Probabilistic Context-Free Grammar

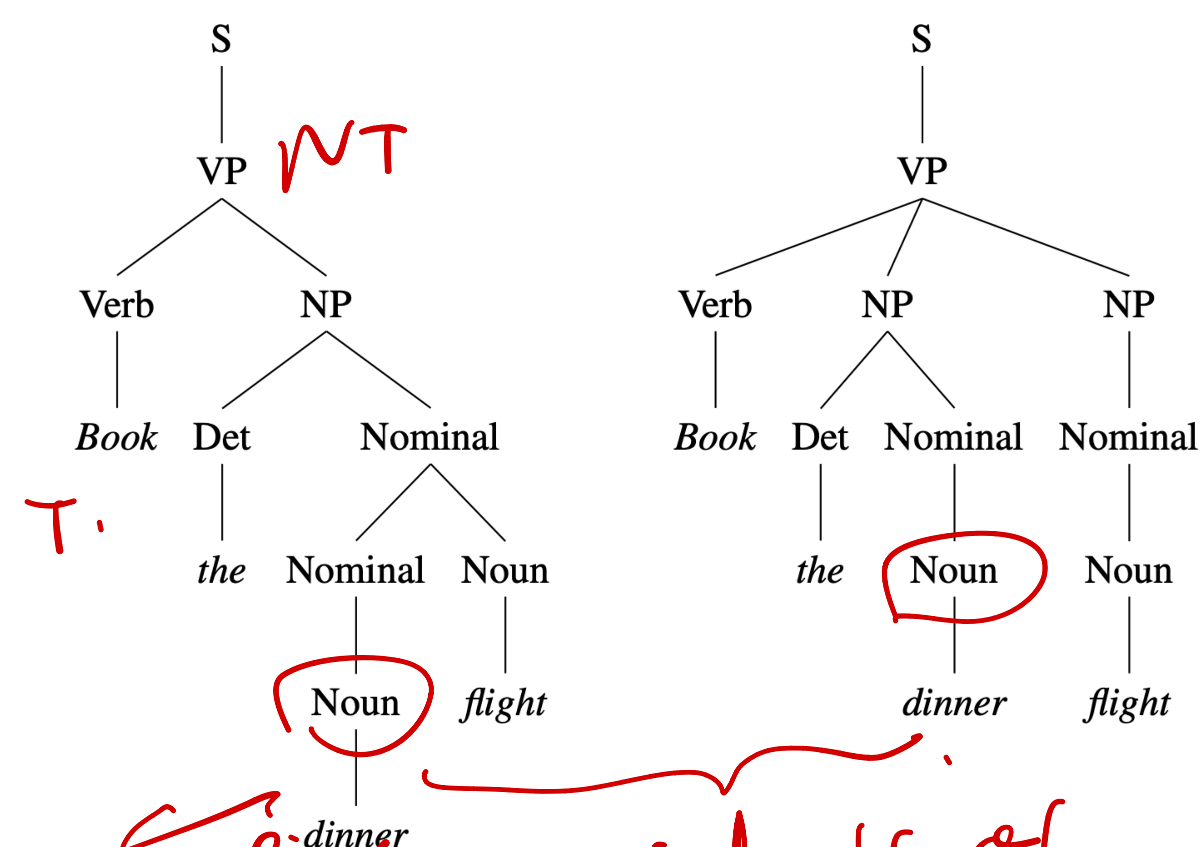
COMP90042

Natural Language Processing

Lecture 15

Ambiguity In Parsing

- Context-free grammars assign hierarchical structure to language
  - Linguistic notion of a '*syntactic constituent*'
  - Formulated as generating all strings in the language; or
  - Predicting the structure(s) for a given string
- Raises problem of ambiguity, e.g., which is better?



book the flight on behalf of dinner

Outline

- Probabilistic context-free grammars (PCFGs)
- Parsing using dynamic programming
- Limitations of ‘context-free’ assumption and some solutions:
    - parent annotation
    - head lexicalisation

