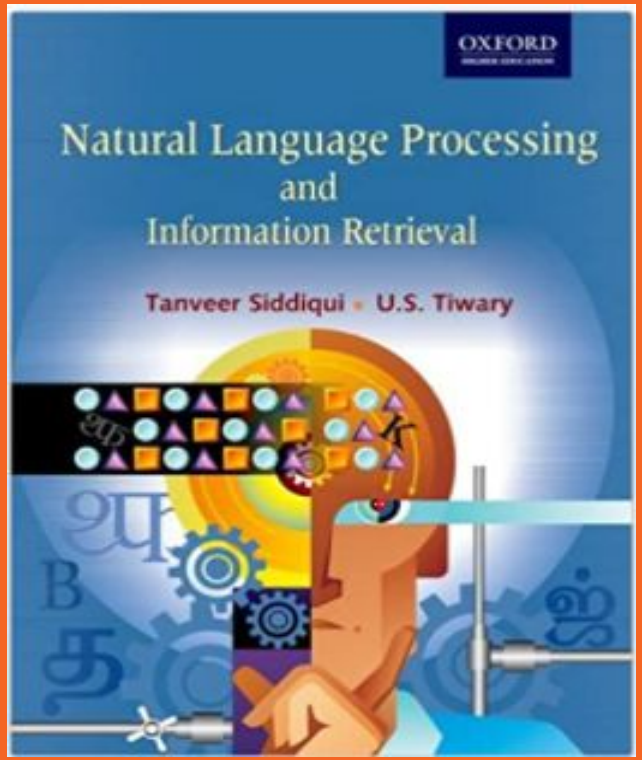
Module -2

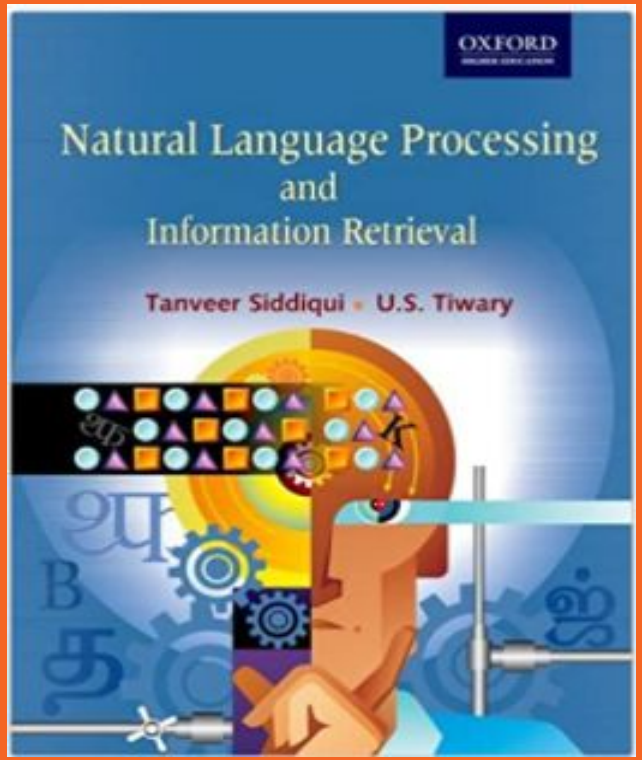
Word level Analysis
And
Syntactic Analysis



Chapter 3

Word Level Analysis

- Introduction
- Regular Expressions
- Finite State Automata
- Morphological parsing
- Spelling Error Detection and Correction
- Word and word classes
- Parts-of-Speech Tagging



Introduction

- Word level analysis in NLP carried out at *word level* includes :
  - Characterizing word sequences
  - Identifying morphological variants
  - Detecting and correcting misspelled words
  - Identifying correct part-of-speech of a word.

Introduction

- *Regular Expressions* are used for *describing text strings.*
- Example: To find word ‘supernova’ by using search engine and in information retrieval applications
- *Implementation of RE* using *Finite-State Automaton(FSA).*
- Used in speech recognition and synthesis, spell checking, information extraction.
- *Errors in typing and spelling* are *common in text processing*.
- An interactive facility to correct errors, Identifying word with different meanings depending on the context.

Regular Expressions

- *Regular Expressions (regexes)* are powerful way to find and replace strings that take a defined format.
- Regular expressions can be used to *parse* dates, urls and email addresses, log files,configuration files, command line switches or programming scripts.
- RE are the *useful tools* for the design of *language coimpilers* .
- RE also used in *NLP* for tokenization, describing lexicons, morphological analysis etc.

Regular Expressions

- *In computer science,* regular expressions was powerful by *unix-based editor, "ed"*.
- *Perl* was the first language that provides integrated *support for regular expressions.* Uses *slash* around the regular expression.
- RE was first introduced by Kleene (1956).
- RE is an *algebraic formula* consisting of Pattern,set of strings.
- For example: the expression */a/* => set containing the string *'a'*
- */supernova/* =>set contains the string *'supernova'.*

Regular Expressions

- Some of the *simple Regular Expressions* are as below:

Regular expression	Example patterns
/book/	The world is a book, and those who do not travel read only one page.
/book/	Reporters, who do not read the stylebook, should not criticize their editors.
/face/	Not everything that is faced can be changed. But nothing can be changed until it is faced.
/a/	Reason, Observation, and Experience—the Holy Trinity of Science.

Regular Expressions- Character Classes

- Characters are grouped by putting between [ ]
- Example: */[0123456789]/* => Any single digit.
- */[0-9]/* => Any one digit from 0 to 9
- */[m-p]/* => Any one letter m,n,o or p
- */[^x]/* =>Single character except x

Regular Expressions- Character Classes

- Use of square brackets in regular expression are as shown below:

Table 3.2 Use of square brackets

RE	Match	Example patterns matched
[abc]	Match any of a, b, and c	'Refresher course will start tomorrow'
[A–Z]	Match any character between A and Z (ASCII order)	'the course will end on Jan. 10, 2006'
[^A–Z]	Match any character other than an uppercase letter	'TREC Conference'
[^abc]	Match anything other than a, b, and c	'TREC Conference'
[+*?.]	Match any of +, *, ?, or the dot.	'3 +2 = 5'
[a^]	Match a or ^	'^ has three different uses.'

Regular Expressions- Character Classes

- Regular expressions are *case sensitive*.
- /s/ => Matches lower case ‘s’ but not ‘S’. Means matches string sana but not String ‘Sana’
- /[sS]/ => Match the string either ‘s’ or ‘S’.
- /[sS]upernova[sS]/ =>Matches any of the strings ‘supernovas’ or ‘Supernovas’ or ‘supernovaS’ or ‘SupernovaS’ but not the string ‘supernova’.
- This can be achieved using question mark as /?/

Regular Expressions- Character Classes

- *?* => zero or one occurrence of the previous character.
- The regular expression

  /supernovas?/ => Matches ‘supernova’ or ‘supernovas’.
- * => Specifies zero or more occurrences of a preceding character or RE.
- * is called as *Kleene * (pronounced as cleany *)
- */b*/* =>Match any string containing zero or more occurrences of b.
- I.e. *‘b’,’bb’* or *‘ bbb’* etc.

Regular Expressions- Character Classes

- */[ab]*/* => zero or more occurrence of “a”s or “b”s.
- This will match strings: *‘aa’ , ’bb’* or *‘abab’*
- + => Specifies one or more occurrences of a preceding character
- + is called as *Kleene +.*
- */a+/* => one or more occurrences of ‘a’.
- */[0-9]+/* => sequence of digits
- ^ *(caret)* - anchor =>Matches at the beginning of a line.
- $ *(dollar)* - anchor =>Matches at the end of a line

Regular Expressions- Character Classes

- /^ The nature\.$/

  Here, it Search for a line containing the phrase *'The nature.'*

  nothing else.

- . => Wildcard character, Matches any single character.
- */./* => Any single character
- */.at/* => Matches any of the string such as : cat, bat,gat,kat,4at etc.

Regular Expressions- Character Classes

- Some of the special characters:

Table 3.3 Some special characters

RE	Description
.	The dot matches any single character.
\n	Matches a new line character (or CR+LF combination).
\t	Matches a tab (ASCII 9).
\d	Matches a digit [0–9].
\D	Matches a non-digit.
\w	Matches an alphanumeric character.
\W	Matches a non-alphanumberic character.
\s	Matches a whitespace character.
\S	Matches a non-whitespace character.
\	Use \ to escape special characters. For example, \. matches a dot, * matches a * and \\ matches a backslash.

Regular Expressions- Character Classes

- /…..berry/ => Matches ten-letter strings that end with *berry.*
- This finds pattern like : *strawberry, sugarberry* and *blackberry* but *fails* the match *blueberry* and *hackberry.*
- | (Pipe) => disjunction operator,
- */blackberry|blackberries/* =>Matches either ‘blackberry’, or ‘blackberries’.
- */blackberry|ies/* => Matches either ‘blackberry’, or ‘ies’.

Regular Expressions- Character Classes

- **Example:** To check if string is ***email address*** or not
- An email address consist of a *non-empty sequence of characters* followed by the *'@ '* symbol, followed by another *non-empty sequence of characters* ending with pattern like .xx , .xxx , .xxxx etc.
- The regular expression for an email address is :

  ^[A-Za-z0-9_\.-]+ + @[A-Za-z0-9_\.-]+ +[A-Za-z0-9_][A-Za-z0-9_]$

Regular Expressions- Character Classes

- **Example:** To check if string is ***email address*** or not
  ^[A-Za-z0-9_\.-]+ + @[A-Za-z0-9_\.-]+ +[A-Za-z0-9_][A-Za-z0-9_]$
- Parts of regular expression are:

Table 3.4 Parts of regular expression of Example 3.1

Pattern	Description
^[A-Za-z0-9_\.-]+	Match a positive number of acceptable characters at the start of the string.
@	Match the @ sign.
[A-Za-z0-9_\.-]+	Match any domain name, including a dot.
[A-Za-z0-9_][A-Za-z0-9_]$	Match two acceptable characters but not a dot. This ensures that the email address ends with .xx, .xxx, .xxxx, etc.

- Above example works for *most of the cases*.But it is *may not be accurate enough* to *match all correct addresses*.
- It may accept *non-working email addresses*.So *fine tuning is required* for accurate characterization.

Finite -State Automata

- Consider the game of *playing board* ; dice are thrown .
- All possible positions of the pieces on the board => *states.*
- State in which the game begins =>*initial state*
- State corresponding to the winning position =>*final state*
- **Example:**Machine with input,processor,memory and output device

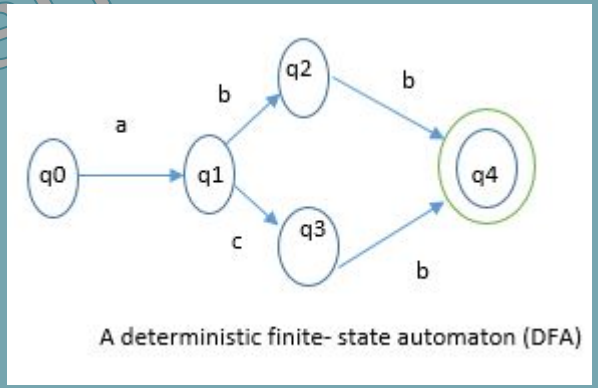
  Here, machine starts in initial state,Checks input goes to next state,If process perfectly then reaches final state & terminates.
- If machine stucks in between,then *non-final state*-reject input

Finite -State Automata

- In ‘Finite Automaton’
  - *‘Finite’* refers *number of states and the alphabet of input symbols* is finite.
  - *‘Automaton’* refers machine *moves automatically*. I.e. change of state is completely governed by the *input*.=>*deterministic*
- ***Properties*** of finite automaton:
  - A finite set of *states*, one is *initial/ start* state , and one more is *final state.*
  - A finite *Alphabet set*, $\sum$, consisting of input symbols
  - A finite set of *transitions* that specify each state and each symbol of input alphabet

Finite -State Automata

- **Deterministic Finite state Automaton (DFA)** : *Exactly one transition leading* out of a state is possible for the *different input symbol*
- **Example: Suppose** ∑ ={ a,b } , the set of states ={q0,q1,q2,q3,q4} with q0 being the start state and q4 the final state, we have the following rules of transition:
  - From state q0 and with input a, go to state q1.
  - From state q1 and with input b, go to state q2.
  - From state q1 and with input c, go to state q3.
  - From state q2 and with input b, go to state q4.
  - From state q3 and with input b, go to state q4.



A deterministic finite- state automaton (DFA)

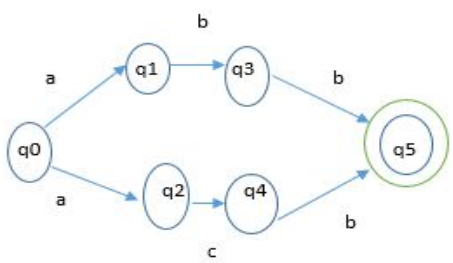
- Exactly one transition leading out of each state

Finite -State Automata

- *So, Deterministic finite automaton (DFA)* can be defined as 5 tuple (Q,∑, δ,S,F) below:
  - Where,Q is set of states,∑ is an alphabet,S is the start state, F⊆Q is a set of final states , δ is a transition function.
  - Transition function δ defines mapping from Q X ∑ to Q. i.e. for each state q and symbol a, there is at most one transition possible as shown above.
- 'Finite Automaton' used in *wide variety of areas*: linguistics, electrical engineering, computer science, mathematics and logic
- These are important tool in computational linguistics and used in mathematical device to implement *regular expressions*.

Finite -State Automata

- **Non-Deterministic Finite Automaton (NFA):** maps Q X (∑ U {ε}) to a subset of the power set of Q. i.e. for each state,there can be more than one transition on a given symbol, each leading to a different state.
- Q X (∑ U {ε}) , means NFA can make use of ε, to do state transition,if no symbols(a,b,..) are defined.
- In *NFA*, *More than one transition* out of a state is possible for the *same input symbol*

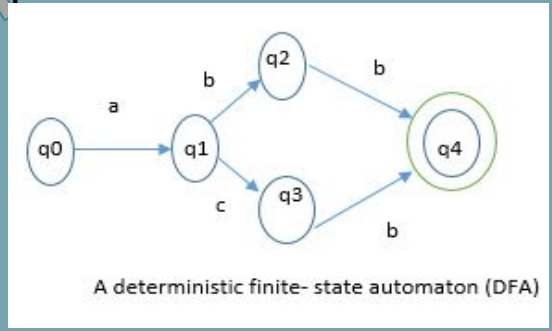


Non- deterministic Finite- state Automaton (NFA)

- Here,There are *2 possible transitions* from state ***q0 on input symbol a***

Finite -State Automata

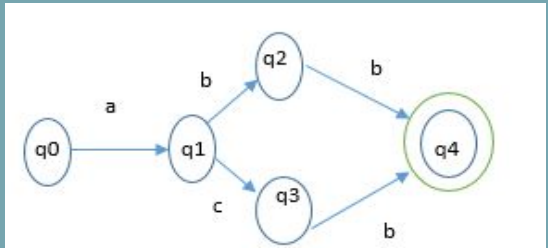
- *Path* is a sequence of transitions beginning with the *start state*
- A path leading to one of the final states is a *successful path*.
- Finite State Automata(FSA) encodes set of strings that can be formed by *concatenating the symbols* along each successful path.
- **Example:**Consider input *ac* , for DFA =>we start with q0 input is a and reaches to q1 and got q3 by input c =>*not reached final state*
- =>*Unsuccessful automaton*=> String *"ac"* is not recognized by the automaton.
- **Example:**Consider input *acb* =>reached final state. =>successful termination =>*Language defined* by the automaton. It can be described by the RE as : */abb|acb/*



A deterministic finite- state automaton (DFA)

Finite -State Automata

- *Listing all the state transitions* is inconvenient as it quite long so we have represent automaton as => *State-transition table.*
- In transition table,Rows => *states* , Columns=> *input (symbol)* , ϕ => *Missing transition*



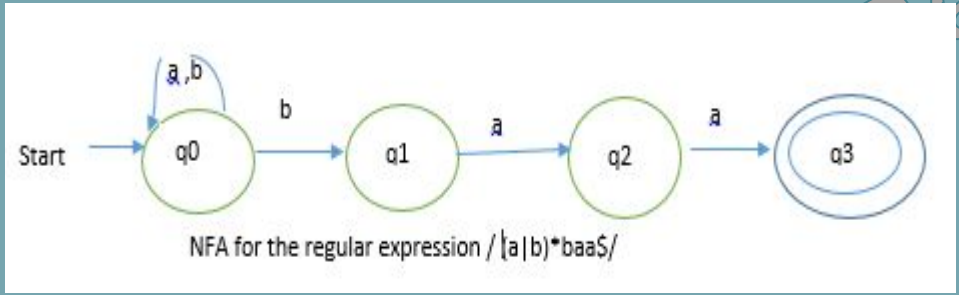
A deterministic finite- state automaton (DFA)

	Input		
State	**a**	**b**	**c**
Start:q0	q1	ϕ	ϕ
q1	ϕ	q2	q3
q2	ϕ	q4	ϕ
q3	ϕ	q4	ϕ
final:q4	ϕ	ϕ	ϕ

The State-Transition Table for the DFA

Finite -State Automata

- **Example:** Language consisting of all strings containing *a*s and *b*s and ending with *baa*
- This language can be specified by regular expression */(a|b)*baa$/*
- *NFA* and *State-transition table* can be represented as below:



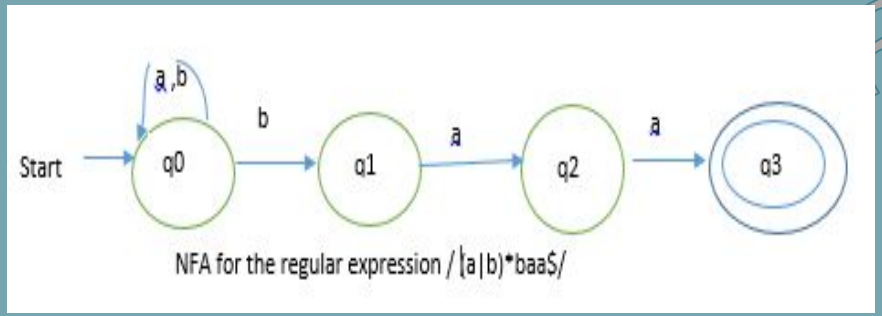
NFA for the regular expression / (a|b)*baa$/

	Input	
State	a	b
Start:q0	{q0}	{q0,q1}
q1	{q2}	ϕ
q2	{q3}	ϕ
final:q3	ϕ	ϕ

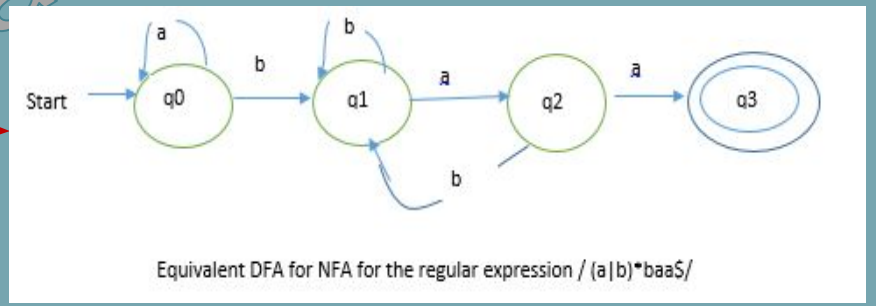
The State-Transition Table for the NFA

Finite -State Automata

- Two automata that define the same language are said to be *equivalent*.
- NFA can be converted to an equivalent DFA and vice versa
- **Example:** Language consisting of all strings containing *a*s and *b*s and ending with *baa* .Regular expression is : */(a|b)*baa$/*
- The Equivalent DFA for the NFA shown as below:



NFA for the regular expression / (a|b)*baa$/



Equivalent DFA for NFA for the regular expression / (a|b)*baa$/

Morphological Parsing

- Sub-discipline of linguistics.
- Studies *word structure* and the *formation of words* from smaller units (*morphemes*).

- **Goal of Morphological parsing:**
  - To *discover* morphemes that build a given word.

Morphological Parsing

- ***Morphemes***
- Are the *smallest meaning-bearing units* in a language. (are the smallest meaningful constituents of words).
- Words are composed of morphemes (one or more).
- **Example :**
  - *bread* – consists of single morpheme
  - *eggs* – consist of two: morpheme *egg* and morpheme *-s*

Note: For your reference- other morphemes:

- *sing-er-s*, *home-work*, *un-kind-ly*, *flipp-ed*
- *De-nation-al-iz-ation*
- *auto-servis-u* (voluntary service)
- Plural morpheme: *cat-s, dog-s, judg-es*
- Opposite: *un-happy, in-comprehensive, im-possible, ir-rational*

Morphological Parsing

- There are ***two broad classes*** of morphemes:
  - Stems
  - Affixes

Stems:

- The main morpheme in a word.
- Contains the central meaning.
- A simple key to word similarity.

