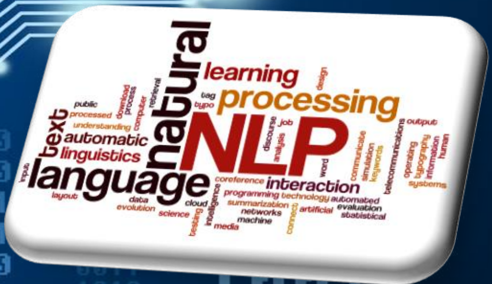
Cairo University

Cairo University
Faculty of Engineering
Computer Engineering Department

# Natural Language Processing

Dr. Sandra Wahid



# Syntactic Parsing

- It is the task of assigning a **syntactic structure** to a sentence.
- It useful in applications such as:
  - Grammar checking: sentence that cannot be parsed may have grammatical errors (or at least be hard to read).
  - Semantic analysis
  - Machine translation
  - Question answering: for example to answer the question: “Which flights to Denver depart before the Seattle flight?”

  →we’ll need to know that the questioner wants a list of flights going to Denver, not flights going to Seattle.

Two views of syntactic structures

- Constituency Parsing
- Dependency Parsing

# Constituency

- Syntactic constituency is the idea that **groups of words** can behave as single units, or constituents.

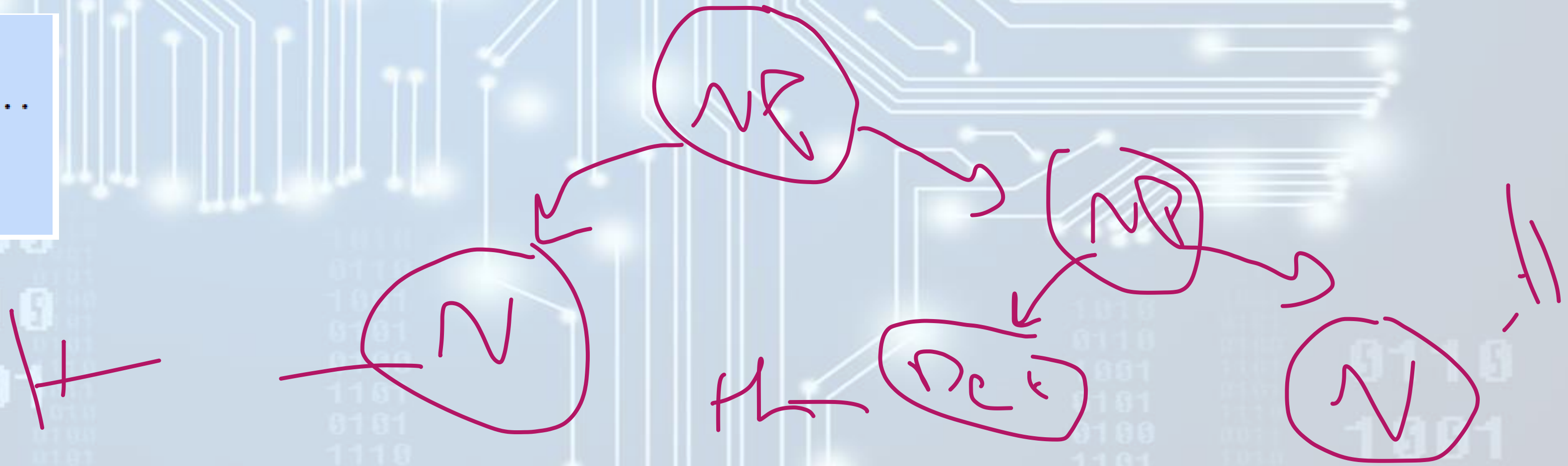
- Example: ***noun phrase*** "a sequence of words surrounding at least one noun" form constituents. Why??

| | |
|---|---|
| Harry the Horse | a high-class spot such as Mindy's |
| the Broadway coppers | the reason he comes into the Hot Box |
| they | three parties from Brooklyn |

All these are examples of noun phrases

they can all appear in **similar syntactic environments**, for example, before a verb.

three parties from Brooklyn *arrive*...
a high-class spot such as Mindy's *attracts*...
the Broadway coppers *love*...
they *sit*



# Context-Free Grammars

- The most widely used formal system for modeling constituent structure in English and other natural languages is the Context-Free Grammar (CFG).
- Also called Phrase-Structure Grammars.
- A context-free grammar consists of a set of **rules or productions**, each of which expresses the ways that symbols of the language can be grouped and ordered together, and a **lexicon** of words and symbols.
- Example: the following productions express that an NP (or noun phrase) can be composed of either a ProperNoun or a determiner (Det) followed by a Nominal, a Nominal in turn can consist of one or more Nouns.

$$
\begin{aligned}
NP &\rightarrow Det\ Nominal \\
NP &\rightarrow ProperNoun \\
Nominal &\rightarrow Noun \mid Nominal\ Noun
\end{aligned}
$$

  - Context-free rules can be hierarchically embedded, so we can combine the previous rules with others that express facts about the lexicon:

$$
\begin{aligned}
Det &\rightarrow a \\
Det &\rightarrow the \\
Noun &\rightarrow flight
\end{aligned}
$$

