Phân tích cú pháp (Parsing)

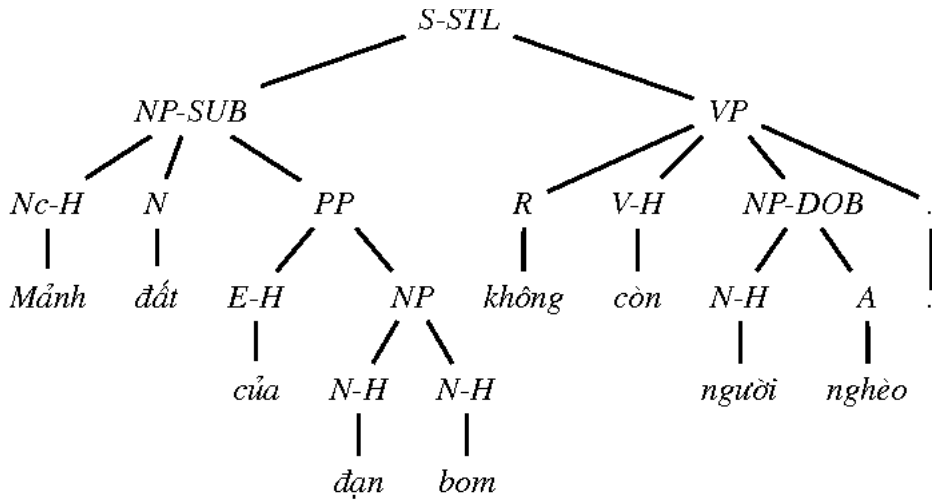
TDTU

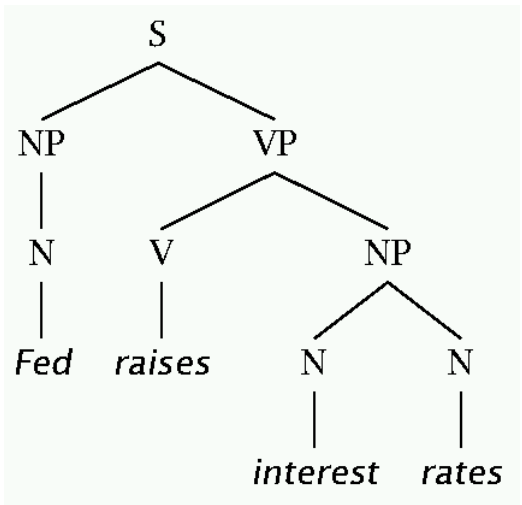
Nội dung

1. Phân tích cú pháp là gì
2. Tiếp cận phân tích cú pháp thành phần và phụ thuộc
3. Ý nghĩa và ứng dụng của phân tích cú pháp
4. Văn phạm CFG và phân tích cú pháp
5. Văn phạm PCFG
6. Cấu trúc phụ thuộc và phân tích

Phân tích cú pháp là gì?

- Mục tiêu của phân tích cú pháp là xác định thông tin về cấu trúc của một câu, tức là xác định mối quan hệ giữa các thành phần trong câu.
- Structure of a sentence, it means the syntactic information.





Hai tiếp cận trong phân tích cú pháp

- Phân tích cú pháp thành phần (cú pháp cấu trúc cụm): Constituency parsing = Phrase structure grammar
- Phân tích cú pháp phụ thuộc: Dependency parsing

Phân tích thành phần (cấu trúc cụm)
Constituency = grammar phrase structure

- **Starting unit: words**

  the, cat, cuddly, by, door
- **Words combine into phrases**

  the cuddly cat, by the door
- **Phrases can combine into bigger phrases**
  - the cuddly cat by the door

Phân tích thành phần (cấu trúc cụm)
Constituency = grammar phrase structure

Phrase structure organizes words into nested constituents

Can represent the grammar with CFG rules

Starting unit: words are given a category (part of speech = pos)

the, cat, cuddly, by, door
Det N Adj P N

Words combine into phrases with categories

the cuddly cat, by the door
NP → Det Adj N PP → P NP

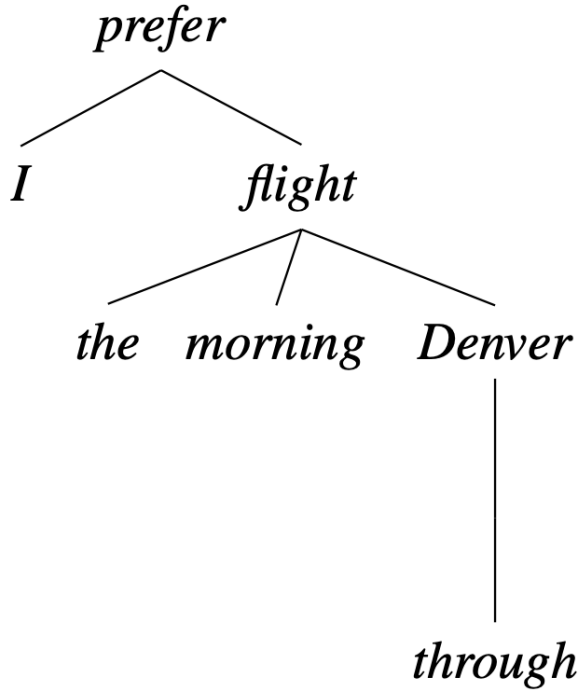
Phrases can combine into bigger phrases recursively

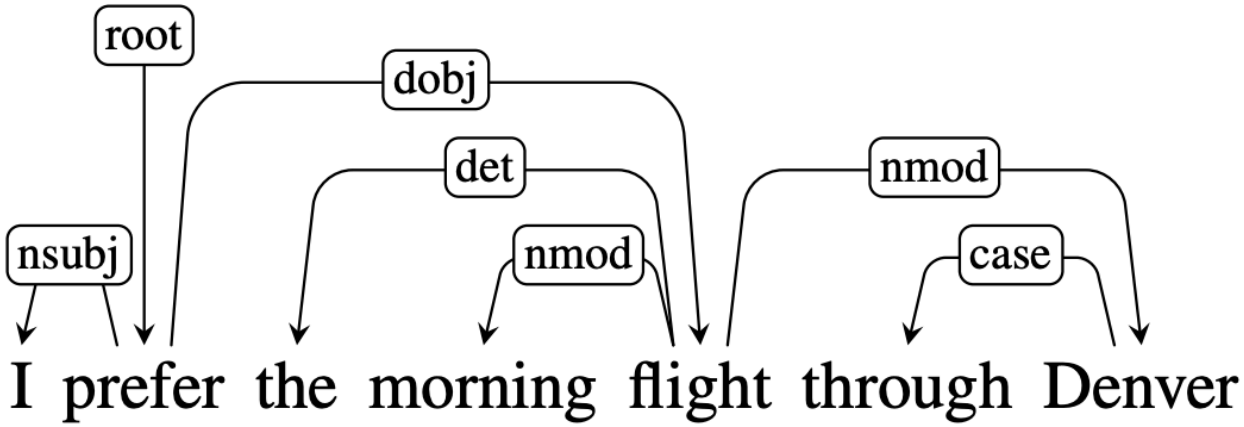
the cuddly cat by the door

NP → NP PP

Cú pháp phụ thuộc và cấu trúc phụ thuộc (dependency grammar and dependency structure)

- Dependency syntax postulates that syntactic structure consists of relations between lexical items, normally binary asymmetric relations ("arrows") called dependencies





Relation tags

Clausal Argument Relations	Description
NSUBJ	Nominal subject
DOBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal complement
XCOMP	Open clausal complement
Nominal Modifier Relations	**Description**
NMOD	Nominal modifier
AMOD	Adjectival modifier
NUMMOD	Numeric modifier
APPOS	Appositional modifier
DET	Determiner
CASE	Prepositions, postpositions and other case markers
Other Notable Relations	**Description**
CONJ	Conjunct
CC	Coordinating conjunction

Figure 15.2 Selected dependency relations from the Universal Dependency set. (de Marneffe et al., 2014)

Ý nghĩa và ứng dụng của PTCP

- Lấy thông tin cú pháp và ứng dụng cho bài toán hiểu ngôn ngữ
- Cấu trúc tiền đề cho phân tích ngữ nghĩa
- Ứng dụng:
  - ?
  - ?
  - ?
  - ?

Ý nghĩa và ứng dụng của PTCP

- Lấy thông tin cú pháp và ứng dụng cho bài toán hiểu ngôn ngữ
- Cấu trúc tiền đề cho phân tích ngữ nghĩa
- Ứng dụng:
  - Phân tích câu hỏi trong Question Answering
  - Dịch máy trong tiếp cận dựa trên Rule
  - Các ứng dụng trích chọn thông tin (information extraction)
  - Kiểm lỗi ngữ pháp (grammar checker)

Thách thức trong phân tích cú pháp (Challenge in parsing)

- Đối với ngôn ngữ tự nhiên, phân tích cú pháp có tính nhập nhằng rất cao.
- Ví dụ:

I saw a man on a hill with a telescope.

Thách thức trong phân tích cú pháp (Challenge in parsing)

- Đối với ngôn ngữ tự nhiên, phân tích cú pháp có tính nhập nhằng rất cao.
- Ví dụ:

I	saw	a	man	on	a	hill	with	a	telescope
N	V	Det	N	In	Det	N	In	Det	N

1. There's a man on a hill, and I'm watching him with my telescope.
2. There's a man on a hill, who I'm seeing, and he has a telescope.
3. There's a man, and he's on a hill that also has a telescope on it.
4. I'm on a hill, and I saw a man using a telescope.
5. There's a man on a hill, and I'm sawing him with a telescope.

