic Analysis

- The third phase of NLP is syntactic analysis, also known as parsing or syntax analysis. The goal of this phase is to extract exact meaning, or dictionary meaning, from the text. Syntax analysis examines the text for meaning by comparing it to formal grammar rules. The sentence "hot ice cream," for example, would be rejected by a semantic analyzer.
- In this sense, syntactic analysis or parsing can be defined as the process of analysing natural language strings of symbols in accordance with formal grammar rules. The term 'parsing' derives from the Latin word 'pars,' which means 'part'.

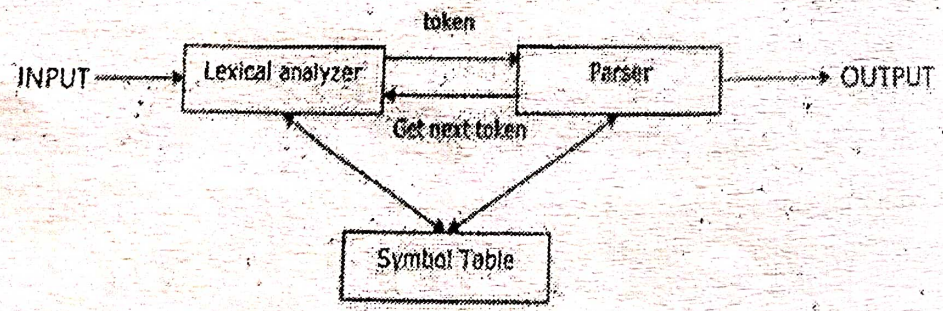


Fig. 2.2.1 Parser

- It is used to carry out the parsing task. It is a software component that takes input data (text) and returns a structural representation of the input after checking for correct syntax using formal grammar. It also constructs a data structure in the form of a parse tree, an abstract syntax tree, or another hierarchical structure.

2.2.1 Syntactic Representations of Natural Language

- A parse tree is a tree that, in accordance with a formal grammar, reveals the relationships between words or subphrases, for instance, to highlight a sentence's syntactical structure. The features of the final tree will differ depending on the grammar we use.
- Both dependency parsing and constituency parsing make use of distinct grammars. The resulting trees will be very different because they are based on completely different assumptions. However, the ultimate objective is syntactic information extraction in both instances.

Constituency Parsing.

- The formalism of context-free grammars serves as the foundation for the constituency parse tree. The sentence is broken up into constituents, or sub-phrases that fall under a particular grammar category, in this kind of tree.
- Examples of noun phrases include "a dog," "a computer on the table," and "the nice sunset" in English, while verb phrases include "eat a pizza" and "go to the beach."
- Using a set of rules, the grammar specifies how to construct valid sentences. The rule states, for instance, that we can combine a noun phrase (NP) with a verb (V) to create a verb phrase (VP).
- These rules can be used to create valid sentences, but we can also use them to extract the syntactical structure of a given sentence in accordance with grammar.
- An example of a constituency parse tree for the straightforward sentence "I saw a fox" follows :

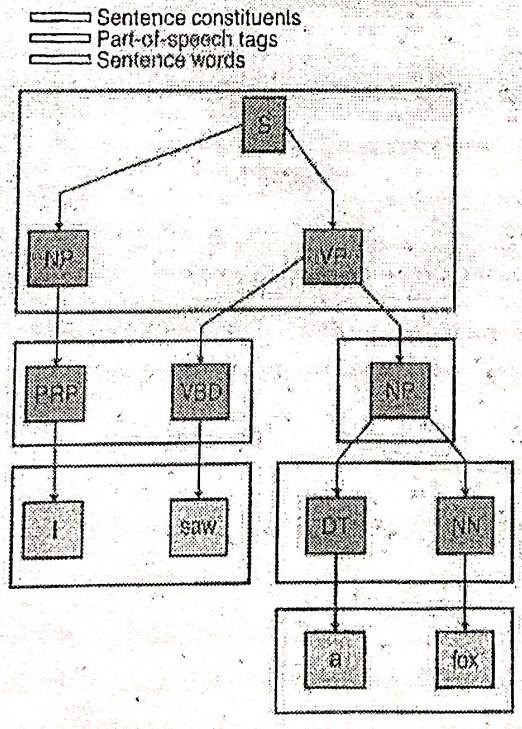


Fig. 2.2.2

- The sentence's words are always the terminal nodes in a constituency parse tree. In most cases, each word has a parent node that contains its part-of-speech tag (noun, adjective, verb, etc.), although other graphical representations may not include this.
- The sentence's constituents are represented by all other non-terminal nodes, which typically fall into one of three categories: verb phrase, noun phrase, or prepositional phrase (PP).

- Our sentence has been divided into a noun phrase consisting of the singular word "I" and a verb phrase consisting of the phrase "saw a fox" at the first level below the root in this example. This indicates that grammar has a rule like that states that a noun phrase and a verb phrase can be combined to form a sentence.
- The verb phrase is also broken up into a verb and another noun phrase. This corresponds, as we can imagine, to yet another grammar rule.
- Dependency parsing does not make use of phrasal constituents or sub-phrases, unlike constituency parsing. Instead, the sentence's syntax is represented by dependencies between words, which are like directed, typed edges in a graph.
- Formally, a dependency parse tree is a graph in which each edge links two words and the set of vertices contains the sentences' words. There must be three requirements for the graph :
  1. There can only be one root node with no edges coming in.
  2. There must be a route from the root to each node in for
  3. Except for the root, each node must have exactly one incoming edge.
- Additionally, the grammatical relationship that exists between the two words is described by the type assigned to each edge in.
- Let's take a look at the previous example if we use dependency parsing :

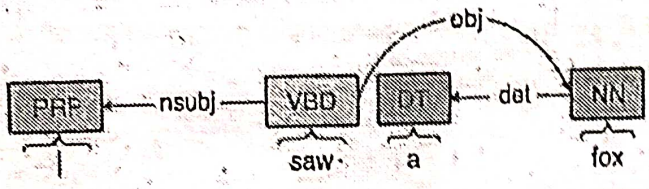


Fig. 2.2.3

- The outcome is completely different, as we can see. This way, the verb in the sentence is at the root of the tree, and the edges between words show how they relate to one another.
- For instance, the word "saw" has an outgoing edge that is of the type nsubj to the word "I," indicating that "I" is the verb's nominal subject. We say that "I" is dependent on "saw" in this instance.

2.2.2 Parsing Algorithms

Types of parsing :

1. Top down parsing
2. Bottom up parsing

1. **Top-down parsing** : A parser can start with the start symbol and try to transform it to the input string. Example : LL Parsers.
2. **Bottom-up parsing** : A parser can start with input and attempt to rewrite it into the start symbol. Example : LR Parsers.

2.2.2(A) Top-Down Parsing

It can be viewed as an attempt to find a left-most derivation for an input string or an attempt to construct a parse tree for the input starting from the root to the leaves.

Types –Top Down Parsing

1. Recursive descent parsing
2. Predictive parsing

1. Recursive descent parsing

- Typically, top-down parsers are implemented as a set of recursive functions that descent through a parse tree for a string. This approach is known as recursive descent parsing, also known as LL(k) parsing where the first L stands for left-to-right, the second L stands for leftmost-derivation, and k indicates k-symbol lookahead.
- Therefore, a parser using the single-symbol look-ahead method and top-down parsing without backtracking is called LL(1) parser. In the following sections, we will also use an extended BNF notation in which some regulation expression operators are to be incorporated.
- This parsing method may involve backtracking.

Example for : backtracking

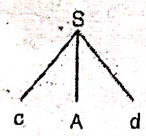
Consider the grammar G : S → cAd

A → ab | a

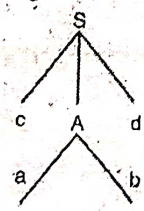
and the input string w = cad.

The parse tree can be constructed using the following top-down approach :

Step 1 : Initially create a tree with single node labeled S. An input pointer points to 'c', the first symbol of w. Expand the tree with the production of S.

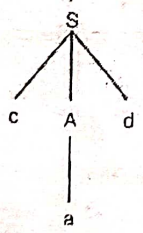


Step 2 : The leftmost leaf 'c' matches the first symbol of w, so advance the input pointer to the second symbol of w 'a' and consider the next leaf 'A'. Expand A using the first alternative.



Step 3 : The second symbol 'a' of w also matches with second leaf of tree. So advance the input pointer to third symbol of w 'd'. But the third leaf of tree is b which does not match with the input symbol d. Hence discard the chosen production and reset the pointer to second backtracking.

Step 4 : Now try the second alternative for A.



Now we can halt and announce the successful completion of parsing.

2. **Predictive parsing**

- It is possible to build a nonrecursive predictive parser by maintaining a stack explicitly, rather than implicitly via recursive calls. The key problem during predictive parsing is that of determining the production to be applied for a nonterminal. The nonrecursive parser in figure looks up the production to be applied in parsing table. In what follows, we shall see how the table can be constructed directly from certain grammars.

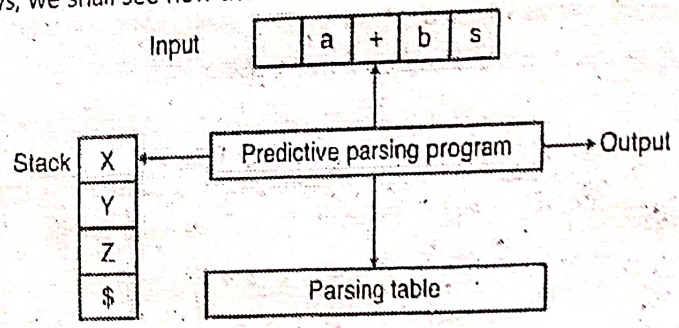


Fig. 2.2.4 : Model of a nonrecursive predictive parser

- A table-driven prediction parser has an input buffer, a stack, a parse table, and an output stream. The input buffer is followed by the string to parse and the symbol $ used as a right-end marker to mark the end of the input string. The stack contains a series of grammar symbols with a trailing $ to mark the end of the stack.
- Initially, the stack contains grammar start symbols above the $. The parse table is a two-dimensional array M[A,a]. where A is a nonterminal and a is a terminal or $ symbol. The parser is controlled by a program that works as follows: The program takes into account X, the symbol on top of the stack, and a, the current input symbol. These two symbols determine the parser's action. You have three options.

  1 If X= a=$, the parser halts and announces successful completion of parsing.

  2 If X=a!=$, the parser pops X off the stack and advances the input pointer to the next input symbol.

  3 If X is a nonterminal, the program consults entry M[X, a] of the parsing table M. This entry will be either an X-production of the grammar or an error entry. If, for example, M [X, a] = {X - > UVW}, the parser replaces X on top of the stack by WVU(with U on top). As output, we shall assume that the parser just prints the production used; any other code could be executed here. If M [X, a] = error, the parser calls an error recovery routine

Algorithm for Nonrecursive predictive arsing :

- **Input :** A string w and a parsing table M for grammar G.
- **Output :** If w is in L(G), a leftmost derivation of w; otherwise, an error indication.

Method :

- Initially, the parser is in a configuration in which it has $S on the stack with S, the start symbol of G on top, and w$ in the input buffer. The program that utilizes the predictive parsing table M to produce a parse for the input is shown as follows,

```
set ip to point to the first symbol of w$. repeat
let X be the top stack symbol and a the symbol pointed to by ip. if X is a terminal of $ then
if X = a then
pop X from the stack and advance ip else error()
```

```
else
if M [X, a] = X → Y1, Y2 ... Yk then begin pop X from the stack; push Yk,Yk-1 ... Y1 onto the stack, with Y1 on top;
output the production X → Y1, Y2 ... Yk
end
else error()
until X = $ /* stack is empty */
```

Predictive parsing table construction :

The construction of a predictive parser is aided by two functions associated with a grammar G :

1. FIRST
2. FOLLOW

Rules for first( ):

1. If X is terminal, then FIRST(X) is {X}.
2. If $X \rightarrow \epsilon$ is a production, then add ϵ to FIRST(X).
3. If X is non-terminal and $X \rightarrow a\,\alpha$ is a production then add a to FIRST(X).
4. If X is non-terminal and $X \rightarrow Y_1, Y_2 \ldots Y_k$ is a production, then place a in FIRST(X) if for some i, a is in FIRST(Yi) , and ϵ is in all of FIRST (Y_1), ..., FIRST(Y_{i-1}); that is, $Y_1, \ldots Y_{i-1} \Rightarrow \epsilon$. If ϵ is in FIRST(Y_j) for all j = 1, 2, .., k, then add ϵ to FIRST (X).