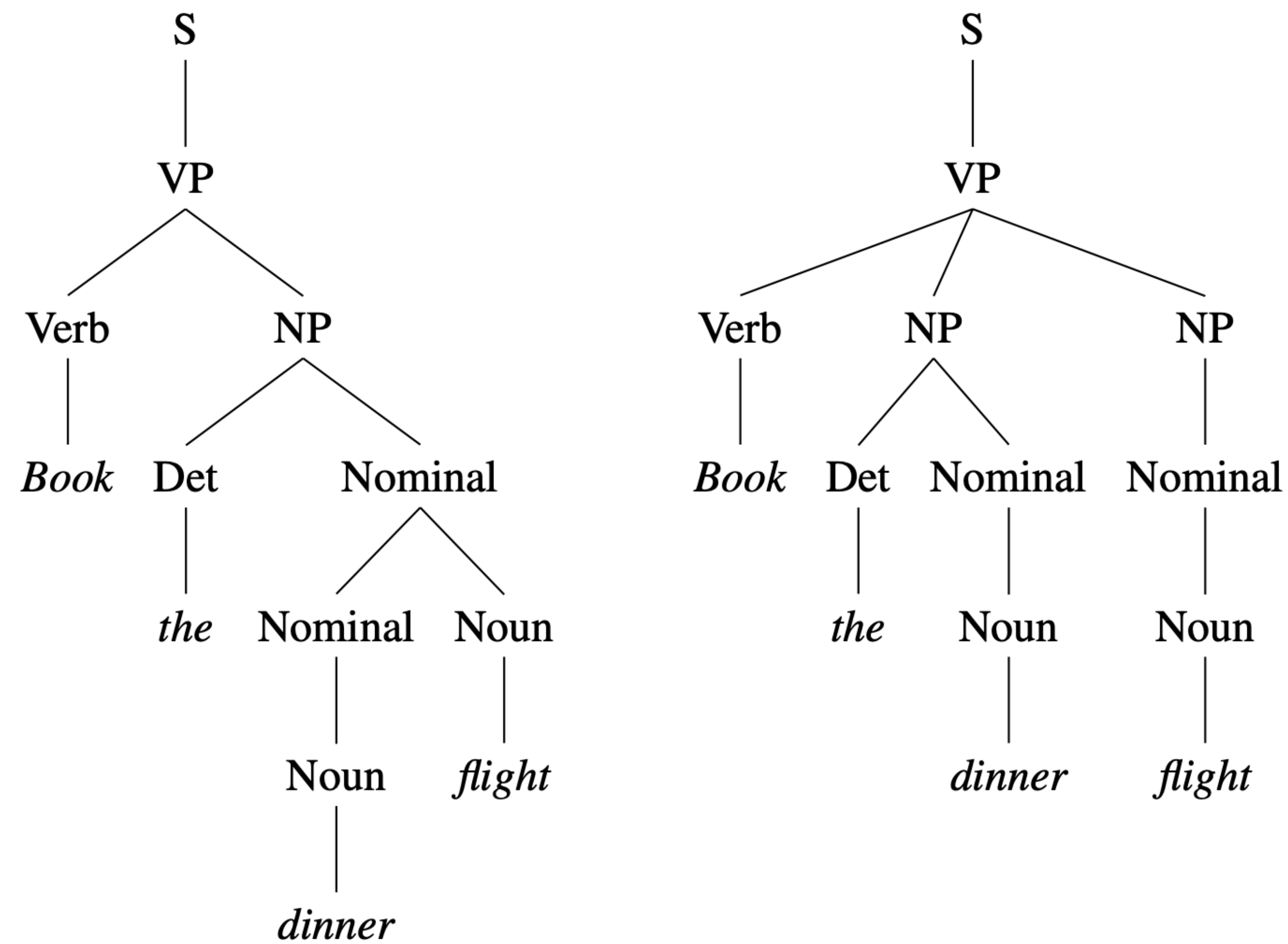
Basics of Probabilistic CFGs

- As for CFGs, same symbol set:
  - Terminals: words such as *book*
  - Non-terminal: syntactic labels such as NP or NN
- Same productions (rules)
  - LHS non-terminal → ordered list of RHS symbols
- In addition, store a **probability** with each production
  - NP → DT NN [p = 0.45]
  - NN → cat [p = 0.02]
  - NN → leprechaun [p = 0.00001]
  - …

Probabilistic CFGs

- Probability values denote **conditional**
  - Pr(LHS → RHS)
  - Pr(RHS | LHS)
- Consequently they:
  - must be positive values, between 0 and 1
  - must sum to one for given LHS
- E.g.,
  - NN → aadvark [p = 0.0003]
  - NN → cat [p = 0.02]
  - NN → leprechaun [p = 0.0001]
  - $\sum_x$ Pr(NN → *x*) = 1

$\sum_x \Pr(NN \rightarrow x) = 1$



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

Stochastic Generation with PCFGs

Almost the same as for CFG, with one twist:

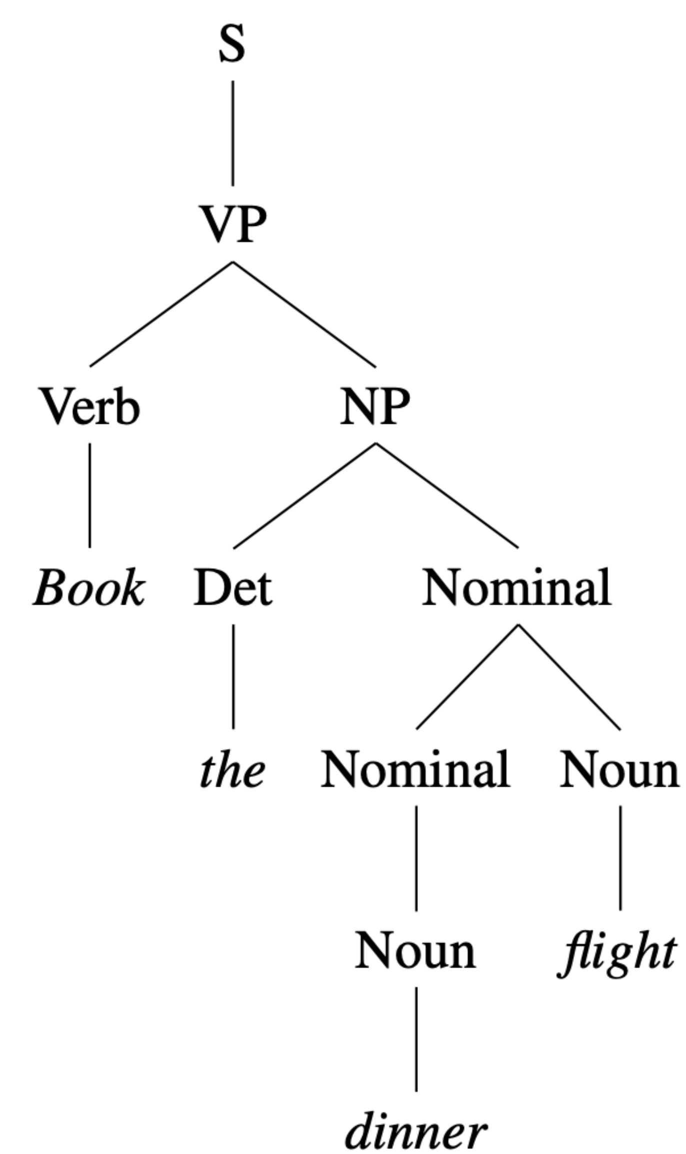
1. Start with S, the sentence symbol
2. Choose a rule with S as the LHS
   - **Randomly select a RHS** according to Pr(RHS | LHS) e.g., S → VP
   - Apply this rule, e.g., substitute VP for S
3. Repeat step 2 for each non-terminal in the string (here, VP)
4. Stop when no non-terminals remain

Gives us a tree, as before, with a sentence as the yield

How Likely is a Tree?