Parsing

Context Free Grammar and Syntactic Parsing

CFG and Constituency Parsing

- Văn phạm phi ngữ (CFG) cảnh biểu diễn cấu trúc cụm
- Cây cú pháp và luật
- Thuật toán phân tích CKY
- Thuật toán phân tích Earley

A Phrase Structure Grammar

$S \rightarrow NP\ VP$

$VP \rightarrow V\ NP$

$VP \rightarrow V\ NP\ PP$

$NP \rightarrow NP\ NP$

$NP \rightarrow NP\ PP$

$NP \rightarrow N$

$NP \rightarrow e$

$PP \rightarrow P\ NP$

$N \rightarrow$ people

$N \rightarrow$ fish

$N \rightarrow$ tanks

$N \rightarrow$ rods

$V \rightarrow$ people

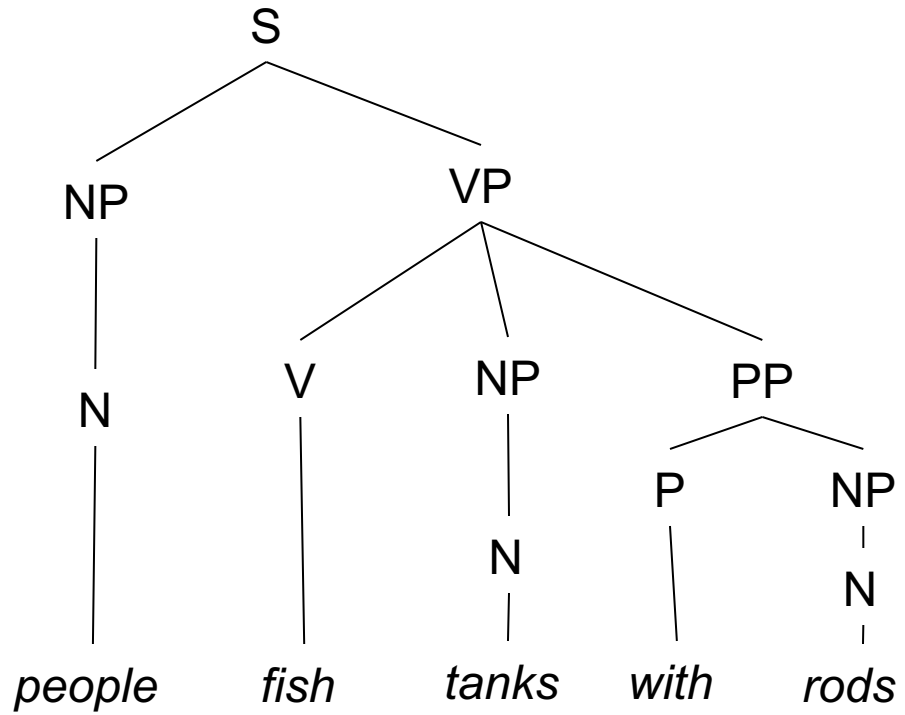
$V \rightarrow$ fish

$V \rightarrow$ tanks

$P \rightarrow$ with

Sentence: *“people fish with rods”*

Phân tích cú pháp



$S \rightarrow NP\ VP$

$VP \rightarrow V\ NP$

$VP \rightarrow V\ NP\ PP$

$NP \rightarrow NP\ NP$

$NP \rightarrow NP\ PP$

$NP \rightarrow N$

$NP \rightarrow e$

$PP \rightarrow P\ NP$

$N \rightarrow$ people

$N \rightarrow$ fish

$N \rightarrow$ tanks

$N \rightarrow$ rods

$V \rightarrow$ people

$V \rightarrow$ fish

$V \rightarrow$ tanks

$P \rightarrow$ with

Phrase structure grammars = context-free grammars (CFGs)

- G = (T, N, S, R)
  - T is a set of terminal symbols
  - N is a set of nonterminal symbols
  - S is a start symbol (S ∈ N)
  - R is a set of rules/productions of the form $X \rightarrow \gamma$
    - $X \in N$ and $\gamma \in (N \cup T)^*$
- A grammar G generates a language L.

Phân tích cú pháp

- Là bài toán với văn phạm đã cho và một câu input, sinh cây cú pháp (parsed tree).
- Các vấn đề cần giải quyết:
  - Xây dựng văn phạm.
  - Thuật toán phân tích (time complexity)
  - Giải quyết tính nhập nhằng trong phân tích cú pháp.

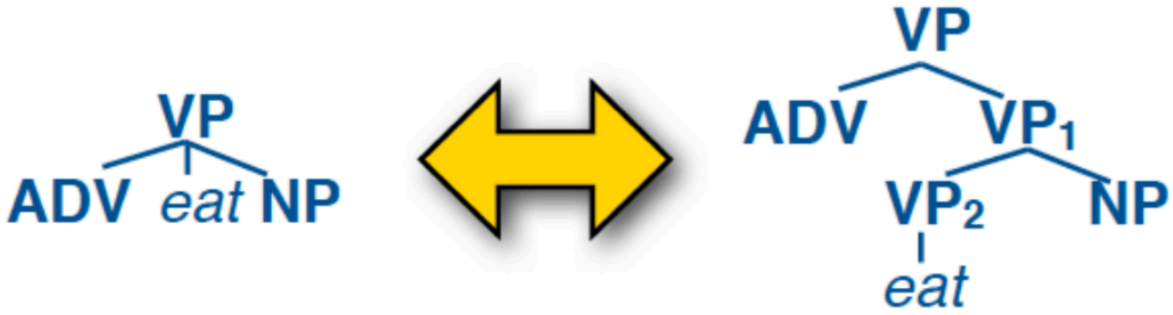
Các tiếp cận phân tích cú pháp

- Top-down vs. bottom-up:
  - Top-down: (goal-driven): from the start symbol down.
  - Bottom-up: (data-driven): from the symbols up.
- Naive vs. dynamic programming:
  - Naive:enumerate everything.
  - Backtracking: try something, discard partial solutions.
  - Dynamic programming: save partial solutions in a table.
- Examples:
  - CKY: bottom-up dynamic programming.
    Earley parsing:top-down dynamic programming.

Thuật toán CYK và văn phạm chuẩn Chomsky (Chomsky Normal Form)

- Luật trong CNF chỉ thuộc một trong 2 dạng:
  - Two non-terminals: (e.g.,VP→ADVVP)
  - One terminal: VP → eat
- Mỗi luật CFG có thể viết về dạng tương đương của luật CNF

Ví dụ:

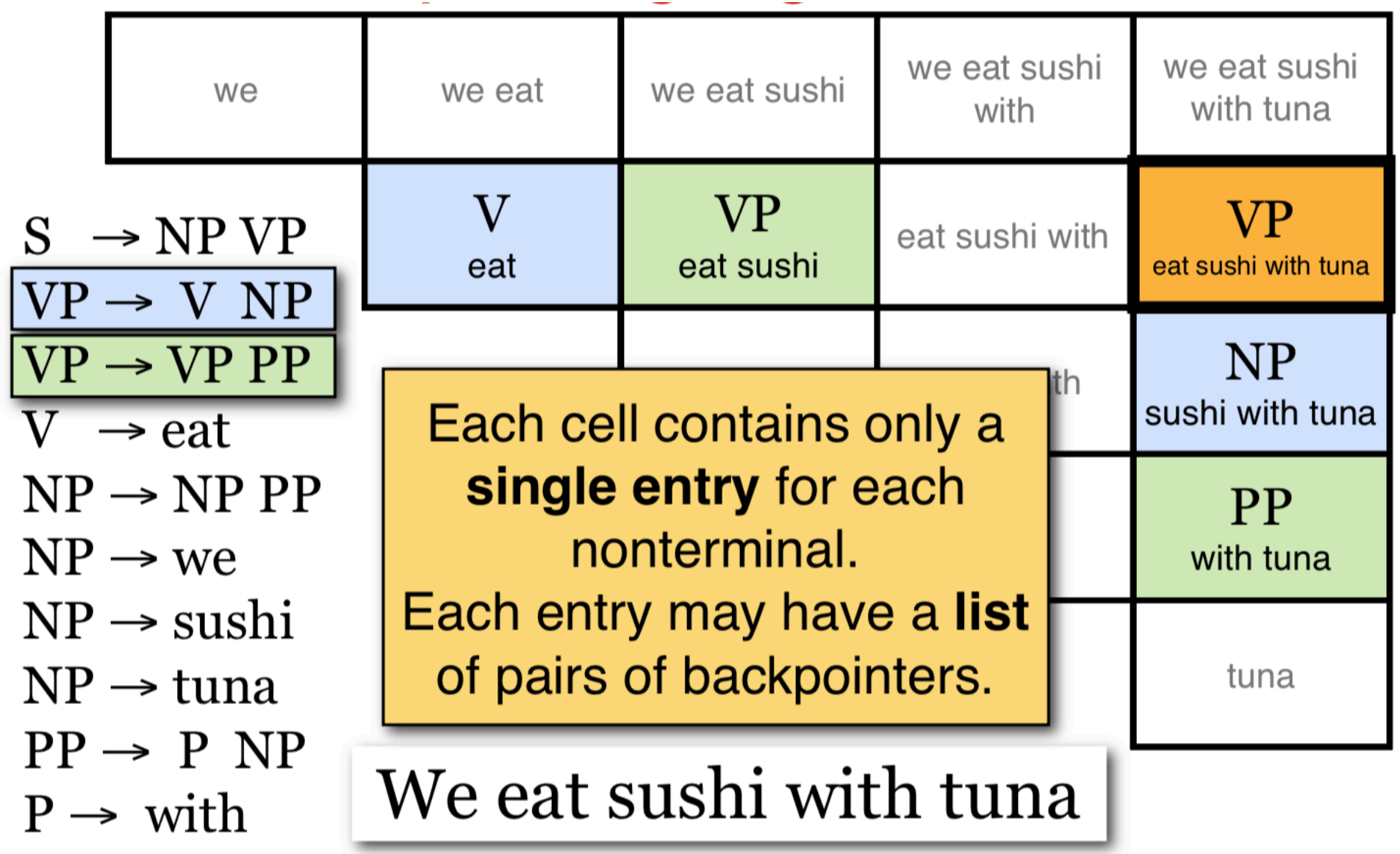


Chuyển về dạng CNF?: VP → VBD NP PP PP

CKY chart parsing algorithm

- Bottom-up parsing: start with the words
- Dynamic programming:
  - save the results in a table/chart
  - re-use these results in finding larger constituents
- Complexity: $O(n^3|G|)$
  n: length of string, $|G|$: size of grammar)

Ví dụ


we
we eat
we eat sushi
we eat sushi with
we eat sushi with tuna
V
eat
VP
eat sushi
eat sushi with
VP
eat sushi with tuna
NP
sushi with tuna
PP
with tuna
tuna
S → NP VP
VP → V NP
VP → VP PP
V → eat
NP → NP PP
NP → we
NP → sushi
NP → tuna
PP → P NP
P → with
Each cell contains only a **single entry** for each nonterminal.
Each entry may have a **list** of pairs of backpointers.
We eat sushi with tuna

