

Context-Free Grammar

COMP90042

Natural Language Processing

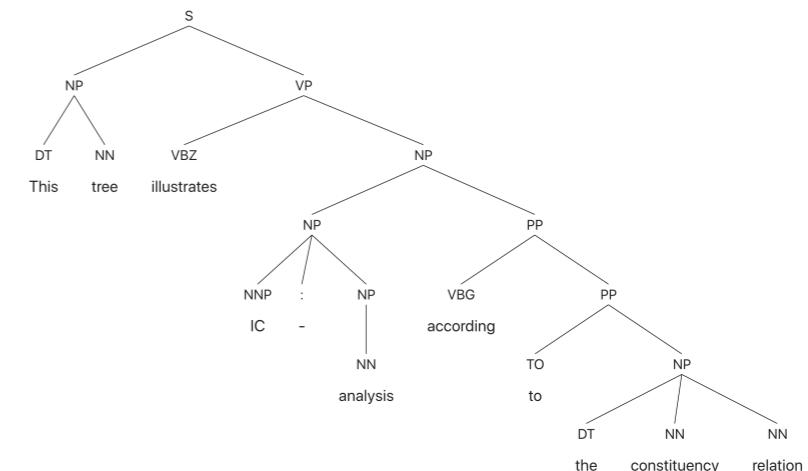
Lecture 14

Semester 1 2022 Week 7

Jey Han Lau



THE UNIVERSITY OF
MELBOURNE



Recap

- Center embedding
 - ▶ The cat loves Mozart
 - ▶ The cat **the dog chased** loves Mozart
 - ▶ The cat **the dog the rat bit chased** loves Mozart
 - ▶ The cat **the dog the rat the elephant admired bit chased** loves Mozart
- Cannot be captured by regular expressions (S^nV^n)
- **Context-free grammar!**

Basics of Context-Free Grammars

- **Symbols**

- **Terminal:** word such as book
- **Non-terminal:** syntactic label such as NP or VP

convention:

↳ lowercase for terminals
↳ uppercase for non-terminals

- **Productions (rules)**

- $W \rightarrow X Y Z$
- Exactly one non-terminal on left-hand side (LHS)
- An ordered list of symbols on right-hand side (RHS);
can be **terminals** or **non-terminals**

- **Start symbol:** S

Why “Context Free”

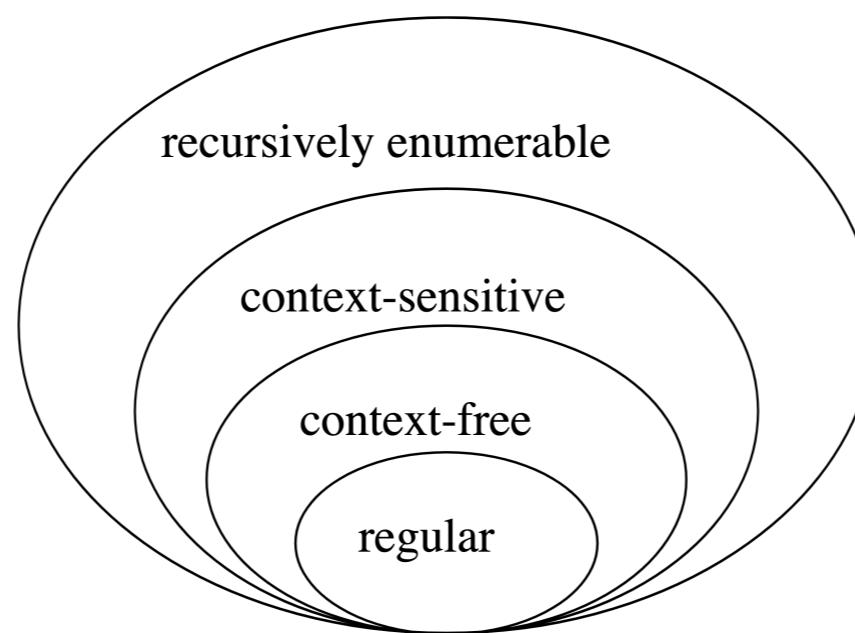
→ 口和当前状态有关

$$\underset{x}{W} \rightarrow X Y Z$$

- Production rule depends only on the LHS (and not on ancestors, neighbours)
 - Analogous to Markov chain
 - Behaviour at each step depends only on current state

Context-Free vs. Regular

- Context-free languages **more general** than regular languages
 - ▶ Allows recursive nesting



CFG Parsing

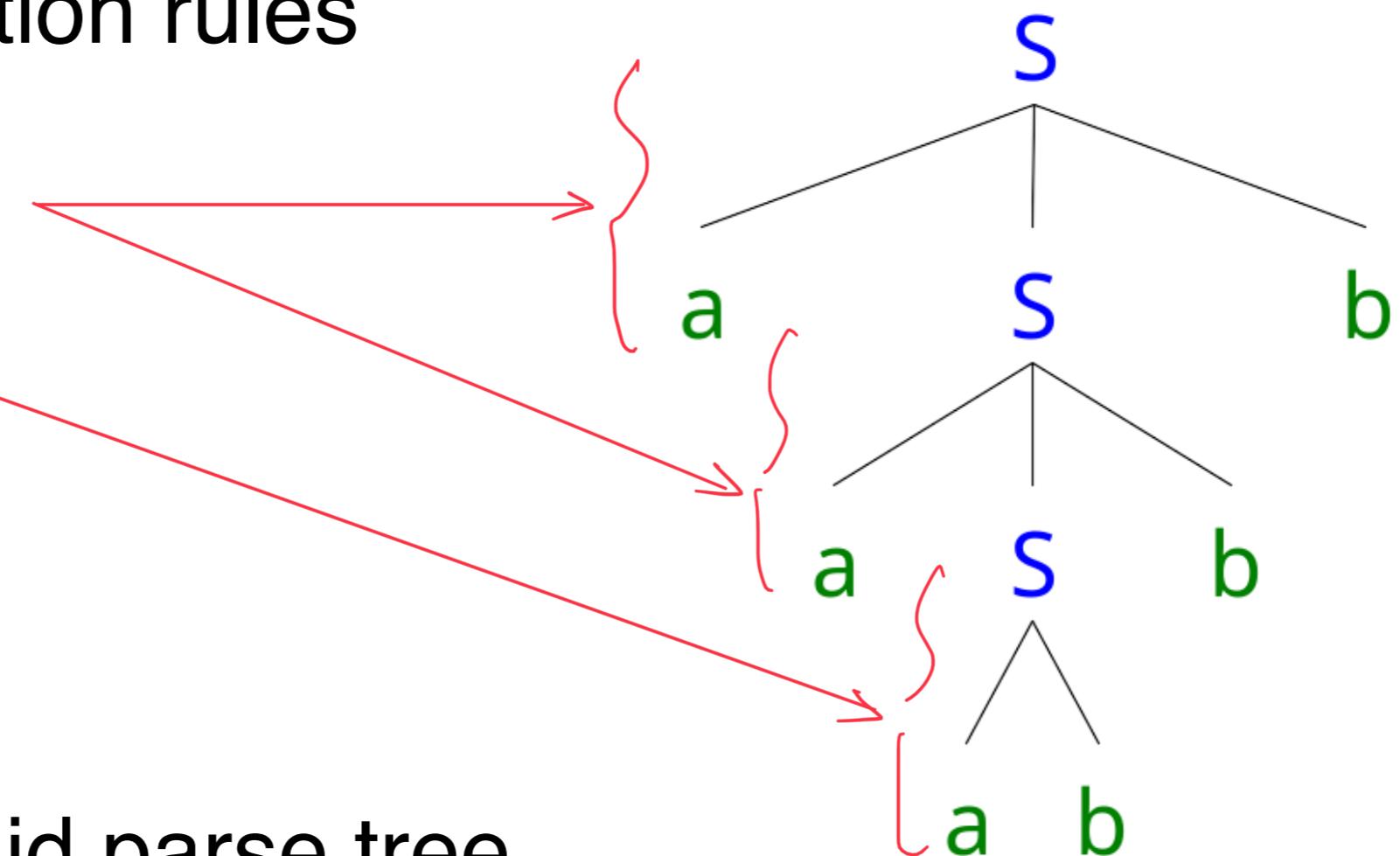
- Given production rules

- $S \rightarrow a S b$
- $S \rightarrow a b$

- And a string

- aaabbb

- Produce a valid parse tree



What This Means?