- Given a tree, we can compute its probability
  - Decomposes into probability of each production
- E.g., for (left) tree,
  - P(tree) =

    P(S → VP) ×
    P(VP → Verb NP) ×
    P(Verb → *Book*) ×
    P(NP → Det Nominal) ×
    P(Det → *the*) ×
    P(Nominal → Nominal Noun) ×
    P(Nominal → Noun) ×
    P(Noun → *dinner*) ×
    P(Noun → *flight*)



How Likely is a Tree?

P(tree)

= P(S → VP) × P(VP → Verb NP) × P(Verb → Book) ×
P(NP → Det Nominal) × P(Det → the) × P(Nominal → Nominal Noun) ×
P(Nominal → Noun) × P(Noun → dinner) × P(Noun → flight)

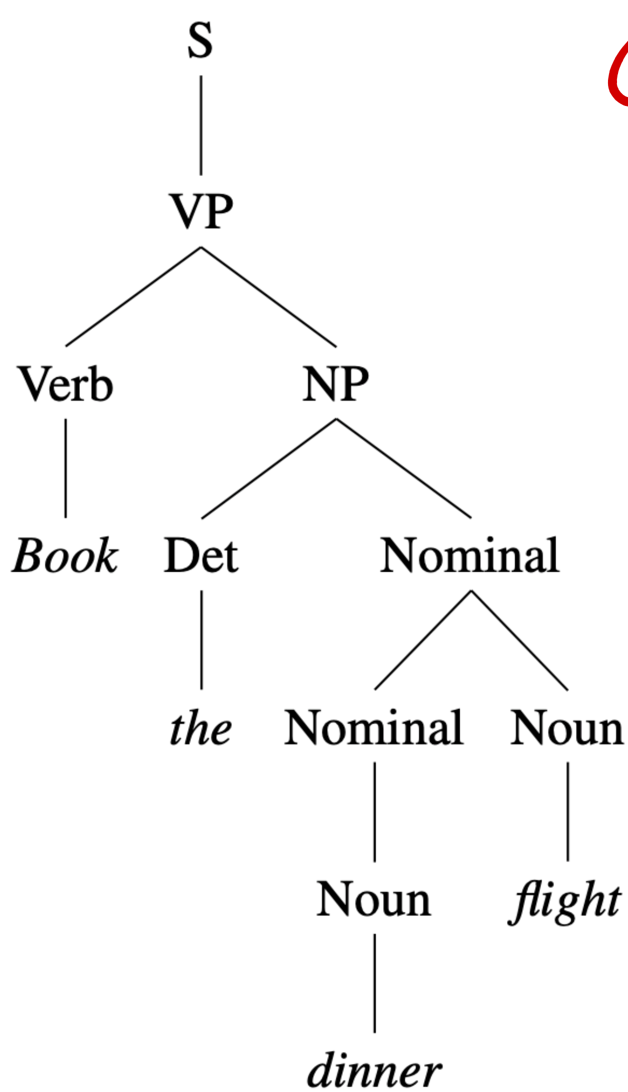
= 0.05 × 0.20 × 0.30 ×
0.20 × 0.60 × 0.20 ×
0.75 × 0.10 × 0.40

= 2.2 × 10-6

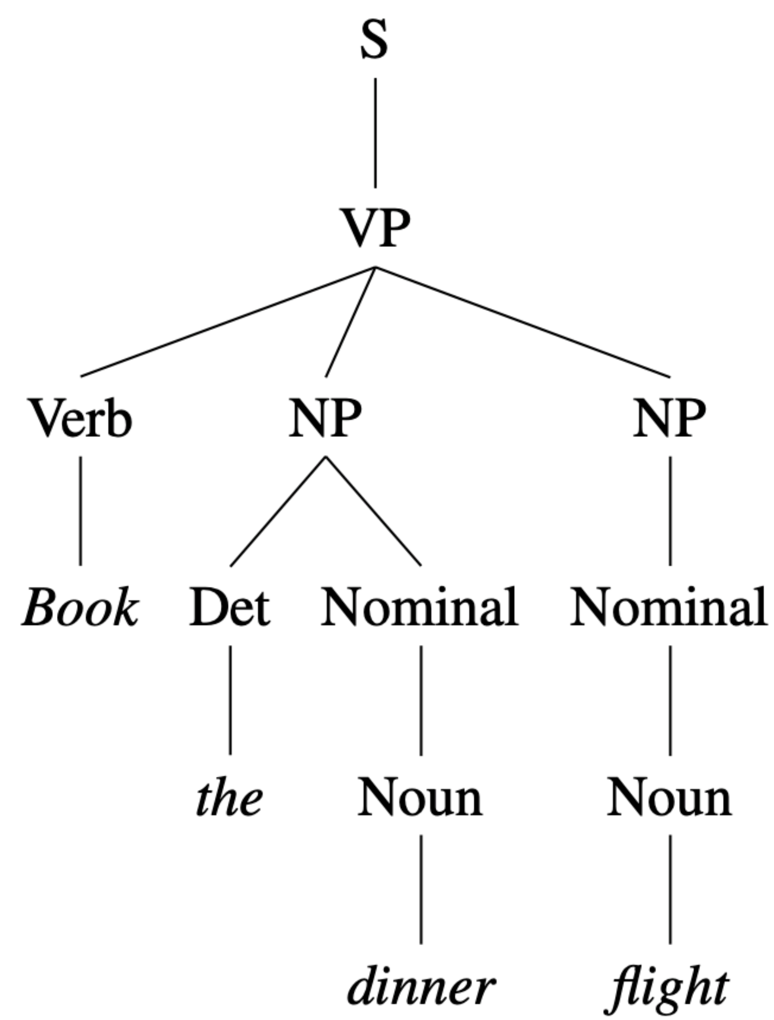
Rules			P
S	→	VP	.05
VP	→	Verb NP	.20
NP	→	Det Nominal	.20
Nominal	→	Nominal Noun	.20
Nominal	→	Noun	.75
Verb	→	book	.30
Det	→	the	.60
Noun	→	dinner	.10
Noun	→	flight	.40

Resolving Parse Ambiguity

- Can select between different trees based on P(T)
- $P(T_{left}) = 2.2 \times 10^{-6}$ $P(T_{right}) = 6.1 \times 10^{-7}$



choose one.