Rules for follow( ) :

1. If S is a start symbol, then FOLLOW(S) contains $.
2. If there is a production $A \rightarrow \alpha B \beta$, then everything in FIRST(β) except ϵ is placed in follow(B).
3. If there is a production $A \rightarrow \alpha B$, or a production $A \rightarrow \alpha B \beta$ where FIRST(β) contains ϵ, then everything in FOLLOW (A) is in FOLLOW (B).

Algorithm for construction of predictive parsing table :

- **Input :** Grammar G
- **Output :** Parsing table M
- **Method :**
  1. For each production $A \rightarrow \alpha$ of the grammar, do steps 2 and 3.
  2. For each terminal a in FIRST(α), add $A \rightarrow \alpha$ to M[A, a].
  3. If ϵ is in FIRST(α), add $A \rightarrow \alpha$ to M[A, b] for each terminal b in FOLLOW(A). If ϵ is in FIRST(α) and $ is in FOLLOW(A) , add $A \rightarrow \alpha$ to M[A, $].
  4. Make each undefined entry of M be error.

Example :

Consider the following grammar :

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid id$

After eliminating left-recursion the grammar is

$E \rightarrow TE'$

$E' \rightarrow +TE' \mid \in$

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \in$

$F \rightarrow (E) \mid id$

First( ) :

FIRST(E) = { (, id}

FIRST(E') = {+, ∈ }

FIRST(T) = { (, id}

FIRST(T') = {*, ∈ }

FIRST(F) = { (, id }

Follow( ) :

FOLLOW(E) = { $,) }

FOLLOW(E') = { $,) }

FOLLOW(T) = { +, $,) }

FOLLOW(T') = { +, $,) }

FOLLOW(F) = {+, * , $,) }

Predictive parsing Table :

Non terminal	id	+	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \in$	$E' \rightarrow \in$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \in$	$T' \rightarrow *FT'$		$T' \rightarrow \in$	$T' \rightarrow \in$
F	$F \rightarrow id$			$F \rightarrow (E)$		

Stack Implementation :

Stack	Input	Output
$E	id + id * id $	
$E' T	id + id * id $	$E \rightarrow TE'$

Stack	Input	Output
$E' T'F	Id + Id * Id $	T → FT'
$E' T' Id	Id + Id * Id $	F → Id
$E' T'	+Id * Id $	
$E'	+Id * Id$	T' → ∈
$E' T+	+Id * Id$	E'→ TE'
SE'T	id * Id$	
$E' T' F	Id * Id $	T → FT'
$E'T'id	id * Id $	F → id
$E' T'	*Id $	
$E' T' F*	*id $	T' → *FT'
$E' T' F	Id $	
$E' T' id	id $	F → id
$E' T'	$	
$E'	$	T' → ∈
$	$	E' → ∈

LL(1) grammar :

The parsing table entries are single entries. So each location has not more than one entry. This type of grammar is called LL(1) grammar.

Consider this following grammar :

S → iEtS | iEtSeS| a

E → b

After eliminating left factoring, we have

S → iEtSS'|a

S' → eS | ε

E → b

To construct a parsing table, we need FIRST() and FOLLOW() for all the non-terminals.

FIRST(S) = { i, a }

FIRST(S') = {e, ∈ }

FIRST(E) = { b}

FOLLOW(S) = { $, e }

FOLLOW(S') = { $, e }

FOLLOW(E) = {t}

Parsing table :

Non-terminal	a	b	e	i	t	$
S	S → a			S → iTtSS'		
S'			S' → eS S → ε			S' →ϵ
E		E → b				

- Since there are more than one production, the grammar is not LL(1) grammar.
- Actions performed in predictive parsing :
  1. Shift
  2. Reduce
  3. Accept
  4. Error
- **Implementation of predictive parser :**
  1. Elimination of left recursion, left factoring and ambiguous grammar.
  2. Construct FIRST() and FOLLOW() for all non-terminals.
  3. Construct predictive parsing table.
  4. Parse the given input string using stack and parsing table.

2.2.2(B) Bottom-Up Parsing

Constructing a parse tree for an input string beginning at the leaves and going towards the root is called bottom-up parsing. A general type of bottom-up parser is a shift-reduce parser.

Shift-reduce parsing :

Shift-reduce parsing is a type of bottom-up parsing that attempts to construct a parse tree for an input string beginning at the leaves (the bottom) and working up towards the root (the top).

Example :

Consider the grammar :

S → aABe

A → Abc | b

B → d

The sentence to be recognized is abbcde.

Reduction (leftmost)	Rightmost derivation
abbcde (A → b)	S → aABe
aAbcde(A → Abc)	→ aAde
aAde (B → d)	→ aAbcde
aABe (S → aABe)	→ abbcde
S	

The reductions trace out the right-most derivation in reverse.

Handles :

A handle of a string is a substring that matches the right side of a production, and whose reduction to the non-terminal on the left side of the production represents one step along the reverse of a rightmost derivation.

Example :

Consider the grammar :

$E \rightarrow E + E$

$E \rightarrow E * E$

$E \rightarrow (E)$

$E \rightarrow id$

And the input string $id_1 + id_2 * id_3$

The rightmost derivation is :

$E \rightarrow E + E$

$\rightarrow E + E * E$

$\rightarrow E + E * id_3$

$\rightarrow E + id_2 * id_3$

$\rightarrow id_1 + id_2 *$

In the above derivation the underlined substrings are called handles.

Handle pruning :

A rightmost derivation in reverse can be obtained by "handle pruning". (i.e.) if w is a sentence or string of the grammar at hand, then w = γn, where γn is the nth rightsentinel form of some rightmost derivation.

Actions in shift-reduce parser :

- **shift** - The next input symbol is shifted onto the top of the stack.
- **reduce** - The parser replaces the handle within a stack with a non-terminal.
- **accept** - The parser announces successful completion of parsing.
- **error** - The parser discovers that a syntax error has occurred and calls an error recovery routine.

Conflicts in shift-reduce parsing :

There are two conflicts that occur in shift-reduce parsing :

1. **Shift-reduce conflict :** The parser cannot decide whether to shift or to reduce.
2. **Reduce-reduce conflict :** The parser cannot decide which of several reductions to make.

Stack implementation of shift-reduce parsing :

Stack	Input	Action
$	$id_1 + id_2 * id_3$ $	Shift
$$id_1$	$+id_2 * id_3$ $	Reduce by E → id
$E	$+id_2 * id_3$ $	Shift
$ E+	$id_2 * id_3$ $	Shift
$E+$id_2$	$*id_3$ $	reduce by E → id
$ E + id_2	$* id_3$ $	Shift
$E + E*	id_3 $	Shift
$E + E* id_3	$	reduce by E → id
$E + E * E	$	Reduce by E → E * E
$E + E	$	Reduce E → E + E
$E	$	Accept

1. **Shift-reduce conflict :**

Example :

Consider the grammar :

E → E + E | E * E | id and input id + id * id

Stack	Input	Action	Stack	Input	
$E + E	*id $	Reduce by E → E + E	$E + E	*id $	Shift
$E	*id $	Shift	$E + E*	id $	Shift
$E*	id $	Shift	$E + E * id	$	Reduce by E → id
$E * id	$	Reduce by E → id	$E + E * E	$	Reduce by E → E * E
$ E * E	$	Reduce by E → E * E	$E + E	$	Reduce by E → E * E
$E			$E		

2. Reduce-reduce conflict :

Consider the grammar :

$M \rightarrow R + R \mid R + c \mid R$

$R \rightarrow c$

and input c+c

Stack	Input	Action	Stack	Input	Action
$	c + c $	Shift	$	c + c $	Shift
$c	+c$	Reduce by R → c	$c	+ c $	Reduce by R → c
$R	+c$	Shift	$R	+c$	Shift
$R+	C$	Shift	$R+	C$	Shift
$ R+c	$	Reduce by R →c	$R + c	$	Reduce by M → R + C
$ R+R	$	Reduce by M → R + R	$M	$	
$M	$				

Viable prefixes :

- α is a viable prefix of the grammar if there is w such that αw is a right
- The set of prefixes of right sentinel forms that can appear on the stack of a shift-reduce parser are called viable prefixes
- The set of viable prefixes is a regular language.

Operator-precedence parsing :

An efficient way of constructing shift-reduce parser is called operator-precedence parsing. Operator precedence parser can be constructed from a grammar called Operator-grammar. These grammars have the property that no production on right side is ε or has two adjacent non-terminals.

Example :

- Consider the grammar :

  $E \rightarrow EAE \mid (E) \mid -E \mid id$

  $A \rightarrow + \mid - \mid * \mid / \mid \uparrow$

- Since the right side EAE has three consecutive non-terminals, the grammar can be written as follows :

  $E \rightarrow E + E \mid E - E \mid E * E \mid E / E \mid E \uparrow E \mid -E \mid id$

Operator precedence relations :

- There are three disjoint precedence relations namely

  <· less than

  =· equal to

  ·> greater than

- The relations give the following meaning :

  a <· b - a yields precedence to b

  a = b - a has the same precedence as b

  a ·> b - a takes precedence over b

- **Rules for binary operations :**

  1. If operator θ1 has higher precedence than operator θ2, then make

     θ1 ·>θ2 and θ2 <· θ1

  2. If operators θ1 and θ2, are of equal precedence, then make

     θ1 ·>θ2 and θ2 ·>θ1 if operators are left associative

     θ1 <· θ2 and θ2 <· θ1 if right associative

  3. Make the following for all operators θ :

     θ <· id ,id ·> θ

     θ <· (, (<· θ

) ·>θ, θ ·>)

     θ ·>$, $<· θ

  Also make

  (=) , (<· (,) ·>) , (<· id, id ·>) , $ <· id , id ·> $,

Example :

- Operator-precedence relations for the grammar

  E → E + E | E – E | E * E |E / E| E↑E | (E) | – E | id is given in the following table assuming

  1. ↑ is of highest precedence and right-associative
  2. * and / are of next higher precedence and left-associative, and
  3. + and – are of lowest precedence and left- Note that the blanks in the table denote error entries.

Table 2.2.1 : Operator-precedence relations

	+	–	*	/	↑	id	(	)	$
+	·>	·>	<·	<·	<·	<·	<·	·>	·>
–	·>	·>	<·	<·	<·	<·	<·	·>	·>
*	·>	·>	·>	·>	<·	<·	<·	·>	·>
/	·>	·>	·>	·>	<·	<·	<·	·>	·>
↑	·>	·>	·>	·>	<·	<·	<·	·>	·>
id	·>	·>	·>	·>	·>			·>	·>
(	<·	<·	<·	<·	<·	<·	<·	=	
)	·>	·>	·>	·>	·>			·>	·>
$	<·	<·	<·	<·	<·	<·	<·		

Operator precedence parsing algorithm :

- **Input :** An input string w and a table of precedence relations.
- Output : If w is well formed, a skeletal parse tree ,with a placeholder non-terminal E labeling all interior nodes; otherwise, an error indication.
- **Method :** Initially the stack contains $ and the input buffer the string w$. To parse, we execute the following program :

```
(1) Set ip to point to the first symbol of w$;
(2) repeat forever
(3) if $ is on top of the stack and ip points to $ then
(4) return else begin
(5) let a be the topmost terminal symbol on the stack and let b be the symbol pointed to by ip;
(6) if a <. b or a = b then begin
(7) push b onto the stack;
(8) advance ip to the next input symbol; end;
(9) else if a .>b then /*reduce*/
(10) repeat
(11) pop the stack
(12) until the top stack terminal is related by <. to the terminal most recently popped
(13) else error( ) end.
```

Stack implementation of operator precedence parsing :

- Operator precedence parsing uses a stack and precedence relation table for its implementation of above algorithm. It is a shift-reduce parsing containing all four actions shift, reduce, accept and error.
- The initial configuration of an operator precedence parsing is
  - STACK : $
  - INPUT : w$

  where w is the input string to be parsed.