- If English can be represented with CFG:
 - ▶ ① first develop the production rules
 - ▶ ② can then build a “parser” to automatically judge whether a sentence is grammatical!
- But is natural language context-free?
 - Not quite: cross-serial dependencies ($a^m b^n c^m d^n$)
交叉依赖.

Swiss-German:

...de Karl d'Maria em Peter de Hans laat hälfe lärne schwüme

English:

...Charles lets Mary help Peter to teach John to Swim

But...

- CFG strike a good balance:
 - ▶ CFG covers most syntactic patterns
 - ▶ CFG parsing is computational efficient
- We use CFG to describe a core fragment of English syntax

Outline

- Constituents
- CYK Algorithm
- Representing English with CFGs

Constituents

Syntactic Constituents

- Sentences are broken into **constituents**
 - ▶ word sequence that function as a **coherent unit** for linguistic analysis
 - ▶ helps build CFG production rules
- Constituents have certain **key properties**:
 - ▶ movement
 - ▶ substitution
 - ▶ coordination



Units / clause / seq of words / phrases

Movement

- Constituents can be moved around sentences
 - Abigail gave [her brother] [a fish]
 - Abigail gave [a fish] to [her brother]
- Contrast: [gave her], [brother a]

Substitution

- Constituents can be substituted by other phrases of the same type
 - ▶ Max thanked [his older sister]
 - ▶ Max thanked [her]
- Contrast: [Max thanked], [thanked his]

Coordination

- Constituents can be conjoined with coordinators like and and or
 - [Abigail] and [her young brother] brought a fish
 - Abigail [bought a fish] and [gave it to Max]
 - Abigail [bought] and [greedily ate] a fish

Constituents and Phrases

- Once we identify constituents, we use **phrases** to describe them
- Phrases are determined by their **head word**:
 - ▶ noun phrase: her younger brother
 - ▶ verb phrase: greedily ate it
- We can use CFG to formalise these intuitions



away a pie *a lecture*
He gave a lecture and away a pie.

Which of the following is a constituent

- a lecture
- gave a lecture
- a pie
- away a pie

PollEv.com/jeyhanlau569



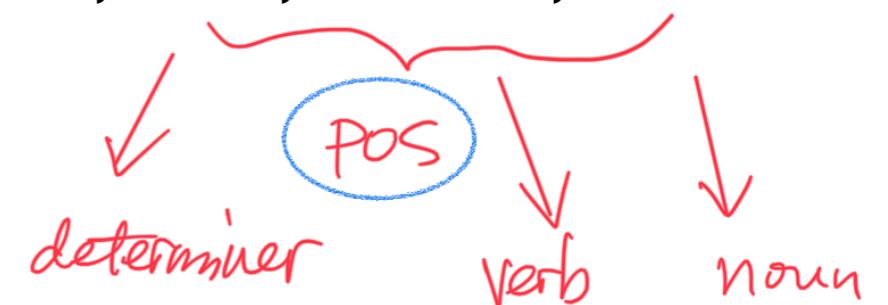
A Simple CFG for English

Terminal symbols: *rat, the, ate, cheese*

Non-terminal symbols: S, NP, VP, DT, VBD, NN

Productions:

- S → NP VP
- NP → DT NN
- VP → VBD NP
- DT → *the*
- NN → *rat*
- NN → *cheese*
- VBD → *ate*



Generating Sentences with CFGs

Always start with S (the sentence/start symbol)

S

Apply a rule with S on LHS ($S \rightarrow NP\ VP$), i.e substitute RHS

NP VP

Apply a rule with NP on LHS ($NP \rightarrow DT\ NN$)

DT NN VP

Apply rule with DT on LHS ($DT \rightarrow the$)

the NN VP

Apply rule with NN on LHS ($NN \rightarrow rat$)

the rat VP

$S \rightarrow NP\ VP$

$NP \rightarrow DT\ NN$

$VP \rightarrow VBD\ NP$

$DT \rightarrow the$

$NN \rightarrow rat$

$NN \rightarrow cheese$

$VBD \rightarrow ate$

Generating Sentences with CFGs

Apply rule with VP on LHS ($VP \rightarrow VBD\ NP$)

the rat VBD NP

Apply rule with VBD on LHS ($VBD \rightarrow ate$)

the rat ate NP

Apply rule with NP on LHS ($NP \rightarrow DT\ NN$)

the rat ate DT NN

Apply rule with DT on LHS ($DT \rightarrow the$)

the rat ate the NN

Apply rule with NN on LHS ($NN \rightarrow cheese$)

the rat ate the cheese

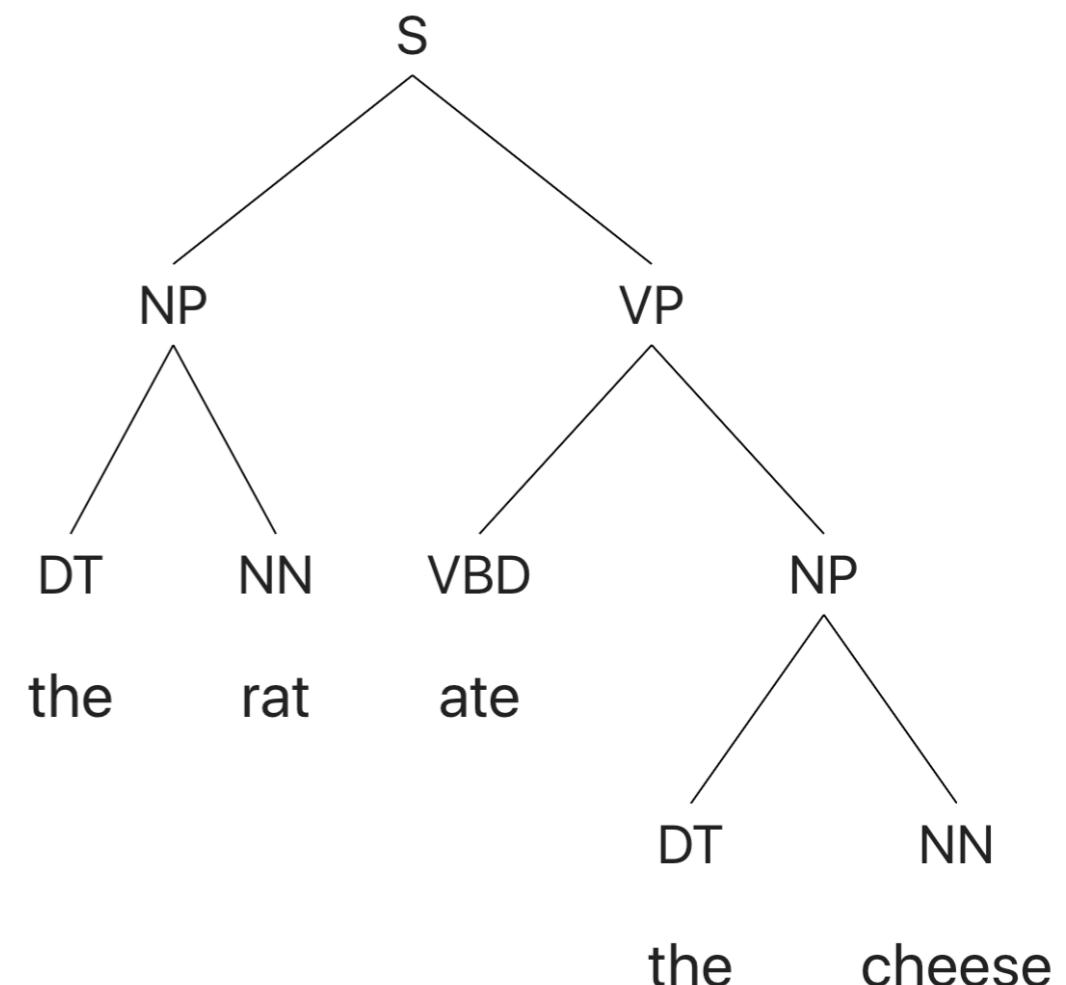
S → NP VP
NP → DT NN
VP → VBD NP
DT → *the*
NN → *rat*
NN → *cheese*
VBD → *ate*

No non-terminals left, we're done!