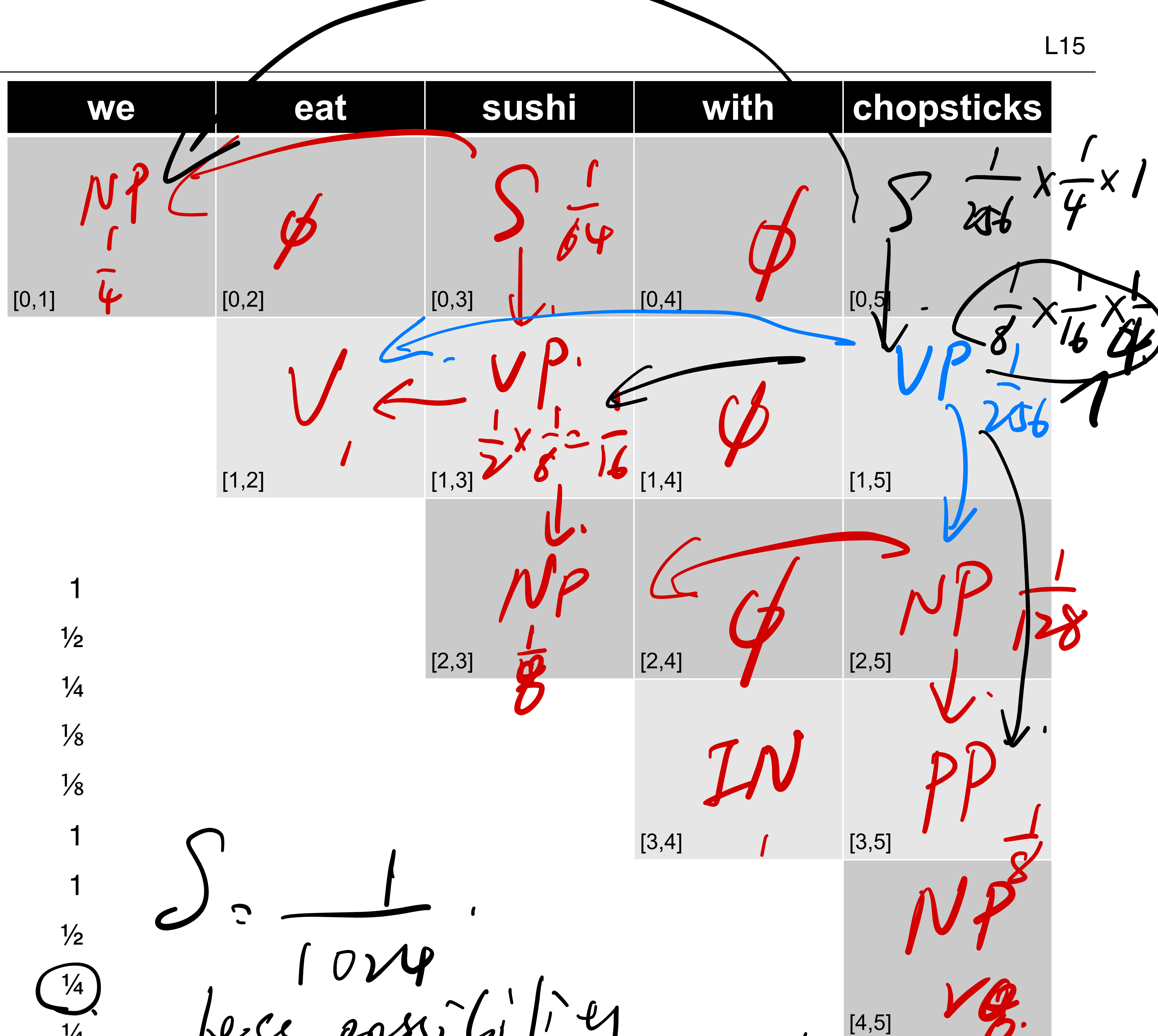
Parsing PCFGs

- Instead of selecting between two trees, can we select a tree from the set of all possible trees?
- Before we looked at
  - CYK
  - for unweighted grammars (CFGs)
  - finds **all possible trees**
- But there are often 1000s, many completely nonsensical
- Can we solve for the **most probable tree**?

CYK for PCFGs

- CYK finds **all trees** for a sentence; we want **best** tree
- Prob. CYK follows similar process to standard CYK
- Convert grammar to Chomsky Normal Form (CNF)
  - E.g., VP → Verb NP NP [0.10]
  - becomes VP → Verb NP+NP [0.10]
    NP+NP → NP NP [1.0]
  - where NP+NP is a new symbol.

PCFG Parsing Example



S	→	NP VP	1
NP	→	NP PP	½
	→	we	¼
	→	sushi	⅛
	→	chopsticks	⅛
PP	→	IN NP	1
IN	→	with	1
VP	→	V NP	½
	→	VP PP	¼
	→	MD V	¼
V	→	eat	1

$S = \frac{1}{1024}$.

less possibility cancelled.

we	eat	sushi	with	chopsticks
NP 1/4 [0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	V 1 [1,2]	[1,3]	[1,4]	[1,5]
		NP 1/8 [2,3]	[2,4]	[2,5]
			IN 1 [3,4]	[3,5]
				NP 1/8 [4,5]

S	→	NP VP	1
NP	→	NP PP	½
	→	we	¼
	→	sushi	⅛
	→	chopsticks	⅛
PP	→	IN NP	1
IN	→	with	1
VP	→	V NP	½
	→	VP PP	¼
	→	MD V	¼
V	→	eat	1

we	eat	sushi	with	chopsticks
NP 1/4 [0,1]	Ø [0,2]	[0,3]	[0,4]	[0,5]
	V 1 [1,2]	[1,3]	[1,4]	[1,5]
		NP 1/8 [2,3]	[2,4]	[2,5]
			IN 1 [3,4]	[3,5]
				NP 1/8 [4,5]