  Example :

  Consider the grammar E → E + E | E – E | E * E | E / E | E ↑ E | (E) | id. Input string is id + id * id. The implementation is as follows :

STACK	INPUT	COMMENT
$	<· id + id * id$	shift id
$ id	-> + id * id $	Pop the top of the stack id
$	<· + id * id $	Shift +
$+	<· id * id $	Shift id
$ + id	-> *id $	pop id

STACK	INPUT	COMMENT
$+	<· *id $	Shift *
$ + *	<· id $	Shift id
$ + *id	·> $	pop id
$ + *	·> $	pop *
$+	·> $	pop +
$	$	Accept

Advantages of operator precedence parsing :

1. It is easy to implement.
2. Once an operator precedence relation is made between all pairs of terminals of a grammar, the grammar can be ignored. The grammar is not referred anymore during implementation.

Disadvantages of operator precedence parsing :

1. It is hard to handle tokens like the minus sign (–) which has two different precedence.
2. Only a small class of grammar can be parsed using operator-precedence parser.

2.2.2(C) LR Parsers

An efficient bottom-up syntax analysis technique that can be used CFG is called LR(k) parsing. The 'L' is for left-to-right scanning of the input, the 'R' for constructing a rightmost derivation in reverse, and the 'k' for the number of input symbols. When 'k' is omitted, it is assumed to be 1.

Advantages of LR parsing :

1. It recognizes virtually all programming language constructs for which CFG can be written.
2. It is an efficient non-backtracking shift-reduce parsing method.
3. A grammar that can be parsed using LR method is a proper superset of a grammar that can be parsed with predictive parser
4. It detects a syntactic error as soon as possible.

Drawbacks of LR method :

It is too much of work to construct a LR parser by hand for a programming language grammar. A specialized tool, called a LR parser generator, is needed. Example : YACC.

Types of LR parsing method :

1. SLR- Simple LR

   Easiest to implement, least powerful.
2. CLR- Canonical LR

   Most powerful, most expensive.
3. LALR- Look-Ahead LR

   Intermediate in size and cost between the other two methods.

The LR parsing algorithm :

The schematic form of an LR parser is as follows :

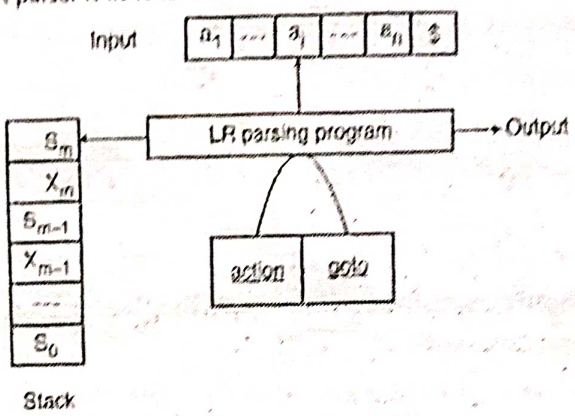


Fig. 2.2.5 : Model of an LR parser

- It consists of an input, an output, a stack, a driver program, and a pa parts (action and goto).
  - The driver program is the same for all LR parser.
  - The parsing program reads characters from an input buffer one at a time.
  - The program uses a stack to store a string of the form $s_0 X_1 s_1 X_2 s_2 \ldots X_m s_m$, where s_m is on top. Each Xi is a grammar symbol and each si is a state.
  - The parsing table consists of two parts : action and goto functions.
- **Action :** The parsing program determines s_m, the state currently on top of stack, and ai, the current input symbol. It then consults action[s_m, a_i] in the action table which can have one of four values :
  1. shift s, where s is a state,
  2. reduce by a grammar production A → β,
  3. accept,
  4. error.
- **Goto :** The function goto takes a state and grammar symbol as arguments and produces a state.

LR Parsing algorithm :

- **Input :** An input string w and an LR parsing table with functions action and goto for grammar G.
- **Output :** If w is in L(G), a bottom-up-parse for w; otherwise, an error indication.
- **Method :** Initially, the parser has s_0 on its stack, where s_0 is the initial state, and w$ in the input buffer. The parser then executes the following program :

```
set ip to point to the first input symbol of w$; repeat forever begin
    lets be the state on top of the stack and
a the symbol pointed to by ip;
    if action[s, a] = shift s' then begin
    push a then s' on top of the stack; advance ip to the next input symbol end
    else if action[s, a] = reduce A →β then begin pop 2 * | β | symbols off the stack;
```

let s' be the state now on top of the stack; push A then goto[s', A] on top of the stack; output the production A→β

end

else if action[s, a] = accept then

return

else error( )

end

2.2.2(D) CYK Algorithm

- The CYK algorithm is a context-free grammar parsing algorithm. Used to check if a given string can be inferred from the language produced by a given grammar.
- Also known as the membership algorithm, because it indicates whether a given string is a member of a given grammar. It was originally developed by his three Russian scientists named Cocke, Younger and Kasami, hence the name CYK.
- It is used to decide whether a given string belongs to the language of grammar or not.
- The CYK algorithm requires the structure of the grammar to be in Chomsky normal form. The CYK algorithm also uses dynamic programming or table fill algorithms.
- A grammar is CNF if each rule has one of the following forms :
  - $A \rightarrow BC$ (at most two variables on the right-hand side)
  - $A \rightarrow a$ (a single terminal on the right-hand side)
  - $S \rightarrow \varphi$ (null string)
- If the given Grammar is not in the CNF form, convert it to the CNF form before applying the CYK Algorithm.

Algorithm :

- In the CYK algorithm, Construct a triangular table of the length of your given string.

Table 2.2.1 : Table for string 'w' that has length 3

x(1,3)		
x(1,2)	x(2,3)	
x(1,1)	x(2,2)	x(2,3)
w1	w2	w3

- Each row corresponds to the length of the substrings of the given word 'w':
  - The bottom row will have strings of length 1.
  - The second row from the bottom will have strings of length 2 and so on.
- $X(i, i)$ is a set of variables A such that $A \rightarrow w_i$ is a production of grammar G, where w_i is part of the given string 'w' or the whole string.
- Compare at most n pairs of previously computed sets. For strings with a length of 2, compare two pairs, and for strings with a length of 3 compare three sets. In this way, the next sets (A) can be computed, which are derived from the given grammar as per the following formula :

$(X(i, i), X((i + 1), j), (X (i, i + 1), X (i + 2, j)) \ldots (X (i, j - 1), Xj)$

- In the equation, $X(i, j)$ refers to the set of variables derived from the production rules.
- **Rule** : If the top row consists of the starting variable of the grammar, then the given string can be derived from it.

Example :

- Let's look at an example to get a better understanding of the algorithm.
- Consider the following CNF grammar G :
  - S → AB | BC
  - A → BA | a
  - B → CC | b
  - C → AB | a
- The given word ***w=baaba***.

Step 1 : Constructing the triangular table

5	x(1,5)				
4	x(1,4)	x(2,5)			
3	x(1,3)	x(2,4)	x(3,5)		
2	x(1,2)	x(2,3)	x(3,4)	x(4,5)	
1	x(1,1)	x(2,2)	x(3,3)	x(4,4)	x(5,5)
	'b'	'a'	'a'	'b'	'a'

Step 2 : Populating the table

- The first row is computed simply by looking at the grammar to see which production is producing the string of length 1 for the given word 'w'.
- The second row is computed by comparing the two strings that were computed previously. So we can have "w": "*ba*', "*aa*', '*ab*', and '*ba*'.
- For the third row, there are three possible substrings of length three in the given word, namely '*w*':'*baa*', '*aab*', and '*aba*'. For the substring '*baa*', there are two possibilities:['*b*','*aa*'] and ['*ba*','*a*'].

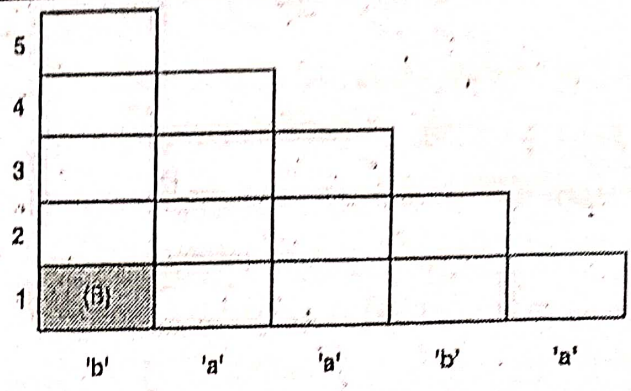
  Finding the sets to compute these: ['*b*','*aa*'] = {*B*} {*B*} ['*ba*','*a*'] = {*S*, *A*} *A*, C}
- We take the union of these as follows:

  {B} {B} U {S,A} {A,C} {BB} U {SA, SC, AA, AC} → {BB, SA, SC, AA, AC},
- Then, we look in the grammar rules for populating the table.
- We'll repeat this for all substrings with lengths greater than 33.
- The illustration below shows how the table is populated.

Populating Row 1

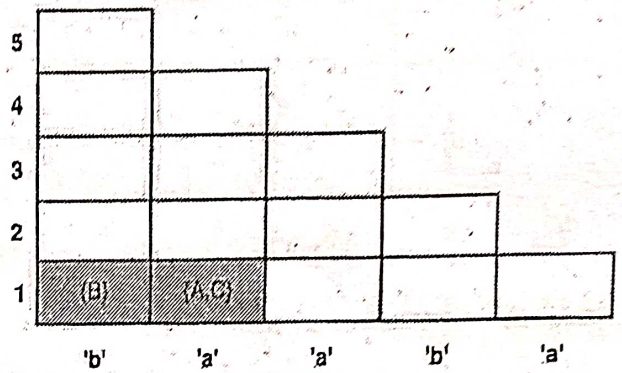
If we see the Grammar given the string 'b' is produced by variable B. Write B in the first box of the row 1 as :

'b' : X(1,1) = {B}



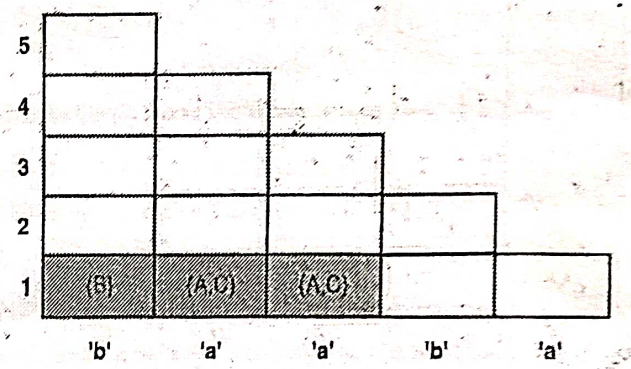
- Similarly for string 'a', it is produced by A and C both so

'a' : X(2,2) = {A, C}



for string 'a', again it is produced by A and C as follows :

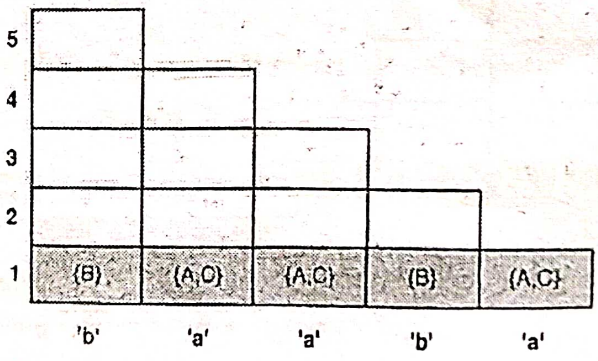
'a' : X(3,3) = {A, C}



Similarly for string 'b', 'a' :