# Context-Free Grammars

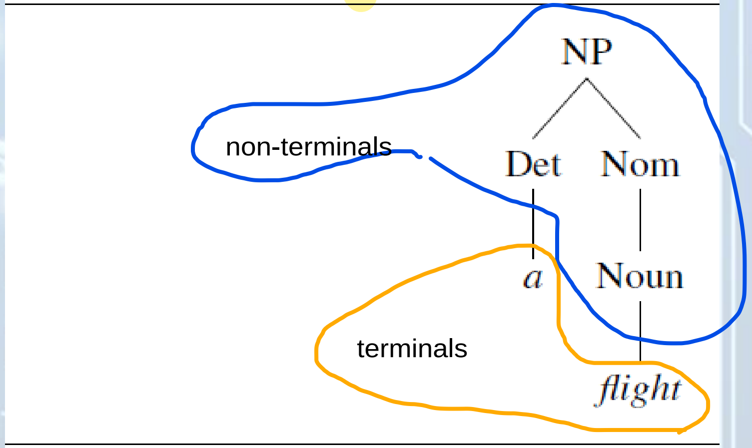
B -> acGVxl

A -> BC
B -> play
RHS
1. single terminal
2. 2 non-terminal symbols.

- The symbols that are used in a CFG are divided into two classes:
  - **Terminals:** the symbols that correspond to words in the language, the lexicon is the set of rules that introduce these terminal symbols.
  - **Non-terminals:** the symbols that express abstractions over these terminals.
- Format of context-free rule: <a single non-terminal symbol> → <ordered list of one or more terminals and non-terminals>
- CFG can be used to generate a set of strings. This sequence of rule expansions is called a **derivation** of the string of words. It is common to represent a derivation by a **parse tree.**

$$
\begin{aligned}
NP &\rightarrow Det\ Nominal \\
NP &\rightarrow ProperNoun \\
Nominal &\rightarrow Noun \mid Nominal\ Noun
\end{aligned}
$$

$$
\begin{aligned}
Det &\rightarrow a \\
Det &\rightarrow the \\
Noun &\rightarrow flight
\end{aligned}
$$




A parse tree for “a flight”.

a flight hena hya gomla esmya -> NP -> noun phrase
el NP mokwn mn 7agten -> Det + Nominal
fa el a hena hya el Det. w el nominal hena hwa no3o Noun w el noun el hwa el flight.

VP -> verb phrase.

# Context-Free Grammars

- The formal language defined by a CFG is the **set of strings** that are derivable from the designated **start symbol.**
- Each grammar must have **one designated start symbol**, which is often called **S** where S is usually **interpreted as the "sentence" node.**

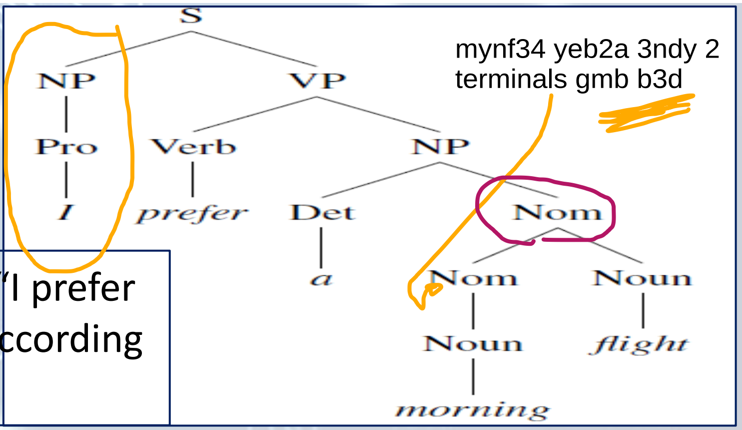
| Grammar Rules | | Examples | |
|---|---|---|---|
| $S$ → | $NP\ VP$ (verb phrase) | I + want a morning flight | |
| $NP$ → | $Pronoun$ | I | pronouns -> I,we,you,they.... |
| \| | $Proper\text{-}Noun$ | Los Angeles | esm m3rof, Paris, USA .... |
| \| | $Det\ Nominal$ | a + flight | The wall, an elephant.... |
| $Nominal$ → | $Nominal\ Noun$ | morning + flight | adj + Noun |
| \| | $Noun$ | flights | |
| $VP$ → | $Verb$ | do | fe3l bs |
| \| | $Verb\ NP$ | want + a flight | fe3l w mf3ol beh. |
| \| | $Verb\ NP\ PP$ | leave + Boston + in the morning | fe3l w mf3ol bh, w zarf aw shebh gomla. |
| \| | $Verb\ PP$ (prepositional phrase) | leaving + on Thursday | |
| $PP$ → | $Preposition\ NP$ | from + Los Angeles | zarf zman aw mkan. |

gomla fe3lya.

