UNIT 3

syntactic analysis

Syntactic analysis may be defined as:

1- determining the relevant components of a sentence

2- describing these parts grammatically.

- The component parts of a sentence are called **constituents.**

Constituency

The idea: Groups of words may behave as a single unit or phrase, called a **consituent**.

E.g. Noun Phrase

Kermit the frog
they
December twenty-sixth
the reason he is running for president

Constituent Phrases

For constituents, we usually name them as phrases based on the word that heads the constituent:

the man from Amherst	is a Noun Phrase (NP) because the head man is a noun
extremely clever	is an Adjective Phrase (AP) because the head clever is an adjective
down the river	is a Prepositional Phrase (PP) because the head down is a preposition
killed the rabbit	is a Verb Phrase (VP) because the head killed is a verb

Context Free Grammars

Context-free grammar

The most common way of modeling constituency.

CFG = Context-Free Grammar = Phrase Structure Grammar = BNF = Backus-Naur Form

Context-free grammar

$G = (T, N, S, R)$

- T is set of terminals (lexicon)
- N is set of non-terminals
- S is start symbol (one of the nonterminals)
- R is rules/productions of the form $X \rightarrow \gamma$, where X is a nonterminal and γ is a sequence of terminals and nonterminals (may be empty).
- A grammar G generates a language L.

An example context-free grammar

$G = (T, N, S, R)$

$T = \{$*that, this, a, the, man, book, flight, meal, include, read, does*$\}$

$N = \{$S, NP, NOM, VP, Det, Noun, Verb, Aux$\}$

$S = S$

$R = \{$

S → NP VP
S → Aux NP VP
S → VP

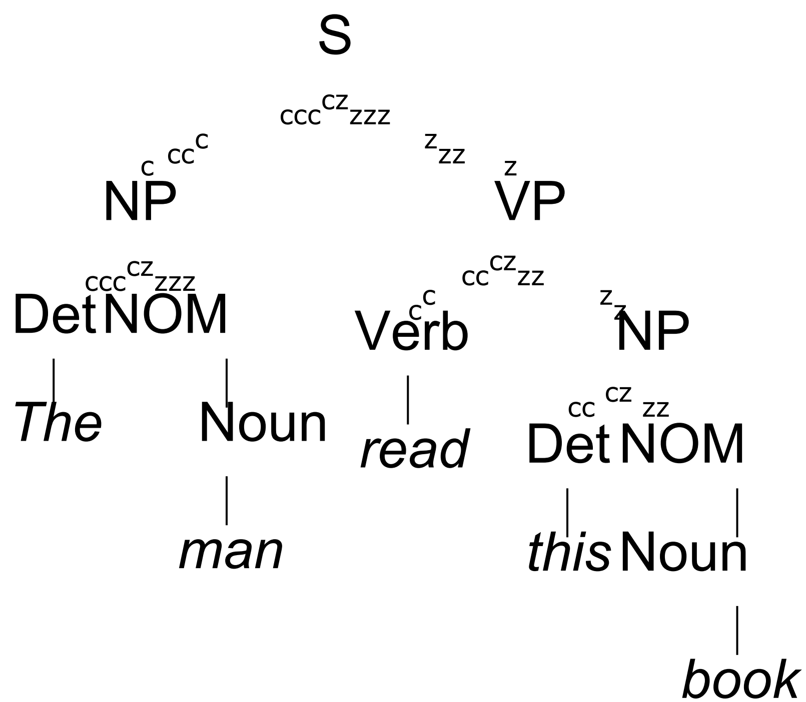
NOM → Noun
NOM → Noun NOM
VP → Verb
VP → Verb NP

NP → Det NOM

Det → *that* | *this* | *a* | *the*
Noun → *book* | *flight* | *meal* | *man*
Verb → *book* | *include* | *read*
Aux → *does*

$\}$

Parse tree



CFGs can capture recursion

Example of seemingly endless recursion of embedded prepositional phrases: PP → Prep NP
NP → Noun PP

$[_S$ The mailman ate his $[_{NP}$ lunch $[_{PP}$ with his friend $[_{PP}$ from the cleaning staff $[_{PP}$ of the building $[_{PP}$ at the intersection $[_{PP}$ on the north end $[_{PP}$ of town]]]]]]].

(Bracket notation)

Grammaticality

A CFG defines a formal language = the set of all sentences (strings of words) that can be derived by the grammar.

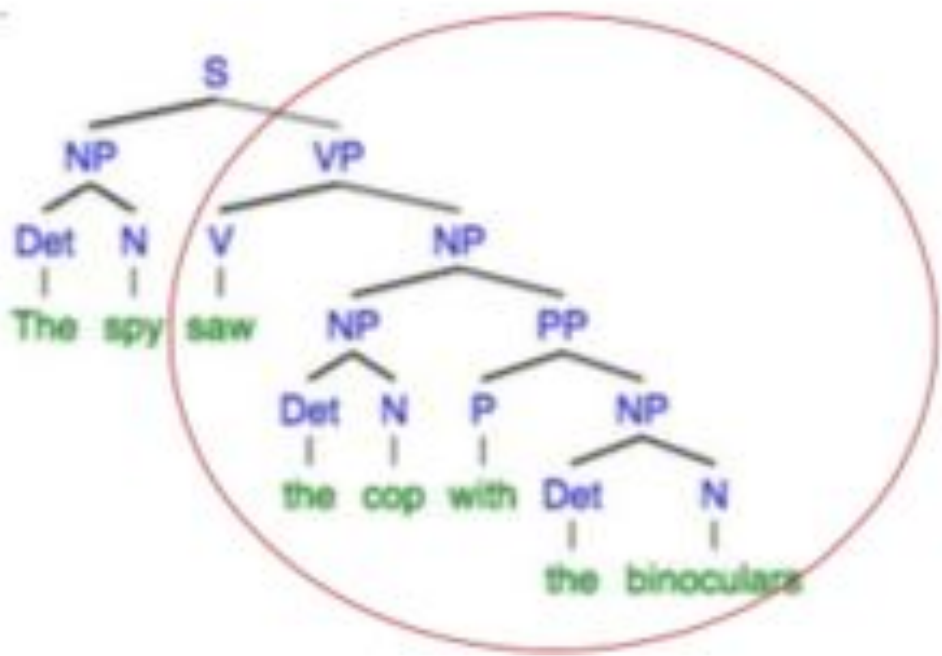
Sentences in this set said to be **grammatical**.

Sentences outside this set said to be **ungrammatical**.

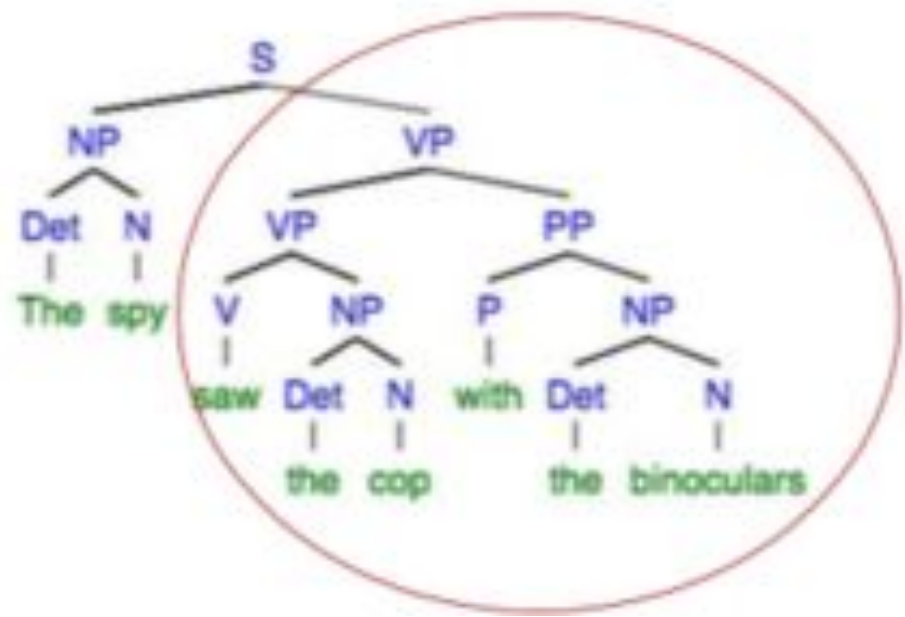
Ambiguity

some sequences of words can be assigned more than one syntactic structure -> different Parse trees

1.



2.