Affixes:

- Modify the meaning given by the stem.

Morphological Parsing

Affixes

- *Modify* the meaning given by the stem.

- Affixes are divided into:
  - *Prefix* – morphemes which appear before a stem
  - *Suffix* - morphemes applied to the end of the stem
  - *Infix* - morphemes that appear inside a stem
  - *Circumfix* - morphemes applied to either end of the stem

- Prefixes and suffixes are quite common in Urdu, Hindi and English.

Morphological Parsing

- **Affixes Example:**
  - *Prefix* – morphemes which appear before a stem

    Example:*Un*happy

    *be*waqt (Urdu-meaning:untimely)

    अप मान, *सं* सार, *उप* वन (Hindi)

    *ಅ*ತೃಪ್ತಿ(Athrupti- Kannada)
  - *Suffix* - morphemes applied to the end of the stem

    Example: Happi*ness*

    tree*s*, bird*s*

    Ghodh*on*(Urdu)

    शीत*लता*

    ಮರ*ಗಳು*,ಉಪಯೋ*ಗಿಸು* (upayog*isu)*

Morphological Parsing

- **Affixes Example:**
  - *Infix* - morphemes that appear inside a stem
  - Common in Austronesian and Austroasiatic languages (Tagalog -philippines, Khmer-cambodia)

    Example:

    Kayu- 'wood'=>kayu-in- => 'kinayu'(gathered wood)

    basa - 'read' => b·*um*·asa - 'readpast'

    sulat - 'write' => s·*um*·ulat - 'wrote'

    Very *rare* in English:

    abso·bloody·lutely (emphatic or humorous form of absolutely)

Morphological Parsing

- **Affixes Example:**
  - *Circumfix* - morphemes applied to either end of the stem

    Example: *in*-correct-*ly*

    im-matur-*ity*

    un-bear-*able*

- **Word Formation:**
- 3 main ways for word formation:
  - Inflection
  - Derivation
  - Compounding

Morphological Parsing

- **Word Formation:** 3 ways

1. **Inflection**
   - *Root word* is combined with a *grammatical morpheme* to yield a word of the *same class*.

     Example: bring, brings, brought, bringing

2. **Derivation**
   - Word stem is combined with a *grammatical morpheme* to yield a word belonging to *different class.*

     Example: compute (verb)=> computation(noun)

     Formation of noun from verb/adjective => Nominalization

     Example: require (v)=>requirement (N), appear(v)=>appearance(N)

Morphological Parsing

- **Word Formation:** 3 ways

3. Compounding

- Merging *two or more words* to form a *new word*.

  N + N → N: *rain-bow*

  V + N → V: *pick-pocket*

  P + V → V: *over-do*

  N + P → N: *desk-top*

  P + V → V: *over-look*

  Adj + Adj → Adj: *bitter-sweet*

Morphological Parsing

Why Morphological Analysis

- *New words* are continually forming a NL
- Morphologically *related* to *known words*
- Understanding morphology => understand *syntactic and semantic properties* of new words.

Example:

- In parsing, *Agreement features* of *words*
- In Information Retreival (IR) , *Identify* the presence of a *query word* in a document in spite of morphological variants .

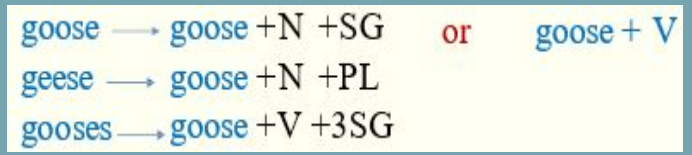
Morphological Parsing

Why Morphological Analysis

- In Parsing, Takes an input and produces some sort of structures.
- Morphological parsing, takes as input the *inflected surface form* of each word in *text*
- Output the parsed form consisting of canonical form (lemma) of the *words* and a *set of tags* showing its *syntactic category* and *morphological characteristics*.

Example:

- POS(noun,pronoun,adjectives..)/inflectional properties (gender, number, person, tense, …)



Morphological Parsing

What do we need to build a morphological parser?

A morphological parser uses the following information sources:

- **Lexicon:** list of stems and affixes with basic information about them (corresponding p.o.s.)
- **Morphotactics:** describes a way of ordering (arranging) morphemes that constitute a word

Example: *rest-less-ness* vs *rest-ness-less*

Orthographic rules: spelling rules that specify the changes that occur when 2 morphemes combine.

Example: *easy-> easier not 'easyer'*

y ->ier

Morphological Parsing

- Morphological analysis can be *avoided* if an *exhaustive (complete) lexicon* is available
- Which lists all the word-forms of all the roots.
- **Example:** *Exhaustive lexicon* with feature of all the roots are as shown below:

Word form	Category	Root	Gender	Number	Person
Ghodhaa	noun	GhoDaa	masculine	singular	3rd
Ghodhii	-do-	-do-	feminine	-do-	-do-
Ghodhon	-do-	-do-	masculine	plural	-do-
Ghodhe	-do-	-do-	-do-	-do-	-do-

Morphological Parsing

Limitations of exhaustive lexicon with features

- Puts heavy demand on *memory.*
  - List every form of the word=>large number of, redundant entries.
- Fails to capture *linguistic generalization. That means,*
  - Fails to show the *relationship* between *different roots* having *similar word forms.*
  - Essential to *develop a system* capable of understanding *unknown words*.
- For morphologically *complex languages* like Turkish, number of possible *word-forms* may be theoretically *infinite*.
- Not practical to list all possible word-forms for such languages.
- These Limitations made *morphological parsing* is *necessary*.

Morphological Parsing

Morphological Systems - Stemmers

- *Simplest* morphological system.
- Collapses morphological variations of a given *word* (word-forms) to *one lemma* or stem.
- *Do not require* a lexicon.
- Specifically used in Information Retrieval (IR)
- 2 widely used stemming algorithms has been developed by :
  - Lovins (1968)
  - Porter (1980)

Morphological Parsing

Morphological Systems - Stemmers

- Use *rewrite rules* of the form:

  ier → y (eg., *earlier* →*early)*

  ing → ε *(eg., playing* → *play*)

- *Stemming algorithms* work in **2 steps**:

  1.**Suffix removal:** Removes predefined endings from words.

  2.**Recoding**: Adds predefined endings to the output of the step 1.

- These two steps can be performed:

  1.Sequentially (Lovins's)

  2.Simultaneously (Porter's)

Morphological Parsing

Morphological Systems - Stemmers

Porter Stemmer(1980)

- Used for tasks in which you only *care* about the *stem*.
- IR, topic detection, document similarity etc.
- *Lexicon-free* morphological analysis.
- Cascades *rewrite rules*

  Example: *misunderstanding → misunderstand → understand …*
- Easily implemented as an FST with rules

  Example:*ier → y* (eg., *earlier →early)*

  ing → ε (eg., playing → play)

Morphological Parsing

Morphological Systems - Stemmers

Porter Stemmer(1980)

- Porters algorithm makes use of following transformational rule for the word : "rotational" ---> "rotate"

  ational ---> ate

- But transformation of the word 'organization' ---> 'organ' and 'noise' ---> 'noisy' are *not perfect*
- It reduces only suffixes; *prefixes* and *compounds* are *not reduced.*

Morphological Parsing

Two-level Morphological Model

- First proposed by Kimmo Koskenniemi (1983)
- Applicable to highly inflected languages.
- Word is represented as a *correspondence* between its *lexical form* and its *surface level form*.
- *Surface form* → Actual spelling of the word
- *Lexical form* → Concatenation of its constituent morphemes with morphological features.
- *Morphological parsing* -> Mapping from the surface level into morpheme and feature sequences on the lexical level.

Morphological Parsing

Two-level Morphological Model

- *Surface form* → Actual spelling of the word
- *Lexical form* → Concatenation of its constituent morphemes with morphological features.

Surface Level	p	l	a	y	i	n	g
Lexical	p	l	a	y	+V	+PP	

Surface Level	b	o	o	k	s	
Lexical	b	o	o	k	+N	PL

- **Example:** books → first component is the stem, *book.* And second component is the morphological information, that specifies the surface level form is a plural noun (+N+PL).

Morphological Parsing

Finite-State Transducer (FST):

- It is a kind of finite- state automata
- FSTs *map* between *one set of symbols* and *another* using a FSA, whose alphabet Σ is composed of *pairs* of symbols *from input* and *output alphabets*.
- FST is a *2-state automaton* that:
  - Recognizes
  - Generates a pair of strings

Morphological Parsing

Finite-State Transducer (FST):

- FST is a 6-tuple {Σ_1, Σ_2, Q, δ, S, F} consisting of
  - **Q:** set of states.
  - **Σ:** an alphabet of complex symbols, each an *i*:*o* pair s.t. *i* $\in$ Σ_1 (an input alphabet) an d *o* $\in$ Σ_2.
  - **Σ_1:** the alphabet of input
  - **Σ_2:** the alphabet of output
  - **S:** a start state
  - **F:** a set of final states in Q; F C Q
  - **δ:** a transition function mapping; Q x (Σ_1 U {ε}) x (Σ_2 U {ε}) to power set of Q.

Example *:hot → cot*

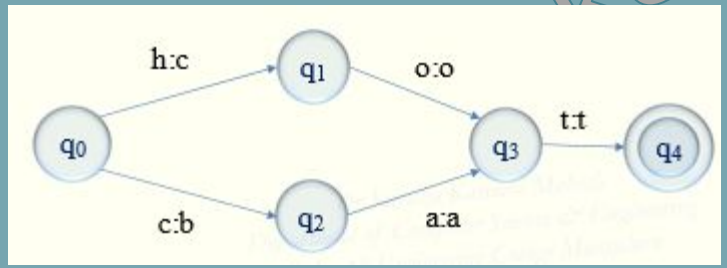
Morphological Parsing

Finite-State Transducer (FST):

- Below Figure, A simple *finite- state transducer* that accepts two input strings, *hot* and cat maps them onto *cot* and *bat* respectively.

Example :hot → cot

cat → bat



Finite-State transducer

Morphological Parsing

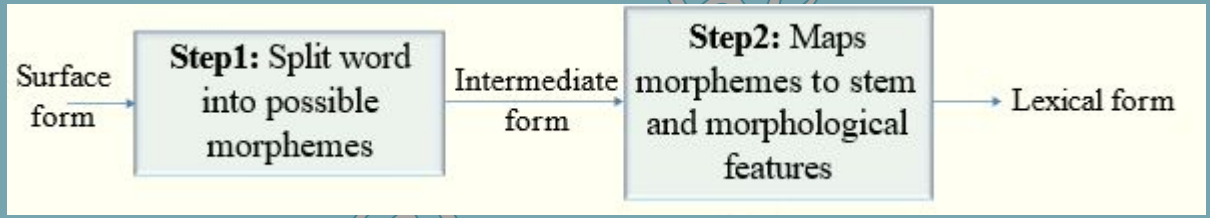
Finite-State Transducer (FST):

- Like *FSAs* encode *regular languages*, *FSTs* encode *regular relations.*
- Regular relation is the relation between regular languages.
- Regular language encoded on the upper side of an FST is called *upper language.*
- Regular language encoded on the lower side is termed as *lower language*.

Morphological Parsing

FST- Two level morphological parser:

- Implementing *Two-level Morphological Model* using FSTs to get from the *surface form* of a word to its morphological analysis i.e. *Lexical form*
- We proceed in *2 steps*:



bird	bird	bird + N +SG
birds	bird + s	bird + N +PL
goose	goose	goose + N + SG
geese	geese	geese + N + PL
boxes	box + s	box + N + PL

Figure: Two-step morphological parser.

Morphological Parsing

FST- Implementing two level morphological parsing:

- Implementing *Two-level Morphological Model* using FSTs Involves two steps:

  1. *Split* the words up into its possible components.

     cats → cat + s

- *Output* is *concatenation of morphemes*. i.e., *Stems + Affixes.*
- **+** indicates *morpheme boundaries.*
- Also, consider *spelling rules*.

  boxes → boxe + s — *boxe* is stem, *s* the suffix.

  box + *s* — *box* is stem, *e* has been introduced *due to* the *spelling rule.*

Morphological Parsing

Implementing two level morphological parsing:

- There can be *more than one representation* for a given word.

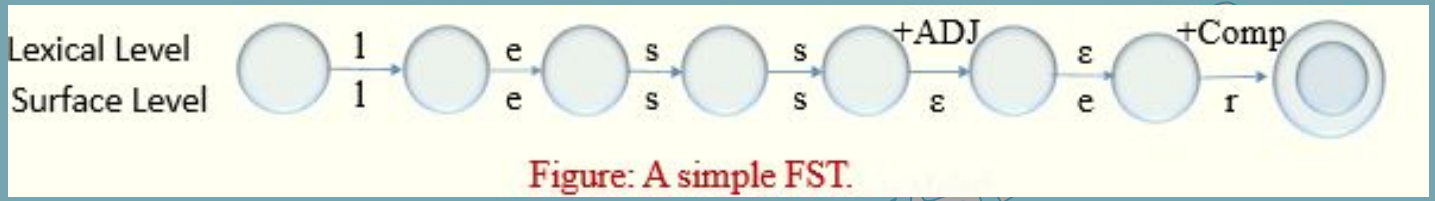


Figure: A simple FST.

- A transducer that does the *mapping* (translation) required by the first step for the *surface* form 'lesser' from *lexical* represents information that:
  - The *comparative form* of the *adjective* 'less' is *'lesser'*,
  - **ε** is the empty string.
- The automaton is inherently *bi-directional.*
- Same transducer can be used for *analysis* (surface input, 'upward application') or for *generation* (lexical input, 'downward' application).

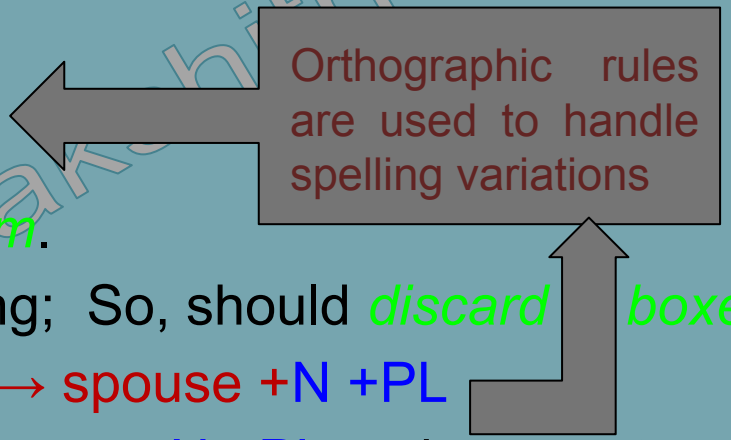
Morphological Parsing

Implementing two level morphological parsing:

2. Use a lexicon to *look up categories of the stems* and *meaning* of the *affixes*.

birds → bird + s → bird +N +PL

boxes → box + s → box +N +PL



- Learn that *'boxe'* is *not a legal stem*.
- boxe + s is incorrect way of splitting; So, should *discard* *boxes*.
- However, spouses → spouse + s → spouse +N +PL

  parses → parse + s → parse +N +PL is correct.

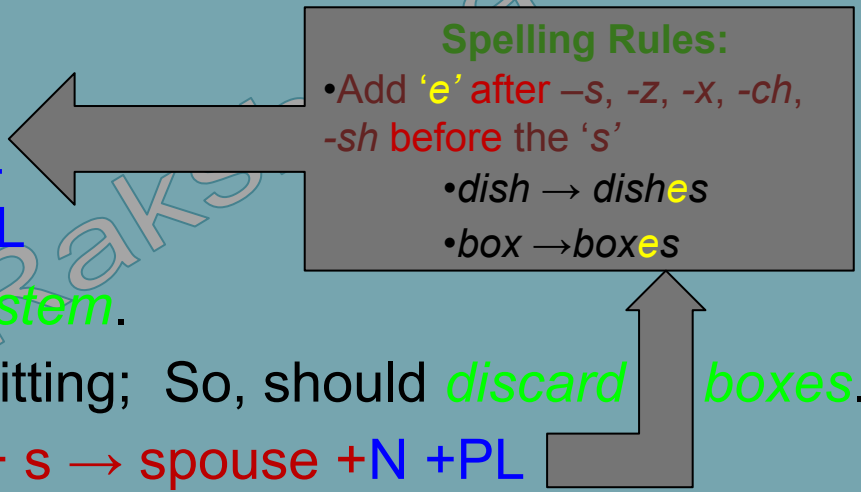
Morphological Parsing

Implementing two level morphological parsing:

2. Use a lexicon to *look up categories of the stems* and *meaning* of the *affixes*.

birds → bird + s → bird +N +PL

boxes → box + s → box +N +PL



- Learn that *'boxe'* is *not a legal stem*.
- boxe + s is incorrect way of splitting; So, should *discard* *boxes*.
- However, spouses → spouse + s → spouse +N +PL
  parses → parse + s → parse +N +PL are correct.

Morphological Parsing

Implementing two level morphological parsing:

- Implementation of the two steps with transducers requires building *two transducers*:
  - One that maps the *surface form* to the *intermediate form.*
  - Another that maps the *intermediate form to the lexical form.*

Morphological Parsing

Implementing Two-level Morphological Model using FSTs

- **Example:** FST-based morphological parser for singular and plural nouns in English.
- *Considerations:*
- *Plural form* of regular nouns *usually end with –s* or *–es.*
- However, a *word ending* in *'s' need not necessarily* be the *plural form* of a word.
- There are several singular words ending in 's' (e.g., *miss* and *bliss*).
- One of the *required translations* is the *deletion* of the *'e'*, when introducing a morpheme boundary.
- Required for words ending in *-xes*, *-ses, -zes* (e.g., *boxes* and *suffixes*).

Morphological Parsing

Implementing Two-level Morphological Model using FSTs

- FST-based morphological parser for singular and plural nouns in English.
- **Example:** Required for words ending in *-xes*, *-ses*, *-zes* (e.g., *boxes* and *suffixes*) must be deleted.Rest of the words appended with -s(eg:cats)

boxes → box +s
suffixes → suffix + s
cats → cat+s

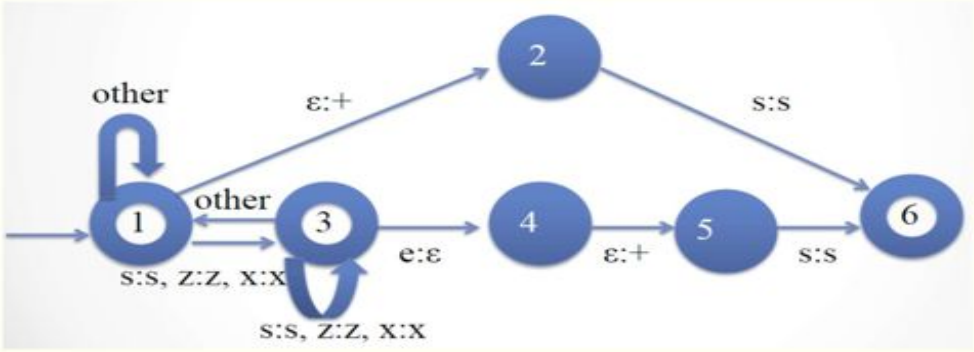


Figure: A simplified FST, mapping English nouns to the intermediate form.

Morphological Parsing

Implementing Two-level Morphological Model using FSTs:STEP1

- *Sequences of states* that the transducer undergoes, given the surface forms birds as input.

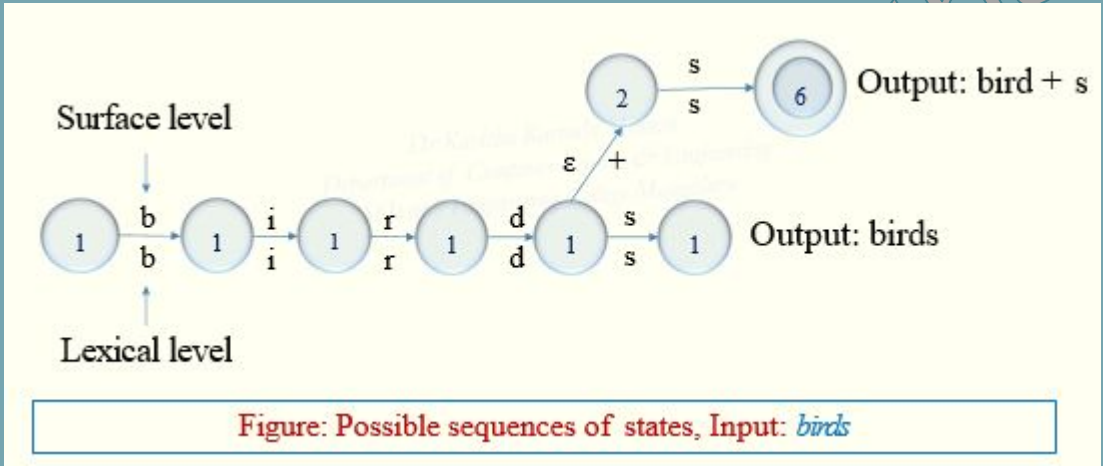


Figure: Possible sequences of states, Input: *birds*

Morphological Parsing

Implementing Two-level Morphological Model using FSTs:STEP1

- *Sequences of states* that the transducer undergoes, given the surface forms boxes as input.