The grammar for $\mathcal{L}_0$, with example phrases for each rule.

| | |
|---|---|
| $Noun$ → | *flights* \| *flight* \| *breeze* \| *trip* \| *morning* |
| $Verb$ → | *is* \| *prefer* \| *like* \| *need* \| *want* \| *fly* \| *do* |
| $Adjective$ → | *cheapest* \| *non-stop* \| *first* \| *latest* \| *other* \| *direct* |
| $Pronoun$ → | *me* \| *I* \| *you* \| *it* |
| $Proper\text{-}Noun$ → | *Alaska* \| *Baltimore* \| *Los Angeles* \| *Chicago* \| *United* \| *American* |
| $Determiner$ → | *the* \| *a* \| *an* \| *this* \| *these* \| *that* |
| $Preposition$ → | *from* \| *to* \| *on* \| *near* \| *in* |
| $Conjunction$ → | *and* \| *or* \| *but* |

The lexicon for $\mathcal{L}_0$.




mynf34 yeb2a 3ndy 2 terminals gmb b3d

The parse tree for "I prefer a morning flight" according to grammar L0.

# Formal Definition of Context-Free Grammar

- A context-free grammar G is defined by four parameters: *N, Σ, R, S* (technically this is a "4-tuple").

| | |
|---|---|
| $N$ | a set of **non-terminal symbols** (or **variables**) |
| $\Sigma$ | a set of **terminal symbols** (disjoint from $N$) |
| $R$ | a set of **rules** or productions, each of the form $A \rightarrow \beta$ , |
| | where $A$ is a non-terminal, eh el kwa3ed elly lazm amshy 3lehom. |
| | $\beta$ is a string of symbols from the infinite set of strings $(\Sigma \cup N)^*$ |
| $S$ | a designated **start symbol** and a member of $N$ hbd2 fen |

- The following conventions are followed:

| | |
|---|---|
| Capital letters like $A$, $B$, and $S$ | Non-terminals |
| $S$ | The start symbol |
| Lower-case Greek letters like $\alpha$, $\beta$, and $\gamma$ | Strings drawn from $(\Sigma \cup N)^*$ |
| Lower-case Roman letters like $u$, $v$, and $w$ | Strings of terminals |

# Grammar Equivalence and Normal Form

- A formal language is defined as a (possibly infinite) set of strings of words.
- This suggests that we could ask if two grammars are equivalent by asking if they generate the same set of strings. In fact, it is possible to have two distinct context-free grammars generate the same language.

- We usually distinguish **two kinds of grammar equivalence**:
  - **strong equivalence:** two grammars are strongly equivalent if they generate the same set of strings and if they assign the same phrase structure to each sentence (allowing merely for renaming of the non-terminal symbols).
  - **weak equivalence:** two grammars are weakly equivalent if they generate the same set of strings but do not assign the same phrase structure to each sentence.

- It is sometimes useful to have a normal form for grammars, in which each of the productions takes a particular form.

# Chomsky Normal Form (CNF)

- A context-free grammar is in CNF if:
  - ***ε-free*** (no empty rules).
  - each production is either of the form ***A → B C*** or ***A → a***.

    (each rule either has two non-terminal symbols or one terminal symbol.)
- Any context-free grammar can be converted into a weakly equivalent Chomsky normal form grammar.
  - For example, a rule of the form: A → B C D can be converted into the following two CNF rules: A → B X and X → C D
  - This conversion is done to perform efficient parsing.
- Conversion to CNF steps:

1. Get rid of all $\varepsilon$ productions.
2. Get rid of all productions where RHS is one variable.
3. Replace every production that is too long by shorter productions.
4. Move all terminals to productions where RHS is one terminal.

A -> aB
let G -> a
A -> GB
G -> a

# Chomsky Normal Form (CNF)

**1) Eliminate ε Productions:**

- Determine the nullable variables (those that generate ε)
- Go through all productions, and for each, omit every possible subset of nullable variables. For example, if P → AxB with both A and B nullable, add productions P → xB | Ax | x.
- After this, delete all productions with empty RHS.

**2) Eliminate Variable Unit Productions:**

- A unit production is where RHS has only one variable.
- Consider production A → B. Then for every production B → α, add the production A → α.
- Repeat until done (but don't re-create a unit production already deleted).

**3) Replace Long Productions by Shorter Ones:**

- For example, if have production A → BCD, then replace it with A → BE and E → CD.

**4) Move Terminals to Unit Productions:**

- For every terminal on the right of a non-unit production, add a substitute variable.
- For example, replace production A → bC with productions A → BC and B → b.

# Chomsky Normal Form (CNF)

- Example1:

Consider the CFG:

$$S \to \mathtt{a}X\mathtt{b}X$$
$$X \to \mathtt{a}Y \mid \mathtt{b}Y \mid \varepsilon$$
$$Y \to X \mid \mathtt{c} \mid \epsilon$$

S -> aXbX | abX | aXb | ab
X -> aY | bY | a | b
Y -> aY | bY | a | b | c

**1]** The variable $X$ is nullable; and so therefore is $Y$. After elimination of $\varepsilon$, we obtain:

$$S \to \mathtt{a}X\mathtt{b}X \mid \mathtt{ab}X \mid \mathtt{a}X\mathtt{b} \mid \mathtt{ab}$$
$$X \to \mathtt{a}Y \mid \mathtt{b}Y \mid \mathtt{a} \mid \mathtt{b}$$
$$Y \to X \mid \mathtt{c}$$

hena lw hntb2 tany rule, hanshel el X, w n3wd bl kema bta3tha baa.
Y -> AY | bY | a | b | c