'b' : X(4,4) = {B}

'a' : X(5,5) = {A ,C}

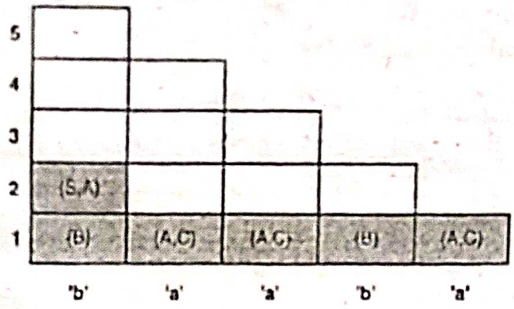


Populating row 2 :

- For row 2, strings of length 2 are calculated like 'ba', 'aa', 'ab', & 'ba':
- For string 'ba': X(1, 2) : (X(i, j), X(i + 1, j)) = X(2, 2))

  = {B} {A,C} = {BA, BC}
- If you see the production rules, BA and BC are produced by Var A and S respectively

So, X(1, 2) = {S ,A}

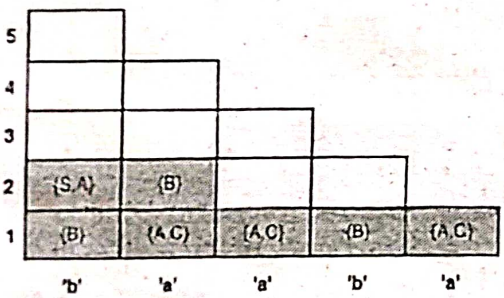


For string 'aa' : X(2, 3) :

(X (i, j), X(i + 1, j)) = X(2, 2), X(3, 3)) = {A,C} {A,C} = AA, AC, CA, CC}

If you see the production rules, CC is produced by Var B and AA, AC and CA respectively

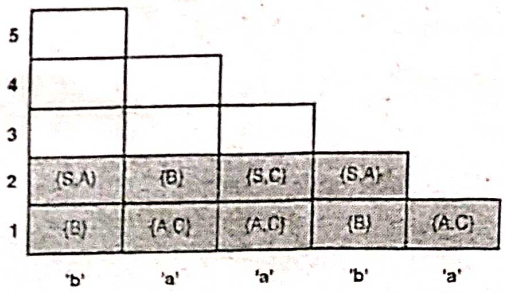
So, X(2, 3) = {B}



Similarly, for strings 'ab' and 'ba':

'ab' : X(3, 4) = {S, C}

'ba' : X(4, 5) = {S, A}



Populating row 3

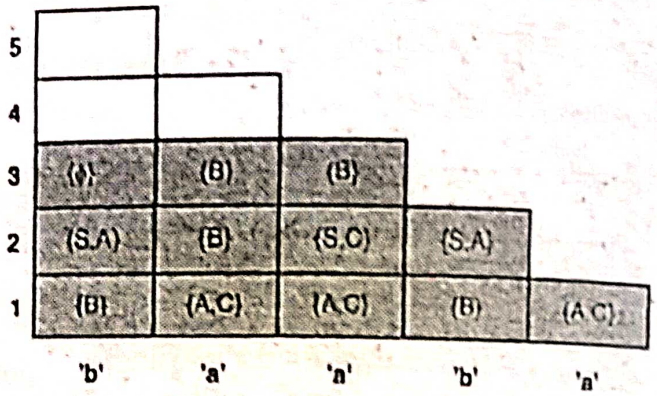
For row 3, strings of length3 are calculated like 'baa', 'aab' and 'aba'.

As we have already seen in row 2 the explanation so,

'baa' : X(1,3) = {Ø}

'aab' : X(2,4) = {B}

'aba' : X(3,5) = {B}



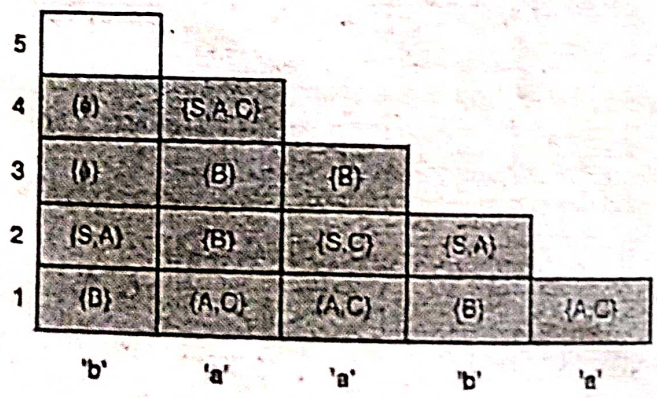
Populating row 4

For row 4, strings of length 4 are calculated like 'baab', 'aaba'

So;

'baab' : X (1, 4) = {Ø}

'aaba' : X (2, 5) = {S, A, C}



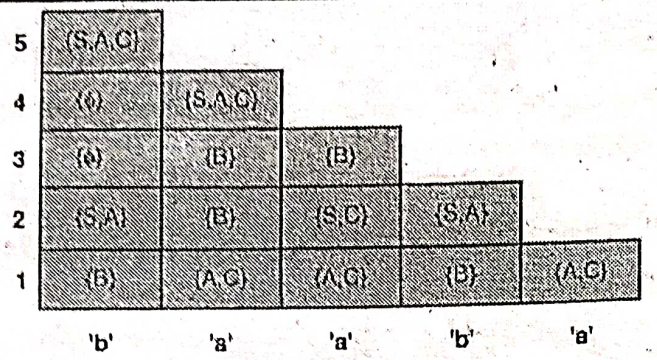
Populating row 5

For row 5, strings of length 5 are calculated like 'baaba'

Here union of X(1, 4) and X(2, 5) is taken.

So,

$$X(1,5) = X(1,4) \cup X(2,5)$$
$$= \{Ø\} \cup \{S, A, C\}$$
$$= \{S, A, C\}$$



This is the final Triangular table.

Step 3 : Check the table

Finally, we need to check if the given word is in the language of the grammar. As we can see in the table, the cell $X(1, 5) = (S, A, C)$. S is present in the final box. So, we can say that 'baaba' $\in$ L(G).

Time complexity

The running time of the algorithm is O (n^3).

2.2.3 Probabilistic Context-free Grammars and Statistical Parsing

- In this section, we want to escape the linear tyranny of these n-gram and HMM tagging models and start exploring more complex concepts in grammars.
- Even in the most traditional grammatical forms, syntax is intended to show more than just linear order.
- Languages have complex recursive structures, and such a tree-based model can capture this.
- For example, Kupiec (1992b) states that his HMM-based tagger has the following structural problems :

The velocity of the seismic waves rises to .

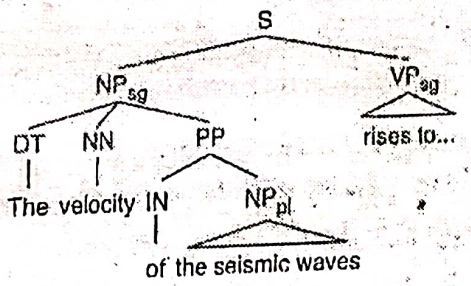


Fig. 2.2.6

- The verb agrees in number with the noun velocity which is the head of the preceding noun phrase, and not with the noun that linearly precedes it.

PCFG

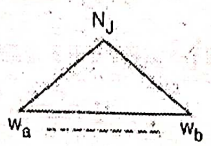
- The simplest probabilistic model for recursive integration is PCFG, PROBABILISTIC a Probabilistic (sometimes called Stochastic) Context-free grammar CONTEXTFREE _ G RAMMAR is simply a CFG with probabilities added to the rules, which denotes different rewrite possibilities.
- PCFG is the simplest and most natural probabilistic model for tree structures, the underlying math is well understood, the algorithms for them are a natural evolution of the algorithms used with the HMM and PCFG provide a sufficiently general computational device that they can simulate various other forms of probabilistic conditioning.

- A (generative) PCFG consists of
  - Terminals $w^1, w^2, \ldots w^V$
  - Nonternimals $N^1, N^2, \ldots, N^n$
  - Start symbol N^1
  - Rules $N^i \rightarrow \xi^j$ where ξ^j is a sequence of terminals and nonterminals
  - Rule probabilities such that $\forall_i, \sum_j P(N^i \rightarrow \xi^j) = 1$

Notation for the PCFG :

Notation	Meaning
G	Grammar (PCFG)
$\mathcal{L}$	Language (Generated or accepted by a grammar) Parse tree
$(N^1, \ldots, N^n)$	Nonterminal vocabulary (N^1 is start symbol)
$(w^1, \ldots, w^V)$	Terminal vocabulary
$W_1, \ldots, W_m$	Sentence to be parsed
N^j_{pq}	Non terminal N^j spans positions p through q in string
$\alpha_j(p, q)$	Outside probabilities
$\beta_j(p, q)$	Inside probabilities

- Note that when we write $P(N^i - \xi^j)$ we always mean $P(N^i - \xi^j \mid N^i)$ That is, we are giving the probability distribution of the daughters for a certain head. Such a grammar can be used either to parse or generate sentences of the language, and we will switch between these terminologies quite freely.
- Before parsing sentences with a PCFG, we need to establish some notation. We will represent the sentence to be parsed as a sequence of words $w_1, \ldots, W_n$ and use w_{ab} to denote the subsequence $w_a \ldots w_b$.
- We denote a single rewriting operation of the grammar by a single arrow $\rightarrow$. If as a result of one or more rewriting operations we are able to rewrite a non terminal N^j as a sequence of $w_a, \ldots w_b$, then we will say that N^j dominates the words $w_a \ldots w_b$, and write either $N^j \Rightarrow w_a \ldots w_b$, or yield $(N^j) \Rightarrow w_a \ldots w_b$. This following sub tree illustrates this situation, a sub tree with root non terminal N^j dominating all and only the words from $w_a \ldots w_b$ in the string :



- To say that a non terminal N^j spans positions a through b in the string, but not to specify what words are actually contained in this subsequence, we will write N^j_{ab}
- The probability of a sentence (according to a grammar G) is given by where t is a parse tree of the sentence.

$$P(w_1 n) = \sum_t P(w_{1,n}, t)$$

- A simple Probabilistic Context Free Grammar (PCFG) consist of the non terminals S, NP, PP, VP, P, V. We adopt the common convention whereby the start symbol N_1 is denoted by S.