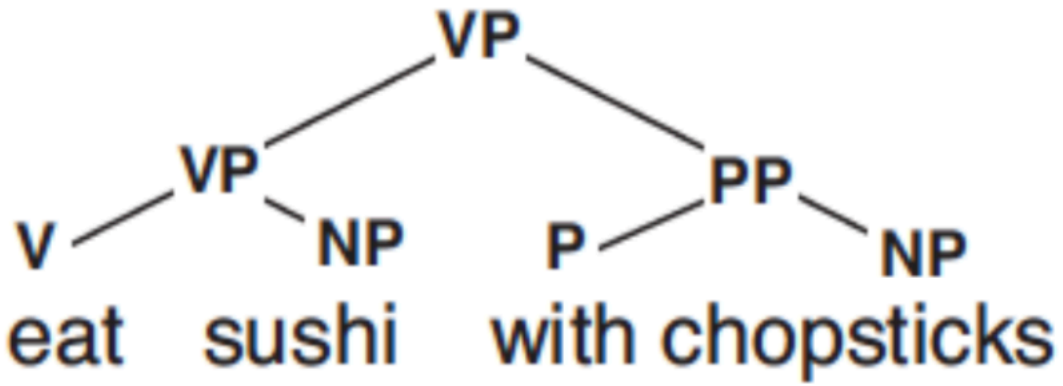
Bài tập 1

$$S \rightarrow NP\ VP$$
$$VP \rightarrow V\ NP$$
$$VP \rightarrow VP\ PP$$
$$PP \rightarrow P\ \ NP$$
$$NP \rightarrow NP\ PP$$

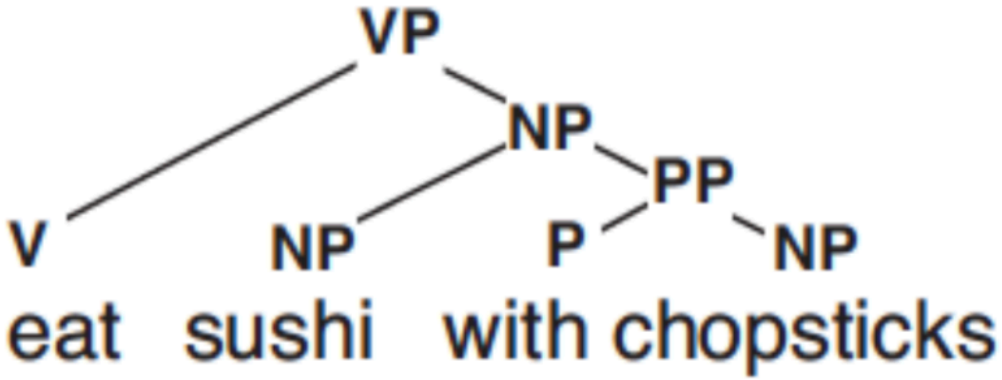
I eat sushi with chopsticks

NP V NP P NP

Đáp án



$$S \rightarrow NP\ VP$$
$$VP \rightarrow V\ NP$$
$$VP \rightarrow VP\ PP$$
$$PP \rightarrow P\ \ NP$$
$$NP \rightarrow NP\ PP$$

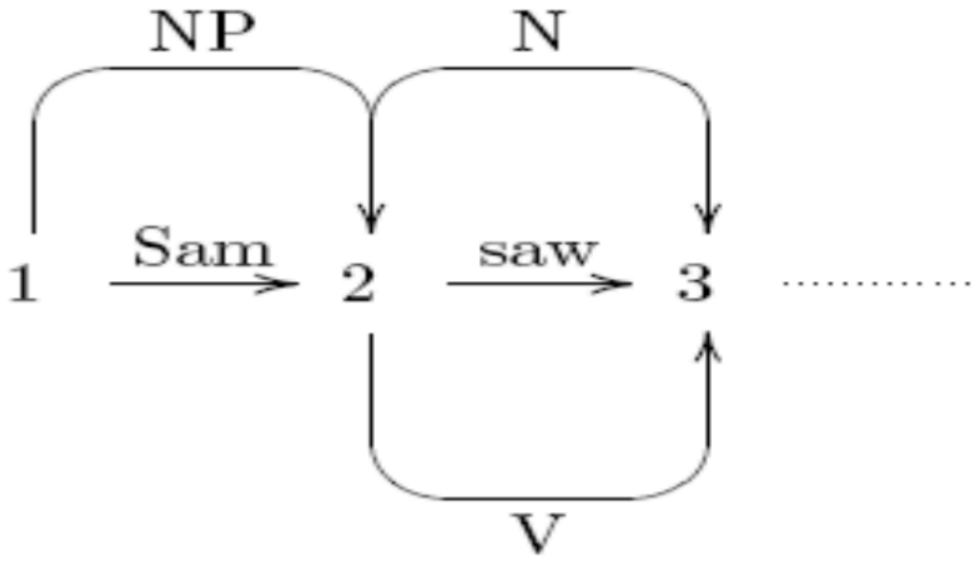


Thuật toán CYK

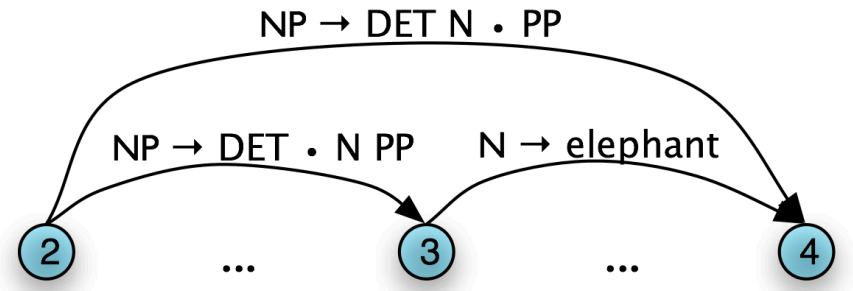
CYK ($\mathcal{G}, w$)
$\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, \mathsf{S})$, $\Sigma \cup \mathcal{V} = \{\mathsf{X}_1, \ldots, \mathsf{X}_r\}$, $w = w_1 w_2 \ldots w_n$.
begin
Initialize the 3d array $\mathsf{B}[1 \ldots n, 1 \ldots n, 1 \ldots r]$ to FALSE
for $i = 1$ to n **do**
for $(\mathsf{X}_j \to x) \in \mathcal{R}$ **do**
if $x = w_i$ **then** $\mathsf{B}[i, i, j] \leftarrow$ TRUE.
for $i = 2$ to n **do** /* Length of span */
for $L = 1$ to $n - i + 1$ **do** /* Start of span */
$R = L + i - 1$ /* Current span $s = w_L w_{L+1} \ldots w_R$ */
for $M = L + 1$ to R **do** /* Partition of span */
/* $x = w_L w_{L+1} \ldots w_{M-1}$, $y = w_M w_{M+1} \ldots w_R$, and $s = xy$ */
for $(\mathsf{X}_\alpha \to \mathsf{X}_\beta \mathsf{X}_\gamma) \in \mathcal{R}$ **do**
/* Can we match X_β to x and X_γ to y? */
if $\mathsf{B}[L, M-1, \beta]$ and $\mathsf{B}[M, R, \gamma]$ **then**
$\mathsf{B}[L, R, \alpha] \leftarrow$ TRUE /* If so, then can generate s by X_α! */
for $i = 1$ to r **do**
if $\mathsf{B}[1, n, i]$ **then return** TRUE
return FALSE

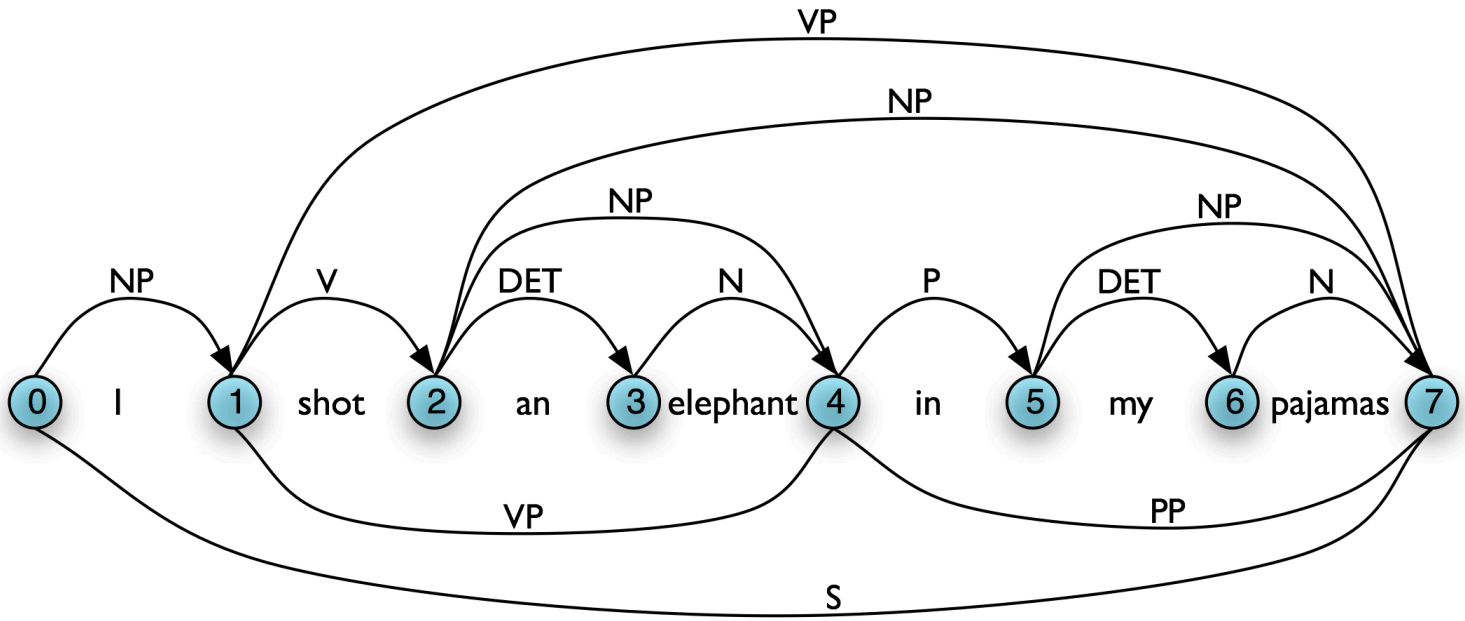
Các Thuật toán phân tích Chart Parsing

- The chart provides a compact representation of local ambiguity.
- The Basic Principle: Avoid duplication, Represent everything, but only represent it once.