CFG Trees

- Generation corresponds to a **syntactic tree**
 - Non-terminals are **internal nodes**
 - Terminals are **leaves**

(S (NP (DT the)
 (NN rat))
(VP (VBG ate)
 (NP (DT the)
 (NN cheese))))



- CFG parsing is the **reverse** process (sentence → tree)

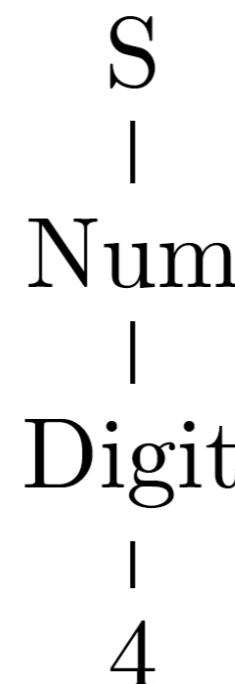
A CFG for Arithmetic Expressions

$$S \rightarrow S \text{ OP } S \mid \text{NUM}$$
$$\text{OP} \rightarrow + \mid - \mid \times \mid \div$$
$$\text{NUM} \rightarrow \text{NUM DIGIT} \mid \text{DIGIT}$$
$$\text{DIGIT} \rightarrow 0 \mid 1 \mid 2 \mid \dots \mid 9$$

- S = starting symbol
- \mid = operator OR
- Recursive, NUM and S can produce themselves

Parsing

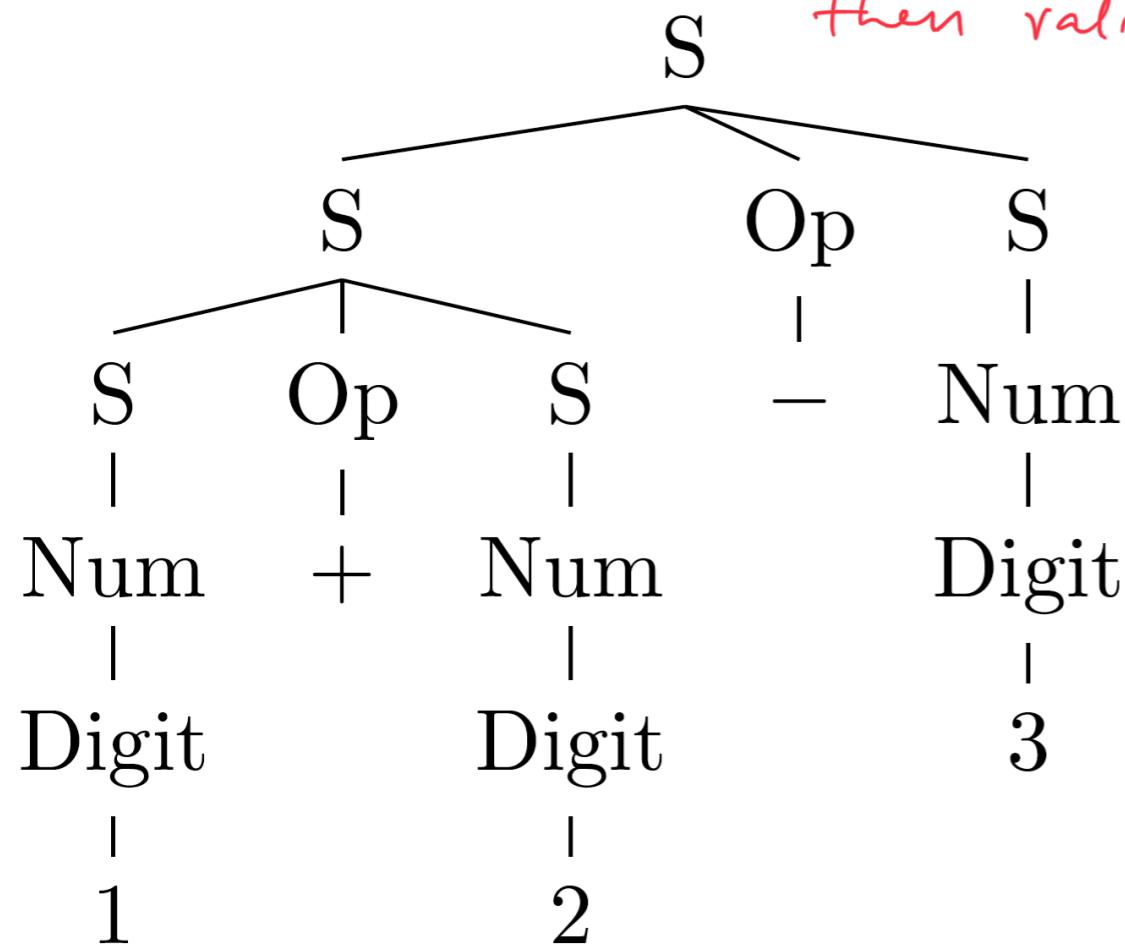
- Is '4' a valid string?

$$S \rightarrow S \text{ OP } S \mid \text{NUM}$$
$$\text{OP} \rightarrow + \mid - \mid \times \mid \div$$
$$\text{NUM} \rightarrow \text{NUM DIGIT} \mid \text{DIGIT}$$
$$\text{DIGIT} \rightarrow 0 \mid 1 \mid 2 \mid \dots \mid 9$$


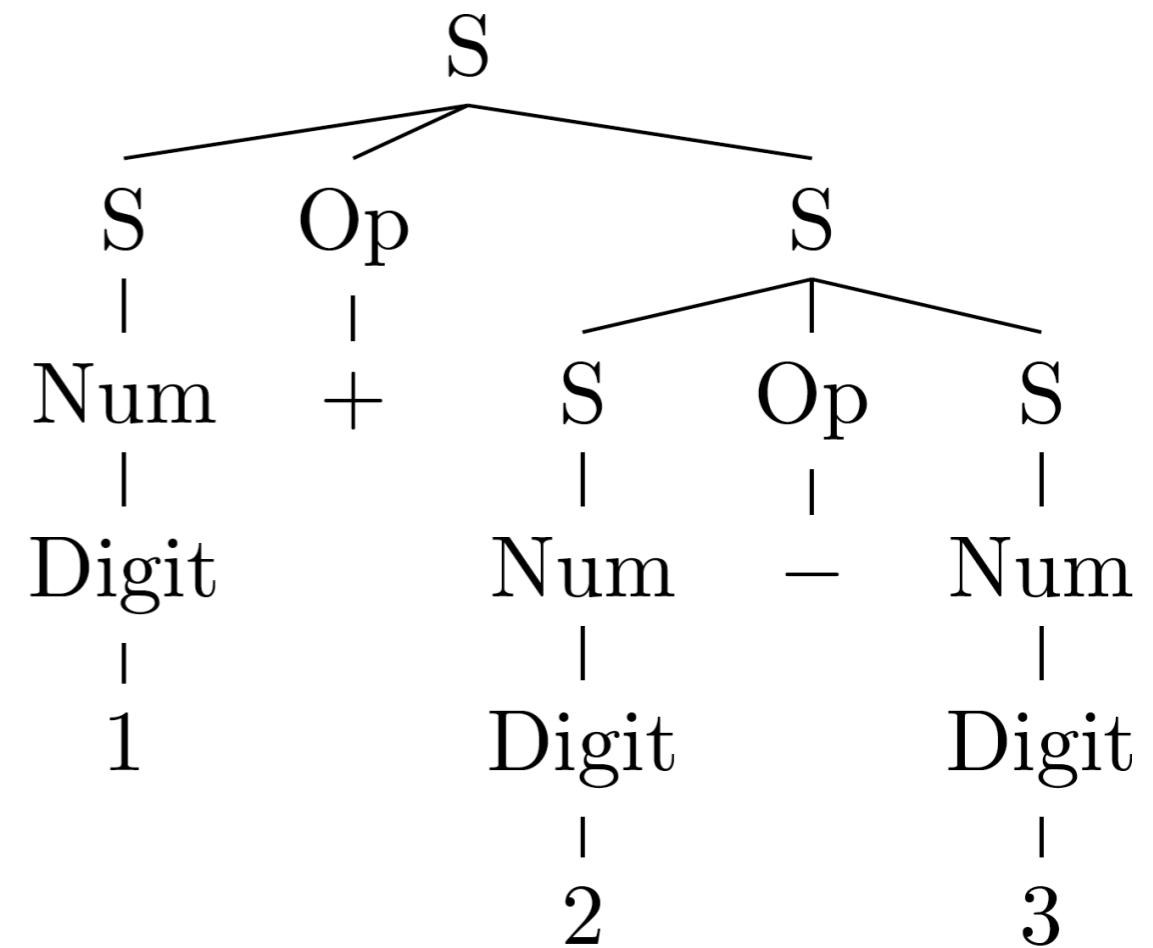
Parsing

- Is '1+2-3' a valid string?