S	→	NP VP	1
NP	→	NP PP	½
	→	we	¼
	→	sushi	⅛
	→	chopsticks	⅛
PP	→	IN NP	1
IN	→	with	1
VP	→	V NP	½
	→	VP PP	¼
	→	MD V	¼
V	→	eat	1

we	eat	sushi	with	chopsticks
NP 1/4 [0,1]	Ø [0,2]	[0,3]	[0,4]	[0,5]
	V 1 [1,2]	**VP** 1/16 (1/2 * 1 * 1/8) [1,3]	[1,4]	[1,5]
		NP 1/8 [2,3]	[2,4]	[2,5]
			IN 1 [3,4]	[3,5]
				NP 1/8 [4,5]

S	→	NP VP	1
NP	→	NP PP	½
	→	we	¼
	→	sushi	⅛
	→	chopsticks	⅛
PP	→	IN NP	1
IN	→	with	1
VP	→	V NP	½
	→	VP PP	¼
	→	MD V	¼
V	→	eat	1

we	eat	sushi	with	chopsticks
NP 1/4 [0,1]	Ø [0,2]	**S** 1/64 (1 * 1/4 * 1/64) [0,3]	[0,4]	[0,5]
	V 1 [1,2]	**VP** 1/16 [1,3]	[1,4]	[1,5]
		NP 1/8 [2,3]	[2,4]	[2,5]
			IN 1 [3,4]	[3,5]
				NP 1/8 [4,5]

S	→	NP VP	1
NP	→	NP PP	½
	→	we	¼
	→	sushi	⅛
	→	chopsticks	⅛
PP	→	IN NP	1
IN	→	with	1
VP	→	V NP	½
	→	VP PP	¼
	→	MD V	¼
V	→	eat	1

we	eat	sushi	with	chopsticks
NP 1/4 [0,1]	Ø [0,2]	**S** 1/64 [0,3]	Ø [0,4]	[0,5]
	V 1 [1,2]	**VP** 1/16 [1,3]	Ø [1,4]	[1,5]
		NP 1/8 [2,3]	Ø [2,4]	[2,5]
			IN 1 [3,4]	[3,5]
				NP 1/8 [4,5]

S	→	NP VP	1
NP	→	NP PP	½
	→	we	¼
	→	sushi	⅛
	→	chopsticks	⅛
PP	→	IN NP	1
IN	→	with	1
VP	→	V NP	½
	→	VP PP	¼
	→	MD V	¼
V	→	eat	1

we	eat	sushi	with	chopsticks
NP 1/4 [0,1]	Ø [0,2]	**S** 1/64 [0,3]	Ø [0,4]	[0,5]
	V 1 [1,2]	**VP** 1/16 [1,3]	Ø [1,4]	[1,5]
		NP 1/8 [2,3]	Ø [2,4]	[2,5]
			IN 1 [3,4]	**PP** 1/8 (1 * 1 * 1/8) [3,5]
				NP 1/8 [4,5]

S	→	NP VP	1
NP	→	NP PP	½
	→	we	¼
	→	sushi	⅛
	→	chopsticks	⅛
PP	→	IN NP	1
IN	→	with	1
VP	→	V NP	½
	→	VP PP	¼
	→	MD V	¼
V	→	eat	1

we	eat	sushi	with	chopsticks
NP 1/4 [0,1]	Ø [0,2]	**S** 1/64 [0,3]	Ø [0,4]	[0,5]
	V 1 [1,2]	**VP** 1/16 [1,3]	Ø [1,4]	[1,5]
		NP 1/8 [2,3]	Ø [2,4]	**NP** 1/128 (1/2 * 1/8 * 1/8) [2,5]
			IN 1 [3,4]	**PP** 1/8 [3,5]
				NP 1/8 [4,5]

S	→	NP VP	1
NP	→	NP PP	½
	→	we	¼
	→	sushi	⅛
	→	chopsticks	⅛
PP	→	IN NP	1
IN	→	with	1
VP	→	V NP	½
	→	VP PP	¼
	→	MD V	¼
V	→	eat	1

we	eat	sushi	with	chopsticks
NP 1/4 [0,1]	Ø [0,2]	**S** 1/64 [0,3]	Ø [0,4]	[0,5]
	V 1 [1,2]	**VP** 1/16 [1,3]	Ø [1,4]	**VP** 1/256 (1/2 * 1 * 1/128) [1,5]
		NP 1/8 [2,3]	Ø [2,4]	**NP** 1/128 [2,5]
			IN 1 [3,4]	**PP** 1/8 [3,5]
				NP 1/8 [4,5]

S	→	NP VP	1
NP	→	NP PP	½
	→	we	¼
	→	sushi	⅛
	→	chopsticks	⅛
PP	→	IN NP	1
IN	→	with	1
VP	→	V NP	½
	→	VP PP	¼
	→	MD V	¼
V	→	eat	1