Các Thuật toán phân tích Chart Parsing





Thuật toán phân tích Earley

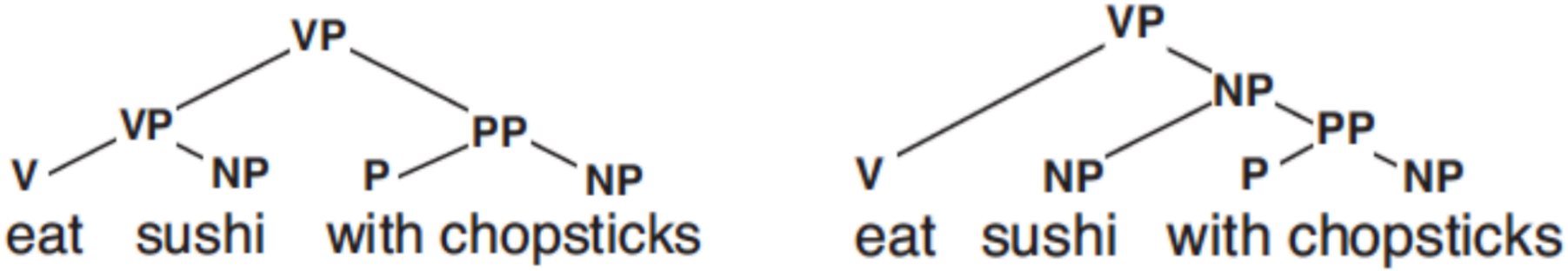
1. Left to right
2. Dynamic programming
3. Using chart:
   ambiguous grammars (including grammars of natural languages). It uses dynamic programming approach -- *partial hypothesized results are stored in a structure called a chart and can be re-used.* This eliminates backtracking and prevents a combinatorial explosion.

Parsing

Probability Context Free Grammar and Syntactic Parsing

Grammars are ambiguous

A grammar might generate multiple trees for a sentence:



What's the most likely parse τ for sentence S ?

We need a model to compute P(τ)

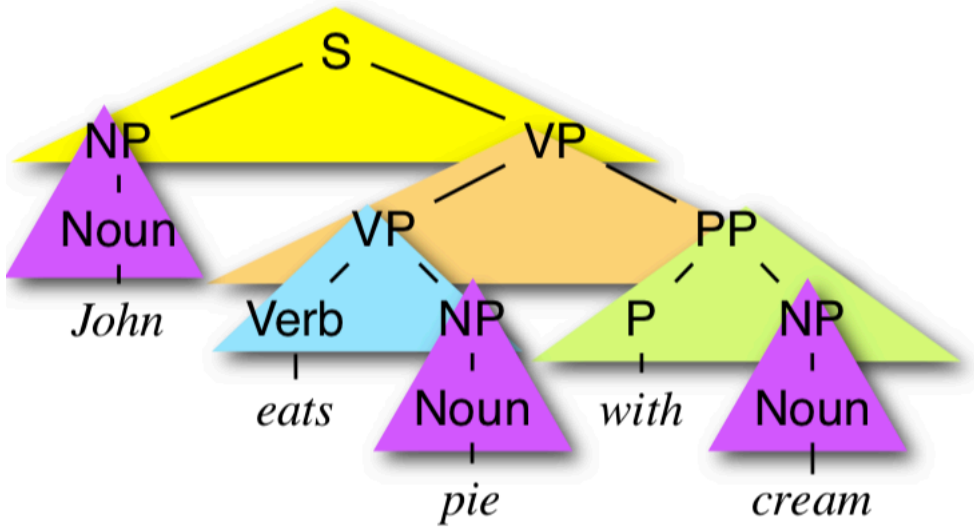
Probabilistic Context-Free Grammars

For every nonterminal X, define a probability distribution $P(X \rightarrow \alpha \mid X)$ over all rules with the same LHS symbol X:

S	→	NP VP	0.8
S	→	S conj S	0.2
NP	→	Noun	0.2
NP	→	Det Noun	0.4
NP	→	NP PP	0.2
NP	→	NP conj NP	0.2
VP	→	Verb	0.4
VP	→	Verb NP	0.3
VP	→	Verb NP NP	0.1
VP	→	VP PP	0.2
PP	→	P NP	1.0

Computing P(τ) with a PCFG

The probability of a tree τ is the product of the probabilities of all its rules:



S	→	NP VP	0.8
S	→	S conj S	0.2
NP	→	Noun	0.2
NP	→	Det Noun	0.4
NP	→	NP PP	0.2
NP	→	NP conj NP	0.2
VP	→	Verb	0.4
VP	→	Verb NP	0.3
VP	→	Verb NP NP	0.1
VP	→	VP PP	0.2
PP	→	P NP	1.0

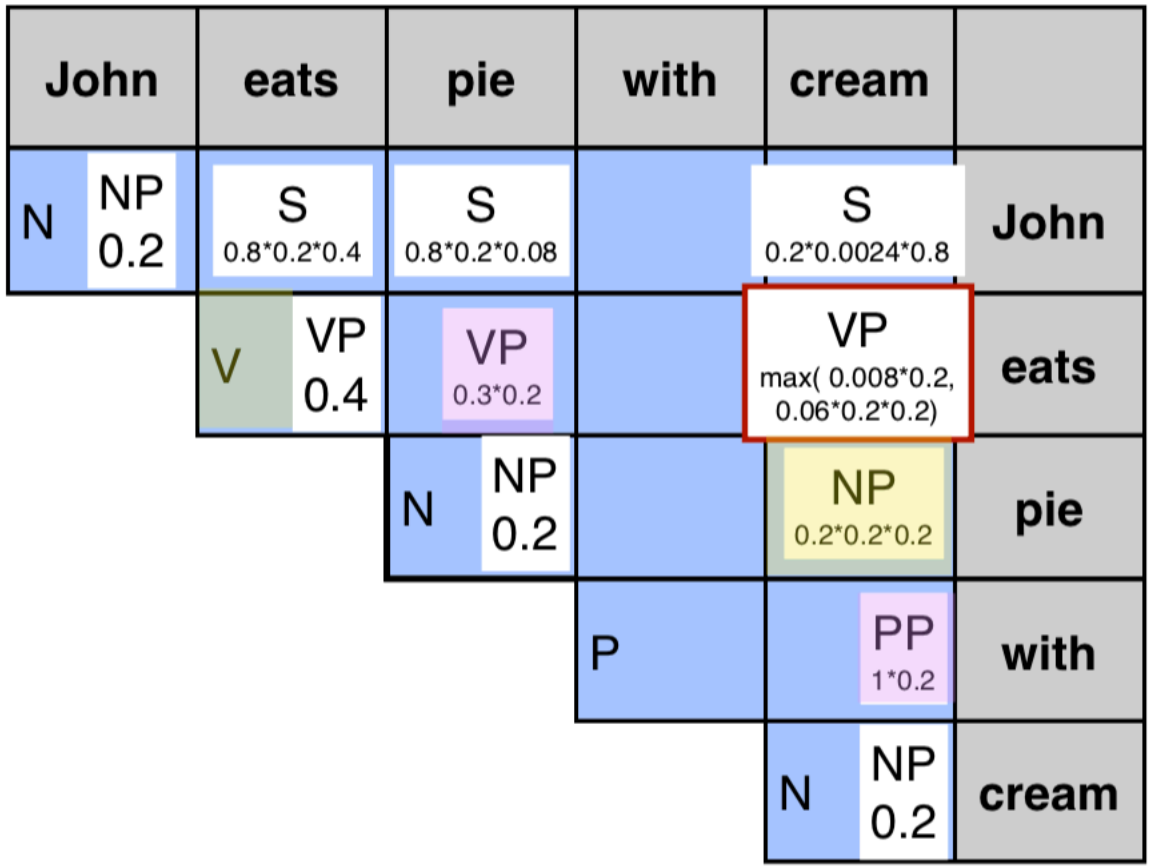
$$P(\tau) = 0.8 \times 0.3 \times 0.2 \times 1.0 \times 0.2^3$$

$$= \mathbf{0.00384}$$

Ví dụ

Input: POS-tagged sentence

`John_N eats_V pie_N with_P cream_N`



S	→	NP VP	0.8
S	→	S conj S	0.2
NP	→	Noun	0.2
NP	→	Det Noun	0.4
NP	→	NP PP	0.2
NP	→	NP conj NP	0.2
VP	→	Verb	0.4
VP	→	Verb NP	0.3
VP	→	Verb NP NP	0.1
VP	→	VP PP	0.2
PP	→	P NP	1.0

Bài tập

$S \rightarrow NP\ VP$	1.0
$VP \rightarrow V\ NP$	0.7
$VP \rightarrow VP\ PP$	0.3
$PP \rightarrow P\ \ NP$	1.0
$NP \rightarrow NP\ PP$	1.0

I eat sushi with chopsticks

NP V NP P NP

Weaknesses of PCFGs

- Lack of sensitivity to lexical information

(a)

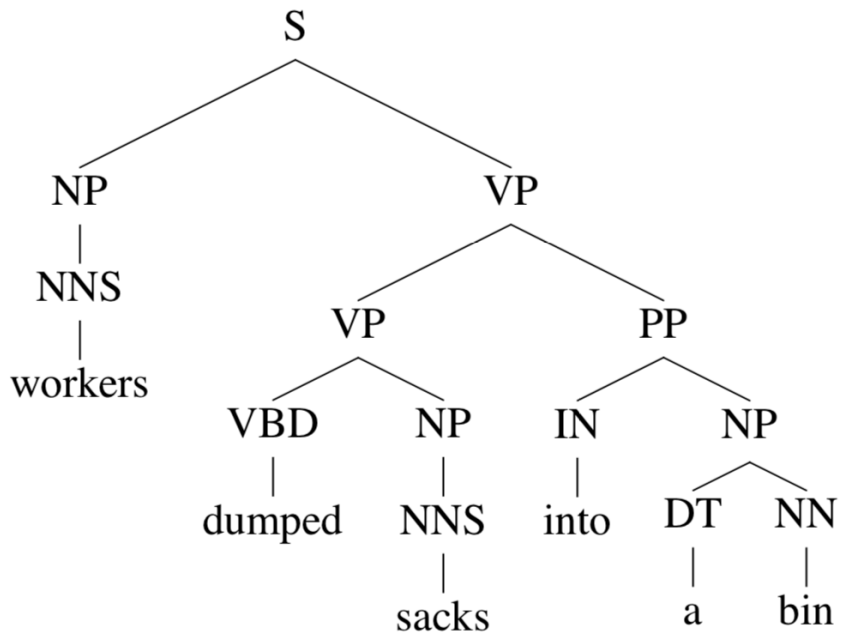
Rules
S → NP VP
NP → NNS
VP → VP PP
VP → VBD NP
NP → NNS
PP → IN NP
NP → DT NN
NNS → workers
VBD → dumped
NNS → sacks
IN → into
DT → a
NN → bin

