NLP — Unit I & Unit II

A semester-exam study guide with explanations, diagrams, numerical worked examples, and practice questions.

Contents

Unit I — Foundations of NLP

1. Introduction to NLP and its applications

Definition: Natural Language Processing (NLP) is the study and engineering of systems that can understand, interpret, and generate human language in a useful way. It blends linguistics, statistics, and machine learning.

Major application areas (short list): - Text classification (spam detection, sentiment analysis) - Information extraction (NER, relation extraction) - Machine translation (EN ↔ FR, etc.) - Summarization (news, documents) - Question answering and chatbots - Speech recognition (ASR) and text-to-speech (TTS) - Search and retrieval (semantic search)

Exam tip: When asked for applications, give 3 concrete examples and one short sentence on how NLP helps (e.g., sentiment analysis helps measure public opinion automatically).

---

2. Components of NLP: NLU vs NLG

NLU (Natural Language Understanding) - Goal: Map text to a meaning representation. - Tasks: tokenization, POS tagging, parsing, NER, semantic role labeling, coreference resolution, intent detection. - Example pipeline: raw text → tokenizer → POS tagger → parser → semantic analyzer.

NLG (Natural Language Generation) - Goal: Produce coherent, grammatical text from data or meaning representations. - Tasks: content planning, sentence planning, surface realization, text editing. - Example: Data → content selection → microplanning (decide words) → realization (generate sentences).

Comparison table (exam-ready):

Aspect	NLU	NLG
Direction	Text → Meaning	Meaning/Data → Text
Example task	NER, parsing	Summarization, text generation
Typical errors	misclassification, missed entities	ungrammatical output, hallucination

Diagram (conceptual):

```
      +--------+            +--------+
      |  Input |--NLU-->[Semantic Representation]--NLG-->| Output |
      +--------+            +--------+
```

---

3. Morphology: Lexemes, Morphemes, Morphological Models

Definitions: - **Morpheme:** Smallest meaningful unit in a language (e.g., `un-`, `happy`, `-s`). - **Lexeme:** Abstract unit of lexical meaning; different surface word forms (inflections) belong to one lexeme (e.g., `run, runs, ran, running` belong to lexeme RUN).

Types of morphemes: - Free morpheme: can stand alone (`book`, `run`). - Bound morpheme: must attach (`-ed`, `-s`, `un-`). - Inflectional morphemes: modify tense/number/degree (English: `-s`, `-ed`, `-ing`). - Derivational morphemes: create new lexemes or change word class (`happy` → `unhappy`, `act` → `action`).

Morphological Models: 1. **Concatenative morphology** (affixation): base + suffix/prefix. Example: `play` + `-ed` → `played`. 2. **Non-concatenative morphology**: internal changes, common in Semitic languages (e.g., Arabic root `k-t-b` → `kataba`, `kitab`). 3. **Finite State Transducers (FSTs)**: common formalism to model morphology—mapping between lexical and surface forms using states and transitions. 4. **Paradigm-based models**: view a lexeme as a paradigm of forms (e.g., `go`, `went`, `gone`).

Morphological analysis pipeline (diagram):

```
Raw word -> tokenizer -> candidate segmentation -> morphological analyzer
(FST or NN) -> lemma + features (POS, tense, number)
```

Worked example — segmentation: - Input: `unhappiness` - Segmentation: `un-` + `happy` + `-ness` - Interpretation: derivational prefix `un-` (negation) + adjective `happy` -> noun-former `-ness` => `state of not happy`.

Numerical exercise (morphology): Given a small lexicon where stems = {`play`, `try`}, suffixes = {`-s`, `-ed`, `-ing`}. - How many surface forms can you generate? Answer: 2 stems × 4 forms (base + 3 suffixes) = **8** forms (`play, plays, played, playing, try, tries, tried, trying`). Note: `try` + `-s` -> `tries` involves orthographic rule (y→ies).

---

4. Document structure analysis and complexity

Document structure levels: - Character → Token → Sentence → Paragraph → Section → Document

Key tasks & why they matter: - **Sentence segmentation**: identify sentence boundaries (challenging with abbreviations: `e.g.`, `Mr.`). - **Paragraph/section detection**: helpful for summarization and indexing. - **Document layout analysis**: for scanned documents, PDFs — detect headers, footers, figures, tables.

Complexity aspects: - **Computational complexity**: many NLP operations are linear in text length (O(n)) — tokenization, simple taggers; parsing algorithms may be superlinear (e.g., CYK O(n^3)). - **Readability formulas (numerical)** — useful as document-level features: - **Flesch Reading Ease**: 206.835 − 1.015 × (total words / total sentences) − 84.6 × (total syllables / total words) - **Flesch–Kincaid Grade Level**: 0.39 × (words/sentences) + 11.8 × (syllables/words) − 15.59

Worked numerical example — Flesch Reading Ease: - Suppose a document has 120 words, 8 sentences, and 180 syllables. - words/sentences = 120/8 = 15 - syllables/words = 180/120 = 1.5 - Score = 206.835 − 1.015×15 − 84.6×1.5 = 206.835 − 15.225 − 126.9 = 64.71 - Interpretation: 60–70 = standard; fairly easy to read.