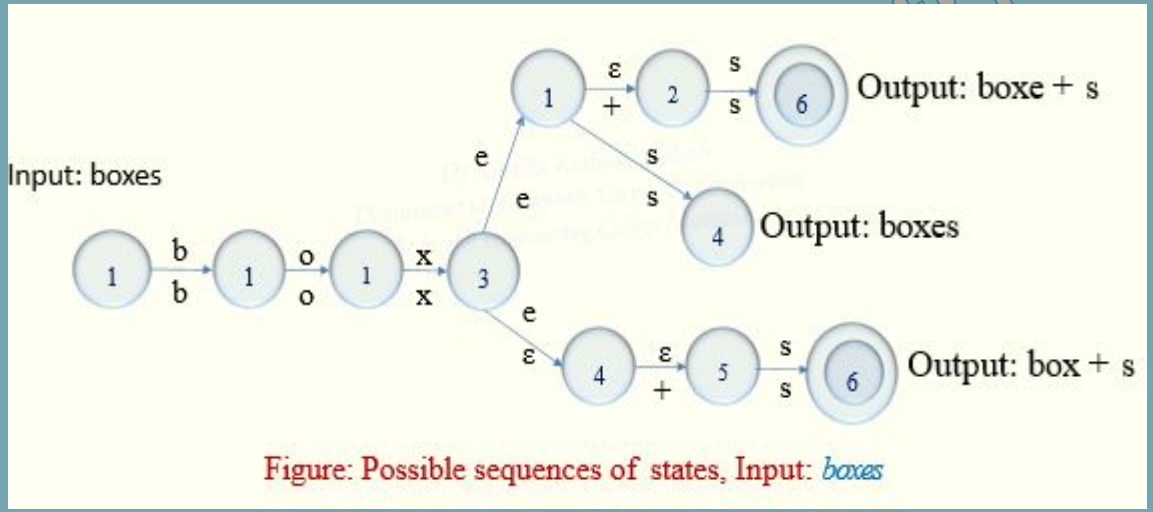
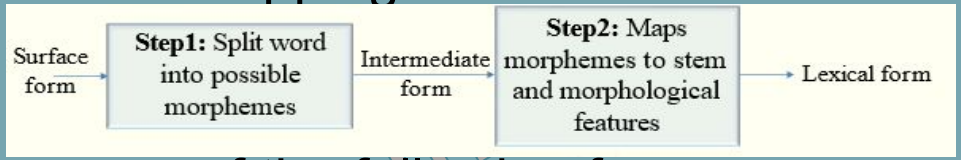


Figure: Possible sequences of states, Input: *boxes*

Morphological Parsing

Implementing Two-level Morphological Model using FSTs:STEP2

- Develop a transducer that does the mapping from the *intermediate level* to the *lexical level*.



- The input to the transducer has one of the following forms:

1. Regular noun stem, e.g., bird, cat
2. Regular noun *stem* +s, e.g., bird + s
3. Singular irregular noun stem, e.g., goose
4. Plural irregular noun stem, e.g., geese

Morphological Parsing

Implementing Two-level Morphological Model using FSTs:STEP2

- Develop a transducer that does the mapping from the *intermediate level* to the *lexical level*.
- The input to the transducer has one of the following forms:

1. Regular noun stem, e.g., bird, cat → *Map all symbols of the stem to themselves* and then output *N* and *sg*.
2. Regular noun *stem +s*, e.g., bird + s → *Map all symbols of the stem to themselves*; but then output *N* and *replace PL with s*.
3. Singular irregular noun stem, e.g., goose → Same as in first case
4. Plural irregular noun stem, e.g., geese → *Map irregular plural noun stem to the corresponding singular stem* and *add N* and *PL.*

Morphological Parsing

Implementing Two-level Morphological Model using FSTs:STEP2

- Develop a transducer that does the mapping from the *intermediate level* to the *lexical level*.

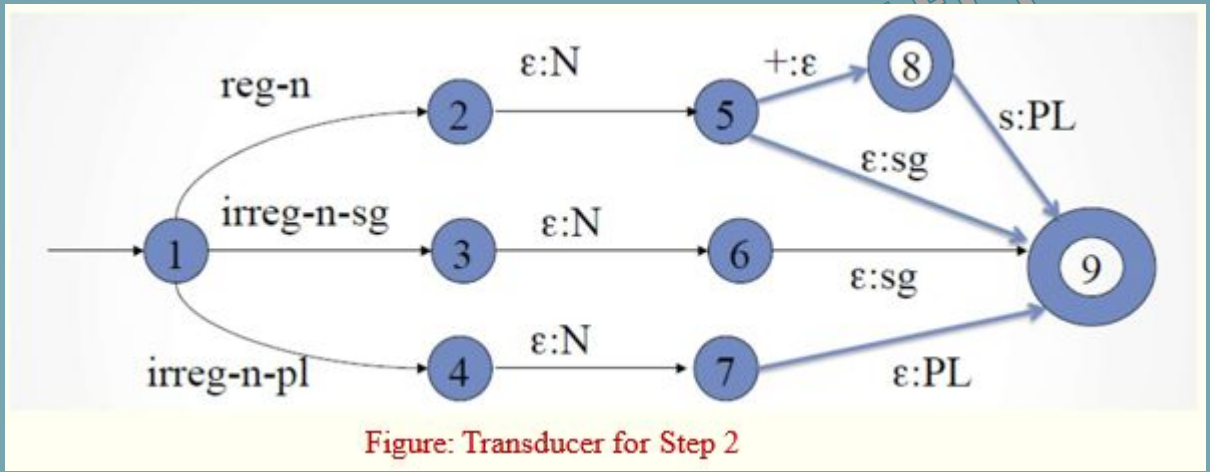
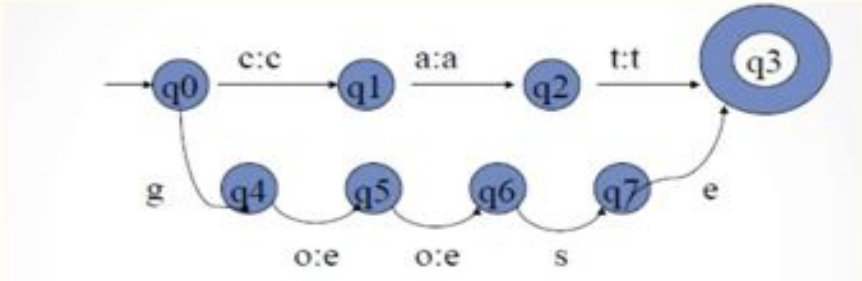


Figure: Transducer for Step 2

Morphological Parsing

Implementing Two-level Morphological Model using FSTs:STEP2

Surface: c : a : t
Lexical : c : a : t
so,
Pairs c:c → c
c a t



Reg-n	Irreg-pl-n	Irreg-sg-n
c a t	g o:e o:e s e	g o o s e

Figure: Stem Mappings

Morphological Parsing

Implementing Two-level Morphological Model using FSTs:STEP2

- Composing this transducer with previous STEP 1, we get 2 level transducer.Produces output to *Lexical level* (downward direction)

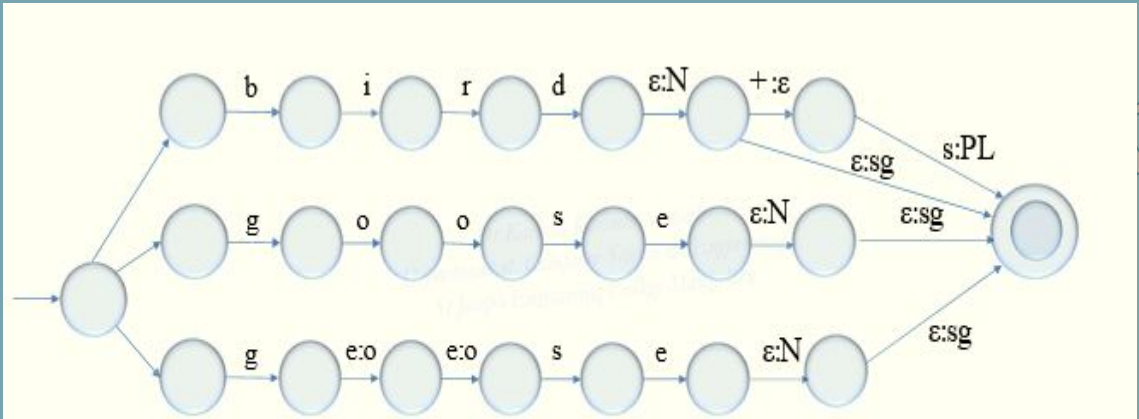


Figure: A transducer mapping nouns to their stem and morphological features

Intermediate word→ Lexical Level

bird+s → bird +N+PL

b:b i:i r:r d:d +ε :N +s:PL

Stem

Morphological features

Spelling Error Detection and Correction

- Errors of typing and spelling constitute a very common source of variation between strings.
- **Sources of Common String Variations**
  - Typing errors
  - Spelling errors
- In early investigations,80% were single-error misspellings (Damearu, 1964)
- Common *typing mistakes* involve *single character :*
  - Omission
  - Insertion
  - Substitution
  - reversal (transposition) of two adjacent letters

Spelling Error Detection and Correction

Sources of Common String Variations – Typing Errors:

1. *Omission* of a single letter

   Example : '*concept*' typed as '*concpt*'

2. *Insertion* of a single letter

   Example : '*error*' typed as '*errorn*'

3. *Substitution* of a single letter

   Example :'*error*' typed as '*errpr*'

4. *Transposition* of two adjacent letters

   Example : '*are*' typed as '*aer*'

- Most common type was *substitution*, followed by *omission* of a letter and then *insertion* of a letter. *(Shafer and Hardwick, 1968)*

Spelling Error Detection and Correction

Sources of Common String Variations – Typing Errors:

- Optical character recognition (OCR)/Automatic reading devices introduce errors are grouped into *five classes*:

1. Substitution
   - **Example :** due to visual similarity (*1→l, c → e, r → n*)
2. Multi-substitution (or framing)
   - **Example :** due to visual similarity *m → rn*
3. Space deletion
4. Space insertion
5. Failures when OCR algorithm fails to select a letter with sufficient accuracy.
   - ***Error Correction*** using '*context*' or using '*linguistic structures*'.

Spelling Error Detection and Correction

- **Sources of Common String Variations – Spelling Errors**
- Many approaches to *speech recognition* deal with strings of *phonemes*(symbols representing sounds), attempt to match a spoken utterance with a dictionary of known utterances.
- In speech recognition, errors are mainly *phonetic*.
- Misspelled word is *pronounced in the same way* as correct word.
- *Distort the word* by more than a single insertion, deletion or substitution.
- Phonetic variations are common in transliteration.

Spelling Error Detection and Correction

Spelling Error – Examples

- Two distinct categories:

  1.Non-word errors

  2.Real-word errors

1.Non-word errors

- Error resulting in a *word* that does *not appear in* a given *lexicon* or is *not a valid orthographic word.*
- *Techniques for detection (Now considered a solved problem)*
  - n-gram analysis
  - dictionary lookup

Spelling Error Detection and Correction

Spelling Error – Examples

2.Real-word errors

- Occurs in *actual words* of the language.Due to :
  - Typographical mistakes or Spelling errors
- **Example :** Substituting the spelling of a *homophone* or *near-homophone* such as:'*piece*' for '*peace*' or '*meat*' for '*meet*'
- Real-word errors may cause *local syntactic errors*, *global syntactic errors*,*semantic* errors or errors at *discourse* or *pragmatic* levels.
- *Impossible* to decide that a *word is wrong* without some *contextual information*.

Spelling Error Detection and Correction

Spelling Correction Approaches

- Consists of:
  - *Detecting Errors*- Finding misspelled words.
  - *Correcting Errors*- Suggesting correct words to a misspelled one.

Approaches addressed in 2 ways:

1. Isolated-error detection and correction.
2. Context-dependent error detection and correction.

Spelling Error Detection and Correction

1. **Isolated-error detection and correction.**

- Each word is checked separately, independent of its context.
- Why not Dictionary Lookup ? → *Problems !!!*

  1.Requires the existence of a lexicon containing all correct words – *compilation time and space issues*.

  2.Highly productive languages → *impossible* to list *all correct words*.

  3.Strategy fails when spelling error produces a word that belongs to a lexicon.

  Example : '*theses*' in place of '*these*' (Real-word error).

- 4. *Larger the lexicon* → Error goes *undetected*.
  - chance of word being found in larger lexicon is greater.

Spelling Error Detection and Correction

2. Context-dependent detection and correction

- Use the *context* of a word to detect and correct errors.
- Requires *grammatical analysis*
- More *complex*
- *Language dependent*
- Employs *isolated-word method* to obtain *candidate words* before making a selection depending on context.

Spelling Error Detection and Correction

Spelling Correction Algorithms: Broadly classified by Kukich (1992) as follows:

- Similarity key techniques
- n-gram based techniques
- Neural nets
- Rule-based Techniques
- Minimum Edit Distance

Spelling Correction Algorithms

Similarity key techniques:

- Change given *string into a key* such that *similar strings* will change into the *same key*.
- Used in SOUNDEX system for phonetic spelling correction applications.

For your reference:

Get the Soundex code for the surname JONES

- Write the first letter: J
- Scratch out all letters whose code are zero: 'O' and 'E'
- code the remaining letters:
  - N code is '5': 5
  - S code is '2': 2
- No more letters are left but need the required three digits, so fill the remaining positions with zero: 0
- The Soundex for Jones is J520

Soundex Code	Letters
1	B, F, P, V
2	C, G, J, K, Q, S, X, Z
3	D, T
4	L
5	M,N
6	R
No Code	A, E, I, O, U, H, W, Y

Spelling Correction Algorithms

n-gram based techniques:

- Can be used for *real-word* and *non-word* error detection.

n-gram for Non-word error Detection

- Based on the idea that certain *bi-grams* and *tri -grams never occur* or *rarely occur.*

  Example: *qst*, *qd*

- Strings containing these *unusual n-grams* → *possible spelling errors*
- Require large corpus/dictionary as training data for *compiling n-gram table of possible combinations of letters.*

Spelling Correction Algorithms

n-gram based techniques:

- Can be used for *real-word* and *non-word* error detection.

n-gram for real-word error Detection

- Calculate the *Likelihood of one character following another.*
- Use the information to find possible correct word candidates.

Spelling Correction Algorithms

Neural Nets:

- Have the ability to do *associative recall* based on incomplete and noisy data.
- Train neural nets to adapt to *specific spelling error patterns*.
- Computationally *expensive*.

Note: reference content for more information:
Cherkassky, Vladimir, et al. *"Conventional and associative memory approaches to automatic spelling correction."* Engineering Applications of Artificial Intelligence 5.3 (1992): 223-237.

Spelling Correction Algorithms

Rule-based Techniques:

- *Set of rules* (heuristics) derived from knowledge of *common spelling error pattern* is used to *transform misspelled words* into *valid words*.
- Observation: many error occurrence occurs from '*ue*' typed as '*eu*'

  Form a rule *ue → eu*

Spelling Correction Algorithms

Minimum Edit Distance:

- **Definition:** Minimum number of operations (*insertions(I)*, *deletions(D)*, *substitutions/replacements(R)*) required to transform one string into another.
- Edit distance is also called as the *Levenshtein distance*.
- A string over the alphabet I, D, R, M(Minimum) that describes a transformation of one string to another is called as an *edit transcript*.

Spelling Correction Algorithms

Minimum Edit Distance:

- **Example:** Transformation of string "*tutor*" to "*tumour*" and its associated *'edit transcript '* (*MMRMIM*).

M	M	R	M	I	M
t	u	t	o		r
t	u	m	o	u	r

Here, Minimum Edit Distance = 2

M	M	D	R	M	I	M
t	u	t	_	o	_	r
t	u	_	**m**	o	u	r

Here, Minimum Edit Distance = 3

- So, we have better alignment as cost of **2**

Spelling Correction Algorithms

Minimum Edit Distance: Permitted edit operations:

- Insertion – I

  A dash in the upper string indicates *Insertion*.
- Deletion – D

  A dash in the lower string indicates *Deletion*.
- Substitution (Replacement) – R

  A *Substitution* occurs when the two alignment symbols do not match.
- *Levenshtein Distance* between two sequences is obtained by assigning a unit cost to each operation.

Spelling Correction Algorithms

Minimum Edit Distance:

- Edit distance is viewed as a *string alignment problem.*
- An alignment is an equivalent alternative to an edit transcript for indicating differences and similarities between strings.
- By aligning two strings, we can *measure the degree* to which they *match*.
- As above example, there can be *more than one possible alignment* between two strings.
- The *best possible alignment* corresponds to the *minimum edit distance* between the strings.

Spelling Correction Algorithms

Minimum Edit Distance:

- Minimum Edit Distance between two strings can be represented as a binary function, **ed**, which maps two strings to their edit distance.
- **ed** is symmetric.
- For any two strings, *s (source)* and *t (target)*, **ed(s, t)** is always equal to **ed(t, s)**.

Spelling Correction Algorithms

Dynamic Programming for finding Minimum Edit Distance

- Table-driven approach is applied to solve problems by combining solutions to sub-problems.
- The most *classic inexact matching problem* solved by *dynamic programming*: the *edit distance problem*.
- Dynamic programming algorithm for minimum edit distance is implemented by created an *edit distance matrix*.
- The matrix has *one row* for *each symbol* in the source string and *one column* for *each symbol* in the target string.

Spelling Correction Algorithms

Dynamic Programming for finding Minimum Edit Distance

- The *(i, j)th cell* in this matrix represents the *distance* between the *first i* characters of the source and the *first j characters* of target string.
- Each cell is computed as a simple function of its *surrounding cells*.
- By starting at the beginning of the matrix, it is possible to *fill each entry iteratively*.

Spelling Correction Algorithms

Dynamic Programming for finding Minimum Edit Distance

- The edit distance between strings *X[1.. n]* and *Y[1.. m]* can be computed applying dynamic programming.
- Define *dist(i, j)* to be the edit distance of prefixes *X[1..i]* and *Y[1..j]*
- *dist(n, m)* is the edit distance of X and Y.
- Dynamic programming computes *dist(n, m)* by computing *dist(i, j)* for all $i \leq n$ and $j \leq m$.

Spelling Correction Algorithms

Dynamic Programming for finding Minimum Edit Distance

- How to determine the dist(i, j) values ?
- **Base conditions for dist(i, j):**

  dist[i, 0] = i

  dist[0, j] = j

- How to edit *first i characters of string X* to *zero character of Y* ?

  With *i deletions of X* !! → **dist[i, 0] = i**, i , 0 ≤ i ≤ n.

- How to transform *zero characters of string X* to *j characters of Y* ?

  Insert j characters of Y !! → **dist[0, j] = j**, j , 0 ≤ j ≤ m.