(b)

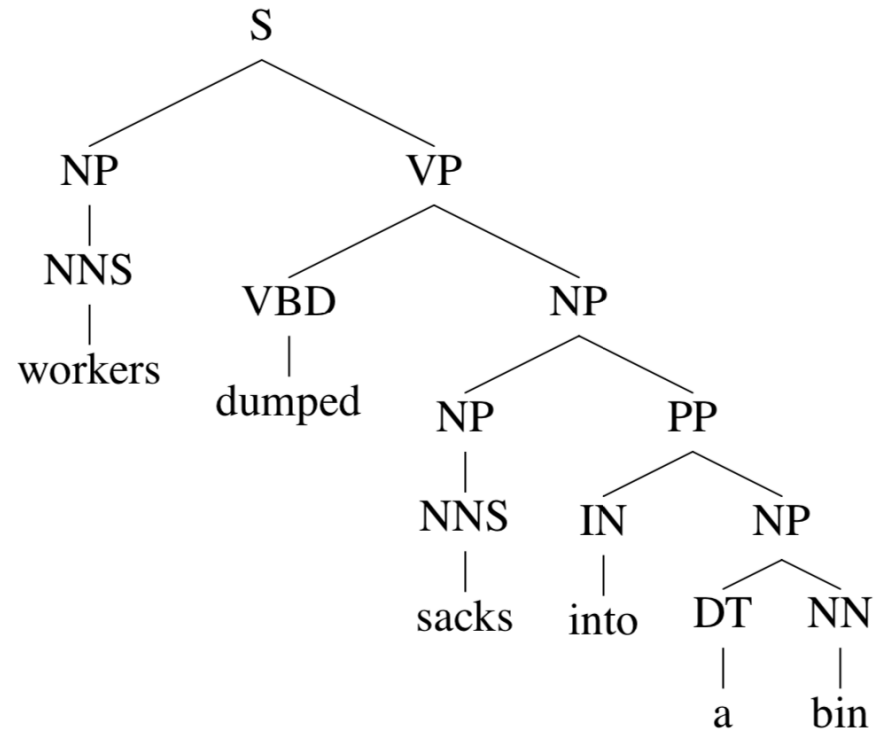
Rules
S → NP VP
NP → NNS
NP → NP PP
VP → VBD NP
NP → NNS
PP → IN NP
NP → DT NN
NNS → workers
VBD → dumped
NNS → sacks
IN → into
DT → a
NN → bin

If $P(\text{NP} \rightarrow \text{NP PP} \mid \text{NP}) > P(\text{VP} \rightarrow \text{VP PP} \mid \text{VP})$ then (b) is more probable, else (a) is more probable.

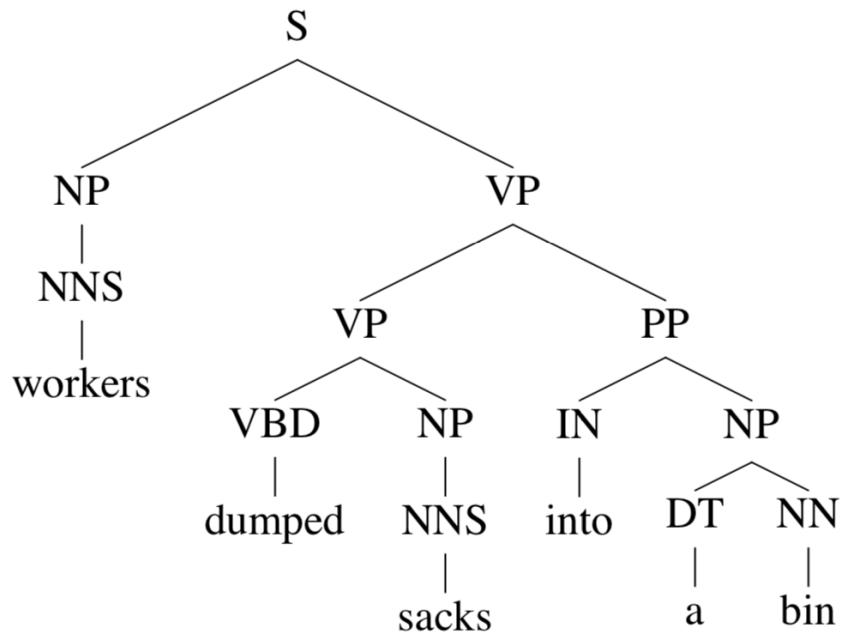
(a)


S
NP
VP
NNS
VP
PP
workers
VBD
NP
IN
NP
dumped
NNS
into
DT
NN
sacks
a
bin

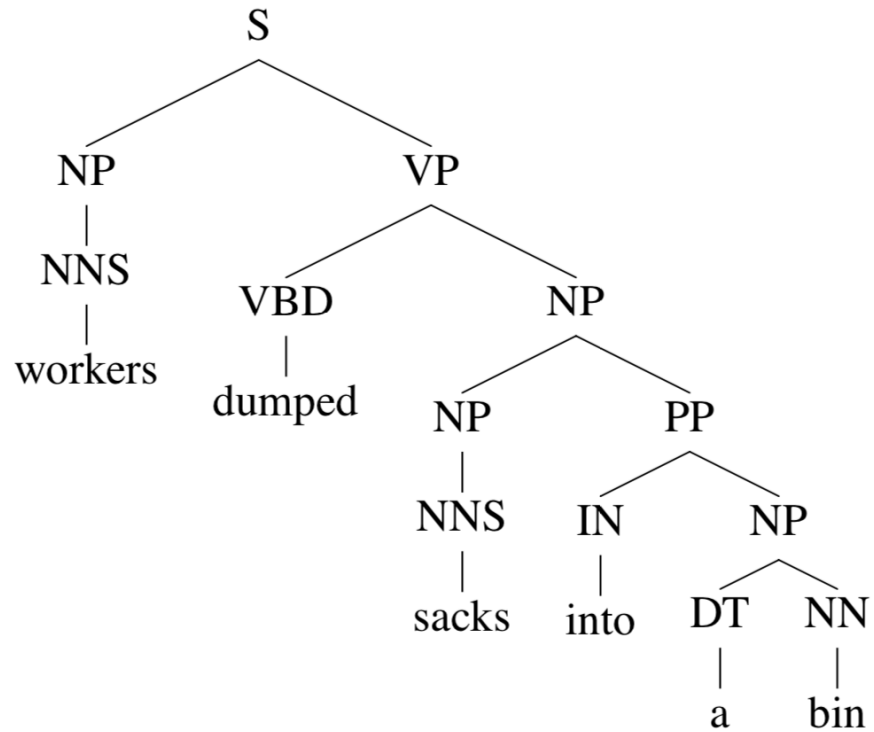
(b)


S
NP
VP
NNS
VBD
NP
workers
dumped
NP
PP
NNS
IN
NP
sacks
into
DT
NN
a
bin

(a)



(b)



If $P(\text{NP} \rightarrow \text{NP PP} \mid \text{NP}) > P(\text{VP} \rightarrow \text{VP PP} \mid \text{VP})$ then (b) is more probable, else (a) is more probable.

Attachment decision is completely independent of the words

Solution: Head-Driven Phrase Structure Grammar

Add annotations specifying the "head" of each rule:

S	⇒	NP	VP
VP	⇒	Vi	
VP	⇒	Vt	NP
VP	⇒	VP	PP
NP	⇒	DT	NN
NP	⇒	NP	PP
PP	⇒	IN	NP

Vi	⇒	sleeps
Vt	⇒	saw
NN	⇒	man
NN	⇒	woman
NN	⇒	telescope
DT	⇒	the
IN	⇒	with
IN	⇒	in

Note: S=sentence, VP=verb phrase, NP=noun phrase, PP=prepositional phrase, DT=determiner, Vi=intransitive verb, Vt=transitive verb, NN=noun, IN=preposition

Head-Driven Phrase Structure Grammar

Each context-free rule has one “special” child that is the head of the rule. e.g.,

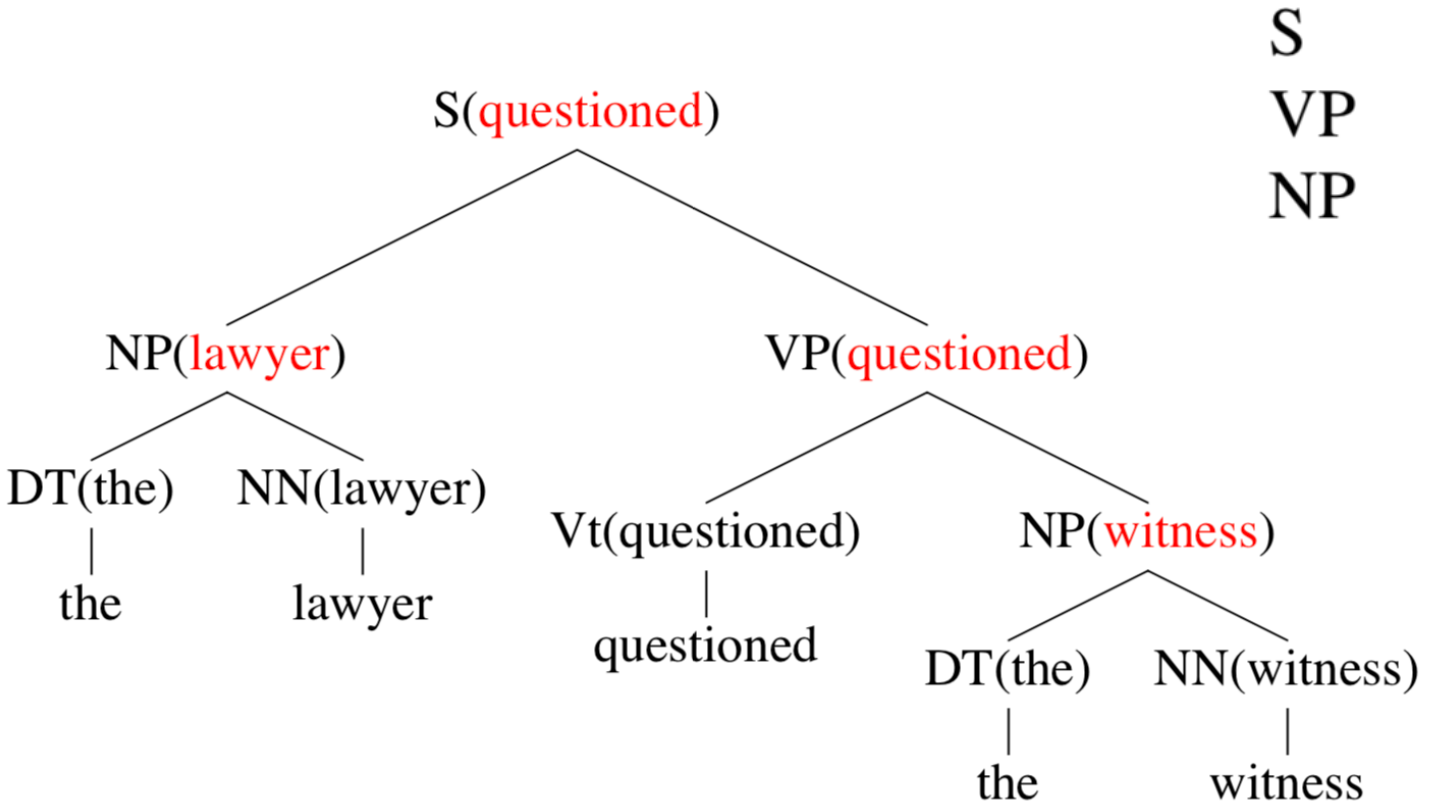
S	⇒	NP	VP		(VP is the head)
VP	⇒	Vt	NP		(Vt is the head)
NP	⇒	DT	NN	NN	(NN is the head)

