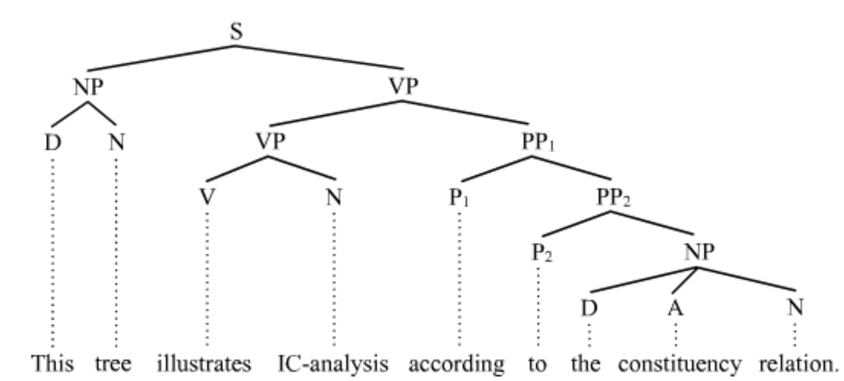
Context-Free Grammar

COMP90042

Natural Language Processing

Lecture 14



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 to use upper and lower-case to distinguish, or else “quotes” for 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

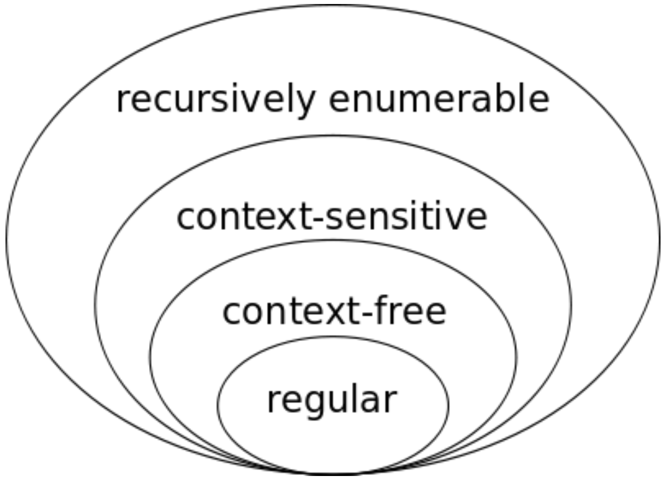
Why "Context Free"

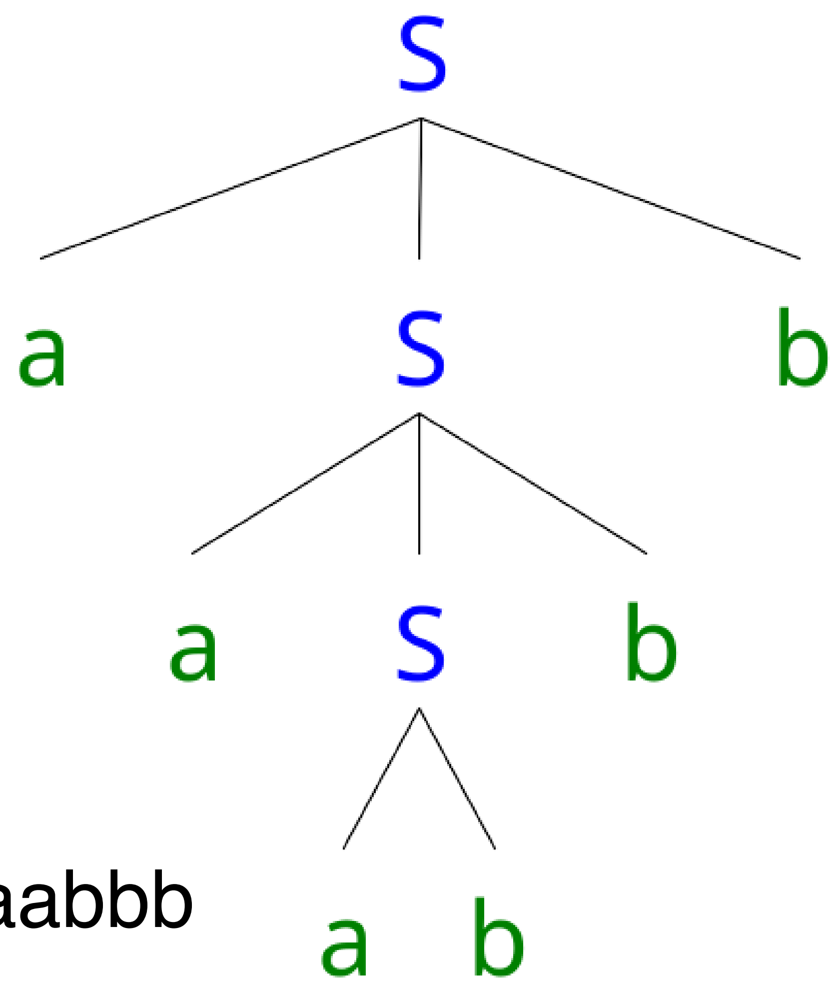
$$W \rightarrow X\ Y\ Z$$

- 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 for $a^n b^n$:
  - S → a S b
  - S → a b





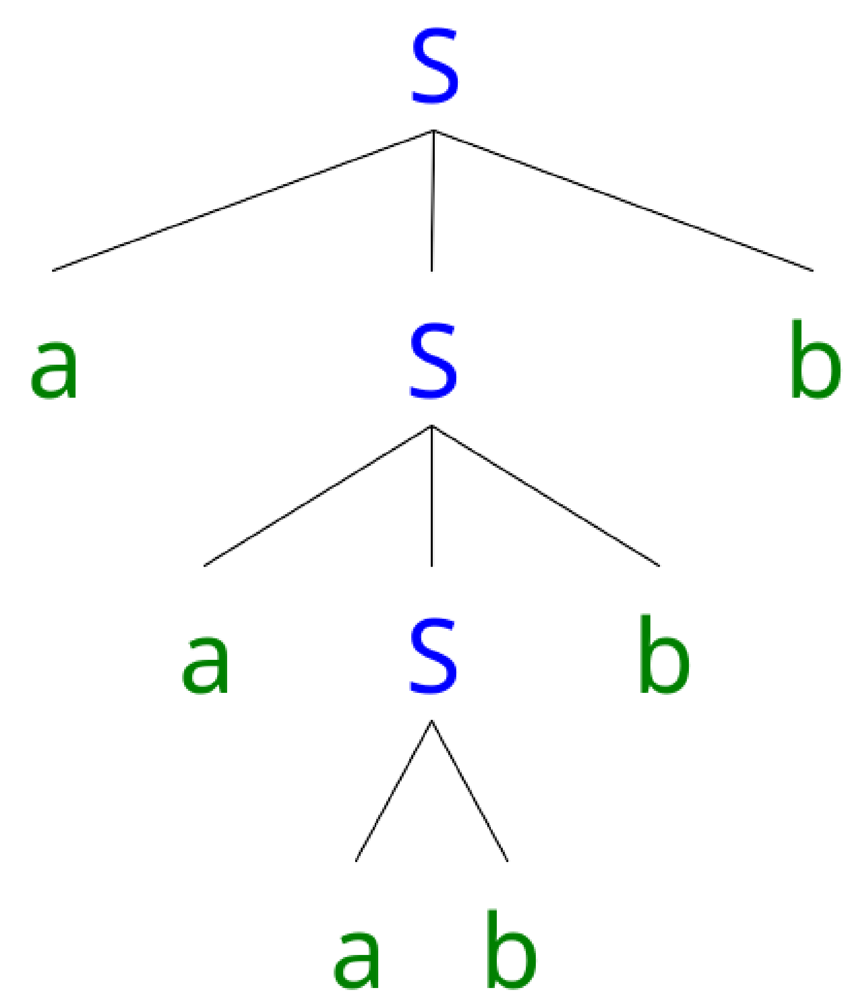
Parse tree for aaabbb

Parsing

- Given a string and production rules
- Produce a valid parse tree

  - S → a S b
  - S → a b

aaabbb



What This Means?