Spelling Correction Algorithms

Dynamic Programming for finding Minimum Edit Distance

- **Example:**Finding minimum edit distance between string tutor and tumour

	#	t	u	m	o	u	r
#							
t							
u							
t							
o							
r							

Spelling Correction Algorithms

- **Example:**Finding minimum edit distance between string tutor and tumour

dist[0,j] ← j

Base Condition:

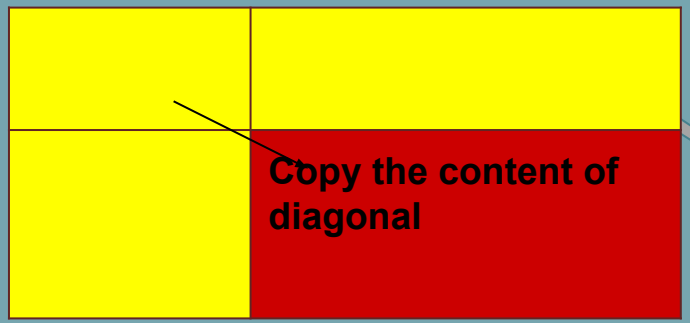
dist[i,0] ← i

	#	t	u	m	o	u	r
#	0	1	2	3	4	5	6
t	1						
u	2						
t	3						
o	4						
r	5						

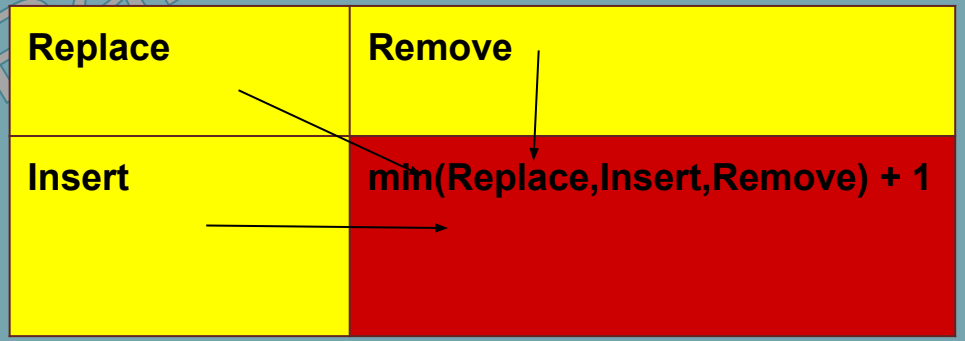
Spelling Correction Algorithms

- **How to compute the remaining cells?**
- Inner cells can be computed in any order
- row-wise,column-wise or in successive anti-diagonals such that the *three values* required by the recurrence *have been computed*.

If source == target



If source != target



Spelling Correction Algorithms

	#	t	u	m	o	u	r
#	0	1	2	3	4	5	6
t	1						
u	2		dist[i-1,j-1]	dist[i,j-1]			
t	3		dist[i-1,j]	dist[i,j]			
o	4						
r	5						

Spelling Correction Algorithms

- **Inductive Case for i, j > 0:to compute the remaining cells**

  - dist(i − 1, j) + *insert_cost*,
  - dist (i, j) = **min** [dist(i, j−1) + *delete_cost*,
  - dist (i−1, j−1) + *sub_cost* t(source$_i$, target $_j$)]

- Where t(source$_i$, target$_j$) = 0, if (source[i] = = target[j])

  = 1, if (source[i] ≠ target[j])

Spelling Correction Algorithms

- **Example:**Finding minimum edit distance between string tutor and tumour

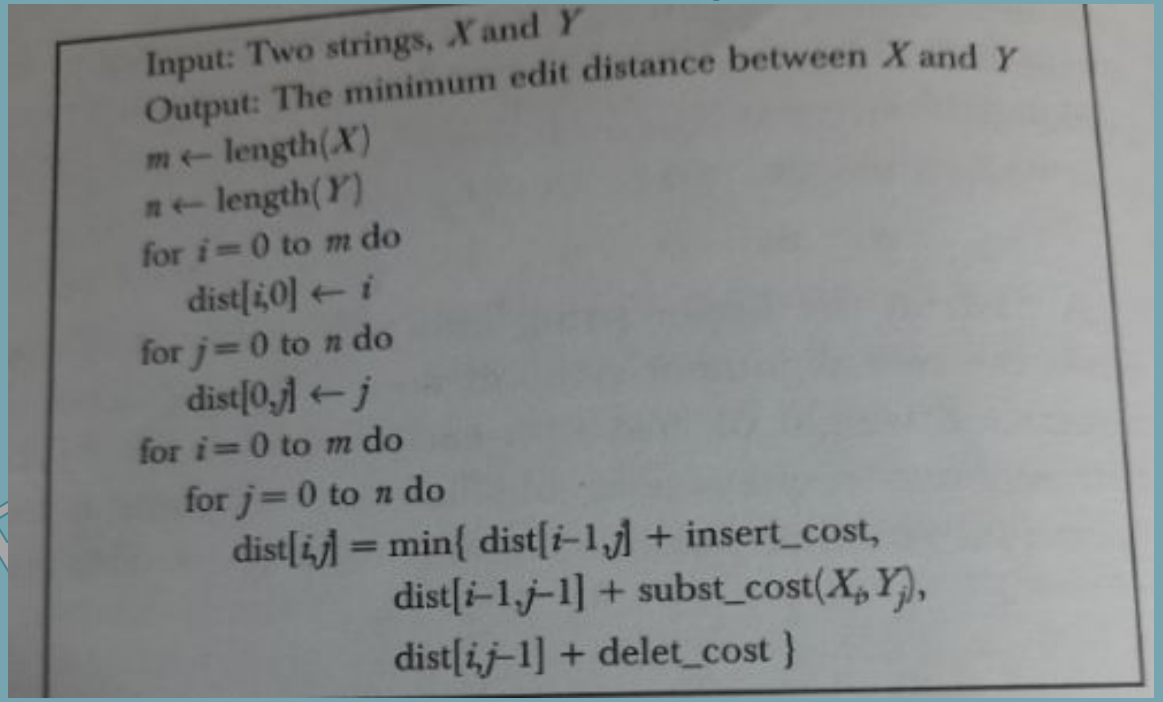
	#	t	u	m	o	u	r
#	0	1	2	3	4	5	6
t	1	0	1	2	3	4	5
u	2	1	0	1	2	3	4
t	3	2	1	1	2	3	4
o	4	3	2	2	1	2	3
r	5	4	3	3	2	2	**2**

Minimum Edit Distance is: 2
dist(m,n):2

Spelling Correction Algorithms

- *Minimum edit distance algorithm* between two strings can be shown as below:

- Useful for determining accuracy in *speech Recognition systems*



Input: Two strings, X and Y
Output: The minimum edit distance between X and Y
$m \leftarrow$ length(X)
$n \leftarrow$ length(Y)
for $i = 0$ to m do
  dist[i,0] $\leftarrow i$
for $j = 0$ to n do
  dist[0,j] $\leftarrow j$
for $i = 0$ to m do
  for $j = 0$ to n do
    dist[i,j] = min{ dist[i–1,j] + insert_cost,
                dist[i–1,j–1] + subst_cost(X_i, Y_j),
                dist[i,j–1] + delet_cost }

Words and word classes

- Words are classified into categories called parts-of-speech.
- Referred to as *word classes* or *lexical categories.*
- Lexical Categories are defined by their syntactic and morphological behavior.
- Common lexical categories: Nouns, Verbs
- Other lexical categories: Adjectives, adverbs, prepositions and conjunctions

Words and word classes

- Word classes in English are as shown below:

Word Classes in English

NN	noun	*student, chair, proof, mechanism*
VB	verb	*study, increase, produce*
ADJ	adj	*large, high, tall, few*
JJ	adverb	*carefully, slowly, uniformly*
IN	preposition	*in, on, to, of*
PRP	pronoun	*I, me, they*
DET	determiner	*the, a, an, this, those*

Words and word classes

Categories of Word Classes

- Open Word Classes
- Closed Word Classes

Open Word Classes

- *Constantly acquire* new members.
- Nouns, verbs (except auxiliary verbs), adjectives, adverbs, and interjections. *Covid* (Noun), *Simples* (Noun), *chillax* (Verb), *whatevs* (Adv), *buzzy* (Adj), *Phew, Eww, Ooh-la-la* (Interjection).

Closed Word Classes

- *Do not* or *only infrequently* acquire new members.
- Prepositions, auxiliary verbs, delimiters, conjunction, and particles.

Part-of-Speech Tagging

- Part-of-speech (POS) Tagging is the *process of assigning* a *part-of- speech*(noun,verb ,pronoun,preposition, adverb and adjective) to *each word* in a sentence.
- Input: Sequence of words of a natural language sentence.
- Output: Single best POS tag for each word.
- **Example:**

Book/VB *that*/DT *flight*/NN

Part-of-Speech Tagging

How do we decide which words go in which classes?

- Many words may belong to more than one lexical category.
- **Example:** In English, word 'book' can be as noun:

  I am reading a good book
- Word 'book' can be as verb:

  The police booked the snatcher
- **Example:** In Hindi, word 'sona' (noun /verb ???) means gold(noun) or sleep(verb)
- Only one of the possible meanings is used at a time.
- Determine the correct lexical category of a word in its context.
- *Tag* assigned by a tagger is the *most likely* for a *particular use* of word in a sentence.

Part-of-Speech Tagging

Tag set

- *Collection of tags* used by a particular tagger.

Tag set types - What set of parts of speech do we use?

- Most use some basic categories – noun, verb, adjective, preposition
- Tag sets differ in how they define categories and how they divide words into categories.
- Example: 'eat' and 'eats' tagged as:
  - Coarse-grained – *eat*/*eats* --Verb
  - Fine-grained – distinct tags
  - *eats*/V → *eat*/VB, *eat*/VBP, eats/VBZ, ate/VBD, aten/VBN *eating*/VBG, ...

Part-of-Speech Tagging

- Additionally, capture *morpho-syntactic* information. (SG /PL /number/ gender /tense).
- Consider the following sentences:

  Zuha eats an apple daily.
  Aman ate an apple yesterday
  They have eaten all the apples in the basket
  I like to eat guavas

- The word *eat* has *distinct grammatical form* in each of the four sentences.
  - *eat* → Verb, base form;
    Verb, *3rd person singular present*
  - *ate* → Verb, *past tense*
  - *eaten* → Verb, *past participle*
  - *eats* → Verb non-*3rd person singular present*

Part-of-Speech Tagging

- Consider the following ***ungrammatical*** sentences :

  They eaten all the apples

  I like to eats guava

What set of parts of speech do we use?

- *Number of tags* used by different taggers varies.
- There are various standard tag sets to choose from:
- Some have a *lot more tags* than others.
- *Accurate tagging* can be done with even large tag sets.
- The *larger the tag set*, the greater the information captured about a linguistic context.The choice of tag set is based on the application.

Part-of-Speech Tagging

What set of parts of speech do we use?

- Tags from Penn Treebank tag set: contains nearly *45 tags*

Number	Tag	Description
1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential *there*
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural
16.	PDT	Predeterminer
17.	POS	Possessive ending
18.	PRP	Personal pronoun
19.	PRP$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
23.	RP	Particle
24.	SYM	Symbol
25.	TO	*to*
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or present participle
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd person singular present
33.	WDT	Wh-determiner
34.	WP	Wh-pronoun
35.	WPS	Possessive wh-pronoun
36.	WRB	Wh-adverb

Possible tags for the word to ***eat***	
eat	VB
ate	VBD
eaten	VBN
eats	VBP

Part-of-Speech Tagging

What set of parts of speech do we use?

- Number of tags used by different taggers varies.
- These categories are based on *morphological* and *distributional similarities* (what words/types of words occur on the two sides of a word) and not, as you might think, semantics.
- In some cases, *tagging is fairly straightforward*, in other cases it is not.
- Tagging might become *complicated* and require *manual correction.*

Part-of-Speech Tagging

What set of parts of speech do we use?

Number of tags used by different taggers varies.

http://www.comp.leeds.ac.uk/amalgam/tagsets/tagmenu.html

- Brown Corpus (Francis & Kucera ‘82), 1M words, 87 tags.

  http://www.comp.leeds.ac.uk/amalgam/tagsets/brown.html

- Penn Treebank: hand-annotated corpus of Wall, Street Journal, 1M words, 45-46 tags

  http://www.comp.leeds.ac.uk/amalgam/tagsets/upenn.html

Part-of-Speech Tagging

What set of parts of speech do we use?

- **Example:** Number of tags used by different taggers *varies*.
- Tagset used is Penn Treebank /Tagged sentences:

  Speech/NN sounds /NNS were/VBD sampled/VBN by/IN a/DT microphone/NN

- Another tagging possible for the above sentence as:

  Speech/NN sounds /VBZ were/VBD sampled/VBN by/IN a/DT microphone/NN → tagged sequence is not correct. Leads to semantic incoherence.

Part-of-Speech Tagging

Applications of Parts of speech tagging:

- Is an early stage of text processing in many *NLP applications* including speech synthesis, machine translation, information retrieval and information extraction.
- Tagging is *not complex* as parsing.
- In tagging, a *complete parse tree is not built*; part-of-speech is assigned to words using contextual information.

Part-of-Speech Tagging

Categories of Parts of speech tagging:

- **Part-of speech tagging** methods fall under the **3 general categories**:
  - Rule-based(linguistic)
  - Stochastic (data-driven)
  - Hybrid

Part-of-Speech Tagging

Rule-based Taggers:

- *Hand-coded rules* to assign tags to words
- Use lexicon to obtain a list of candidate tags.
- Then use rules to discard incorrect tags.
- Rule-based taggers are *2 stage architecture*:

1. Simply dictionary lookup procedure→ returns a set of potential tags(parts-of-speech)and appropriate syntatic features for each word.
2. Set hardcoded rules → to discard contextually illegitimate tags to get single part-of speech for each word.

Part-of-Speech Tagging

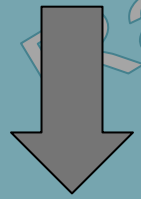
Rule-based Taggers:Example

The show must go on

Potential tags for *'show'* {VB, NN}

- **Ambiguity resolution by following rule:**

IF preceding word is determiner THEN eliminate VB tag



- Rule disallows verbs after a determiner → *'show'* can only be a *noun*

Part-of-Speech Tagging

Rule-based Taggers:Example

- **Morphological information**

  IF word ends in –ing and preceding word is a verb THEN label it a verb (VB)

- **Capitalization information**
- Tagging of *unknown nouns*.

Part-of-Speech Tagging

Rule-based Taggers:Disadvantages:

- *Time spent* in writing rule set.
- Usable for *only one language*.Using to another language requires a rewrite of most of the program

Part-of-Speech Tagging

Stochastic Taggers:

- Standard stochastic tagger algorithm is HMM (Hidden Markov Model) tagger

Markov model

- Probability of a chain of symbols ~ probabilities of its parts or *n-grams.*
- Simplest n-gram model is the *unigram model*, which assigns the most likely tag(part-of speech) to each token

Part-of-Speech Tagging

Stochastic Taggers:

Unigram model

- Assigns the *most likely tag (POS)* to each token
- Needs to be trained using a *tagged training corpus* before it can be used to tag data.
- Most likely statistics are gathered over the corpus and used for tagging
- *Context* used by the *unigram tagger* is the *text of the word* itself.

Part-of-Speech Tagging

Stochastic Taggers:

Unigram model

Example:

(1)She had a *fast*

(2)Muslims *fast* during Ramadan

(3)Those who were injured in the accident need to be helped *fast*.

- Since *fast* is used as an *adjective* frequently tagger will assign as JJ than considering it as in the example as *noun,verb or adverb*. This results *incorrect tagging*
- More context → more accurate predictions – *Better tagging* decision

Part-of-Speech Tagging

Stochastic Taggers:

Bigram model

§*Current word* and the *tag of the previous word* is used in the tagging process.

Example:

1. *She had a fast*
2. *Muslims fast during Ramadan*
3. *Those who were injured in the accident need to be helped fast.*

More likely tag sequence

DT NN vs DT JJ (JJ → Adjective)

- Bigram model will *assign* a correct tag (NN) to the word *fast* in (1).
- Bigram model will *assign* a correct tag (RB-adverb) to the word *fast* in (3) which follows verb.

Part-of-Speech Tagging

Stochastic Taggers:

n-gram model

- *Current word* and the *tag of the previous n-1 words* in assigning the tag to a word.
- *Context* in a tri-gram model

Token	w_{n-2}	w_{n-1}	w_n	w_{n+1}
Tags	t_{n-2}	t_{n-1}	t_n	t_{n+1}

Part-of-Speech Tagging

Stochastic Taggers:

Tagging a Sentence

Input: Sequence of words (Sentence)

Objective: Find the most probable tag sequence for the sentence

- Let **W** be the sequence of words,

  W= $w_1, w_2, ... w_n$

- The task is to find the tag sequence **T**

  T= $t_1, t_2, ... t_n$ which maximizes P(T/W)

Part-of-Speech Tagging

Stochastic Taggers:

Tagging a Sentence

- The task is to find the tag sequence:

  $T = t_1, t_2, ... t_n$ which maximizes $P(T|W)$

  i.e., $T^I = \text{argmax}_T P(T|W)$

- *Applying Bayes Rule*, **P(T|W)** can be estimated using expression:

  $P(T|W) = P(W|T) * P(T)/P(W)$ ------(1)

Here, **P(W)** → Probability of the word sequence Remains the same for each tag sequence, So we can drop it.

Expression will be: **$T^I = \text{argmax}_T P(W|T) * P(T)$ ------(2)**

Part-of-Speech Tagging

Stochastic Taggers:

Tagging a Sentence

- Using Markov assumption,

$$P(T) = P(t_1) * P(t_2|\ t_1) * P(t_3|\ t_1\ t2) .. .. .. * P(t_n|\ t_1\ t_2 \dots\ t_{n-1}) \quad \text{------(3)}$$

P(W|T) → Probability of seeing a word sequence, given a tag sequence.
Example: Probability of seeing

'The tomato is rotten' given **'DT NNP VB JJ'**

Assumptions:

- The words are independent of each other.
- The probability of a word is dependent only on its tag.

Part-of-Speech Tagging

Stochastic Taggers:

Tagging a Sentence

- **P(W|T)** → Probability of seeing a word sequence, given a tag sequence.

$$P(W|T) = P(w_1|t_1) * P(w_2|t_2) * .. .. * P(w_i|t_i) * .. ....P(w_n|t_n)$$

$$\Downarrow$$

$$\mathbf{P(W|T) = \Pi_{i=1}^{n} P(w_i|t_i) \text{ -------(4)}}$$

Part-of-Speech Tagging

Stochastic Taggers:

Tagging a Sentence

$$T^{I} = \text{argmax}_{T} P(W|T) * P(T) \quad \text{------(2)}$$

- So, using equations (3) and (4) in equation (2),
- We have:

$$P(W|T) * P(T) = \Pi_{i=1}^{n} P(w_i|t_i) * P(t_1) * P(t_2|t_1) * P(t_3|t_1\, t2) \ldots * P(t_n|t_1\, t_2 \ldots t_{n-1})$$

- Approximating the *tag history* using only the *two previous tags*,

$$P(T) = P(t_1) * P(t_2|t_1) * P(t_3|t_1\, t_2) \ldots * P(t_n|t_{n-2}\, t_{n-1})$$

Part-of-Speech Tagging

Stochastic Taggers:

Tagging a Sentence

- So, we have:
  $P(W|T) * P(T) = \Pi_{i=1}^{n} P(w_i|t_i) * P(t_1) * P(t_2|t_1) * \Pi_{i=3}^{n} P(t_i|t_{i-2}\, t_{i-1})$
- Estimating the probability from relative frequencies via Maximum Likelihood Estimation as:

$$P(t_i|t_{i-2}, t_{i-1}) = \frac{C(t_{i-2}, t_{i-1}, t_i)}{C(t_{i-2}, t_{i-1})} \qquad P(w_i|t_i) = \frac{C(w_i, t_i)}{C(t_i)}$$

- Where $C(t_{i-2}, t_{i-1}, t_i)$ is the number of occurrences of t_i followed by t_{i-2}, t_{i-1}

Part-of-Speech Tagging

Stochastic Taggers:

Tagging a Sentence

- **Example: *'The tomato is rotten'* given 'DT NNP VB JJ' - using trigram model**
- **=P(W/T)*P(T)**

P(The/DT)*P(tomato/NNP)*P(is/VB)*P(rotten/JJ)*P(DT)*P(NNP/DT)*P(VB/DT,NNP)* P(JJ/NNP,VB)

- Here, P(VB/DT,NNP) calculated as:C(DT,NNP,VB)/C(DT,NNP)

Part-of-Speech Tagging

Stochastic Taggers

- Stochastic models have the advantage of being *accurate* and *language independent.*
- Most of the stochastic taggers have accuracy: 96-97%, measured as a percentage of words
- 96% means:for a sentence containing 20 words, error rate per sentence will be 1-0.96^20= 56%, ~ one word per sentence
- Drawback:Taggers requires manually *tagged corpus* for *training*.