- The terminals are the words in italics. The table shows the grammar rules and their probabilities. The slightly unusual NP rules have been chosen so that this grammar is in Chomsky Normal Form.

$$= \sum_{(t + \text{yield}(t) = w_{1,m})} P(t)$$

- Moreover, it is easy to find the probability of a tree in a PCFG model. One just multiplies the probabilities of the rules that built its local subtrees.
- **Example :** astronomers saw stars with ears

S	→	NP VP	1.0	NP	→	NP PP	0.4
PP	→	P NP	1.0	NP	→	astronomers	0.1
VP	→	V NP	0.7	NP	→	Ears	0.18
VP	→	VP PP	0.3	NP	→	Saw	0.04
P	→	With	1.0	NP	→	stars	0.18
V	→	Saw	1.0	NP	→	telescopes	0.1

The sentence has two parses with probabilities

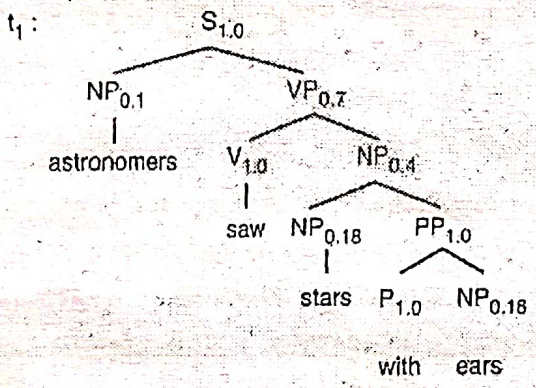


Fig. 2.2.7

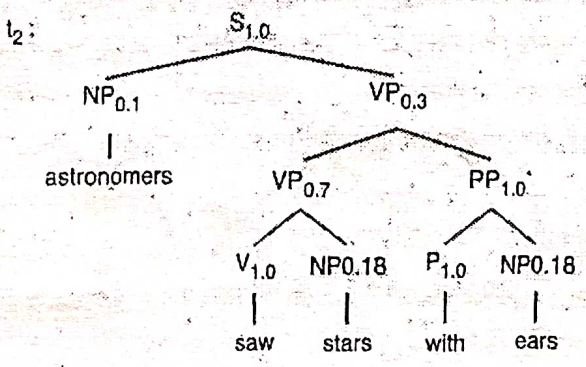


Fig. 2.2.8

$P(t_1) = 1.0 \times 0.1 \times 0.7 \times 1.0 \times 0.4 \times 0.18 \times 1.0 \times 1.0 \times 0.18 = 0.0009072$

$P(t_2) = 1.0 \times 0.1 \times 0.3 \times 0.7 \times 1.0 \times 0.18 \times 1.0 \times 1.0 \times 0.18 = 0.0006804$

$P(w_{15}) = P(t_1) + P(t_2) = 0.0015876$

Assumptions of PCFGs :

- The conditions that we need are :
  - **Place invariance :** The probability of a subtree does not depend on where in the string the words dominates are (this is like time invariance in HMMs)

$$\forall k \; P\left(N^{j}_{k(k+c)} \rightarrow \xi\right) \text{ is tha same}$$

- **Context-free :** The probability of not dominated by the subtree.

$$P(N^{j}_{kl} \rightarrow \xi \mid \text{anything outside k through 1} = P(N^{j}_{kl} \rightarrow \xi))$$

- **Ancestor-free :** The probability of a subtree does not depend on nodes in the derivation outside the subtree.

$$P(N^{j}_{kl} \rightarrow \xi \mid \text{any ancestor nodes outside } N^{j}_{kl} = P(N^{j}_{kl} \rightarrow \xi))$$

- Using these conditions we can justify the calculation of the probability of a tree in terms of just multiplyin probabilities attached to rules. But to show an example, we need to be able to distinguish tokens of a nontermina Therefore, let the upper left index in 'N' be an arbitrary identifying index for a particular token of a nonterminal.

Features of PCFGs :

Here we give some reasons to use a PCFG, and also some idea of their limitations :

- As grammars expand to give coverage of a large and diverse corpus of text, the grammars become increasingly ambiguous. There start to be many structurally different parses for most word sequences. A PCFG gives some idea of the plausibility of different parses.
- A PCFG does not give a very good idea of the plausibility of different parses, since its probability estimates are based purely on structural factors, and do not factor in lexical co-occurrence.
- PCFGs are good for grammar induction. CFGs cannot be learned (in the sense of identification in the limit - that is whether one can identify a grammar if one is allowed to see as much data produced by the grammar as one wants) without the use of negative evidence (the provision of ungrammatical examples). But PCFGs can be learned from positive data alone.
- Robustness. Real text tends to have grammatical mistakes, disfluenties, and errors. This problem can be avoided to some extent with a PCFG by ruling out nothing in the grammar, but by just giving implausible sentences a low probability.
- PCFGs give a probabilistic language model for English (whereas a CFG does not).
- The predictive power of a PCFG as measured by entropy tends to be greater than that for a finite state grammar (i.e., an HMM) with the same number of parameters.
- In practice, a PCFG is a worse language model for English than an n-gram model (for $n > 1$). An n-gram model takes some local lexical context into account, while a PCFG uses none.
- PCFGs are not good models by themselves, but we could hope to combine the strengths of a PCFG and a trigram model
- PCFGs have certain biases, which may not be appropriate. All else being equal, in a PCFG, the probability of a smaller tree is greater than a larger tree. This is not totally wild - it is consonant with Minimal Attachment heuristic - but it does not give a sensible model of actual sentences, which peak in frequency at some intermediate length.

2.3 Semantic Analysis

- Semantic analysis is a subfield of natural language processing (NLP) that attempts to understand the meaning of natural language. Understanding natural language can seem like a simple process for us humans. However, human language is so complex and subjective that interpreting it is a fairly complex task for machines. Semantic analysis of natural language captures the meaning of a given text considering context, sentence logical structure, and grammatical role.
- Semantic analysis begins with lexical semantics, which examines the meaning of individual words (that is, dictionary definitions). Semantic analysis then examines the relationships between individual words and analyzes the meaning of words that combine to form sentences. This analysis gives you a clear understanding of the words in context. For example, it provides context for understanding the following statement :
  - "The boy ate the apple" defines an apple as a fruit.
  - "The boy went to Apple" defines Apple as a brand or store.
  - part of semantic analysis
- Semantic analysis of natural language can be divided into two main parts of his :
  1. **Lexical Semantic Analysis :** Lexical semantic analysis involves understanding the meaning of each word in a text individually. This basically refers to dictionary lookup. It means that the words in the text are destined to be carried.
  2. **Compositional semantic analysis :** Knowing the meaning of every word in a text is important, but not sufficient to fully understand the meaning of the text.

2.3.1 Lexical Semantic

- The lexicon has a highly systematic structure that governs what words can mean, and how they can be used. This structure consists of relations among words and their meanings, as well as the internal structure of individual words. The study of this systematic, meaning related, structure is called **Lexical Semantics.**
- Before moving on, we will first introduce a few new terms, since the ones we have been using thus far are entirely too vague. In particular, the word *word* has by now been used in so many different ways that it will prove difficult to make unambiguous use of it in this chapter. Instead, we will focus on the notion of a **lexeme**, an individual entry in the lexicon.
- A lexeme should be thought of as a pairing of a particular orthographic and phonological form with some form of symbolic meaning representation. The **lexicon** is therefore a finite list made up of lexemes. When appropriate, we will use the terms orthographic form, and phonological form, to refer to the appropriate form part of this pairing, and the term **sense** to refer to a lexeme's meaning component.

Lemmas and Senses

- Let's start by looking at how one word (we'll choose mouse) might be defined in a dictionary :

  mouse (N)

  1. any of numerous small rodents...
  2. a hand-operated device that controls a cursor...

- Here the form mouse is the lemma, also called the citation form. The form *mouse* would also be the lemma for the word *mice*; dictionaries don't have separate definitions for inflected forms like *mice*. Similarly *sing* is the lemma for *sing, sang, sung*. In many languages the infinitive form is used as the lemma for the verb, so Spanish *dormir* "to sleep" is the lemma for *duermes* "you sleep". The specific forms *sung* or *carpets* or *sing* or *duermes* are called **wordforms**.
- As the example above shows, each lemma can have multiple meanings; the lemma mouse can refer to the rodent or the cursor control device. We call each of these aspects of the meaning of mouse a word sense. The fact that lemmas can be polysemous (have multiple senses) can make interpretation difficult.

2.3.1(A) Ambiguous Words

- **Ambiguous words** are words that have more than one meaning. If a word is in isolation, or if the sentence cannot clarify the ambiguity, that word remains ambiguous.
- Words are ambiguous: the same word can be used to mean different things. Let's start by looking at how one word (we'll choose mouse) might be defined in a dictionary: the word "mouse" has (at least) two meanings :
  1) a small rodent, or
  2) a hand-operated device to control a cursor.

  Or the word

  "bank" can mean :
  1) a financial institution or
  2) a sloping mound.
- ***Jake saw her duck.*** **Duck is an example of an ambiguous word as it can mean "a bird" or "bend."** The word remains ambiguous because the sentence does not provide enough context for the word duck.
- **You heard it right; Bill is running. In this example,** the word running is an ambiguous word. In this example, the word *running* is an ambiguous word. Also, the sentence does not clarify the ambiguity of this word. *Is Bill running as in exercising? or Is Bill running for office?* are some of the questions that come to a reader's mind when facing this example.

Ambiguity :

- There are two types of ambiguity : Genuine ambiguities, where a sentence really can have two different meanings to an intelligent hearer, and "computer" ambiguities, where the meaning is entirely clear to a hearer but a computer detects more than one meaning.
- Genuine ambiguity is not a serious problem for NLP problems; it's comparatively rare, and you can't expect computers to do better with natural language than people.
- Computer ambiguity is a very serious problem; it is extremely common, and it is where computers do much much worse than humans..

Types of ambiguity :

1. **Lexical ambiguity :** Words have multiple meanings. "I saw a bat." bat = flying mammal / wooden club? saw = past tense of "see" / present tense of "saw" (to cut with a saw.)

2. **Syntactic ambiguity :** A sentence has multiple parse trees. Particularly common sources of ambiguity in English are :

- **Phrase attachment :** "Mary ate a salad with spinach from Califonia for lunch on Tuesday." "with spinach" can attach to "salad" or "ate" "from California" can attach to "spinach", "salad", or "ate". "for lunch" can attach to "California", "spinach", "salad", or "ate" and "on Tuesday" can attach to "lunch", "California", "spinach", "salad" or "ate". (Crossovers are not allowed, so you cannot both attach "on Tuesday" to "spinach" and attach "for lunch" to salad. Nonetheless there are 42 possible different parse trees.)
- **Conjunction :** "Mary ate a salad with spinach from Califonia for lunch on Tuesday and Wednesday." "Wednesday" can be conjoined with salad, spinach, California, lunch, or Tuesday.
- **Noun group structure :** English allows long series of nouns to be strung together using the enormously ambiguous rule NG -> NG NG. E.g. "New York University Martin Luther King Jr. scholarship program projects coordinator Susan Reid". Even taking "New York" "Martin Luther King Jr." and "Susan Reid" to be effectively single elements, this is 8 elements in a row, and has 429 possible parses.

3. **Semantic ambiguity :**

- Even after the syntax and the meanings of the individual words have been resolved, there are two ways of reading the sentence.
- "Lucy owns a parrot that is larger than a cat", "a parrot" is extensionally quantified, "a cat" is either universally quantified or means "typical cats."
- **Other examples :** "The dog is chasing the cat." vs. "The dog has been domesticated for 10,000 years." In the first sentence, "The dog" means to a particular dog; in the second, it means the species "dog".
- "John and Mary are married." (To each other? or separately?) Compare "John and Mary got engaged last month. Now, John and Mary are married." vs. "Which of the men at this party are single? John and Jim are married; the rest are all available."
- "John kissed his wife, and so did Sam". (Sam kissed John's wife or his own?)
- Compare "Amy's car", "Amy's husband", "Amy's greatest fear", "Michaelangelo's David" etc.

4. **Anaphoric ambiguity :**

- A phrase or word refers to something previously mentioned, but there is more than one possibility.
- "Margaret invited Susan for a visit, and she gave her a good lunch." (she = Margaret; her = Susan)
- "Margaret invited Susan for a visit, but she told her she had to go to work" (she = Susan; her = Margaret.)
- "On the train to Boston, George chatted with another passenger. The man turned out to be a professional hockey player." (The man = another passenger).
- "Bill told Amy that he had decided to spend a year in Italy to study art." "That would be his life's work." (That = art) "After he had done that, he would come back and marry her." (That = spending a year in Italy) "That was the upshot of his thinking the previous night" (That = deciding) "That started a four-hour fight." (That = telling Amy)
- **Son :** I watched a guy do 50 pushups. Can you do that, dad?
- **Father :** Sure! Not to brag, but I could probably watch a guy do 100 pushups.