- If English can be represented with CFG:
  - we can build a "parser" to automatically judge whether a sentence is grammatical!
- But is natural language context-free?
- Not quite: cross-serial dependencies ($a^m b^n c^m d^n$)

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…

- Context-free representations strike a good balance:
  - CFG cover most syntactic patterns
  - CFG parsing is computational efficient
- We use CFG to describe a core fragment of English syntax

Syntactic Constituents

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

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
- Contrast: [brother brought], [bought a]

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

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* → *NP VP*), i.e substitute RHS

NP VP

Apply a rule with NP on LHS (*NP* → *DT NN*)

DT NN VP

Apply rule with DT on LHS (DT → *the*)

the NN VP

Apply rule with NN on LHS (NN → *rat*)

the rat VP

In each step we rewrite the left-most non-terminal

Generating Sentences with CFGs

Apply rule with VP on LHS (VP → VBD NP)

the rat VBD NP

Apply rule with VBD on LHS (VBD → *ate*)

the rat ate NP

Apply rule with NP on LHS (NP → DT NN)

the rat ate DT NN

Apply rule with DT on LHS (DT → *the*)

the rat ate the NN

Apply rule with NN on LHS (NN → *cheese*)

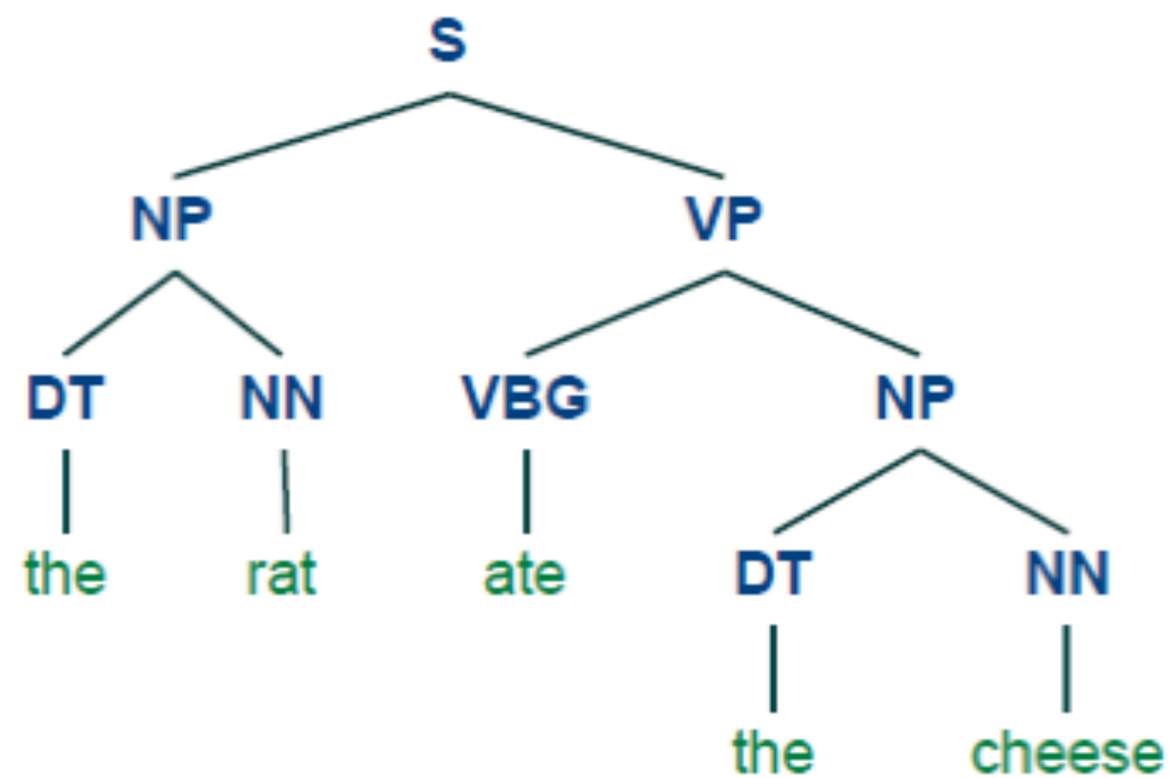
the rat ate the cheese

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))))
```



- Parsing is the **reverse** process

A CFG for Arithmetic Expressions

$$
\begin{aligned}
\textsc{S} &\rightarrow \textsc{S Op S} \mid \textsc{Num} \\
\textsc{Op} &\rightarrow + \mid - \mid \times \mid \div \\
\textsc{Num} &\rightarrow \textsc{Num Digit} \mid \textsc{Digit} \\
\textsc{Digit} &\rightarrow 0 \mid 1 \mid 2 \mid \ldots \mid 9
\end{aligned}
$$

- S = starting symbol
- ‘|’ = operator OR
- Recursive, NUM and S can produce themselves

Parsing

$$
\begin{aligned}
\textsc{S} &\rightarrow \textsc{S Op S} \mid \textsc{Num} \\
\textsc{Op} &\rightarrow + \mid - \mid \times \mid \div \\