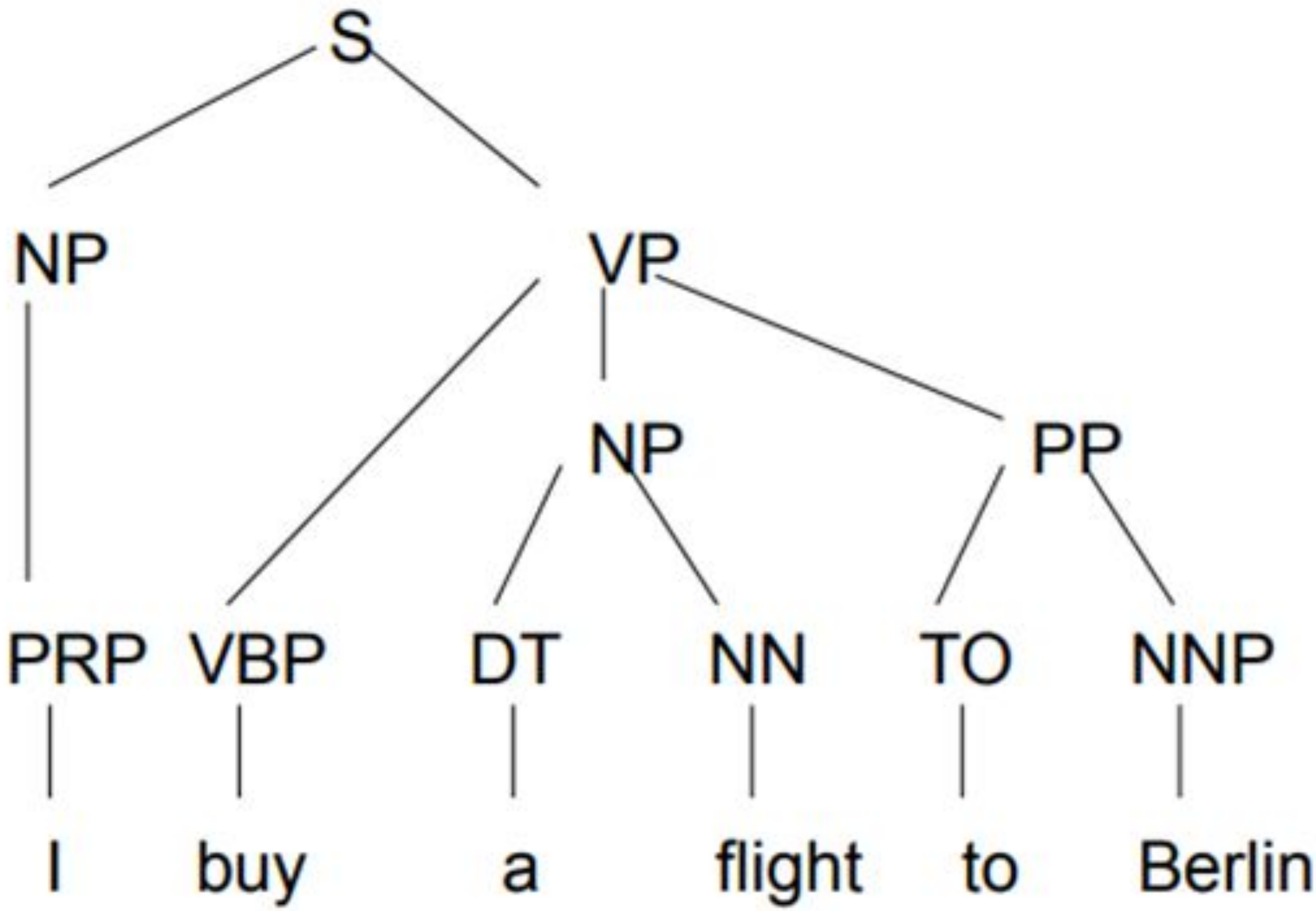


- Given a string and a grammar, return proper parse tree,

NP $\rightarrow$ PRP
NP $\rightarrow$ DT NN
PP $\rightarrow$ TO NNP
VP $\rightarrow$ VBP NP PP
S $\rightarrow$ NP VP

+

I buy a flight to Berlin.

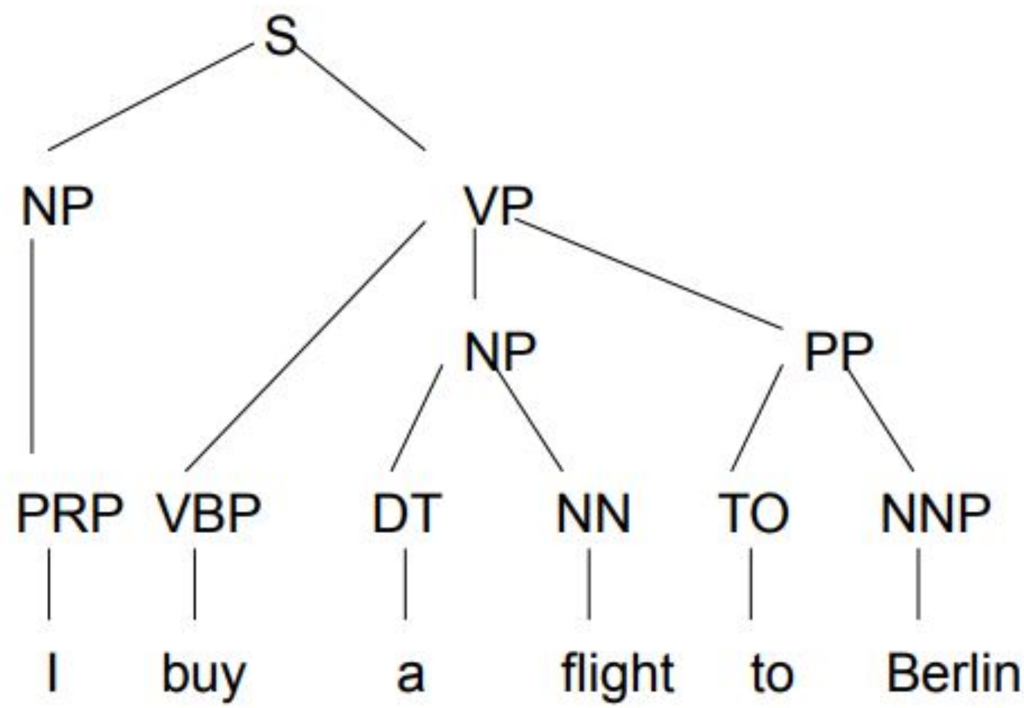

S
NP
VP
NP
PP
PRP
VBP
DT
NN
TO
NNP
I
buy
a
flight
to
Berlin

- Given a string and a grammar, return proper parse tree,

NP → PRP
NP → DT NN
PP → TO NNP
VP → VBP NP PP
S → NP VP

+

I buy a flight to Berlin.



S → NP VP	VP → Verb NP
S → VP	VP → Verb
NP → Det Nominal	PP → Preposition NP
NP → Noun	Det → this ǀ that ǀ a ǀ the
NP → Det Noun PP	Verb → sleeps ǀ sings ǀ open ǀ saw ǀ paint
Nominal → Noun	Preposition → from ǀ with ǀ on ǀ to
Nominal → Noun Nominal	Pronoun → she ǀ he ǀ they

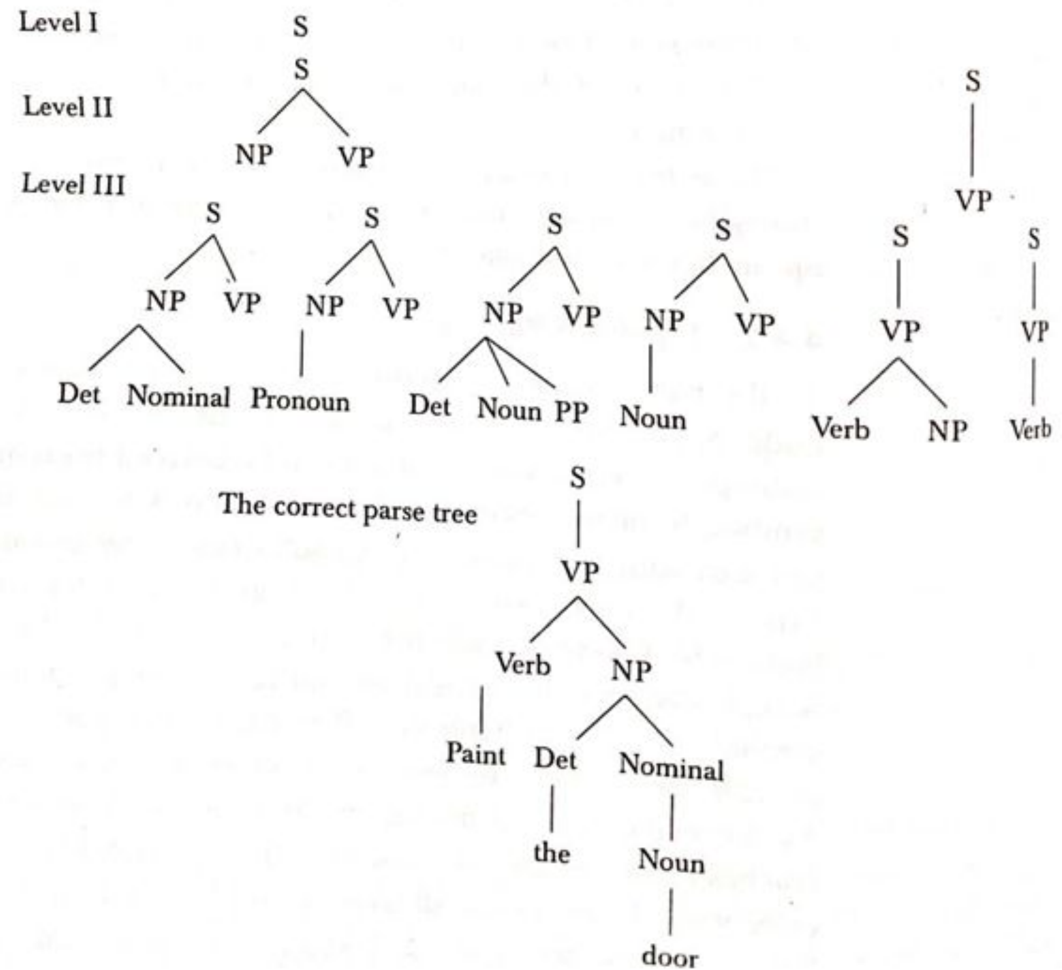
Consider the grammar shown in Table 4.2 and the sentence

Paint the door.