Some intuitions:

– The central sub-constituent of each rule.

– The semantic predicate in each rule.

Adding Headwords to Trees



$$S \Rightarrow NP \; VP$$
$$VP \Rightarrow Vt \; NP$$
$$NP \Rightarrow DT \; NN$$

A constituent receives its headword from its **head child**.

Lexicalized PCFGs

- The basic idea in lexicalized PCFGs will be to replace rules such as

$$S \rightarrow NP\ VP$$

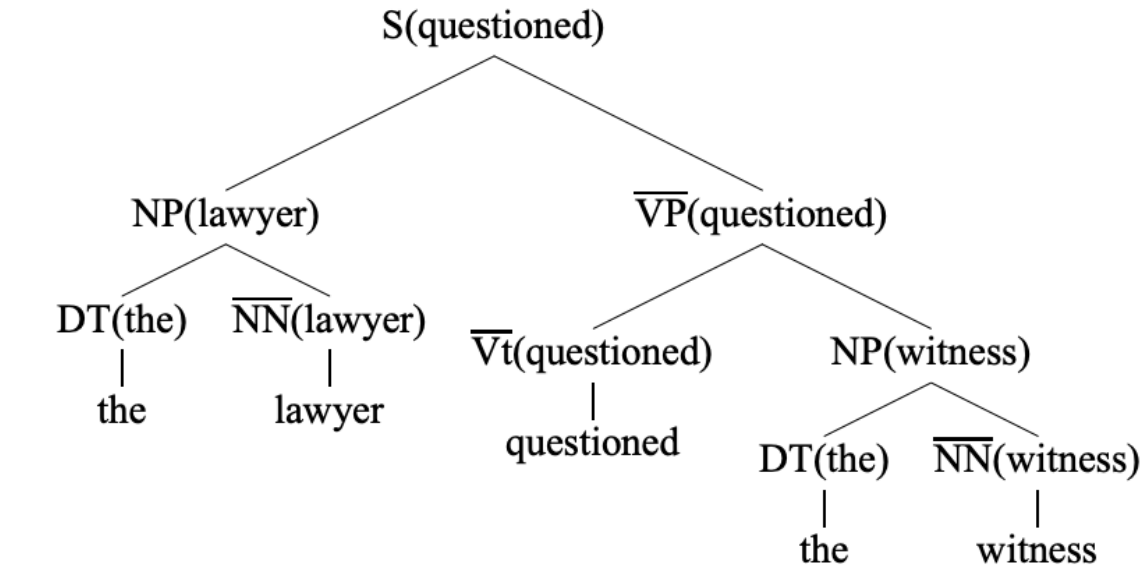
with lexicalized rules such as

$$S(examined) \rightarrow NP(lawyer)\ VP(examined)$$

Example

In this case the parse tree consists of the following sequence of rules:

$S(\text{questioned}) \rightarrow_2 NP(\text{lawyer})\ VP(\text{questioned})$
$NP(\text{lawyer}) \rightarrow_2 DT(\text{the})\ NN(\text{lawyer})$
$DT(\text{the}) \rightarrow \text{the}$
$NN(\text{lawyer}) \rightarrow \text{lawyer}$
$VP(\text{questioned}) \rightarrow_1 Vt(\text{questioned})\ NP(\text{witness})$
$NP(\text{witness}) \rightarrow_2 DT(\text{the})\ NN(\text{witness})$
$DT(\text{the}) \rightarrow \text{the}$
$NN(\text{witness}) \rightarrow \text{witness}$



Parameter Estimation in Lexicalized PCFGs

The number of rules (and therefore parameters) in the model is very large. However with appropriate smoothing—using techniques described earlier in the class, for language modeling—we can derive estimates that are robust and effective in practice.

S(examined) → NP(lawyer) VP(examined)

X = S
H = examined
R = S → NP VP
M = lawyer

$$
\begin{aligned}
& q(\text{S(examined)} \rightarrow_2 \text{NP(lawyer) VP(examined)}) \\
= \quad & P(R = \text{S} \rightarrow_2 \text{NP VP}, M = \text{lawyer} | X = \text{S}, H = \text{examined})
\end{aligned}
$$

Parameter Estimation in Lexicalized PCFGs

$$
\begin{aligned}
& P(R = \mathrm{S} \rightarrow_2 \mathrm{NP\ VP}, M = \text{lawyer} | X = \mathrm{S}, H = \text{examined}) \\
= \quad & P(R = \mathrm{S} \rightarrow_2 \mathrm{NP\ VP} | X = \mathrm{S}, H = \text{examined}) && (3) \\
& \times P(M = \text{lawyer} | R = \mathrm{S} \rightarrow_2 \mathrm{NP\ VP}, X = \mathrm{S}, H = \text{examined}) && (4)
\end{aligned}
$$

$$
\begin{aligned}
q_{ML}(\mathrm{S} \rightarrow_2 \mathrm{NP\ VP} | \mathrm{S}, \text{examined}) &= \frac{\text{count}(R = \mathrm{S} \rightarrow_2 \mathrm{NP\ VP}, X = \mathrm{S}, H = \text{examined})}{\text{count}(X = \mathrm{S}, H = \text{examined})} \\
q_{ML}(\mathrm{S} \rightarrow_2 \mathrm{NP\ VP} | \mathrm{S}) &= \frac{\text{count}(R = \mathrm{S} \rightarrow_2 \mathrm{NP\ VP}, X = \mathrm{S})}{\text{count}(X = \mathrm{S})}
\end{aligned}
$$

Parameter Estimation in Lexicalized PCFGs

Let a rule be $\mathrm{LHS} \rightarrow L_n L_{n-1} \ldots L_1 H R_1 \ldots R_{n-1} R_n$
Idea: break it into smaller probabilities, "generating" the head and then left dependents and right dependents:

$$p(H|\mathrm{LHS}) \times p_L(L_1|H, \mathrm{LHS}) \ldots p_L(L_n|H, \mathrm{LHS}) \times p_L(\mathrm{STOP} \mid L_n, \mathrm{LHS})$$
$$\times p_R(R_1|H, \mathrm{LHS}) \ldots p_R(R_n|H, \mathrm{LHS}) \times p_R(\mathrm{STOP} \mid R_n, \mathrm{LHS})$$

Penn Treebank

The Penn Treebank (PTB) project selected 2,499 stories from a three year Wall Street Journal (WSJ) collection of 98,732 stories for syntactic annotation. These 2,499 stories have been distributed in both Treebank-2 (LDC95T7) and Treebank-3 (LDC99T42) releases of PTB. Treebank-2 includes the raw text for each story. Three "map" files are available in a compressed file (pennTB_tipster_wsj_map.tar.gz) as an additional download for users who have licensed Treebank-2 and provide the relation between the 2,499 PTB filenames and the corresponding WSJ DOCNO strings in TIPSTER.

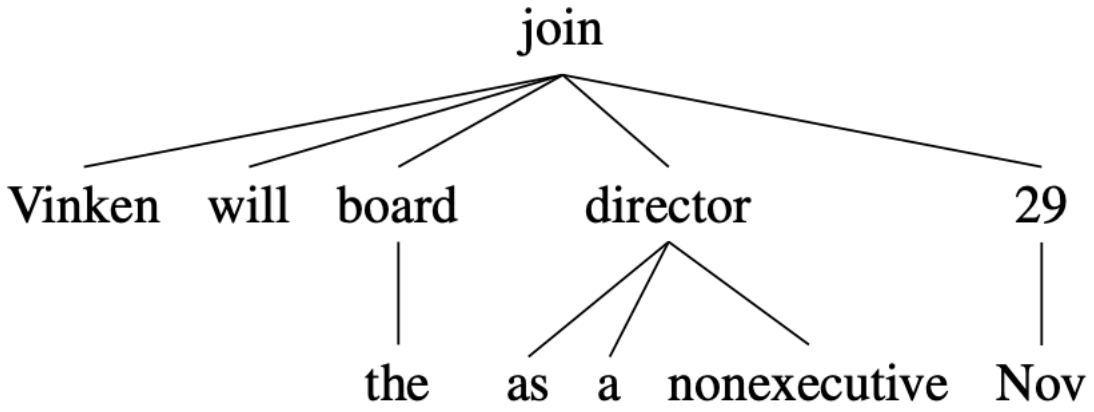
```
(S (NP-SBJ I)
   (VP wish
       (SBAR 0
             (S (NP-SBJ I)
                (VP was
                    (NP-MNR-PRD that way)))))
   .
   E_S))
```