\textsc{Num} &\rightarrow \textsc{Num Digit} \mid \textsc{Digit} \\
\textsc{Digit} &\rightarrow 0 \mid 1 \mid 2 \mid \ldots \mid 9
\end{aligned}
$$

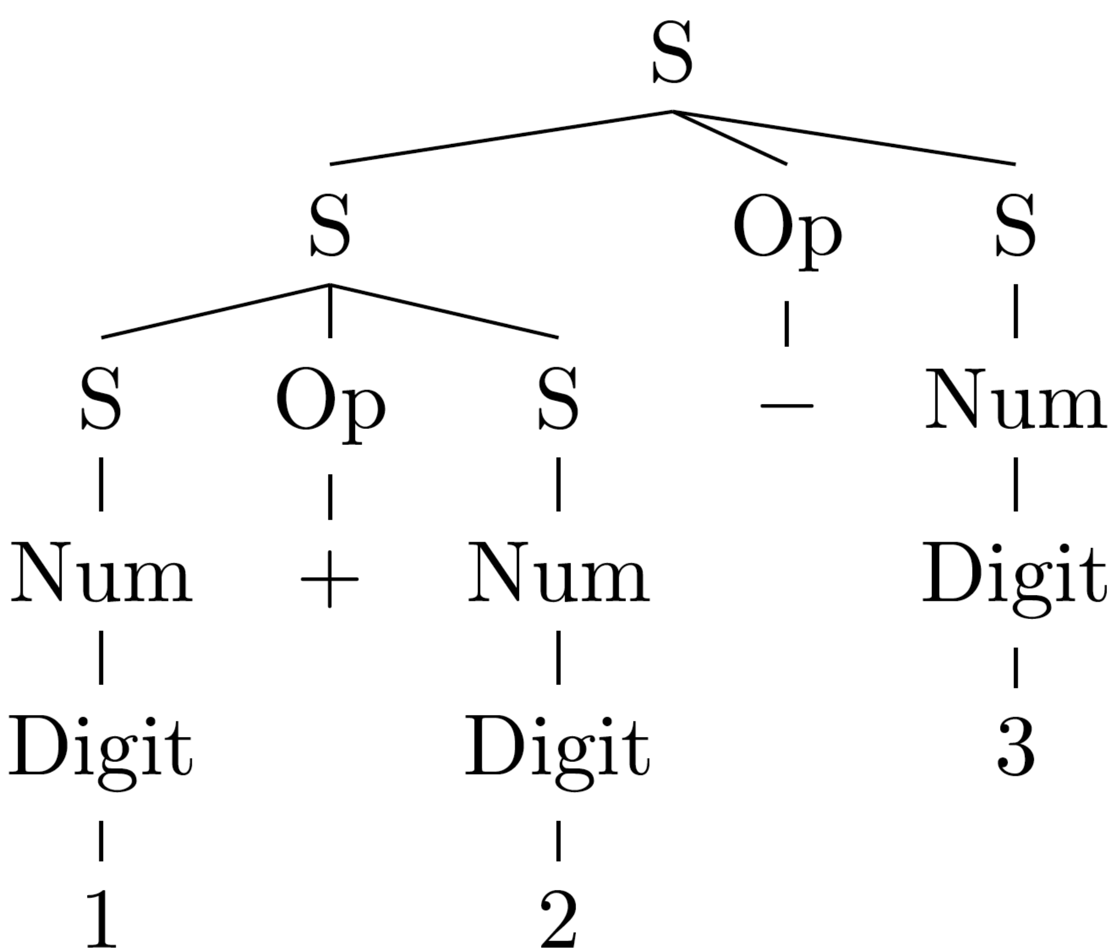
- Is ‘4’ a valid string?

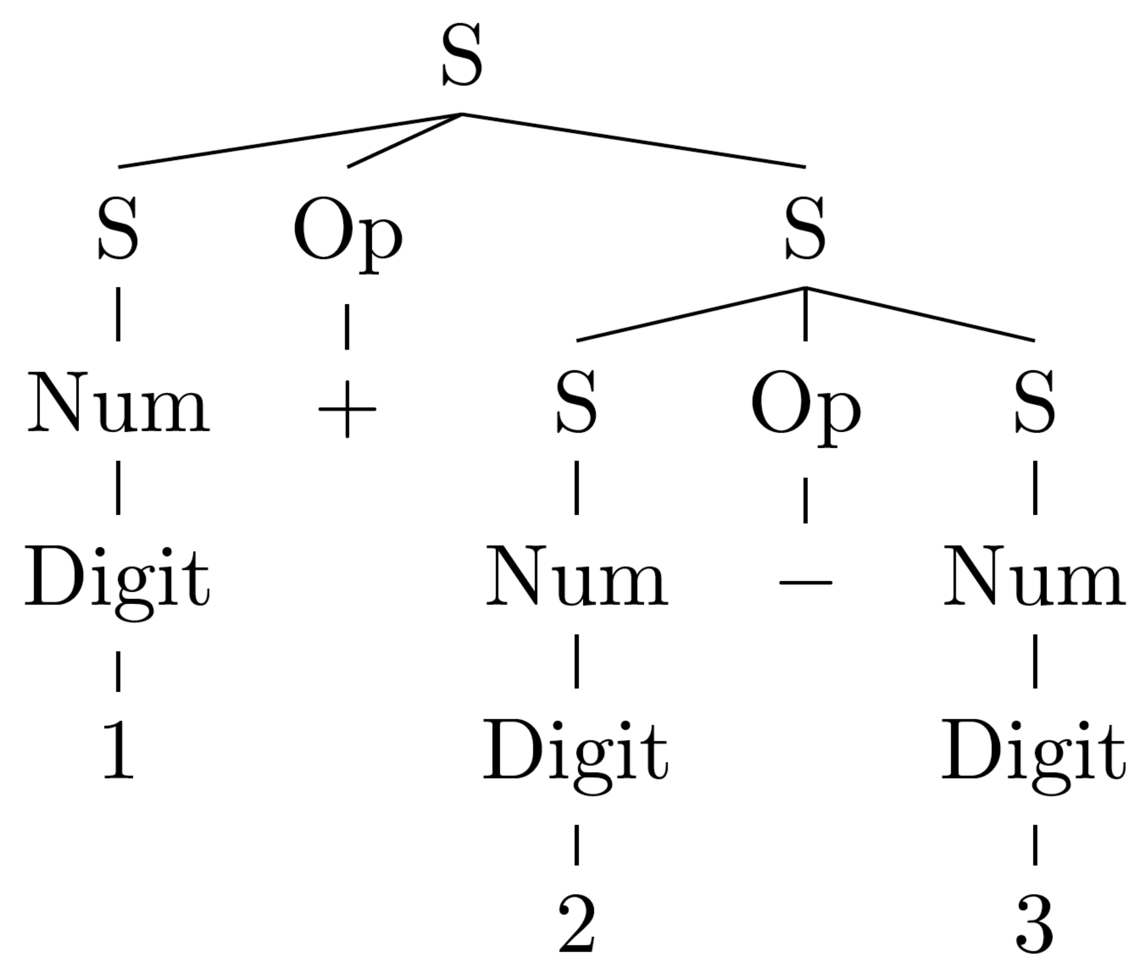
```
  S
  |
 Num
  |
Digit
  |
  4
```

Parsing

$$
\begin{aligned}
\textsc{S} &\rightarrow \textsc{S Op S} \mid \textsc{Num} \\
\textsc{Op} &\rightarrow + \mid - \mid \times \mid \div \\
\textsc{Num} &\rightarrow \textsc{Num Digit} \mid \textsc{Digit} \\
\textsc{Digit} &\rightarrow 0 \mid 1 \mid 2 \mid \ldots \mid 9
\end{aligned}
$$

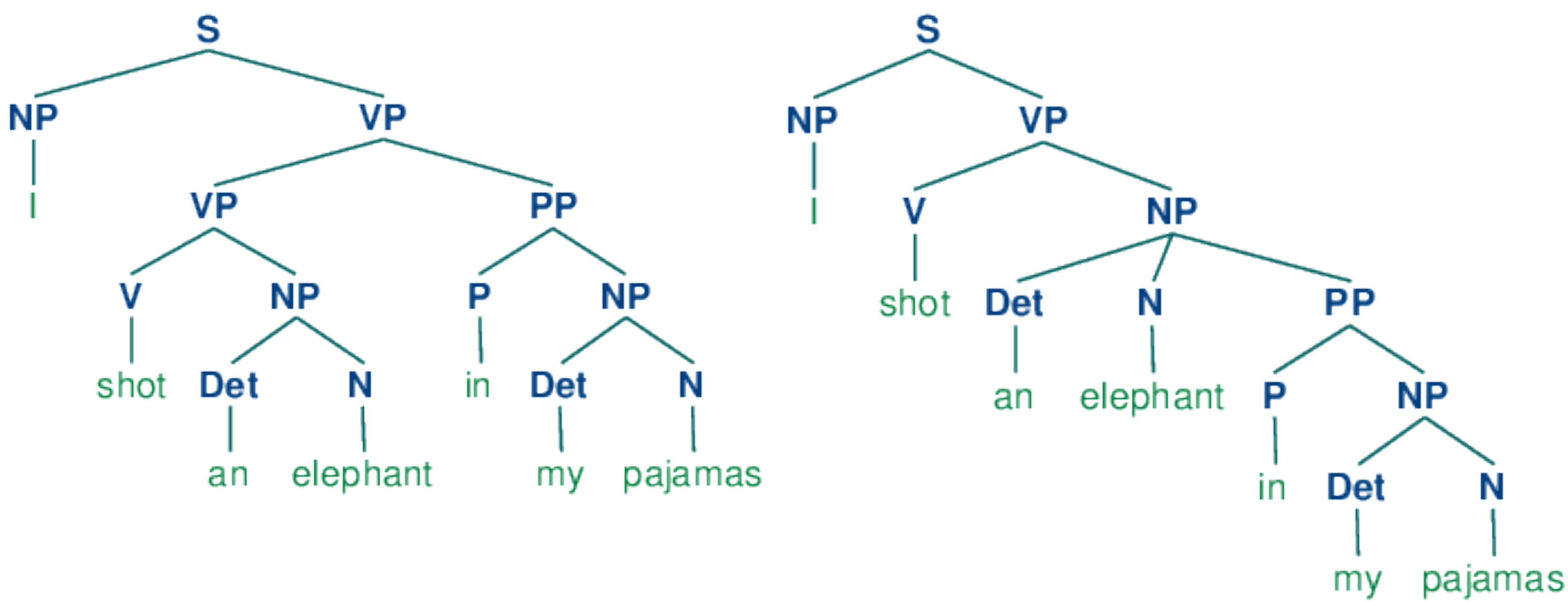
- Is ‘1+2-3’ a valid string?





Parse Ambiguity

- Often more than one tree can describe a string
- "*While hunting in Africa, I shot an elephant in my pajamas. How he got into my pajamas, I don't know.*"
  Animal Crackers (1930)



http://www.nltk.org/book/ch08.html

Parsing CFG

CYK Algorithm

- Bottom-up approach to parsing in CFG