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

Consider the grammar shown in Table 4.2 and the sentence

Paint the door.

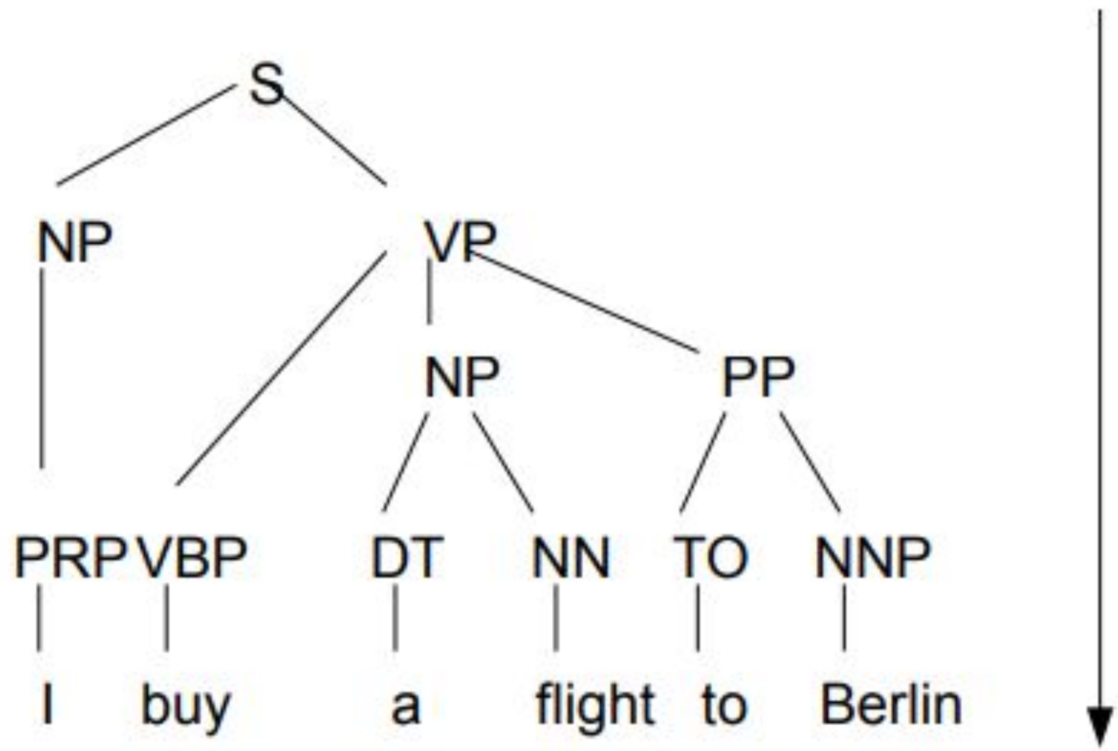


Parsing Algorithms

- Top-Down
- Bottom-up

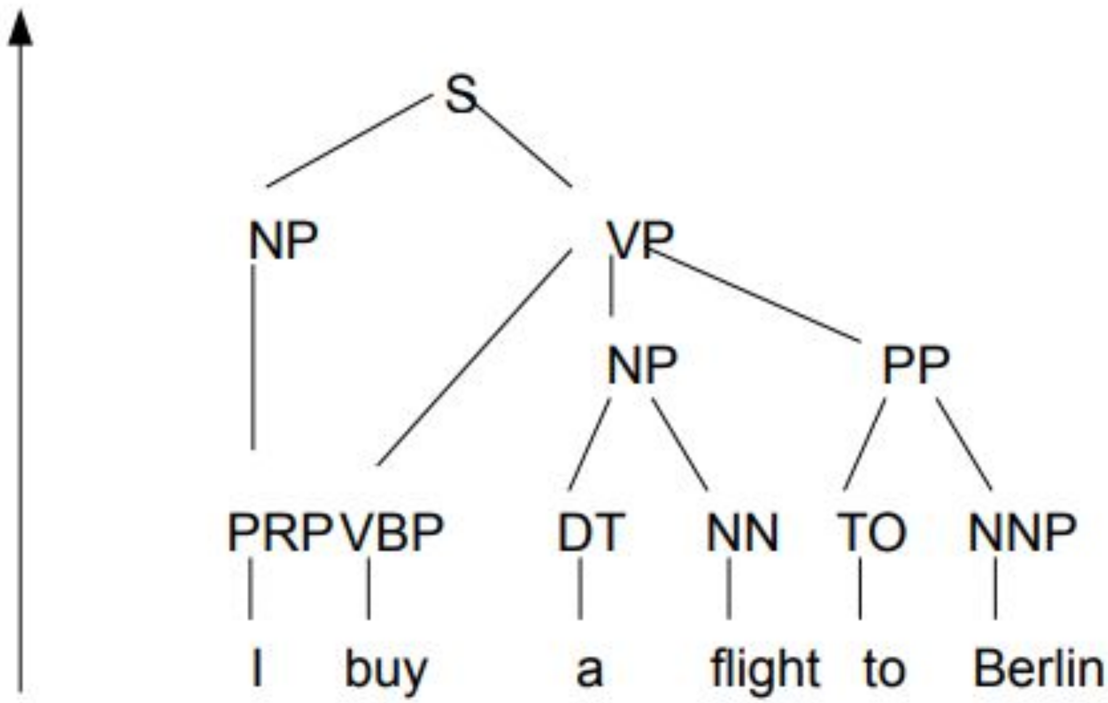
Parsing Algorithms

- Top-Down
  - Start with the rules that contains the S
  - Work on the way down to the words



Parsing Algorithms

- Bottom-Up
  - Start with trees that link up with the words
  - Work on the way up to larger and larger trees



Sr. No.	Key	Top Down Parsing	Bottom Up Parsing
1	Strategy	Top down approach starts evaluating the parse tree from the top and move downwards for parsing other nodes.	Bottom up approach starts evaluating the parse tree from the lowest level of the tree and move upwards for parsing the node.
2	Attempt	Top down parsing attempts to find the left most derivation for a given string.	Bottom up parsing attempts to reduce the input string to first symbol of the grammer.
3	Derivation Type	Top down parsing uses leftmost derivation.	Bottom up parsing uses the rightmost derivation.
4	Objective	Top down parsing searches for a production rule to be used to construct a string.	Bottom up parsing searches for a production rule to be used to reduce a string to get a starting symbol of grammer.

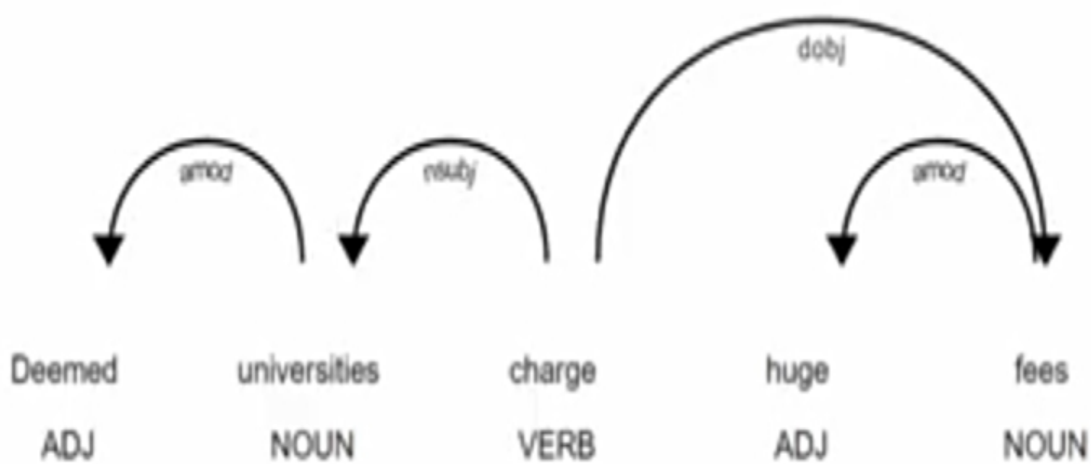
WHAT IS DEPENDENCY PARSING?

- The term Dependency Parsing (DP) refers to the process of examining the dependencies between the phrases of a sentence in order to determine its grammatical structure. The process assumes that there is a direct relationship between each linguistic unit in a sentence.
- The relationships between each linguistic unit, or phrase, in the sentence are expressed by directed arcs.