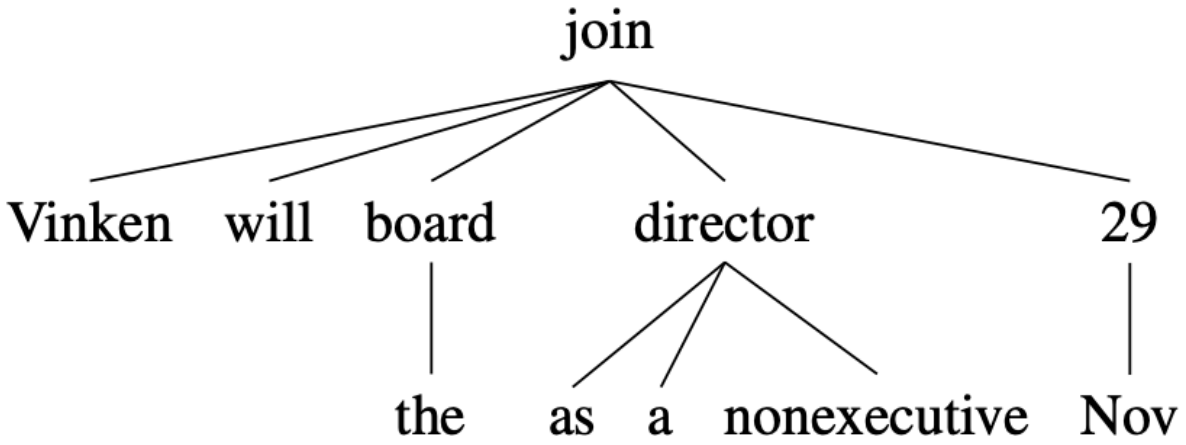
Parsing

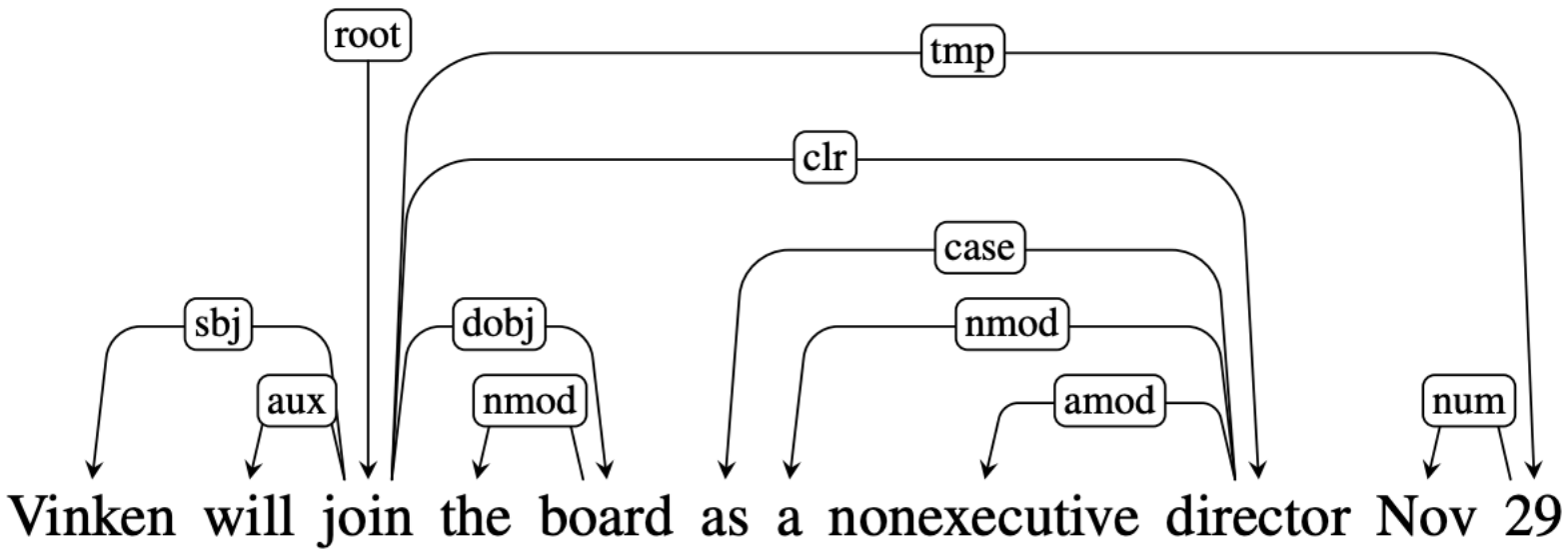
Dependency Parsing

Cú pháp phụ thuộc và cấu trúc phụ thuộc (dependency grammar and dependency structure)

- Dependency syntax postulates that syntactic structure consists of relations between lexical items, normally binary asymmetric relations ("arrows") called dependencies



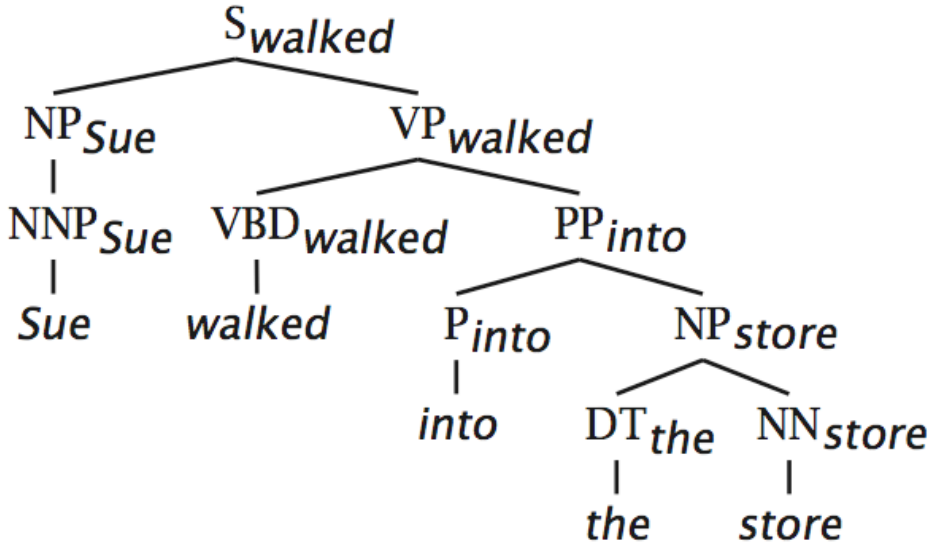

join
Vinken
will
board
director
29
the
as
a
nonexecutive
Nov


root
tmp
clr
case
sbj
dobj
nmod
aux
nmod
amod
num
Vinken
will
join
the
board
as
a
nonexecutive
director
Nov
29

Relation between phrase structure and dependency structure

- A dependency grammar has a notion of a head. Officially, CFGs don't.
- But modern linguistic theory and all modern statistical parsers (Charniak, Collins, Stanford, …) do, via hand-written phrasal "head rules":
  - The head of a Noun Phrase is a noun/number/adj/…
  - The head of a Verb Phrase is a verb/modal/….
- The head rules can be used to extract a dependency parse from a CFG parse

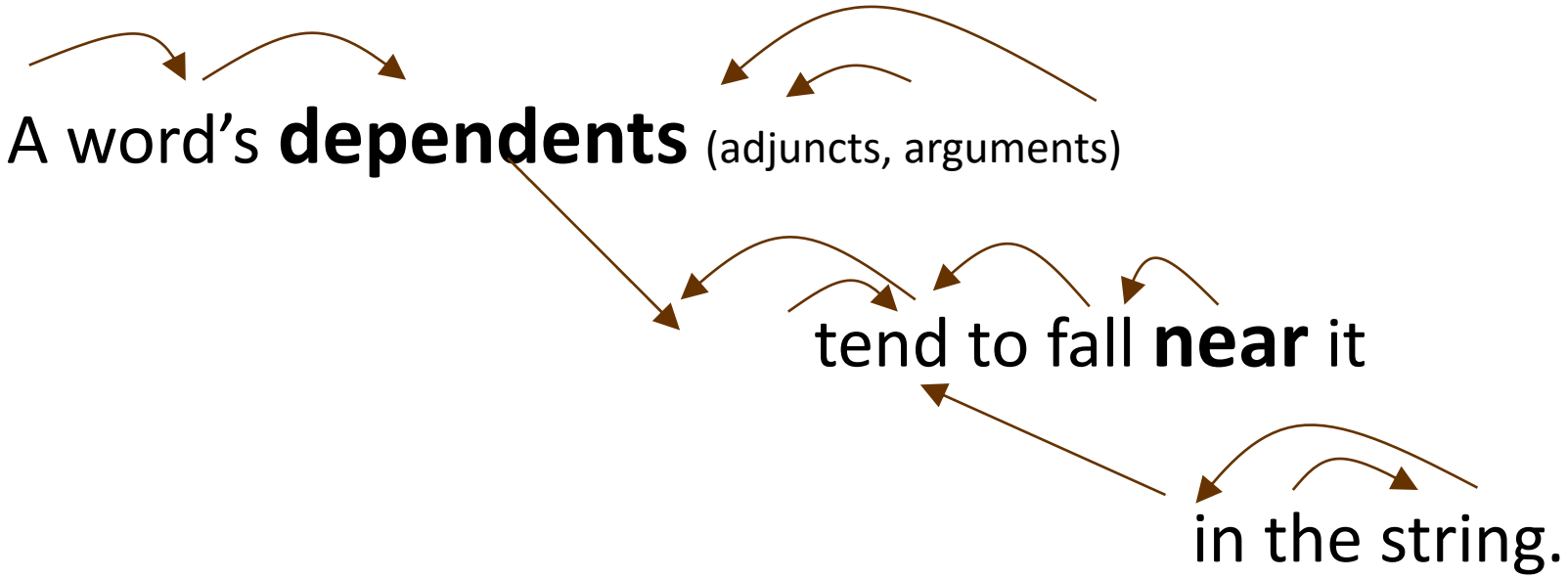
  - The closure of dependencies give constituency from a dependency tree
  - But the dependents of a word must be at the same level (i.e., "flat") – there can be no VP!