*↳ find ≥ 1 parsing tree
then valid*



$$\begin{aligned}
 S &\rightarrow S \text{ OP}(S \mid \text{NUM}) \\
 \text{OP} &\rightarrow + \mid - \mid \times \mid \div \\
 \text{NUM} &\rightarrow \text{NUM DIGIT} \mid \text{DIGIT} \\
 \text{DIGIT} &\rightarrow 0 \mid 1 \mid 2 \mid \dots \mid 9
 \end{aligned}$$



CYK Algorithm

CYK Algorithm

- Bottom-up parsing
- Tests whether a string is valid given a CFG,
without enumerating all possible parses
- Core idea: form small constituents first, and merge
them into larger constituents
- Requirement: CFGs must be in **Chomsky Normal
Forms**

Convert to Chomsky Normal Form

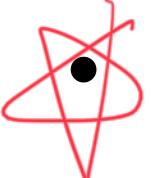
- Change grammar so all rules of form:
 - ▶ $A \rightarrow BC$
 - ▶ $A \rightarrow a$
- Convert rules of form $A \rightarrow Bc$ into:
 - ▶ $A \rightarrow BX$
 - ▶ $X \rightarrow c$

Convert to Chomsky Normal Form

- Convert rules $A \rightarrow B C D$ into:
 - ▶ $A \rightarrow B Y$
 - ▶ $Y \rightarrow C D$
 - ▶ E.g. $VP \rightarrow VP NP NP$
for ditransitive cases, “*sold [her] [the book]*”
- X, Y are new symbols we have introduced

Convert to Chomsky Normal Form

- CNF **disallows unary rules**, $A \rightarrow B$.

 • Imagine $NP \rightarrow S$; and $S \rightarrow NP \dots$ leads to infinitely many trees with same yield.

} Why Chomsky Normal Form?
:: CYK 保証 唯一解.

- Replace RHS non-terminal **with its productions**

- $A \rightarrow B, B \rightarrow cat, B \rightarrow dog$
- $A \rightarrow cat, A \rightarrow dog$



The CYK Parsing Algorithm

- Convert grammar to Chomsky Normal Form (CNF)
- Fill in a parse table (left to right, bottom to top)
- Use table to derive parse
- S in top right corner of table = success!
- Convert result back to original grammar

we	eat	sushi	with	chopsticks
[0,1]	[0,2]	[0,3]	[0,4]	[0,5]
S → NP VP	[1,2]	[1,3]	[1,4]	[1,5]
NP → NP PP				
PP → IN NP				
VP → V NP				
VP → VP PP		[2,3]	[2,4]	[2,5]
NP → we				
NP → sushi				
NP → chopsticks				
IN → with			[3,4]	[3,5]
V → eat				[4,5]

we	eat	sushi	with	chopsticks
NP [0,1]				
	V [1,2]			
$S \rightarrow NP VP$ $NP \rightarrow NP PP$ $PP \rightarrow IN NP$ $VP \rightarrow V NP$ $VP \rightarrow VP PP$ $NP \rightarrow we$ $NP \rightarrow sushi$ $NP \rightarrow chopsticks$ $IN \rightarrow with$ $V \rightarrow eat$	[1,3]	NP [2,3]	IN [3,4]	
				NP [4,5]

convert to non-terminal

left \rightarrow right
bottom \rightarrow up

$S \rightarrow NP VP$

$NP \rightarrow NP PP$

$PP \rightarrow IN NP$

$VP \rightarrow V NP$

$VP \rightarrow VP PP$

$NP \rightarrow we$

$NP \rightarrow sushi$

$NP \rightarrow chopsticks$

$IN \rightarrow with$

$V \rightarrow eat$

we	eat	sushi	with	chopsticks
NP	\emptyset <i>parent TO produce NP & V \rightarrow null</i>			
[0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	V			
	[1,2]	[1,3]	[1,4]	[1,5]
		NP		
		[2,3]	[2,4]	[2,5]
			IN	
			[3,4]	[3,5]
				NP
				[4,5]

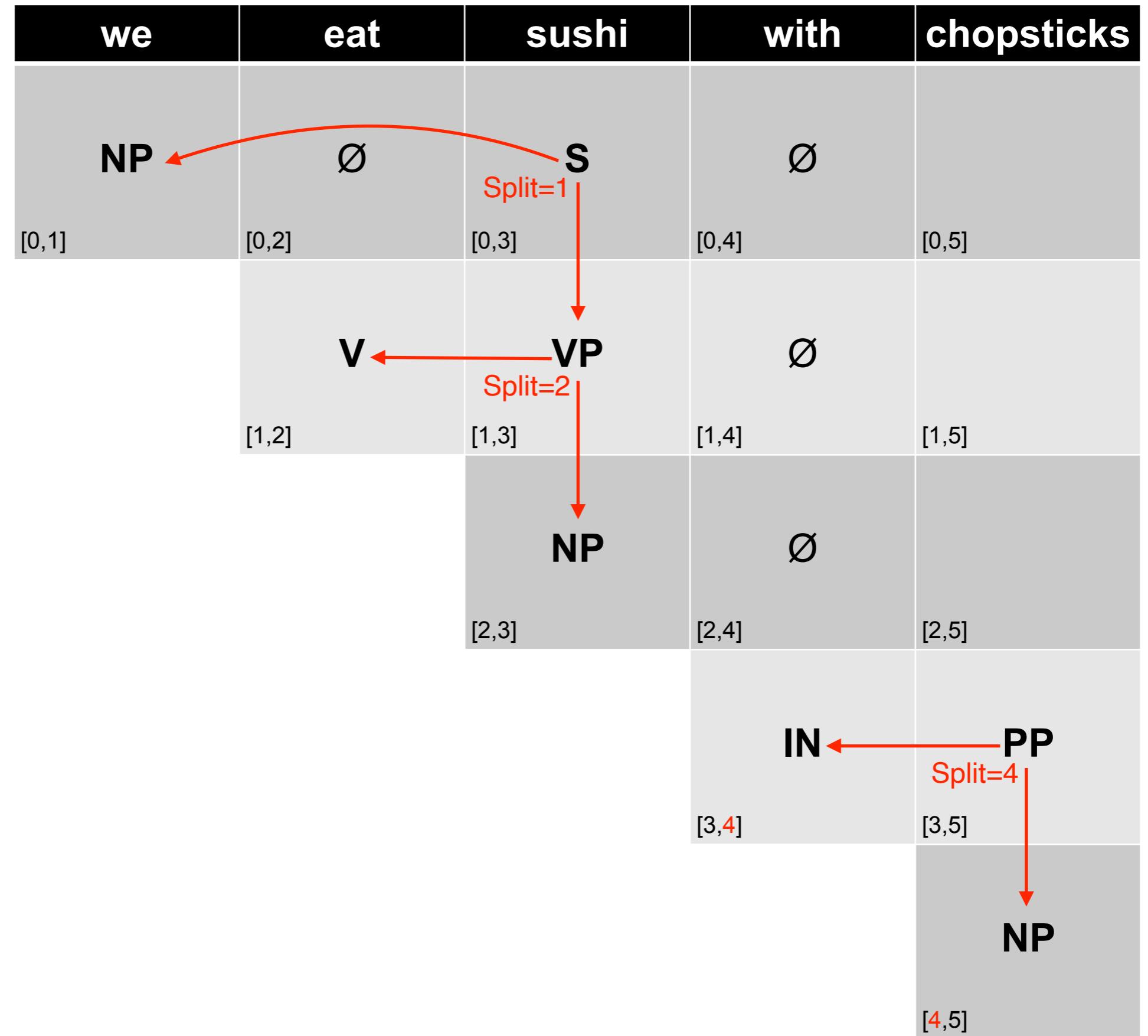
we	eat	sushi	with	chopsticks
NP	\emptyset			
[0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	V ← VP → Split=2 [1,2] [1,3]	Index that separate "eat" & "sushi" NP [2,3]	[1,4]	[1,5]
S → NP VP NP → NP PP PP → IN NP VP → V NP VP → VP PP NP → we NP → sushi NP → chopsticks IN → with V → eat		[2,4]	[3,4]	[3,5]
			IN	
				NP [4,5]