DEPENDENCE TAG

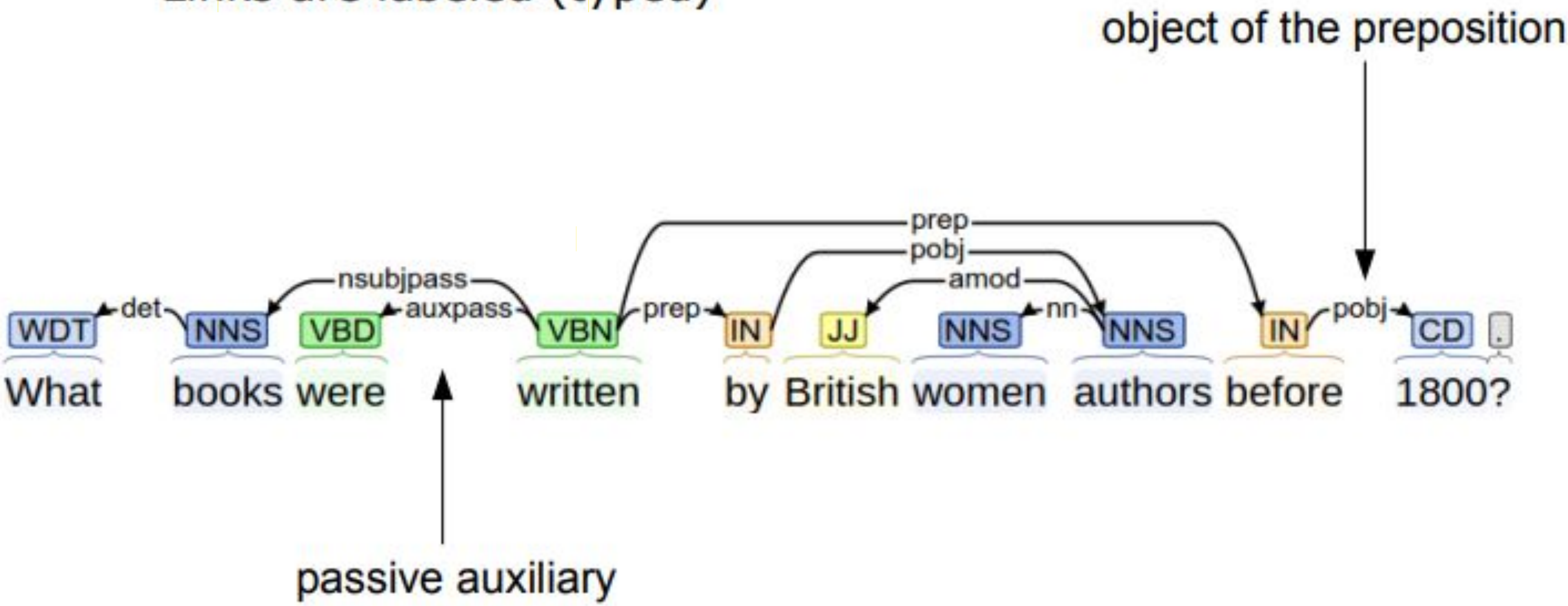
A dependence tag indicates the relationship between two phrases. For example, in the dependency graph below huge modifies the word fees.

Where the arrow starts represents the pinnacle and where it ends represents the dependent or the child. In this example fees is the pinnacle and huge is the child. The dependency between the two terms is represented by amod which stands for adjectival modifier.

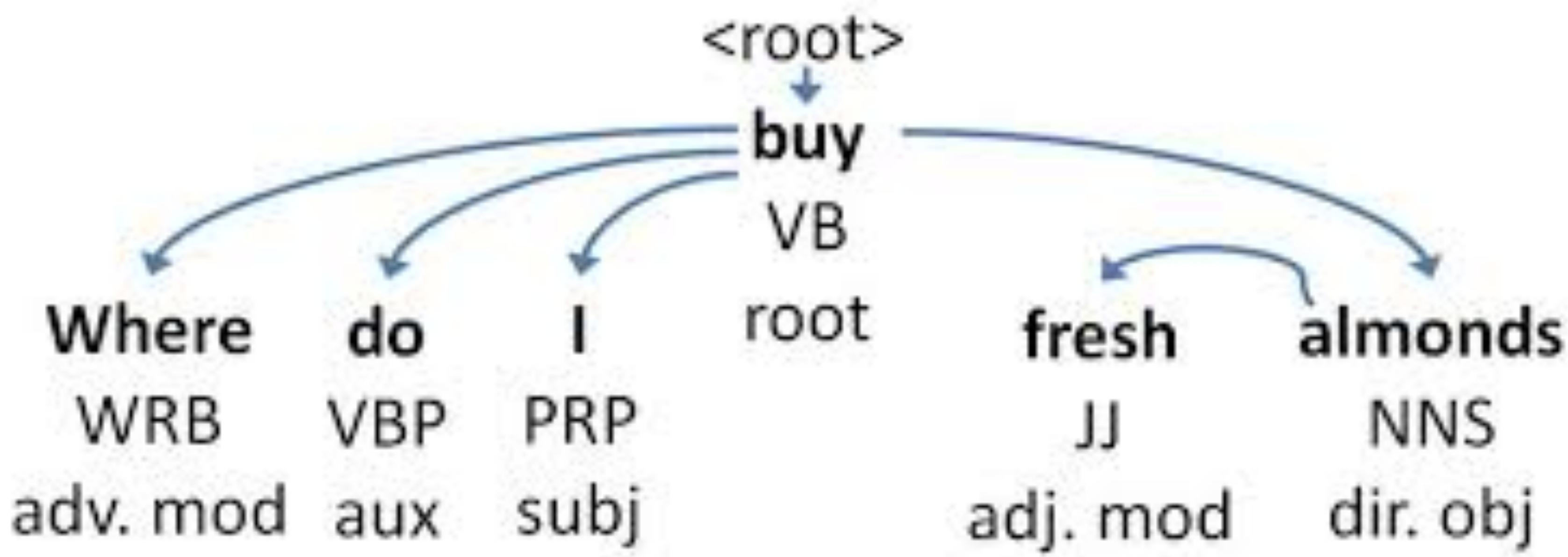


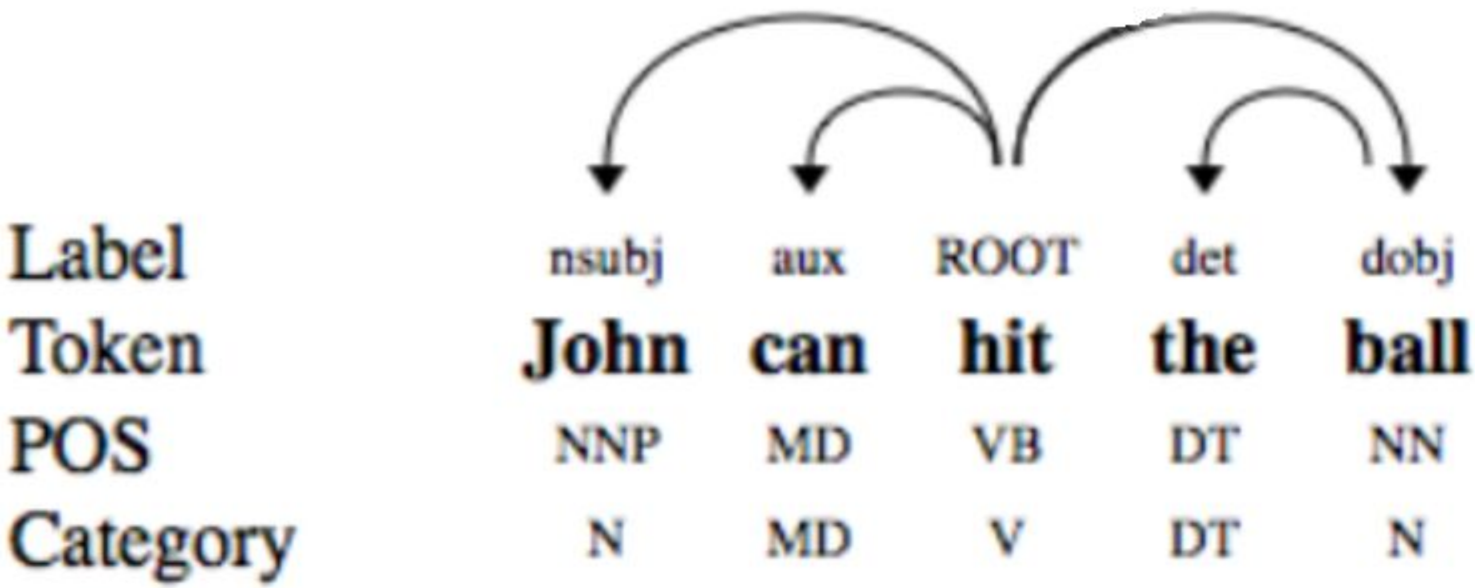
Dependency grammars

- No constituents, but typed dependencies
  - Links are labeled (typed)



Dependency Tag	Description	Dependency Tag	Description
acl	clausal modifier of a noun (adnominal clause)	compound	compound
acl:relcl	relative clause modifier	compound:lvc	gentle verb building
advcl	adverbial clause modifier	compound:prt	phrasal verb particle
advmod	adverbial modifier	compound:redup	reduplicated compounds
advmod:emph	emphasizing phrase, intensifier	compound:svc	serial verb compounds
advmod:lmod	locative adverbial modifier	conj	conjunct
amod	adjectival modifier	cop	copula
appos	appositional modifier	csubj	clausal topic
aux	auxiliary	csubj:move	clausal passive topic
aux:move	passive auxiliary	dep	unspecified dependency
case	case-marking	det	determiner
cc	coordinating conjunction	det:numgov	pronominal quantifier governing the case of the noun
cc:preconj	preconjunct	det:nummod	pronominal quantifier agreeing with the case of the noun
ccomp	clausal complement	det:poss	possessive determiner
clf	classifier	discourse	discourse ingredient