- In many cases, there is no explicit antecedent.
- "I went to the hospital, and they told me to go home and rest." (They = the hospital staff.)
- "I went to the hospital, and they told me to go home and rest." (They = the hospital staff)
- **Non-literal speech :** "The White House announced today that ..." ("White House" = the President's staff) (Mentonymy) "The price of tomatoes in Des Moines has gone through the roof" (= increased greatly) Metaphor.
- **Ellipsis :** The omission of words that are needed for grammatical completion, and are "understood". This is very common in speech, less so in writing. E.g. "I am allergic to tomatoes. Also fish." Understood as "I am also allergic to fish" rather than "Also, fish are allergic to tomatoes." "Mozart was born in Salzburg and Beethoven, in Bonn". Understood as "Mozart was born in Salzburg and Beethoven was born in Bonn"

2.3.2 Relations Among Lexemes & their Senses – Homonymy, Polysemy, Synonymy, Hyponymy

Word Senses :

- A sense (or word sense) is a discrete representation of one aspect of the meaning of a word. Loosely following lexicographic tradition, we represent each sense with a superscript: bank1 and bank2, mouse1 and mouse2. In context, it's easy to see the different meanings :

  mouse 1 : .....a mouse controlling a computer system in 1968.

  mouse 2 : .... a quiet animal like a mouse

  bank 1 : ... a bank can hold the investments in a custodial account ...

  bank 2 : ... as agriculture burgeons on the east bank, the river ...

- The senses of a word might not have any particular relation between them; it may be almost coincidental that they share an orthographic form.
- For example, the *financial institution* and *sloping mound* senses of bank seem relatively unrelated. In such cases we say that the two senses are homonyms, and the relation between the senses is one of homonymy.
- Thus bank1 ("financial institution") and bank2 ("sloping mound") are homonyms, as are the sense of bat meaning 'club for hitting a ball' and the one meaning 'nocturnal flying animal'. We say that these two uses of bank are homographs, as are the two uses of bat, because they are written the same.
- Two words can be homonyms in a different way if they are spelled differently but pronounced the same, like write and right, or piece and peace. We call these homophones; they are one cause of real-word spelling errors.

Relations between Senses

1. **Synonym :**
   - It is a relationship between two distinct lexemes with the same meaning (i.e. they can be substituted for one another in a given context without changing its meaning and correctness).

     e.g. I received a gift/present
   - The substitutability may not be valid for any context due to small semantic differences (e.g. *price/fare of a service – the bus fare/the ticket price*).
   - In general substitutability depends on the "semantic intersection" of the senses of the two lexemes and, in some cases, also by social factors (*father/dad*).

2. **Antonyms :**

- Whereas synonyms are words with identical or similar meanings, antonyms are words with an opposite meaning, like :

  long/short big/little fast/slow cold/hot dark/light

  rise/fall up/down in/out

- Two senses can be antonyms if they define a binary opposition or are at opposite ends of some scale. This is the case for long/short, fast/slow, or big/little, which are at opposite ends of the length or size scale.
- Also, It is the relation between two lexical items having symmetry between their semantic components relative to an axis. The scope of antonymy is as follows :
  - **Application of property or not** – Example is 'life/death', 'certitude/incertitude'
  - **Application of scalable property** – Example is 'rich/poor', 'hot/cold'
  - **Application of a usage** – Example is 'father/son', 'moon/sun'.

3. **Reversives**

- Another group of antonyms, reversives, describe change or movement in opposite directions, such as rise/fall or up/down.
- Antonyms thus differ completely with respect to one aspect of their meaning their position on a scale or their direction but are otherwise very similar, sharing almost all other aspects of meaning. Thus, automatically distinguishing synonyms from antonyms can be difficult.

4. **Hyponym and Hypernym**

- A word (or sense) is a **hyponym** of another word or sense if the first is more specific, denoting a subclass of the other.
- For example, car is a hyponym of vehicle, dog is a hyponym of animal, and mango is a hyponym of fruit.
- Conversely, we say that vehicle is a **hypernym** of car, and animal is a hypernym of dog. It is unfortunate that the two words (hypernym and hyponym) are very similar and hence easily confused; for this reason, the word superordinate is often used instead of hypernym.

Super ordinate	Vehicle	Fruit	Furniture	Mammal
Subordinate	Car	Mango	Chair	Dog

- We can define hypernymy more formally by saying that the class denoted by the superordinate extensionally includes the class denoted by the hyponym. Thus, the class of animals includes as members all dogs, and the class of moving actions includes all walking actions. Hypernymy can also be defined in terms of entailment.
- Hypernymy is useful for tasks like textual entailment or question answering; knowing that leukemia is a type of cancer, for example, would certainly be useful in answering questions about leukemia.

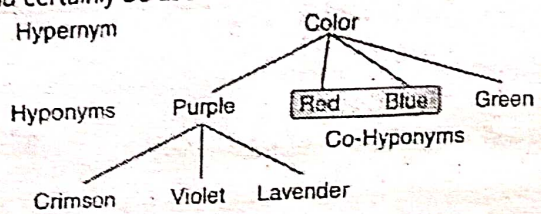


Fig. 2.3.1

5. **Meronym :**

- Another common relation is meronymy, the part-whole relatlon. A leg is part of a chair; a wheel is part of a car. We say that wheel is a meronym of car, and car is a holonym of wheel.
- Meronymy is not just a single relation but a bundle of different part-to-whole relationships.
- "In one ***context*** *finger* is an appropriate meronym of ***hand***, and in other cases *flesh* is an appropriate meronym of *hand. Finger* and *flesh*, however, are not co-meronyms of *hand*, since different relational criteria (functional part versus material) are applied in each case."

6. **Troponymy**

In linguistics, troponymy is the presence of a 'manner' relation between two lexemes. In WordNET Verbs describing events that necessarily and uni directionally entail one another are linked: {buy}-{pay}, {succeed}-{try}, {show}-{see}, etc. basically the in the hierarchy verbs towards the bottom shows the manners are characterizing the events like communication-talk-whisper.

7. **Structured Polysemy and Metonymy**

- The senses of a word can also be related semantically, in which case we call the relationship between them structured polysemy. Consider this sense bank :

  The bank is on the corner of Nassau and Witherspoon.
- This sense, perhaps bank4, means something like "the building belonging to a financial institution". These two kinds of senses (an organization and the building associated with an organization) occur together for many other words as well (school, university, hospital, etc.). Thus, there is a systematic relationship between senses that we might represent as :

  BUILDING ↔ ORGANIZATION
- This particular subtype of polysemy relation is called **metonymy.** Metonymy is the use of one aspect of a concept or entity to refer to other aspects of the entity or to the entity itself. We are performing metonymy when we use the phrase the White House to refer to the administration whose office is in the White House. Other common examples of metonymy include the relation between the following pairings of senses :

AUTHOR (Jane Austen wrote Emma	↔	WORKS OF AUTHOR (I really love Jane Austen)
FRUITTREE (Plums have beautiful blossoms)	↔	FRUIT (I ate a preserved plum yesterday)

Difference between Polysemy and Homonymy

- Both polysemy and homonymy words have the same syntax or spelling. The main difference between them is that in polysemy, the meanings of the words are related but in homonymy, the meanings of the words are not related.
- For example, if we talk about the same word "Bank", we can write the meaning 'a financial institution' or 'a river bank'. In that case it would be the example of homonym because the meanings are unrelated to each other.

7. **Zeugma :**

- Another practical technique, for determining if two distinct senses are present is to combine two separate uses of a lexeme into a single example using a conjunction, a device has the rather improbable name of **zeugma**.
  - Consider the following ATIS examples.
  - Which of those flights serve breakfast ?

- Does Midwest express serve Philadelphia ?
- Does Midwest express serve breakfast and Philadelphia ?

- We use (?) to mark those examples that are semantically ill-formed. The oddness of the invented third example (a case of zeugma) indicates there is no sensible way to make a single sense of serve work for both breakfast and Philadelphia. We can use this as evidence that serve has two different senses in this case.

2.3.3 WordNet

- WordNet is a large lexical database of English words. Nouns, verbs, adjectives, and adverbs are grouped into sets of cognitive synonyms called 'synsets', each expressing a distinct concept. Synsets are interlinked using conceptual-semantic and lexical relations such as hyponymy and antonymy.
- The most commonly used resource for sense relations in English and many other WordNet languages is the WordNet lexical database (Fellbaum, 1998). English WordNet consists of three separate databases, one each for nouns and verbs and a third for adjectives and adverbs; closed class words are not included. Each database contains a set of lemmas, each one annotated with a set of senses.
- The widespread use of lexical relations in linguistic, psycholinguistic, and computational research has led to a number of efforts to create large electronic databases of such relations.
- These efforts have, in general, followed one of two basic approaches : mining information from existing dictionaries and thesauri, and handcrafting a database from scratch. Despite the obvious advantages of reusing existing resources, WordNet, the most well-developed and widely used lexical database for English, was developed using the latter approach (Beckwith et al., 1991).
- WordNet consists of three separate databases, one each for nouns and verbs, and a third for adjectives and adverbs; closed class lexical items are not included in WordNet. Each of the three databases consists of a set of lexical entries corresponding to unique orthographic forms, accompanied by sets of senses associated with each form.
- The WordNet 3.0 release has 117,798 nouns, 11,529 verbs, 22,479 adjectives, and 4,481 adverbs. The average noun has 1.23 senses, and the average verb has 2.16 senses. WordNet can be accessed on the Web or downloaded locally. Fig. 2.3.2 shows the lemma entry for the noun and adjective bass.

The noun "bass" has 8 senses in WordNet.

1. $bass^1$: (the lowest part of the musical range)
2. $bass^2$: $bass\ part^1$ – (the lowest part in polyphonic music)
3. $bass^3$: $basso^1$ – (an adult male singer with the lowest voice)
4. $sea\ bass^1$, $bass^4$ – (the lean flesh of a saltwater fish of the family serranidae)
5. $freshwater\ bass^1$, $bass^5$ – (any of various North American freshwater fish with lean flesh (especially of the genus Micropterus))
6. $bass^6$, $bass\ voice^1$, $basso^2$ – (The lowest adult male singing voice)
7. $bass^7$: (the member with the lowest range of a family of musical instruments)
8. $bass^8$: (nontechnical name for any of numerous edible marine and freshwater spiny-finned fishes)

Fig. 2.3.2 : A portion of the wordNet 3.0 entry for the noun bass

- It represents nouns, verbs, adjectives, and adverbs but it does not include functional terms in the closed classes (prepositions, conjunctions, etc.)
- The lexemes are grouped into sets of cognitive synonyms (synset), each representing a distinct concept.
- A set of senses (synset) is associated to each lexeme (unique orthographic form). Synsets are linked by conceptual/semantic and lexical relationships.
- Wordnet consists in lexicographic files, an application to load these files into a database and a library of search and browsing functions to visualize and access the database contents.

The Distinction Between WordNET and Thesaurus :

- Where thesaurus is helping us in finding the synonyms and antonyms of the words the WordNET is helping us to do more than that. WordNET interlinks the specific sense of the words wherein thesaurus links words by their meaning only.
- In the WordNET the words are semantically disambiguated if they are in close proximity to each other. Thesaurus provides a level to the words in the network if the words have similar meaning but in the case of WordNET, we get levels of words according to their semantic relations which is a better way of grouping the words.

2.3.3(A) Gloss

- A dictionary can provide useful information about the contexts related to the word senses (called glosses).
- we need to consider the alternative ways that dictionaries and thesauruses offer for defining senses. One is based on the fact that dictionaries or thesauruses give textual definitions for each sense called glosses. Here are the glosses for two senses of bank:

1. Financial institution that accepts deposits and channels the money into lending activities
2. Sloping land (especially the slope beside a body of water)

- Glosses are not a formal meaning representation; they are just written for people. Consider the following fragments from the definitions of *right, left, red,* and *blood* from the *American Heritage Dictionary.*

right *adj.* located nearer the right hand esp. being on the right when facing the same direction as the observer.

left *adj.* located nearer to this side of the body than the right.

red *n.* the color of blood or a ruby.

blood *n.* the red liquid that circulates in the heart, arteries and veins of animals.

- Note that there are eight senses for the noun and one for the adjective, each of which has a **gloss** (a dictionary-style definition), a list of synonyms for the sense, and sometimes also usage examples (shown for the adjective sense). WordNet doesn't represent pronunciation, so doesn't distinguish the pronunciation [b ae s] in bass^4, bass^5, and bass^8 from the other senses pronounced [b ey s].