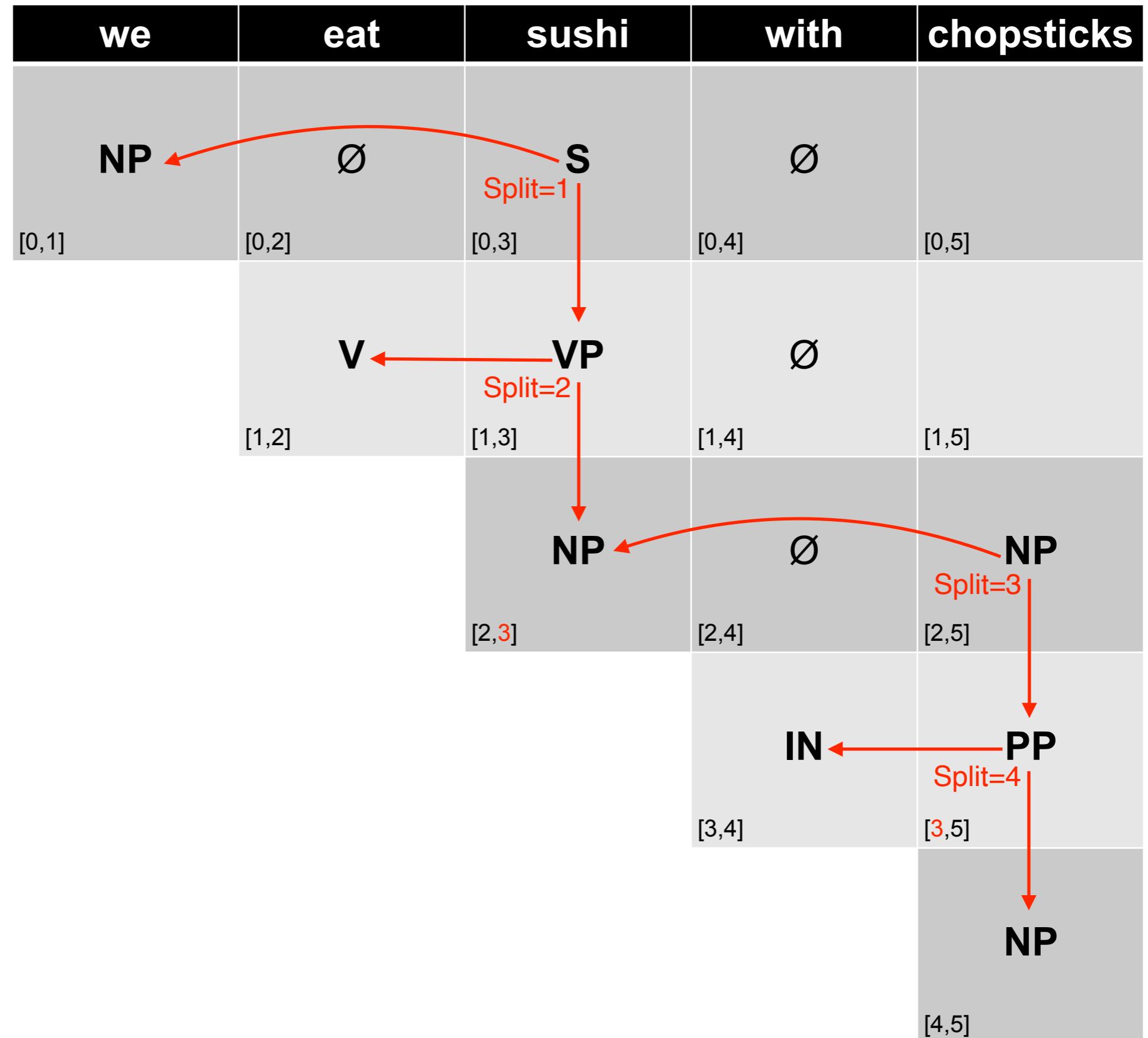
we	eat	sushi	with	chopsticks
NP [0,1]	\emptyset [0,2]	S Split=1 VP Split=2 NP [2,3]	[0,4]	[0,5]
$S \rightarrow NP VP$ NP → NP PP PP → IN NP VP → V NP VP → VP PP NP → we NP → sushi NP → chopsticks IN → with V → eat	[1,2]	[1,3]	[1,4]	[1,5]
			IN [3,4]	[2,5]
			NP [4,5]	

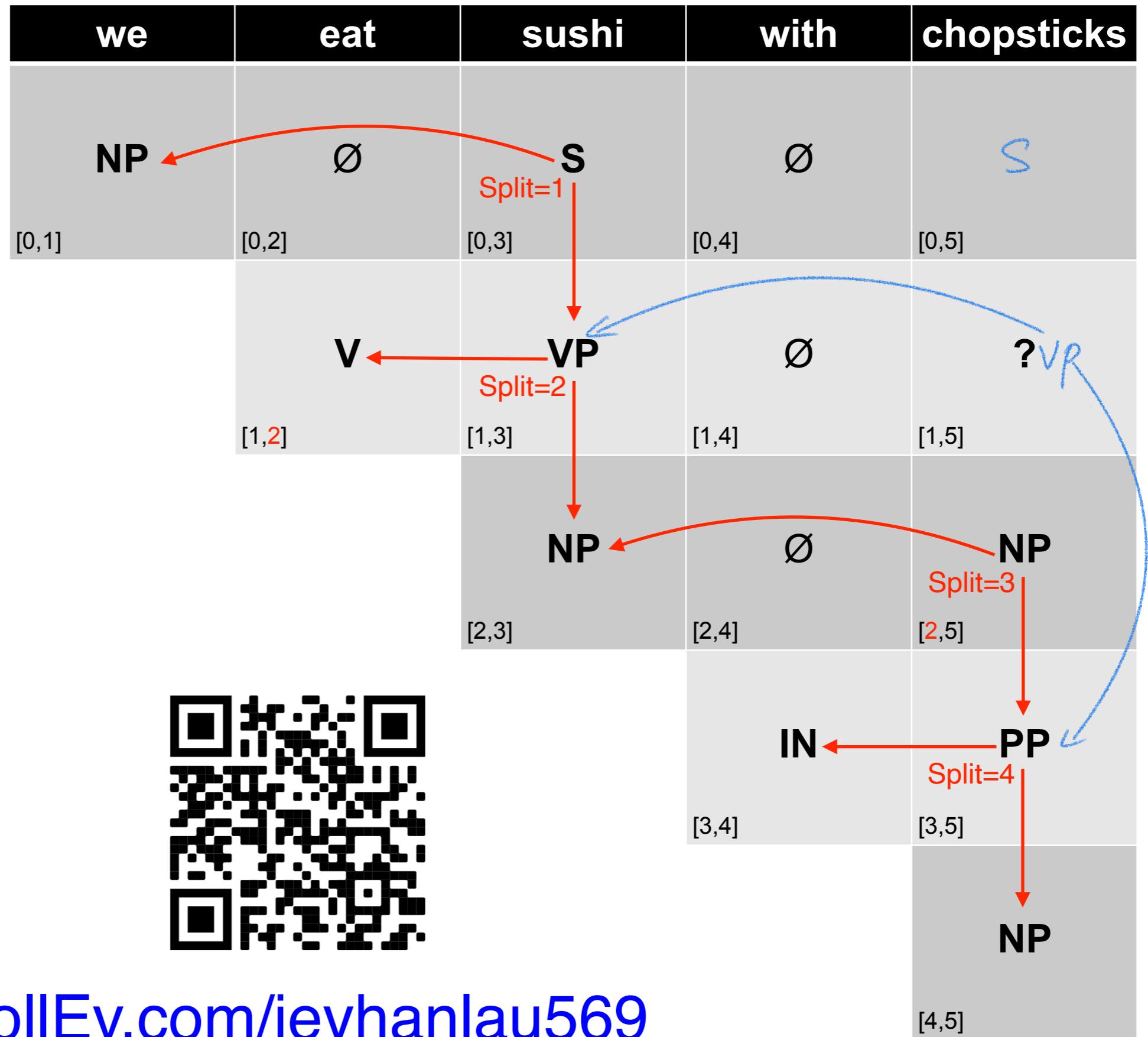
Only consider pairs that share the split.