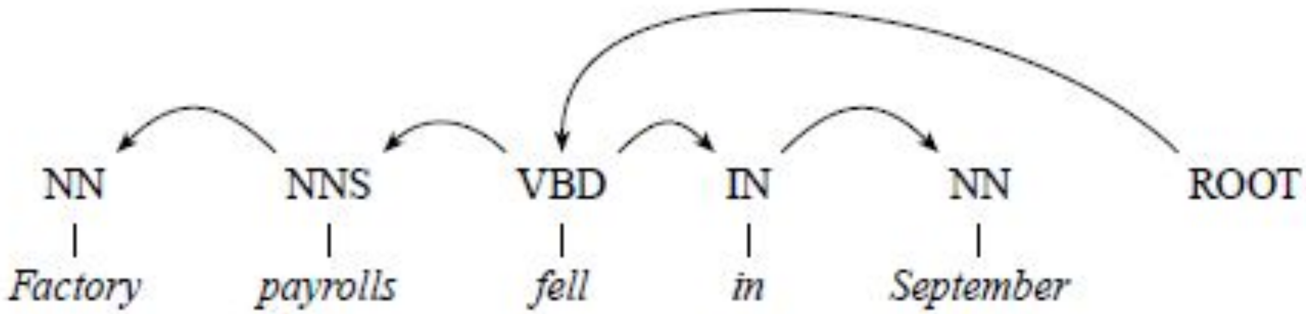

<root>
buy
VB
root
Where
WRB
adv. mod
do
VBP
aux
I
PRP
subj
fresh
JJ
adj. mod
almonds
NNS
dir. obj


Label
Token
POS
Category
nsubj aux ROOT det dobj
John can hit the ball
NNP MD VB DT NN
N MD V DT N

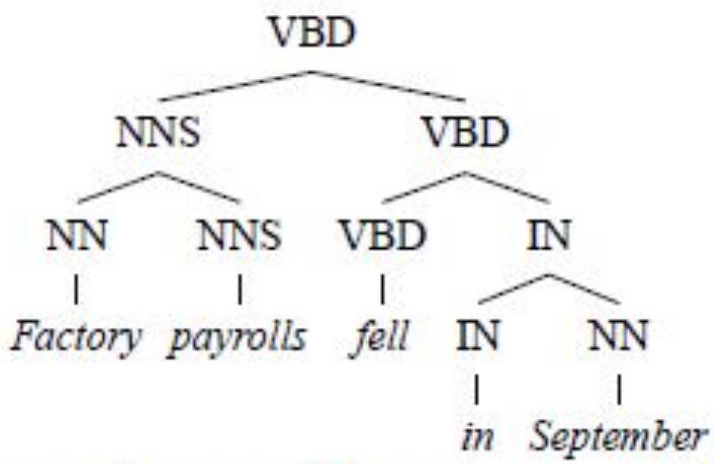
NOW TRY

Factory Payroll fell in September

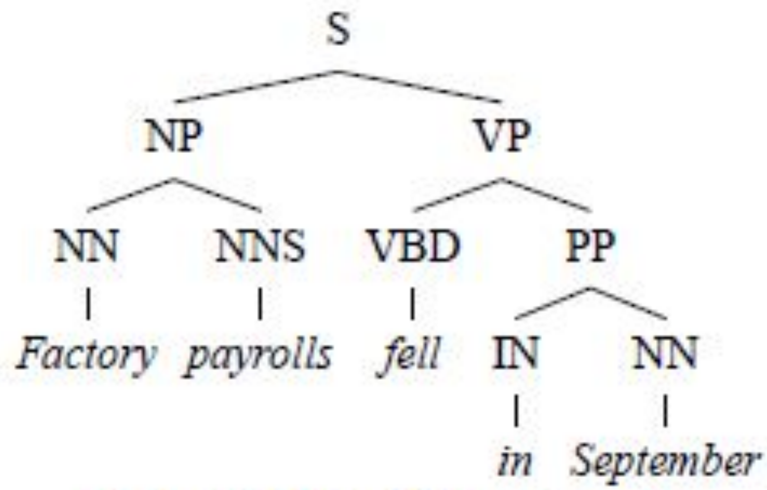
I prefer the morning flight through Denver