Exam tip: If asked to compare complexity, reference algorithmic complexity (e.g., parser runtime) and human-readability metrics.

Unit II — Syntax and Parsing

1. Parsing techniques: Top-down vs Bottom-up

Top-down parsing (goal-driven): starts from the start-symbol and tries to rewrite it to the input sequence. - Example: Recursive descent parser. - Pros: intuitive, easy to implement for grammars without left-recursion. - Cons: can waste work exploring impossible branches, struggles with left-recursive grammars.

Bottom-up parsing (data-driven): starts from input tokens and attempts to build up to the start symbol by combining recognized constituents. - Example: Shift-reduce parser, CYK (chart-based bottom-up in CNF). - Pros: robust for ambiguous grammars, dynamic programming variants avoid repeated work. - Cons: can build partial constituents that never lead to a full parse.

Comparison diagram (conceptual):

```
Top-down: S -> ... -> tokens
Bottom-up: tokens -> ... -> S
```

Practical note: Modern systems often use chart parsers (combining strategies) and statistical models to guide search.

---

2. Treebanks and syntactic representations

Treebanks: annotated corpora with syntactic trees (constituency) or dependency links. - Examples: Penn Treebank (constituency), Universal Dependencies (UD — dependency).

Constituency (phrase structure) trees: show hierarchical phrase structure (NP, VP, PP...). Example tree (ASCII):

```
        S
       /
     NP    VP
     |     /
    She   V    NP
          |     |
         eats apples
```

Dependency trees: show head-dependent relations (direct relations between words). Example (text: `She eats apples`): `eats` is root; `She` → nsubj(eats); `apples` → obj(eats).

Why multiple representations? - Constituency captures phrasal grouping; dependency captures syntactic function and is often simpler for languages with free word order.

Exam tip: Be able to draw both representations for a short sentence and mention differences.

---

3. Parsing algorithms (e.g., CYK, Earley)

CYK (Cocke–Younger–Kasami)

- Requires grammar in **Chomsky Normal Form (CNF)**: rules are A → BC or A → a.
- Dynamic programming table: `T[i][j]` contains nonterminals that can generate substring from i (inclusive) with length j.
- Complexity: **O(n^3 × |G|)** where n is sentence length and |G| grammar size.

Small worked example (CYK) Grammar (already in CNF):

```
S -> NP VP
VP -> V NP
NP -> Det N
Det -> 'the'
N -> 'cat' | 'mouse'
V -> 'chased'
```

Sentence: `the cat chased the mouse` (tokens indexed 1..5)

We show the CYK triangular table conceptually (here we list sets):

- Length 1 (j=1):
- T[1,1] (the) = {Det}
- T[2,1] (cat) = {N}
- T[3,1] (chased) = {V}
- T[4,1] (the) = {Det}
- T[5,1] (mouse) = {N}
- Length 2 (j=2): combine adjacent cells:
- T[1,2] (the cat) = {NP} because Det + N -> NP
- T[2,2] (cat chased) = {} (no rule)
- T[3,2] (chased the) = {}
- T[4,2] (the mouse) = {NP}
- Length 3:
- T[1,3] (the cat chased): check splits
    - split 1: T[1,1]=Det, T[2,2]={} → no
    - split 2: T[1,2]={NP}, T[3,1]={V} → NP + V -> no (no A->BC rule) => none
- T[2,3] (cat chased the) → none
- T[3,3] (chased the mouse) → V + NP -> VP (if rule VP->V NP exists), so T[3,3]={VP}
- Length 5 (whole sentence): combine T[1,2] (NP) + T[3,3] (VP) -> S. So S ∈ T[1,5], sentence parsed.

Exam task (CYK numeric): We will give one in practice questions below.

Earley parser

- Works for any context-free grammar (not just CNF).
- Uses chart of states each with (rule, dot position, span start).
- Three operations: **Predict**, **Scan**, **Complete**.
- Complexity: worst-case O(n^3), average O(n^2) or O(n) on unambiguous grammars.

Why use Earley? Flexible—handles left-recursion and full CFGs.

Pseudo-steps (conceptual): 1. Initialize chart[0] with (S' -> • S, 0). 2. For k from 0..n: apply Predict (add expansions), Scan (match tokens), Complete (advance dots and add new states).

---

4. Ambiguity resolution and multilingual issues

Ambiguity types & examples

- **Lexical ambiguity:** word with multiple POS or senses (e.g., `bank` — financial vs river).
- **Structural (syntactic) ambiguity:** multiple parse trees for a sentence.
- Example: "I saw the man with the telescope." → Did I use the telescope, or did the man have it? (PP attachment ambiguity)
- **Attachment ambiguity:** PP can attach to NP or VP.

Resolution strategies: - **Rule-based disambiguation**: grammar constraints, selectional restrictions. - **Statistical / probabilistic parsing**: rank parses by probability (PCFGs), choose highest-scoring parse. - **Lexicalized models**: incorporate head word features for disambiguation (e.g., lexicalized PCFGs, neural parsers). - **Semantic/pragmatic cues**: world knowledge or semantic role plausibility.