Part-of-Speech Tagging

Stochastic Taggers – Example Problem

- Consider the *sentence*:

  The bird can fly

- And the *tag sequence*

  DT NNP MD VB

- Using *bi-gram approximation*, find the probability of the given sentence

  = ***P(the | DT) * P(bird | NNP) * P(can | MD) * P(fly |VB)*** * ***P(DT) * P(NNP | DT) * P(MD | NNP) * P(VB |MD)***

Part-of-Speech Tagging

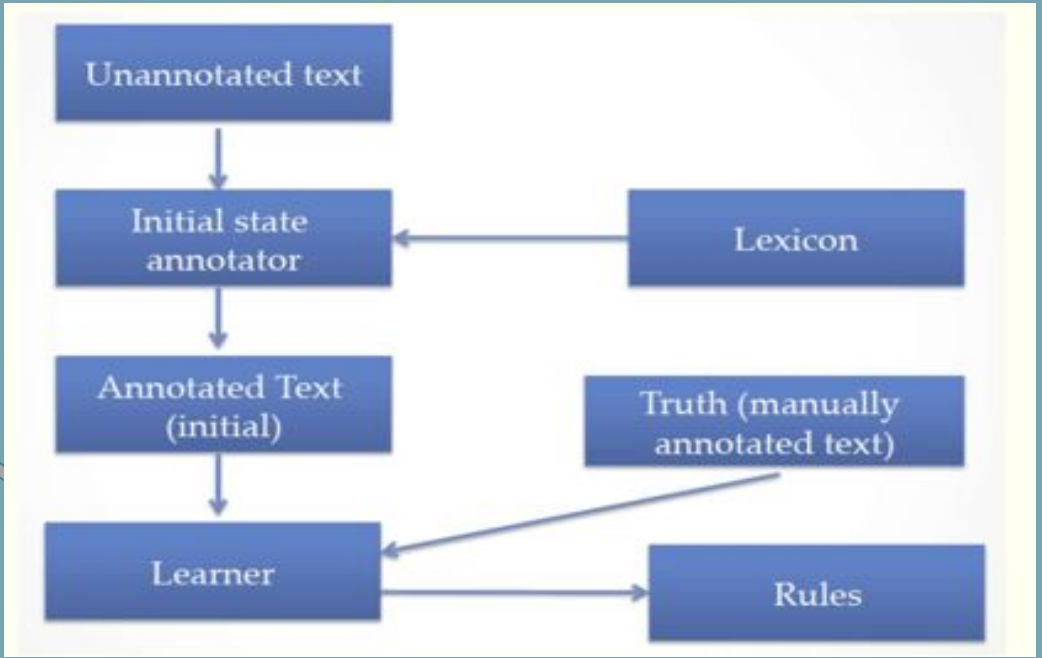
Hybrid Taggers

- Combine the features of both the *rule based* and *stochastic approaches*
- *Rules* are used to *assign tags to words*, and rules are *automatically induced* from the data.
- *Brill tagging* (E Brill, 1995), based on Transformation-based learning (TBL)of tags, is an example of hybrid approach.
- TBL is a *Supervised machine learning technique.*

Part-of-Speech Tagging

Hybrid Taggers

INPUT: *Tagged corpus* and *lexicon*
Step 1: Label every word with most likely tag using lexicon
Step 2: Check every possible transformation and select which improves tagging
Step 3: Retag corpus applying rules
Repeat : 2-3 until some stopping criteria is reached
RESULT: *Ranked sequence of transformation rules*



Part-of-Speech Tagging

Hybrid Taggers -Example:

- Assume that in a corpus, *fish* is most likely to be an noun

  P(NN/fish)=0.91 P(VB/fish)=0.09

- Now consider the following two sentences and their initial tags:

  I/PRP like/VB to/TO eat/VB fish/NNP.

  I/PRP like/VB to/TO fish/NNP

- Most likely tag *fish* is NNP,tagger assigns this tag to the word in both sentence.
- But in second case it is mistake

Part-of-Speech Tagging

Hybrid Taggers -Example:

- After initial tagging ,*transformation rules* are applied ,learns rule and apply mis-tagging of *fish*:

  Change NNP to VB if the previous tag is TO

- As the contextual condition is satisfied, this rule will change fish/NN to fish /VB:

  like/VB to/TO fish/NN → like/VB to/TO fish/VB

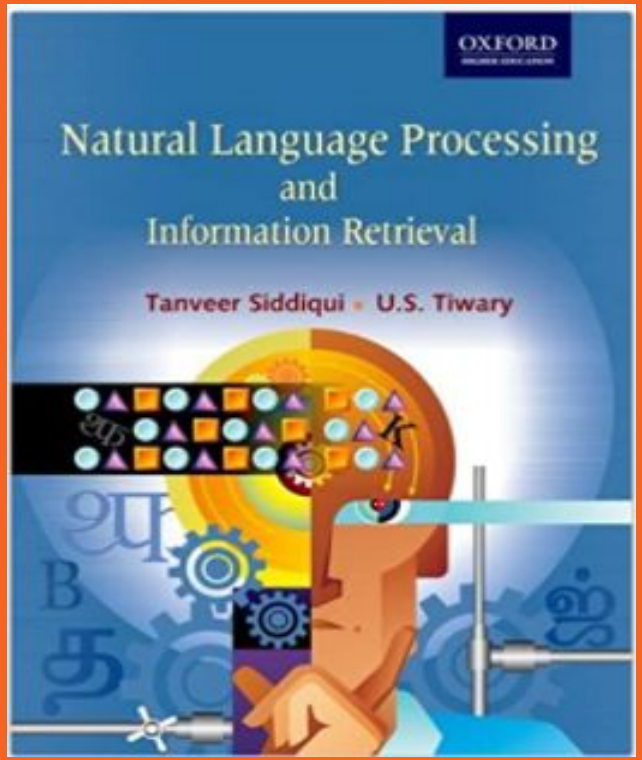
Question Bank- Chapter 3

1. Explain the following: i) Regular Expression ii) Finite-state Automata.-10M
2. What are regular Expressions? Write regular expression for an email address -5M
3. Explain DFA and NFA with examples.-10M
4. Explain the working of two level morphological parser model. Write a simple Finite State Transducers(FST) for mapping english nouns-8M
5. What is morphological parsing and explain two step morphological parser with example-8M
6. Describe the information sources used by the morphological parser-5M
7. How to perform parts of speech tagging and Morphological parsing -10M
8. How to detect spelling error and how to correct it? -10M
9. Describe 2 categories of spelling errors.-04M
10. How to detect spelling errors and correct it-10M
11. Define Minimum edit distance..Explain how value in each cell is completed in terms of three possible paths with respect to the minimum edit distance algorithm-6M
12. Explain Minimum edit distance algorithm and compute the minimum edit distance between **tumour** and **tutor** -8M
13. Write and explain an algorithm for Minimum edit distance spelling correction. Apply the same to find minimum edit distance between words **PEACEFUL** and **PAECFLU** -8M or INTENTION and EXECUTION- 8M
14. What is parts-of speech tagging and explain different methods of parts-of-speech-tagging-8M
15. Explain the different categories involved in parts-of speech tagging-10M
16. Explain the charecter classes,spelling error detection and correction concept by solving minimum edit distance problem.

Pattern: ABCD to AEDCB

Chapter-4

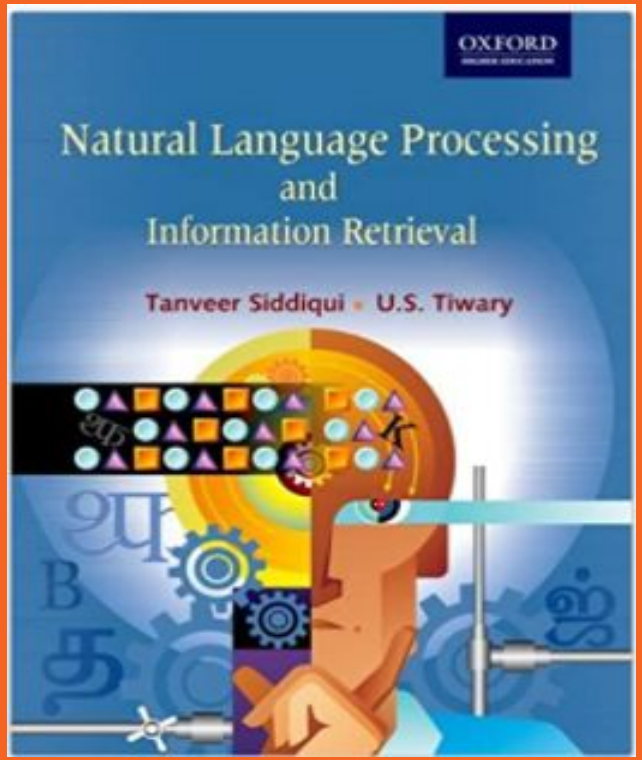
Syntactic Analysis



Chapter 4

Syntactic Analysis

- Introduction
- Context-Free Grammar
- Constituency
- Parsing
  - Top-down Parsing
  - Bottom-up Parsing
  - A-basic Top-down Parser
  - Early Parser



Syntactic Analysis

- **Introduction**
- Word *"syntax"* refers to the *grammatical arrangements of words* in a sentence and *relationships* with each other.
- **Objective:** To find syntactic structure of sentence.
- Structure represented as *tree*.
- *Nodes* in the tree represented as *phrases* and *leaves* as *words*
- *Root* of the tree is a *whole sentence*
- Identification of syntactic structure is done by *parsing*
- Syntactic analysis also considered as 'phrase markers' to a sentence

Context- Free Grammar

- **Context Free Grammar (CFG)** first defined for natural language by Chomsky(1957).
- Used for algol programming language by Backus(1959) and Naur(1960)
- CFG also called as phrase-structure grammar.
- Consists of 4 components:
  1. A set of nonterminal symbols, N
  2. A set of terminal symbols, T
  3. A designated start symbol, S, one of the symbol from N
  4. A set of productions, P of the form: $A \rightarrow \alpha$

Context- Free Grammar

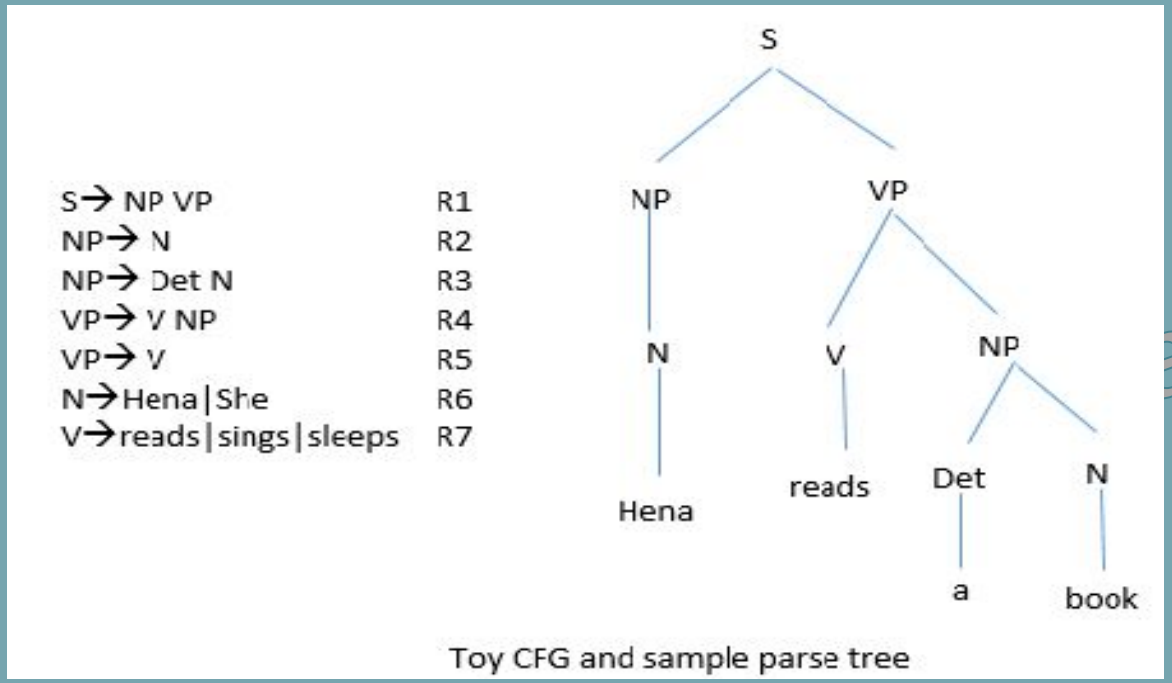
- $A \rightarrow \alpha$, where $A \in N$ and α is a string consisting of terminal and non-terminal symbols.
- A can be rewritten as α
- Also called as *phrase structure rule*.
- It specifies *which elements*(constituents) can occur in a phrase and in *what order*.
- **Example:**

  S →NP VP , states that S consists of NP followed by VP, i.e., a sentence consist of a *noun phrase* followed by a *verb phrase*

Context- Free Grammar

- A language is defined through the concept of *derivation*
- Basic operation is of *rewriting* the symbol appearing on the left hand side of production by its right hand side.
- Which can be represented using *parse tree*
- Parse tree represents *mapping* of a string to its *parse tree*
- **Example:**Consider the *toy grammar sample parse tree*

Context- Free Grammar



Toy CFG and sample parse tree

Example:

- Here, Symbol S *rewritten* as NP VP Using Rule 1(R1)
- R2 and R4 *rewrites* NP,VP a N and V.
- NP *rewritten* as Det N in R3
- Finally using R6 and R7,

- We get the sentence as: *Hena reads a book ------(1)*

Context- Free Grammar

- We can also represent *compact bracketed* notation to represent a *parse tree*.
- The parse tree in a figure above can be represented using following notation:

[S [NP [N Hena]] [VP[V reads] [NP [Det a] [N book]]]]

Constituency

- *Words* group together to form *larger constituents* (phrases) and eventually a *sentence*.
- **Example:** The bird, The rain, The Wimbledon court,The beautiful garden → Noun phrases
- These *constituents* combine with others to form *sentence constituent.*
- **Example:** The bird, can combine with verb phrase, flies, to form sentence "The bird flies"

Constituency

Phrase Level Constructions:

- In natural language *certain group of words* behave as *constituents.*
- Constituents decide whether a *group of words* is a *phrase*,if it can be *substituted* with *some other group of words* without changing the meaning
- If *substitution is possible* then the set of words forms a *phrase*.
- This is called the *substitution test*

Constituency

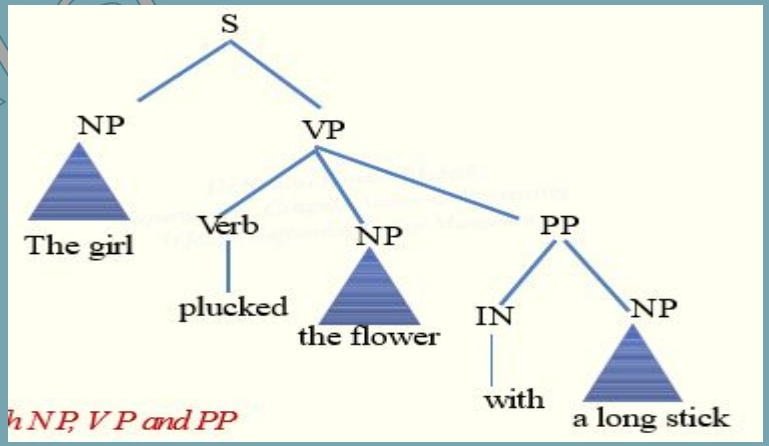
Phrase Level Constructions:

- **Example:** We can *substitute* number of phrases like:
  - Hena reads a book.
  - Hena reads a storybook.
  - Those girls read a book
  - She reads a comic book
- Constituents are: Hena, She and Those girls and a book ,a storybook and a comic book. These forms *phrase*.

Constituency

Phrase Level Constructions:

- Phrase types are defined after their *head*,which is lexical category that determines properties of the phrase
- If the head is *noun*,phrase is noun phrase.If head is *verb* then phrase is verb phrase
- **Example:**A sentence with NP,VP,PP



h NP, V P and PP

Constituency

Noun Phrase:

- Phrase whose *head* is a *noun* or *pronoun*
- Modifiers of a noun phrase can be *determiners or adjective phrases*
- These structures can be represented using *phrase structure rule* as below:

NP→Pronoun
NP→Det Noun
NP→Noun
NP→Adj Noun
NP→Det Adj Noun
We can combine all these in a single phrase structure rule as:
NP→(Det)(Adj) Noun|Pronoun

- Here, ( ) represents optional
- That is, Noun possibly preceded by determiner and an adjective.

Constituency

Noun Phrase:

- Noun phrase may include *post modifiers* more than one adjective.
- It may include *Prepositional Phrase(PP)*.
- After incorporating rule will be as below:

NP→ (Det) (AP) Noun (PP)

- **Examples:Noun phrases:**

		Example consists of:
They	(4.2a)	→Pronoun
The foggy morning	(4.2b)	→Det Adj Noun
Chilled water	(4.2c)	→AP Noun
A beautiful lake in Kashmir	(4.2d)	→Det Adj Noun PP
Cold banana shake	(4.2e)	→Adj and sequence of Noun

Constituency

Noun Phrase:

- *A Noun sequence* is termed as : *nominal*
- To *handle* the *nominal* we can write phrase structure rule as below:

  NP→ (Det) (AP) Nom (PP)

  Nom→ Noun|Noun Noun

- Noun phrase can act as *subject, an object or predicate.*
- **Examples:**

The foggy damped weather disturbed the match	(4.3a)	→NP acts as subject
I would like a nice cold banana shake	(4.3b)	→NP acts as object
Kula botanical garden is a beautiful location	(4.3c)	→NP acts as predicate

Constituency

Verb Phrase:

- Headed by *verb*
- Wide range of phrases can *modify a verb* → complex
- Organizes the various elements of the sentence depends on the syntactic structure.
- **Examples:**

Khushubu *slept*	(4.4a)	VP →Verb
The boy *kicked* the *ball*	(4.4b)	VP →Verb Noun
Khushubu *slept in the garden*	(4.4c)	VP →Verb PP
The boy *gave the girl a book*	(4.4d)	VP →Verb NP NP , two NP's
The boy *gave the girl a book with blue cover*	(4.4e)	VP →Verb NP NP PP

Constituency

Verb Phrase:

- In general, *number of NP's limited* to two but it is *possible to add more than two PPs*
- Rule will be as follows:

  VP→ Verb (NP) (NP) (PP)*

- Here, *objects* may also be *entire clauses* in the sentence like:

  I know that Taj is one of the seven wonders.

- So, alternative phrase statement rule, which NP is replaced by S as below:

  VP→ Verb S

Constituency

Prepositional Phrase:

- Prepositional Phrases (PP) are headed by a *preposition*.
- They consist of a preposition,possibly *followed* by some other constituent, a *noun phrase*.
- **Example:**

  We played volleyball *on the beach*

- Preposition phrase that consists *just a preposition*
- **Example:**

  John went *outside*

- Phrase structure rule that captures the above eventualities as follows:

  PP→ Prep (NP)

Constituency

Adjective Phrase:

- Adjective Phrases (AP) are headed by a *adjectives*.
- They consist of a adjectives,may be *preceded* by an *adverb* and followed by a *PP*.
- **Examples:**

  Ashish is *clever.*

  The train is very *late.*

  My sister is *fond of animals.*

- Phrase structure rule as follows:

  AP→ (Adv) Adj (PP)

Constituency

Adverb Phrase:

- Adverb Phrases (AdvP) are consists of *adverb*.
- *preceded* by a degree *adverb* .
- **Example:**

  Time passes *very quickly*

- Phrase structure rule as follows:

  AdvP→ (Intens) Adv

Note: intens-intensity

Constituency

Sentence level Constructions:

- A sentence can have *varying structures*
- 4 commonly known structures are:
  - Declarative structure
  - Imperative structure
  - Yes-no question structure
  - Wh-question structure
- *Declarative sentence- subject* followed by a *predicate* *(verb gives info about subject)*
- Where, subject is Noun Phrase and predicate is Verb Phrase

  Example: I like horse riding
- Phrase structure rule for declarative sentence as follows:

  S→ NP VP

Constituency

Sentence level Constructions: Imperative sentence

- *Imperative sentence- begin* with *verb phrase* and *lack subject*
- Subject is implicit in sentence and understood to be 'you'.
- These sentences are used for *commands* and *suggestions*
- Phrase structure rule for imperative sentence as follows:

  S→ VP

- **Examples:**

  Look *at the door*

  Give me the book

  Stop talking

  Show me the latest design

Constituency

Sentence level Constructions: Yes-no question structure

- Sentences with *yes-no question* structure ,ask questions which can be answered using yes or no.
- These sentences *begin* with an *auxiliary verb*, followed by a *subject NP*, followed by *VP*
- Phrase structure rule for yes-no sentence as follows:

  S→Aux NP VP
- **Examples:**

  Do you have a red pen?
  Is there a vacant quarter?
  Is the game over?
  Can you show me your album?

Constituency

Sentence level Constructions: wh- question structure

- Sentences with *wh- question* structure are complex.
- Begin with a wh-words - who, which, where, what, why and how
- It may have wh-phrase as a subject or may include another subject
- Rule for Wh-sentence as follows:

  S→Wh-NP VP

- **Example:**

  Which team won the match?

Constituency

Sentence level Constructions: wh- question structure

- Another type of *wh- question* involves *more than one NP.*
- Here, Auxiliary verb comes before the subject NP
- Rule for Wh-questions as follows:

  S→Wh-NP Aux NP VP

- **Example:**

  Which cameras can you show me in your shop?