(a) Classical Dependency Structure

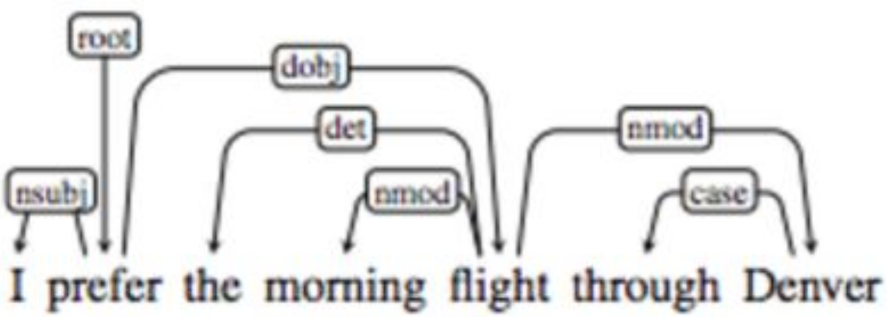


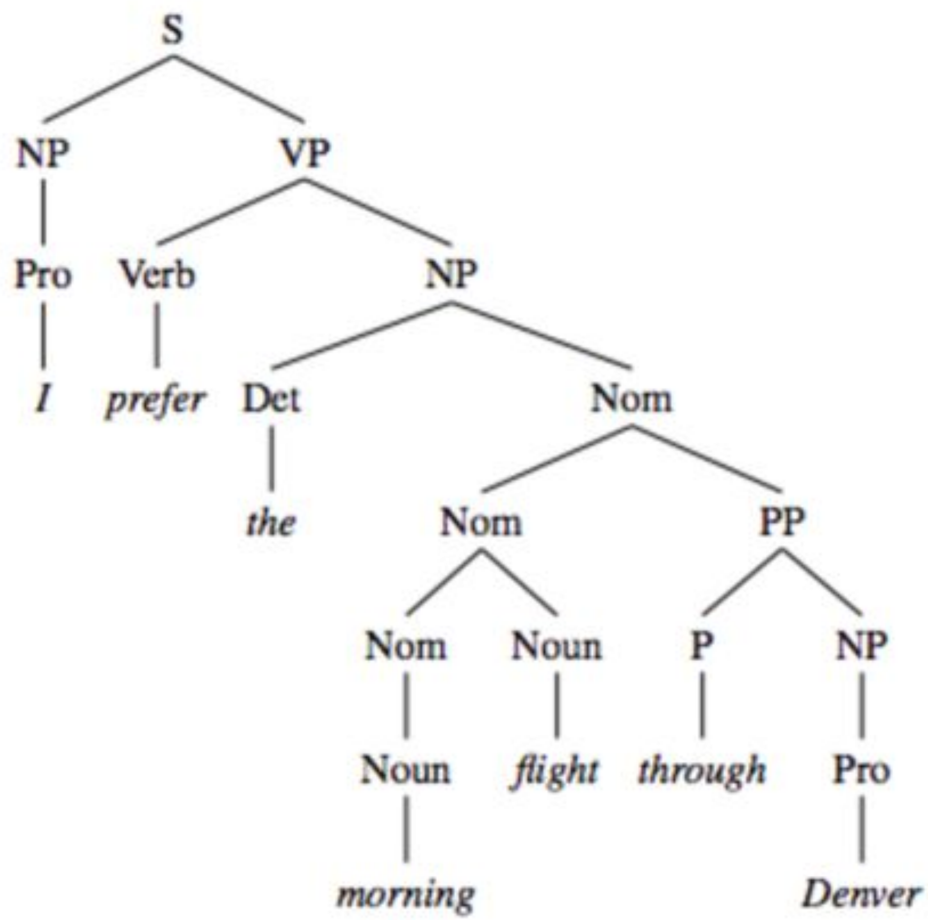
(b) Dependency Structure as CF Tree

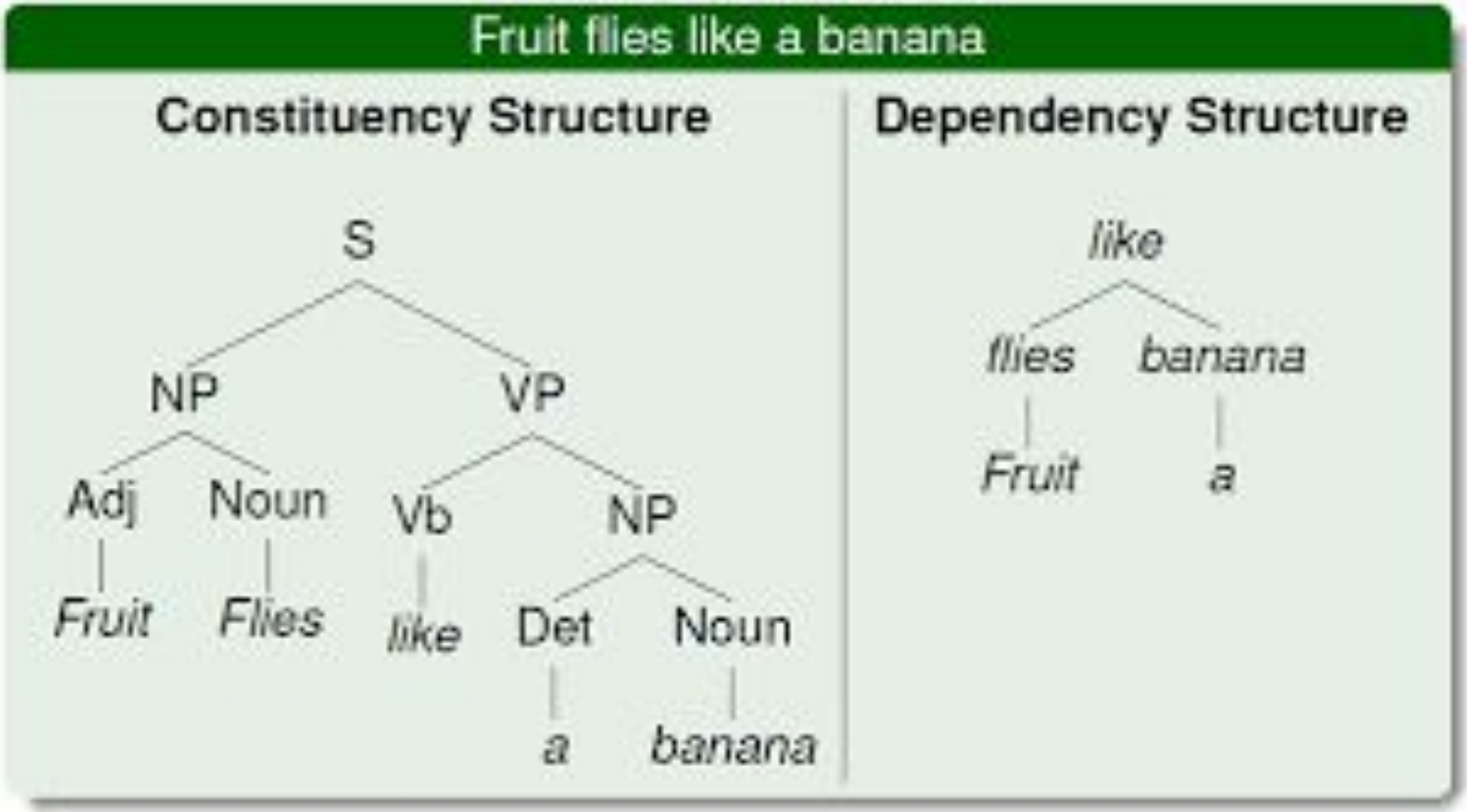


(c) CFG Structure

Figure : Three kinds of parse structures.


root
dobj
det
nmod
nsubj
nmod
case
I prefer the morning flight through Denver


S
NP
VP
Pro
Verb
NP
I
prefer
Det
Nom
the
Nom
PP
Nom
Noun
P
NP
Noun
flight
through
Pro
morning
Denver


Fruit flies like a banana
Constituency Structure
Dependency Structure
S
NP
VP
Adj
Noun
Vb
NP
Fruit
Flies
like
Det
Noun
a
banana
like
flies
banana
Fruit
a

Full Parsing

Goal: build a *complete parse tree* for a sentence.

- Problems with full parsing:
  - Low accuracy
  - Slow
  - Domain Specific
- These problems are relevant for both symbolic and statistical parsers

Light Parsing

Goal: assign a *partial structure* to a sentence.

- Simpler solution space
- Local context
- Non-recursive
- Restricted (local) domain

Chunk Parsing

Goal: divide a sentence into a sequence of chunks.

- Chunks are non-overlapping regions of a text

  [I] saw [a tall man] in [the park]

- Chunks are non-recursive
  - A chunk can not contain other chunks
- Chunks are non-exhaustive
  - Not all words are included in the chunks

Chunk Parsing Examples

- Noun-phrase chunking:

  [I] saw [a tall man] in [the park].
- Verb-phrase chunking:

  The man who [was in the park] [saw me].

Chunks and Constituency

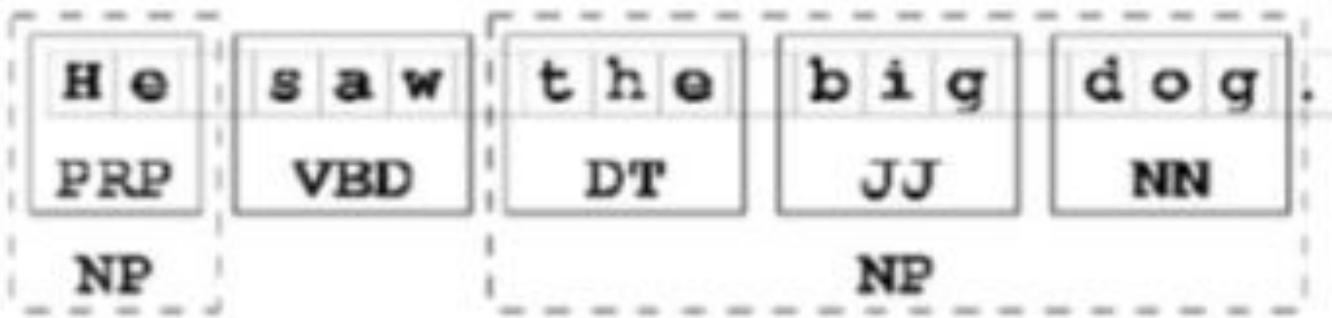
Constituants: [a tall man in [the park]].

Chunks: [a tall man] in [the park].

- Chunks are *not* constituants
  - Constituants are recursive
- Chunks are typically *subsequences* of constituants.
  - Chunks do not cross constituant boundaries

Segmenting vs. Labeling

- Tokenization *segments* the text
- Tagging *labels* the text
- Shallow parsing does both simultaneously.



[$_{NP}$ He] saw [$_{NP}$ the big dog]

Chunk Parsing: Accuracy

Chunk parsing achieves higher accuracy

- Smaller solution space
- Less word-order flexibility *within* chunks than *between* chunks
  - Fewer long-range dependencies
  - Less context dependence
- Better locality
- No need to resolve ambiguity
- Less error propagation

Dynamic Programming Parsing

To avoid extensive repeated work you must cache intermediate results, specifically found constituents

Caching (memoizing) is critical to obtaining a polynomial time parsing algorithm for CFGs

Dynamic programming algorithms based on both top-down and bottom-up search can achieve $O(n^3)$ recognition time where n is the length of the input string.

Dynamic Programming Parsing Methods

CKY (Cocke-Kasami-Younger) algorithm based on bottom-up parsing and requires first normalizing the grammar.

Earley parser is based on top-down parsing and does not require normalizing grammar but is more complex.

Obtaining the best parse

- Call the best parse T(S), where S is your sentence
  - Get the tree which has the highest probability, i.e.
  - $T(S) = \text{argmax}_{T \epsilon \text{parse-trees}(S)} \; P(T)$
- Can use the Cocke-Younger-Kasami (CYK) algorithm to calculate best parse
  - CYK is a form of dynamic programming
  - CYK is a chart parser, like the Earley parser

The CYK algorithm

- Base case
  - Add words to the chart
  - Store P(A → w_i) for every category A in the chart
- Recursive case → makes this dynamic programming because we only calculate B and C once
  - Rules must be of the form A → BC, i.e., exactly two items on the RHS (we call this Chomsky Normal Form (CNF))
  - Get the probability for A at this node by multiplying the probabilities for B and for C by P(A → BC)
    - P(B)*P(C)*P(A → BC)
- For a given A, only keep the maximum probability (again, this is dynamic programming)

```
Let w = w1 w2 w3 wi ...wj...wn
    and wij = wi ...wi+j-1
// Initialization step
for i := 1 to n do
    for all rules A→ wi do
        chart [i,1] = {A}
// Recursive step
for j = 2 to n do
    for i = 1 to n−j+1 do
    begin
        chart [i, j] = ϕ
        for k = 1 to j −1 do
        chart [i, j] := chart[i, j] ∪{A | A → BC is a production and
                            B ∈ chart[i, k] and  C ∈ chart [i+k, j−k]}
    end
if S ∈ chart[1, n] then accept else reject
```

Figure 4.12 The CYK algorithm

CKY Algorithm

function CKY-PARSE(*words, grammar*) **returns** *table*

for $j \leftarrow$ **from** 1 **to** LENGTH(*words*) **do**
  $table[j-1, j] \leftarrow \{A \mid A \rightarrow words[j] \in gram$
  for $i \leftarrow$ **from** $j-2$ **downto** 0 **do**
    for $k \leftarrow i+1$ **to** $j-1$ **do**
      $table[i,j] \leftarrow table[i,j] \cup$
$$\{A \mid A \rightarrow BC \in grammar,$$
$$B \in table[i,k],$$
$$C \in table[k,j]\}$$