**2]** After elimination of the unit production $Y \to X$, we obtain:

$$S \to \mathtt{a}X\mathtt{b}X \mid \mathtt{ab}X \mid \mathtt{a}X\mathtt{b} \mid \mathtt{ab}$$
$$X \to \mathtt{a}Y \mid \mathtt{b}Y \mid \mathtt{a} \mid \mathtt{b}$$
$$Y \to \mathtt{a}Y \mid \mathtt{b}Y \mid \mathtt{a} \mid \mathtt{b} \mid \mathtt{c}$$

S -> aXbX
A -> a
B -> b
-------------
S -> AXBX | ABX | AXB | AB
X -> AY | BY | a | b
Y -> AY | BY | a | b | c
-------------
AXBX -> E = AX, F = BX
S -> EF | AF | EB | AB
X -> AY | BY | a | b
Y -> AY | BY | a | b | c
E -> AX, F -> BX
A -> a
B -> b

# Chomsky Normal Form (CNF)

- Example1:

**3]&4]** Now, break up the RHSs of $S$; and replace a by $A$, b by $B$ and c by $C$ wherever not units:

$$S \to EF \mid AF \mid EB \mid AB$$
$$X \to AY \mid BY \mid a \mid b$$
$$Y \to AY \mid BY \mid a \mid b \mid c$$
$$E \to AX$$
$$F \to BX$$
$$A \to a$$
$$B \to b$$
$$C \to c$$

ay 7aga aktur mn 7rfen 7wlhom
aXbX -> aktur mn 7rfen -> EF where
E = AX, F = BX
el A nfsha hya el a small, wl B hya el b small
fa keda S = EF = AXBX = aXbX fahna nfs el 7aga lakn mshena 3la el rule.

7war enna n7wl el a -> A da rule 4, w da 34an myb2ash feh 2 terminals gmb b3d, w kman mynf34 en el sentence tebd2 b terminal, fa 34an keda bnbdlhom.

S -> EF | AF | EB | AB
X -> AY | BY | a | b
Y -> AY | BY | a | b | c
E -> AX, F -> BX
A -> a
B -> b

# Chomsky Normal Form (CNF)

- Example2:

Convert the following CFG into Chomsky Normal Form:

$$S \to AbA$$
$$A \to Aa \mid \varepsilon$$

S -> AbA | bA | Ab | b
A -> Aa | a
=============
let AbA = EC
E = AB
B = b
C = a
----------------------
S -> EC | BA | AB | b
A -> AC | a
E -> AB
B -> b
C -> a

mtnsash enk mynf34 t2ol
R = Ae
lazm enk t7wl le 2 captial, aw wahda bs small
R = AE or e

dayman wnta shaghal lazm ykon 3ndk 7aga mn 2
ema 7rfen capital gamb b3d, aw 7rf small lw7do, ay 7aga tanya lazm t7awl tghyrha.

1]
$$S \to AbA \mid bA \mid Ab \mid b$$
$$A \to Aa \mid a$$

3] The second step does not apply. After the third step, one has:

$$S \to TA \mid bA \mid Ab \mid b$$
$$A \to Aa \mid a$$
$$T \to Ab$$

4]
$$S \to TA \mid BA \mid AB \mid b$$
$$A \to AC \mid a$$
$$T \to AB$$
$$B \to b$$
$$C \to a$$

# Chomsky Normal Form (CNF)

- Example3:

| | |
|---|---|
| S → NP VP | N → people |
| VP → V NP | N → fish |
| VP → V NP PP | N → tanks |
| NP → NP NP | N → rods |
| NP → NP PP | V → people |
| NP → N | V → fish |
| NP → *e* | V → tanks |
| PP → P NP | P → with |

S → NP VP
VP → V NP
VP → V NP PP
NP → NP NP
NP → NP PP
NP → N
NP → *e*
PP → P NP

1. Remove empty rules

2. Rewrite as:
S → NP VP
S → VP

3. Repeat

N → people
N → fish
N → tanks
N → rods
V → people
V → fish
V → tanks
P → with

S → NP VP
S → VP
VP → V NP
VP → V
VP → V NP PP
VP → V PP
NP → NP NP
NP → NP
NP → NP PP
NP → PP
NP → N
PP → P NP
PP → P

1. Remove unary rule

S → V NP
S → V
S → V NP PP
S → V PP

2. Rewrite for rules where VP appears on the left

N → *people*
N → *fish*
N → *tanks*
N → *rods*
V → *people*
V → *fish*
V → *tanks*
P → *with*

# Chomsky Normal Form (CNF)

- Example3:

S → NP VP
VP → V NP
S → V NP
VP → V
VP → V NP PP
S → V NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP
NP → NP PP
NP → PP
NP → N
PP → P NP
PP → P

Keep removing *unaries*

N → *people*
N → *fish*
N → *tanks*
N → *rods*
V → *people*
S → *people*
V → *fish*
S → *fish*
V → *tanks*
S → *tanks*
P → *with*