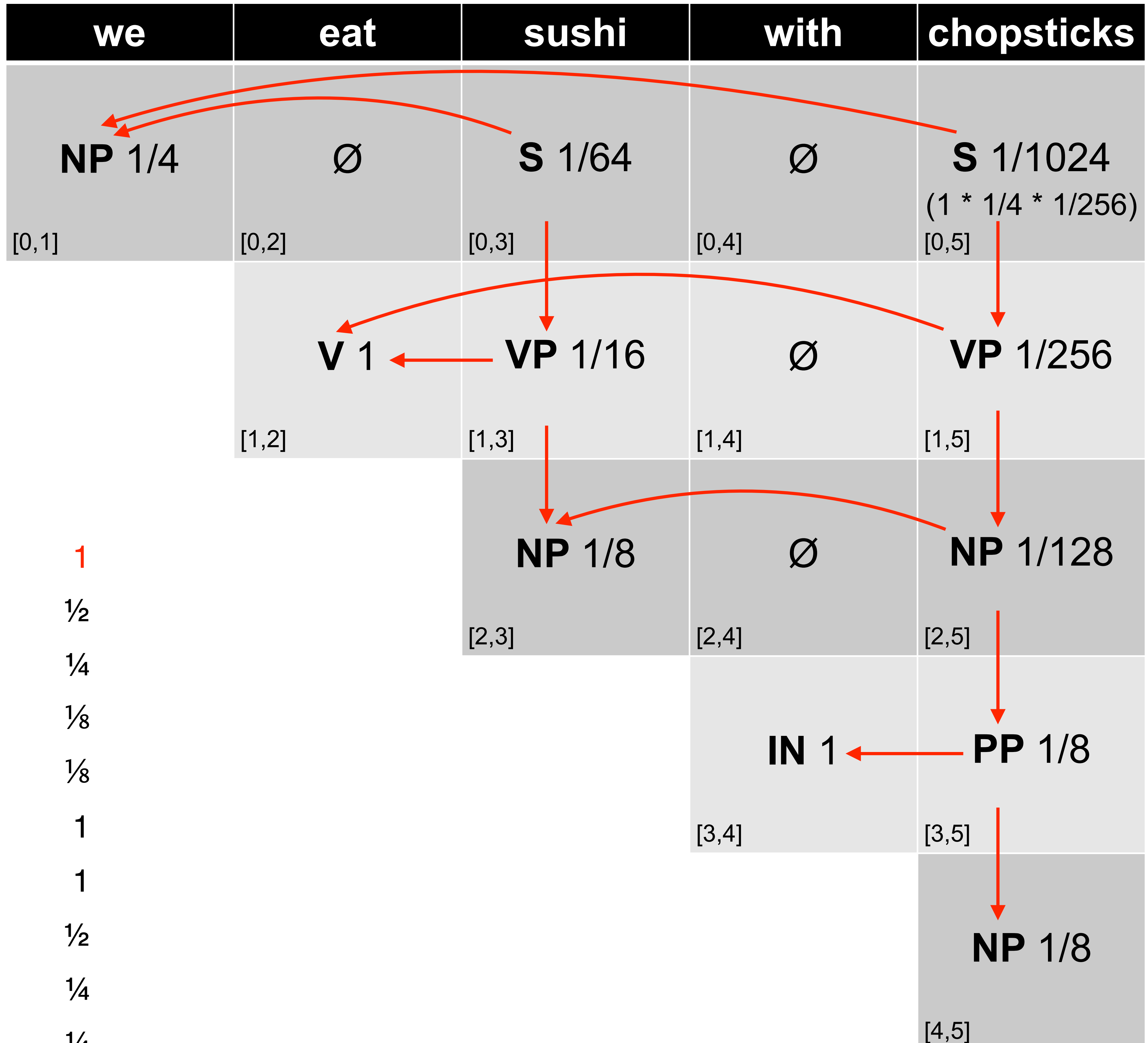
we	eat	sushi	with	chopsticks
NP 1/4 [0,1]	Ø [0,2]	**S** 1/64 [0,3]	Ø [0,4]	[0,5]
	V 1 [1,2]	**VP** 1/16 [1,3]	Ø [1,4]	**VP** 1/512 (1/4*1/16*1/8) [1,5]
		NP 1/8 [2,3]	Ø [2,4]	**NP** 1/128 [2,5]
			IN 1 [3,4]	**PP** 1/8 [3,5]
				NP 1/8 [4,5]

S	→	NP VP	1
NP	→	NP PP	½
	→	we	¼
	→	sushi	⅛
	→	chopsticks	⅛
PP	→	IN NP	1
IN	→	with	1
VP	→	V NP	½
	→	VP PP	¼
	→	MD V	¼
V	→	eat	1