- 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:
  - $A \rightarrow B\ C$
  - $A \rightarrow a$
- Convert rules of form $A \rightarrow B\ c$ into:
  - $A \rightarrow B\ X$
  - $X \rightarrow c$

Convert to Chomsky Normal Form

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

CNF (cont)

- CNF disallows unary rules, A → B.
- Imagine NP → S; and S → NP … leads to infinitely many trees with same yield.
- Replace RHS non-terminal with its productions
  - E.g convert A → B, B → 1, B → 2 into:
  - A → 1, A →2

The CYK Parsing Algorithm

- Convert grammar to Chomsky Normal Form (CNF)
- Fill in a parse table
- 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]
	[1,2]	[1,3]	[1,4]	[1,5]
		[2,3]	[2,4]	[2,5]
			[3,4]	[3,5]
				[4,5]

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

we	eat	sushi	with	chopsticks
NP [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]

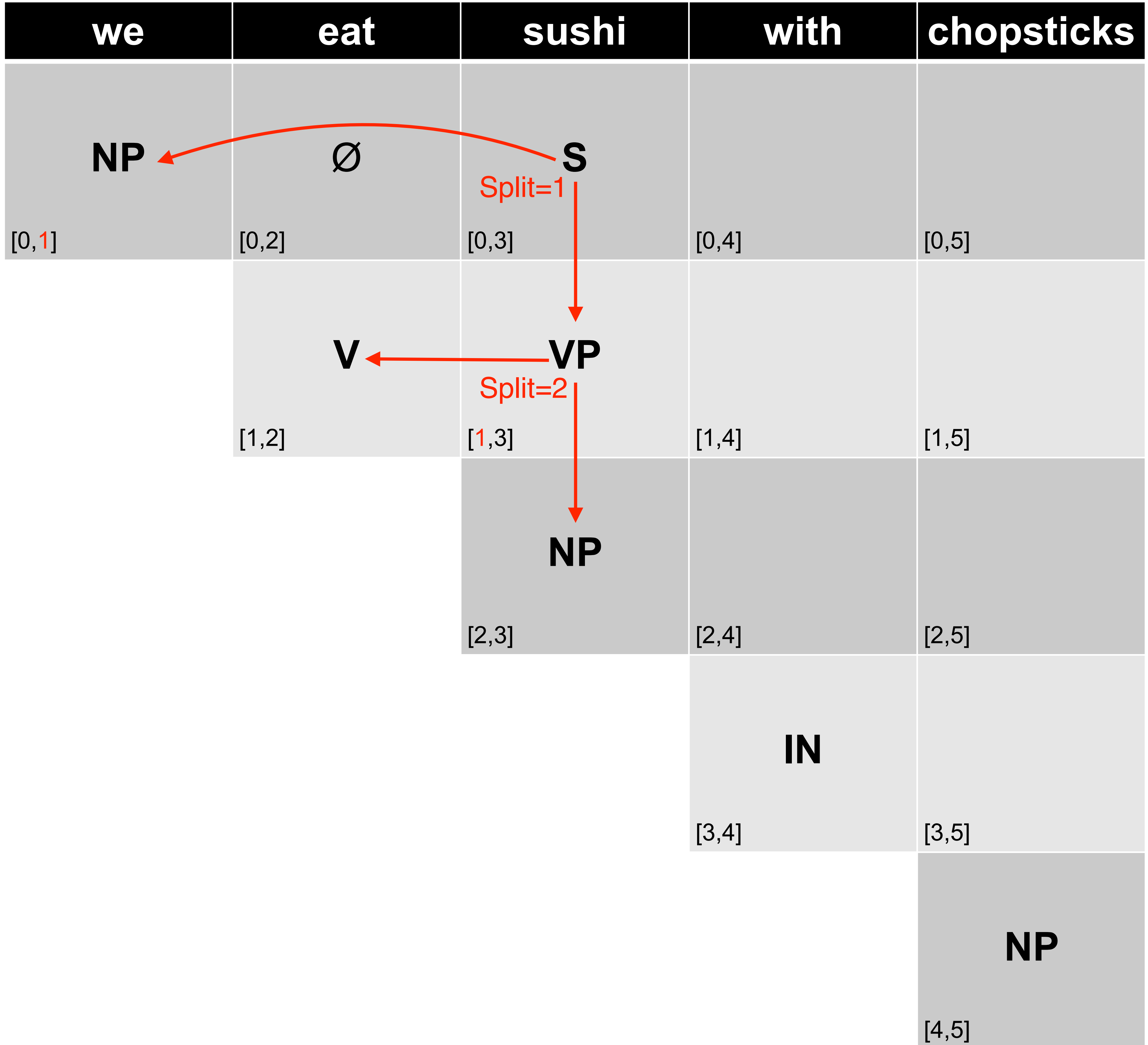
S → NP VP
NP → NP PP
PP → IN NP
VP → V NP
VP → VP PP
NP → we
NP → sushi
NP → chopsticks