⇒

VP → *people*
VP → *fish*
VP → *tanks*

S → NP VP
VP → V NP
S → V NP
VP → V NP PP
S → V NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP
NP → NP PP
NP → PP
NP → N
PP → P NP
PP → P

Keep removing *unaries*

N → *people*
N → *fish*
N → *tanks*
N → *rods*
V → *people*
S → *people*
VP → *people*
V → *fish*
S → *fish*
VP → *fish*
V → *tanks*
S → *tanks*
VP → *tanks*
P → *with*

⇒

NP → *people*
NP → *fish*
NP → *tanks*
NP → *rods*

# Chomsky Normal Form (CNF)

- Example3:

S → NP VP
VP → V NP
S → V NP
VP → V NP PP
S → V NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP PP
NP → P NP
PP → P NP

Done with unary rules

VP → V @VP_P
@VP_P → NP PP

Replace ternary rule with two binary rules by adding a new non-terminal symbol

NP → *people*
NP → *fish*
NP → *tanks*
NP → *rods*
V → *people*
S → *people*
VP → *people*
V → *fish*
S → *fish*
VP → *fish*
V → *tanks*
S → *tanks*
VP → *tanks*
P → *with*
PP → *with*

S → NP VP
VP → V NP
S → V NP
VP → V @VP_V
@VP_V → NP PP
S → V @S_V
@S_V → NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP PP
NP → P NP
PP → P NP

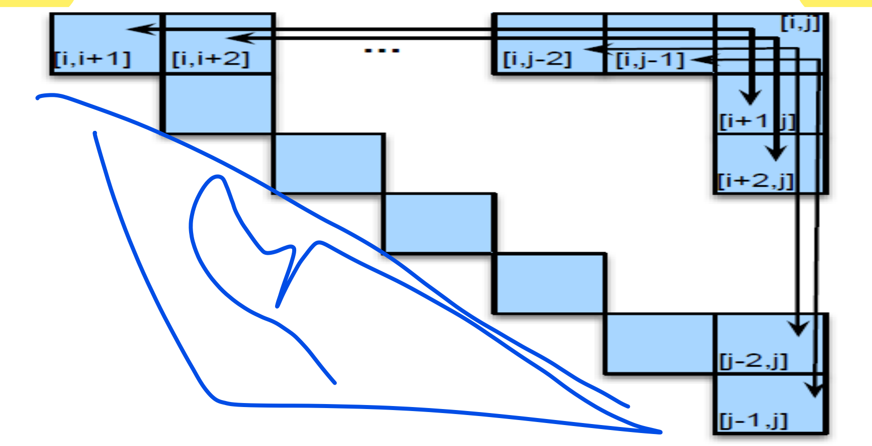
NP → *people*
NP → *fish*
NP → *tanks*
NP → *rods*
V → *people*
S → *people*
VP → *people*
V → *fish*
S → *fish*
VP → *fish*
V → *tanks*
S → *tanks*
VP → *tanks*
P → *with*
PP → *with*

# Constituency Parsing

**CKY (Cocke-Kasami-Younger) Parsing Algorithm:** A dynamic programming approach

to represent all possible parses of the sentence if any

- First need to transform grammar into CNF: produces binary parse trees.
- Bottom-up parsing.
- Dynamic programming: save the results in a table/chart and re-use these results in finding larger constituent.
- A two-dimensional matrix can be used to encode the structure of an entire tree.
- For a sentence of length n, we will work with the upper-triangular portion of an (n+1)x(n+1) matrix.
- Each cell [i,j] in this matrix contains the set of non-terminals that represent all the constituents that span positions i through j of the input.
- It follows then that the cell that represents the entire input resides in position [0,n] in the matrix.
- We fill the upper-triangular matrix a column at a time working from left to right, with each column filled from bottom to top.
- Each cell [i,j] is filled as follows:




# CKY

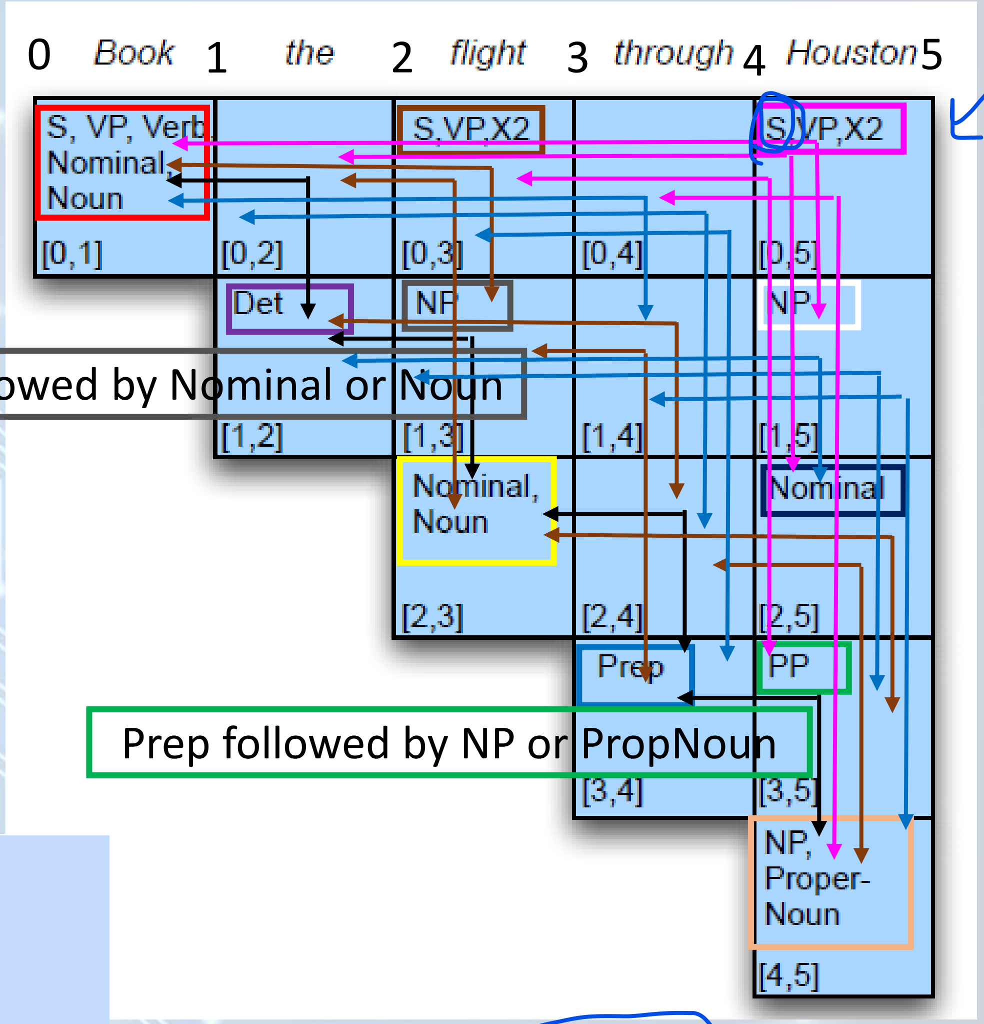
Cell [0,1] spans *Book*
Cell [3,4 ] spans *through*
Cell [2,5] spans *flight through Houston*
Cell [1,4] spans *the flight through*

**$\mathcal{L}_1$ in CNF**

*S → NP VP*
*S → X1 VP*
*X1 → Aux NP*
*S → book | include | prefer*
*S → Verb NP*
*S → X2 PP*
*S → Verb PP*
*S → VP PP*
*NP → I | she | me*
*NP → TWA | Houston*
*NP → Det Nominal*
*Nominal → book | flight | meal | money*
*Nominal → Nominal Noun*
*Nominal → Nominal PP*
*VP → book | include | prefer*
*VP → Verb NP*
*VP → X2 PP*
*X2 → Verb NP*
*VP → Verb PP*
*VP → VP PP*
*PP → Preposition NP*

**Lexicon**

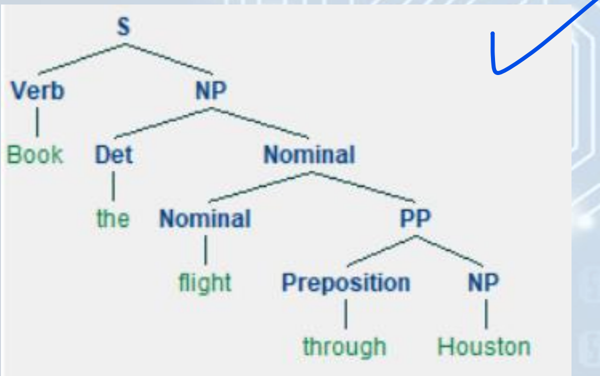
*Det → that | this | the | a*
*Noun → book | flight | meal | money*
*Verb → book | include | prefer*
*Pronoun → I | she | me*
*Proper-Noun → Houston | NWA*
*Aux → does*
*Preposition → from | to | on | near | through*



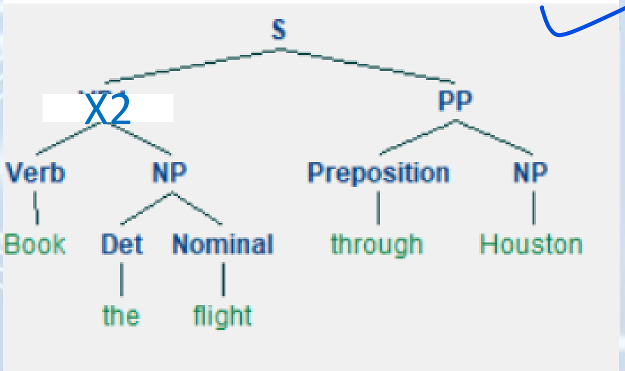
# CKY

- If the cell [0,n] contains S then a valid parse is available for the sentence.
- To get all possible parses for the sentence:
    - Choose an S from cell [0,n] and then recursively retrieve its component constituents from the table.
    - Repeat for all S's in cell [0,n].
- In the example:

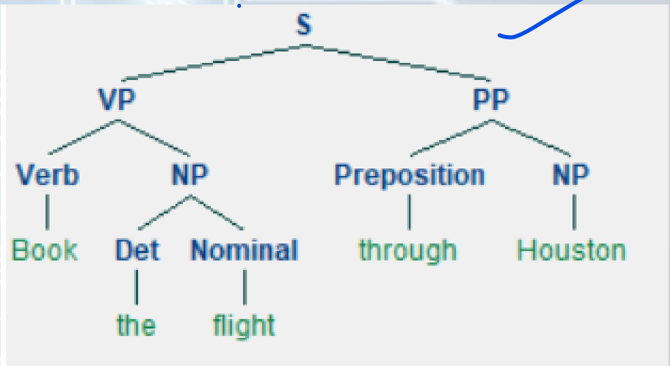
1. S→Verb NP




2. S→X2 PP



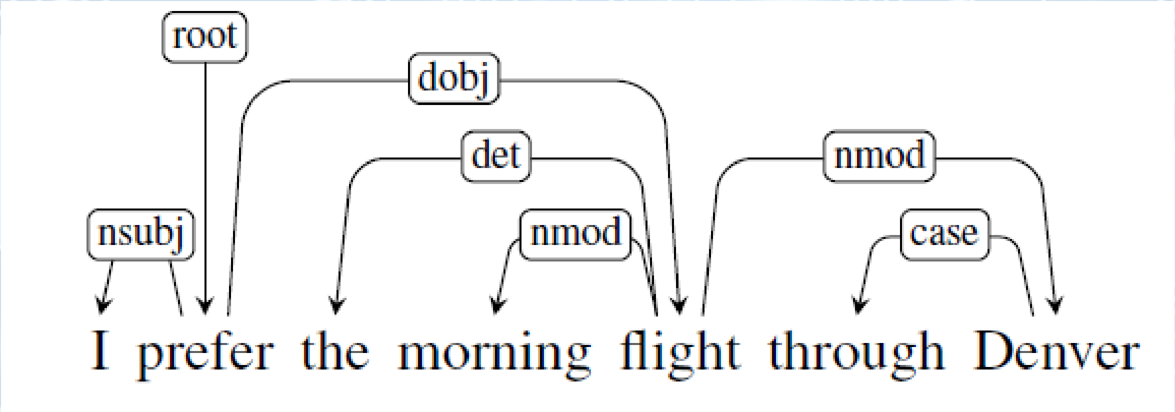

3. S→VP PP




- There may be an exponential number of parses for a given sentence so there are augmentations to the method to retrieve only the **best parse**.

# Dependency Parsing

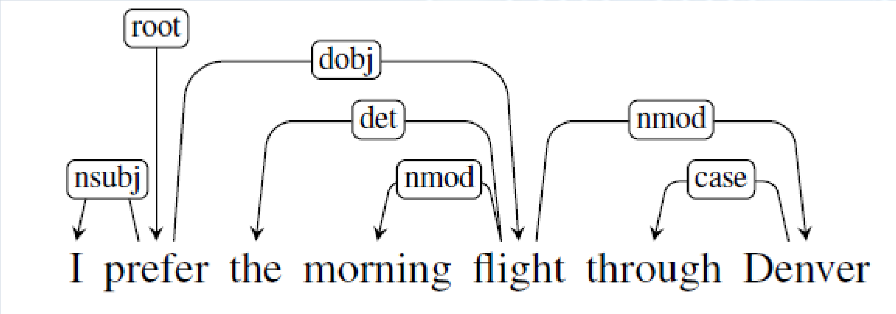
- So far, we have seen context-free grammars and constituent-based representations.
- Another important family of grammar formalisms called **dependency grammars**.
- The syntactic structure of a sentence is described solely in terms of directed **binary grammatical relations** between the words, as in the following **dependency parse**:

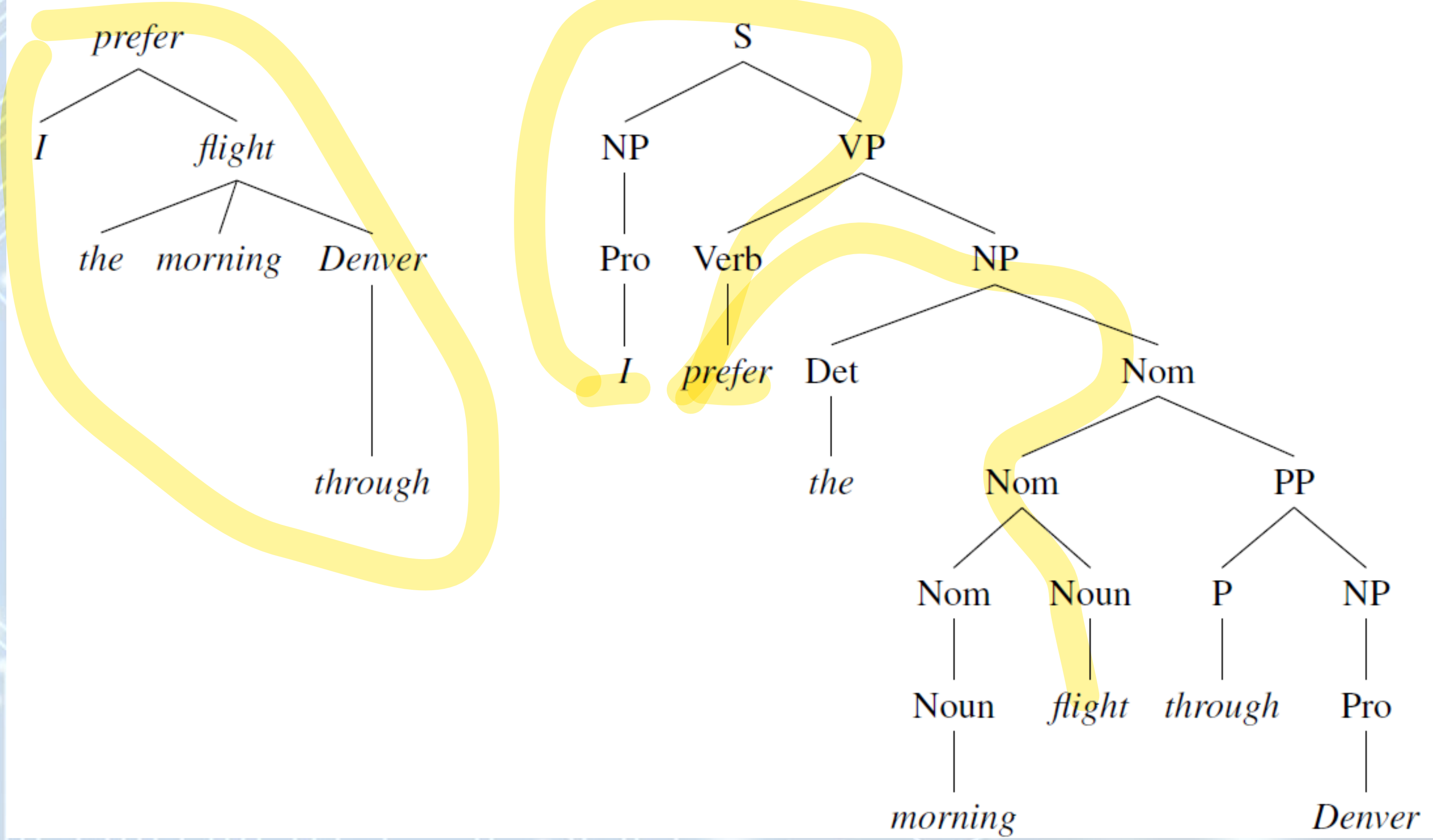



- The parse tree has directed, labeled arcs from ***heads*** to ***dependents***.
- This is called **typed dependency structure** because the labels are drawn from a fixed inventory of grammatical relations.
- A root node explicitly marks the root of the tree, the head of the entire structure.

# Dependency Parsing

- Dependency and constituent analyses for *I prefer the morning flight through Denver:*

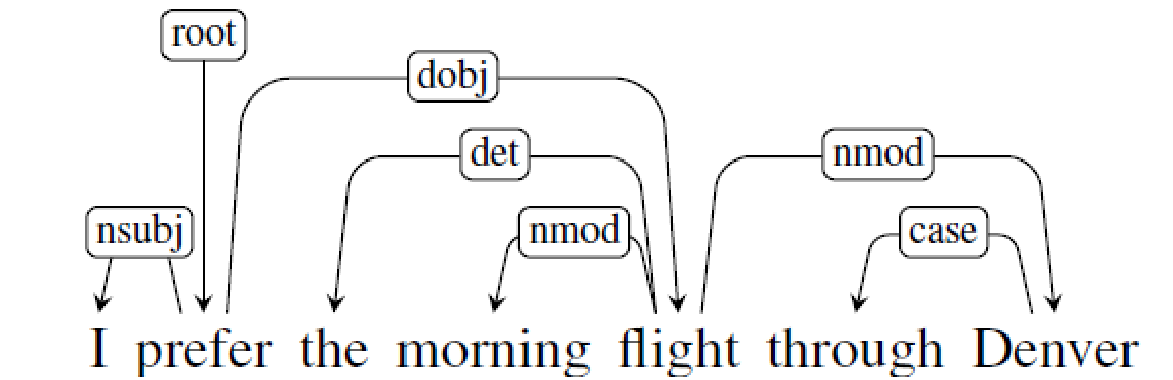



- The arguments to the verb *prefer* are directly linked to it in the dependency structure, while their connection to the main verb is more distant in the phrase-structure tree.
- Similarly, *morning* and *Denver*, modifiers of *flight*, are linked to it directly in the dependency structure.

# Dependency Parsing

- The Universal Dependencies (UD) project provides an inventory of dependency relations that are **cross-linguistically applicable**.
- A subset of the UD relations from the following example:




| Relation | Description |
|---|---|
| nsubj | Nominal subject |
| dobj | Direct object |
| det | Determiner |
| nmod | Nominal modifier |
| case | Prepositions, postpositions and other case markers |

- Dependency parsing approaches include: **transition-based** parsing and **graph-based** parsers.


Thank You