Looping over the columns

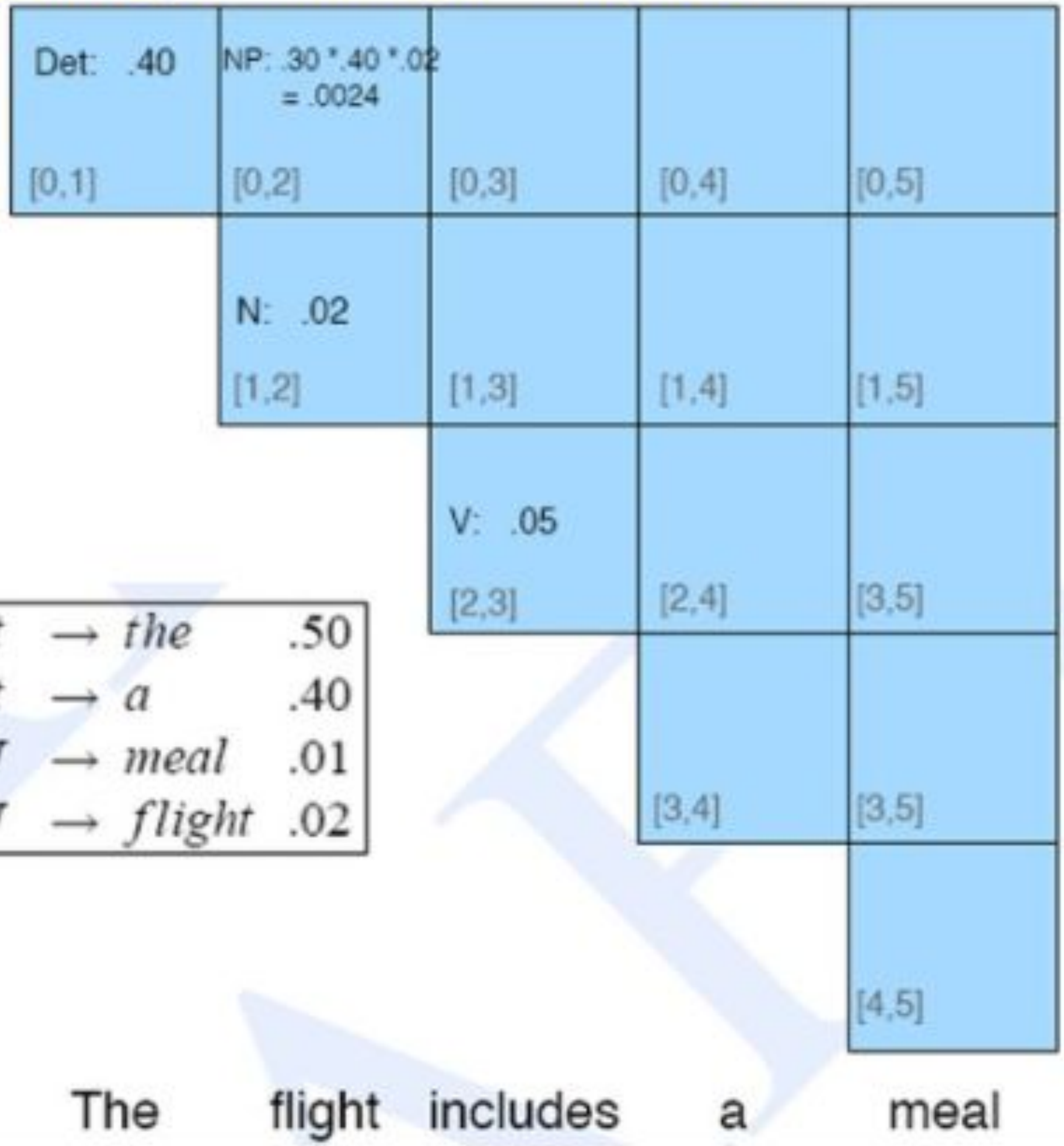
Filling the bottom cell

Filling row i in column j

Looping over the possible split locations between i and j.

Check the grammar for rules that link the constituents in [i,k] with those in [k,j]. For each rule found store the LHS of the rule in cell [i,j].

Example: The flight includes a meal



S	$\rightarrow$	$NP\ VP$	.80	Det	$\rightarrow$	*the*	.50
NP	$\rightarrow$	$Det\ N$	.30	Det	$\rightarrow$	*a*	.40
VP	$\rightarrow$	$V\ NP$	.20	N	$\rightarrow$	*meal*	.01
V	$\rightarrow$	*includes*	.05	N	$\rightarrow$	*flight*	.02

Probabilistic Context-Free Grammars (PCFGs)

- Definition of a CFG:
  - Set of non-terminals (N)
  - Set of terminals (T)
  - Set of rules/productions (P), of the form $A \rightarrow \beta$
  - Designated start symbol (S)
- Definition of a PCFG:
  - Same as a CFG, but with one more function, D
  - D assigns probabilities to each rule in P

Probabilities

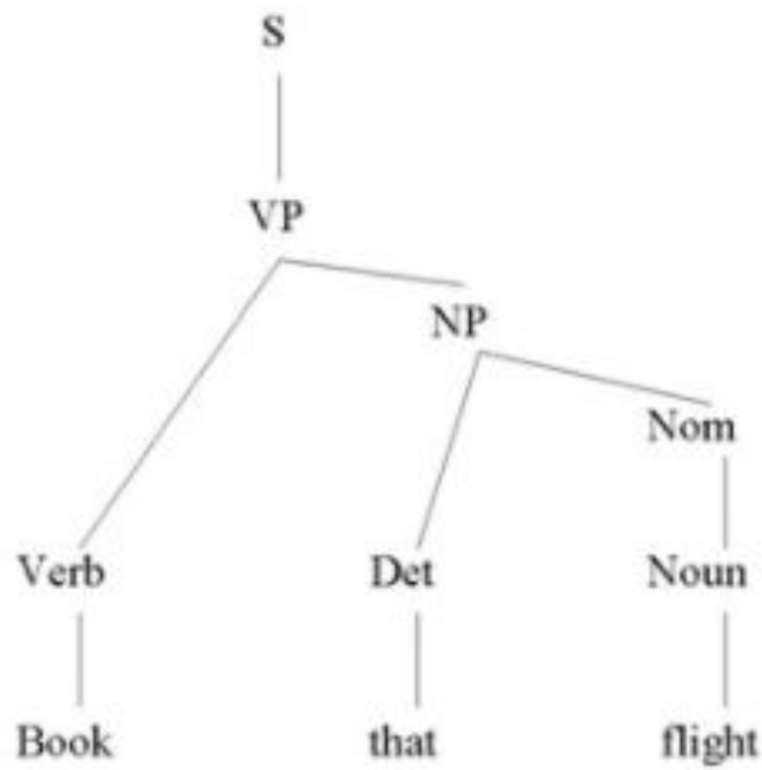
- The function D gives probabilities for a non-terminal A to be expanded to a sequence β.
  - Written as $P(A \rightarrow \beta)$
  - or as $P(A \rightarrow \beta | A)$
- The idea is that, given A as the mother non-terminal (LHS), what is the likelihood that β is the correct RHS
  - Note that $\Sigma_i (A \rightarrow \beta_i \mid A) = 1$
- For example, we would augment a CFG with these probabilities:
  - $P(S \rightarrow NP\ VP \mid S) = .80$
  - $P(S \rightarrow Aux\ NP\ VP \mid S) = .15$
  - $P(S \rightarrow VP \mid S) = .05$

An Example

$S \rightarrow NP\ VP$ [.80]	$Det \rightarrow$ *that* [.10] | *a* [.30] | *the* [.60]
$S \rightarrow Aux\ NP\ VP$ [.15]	$Noun \rightarrow$ *book* [.10] | *flight* [.30]
$S \rightarrow VP$ [.05]	| *meal* [.15] | *money* [.05]
$NP \rightarrow Pronoun$ [.35]	| *flights* [.40] | *dinner* [.10]
$NP \rightarrow Proper\text{-}Noun$ [.30]	$Verb \rightarrow$ *book* [.30] | *include* [.30]
$NP \rightarrow Det\ Nominal$ [.20]	| *prefer;* [.40]
$NP \rightarrow Nominal$ [.15]	$Pronoun \rightarrow$ *I* [.40] | *she* [.05]
$Nominal \rightarrow Noun$ [.75]	| *me* [.15] | *you* [.40]
$Nominal \rightarrow Nominal\ Noun$ [.20]	$Proper\text{-}Noun \rightarrow$ *Houston* [.60]
$Nominal \rightarrow Nominal\ PP$ [.05]	| *TWA* [.40]
$VP \rightarrow Verb$ [.35]	$Aux \rightarrow$ *does* [.60] | *can* [40]
$VP \rightarrow Verb\ NP$ [.20]	$Preposition \rightarrow$ *from* [.30] | *to* [.30]
$VP \rightarrow Verb\ NP\ PP$ [.10]	| *on* [.20] | *near* [.15]
$VP \rightarrow Verb\ PP$ [.15]	| *through* [.05]
$VP \rightarrow Verb\ NP\ NP$ [.05]	
$VP \rightarrow VP\ PP$ [.15]	
$PP \rightarrow Preposition\ NP$ [1.0]	

Using Probabilities to Parse

- P(T): Probability of a particular parse tree
- $P(T,S) = \Pi_{n \epsilon T}\, p(\, r(n)\,) = P(T).P(S|T)$