Dependency Conditioning Preferences

Sources of information:

- bilexical dependencies
- distance of dependencies
- valency of heads (number of dependents)



MaltParser

[Nivre et al. 2008]

- A simple form of greedy discriminative dependency parser
- The parser does a sequence of bottom up actions
  - Roughly like “shift” or “reduce” in a shift-reduce parser, but the “reduce” actions are specialized to create dependencies with head on left or right
- The parser has:
  - a stack σ, written with top to the right
    - which starts with the ROOT symbol
  - a buffer β, written with top to the left
    - which starts with the input sentence
  - a set of dependency arcs A
    - which starts off empty
  - a set of actions

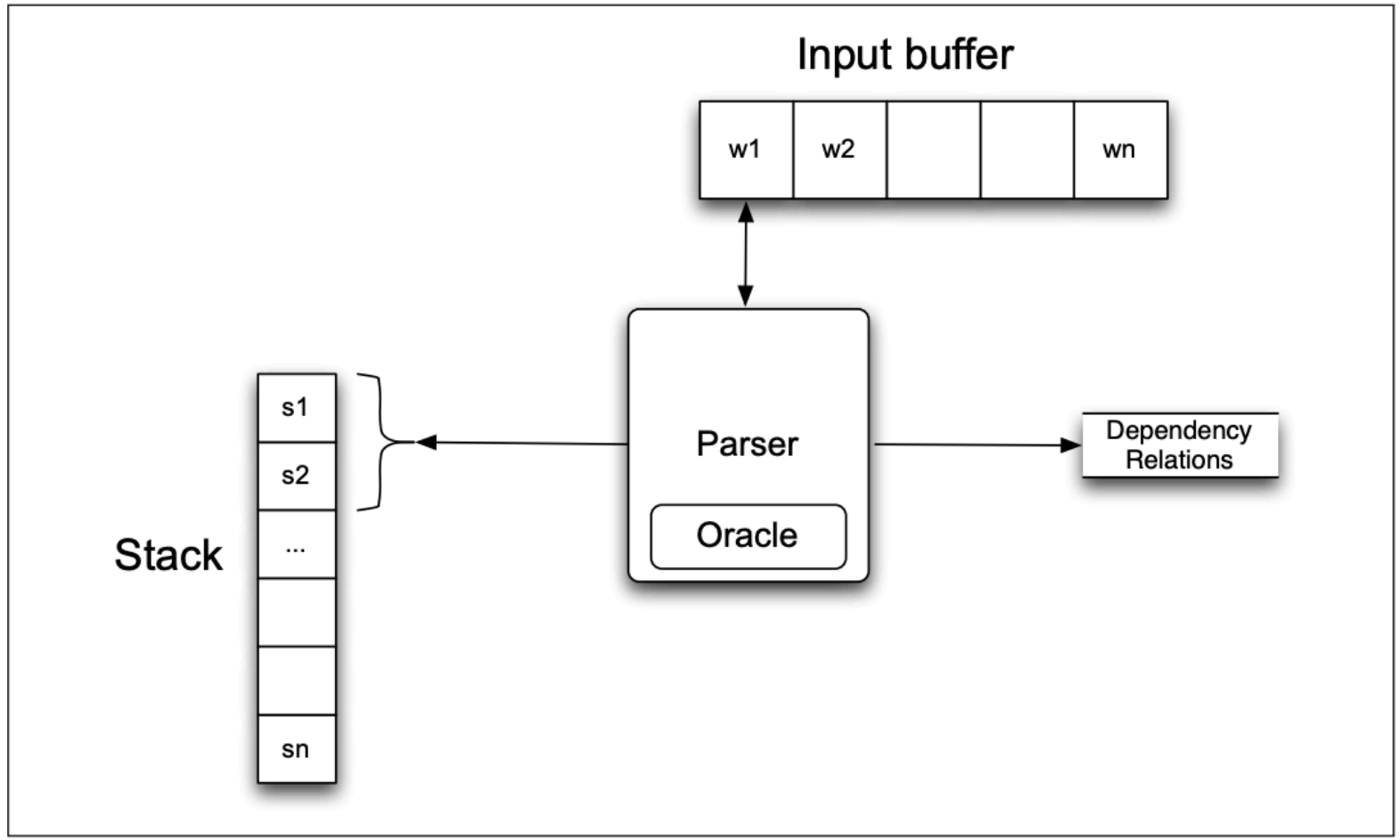
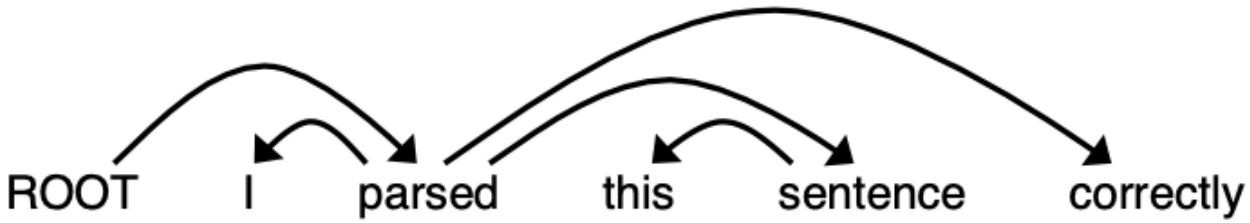


Figure 15.5 Basic transition-based parser. The parser examines the top two elements of the stack and selects an action based on consulting an oracle that examines the current configuration.

Greedy Transition-Based Parsing



Stack	Buffer	New dependency	Transition
[ROOT]	[I, parsed, this, sentence, correctly]		Initial Configuration
[ROOT, I]	[parsed, this, sentence, correctly]		SHIFT
[ROOT, I, parsed]	[this, sentence, correctly]		SHIFT
[ROOT, parsed]	[this, sentence, correctly]	parsed→I	LEFT-ARC

Actions

- Assign the current word as the head of some previously seen word,
- Assign some previously seen word as the head of the current word,
- Or postpone doing anything with the current word, adding it to a store for later processing.

- LEFTARC: Assert a head-dependent relation between the word at the top of the stack and the word directly beneath it; remove the lower word from the stack.
- RIGHTARC: Assert a head-dependent relation between the second word on the stack and the word at the top; remove the word at the top of the stack;
- SHIFT: Remove the word from the front of the input buffer and push it onto the stack.

Basic transition-based dependency parser

Start: σ = [ROOT], β = w_1, …, w_n , A = ∅

1. Shift σ, w_i|β, A ➔ σ|w_i, β, A
2. Left-Arc$_r$ σ|w_i, w_j|β, A ➔ σ, w_j|β, A∪{r(w_j,w_i)}
3. Right-Arc$_r$ σ|w_i, w_j|β, A ➔ σ, w_i|β, A∪{r(w_i,w_j)}

Finish: β = ∅

Notes:

- Unlike the regular presentation of the CFG reduce step, dependencies combine one thing from each of stack and buffer

Actions (“arc-eager” dependency parser)

Start: σ = [ROOT], $\beta = w_1, \ldots, w_n$, A = $\emptyset$

1. Left-Arc$_r$ $\sigma|w_i, w_j|\beta, A \rightarrow \sigma, w_j|\beta, A \cup \{r(w_j,w_i)\}$
   Precondition: $r'(w_k, w_i) \notin A$, $w_i \neq$ ROOT
2. Right-Arc$_r$ $\sigma|w_i, w_j|\beta, A \rightarrow \sigma|w_i|w_j, \beta, A \cup \{r(w_i,w_j)\}$
3. Reduce $\sigma|w_i, \beta, A \rightarrow \sigma, \beta, A$
   Precondition: $r'(w_k, w_i) \in A$
4. Shift $\sigma, w_i|\beta, A \rightarrow \sigma|w_i, \beta, A$

Finish: $\beta = \emptyset$

This is the common “arc-eager” variant: a head can immediately take a right dependent, before *its* dependents are found

Example 1

Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)
7	[root, book, the, flight]	[]	LEFTARC	(the ← flight)
8	[root, book, flight]	[]	RIGHTARC	(book → flight)
9	[root, book]	[]	RIGHTARC	(root → book)
10	[root]	[]	Done	

Figure 15.7 Trace of a transition-based parse.

Example 2

Happy children like to play with their friends .

	[ROOT]	[Happy, children, …]	Ø
Shift	[ROOT, Happy]	[children, like, …]	Ø
LA_{amod}	[ROOT]	[children, like, …]	{amod(children, happy)} = A_1
Shift	[ROOT, children]	[like, to, …]	A_1
LA_{nsubj}	[ROOT]	[like, to, …]	A_1 ∪ {nsubj(like, children)} = A_2
RA_{root}	[ROOT, like]	[to, play, …]	A_2 ∪{root(ROOT, like) = A_3
Shift	[ROOT, like, to]	[play, with, …]	A_3
LA_{aux}	[ROOT, like]	[play, with, …]	A_3∪{aux(play, to) = A_4
RA_{xcomp}	[ROOT, like, play]	[with their, …]	A_4∪{xcomp(like, play) = A_5

Example

1. Left-Arc$_r$ $\sigma|w_i, w_j|\beta, A \rightarrow \sigma, w_j|\beta, A \cup \{r(w_j,w_i)\}$
   Precondition: $(w_k, r', w_i) \notin A$, $w_i \neq$ ROOT
2. Right-Arc$_r$ $\sigma|w_i, w_j|\beta, A \rightarrow \sigma|w_i|w_j, \beta, A \cup \{r(w_i,w_j)\}$
3. Reduce $\sigma|w_i, \beta, A \rightarrow \sigma, \beta, A$
   Precondition: $(w_k, r', w_i) \in A$
4. Shift $\sigma, w_i|\beta, A \rightarrow \sigma|w_i, \beta, A$

Happy children like to play with their friends .

RA$_{xcomp}$	[ROOT, like, play]	[with their, …]	$A_4 \cup \{xcomp(like, play) = A_5$
RA$_{prep}$	[ROOT, like, play, with]	[their, friends, …]	$A_5 \cup \{prep(play, with) = A_6$
Shift	[ROOT, like, play, with, their]	[friends, .]	A_6
LA$_{poss}$	[ROOT, like, play, with]	[friends, .]	$A_6 \cup \{poss(friends, their) = A_7$
RA$_{pobj}$	[ROOT, like, play, with, friends]	[.]	$A_7 \cup \{pobj(with, friends) = A_8$
Reduce	[ROOT, like, play, with]	[.]	A_8
Reduce	[ROOT, like, play]	[.]	A_8
Reduce	[ROOT, like]	[.]	A_8
RA$_{punc}$	[ROOT, like, .]	[]	$A_8 \cup \{punc(like, .) = A_9$

You terminate as soon as the buffer is empty. Dependencies = A_9

Tổng kết

- Hai tiếp cận
- Ý nghĩa
- Các phương pháp phân tích tương ứng