Constituency

Sentence level Constructions: Summary of grammar rules

S→ NP VP

S→ VP

S→Aux NP VP

S→ Wh-NP VP

S→ Wh-NP Aux NP VP

NP → (Det) (AP) Nom (PP)

VP → Verb (NP) (NP) (PP)*

VP → Verb S

AP→ (Adv) Adj (PP)

PP→ Prep (NP)

Nom → Noun | Noun Noun

Constituency

Sentence level Constructions: *coordination*

- Conjoining phrases with *conjunctions* like ‘and’ , ‘or’ , ‘but’.
- A coordinate *noun phrase* can consist of *two other noun phrases*
- **Examples:**

  I ate [NP[NP an apple] and [NP a banana]]
- VP can be conjoined as below:

  It is [VP [VP dazzling] and [VP a raining]]
- Sentence can be conjoined as below:

  [S[S I am reading the book] and [S I am also watching the movie]]
- Rule for Wh-questions as follows:

  NP → NP and NP VP→ VP and VP S→ S and S

Constituency

Sentence level Constructions: *Agreement*

- Most *verbs* use *2 different forms* in *present tense*
- Third person singular subjects and other kind of subjects.
- Third person singular subjects (3Sg) form ends with -s.
- *Non-3sg* does not end with -s
- Whenever there is verb that has noun acting as subject,This agreement is confirmed
- **Examples:**

  Does [NP Priya] sing?
- Here, subject NP is singular, so -es form ‘do’

  Do [NP they] eat?
- Subject NP is plural.Hence ‘do’ form is used.

Constituency

Sentence level Constructions: *Agreement*

- Rules used to handle yes-no questions :

  S→ Aux NP VP

- To take care of subject-verb agreement, we replace the above rule as follows:

  S→ 3sgAux 3sgNP VP

  S→ Non3sgAux Non3sgNP VP

- *Lexicon* can be like below:

  3sgAux → does| has| can

  Non3sgAux → do| have| can

Constituency

Sentence level Constructions: *Feature Structures*

- Sets of *feature-value* pairs
- Used to efficiently capture the *properties of grammatical* categories.
- **Example:** Number property of a noun phrase can be represented by **NUMBER** feature
- The value of NUMBER featurencan take *SG(singular) and PL(plural)*
- *Values* can be *atomic symbols or feature structures*.
- Represented by matrix like diagram called *Attribute Value Matrix(AVM)*

FEATURE1	**VALUE1**
FEATURE2	**VALUE2**
...	
FEATUREn	**VALUEn**

Constituency

Sentence level Constructions: *Feature Structures*

- AVM consists *single NUMBER* feature with value *SG* is represented as below:

  [NUMBER SG]

- *Value* of feature can be left *unspecified* and represented as below:

  [NUMBER [ ]]

- Feature structure can be used to *encode the grammatical category* of a constituent and *features associated* to it.

Constituency

Sentence level Constructions: *Feature Structures*

- **Example:** *Third person singular noun phrase* can be represented as below:

CAT	NP
NUMBER	SG
PERSON	3

- **Example:** *Third person plural noun phrase* can be represented as below:

CAT	NP
NUMBER	PL
PERSON	3

- Here, *Value of feature* **CAT** and **PERSON** remain same in both the structures.

Constituency

Sentence level Constructions: *Feature Structures*

- Feature values *not only atomic* but it can have *another feature structure*
- **Example:** Consider the case of combining the NUMBER and PERSON features into a single AGREEMENT feature.
- In grammatical sense, *grammatical subjects* must agree with their predicates in NUMBER and PERSON properties.
- Using this new feature,*grammatical category 3-PL NP* by following structure:

CAT	NP	
AGREEMENT	NUMBER	PL
	PERSON	3

Constituency

Sentence level Constructions: *Feature Structures*

- Using Feature structures can perform *operations.*
- Operations are:
  - Merging the information content of 2 structures
  - Rejecting the structures that are incompatible.
- These computational techniques are called as ***unification***.
- Unification implemented using *binary operator -* □
- **Advantages:** *CFG rules* can have feature structures to *realize on the constraints of the sentence.*

Constituency

Sentence level Constructions: *Feature Structures*

- **Example:** Performs an equality check

  [NUMBER PL] □ [NUMBER PL] = [NUMBER PL]

  Success, two structures have the same value

- **Example:** Performs a result as non-null values when unspecified structure is given

  [NUMBER PL] □ [NUMBER [ ]] = [NUMBER PL]

  Two structures are compatible and merged into structures PL

- **Example:** result fails as non-null values

  [NUMBER PL] □ [NUMBER SG] = *Fails*

  Two structures are in-compatible

Parsing

- CFG defines syntax of language but *does not define* how structures are assigned.
- Use *rewrite rules* of a grammar to:
  - generate a particular *sequence of words* or
  - *reconstruct its derivation* (Phrase structure tree)
  - Syntactic parser recognizes a sentence and assigns *syntactic structure* to it.
- **Phenomena associated with Syntactic Parsing:**
  - Syntactic ambiguity
  - Garden pathing

Parsing

- **Syntactic ambiguity**
- A *sentence* can have *multiple parses*.
- Many *different phrase structure trees* deriving the *same sequence of words.*
- **Garden pathing**
- Process of *constructing a parse* by exploring the parse tree along different paths, one after the other till, eventually, the right one is found.
- Eg: *The horse ran past the barn fell*

Parsing

- In Eg: *The horse ran past the barn fell*
- In first attempt, come up with the *parse* corresponding to the *sentence The horse ran past the barn ,*leaving no possibility for the word *fell* to be added incrementally in the sentence .
- Finding the *right parse* -> Search process
- Search finds all trees :
  - whose root is the start symbol, S and
  - whose leaves cover exactly the word in the input.

Parsing

- **Constraints that guide the Search Process**

1. **Input:**

- First constraint comes from *words in* the input *sentence.*
- *Valid parse* is one that covers all the words in a sentence.
- *Words* must constitute the *leaves* of the final parse tree.

2. **Grammar**

- Second constraint comes from the *grammar.*
- *Root* of the final parse tree must be the *start symbol* of the grammar.

Parsing

Constraints that guide the Search Process:

Constraints give rise to

Two Search strategies widely used :

- Top-down (Goal-directed search)
- Bottom-up (Data-directed search)

Top-down Parsing

- Start the *search* from the *root node S* and work *downwards* towards the leaves

Assumption

- *Input* can be derived from the *designated start symbol, S*, of the grammar.

Top-down Parsing

- Start the *search from the root node S* and work *downwards* towards the leaves.
- Find *all sub trees* which can start with S.
- *Expand* the root node using all the *grammar rules with S(Eg:* S -> NP VP) on their LHS - Subtrees of the second-level search.
- Similarly, *expand each non-terminal symbol* in the resulting sub-trees using the grammar rules having matching non-terminal symbol on their LHS.
- RHS of the grammar rules provides the nodes to be generated, which are then expanded recursively.

Top-down Parsing

- The *tree grows downward* and eventually reaches a state where the *bottom of the tree* consists only of POS categories.

- All trees whose *leaves do not match words* in the *input sentence* are **rejected**, leaving trees representing successful parse.

- A tree which *matches* exactly with the words in the *input sentence* – **Successful Parse.**

Top-down Parsing

- **Example:** Consider the Sentence : Paint the door and construct top down parsing.
- Consider the following Grammar as below:

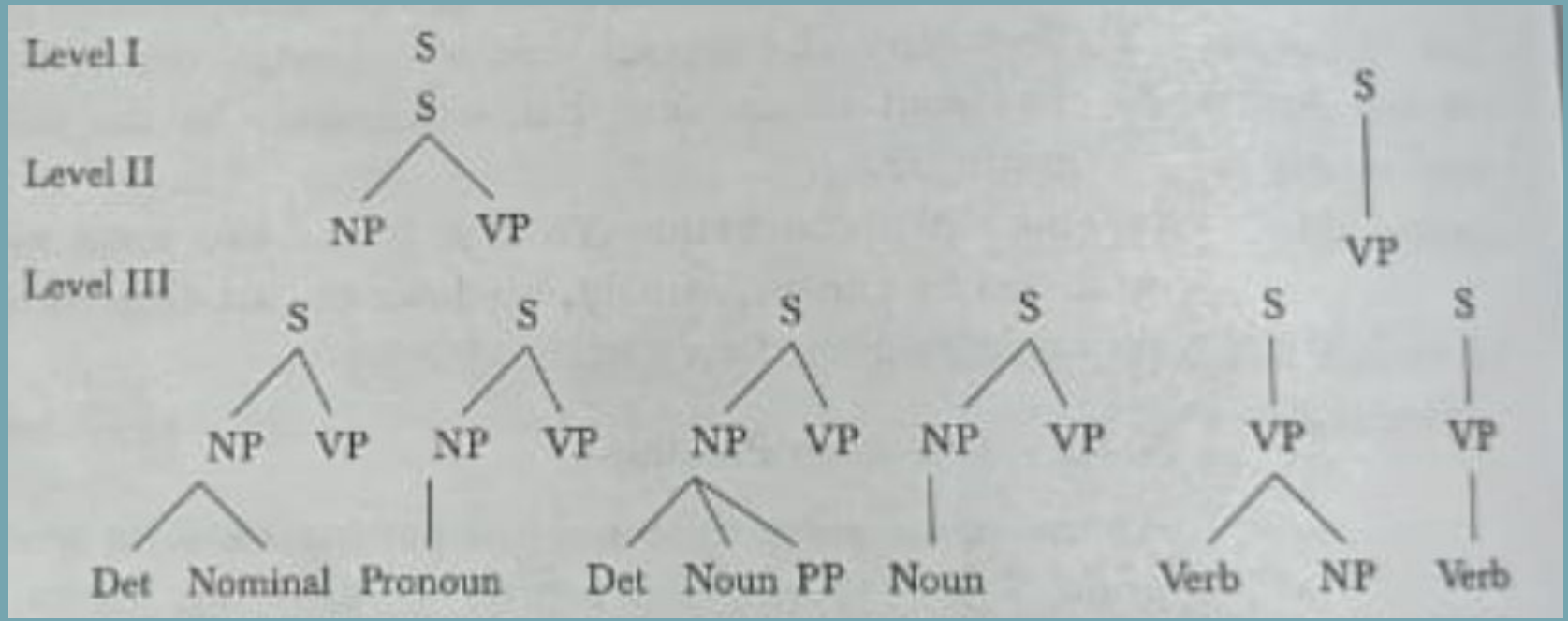
S -> NP VP
S -> VP
NP -> Det Nominal
NP -> Pronoun
NP -> Det Noun PP
NP -> Noun

Nominal -> Noun
Nominal -> Noun Nominal
Noun -> paint | door
VP -> Verb NP
VP -> Verb
PP -> Preposition NP

Det -> this | that | a | the
Verb -> sleeps | sings | open | saw | paint
Preposition -> from | with | on | to
Pronoun -> she | he | they

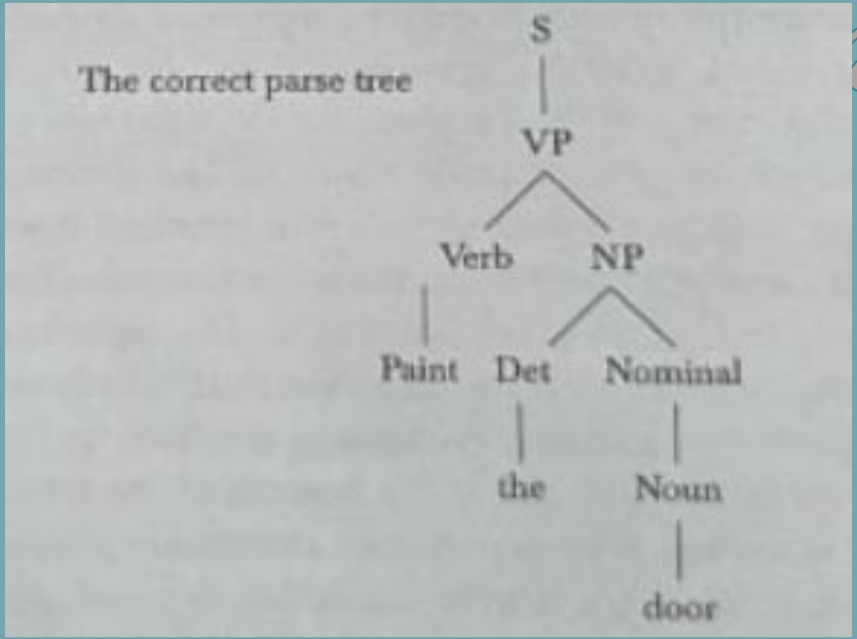
Top-down Parsing

- **Example:** Paint the door



Top-down Parsing

- **Example:** Paint the door
- If we expand *Level III* of 5th parse tree result will be:



Bottom -up Parsing

- Starts with the *words* in the *input sentence.*
- Attempts to construct a *parse tree* in an *upward direction towards the root.*
- Look for *rules in the grammar* where the *RHS matches* some of the *portions in the parse tree* constructed so far, *reduces it* using the *LHS of the production.*
- *Parser reduces* the tree to start symbol → **Successful parse.**

Bottom -up Parsing

- **Example:** Consider the Sentence : Paint the door and construct bottom up parsing.
- Consider the following Grammar as below:

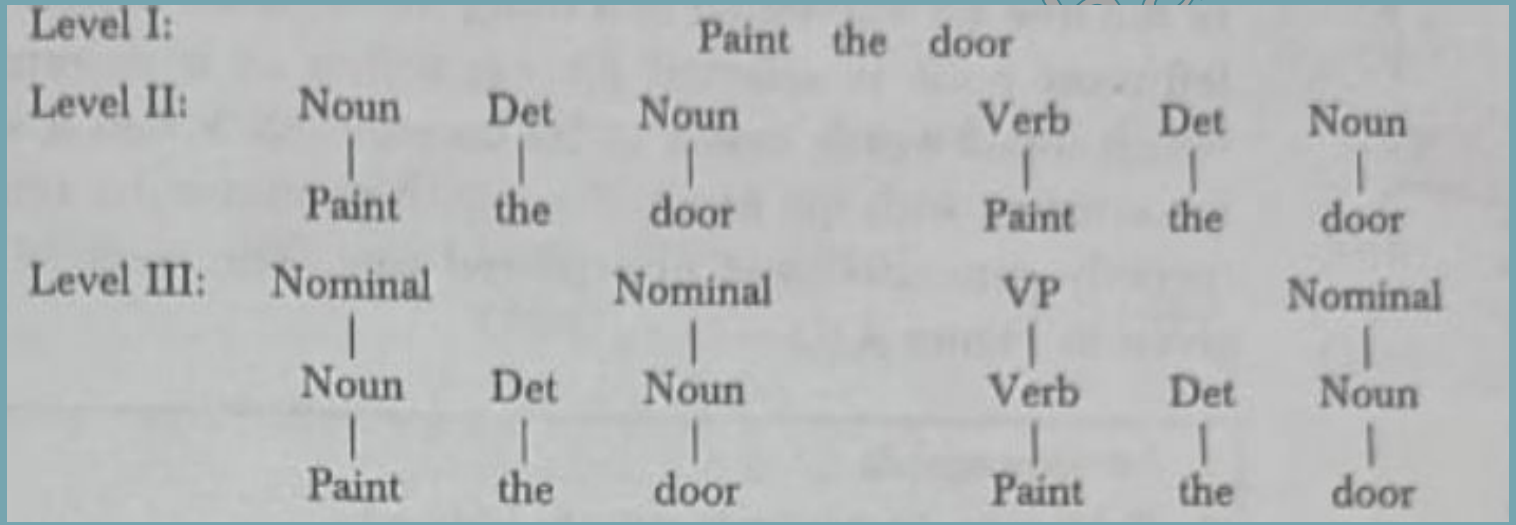
S -> NP VP
S -> VP
NP -> Det Nominal
NP -> Noun
NP -> Pronoun
NP -> Det Noun PP

Nominal -> Noun
Nominal -> Noun Nominal
Noun -> paint | door
VP -> Verb NP
VP -> Verb
PP -> Preposition NP

Det -> this | that | a | the
Verb -> sleeps | sings | open | saw | paint
Preposition -> from | with | on | to
Pronoun -> she | he | they

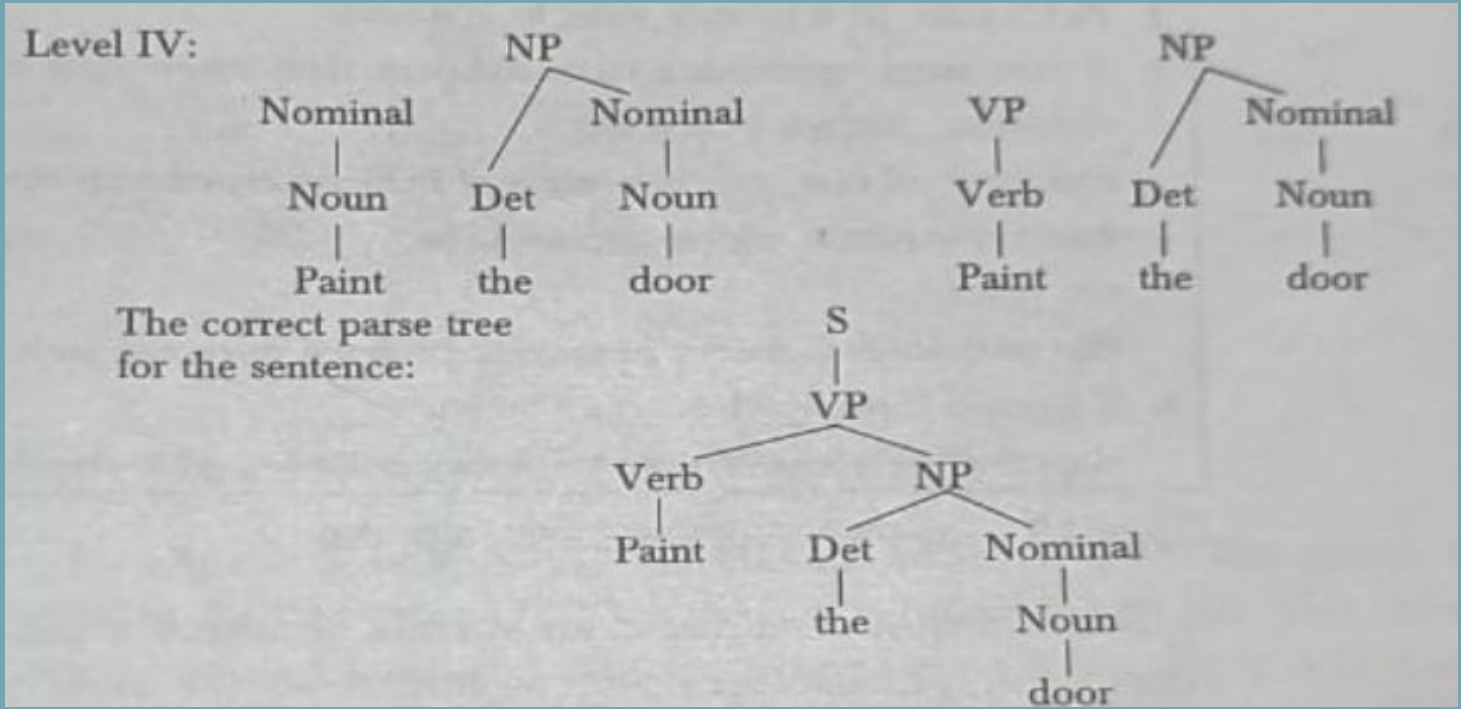
Bottom -up Parsing

- **Example:** Paint the door



Bottom -up Parsing

- **Example:** Paint the door



Top down vs Bottom -up Parsing

Top-down

- As top down *search starts* generating trees with the *start symbol* of the grammar, *Never wastes time* exploring a tree leading to a different root.
- *Wastes considerable time exploring S trees* that eventually result in *words* that are *inconsistent* with the input.
- Top-down parser *generates trees before seeing the input*.

Top down vs Bottom -up Parsing

Bottom-up

- *Never explores a tree* that *does not match the input*.
- *Wastes considerable time* generating trees that have *no chance of leading to an S-rooted tree*.