IN → with
V → eat

we	eat	sushi	with	chopsticks
NP [0,1]	Ø [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]

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

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

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



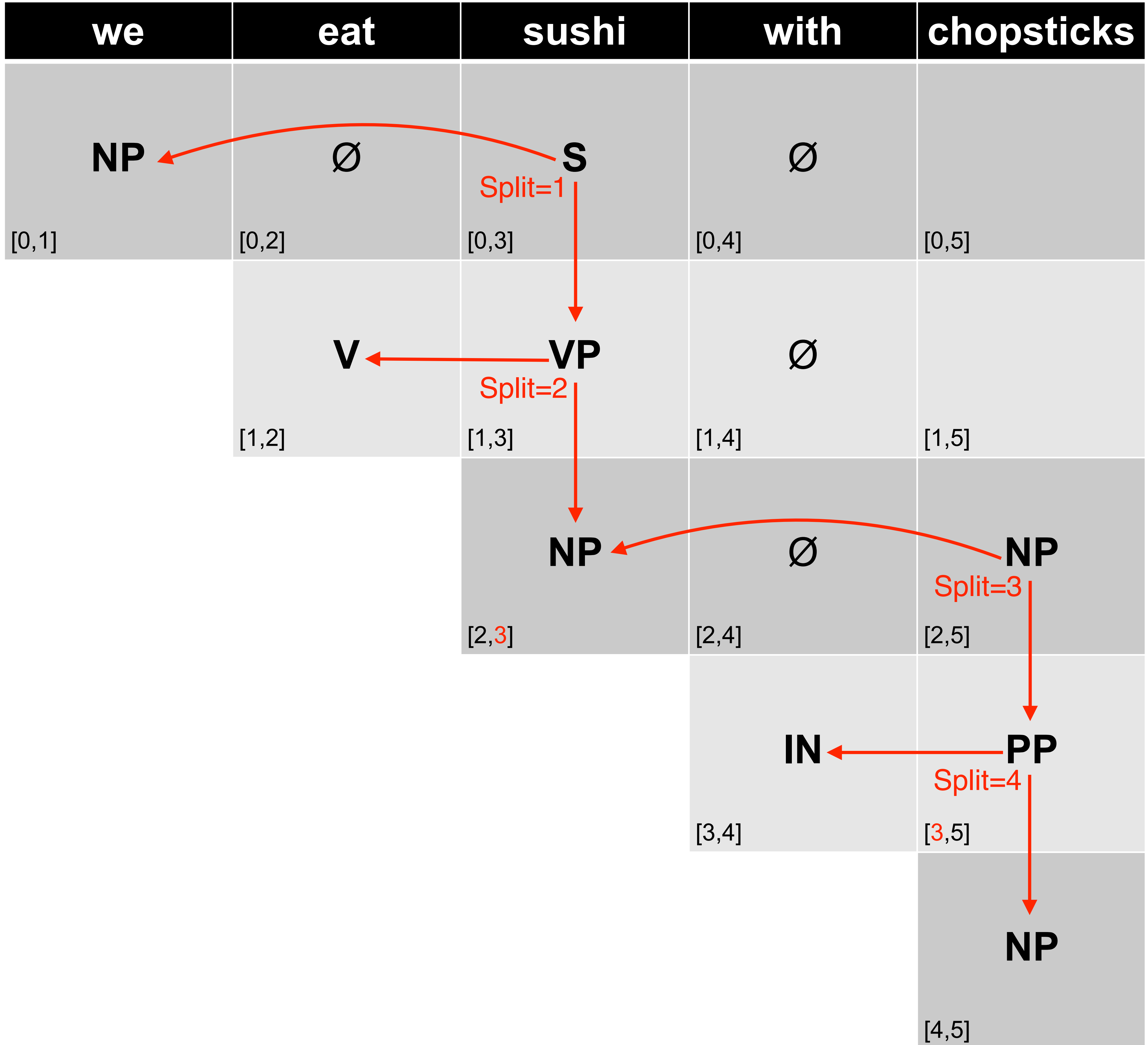
S → NP VP
NP → NP PP
PP → IN NP
VP → V NP
VP → VP PP
NP → we
NP → sushi
NP → chopsticks
IN → with
V → eat

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

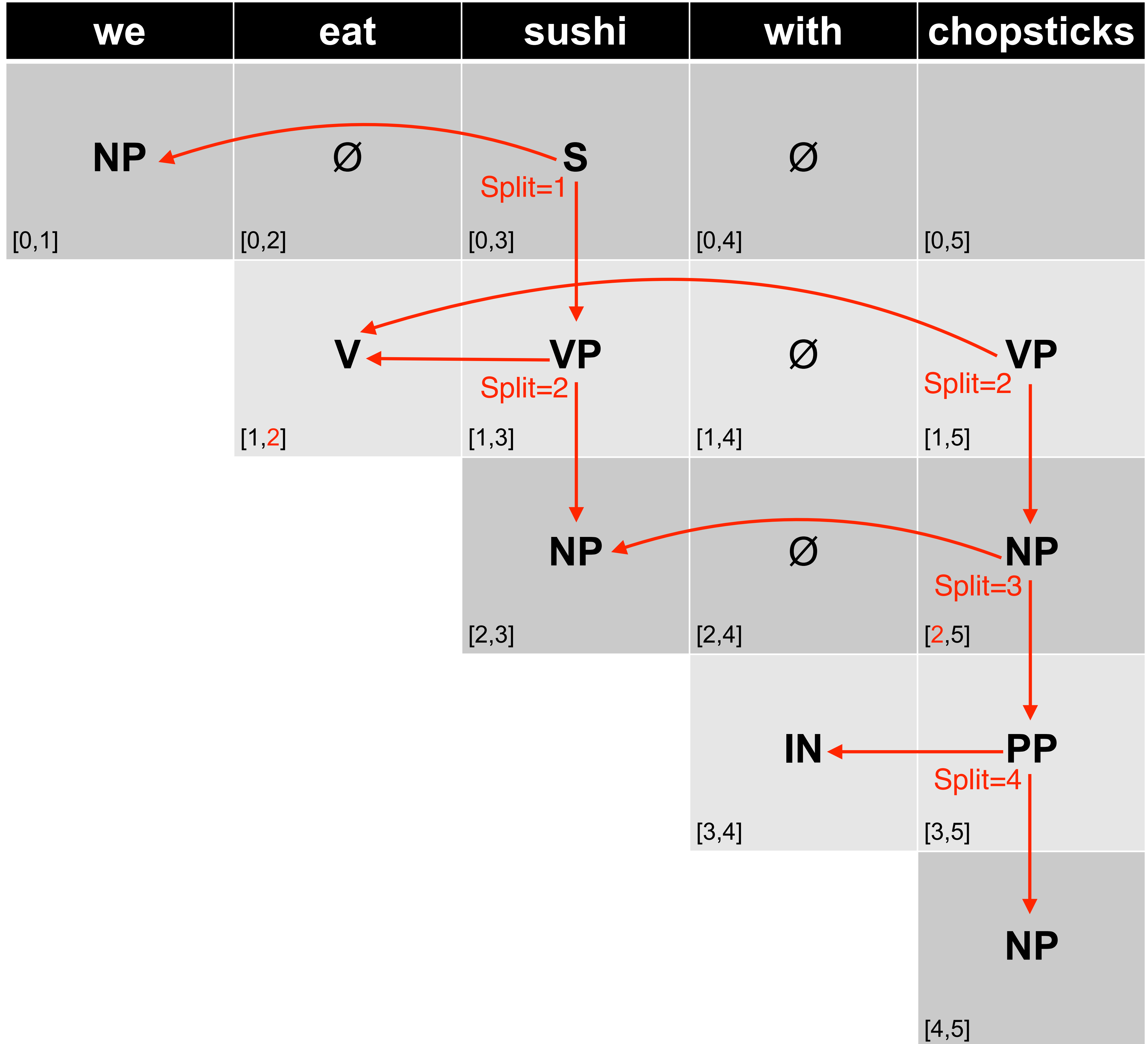
S → NP VP
NP → NP PP
PP → IN NP
VP → V NP
VP → VP PP
NP → we
NP → sushi
NP → chopsticks
IN → with
V → eat

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

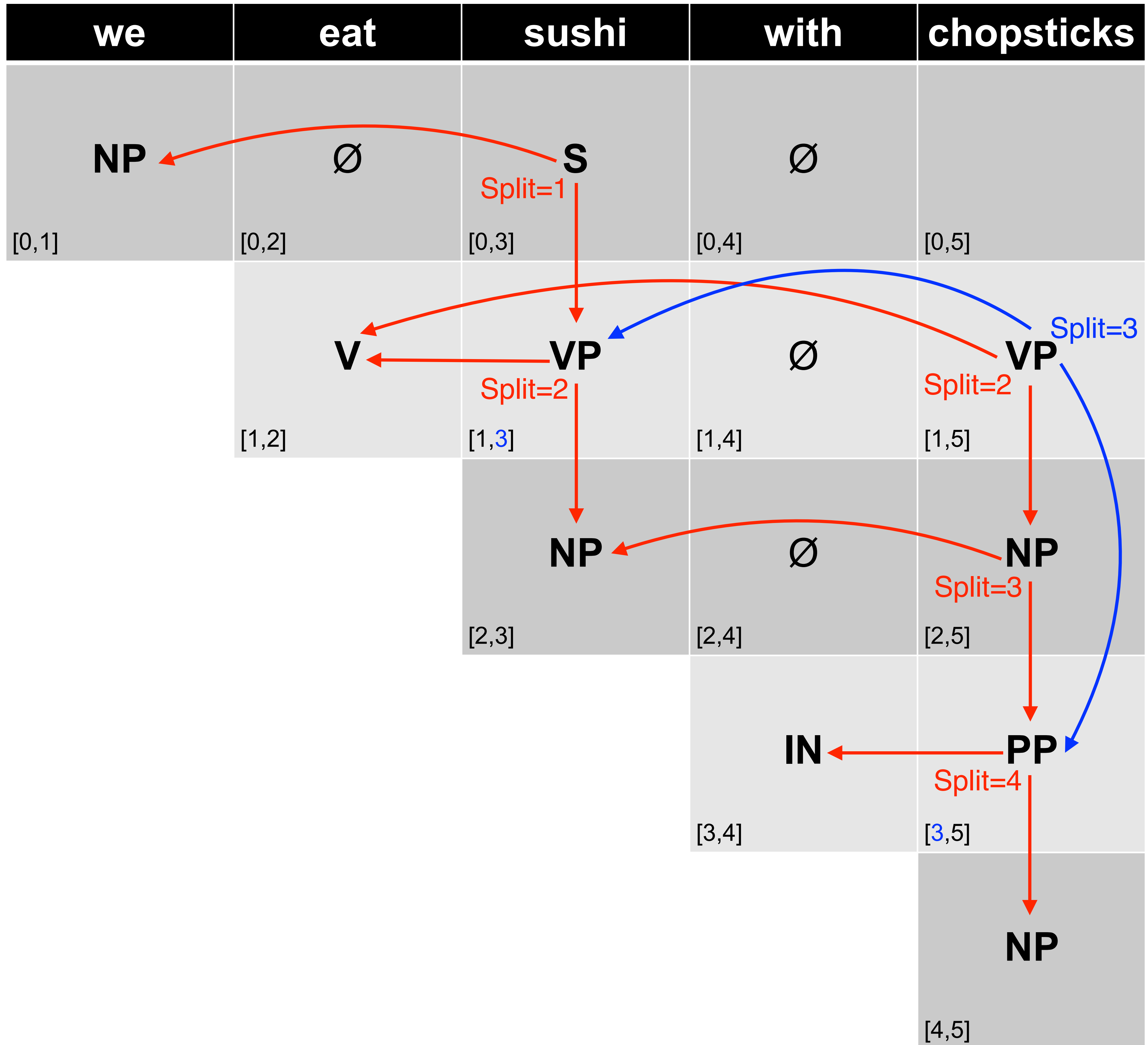
S → NP VP
NP → NP PP
PP → IN NP
VP → V NP
VP → VP PP
NP → we
NP → sushi
NP → chopsticks
IN → with
V → eat



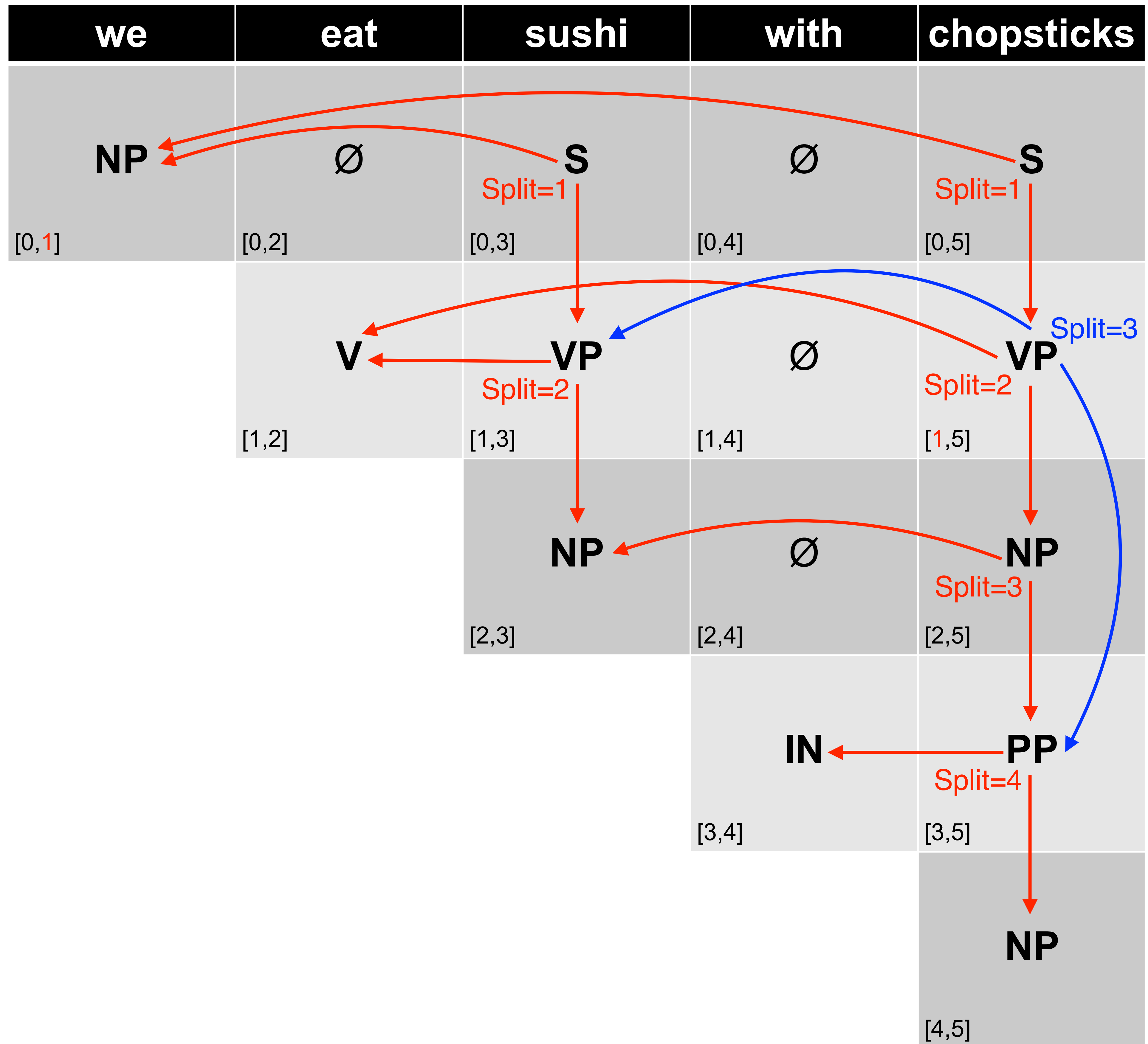
S → NP VP
NP → NP PP
PP → IN NP
VP → V NP
VP → VP PP
NP → we
NP → sushi
NP → chopsticks
IN → with
V → eat



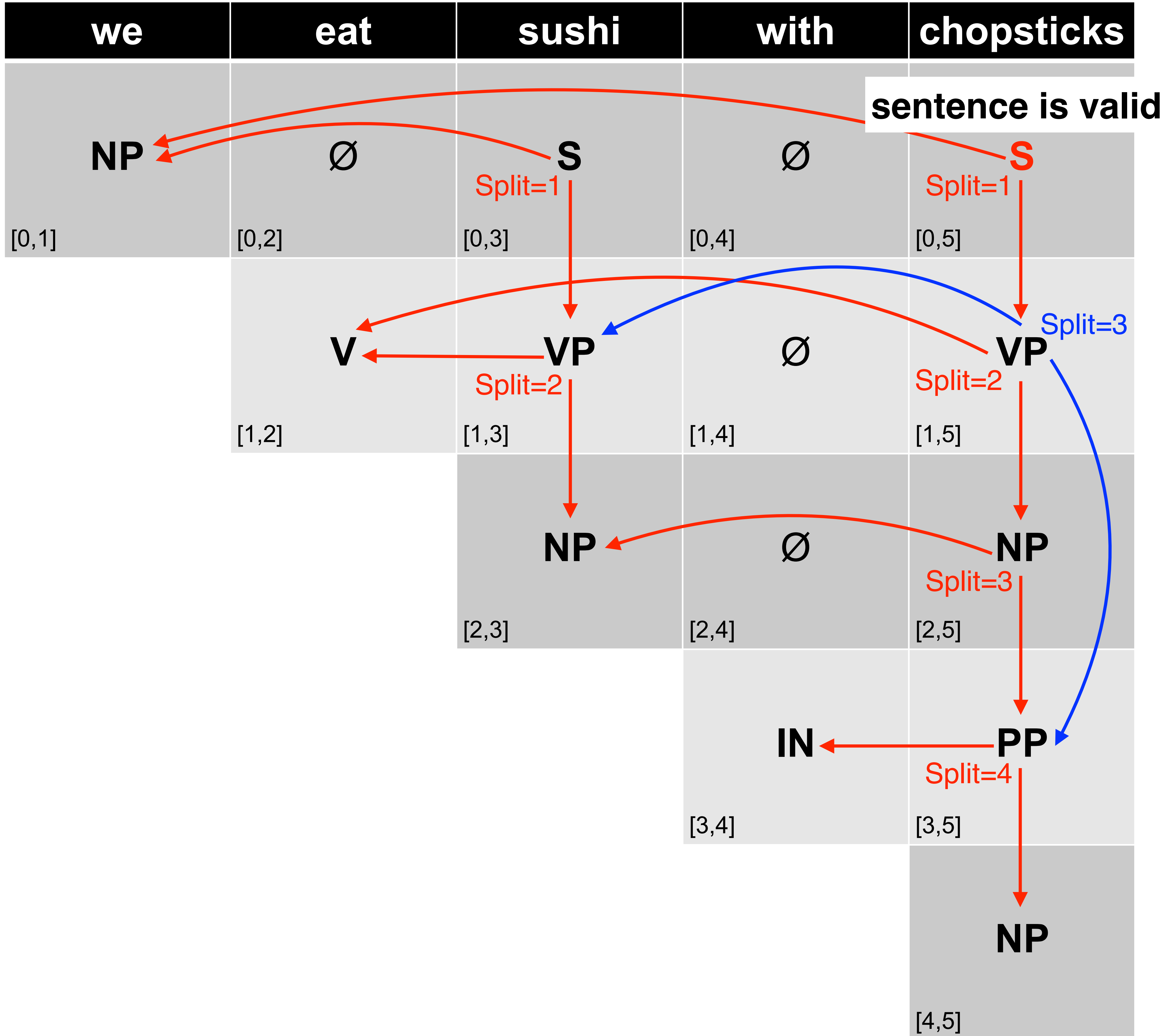