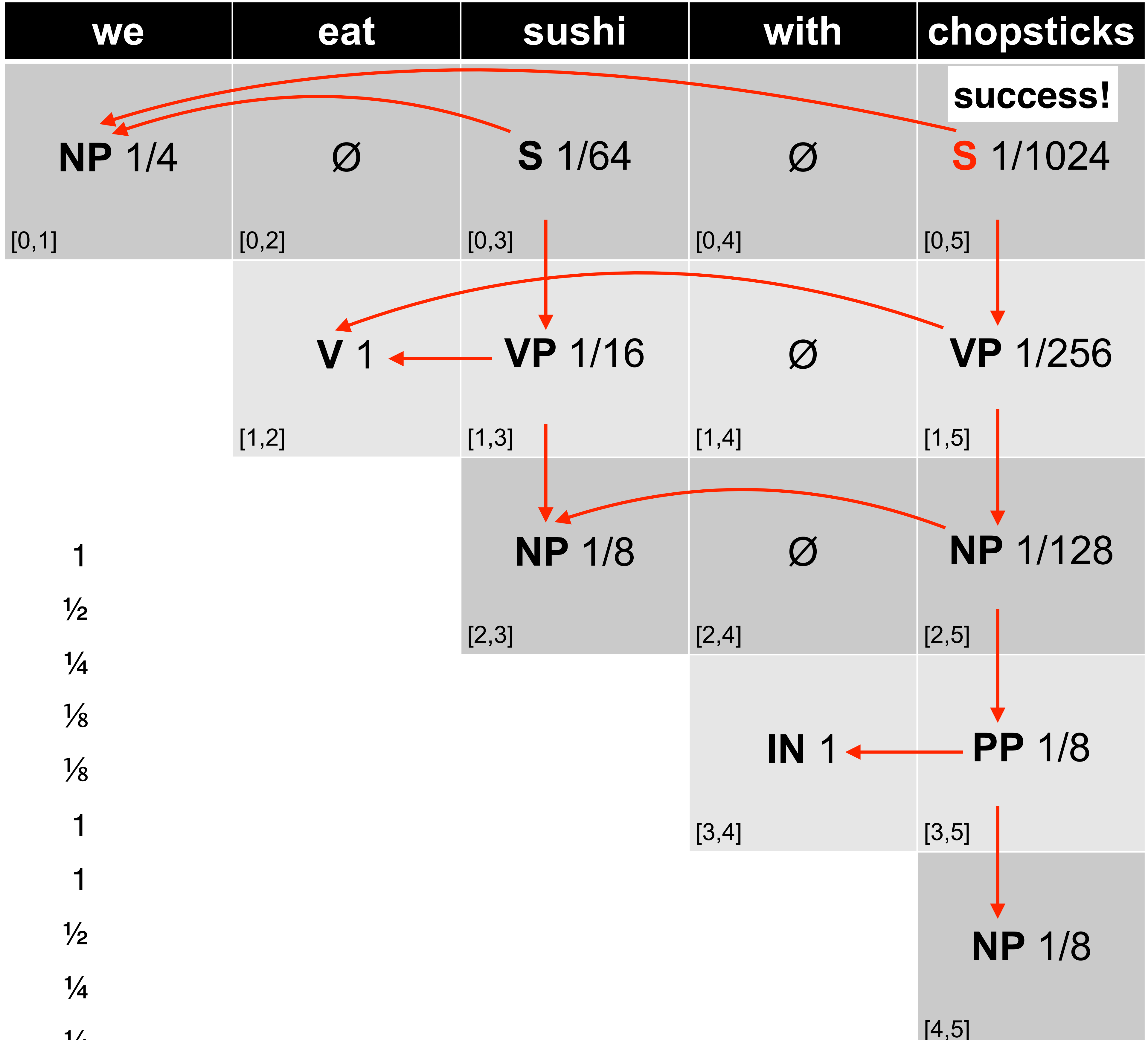
we	eat	sushi	with	chopsticks
NP 1/4 [0,1]	Ø [0,2]	**S** 1/64 [0,3]	Ø [0,4]	[0,5]
	V 1 [1,2]	**VP** 1/16 [1,3]	Ø [1,4]	**VP** 1/256 [1,5]
		NP 1/8 [2,3]	Ø [2,4]	**NP** 1/128 [2,5]
			IN 1 [3,4]	**PP** 1/8 [3,5]
				NP 1/8 [4,5]

1/256 > 1/512!

S	→	NP VP	1
NP	→	NP PP	½
	→	we	¼
	→	sushi	⅛
	→	chopsticks	⅛
PP	→	IN NP	1
IN	→	with	1
VP	→	V NP	½
	→	VP PP	¼
	→	MD V	¼
V	→	eat	1



S	→	NP VP	1
NP	→	NP PP	½
	→	we	¼
	→	sushi	⅛
	→	chopsticks	⅛
PP	→	IN NP	1
IN	→	with	1
VP	→	V NP	½
	→	VP PP	¼
	→	MD V	¼
V	→	eat	1