A Basic Top-down Parser

Approach: *Depth-first, left to right search*

- Depth-first approach expands the *search space incrementally* by *one state at a time.*
- At each step, *left-most unexpanded leaf* of the tree are *expanded first* using the *relevant rule of the grammar*.
- When a state arrives that is *inconsistent* with the *input*, the search continues by *returning* to the *most recently generated* and *unexplored tree*

A Basic Top-down Parser

Approach: Top-down, *Depth-first parsing algorithm:*

- Steps of the algorithm are given below:

1. Initialize agenda
2. Pick a state, let it be curr_state, from agenda
3. If (curr_state) represents a successful parse then return parse tree
   else if curr_stat is a POS then
   if category of curr_state is a subset of POS associated with curr_word
   then apply lexical rules to current state
   else reject
   else generate new states by applying grammar rules and push them into agenda
4. If (agenda is empty) then return failure
   else select a node from agenda for expansion and go to step 3.

A Basic Top-down Parser

Approach: Top-down,*Depth-first parsing algorithm:*

- The algorithm maintains *agenda* of *search states (S)*.
- Each *search states* consist of *partial trees* and a *pointer* to the *next input word* in the sentence.
- Algorithm *starts* with the state at the *front of agenda* and *generates* a set of *new states* by applying *grammar rule* to the *left-most unexpected node* of the tree associated with it.

A Basic Top-down Parser

Approach: Top-down,*Depth-first parsing algorithm:*

- The *newly generated states* are put on the *front of the agenda* in the order defined by the textual order of the grammar rules used to create them.
- Process *continues* until either a *successful parse tree* is discovered or the *agenda is empty*, indicating a *failure*.

A Basic Top-down Parser

- **Example:** Consider the Sentence : Open the door and derivation using top-down,depth first algorithm.
- Consider the following Grammar as below:

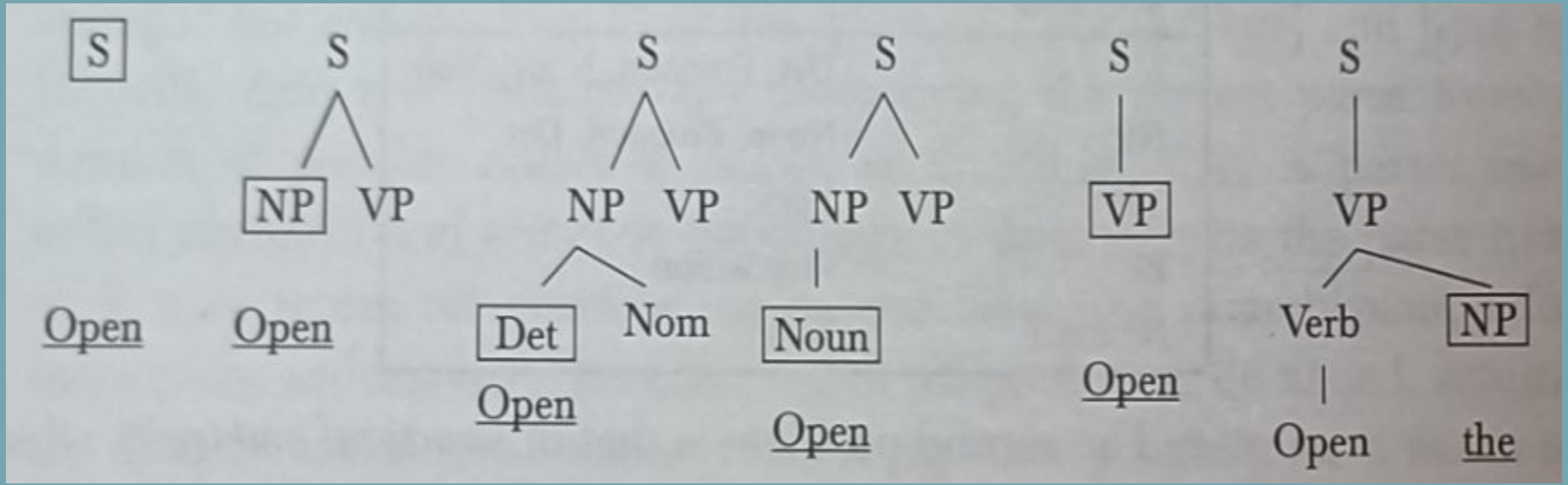
S -> NP VP
S -> VP
NP -> Det Nominal
NP -> Noun

Nominal -> Noun
Nominal -> Noun Nominal
Noun -> paint | door
VP -> Verb NP
VP -> Verb
PP -> Preposition NP

Det -> this | that | a | the
Verb -> sleeps | sings | open | saw | paint
Preposition -> from | with | on | to
Pronoun -> she | he | they

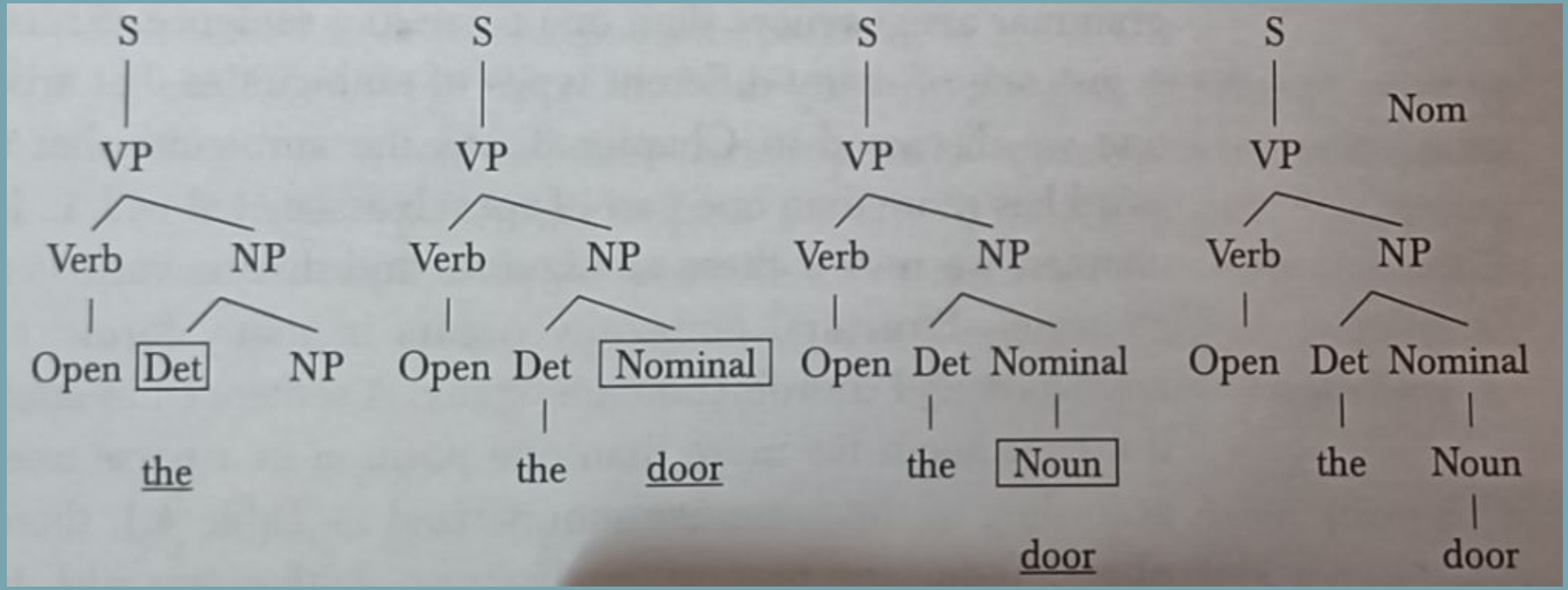
A Basic Top-down Parser

- **Example:** Open the door



A Basic Top-down Parser

- **Example:** Open the door



A Basic Top-down Parser

- **Example:** Open the door
- *Trace of algorithm* on the above sentence, starts with *node S* and input word Open.
- First expands S → NP VP.
- *Expands* unexpanded non terminal NP using the rule

  NP → Det Nominal.
- But the word Open *cannot be derived* from Det.
- Hence *parser eliminates* the *rule*.

A Basic Top-down Parser

- **Example:** Open the door
- *Tries* the second alternative
  NP → Noun
- Which leads to failure.
- *Next search space* on the agenda is
  S→ VP rule
- The expansion of VP using the rule *VP → Verb NP*
- *Successfully matches the input word.*
- Algorithm *proceeds* in a *depth-first, left-to right manner,* to match the rest of the *input words.*

A Basic Top-down Parser

- In a *successful parse*, current *input word must match the first word* in the *derivation of the node* that is being expanded.
- This *information* is utilized in *eliminating spurious parses*.
- Grammar rule that *cannot lead to the input word as the first word* along the left side of derivation, *shouldn't be considered for expansion*.

A Basic Top-down Parser

- The first word along the left side of the derivation is called the *left corner* of the tree.

- S → VP is the only rule that is applicable, as the word '*Open*' cannot be the left corner of the NP.

A Basic Top-down Parser

To utilize this filter:

- *Create a table* containing a list of all *valid left corner categories* for each *non-terminal of the grammar*.
- While *selecting a rule for expansion*, the *table is consulted* to see if the non-terminal associated with the rule has a *POS associated* with the *current input. If not, the rule is not considered*.

A Basic Top-down Parser

- *Left corner table* for grammar is as shown below:

Category	Left Corners
S	Det, Pronoun, Noun, Verb
NP	Noun, Pronoun, Det
VP	Verb
PP	Preposition
Nominal	Noun

A Basic Top-down Parser

Disadvantages

1.Left recursion

- Causes *search to get stuck* in an infinite loop.
- Problem arises if grammar is left recursive.
- **Example:** A-> Aβ, for some β

2.Structural Ambiguity

- Occurs when a grammar assigns *more than one parse to a sentence.*
- Occurs in many forms:
  - *Attachment Ambiguity*
  - *Co-ordination Ambiguity*

A Basic Top-down Parser

Disadvantages

Structural Ambiguity

i)Attachment Ambiguity

- If a constituent fits more than one position in a parse tree.

  Example: Generating prepositional phrase ***'with a long stick'*** in the sentence:

  'The girl plucked the flower with a long stick'

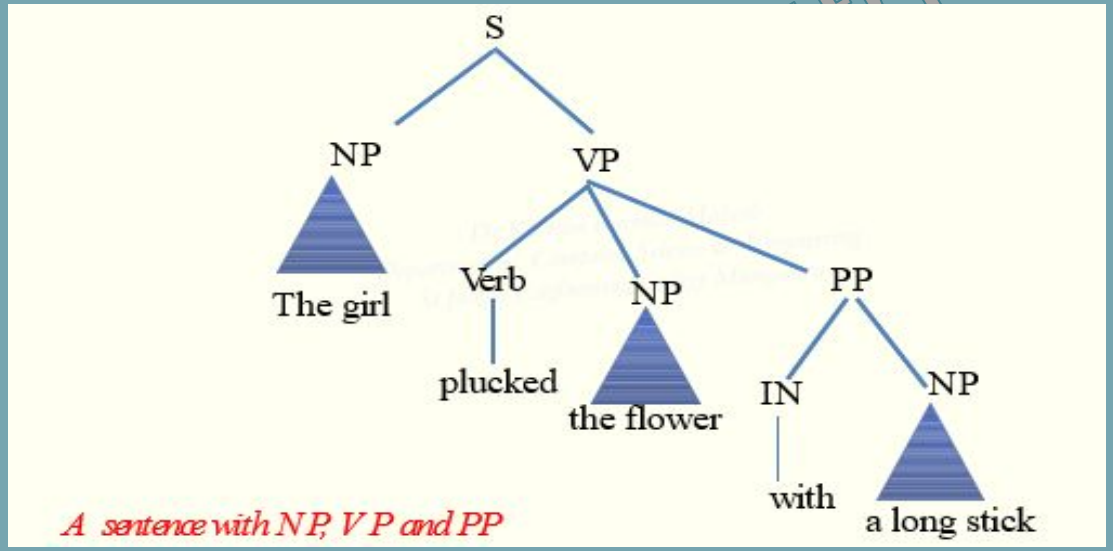
- Can be generated *from the Verb Phrase* as in the parse tree……

A Basic Top-down Parser

Disadvantages

i)Attachment Ambiguity

Example: *'The girl plucked the flower with a long stick'*



A sentence with NP, VP and PP

A Basic Top-down Parser

Disadvantages

i)Attachment Ambiguity

- If a constituent fits more than one position in a parse tree.

  Example: Generating prepositional phrase *'with a long stick'* in the sentence:

  'The girl plucked the flower with a long stick'

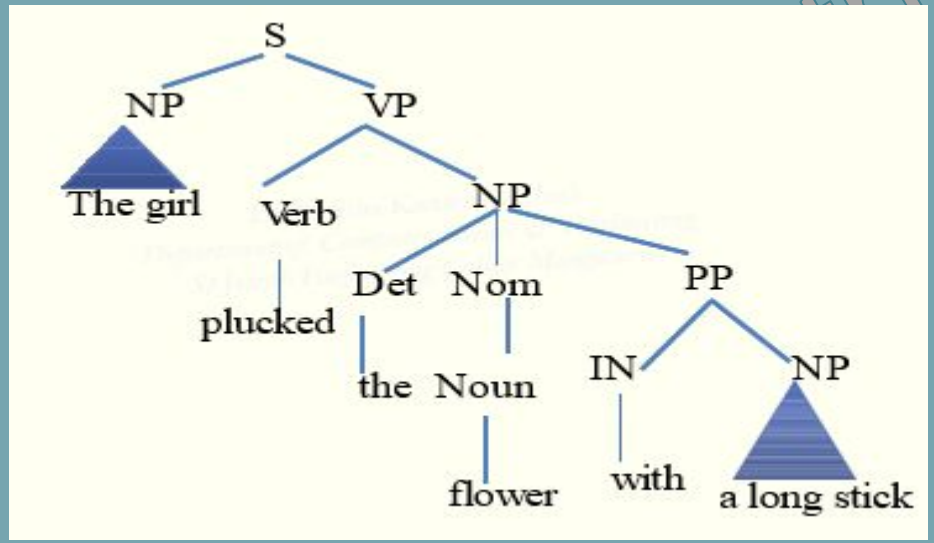
- Can be generated *Noun Phrase* as in the parse tree……

A Basic Top-down Parser

Disadvantages

i)Attachment Ambiguity

Example: *'The girl plucked the flower with a long stick'*



A Basic Top-down Parser

Disadvantages

ii)Coordination Ambiguity

- Unclear which phrases are being combined with a conjunction like ‘and’.
- **Example:**

Beautiful hair *and* eyes

[Beautiful hair] *and* [eyes]

[Beautiful hair] *and* [beautiful eyes]

A Basic Top-down Parser

Disadvantages

iii)Local Ambiguity

- Occurs when *parts of a sentence* are ambiguous.

Example:

- Paint the door is unambiguous.
- But, during parsing, it is not known whether the first word 'Paint' is *Verb or a Noun*.
- Parser makes a few incorrect expansions before discovering that 'Paint' is a verb.

A Basic Top-down Parser

Disadvantages

3.Repeated Parsing

- Problem with respect to *top-down parsing*.
- Often builds valid trees for portions of input that it *discards* during *backtracking.*
- Have to be *rebuilt during subsequent* steps in the parse
- Solution: Use Dynamic programming algorithms.

Earley Parser

- Parallel *Top-down* search.
- Builds a *table of subtrees* for each of the constituents in the input.
- *Repetitive parse* of a constituent arising from backtracking is *eliminated.*
- Reduces the exponential-time problem to polynomial time.
- Can handle recursive rules such as A→AC *without* getting into an *infinite loop.*

Components - Earley Chart

- *n+1* entries, *n* is the number of words in the input.
- Contains a set of states for each word position in the sentence.

Earley Parser

Earley parser

- Algorithm makes a *left to right scan* of input to fill the elements in the chart.
- Builds a *set of states*, one *for each position* in the input string (starting from 0).
- States describe the *condition of the recognition process* at that point of the scan.

Earley Parser

Earley parser

States in each entry provide the following information:

- A *sub-tree* corresponding to a grammar rule.
- Information about the *progress made* in completing the sub-tree.
- *Position of the sub-tree* with respect to input.
- A state is represented as a ***dotted rule*** and *a pair of numbers* representing *starting position* and the *position of dot*.

Earley Parser

Earley parser- Earley Algorithm

- Operations used to process states in the chart:
  - Predictor
  - Scanner
  - Completer

- Algorithm sequentially constructs the sets for each of the *n+1* chart entries.
- Chart[0] is initialized with *dummy state*, $S^{l} \rightarrow .S, [0,0]$

Earley Parser

Earley parser- Earley Algorithm

- At each step *one* of the *three operations* are applicable depending on the state.
  - Application of the operators result in *addition of new states to* either the *current or the next set* of states.
- Presence of a state **S →∝. [0,N]** indicates a successful parse.

Earley Parser

Trace the Earley algorithm to show the sequence of states created in parsing the sentence:

Sana drinks coffee with milk → *Terminals*

S->NP VP	VP->VP PP	
NP->NP PP	VP->Verb NP	Noun->Sana
NP->Noun	Verb->drinks	Noun->milk
PP->Prep NP	Prep->with	Noun->coffee

Earley Parser -early chart

	0					
0						
	.	Sana	drinks	coffee	with	milk

Earley Parser

Earley Parser

Trace the Earley algorithm to show the sequence of states created in parsing the sentence:

Sana drinks coffee with milk → *Terminals*

S->NP VP	VP->VP PP	
NP->NP PP	VP->Verb NP	Noun->Sana
NP->Noun	Verb->drinks	Noun->milk
PP->Prep NP	Prep->with	Noun->coffee

Earley Parser

0

0 $S^1 \rightarrow .S$ $S \rightarrow .NP\ VP$					
	.Sana	drinks	coffee	with	milk

Earley Parser

Trace the Earley algorithm to show the sequence of states created in parsing the sentence:

Sana drinks coffee with milk

S->NP VP	VP->VP PP	
NP->NP PP	VP->Verb NP	Noun->Sana
NP->Noun	Verb->drinks	Noun->milk
PP->Prep NP	Prep->with	Noun->coffee

Earley Parser

	0					
0	$S^1 \rightarrow .S$ $S \rightarrow .NP\ VP$ $NP \rightarrow .NP\ PP$ $NP \rightarrow .Noun$					
		.Sana	drinks	coffee	with	milk

Earley Parser

	0					
0	$S^1 \rightarrow .S$ $S \rightarrow .NP\ VP$ $NP \rightarrow .NP\ PP$ **$NP \rightarrow .Noun$**					
		.Sana	drinks	coffee	with	milk

Earley Parser

0						
0	$S^1 \rightarrow .S$ $S \rightarrow .NP\ VP$ $NP \rightarrow .NP\ PP$ **$NP \rightarrow .Noun$**					
		.Sana	drinks	coffee	with	milk

Earley Parser

Trace the Earley algorithm to show the sequence of states created in parsing the sentence:

Sana drinks coffee with milk

S->NP VP	VP->VP PP	
NP->NP PP	VP->Verb NP	Noun->Sana
NP->Noun	Verb->drinks	Noun->milk
PP->Prep NP	Prep->with	Noun->coffee

Earley Parser

0					
$S^1 \rightarrow .S$ $S \rightarrow .NP$ VP $NP \rightarrow .NP\ PP$ **NP→.Noun** Noun→.Sana					
	.Sana	drinks	coffee	with	milk

0

Earley Parser

	0	1				
0	$S^1 \rightarrow .S$ $S \rightarrow .NP\ VP$ $NP \rightarrow .NP\ PP$ **$NP \rightarrow .Noun$** $Noun \rightarrow .Sana$	Noun->Sana.				
		Sana.	drinks	coffee	with	milk

Earley Parser

	0	1				
1						
0	$S^1 \rightarrow .S$ $S \rightarrow .NP\ VP$ $NP \rightarrow .NP\ PP$ $NP \rightarrow .Noun$ $Noun \rightarrow .Sana$	$Noun \rightarrow Sana.$				
		Sana.	drinks	milk	with	coffee

Earley Parser

	0	1				
1						
0	$S^1 \rightarrow .S$ $S \rightarrow .NP\ VP$ $NP \rightarrow .NP\ PP$ $NP \rightarrow .Noun$ $Noun \rightarrow .Sana$	$Noun \rightarrow Sana.$ $NP \rightarrow Noun.$				
		Sana.	drinks	milk	with	coffee

Earley Parser

	0	1				
1						
0	$S^1 \rightarrow .S$ $S \rightarrow .NP\ VP$ $NP \rightarrow .NP\ PP$ $NP \rightarrow .Noun$ $Noun \rightarrow .Sana$	$Noun \rightarrow Sana.$ $NP \rightarrow Noun.$				
		Sana.	drinks	milk	with	coffee

Earley Parser

	0	1				
1						
0	$S^1 \rightarrow .S$ $S \rightarrow .NP\ VP$ $NP \rightarrow .NP\ PP$ $NP \rightarrow .Noun$ $Noun \rightarrow .Sana$	$Noun \rightarrow Sana.$ $NP \rightarrow Noun.$ $NP \rightarrow NP.PP$				
		Sana.	drinks	milk	with	coffee