we	eat	sushi	with	chopsticks
NP [0,1]	Ø [0,2]	S [0,3] Split=1	Ø [0,4]	
		V [1,2] VP [1,3] Split=2	Ø [1,4]	[1,5]
		NP [2,3]	Ø [2,4]	[2,5]
			IN [3,4]	[3,5]
				NP [4,5]

$S \rightarrow NP\ VP$
 $NP \rightarrow NP\ PP$
 $PP \rightarrow IN\ NP$
 $VP \rightarrow V\ NP$
 $VP \rightarrow VP\ PP$
 $NP \rightarrow we$
 $NP \rightarrow sushi$
 $NP \rightarrow chopsticks$
 $IN \rightarrow with$
 $V \rightarrow eat$

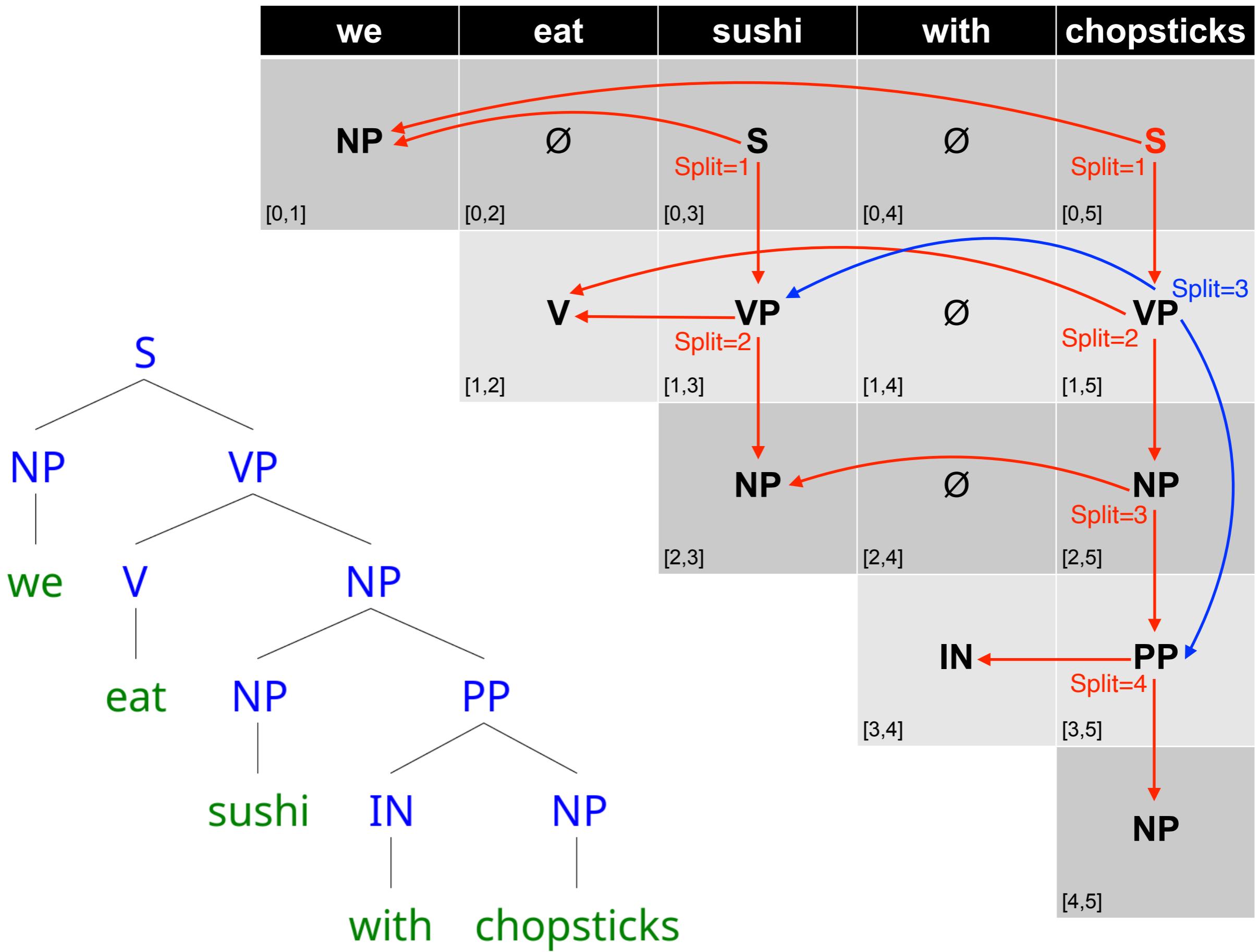


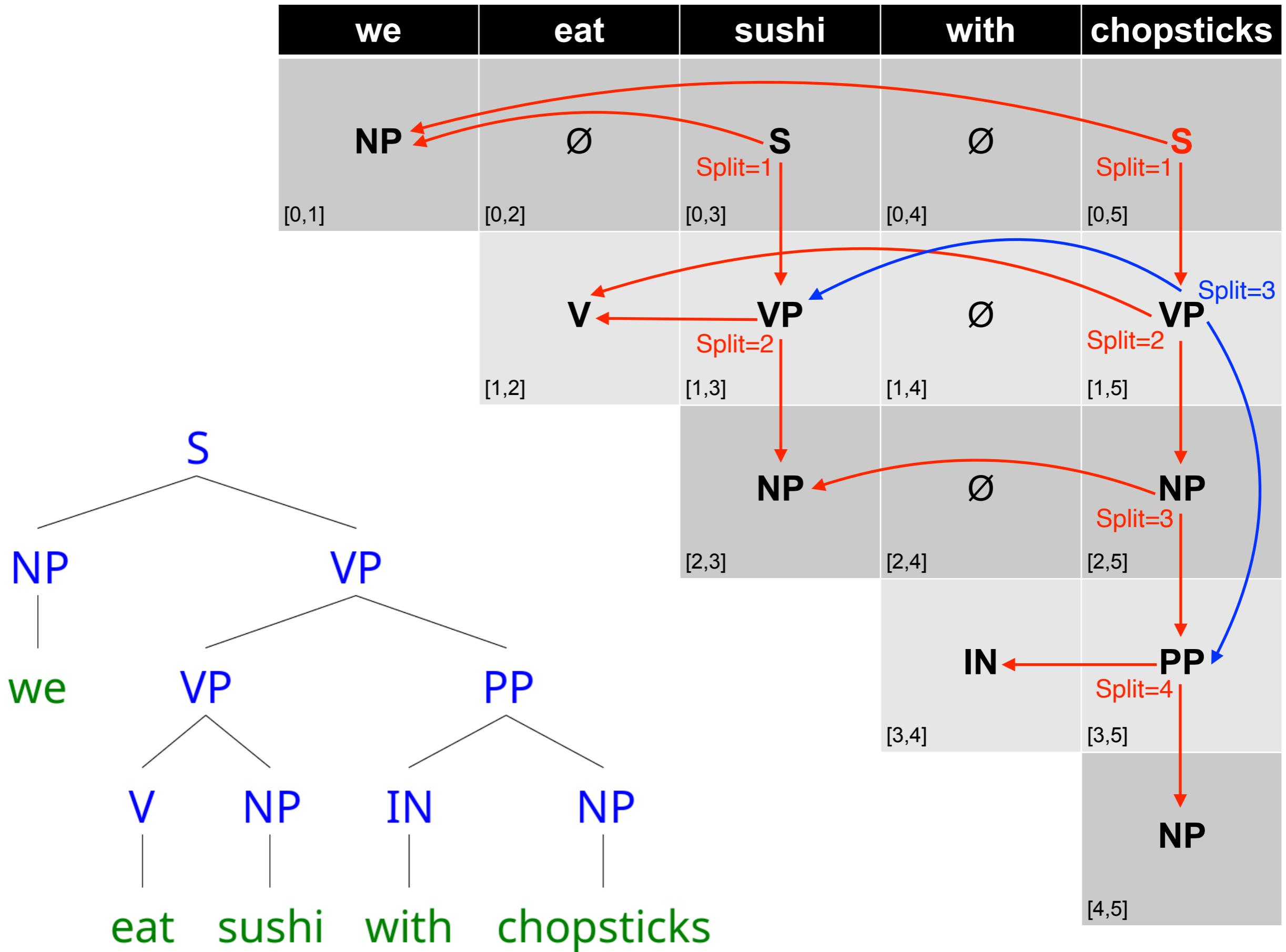




CYK: Retrieving the Parses

- S in the top-right corner of parse table indicates success
- To get parse(s), follow pointers back for each match





CYK Algorithm

```

function CKY-PARSE(words, grammar) returns table

for j  $\leftarrow$  from 1 to LENGTH(words) do
    for all {A | A  $\rightarrow$  words[j]  $\in$  grammar} fill the diagonals (NP  $\rightarrow$  we)
        table[j - 1, j]  $\leftarrow$  table[j - 1, j]  $\cup$  A
    for i  $\leftarrow$  from j - 2 downto 0 do bottom to top
        for k  $\leftarrow$  i + 1 to j - 1 do going through the ‘splits’
            for all {A | A  $\rightarrow$  BC  $\in$  grammar and B  $\in$  table[i, k] and C  $\in$  table[k, j]} create the links if they are in the production rules
                table[i, j]  $\leftarrow$  table[i, j]  $\cup$  A

```

Figure 12.5 The CKY algorithm.

create the links if they are in the production rules

Representing English with CFGs