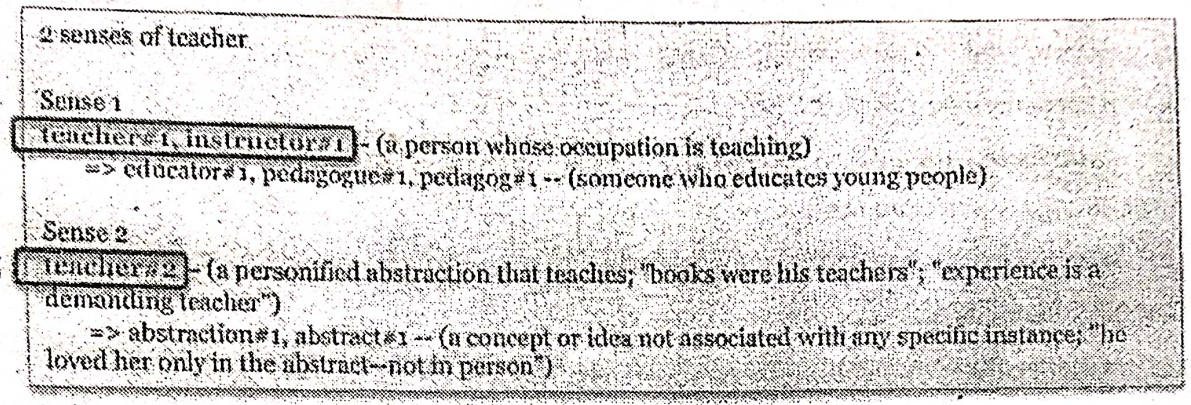
2.3.3(B) Synset

A synset is a set of synonyms that define a concept or word meaning

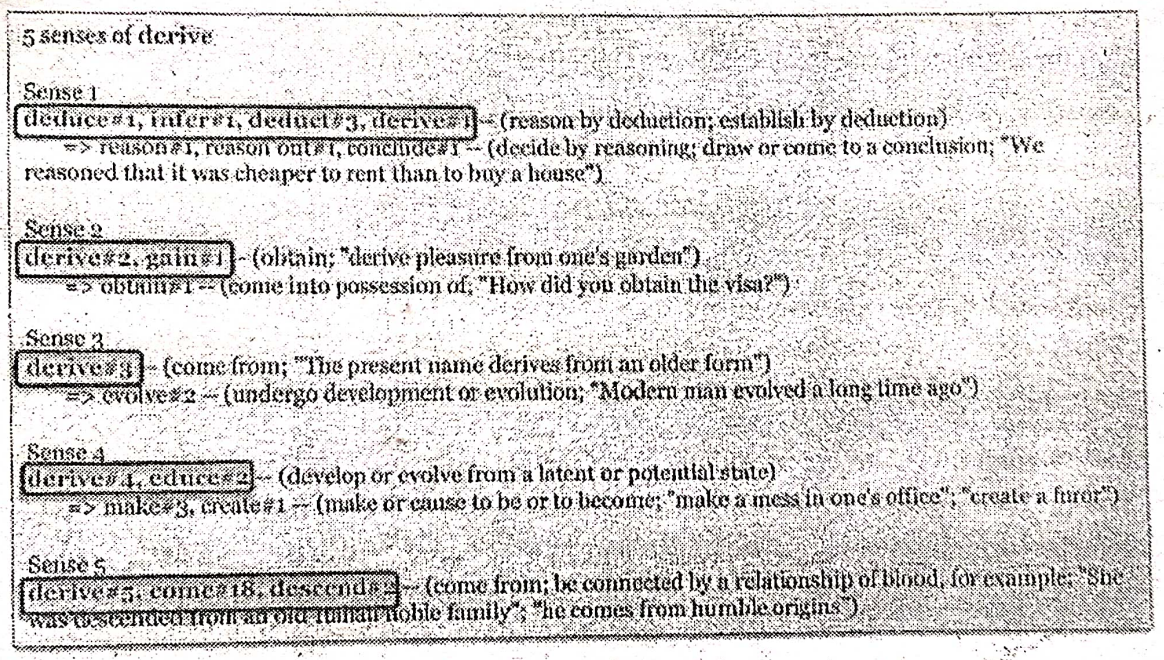
- About half of the synsets (~54%) contains only one term, about one third (~29%) 2 terms, about 10% 3 terms.

- An annotation (gloss) explaining the meaning is associated to each synset (especially to those containing a single term).

A synset contains ~1.75 terms in average)



Synset – verb example



The set of near-synonyms for a WordNet sense is called a synset (for synonym set); synsets are an important primitive in WordNet. The entry for bass includes synsets like (bass[1], deep[6]), or (bass[6], bass voice[1], basso[2]). Thus, instead of representing concepts in logical terms, WordNet represents them as lists of the word senses that can be used to express the concept. Here's another synset.

Example :

(chump[1], fool[2], gull[1], mark[9], patsy[1], fall guy[1], sucker[1], soft touch[1], mug[2])

The gloss of this synset describes it as :

- **Gloss :** A person who is gullible and easy to take advantage of Glosses are properties of a synset, so that each sense included in the synset has the same gloss and can express this concept. Because they share glosses, synsets like this one are the fundamental unit associated with WordNet entries, and hence it is synsets, not wordforms, lemmas, or individual senses, that participate in most of the lexical sense relations in WordNet.

2.3.3(C) Sense Relations in WordNet

Wordnet relationships - names & verbs

- For nouns the following relationships are provided among synsets
  - **Hyperonymy** - X is a kind of Y (car → vehicle)
  - **Hyponymy** – Y is a kind of X (vehicle → car)
  - **Coordinate terms**– Y is a coordinate term of X is X and Y share a common hyperonym (car and motorcycle).
  - **Holonymy** – X is part of Y (wheel → bicycle)
  - **Meronymy** – Y is part of X (bicycle → wheel)
- For verbs the following relationships are provided among synsets
  - **Hyperonymy** – the activity of X is a kind of Y (to see → to perceive)
  - **Troponymy** – the activity Y executes X in some sense (to eat → to devour)
  - **Entailment** – Y is required to perform X (to snore→to sleep)
  - **Coordinate terms** – Terms share a common hyperonym (to hear-to see as cases of to perceive)

Wordnet relations - adjectives & adverbs

- Words can be linked to other words by lexical relationships such as antonomy (words that have opposite meanings)
  - good ⇔ bad, day ⇔ night, exit ⇔ entrance
- For adjectives the following relationships are defined
  - Related nouns– (noisy → noise)
  - Similar to– (noisy → clanking)
- The **descriptive adjectives** are organized into groups containing a main synset (head) and satellite synsets. Each group is organized around a pair (sometimes a triple) of antonyms corresponding to the main terms. The satellite synsets are those linked by the *"Similar to"* relationship.
- Relational adjectives are used to categorize the noun and they have neither a group structure nor an antonym (e.g. musical, nervous).
- For the adverbs the following relationships are defined
  - Base adjective– (slowly → slow)

2.3.4 Word Sense Disambiguation(WSD)

What does this word mean ?

- This **plant** needs to be watered each day.

  ⇒ living plant
- This **plant** manufactures 1000 widgets each day.

  ⇒ factory
- We say that these words all have various **word senses** and that some of the senses are synonymous with one another. The process of choosing the right sense in context is called **word sense disambiguation** (or WSD).

- WSD algorithms take as input a word in context and a fixed inventory of potential word senses and outputs the correct word sense in context.
- Word Sense Disambiguation basically solves the ambiguity that arises in determining the meaning of the same word used in different situations.
- The inventory of sense tags depends on the task. For sense tagging in the context of translation from English to Spanish, the sense tag inventory for an English word might be the set of different Spanish translations.
- For automatic indexing of medical articles, the sense-tag inventory might be the set of MeSH (Medical Subject Headings) thesaurus entries. Or we can use the set of senses from a resource like WordNet, or super senses if we want a coarser-grain set. Fig. 2.3.3 shows some such examples for the word bass.

WordNet sense	Spanish translation	wordnetsupersense	Target word in context
$Bass^4$	Lubina	FOOD	...fish as pacific salmon and striped bass and ...
$Bass^7$	Bajo	ARTIFACT	...play bass because he doesn't have to solo...

Fig. 2.3.3 : Some possible sense tag inventories for bass

- In some situations, we just need to disambiguate a small number of words. In such lexical sample tasks, we have a small pre-selected set of target words and an inventory of senses for each word from some lexicon. Since the set of words and the set of senses are small, simple supervised classification approaches work very well.
- More commonly, however, we have a harder problem in which we have to disambiguate all the words in some text. In this all-words task, the system is given an entire texts and a lexicon with an inventory of senses for each entry and we have to disambiguate every word in the text (or sometimes just every content word).
- The all-words task is similar to part-of-speech tagging, except with a much larger set of tags since each lemma has its own set. A consequence of this larger set of tags is data sparseness.
- Several approached to WSD have been proposed like **Machine Readable Dictionary and knowledge-based, Machine Learning- Supervised methods, Semi-supervised and Unsupervised methods.**

2.3.4(A) Supervised Learning

- WSD can be approached as a classification task.
  - The correct sense is the class to be predicted.
  - The word is represented by a set (vector) of features to be processed as the classifier input.
  - Usually the feature include a representation of the word to be disambiguated (target) and of its context (a given number of words at the left and the right of the target word).
  - The word itself, the word stem, the word PoS can be exploited as features.
- The classifier can be learnt from examples given a labeled dataset.
- Different models can be exploited to implement the classifier (Naïve Bayes, neural networks, decision trees...) .
- The limitation of the learning based approach is scalability when a large number of labeled examples is required.

- Supervised WSD algorithms can use any standard classification algorithm. Features generally include the word identity, part-of-speech tags, and embeddings of surrounding words, usually computed in two ways : collocation features are words or n-grams at a particular location, (i.e., exactly one word to the right, or the two words starting 3 words to the left, and so on). bag of word features are represented as a vector with the dimensionality of the vocabulary (minus stop words), with a 1 if that word occurs in the in the neighborhood of the target word.

2.3.4(B) Naïve Bayes

- The bayesian approach aims at maximizing of the probability of sense s given the feature vector f_w describing the target word

$$\hat{s} = \underset{s \in S}{\text{argmax}}\ p(s \mid f_w) = \underset{s \in S}{\text{argmax}} \frac{p(f_w \mid s)\, p(s)}{p(f_w)}$$

- With the simplifying assumption that the feature vector entries (words in context) are independent of each other $p(f_w|s)$ can be written as

$$p(f_w \mid s) = \prod_{j=1}^{n} p(f_j \mid s)$$

- The probabilities $p(f_j|s)$ model the statistics for distribution of feature j (e.g. a given word) in the context of word w when having the sense s.
- $p(s)$ is the a priori probability of each sense of the word

2.3.4(C) Semi-Supervised WSD : Bootstrapping

- Both the supervised approach and the dictionary-based approaches to WSD require large hand-built resources. supervised training sets in one case, large dictionaries in the other.
- We can instead use bootstrapping or semi-supervised learning, which needs only a very small hand-labeled training set.