S → NP VP
NP → NP PP
PP → IN NP
VP → V NP
VP → VP PP
NP → we
NP → sushi
NP → chopsticks
IN → with
V → eat



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



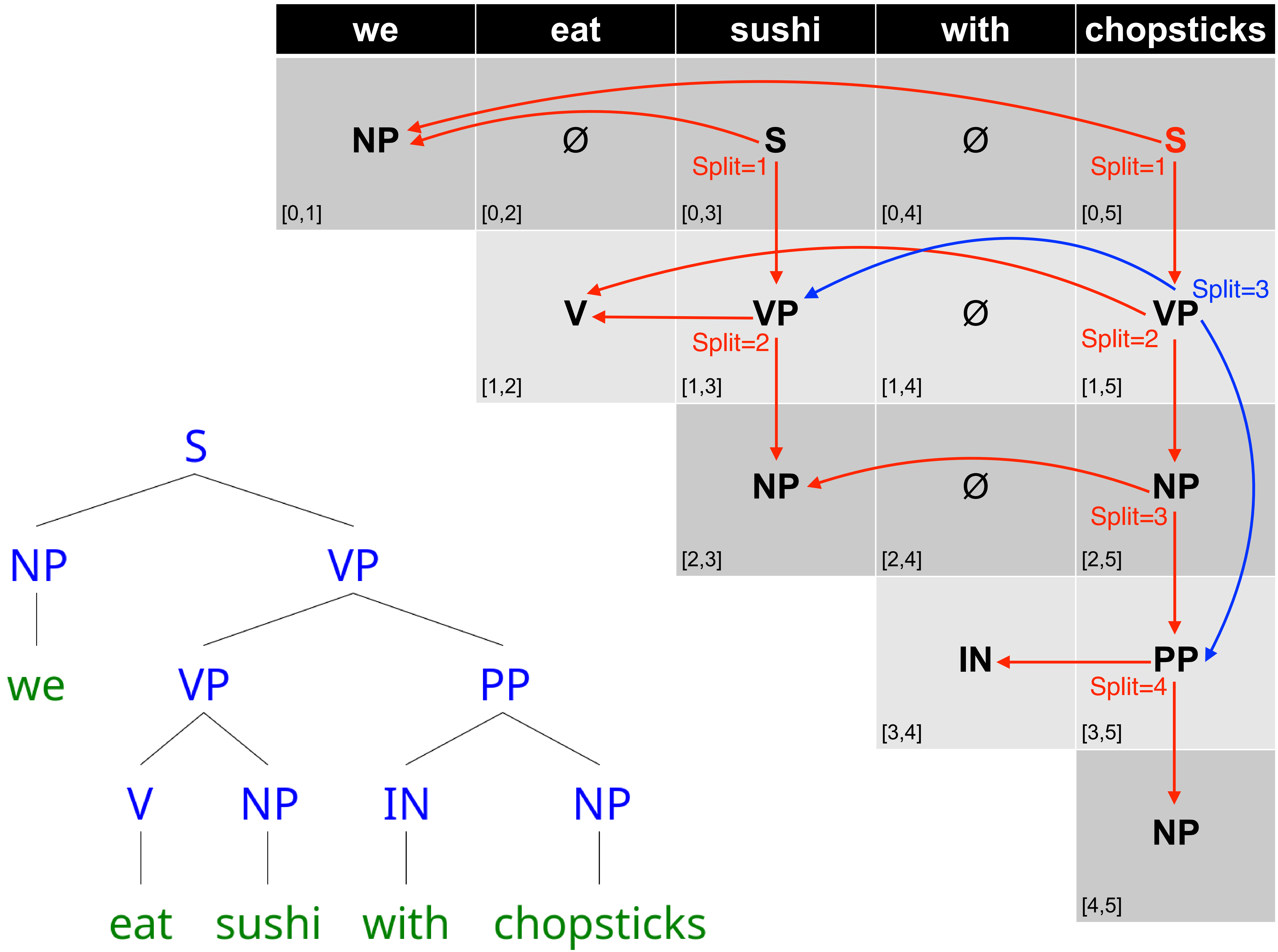
S → NP VP
NP → NP PP
PP → IN NP
VP → V NP
VP → VP PP
NP → we
NP → sushi
NP → chopsticks
IN → with
V → eat

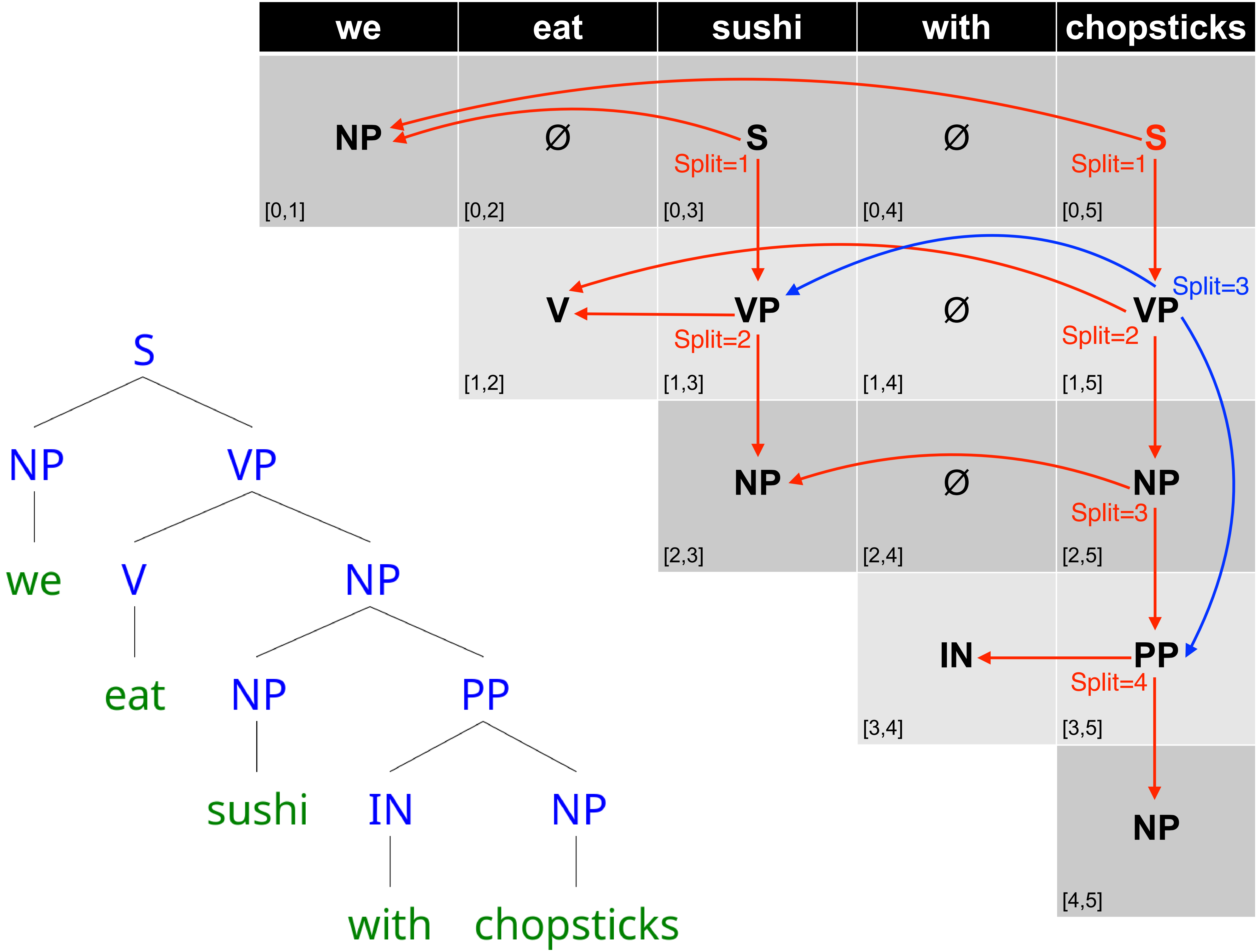


S → NP VP
NP → NP PP
PP → IN NP
VP → V NP
VP → VP PP
NP → we
NP → sushi
NP → chopsticks
IN → with
V → eat

CYK: Retrieving the Parses

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


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



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

CYK Algorithm

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

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

Figure 12.5 The CKY algorithm.

JM3, Ch 12

Representing English with CFGs

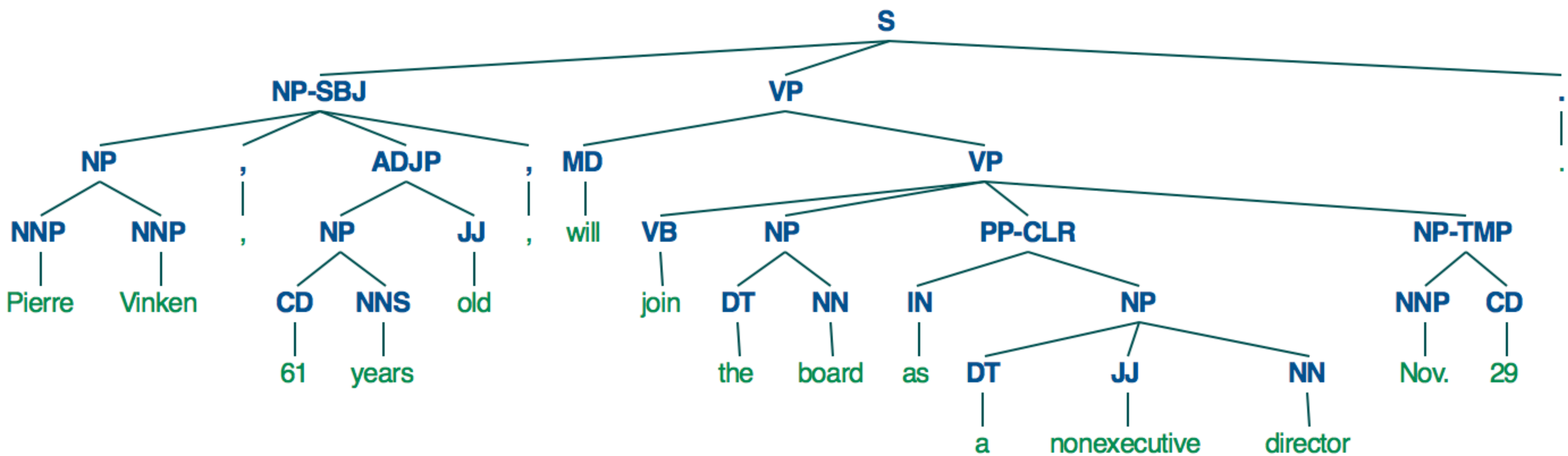
From Toy Grammars to Real Grammars

- 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