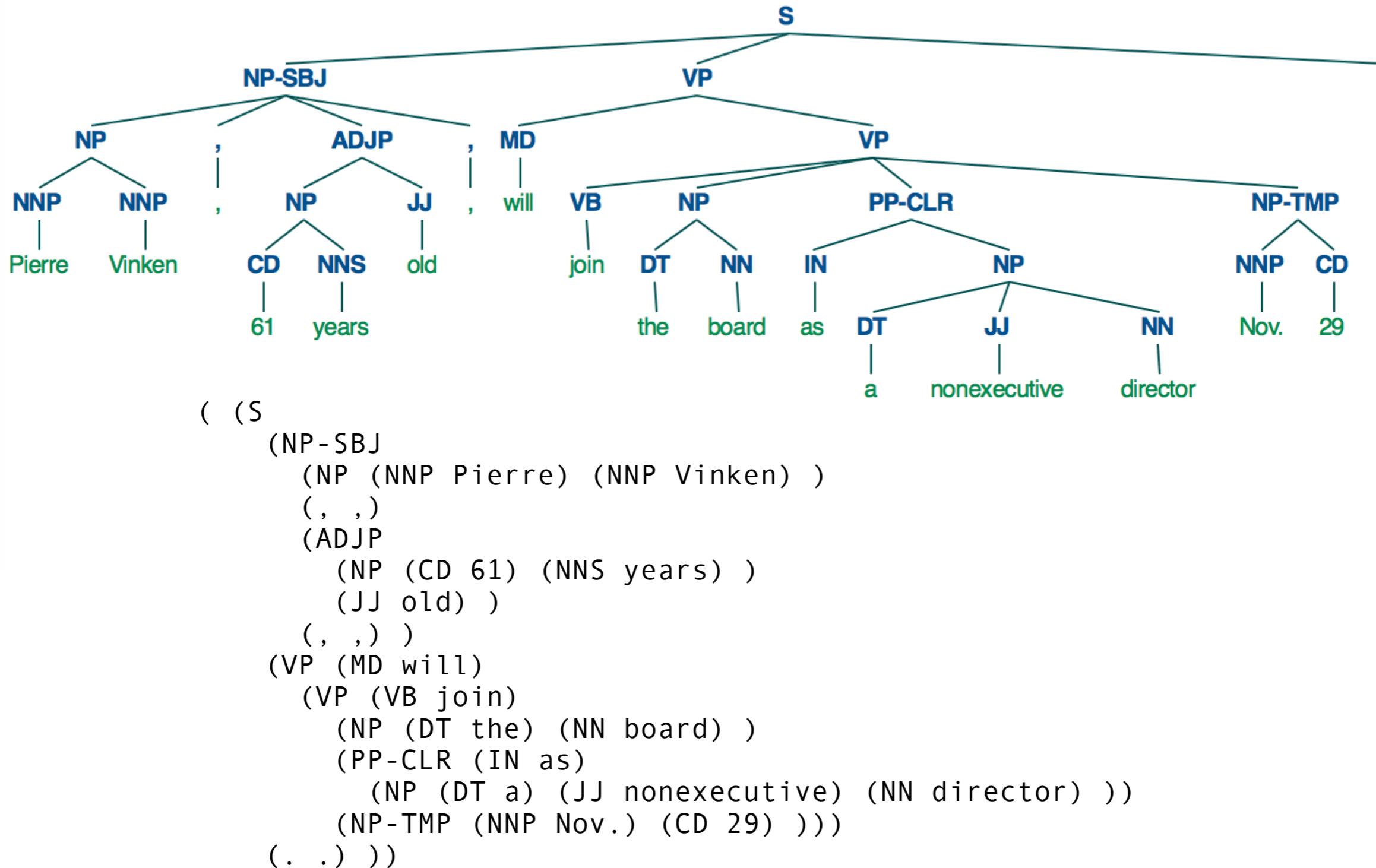
From Toy Grammars to Real Grammars

- Toy grammars with handful of productions good for demonstration or extremely limited domains
- For real texts, we need real grammars
- Many thousands of production rules

Key Constituents in Penn Treebank

- Sentence (S)
- Noun phrase (NP)
- Verb phrase (VP)
- Prepositional phrase (PP)
- Adjective phrase (AdjP)
- Adverbial phrase (AdvP)
- Subordinate clause (SBAR)

Example PTB/0001



Basic English Sentence Structures

- Declarative sentences ($S \rightarrow NP\ VP$)
 - *The rat ate the cheese*
- Imperative sentences ($S \rightarrow VP$)
 - *Eat the cheese!*
- Yes/no questions ($S \rightarrow VB\ NP\ VP$)
 - *Did the rat eat the cheese?*
- *Wh*-subject-questions ($S \rightarrow WH\ VP$)
 - *Who ate the cheese?*
- *Wh*-object-questions ($S \rightarrow WH\ VB\ NP\ VP$)
 - *What did the rat eat?*

English Noun Phrases

- Pre-modifiers
 - DT, CD, ADJP, NNP, NN
 - E.g. *the two very best Philly cheese steaks*
- Post-modifiers
 - PP, VP, SBAR
 - A delivery from Bob coming today that I don't want to miss

NP → DT? CD? ADJP? (NNINNP)+ PP* VP? SBAR?

Verb Phrases

- Auxiliaries
 - MD, AdvP, VB, TO
 - E.g *should really have tried to wait*

VP → (MD V BI TO) AdvP? VP

- Arguments and adjuncts
 - NP, PP, SBAR, VP, AdvP
 - E.g *told him yesterday that I was ready*
 - E.g. *gave John a gift for his birthday to make amends*

VP → VB NP? NP? PP* AdvP* VP? SBAR?