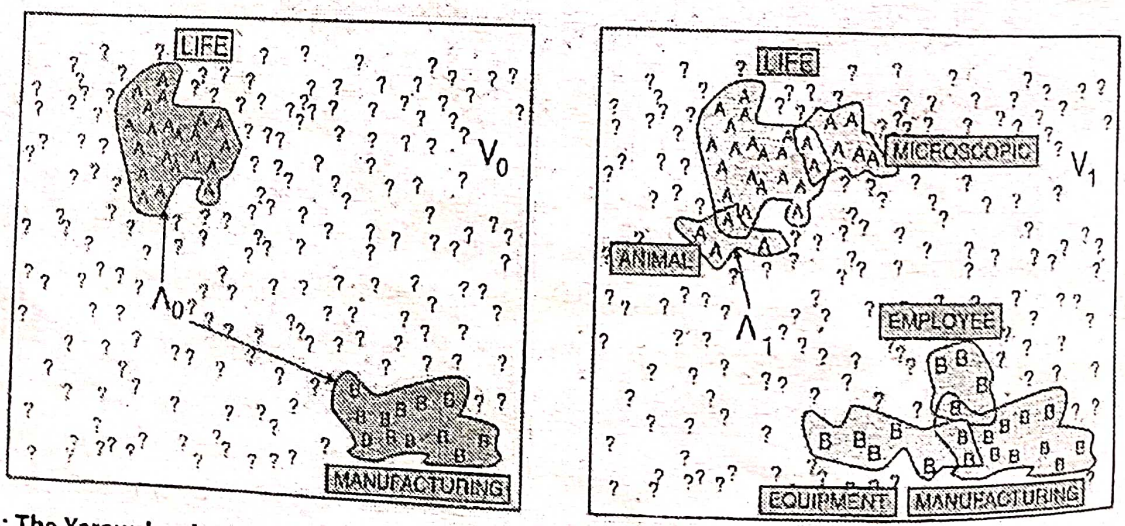


Fig. 2.3.4 : The Yarowsky algorithm disambiguating "Plant" at two stages; "?" indicates an ublabeled observation, A and B are observations labeled as SENSE-A or SENSE-B. The initial stage (a) shows only seed sentences Λ_0 labeled by collocates ("life" and "manufacturing"). An intermediate stage is shown in (b) where more collocates have been discovered ("equipment", "microscopic", etc.) and more instances in V_0 have been moved into Λ_1, living a smaller unlabled set V_1. Figure adapted from Yarowsky (1995).

- A classic bootstrapping algorithm for WSD is the Yarowsky algorithm for learning a classifier for a target word (in a lexical-sample task) (Yarowsky, 1995). The algorithm is given a small seedset Λ_0 of labeled instances of each sense and a much larger unlabeled corpus V_0.
- The algorithm first trains an initial classifier on the seedset Λ_0. It then uses this classifier to label the unlabeled corpus V_0. The algorithm then selects the examples in V_0 that it is most confident about, removes them, and adds them to the training set (call it now Λ_1). The algorithm then trains a new classifier (a new set of rules) on Λ_1, and iterates by applying the classifier to the now-smaller unlabeled set V_1, extracting a new training set Λ_2, and so on.
- With each iteration of this process, the training corpus grows and the untagged corpus shrinks. The process is repeated until some sufficiently low error-rate on the training set is reached or until no further examples from the untagged corpus are above threshold.

2.3.4(D) Dictionary-based Methods

- A dictionary can provide useful information about the contexts related to the word senses (glosses)
- As the name suggests, for disambiguation, these methods primarily rely on dictionaries, treasures and lexical knowledge base.
- They do not use corpora evidences for disambiguation.
- The Lesk method is the seminal dictionary-based method introduced by Michael Lesk in 1986.
- The Lesk definition, on which the Lesk algorithm is based is **"measure overlap between sense definitions for all words in context".**

A simple approach is the Lesk algorithm (1986)

- The algorithm computes the intersection among the glosses associated to the different meanings of the words in the sentence
- The combination yielding the maximum overall intersection is selected (the complexity is combinatorial in the number of senses)

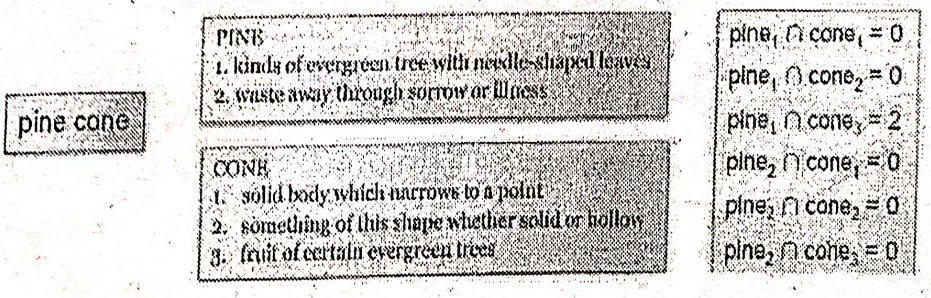


Fig. 2.3.5

Lesk Algorithm :

Lesk Algorithm is a classical Word Sense Disambiguation algorithm introduced by Michael E. Lesk in 1986.

- The Lesk algorithm is based on the idea that words in a given region of the text will have a similar meaning. In the Simplified Lesk Algorithm, the correct meaning of each word context is found by getting the sense which overlaps the most among the given context and its dictionary meaning.
- The most well-studied dictionary-based algorithm for sense disambiguation is the Lesk algorithm, really a family of algorithms that choose the sense whose dictionary gloss or definition shares the most words with the target word's neighborhood. Fig. 2.3.6 shows the simplest version of the algorithm, often called the Simplified Simplified Lesk algorithm

```
function SIMPLIFIED LESK(word, sentence) returns best sense of word

best-sense ← most frequent sense for word
max-overlap ← 0
context ← set of words in sentence
for each sense in senses of word do
  signature ← set of words in the gloss and examples of sense
  overlap ← COMPUTEOVERLAP(signature, context)
  if overlap > max-overlap then
      max-overlap ← overlap
      best-sense ← sense
end
return(best-sense)
```

Fig. 2.3.6 : The simplified Lesk algorithm The compute overlap function returns the number of words in common between two sets, ignoring function words or other words on a stop list. The original Lesk algorithm defines the context in a more complex way. The corpus Lesk algorithm weights each overlapping word w by its-log P(w) and includes labeled training corpus data in the signature.

- As an example of the Lesk algorithm at work, consider disambiguating the word bank in the following context :

  (a) The **bank** can guarantee deposits will eventually cover future tuition costs because it invests in adjustable-rate mortgage securities.

  Given the following two WordNet senses :

$bank^1$	Gloss :	A financial institution that accepts deposits and channels the money into lending activities
	Examples :	"he cashed a check at the bank", "that bank holds the mortgage on my home"
$bank^2$	Gloss :	Sloping land (especially the slope beside a body of water)
	Examples :	"they pulled the canoe up on the bank". "he sat on the bank of the river and watched the currents "

- Sense **$bank^1$** has two non-stop words overlapping with the context (a): deposits and mortgage, while sense **$bank^2$** has zero words, so sense **$bank^1$** is chosen.

2.3.4(E) Applications of Word Sense Disambiguation (WSD)

- Word sense disambiguation (WSD) is applied in almost every application of language technology.
- Let us now see the scope of WSD :

1. Machine Translation

Machine translation or MT is the most obvious application of WSD. In MT, Lexical choice for the words that have distinct translations for different senses, is done by WSD. The senses in MT are represented as words in the target language. Most of the machine translation systems do not use explicit WSD module.

2. Information Retrieval (IR)

Information retrieval (IR) may be defined as a software program that deals with the organization, storage, retrieval and evaluation of information from document repositories particularly textual information. The system basically assists users in finding the information they required but it does not explicitly return the answers of the questions. WSD is used to resolve the ambiguities of the queries provided to IR system. As like MT, current IR systems do not explicitly use WSD module and they rely on the concept that user would type enough context in the query to only retrieve relevant documents.

3. **Text Mining and Information Extraction (IE)**

In most of the applications, WSD is necessary to do accurate analysis of text. For example, WSD helps intelligent gathering system to do flagging of the correct words. For example, medical intelligent system might need flagging of "illegal drugs" rather than "medical drugs"

4. **Lexicography**

WSD and lexicography can work together in loop because modern lexicography is corpusbased. With lexicography, WSD provides rough empirical sense groupings as well as statistically significant contextual indicators of sense.

2.3.5 Latent Semantic Analysis

- Latent Semantic Analysis is a method for extracting relationships between words in a large number of documents using unsupervised learning. This helps us choose the appropriate documents.
- It simply reduces the dimensionality of the massive corpus of text data. Extraneous data adds noise to the process of extracting meaningful insights from data.
- Latent Semantic Analysis is one of the natural language processing techniques for semantic analysis, which means that we are trying to extract some meaning from a corpus of text using statistical methods, and was introduced by Jerome Bellegarde in 2005.
- LSA is essentially a technique for identifying patterns in text documents, or in other words, for extracting relevant and important information from text documents. If we ask whether it is supervised or unsupervised, the answer is clearly unsupervised.
- It is a very useful technique for reducing matrix or topic modelling dimensions, and it is also known as Latent Semantic Indexing (LSI). LSA's main concept and work is to group together all words with similar meanings.

Significance of Term Frequency/ Inverse Document Frequency in LSA

- The number of times an instance or keyword appears in a single document divided by the total number of words in that document is defined as term frequency :

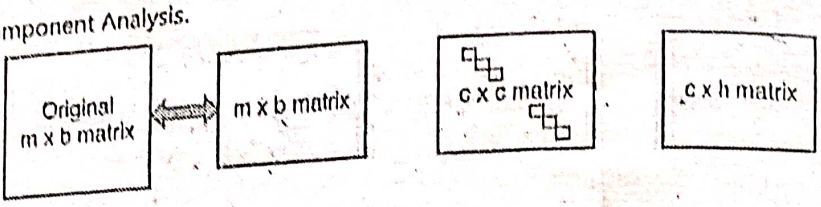
$$TF(t,d) = \frac{\text{Number of times term "t" appears in a document}}{\text{Total Number of terms in a document "d"}}$$

- Because the length of the document varies in each case, term frequency varies with the occurrence of the term.
- Inverse Document Frequency(IDF), signifies how important the term is to be in the collection of documents. IDF calculates the weight of rare term of the text in a collection of documents. The formula of IDF is given by

$$IDF = \log \in \left(\frac{\text{Total number of documents}}{\text{Number of documents that contains term "t"}}\right)$$

- The main idea behind Tf/IDF in Latent Semantic Analysis is to provide each word count and frequency of rare words in order to provide weights based on their rarity. TF/IDF is preferable to conventional counting of occurrence of the word because it only counts the frequency without classification.
- After we have completed the classification using TF/IDF, we usually proceed to the next step, which is the reduction of matrix dimension, because normally with so many features, the input has higher dimensions, and a higher dimension input is difficult to understand and interpret, so we have many techniques to lower the dimension with maximum information gain, which include

1. Singular Value Decomposition(SVD)
2. Principal Component Analysis.



Singular value decomposition

- Singular value decomposition is a method for matrix decomposition from higher to lower, it usually divides the matrix into three matrices. Let us take an input matrix m x b of higher dimension as 'A', to calculate the SVD we will use the formula given below

$$A(m \times b) = U(m \times m) \cdot \sigma VT$$

- Here, σ is a diagonal matrix of size m × n and VT is a transpose of n × n orthogonal matrix. SVD may perform several other tasks but remains efficient primarily for dimension reduction, it is widely used and accepted by machine learning developers.
- The results of SVD are always elegant; it can dramatically reduce more than 150 k parameters or dimensions to an understandable 50 to 70 parameters. The completion of the preceding two tasks fulfils the purpose of latent semantic analysis.
- LSA has many applications, but it is most commonly used in search engines because it is a very useful technique there. For example, if you searched 'sports,' the results also showed cricket and cricketers, which is due to LSA being used in search engines. LSA could also be used for document clustering in text analysis, recommender systems, and creating user profiles.

Review Questions

Q.1 What is Morphology? What are the types of morphology ?

Q.2 Explain Inflectional morphology &Derivational morphology ?

Q.3 Explain Morphological parsing with Finite State Transducers (FST)

Q.4 Define synset and gloss. Explain structured polysemy, reviersives, Zeugma with example.

Q.5 What is parsing? Explain top down and bottom up parsing ?

Q.6 Explain CYK parsing algorithm with example ?

Q.7 Write a short note on probabilistic context free grammar ?

Q.8 What is semantic analysis ?

Q.9 Explain lexical semantic ?

Q.10 What is Word Sense Disambiguation? Illustrate with example how Dictionary-based approach identifies correct sense of an ambiguous word.

Q.11 Explain following Relations among lexemes & their senses, Homonymy, Synonymy, Hyponymy with example

Q.12 Explain cosine similarity between documents.

□□□