```
( (S
    (NP-SBJ
      (NP (NNP Pierre) (NNP Vinken) )
      (, ,)
      (ADJP
        (NP (CD 61) (NNS years) )
        (JJ old) )
      (, ,) )
    (VP (MD will)
      (VP (VB join)
        (NP (DT the) (NN board) )
        (PP-CLR (IN as)
          (NP (DT a) (JJ nonexecutive) (NN director) ))
        (NP-TMP (NNP Nov.) (CD 29) )))
    (. .) ))
```

Basic English Sentence Structures

- Declarative sentences (S → NP VP)
  - *The rat ate the cheese*
- Imperative sentences (S → VP)
  - *Eat the cheese!*
- Yes/no questions (S → VB NP VP)
  - *Did the rat eat the cheese?*
- *Wh*-subject-questions (S → WH VP)
  - *Who ate the cheese?*
- *Wh*-object-questions (S → 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? (NN|NNP)+ PP* VP? SBAR?

NP → PRP

Verb Phrases

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

VP → (MD|VB|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 linguistic structure
- There are relatively fast dynamic programming algorithms to retrieve this structure
- But what about ambiguity?
  - Extreme ambiguity will slow down parsing
  - If multiple possible parses, which is best?

Readings

- E18 Ch. 9.2, 10.1