Multilingual issues

- **Word order variation**: SVO (English) vs SOV (Hindi), free word order (Russian) — affects parser design.
- **Morphologically rich languages**: languages like Turkish, Finnish have rich inflection; tokenization and morphological analysis become essential.
- **Resource scarcity**: many languages lack treebanks or parallel corpora; use transfer learning, multilingual embeddings, or annotation projection.
- **Scripts and tokenization**: Chinese/Japanese lack whitespace – segmentation needed.

Practical note: For multilingual parsing, Universal Dependencies (UD) offers a cross-linguistic annotation standard that helps training cross-lingual models.

Practice questions (Unit I & II)

Below are exam-style problems grouped by marks. Answers / hints follow.

Short questions (2–4 marks)

1. Define morpheme and lexeme. (2 marks)
2. Give two differences between NLU and NLG. (2 marks)
3. What is a treebank? Name one example. (2 marks)
4. State the time complexity of the CYK algorithm. (2 marks)

Medium questions (6–8 marks)

1. Explain the CYK algorithm with a small example (show table construction). (8 marks)
2. Explain top-down vs bottom-up parsing with pros and cons of each. (6 marks)
3. Describe the steps of an NLG pipeline. (6 marks)

Long questions (10–15 marks)

1. Describe Transformer vs RNN approaches for syntactic information. Why are attention-based models preferred now? (15 marks)
2. Discuss ambiguity in parsing with examples and at least three strategies to resolve it. (12 marks)
3. Explain the Earley parsing algorithm and give an example of state transitions on a 3-word sentence. (15 marks)

Numerical / worked problems (show full arithmetic where required)

1. (CYK numeric) Convert the following grammar to CNF if necessary and run CYK on the sentence `the dog chased the cat`.

Grammar:

```
S -> NP VP
VP -> V NP
NP -> Det N | 'dogs' | 'cats'
Det -> 'the'
N -> 'dog' | 'cat'
V -> 'chased'
```

Fill the CYK table and state whether S derives the sentence.

1. (Readability) A document has 300 words, 12 sentences, and 420 syllables. Compute the Flesch Reading Ease score and interpret it.

2. (Morphology) Given stems {`act`, `play`} and suffixes {`-ion`, `-ing`, `-s`}, list all derived forms and mark which are derivational vs inflectional.

Answers / Hints (brief)

Short answers: 1. Morpheme: smallest meaningful unit. Lexeme: set of word forms representing the same lexical item (e.g., `run` forms). 2. NLU: input->meaning; NLG: meaning->text. NLU tasks = parsing/NER; NLG tasks = surface realization/summarization. 3. Treebank: annotated corpus with parse trees. Example: Penn Treebank. 4. CYK complexity: O(n^3 × |G|).

Medium & long hints: 5. For CYK: transform grammar to CNF, fill diagonal with preterminal symbols matching tokens, then fill higher cells by combining smaller spans. Show one or two combination steps. 6. Top-down: start from start symbol; bottom-up: start from tokens. Mention left recursion issue and ambiguity handling. 7. NLG pipeline: content determination → document structuring → microplanning (lexicalization, referring expressions) → surface realization → revision.

Numerical answers: 11. CYK table (sketch): - Tokens: `the` `dog` `chased` `the` `cat` (index 1..5) - Length1: `the` →{Det}, `dog` →{N}, `chased` →{V}, `the` →{Det}, `cat` →{N} - Length2: `the `dog` →{NP}, `dog chased` →{}, `chased the` →{}, `the cat` →{NP} - Length3: `the dog chased` →{} , `dog chased the` →{} , `chased the cat` →{VP} - Length5: `the dog` (NP) + `chased the cat` (VP) → S. So sentence is derived.

1. Flesch: words/sentences = 300/12 = 25; syllables/word = 420/300 = 1.4 Score = 206.835 − 1.015×25 − 84.6×1.4 = 206.835 − 25.375 − 118.44 = 62. (≈62) → fairly easy to read.

2. Derived forms: `act`, `act-ion` (action — derivational), `act-ing` (acting — inflectional/ gerundive - can be considered derivational to noun? mark as gerund/participle), `act-s` (acts — inflectional). Similarly for `play`: `play`, `play-ion` (no common word), `play-ing` (playing — inflectional), `play-s` (plays — inflectional). Mark `-ion` as derivational, `-s` as inflectional, `-ing` can be inflectional (progressive) or derivational when forming nouns (gerund), mention nuance.

Quick revision checklist & exam tips

- Memorise key definitions (morpheme, lexeme, POS, parse tree, CFG, CNF).
- Practice drawing one parse tree (constituency) and one dependency graph for a short sentence.
- Be able to run one CYK example by hand for a 4–6 token sentence.
- For long answers, plan 1 paragraph definition + 2–3 paragraphs explaining mechanism + 1 paragraph limitations/uses.

If you want, I can now: - convert this document into a printable PDF for you, or - expand any section into more detailed lecture-style notes (with more diagrams), or - generate **10 practice questions with full model answers** at 15/10/5 mark levels.

Which would you like next?