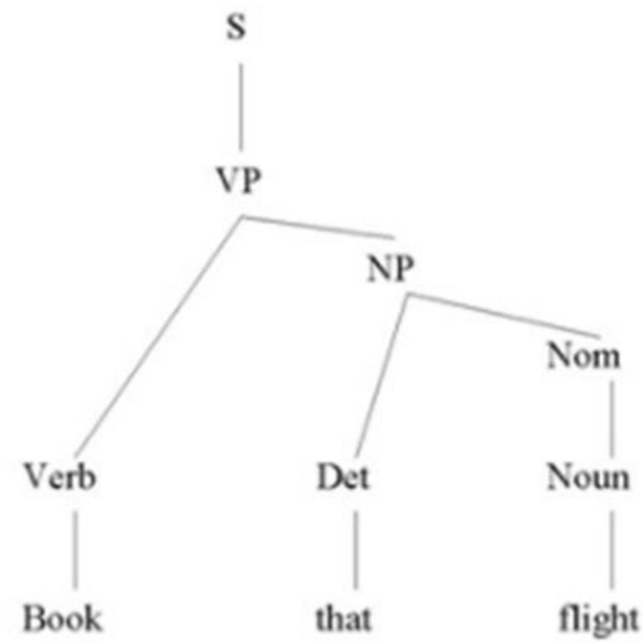


- $P(T) = \Pi_{n \epsilon T}\, p(\, r(n)\,)$

i.e., the product of the probabilities of all the rules r used to expand each node n in the parse tree

Computing probabilities

- We have the following rules and probabilities
  - S → VP .05
  - VP → V NP .40
  - NP → Det N .20
  - V → book .30
  - Det → that .05
  - N → flight .25



- P(T) = P(S→VP)*P(VP→V NP)*…*P(N→flight)

  $= .05 * .40 * .20 * .30 * .05 * .25 = .000015$, or 1.5×10^{-5}

Rules			P
S	→	VP	.05
VP	→	Verb NP	.20
NP	→	Det Nominal	.20
Nominal	→	Nominal Noun	.20
Nominal	→	Noun	.75
Verb	→	book	.30
Det	→	the	.60
Noun	→	dinner	.10
Noun	→	flights	.40

Rules			P
S	→	VP	.05
VP	→	Verb NP NP	.10
NP	→	Det Nominal	.20
NP	→	Nominal	.15
Nominal	→	Noun	.75
Nominal	→	Noun	.75
Verb	→	book	.30
Det	→	the	.60
Noun	→	dinner	.10
Noun	→	flights	.40

Figure 14.2 Two parse trees for an ambiguous sentence. The transitive parse (a) corresponds to the sensible meaning "Book flights that serve dinner", while the ditransitive parse (b) to the nonsensical meaning "Book flights on behalf of 'the dinner'".

$$P(T_{left}) = .05 * .20 * .20 * .20 * .75 * .30 * .60 * .10 * .40 = \mathbf{2.2 \times 10^{-6}}$$
$$P(T_{right}) = .05 * .10 * .20 * .15 * .75 * .75 * .30 * .60 * .10 * .40 = \mathbf{6.1 \times 10^{-7}}$$

Initialization:

for $i := 1$ to n do

for all rules $A \rightarrow w_i$ do

$\varphi[i,1,A] = P(A \rightarrow w_i)$

Recursive Step:

for $j = 2$ to n do

for $i = 1$ to $n-j+1$ do

begin

$\varphi[i,1,A] = \phi$

for $k = 1$ to $j-1$ do

$\varphi'[i,j,A] = \max_k \varphi'[i,k,B] \times \varphi'[k,j,C] \times P(A \rightarrow BC),$

such that $A \rightarrow BC$ is a production rule in grammar

$BP[i,j,A] = \{ k, A, B\}$

end

Figure 4.15 Probabilistic CYK algorithm

Problems with PCFGs

- It's still only a CFG, so dependencies on non-CFG info not captured
  - e.g., Pronouns are more likely to be subjects than objects:
  - $P[(NP \rightarrow Pronoun) \mid NP=subj] >> P[(NP \rightarrow Pronoun) \mid NP=obj]$

	Pronoun	Non-Pronoun
Subject	91%	9%
Object	34%	66%

Problems with PCFGs

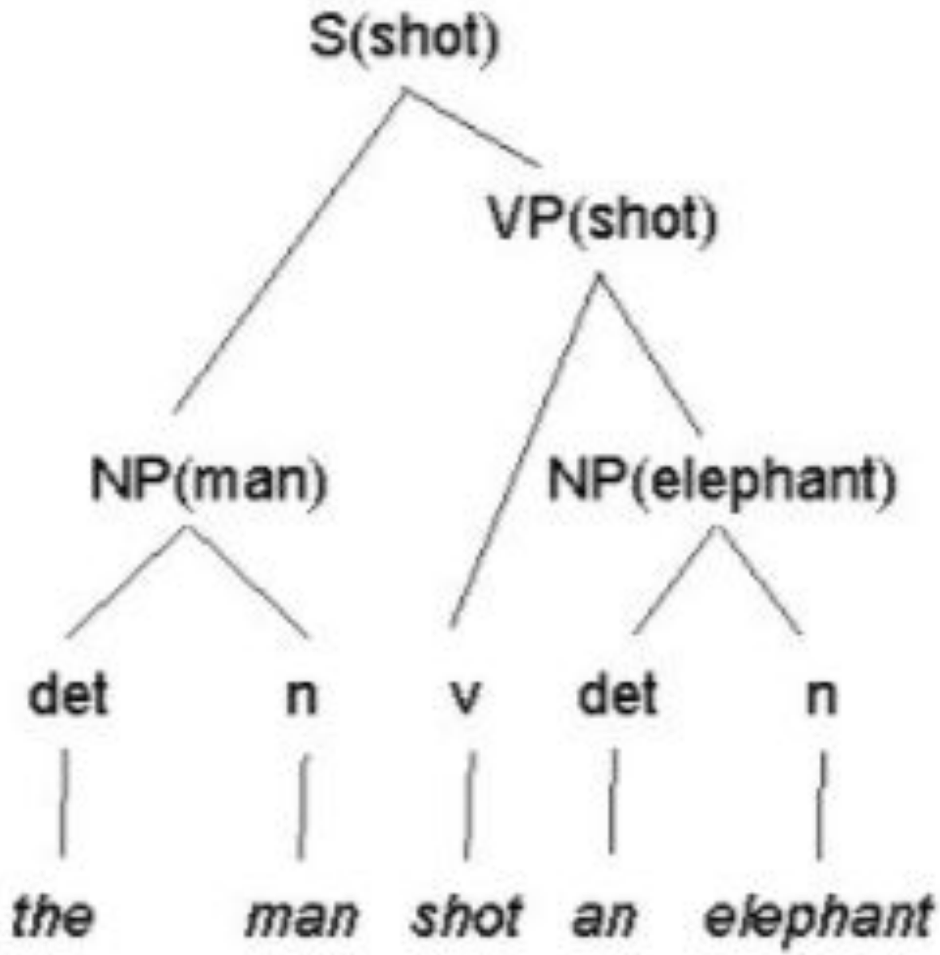
- Ignores lexical information (statistics), which is usually crucial for disambiguation
  - (T1) America sent [[250,000 soldiers] [into Iraq]]
  - (T2) America sent [250,000 soldiers] [into Iraq]
    - send with into-PP always attached high (T2) in PTB!
- To handle lexical information, we'll turn to lexicalized PCFGs

Probabilistic Lexicalized CFGs

- Key notion: "head"
  - Each non-terminal assoc w/lexical head
    - E.g. verb with verb phrase, noun with noun phrase
  - Each rule must identify RHS element as head
    - Heads propagate up tree
  - Conceptually like adding 1 rule per head value
    - VP(dumped) -> VBD(dumped)NP(sacks)PP(into)
    - VP(dumped) -> VBD(dumped)NP(cats)PP(into)

Probabilistic Lexicalised CFGs

- One solution is to identify in each rule that one of the elements on the RHS (daughter) is more important: the “head”
  - This is quite intuitive, e.g. the n in an NP rule, though often controversial (from linguistic point of view)
- Head must be a lexical item
- Head value is percolated up the parse tree
- Added advantage is that PS tree has the feel of a dependency tree


S(shot)
VP(shot)
NP(man)
NP(elephant)
det
n
v
det
n
the
man
shot
an
elephant


shot
man
elephant
the
an

Feature Structure

- Feature structure – feature value pairs
- Features – atomic symbols drawn from some finite set
- Values – Atomic symbols or feature structures themselves
- Represented in terms of attribute-value matrix

AVM:

$$\begin{bmatrix} \text{FEATURE}_1 & value_1 \\ \text{FEATURE}_2 & value_2 \\ \vdots & \\ \text{FEATURE}_n & value_n \end{bmatrix}$$

To make this concrete, consider the

$$\begin{bmatrix} \text{CAT} & NP \\ \text{NUMBER} & sg \\ \text{PERSON} & 3rd \end{bmatrix}$$

This structure can be used to repre

Represents 3rd person singular NP

Feature Structures, Grammar, Parsing

Feature Structures

- describe additional syntactic-semantic information, like category, person, number, e.g. goes ≡ <verb, 3^{rd}, singular>
- specify feature structure constraints (agreements) as part of the grammar rules
- during parsing, check agreements of feature structures (unification)

example

S → NP VP **<NP number> = <VP number>**

S → NP VP **<NP agreement> = <VP agreement>**

For Detailed Explanation on Feature structures, Unification of feature structures watch below video.

https://www.youtube.com/watch?v=wHDCvbZ3zDI