Other Constituents

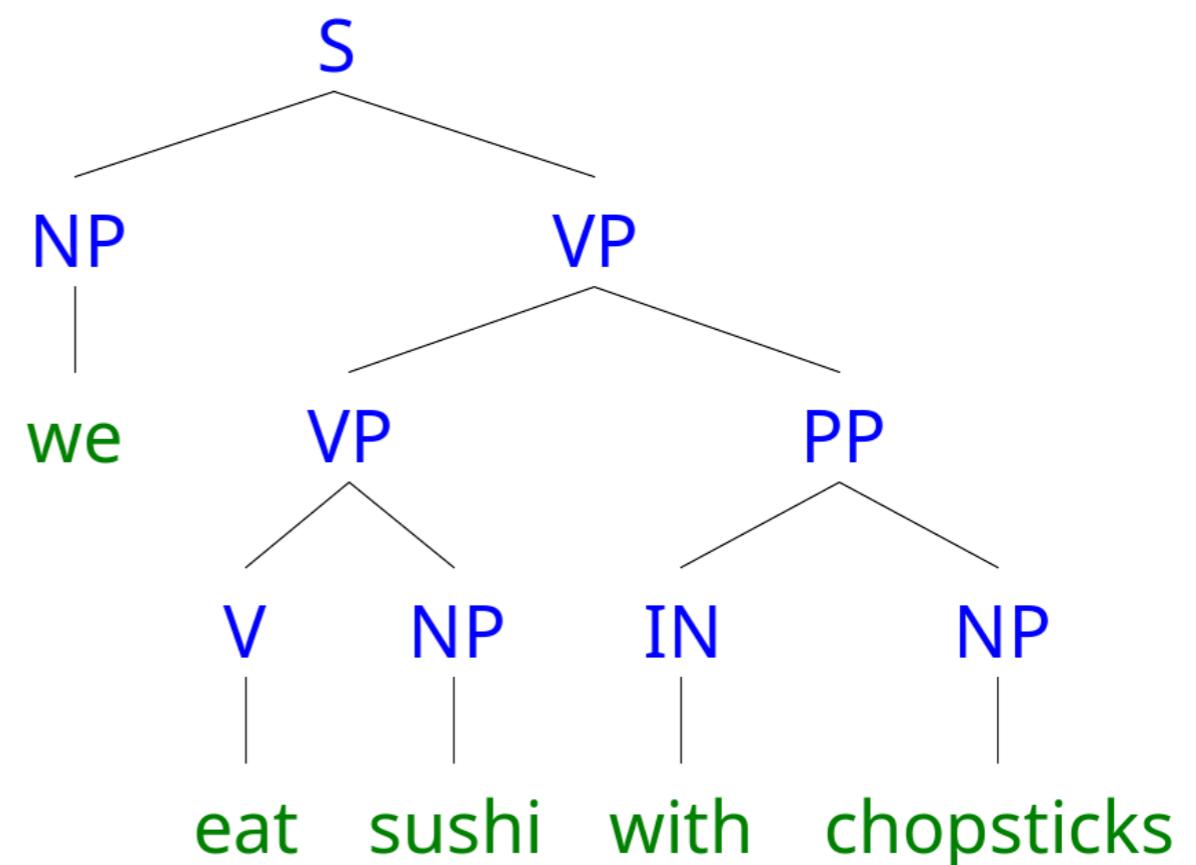
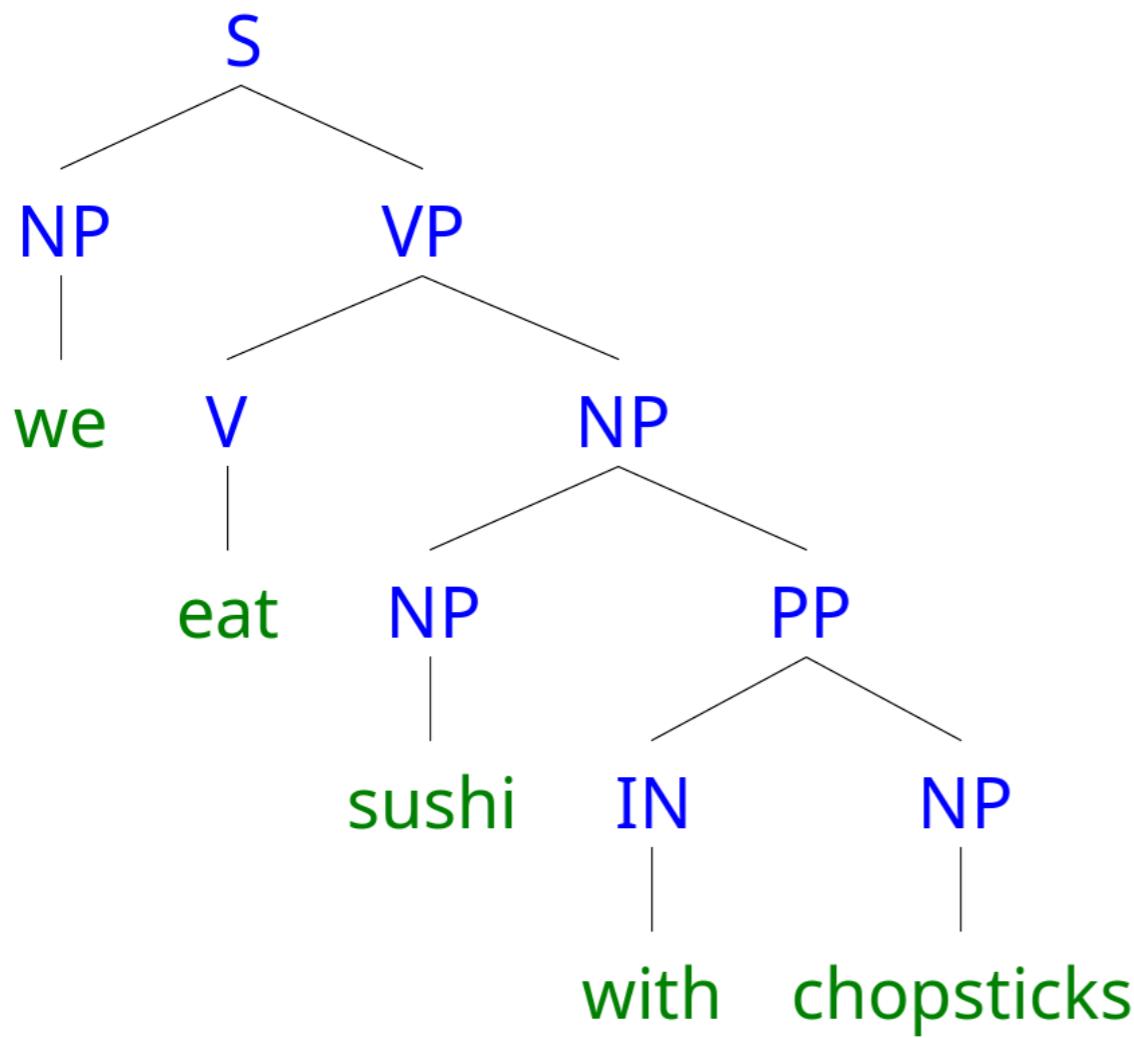
- Prepositional phrase
 - PP → IN NP *in the house*
- Adjective phrase
 - AdjP → (AdvP) JJ *really nice*
- Adverb phrase
 - AdvP → (AdvP) RB *not too well*
- Subordinate clause
 - SBAR → (IN) S *since I came here*
- Coordination
 - NP → NP CC NP; VP → VP CC VP; etc. *Jack and Jill*
- Complex sentences
 - S → S SBAR; S → SBAR S; etc. *if he goes, I'll go*

A Final Word

- Context-free grammars can represent most linguistic structures of natural languages
- There are relatively fast dynamic programming algorithms (CYK) to retrieve this structure

Parse Ambiguity

- But what about ambiguity? Often more than one tree can describe a string



Reading

- E18 Ch. 9.2, 10.1