Earley Parser

	0	1				
1						
0	$S^1 \rightarrow .S$ $S \rightarrow .NP\ VP$ $NP \rightarrow .NP\ PP$ $NP \rightarrow .Noun$ $Noun \rightarrow .Sana$	$Noun \rightarrow Sana.$ $NP \rightarrow Noun.$ $NP \rightarrow NP.PP$ $S \rightarrow NP.VP$				
		Sana.	drinks	milk	with	coffee

Earley Parser

Trace the Earley algorithm to show the sequence of states created in parsing the sentence:

Sana drinks coffee with milk

S->NP VP	VP->VP PP	
NP->NP PP	VP->Verb NP	Noun->Sana
NP->Noun	Verb->drinks	Noun->milk
PP->Prep NP	Prep->with	Noun->coffee

Earley Parser

	0	1				
1		PP→.Prep NP				
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana.	drinks	milk	with	coffee

Earley Parser

Trace the Earley algorithm to show the sequence of states created in parsing the sentence:

Sana drinks coffee with milk

S->NP VP	VP->VP PP	
NP->NP PP	VP->Verb NP	Noun->Sana
NP->Noun	Verb->drinks	Noun->milk
PP->Prep NP	Prep->with	Noun->coffee

Earley Parser

	0	1				
1		PP→.Prep NP VP→.Verb NP VP→.VP PP				
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana.	drinks	coffee	with	milk

Earley Parser

	0	1				
1		PP→.Prep NP VP→.Verb NP VP→.VP PP				
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana.	drinks	coffee	with	milk

Earley Parser

	0	1				
1		PP→.Prep NP VP→.Verb NP VP→.VP PP				
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana.	drinks	coffee	with	milk

Earley Parser

Trace the Earley algorithm to show the sequence of states created in parsing the sentence:

Sana drinks coffee with milk

S->NP VP	VP->VP PP	
NP->NP PP	VP->Verb NP	Noun->Sana
NP->Noun	Verb->drinks	Noun->milk
PP->Prep NP	Prep->with	Noun->coffee

Earley Parser

	0	1				
1		PP→.Prep NP **VP→.Verb NP** VP→.VP PP				
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana.	drinks	coffee	with	milk

Earley Parser

Trace the Earley algorithm to show the sequence of states created in parsing the sentence:

Sana drinks coffee with milk

S->NP VP	VP->VP PP	
NP->NP PP	VP->Verb NP	Noun->Sana
NP->Noun	Verb->drinks	Noun->milk
PP->Prep NP	Prep->with	Noun->coffee

Earley Parser

	0	1				
1		PP→.Prep NP **VP→.Verb NP** VP→.VP PP Verb→.drinks				
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana.	drinks	coffee	with	milk

Earley Parser

	0	1	2	3	4	5
1		PP→.Prep NP **VP→.Verb NP** VP→.VP PP Verb→.drinks	Verb→drinks.			
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana	drinks.	coffee	with	milk

Earley Parser

	0	1	2	3	4	5
1		PP→.Prep NP **VP→.Verb NP** VP→.VP PP Verb→.drinks	**Verb→drinks.**			
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	**Noun→Sana.** **NP→Noun.** NP→NP.PP S→NP.VP				
		Sana	drinks.	coffee	with	milk

Earley Parser

	0	1	2	3	4	5
1		PP→.Prep NP **VP→.Verb NP** VP→.VP PP Verb→.drinks	Verb→drinks. VP→Verb.NP			
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana	drinks.	coffee	with	milk

Earley Parser

Trace the Earley algorithm to show the sequence of states created in parsing the sentence:

Sana drinks coffee with milk

S->NP VP	VP->VP PP	
NP->NP PP	VP->Verb NP	Noun->Sana
NP->Noun	Verb->drinks	Noun->milk
PP->Prep NP	Prep->with	Noun->coffee

Earley Parser

	0	1	2	3	4	5
2			NP→.NP PP NP→.Noun			
1		PP→.Prep NP **VP→.Verb NP** VP→.VP PP Verb→.drinks	Verb→drinks. VP→Verb.NP			
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana	drinks.	coffee	with	milk

Earley Parser

	0	1	2	3	4	5
2			NP→.NP PP **NP→.Noun**			
1		PP→.Prep NP **VP→.Verb NP** VP→.VP PP Verb→.drinks	Verb→drinks. VP→Verb.NP			
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana	drinks.	coffee	with	milk

Earley Parser

	0	1	2	3	4	5
2			NP→.NP PP **NP→.Noun**			
1		PP→.Prep NP **VP→.Verb NP** VP→.VP PP Verb→.drinks	Verb→drinks. VP→Verb.NP			
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana	drinks.	coffee	with	milk

Earley Parser

Trace the Earley algorithm to show the sequence of states created in parsing the sentence:

Sana drinks coffee with milk

S->NP VP	VP->VP PP	
NP->NP PP	VP->Verb NP	Noun->Sana
NP->Noun	Verb->drinks	Noun->milk
PP->Prep NP	Prep->with	Noun->coffee

Earley Parser

	0	1	2	3	4	5
2			NP→.NP PP **NP→.Noun** Noun→.coffee			
1		PP→.Prep NP **VP→.Verb NP** VP→.VP PP Verb→.drinks	Verb→drinks. VP→Verb.NP			
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana	drinks.	coffee	with	milk

Earley Parser

	0	1	2	3	4	5
2			NP→.NP PP **NP→.Noun** Noun→.coffee	Noun→coffee.		
1		PP→.Prep NP **VP→.Verb NP** VP→.VP PP Verb→.drinks	Verb→drinks. VP→Verb.NP			
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana	drinks	coffee.	with	milk

Earley Parser

	0	1	2	3	4	5
2			NP→.NP PP **NP→.Noun** Noun→.coffee	Noun→coffee.		
1		PP→.Prep NP **VP→.Verb NP** VP→.VP PP Verb→.drinks	Verb→drinks. VP→Verb.NP			
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana	drinks	coffee.	with	milk

Earley Parser

	0	1	2	3	4	5
2			NP→.NP PP **NP→.Noun** Noun→.coffee	Noun→coffee. NP→Noun. NP→NP.PP		
1		PP→.Prep NP **VP→.Verb NP** VP→.VP PP Verb→.drinks	Verb→drinks. VP→Verb.NP			
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana	drinks	coffee.	with	milk

Earley Parser

Trace the Earley algorithm to show the sequence of states created in parsing the sentence:

Sana drinks coffee with milk

S->NP VP	VP->VP PP	
NP->NP PP	VP->Verb NP	Noun->Sana
NP->Noun	Verb->drinks	Noun->milk
PP->Prep NP	Prep->with	Noun->coffee

Earley Parser

	0	1	2	3	4	5
3				PP→.Prep NP		
2			NP→.NP PP **NP→.Noun** Noun→.coffee	Noun→coffee. NP→Noun. NP→NP.PP		
1		PP→.Prep NP **VP→.Verb NP** VP→.VP PP Verb→.drinks	Verb→drinks. VP→Verb.NP			
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana	drinks	coffee.	with	milk

Earley Parser

	0	1	2	3	4	5
3				PP→.Prep NP		
2			NP→.NP PP NP→.Noun Noun→.coffee	Noun→coffee. NP→Noun. NP→NP.PP		
1		PP→.Prep NP VP→.Verb NP VP→.VP PP Verb→.drinks	Verb→drinks. VP→Verb.NP			
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana	drinks	coffee.	with	milk

Earley Parser

	0	1	2	3	4	5
3				PP→.Prep NP		
2			NP→.NP PP NP→.Noun Noun→.coffee	Noun→coffee. NP→Noun. NP→NP.PP		
1		PP→.Prep NP VP→.Verb NP VP→.VP PP Verb→.drinks	Verb→drinks. VP→Verb.NP			
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana	drinks	coffee.	with	milk

Earley Parser

Trace the Earley algorithm to show the sequence of states created in parsing the sentence:

Sana drinks coffee with milk

S->NP VP	VP->VP PP	
NP->NP PP	VP->Verb NP	Noun->Sana
NP->Noun	Verb->drinks	Noun->milk
PP->Prep NP	Prep->with	Noun->coffee

Earley Parser

	0	1	2	3	4	5
3				PP→.Prep NP Prep→.with		
2			NP→.NP PP NP→.Noun Noun→.coffee	Noun→coffee. NP→Noun. NP→NP.PP		
1		PP→.Prep NP VP→.Verb NP VP→.VP PP Verb→.drinks	Verb→drinks. VP→Verb.NP			
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana	drinks	coffee.	with	milk

Earley Parser

	0	1	2	3	4	5
3				PP→.Prep NP Prep→.with	Prep→with.	
2			NP→.NP PP NP→.Noun Noun→.coffee	Noun→coffee. NP→Noun. NP→NP.PP		
1		PP→.Prep NP VP→.Verb NP VP→.VP PP Verb→.drinks	Verb→drinks. VP→Verb.NP			
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana	drinks	coffee	with.	milk

Earley Parser

	0	1	2	3	4	5
3				PP→.Prep NP Prep→.with	Prep→with.	
2			NP→.NP PP NP→.Noun Noun→.coffee	Noun→coffee. NP→Noun. NP→NP.PP		
1		PP→.Prep NP VP→.Verb NP VP→.VP PP Verb→.drinks	Verb→drinks. VP→Verb.NP			
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana	drinks	coffee	with.	milk

Earley Parser

	0	1	2	3	4	5
3				PP→.Prep NP Prep→.with	Prep→with. PP→Prep.NP	
2			NP→.NP PP NP→.Noun Noun→.coffee	Noun→coffee. NP→Noun. NP→NP.PP		
1		PP→.Prep NP VP→.Verb NP VP→.VP PP Verb→.drinks	Verb→drinks. VP→Verb.NP			
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana	drinks	coffee	with.	milk

Earley Parser

Trace the Earley algorithm to show the sequence of states created in parsing the sentence:

Sana drinks coffee with milk

S->NP VP	VP->VP PP	
NP->NP PP	VP->Verb NP	Noun->Sana
NP->Noun	Verb->drinks	Noun->milk
PP->Prep NP	Prep->with	Noun->coffee

Earley Parser

	0	1	2	3	4	5
					NP→.NP PP	
3				PP→.Prep NP Prep→.with	Prep→with. PP→Prep.NP	
2			NP→.NP PP NP→.Noun Noun→.coffee	Noun→coffee. NP→Noun. NP→NP.PP		
1		PP→.Prep NP VP→.Verb NP VP→.VP PP Verb→.drinks	Verb→drinks. VP→Verb. NP			
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana	drinks	coffee	with.	milk

Earley Parser

	0	1	2	3	4	5
					NP→.NP PP NP→.Noun	
3				**PP→.Prep NP** Prep→.with	Prep→with. PP→Prep.NP	
2			NP→.NP PP **NP→.Noun** Noun→.coffee	Noun→coffee. NP→Noun. NP→NP.PP		
1		PP→.Prep NP **VP→.Verb NP** VP→.VP PP Verb→.drinks		Verb→drinks. VP→Verb. NP		
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana	drinks	coffee	with.	milk

Earley Parser

	0	1	2	3	4	5
					NP→.NP PP **NP→.Noun**	
3				**PP→.Prep NP** Prep→.with	Prep→with. PP→Prep.NP	
2			NP→.NP PP **NP→.Noun** Noun→.coffee	Noun→coffee. NP→Noun. NP→NP.PP		
1		PP→.Prep NP **VP→.Verb NP** VP→.VP PP Verb→.drinks	Verb→drinks. VP→Verb. NP			
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana	drinks	coffee	with.	milk

Earley Parser

	0	1	2	3	4	5
					NP→.NP PP **NP→.Noun**	
3				**PP→.Prep NP** Prep→.with	Prep→with. PP→Prep.NP	
2			NP→.NP PP **NP→.Noun** Noun→.coffee	Noun→coffee. NP→Noun. NP→NP.PP		
1		PP→.Prep NP **VP→.Verb NP** VP→.VP PP Verb→.drinks	Verb→drinks. VP→Verb. NP			
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana	drinks	coffee	with.	milk

Earley Parser

Trace the Earley algorithm to show the sequence of states created in parsing the sentence:

Sana drinks coffee with milk

S->NP VP	VP->VP PP	
NP->NP PP	VP->Verb NP	Noun->Sana
NP->Noun	Verb->drinks	Noun->milk
PP->Prep NP	Prep->with	Noun->coffee

Earley Parser

	0	1	2	3	4	5
					NP→.NP PP **NP→.Noun** Noun→.milk	
3				**PP→.Prep NP** Prep→.with	Prep→with. PP→Prep.NP	
2			NP→.NP PP **NP→.Noun** Noun→.coffee	Noun→coffee. NP→Noun. NP→NP.PP		
1		PP→.Prep NP **VP→.Verb NP** VP→.VP PP Verb→.drinks		Verb→drinks. VP→Verb. NP		
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana	drinks	coffee	with.	milk

Earley Parser

	0	1	2	3	4	5
					NP→.NP PP **NP→.Noun** Noun→.milk	Noun→milk.
3				**PP→.Prep NP** Prep→.with	Prep→with. PP→Prep.NP	
2			NP→.NP PP **NP→.Noun** Noun→.coffee	Noun→coffee. NP→Noun. NP→NP.PP		
1		PP→.Prep NP **VP→.Verb NP** VP→.VP PP Verb→.drinks		Verb→drinks. VP→Verb. NP		
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana	drinks	coffee	with	milk.

Earley Parser

	0	1	2	3	4	5
					NP→.NP PP **NP→.Noun** Noun→.milk	Noun→milk.
3				**PP→.Prep NP** Prep→.with	Prep→with. PP→Prep.NP	
2			NP→.NP PP **NP→.Noun** Noun→.coffee	Noun→coffee. NP→Noun. NP→NP.PP		
1		PP→.Prep NP **VP→.Verb NP** VP→.VP PP Verb→.drinks		Verb→drinks. VP→Verb. NP		
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana	drinks	coffee	with	milk.

Earley Parser

	0	1	2	3	4	5
					NP→.NP PP **NP→.Noun** Noun→.milk	Noun→milk. NP→Noun.
3				**PP→.Prep NP** Prep→.with	Prep→with. PP→Prep.NP	
2			NP→.NP PP **NP→.Noun** Noun→.coffee	Noun→coffee. NP→Noun. NP→NP.PP		
1		PP→.Prep NP **VP→.Verb NP** VP→.VP PP Verb→.drinks		Verb→drinks. VP→Verb. NP		
0	S[1]→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana	drinks	coffee	with	milk.

Earley Parser

	0	1	2	3	4	5
					NP→.NP PP **NP→.Noun** Noun→.milk	Noun→milk. NP→Noun.
3				**PP→.Prep NP** Prep→.with	Prep→with. PP→Prep.NP	
2			NP→.NP PP **NP→.Noun** Noun→.coffee	Noun→coffee. NP→Noun. NP→NP.PP		
1		PP→.Prep NP **VP→.Verb NP** VP→.VP PP Verb→.drinks		Verb→drinks. VP→Verb. NP		
0	S[1]→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana	drinks	coffee	with	milk.

Earley Parser

	0	1	2	3	4	5
3				**PP→.Prep NP** Prep→.with	NP→.NP PP **NP→.Noun** Noun→.milk Prep→with. PP→Prep.NP	Noun→milk. NP→Noun. NP→NP. PP
2			NP→.NP PP **NP→.Noun** Noun→.coffee	Noun→coffee. NP→Noun. NP→NP.PP		
1		PP→.Prep NP **VP→.Verb NP** VP→.VP PP Verb→.drinks	Verb→drinks. VP→Verb. NP			
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana	drinks	coffee	with	milk.

Earley Parser

	0	1	2	3	4	5
3				**PP→.Prep NP** Prep→.with	NP→.NP PP **NP→.Noun** Noun→.milk Prep→with. PP→Prep.NP	Noun→milk. NP→Noun. NP→NP. PP PP→Prep NP.
2			NP→.NP PP **NP→.Noun** Noun→.coffee	Noun→coffee. NP→Noun. NP→NP.PP		
1		PP→.Prep NP **VP→.Verb NP** VP→.VP PP Verb→.drinks	Verb→drinks. VP→Verb. NP			
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana	drinks	coffee	with	milk.

Earley Parser

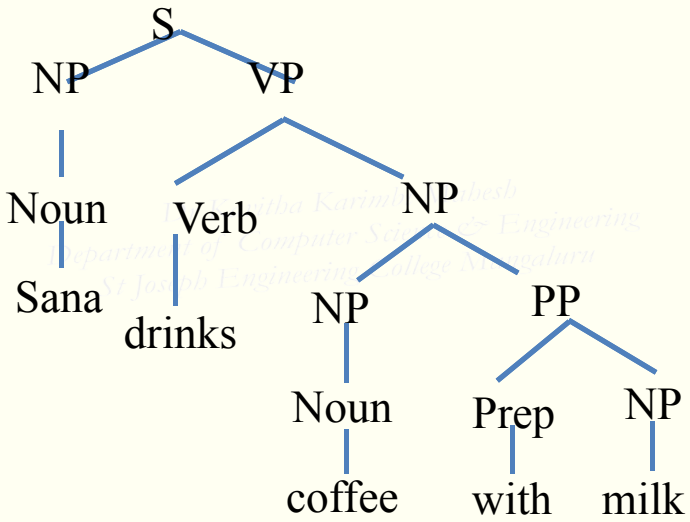
	0	1	2	3	4	5
3					NP→.NP PP **NP→.Noun** Noun→.milk	Noun→milk. NP→Noun. NP→NP. PP
2			NP→.NP PP **NP→.Noun** Noun→.coffee	**PP→.Prep NP** Prep→.with	Prep→with. PP→Prep.NP	PP→Prep NP. NP→NP PP.
1		PP→.Prep NP **VP→.Verb NP** VP→.VP PP Verb→.drinks	Verb→drinks. VP→Verb. NP	Noun→coffee. NP→Noun. NP→NP.PP		
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana	drinks	coffee	with	milk.

Earley Parser

	0	1	2	3	4	5
3				**PP→.Prep NP** Prep→.with	NP→.NP PP **NP→.Noun** Noun→.milk Prep→with. PP→Prep.NP	Noun→milk. NP→Noun. NP→NP. PP PP→Prep NP.
2			NP→.NP PP **NP→.Noun** Noun→.coffee	Noun→coffee. NP→Noun. NP→NP.PP		NP→NP PP.
1		PP→.Prep NP **VP→.Verb NP** VP→.VP PP Verb→.drinks	Verb→drinks. VP→Verb. NP			VP→Verb NP.
0	S^1→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				
		Sana	drinks	coffee	with	milk.

Earley Parser

Example: *'Sana drinks coffee with milk'*



Earley Parser

	0	1	2	3	4	5
3					NP→.NP PP **NP→.Noun** Noun→.milk	Noun→milk. NP→Noun. NP→NP. PP
2			NP→.NP PP **NP→.Noun** Noun→.coffee	**PP→.Prep NP** Prep→.with	Prep→with. PP→Prep.NP	PP→Prep NP. NP→NP PP.
1		PP→.Prep NP **VP→.Verb NP** VP→.VP PP Verb→.drinks	Verb→drinks. VP→Verb. NP	Noun→coffee. NP→Noun. NP→NP.PP		VP→Verb NP.
0	S[1]→.S S→.NP VP NP→.NP PP NP→.Noun Noun→.Sana	Noun→Sana. NP→Noun. NP→NP.PP S→NP.VP				S→NP VP.
		Sana	drinks	coffee	with	milk.

Question Bank- Chapter 4

1.Write a note on different phrase level constructs with suitable example for each phrase -8M

2.Write an algorithm for simple basic top down parser. Illustrate the step by step parsing of the sentence 'open the door' using the same-8M

3. What are advantages and disadvantages of top down and bottom up parsing and give top down and bottom up search space for sentence 'paint the door' by applying following grammar.-10M

S→ NP VP
S→ VP
NP→ Det Nominal
NP→ Noun
NP→ Det Noun PP
Nominal -> Noun
Nominal -> Noun Nominal
VP -> Verb NP
VP -> Verb
PP -> Preposition NP
Det → this| that | a | the
Verb → sleeps | paint | open | sings
Preposition → from | with | on | to
Pronoun → She | he | they

Trace the Earley algorithm to show the sequence of states created in parsing the sentence:

Sana drinks coffee with milk

10 M

S->NP VP	VP->VP PP	
NP->NP PP	VP->Verb NP	Noun->Sana
NP->Noun	Verb->drinks	Noun->milk
PP->Prep NP	Prep->with	Noun->coffee