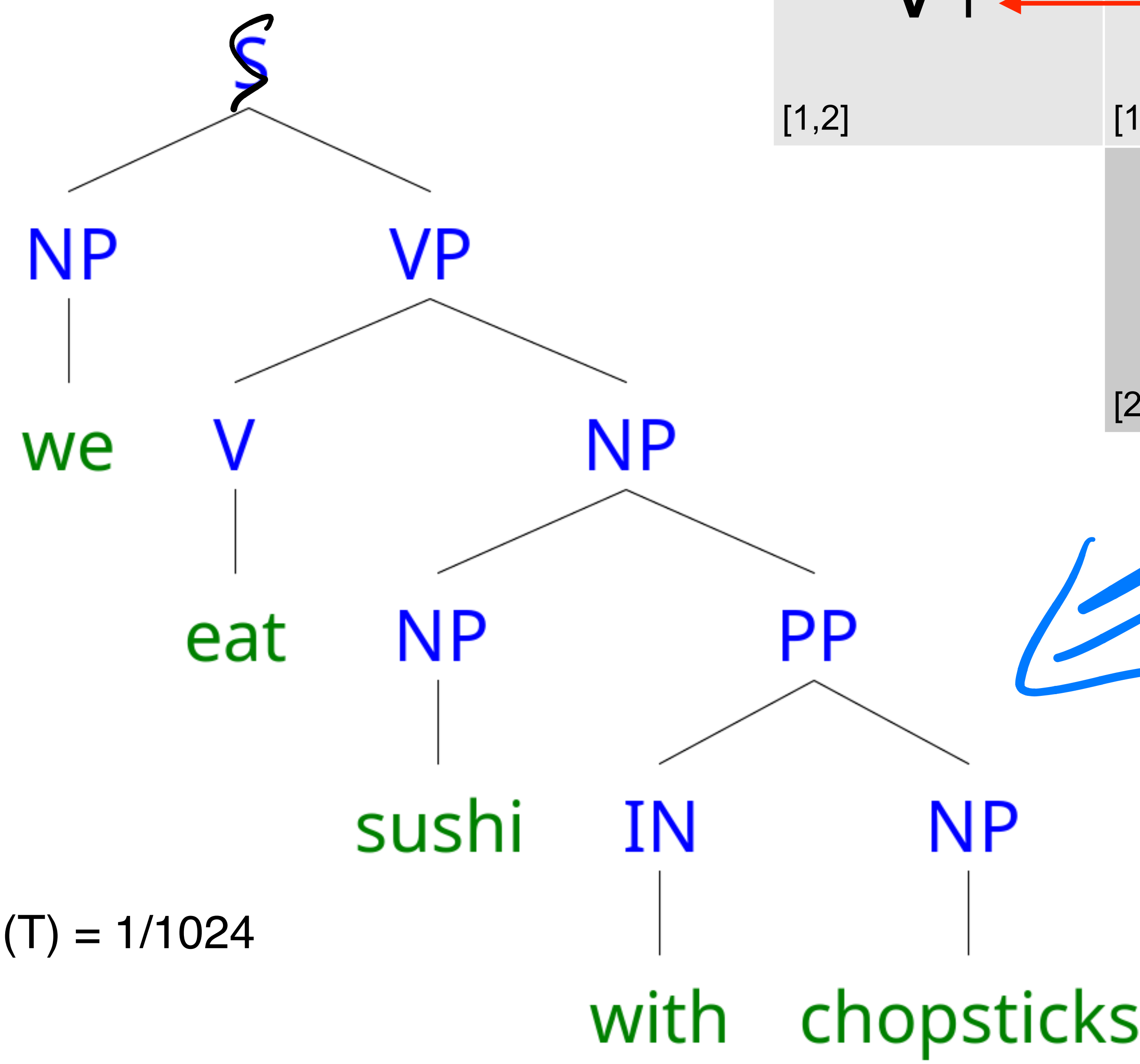


S	→	NP VP	1
NP	→	NP PP	½
	→	we	¼
	→	sushi	⅛
	→	chopsticks	⅛
PP	→	IN NP	1
IN	→	with	1
VP	→	V NP	½
	→	VP PP	¼
	→	MD V	¼
V	→	eat	1

Prob CYK: Retrieving the Parses

- S in the top-right corner of parse table indicates success
- Retain back-pointer to best analysis
- To get parse(s), follow pointers back for each match
- Convert back from CNF by removing new non-terminals

we	eat	sushi	with	chopsticks
NP 1/4 [0,1]	Ø [0,2]	**S** 1/64 [0,3]	Ø [0,4]	**S** 1/1024 [0,5]
	V 1 [1,2]	**VP** 1/16 [1,3]	Ø [1,4]	**VP** 1/256 [1,5]
		NP 1/8 [2,3]	Ø [2,4]	**NP** 1/128 [2,5]
			IN 1 [3,4]	**PP** 1/8 [3,5]
				NP 1/8 [4,5]



P(T) = 1/1024

Prob. CYK

Storing only the best probable tree.

```
function PROBABILISTIC-CKY(words, grammar) returns most probable parse
                                                and its probability
  for j ← from 1 to LENGTH(words) do
     for all { A | A → words[j] ∈ grammar}
        table[j−1, j, A] ← P(A → words[j])
     for i ← from j−2 downto 0 do
        for k ← i+1 to j−1 do
           for all { A | A → BC ∈ grammar,
                      and table[i,k,B] > 0 and table[k,j,C] > 0 }
              if (table[i,j,A] < P(A → BC) × table[i,k,B] × table[k,j,C]) then
                 table[i,j,A] ← P(A → BC) × table[i,k,B] × table[k,j,C]
                 back[i,j,A] ← {k,B,C}
  return BUILD_TREE(back[1, LENGTH(words), S]), table[1, LENGTH(words), S]
```

Source: JM3 Ch 14

PCYK
add possibility

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

  for j ← from 1 to LENGTH(words) do
    for all {A | A → words[j] ∈ grammar}
      table[j−1, j] ← table[j−1, j] ∪ A
    for i ← from j−2 downto 0 do
      for k ← i+1 to j−1 do
        for all {A | A → BC ∈ grammar and B ∈ table[i,k] and C ∈ table[k, j]}
          table[i,j] ← table[i,j] ∪ A
```

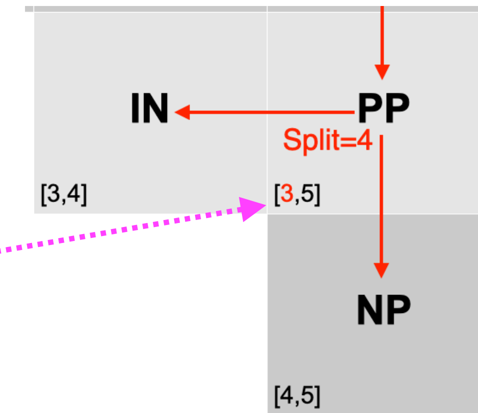


Figure 12.5 The CKY algorithm.

CYK can be thought of as storing all events with probability = 1

validity test now looks to see that the child chart cells have non-zero probability

```
function PROBABILISTIC-CKY(words,grammar) returns most probable parse
                                                  and its probability
  for j ← from 1 to LENGTH(words) do
    for all { A | A → words[j] ∈ grammar}
      table[j−1, j,A] ← P(A → words[j])
    for i ← from j−2 downto 0 do
      for k ← i+1 to j−1 do
        for all { A | A → BC ∈ grammar,
                        and table[i,k,B] > 0 and table[k, j,C] > 0 }
          if (table[i,j,A] < P(A → BC) × table[i,k,B] × table[k,j,C]) then
            table[i,j,A] ← P(A → BC) × table[i,k,B] × table[k,j,C]
            back[i,j,A] ← {k,B,C}
  return BUILD_TREE(back[1, LENGTH(words), S]), table[1, LENGTH(words), S]
```

Instead of storing set of symbols, store the probability of best scoring tree fragment covering span [i,j] with root symbol A

Overwrite lower scoring analysis if this one is better, and record the best production

chart now stores probabilities for each span and symbol

Complexity of CYK

- What's the space and time complexity of this algorithm?
  - in terms of *n* the length of the input sentence

$C(\text{Space}) = n^2 \to$ Table.

$C(\text{time}) = n^3 \to$ 3 loops
- left to right
- bottom to up
- splits & check

Issues with PCFG

CFG.

PCFG Problem 1: **Poor Independence Assumptions**

- Rewrite decisions made independently, whereas inter-dependence is often needed to capture global structure.
  - NP → Det N

    P(RHS | LHS)

  - Probability of this rule independent of rest of tree

does not matter how NP comes.

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

NP statistics in the Switchboard corpus

cannot represent this NP → Det N.

  - No way to represent this contextual differences in PCFG probabilities

Poor Independence Assumptions

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

NP statistics in the Switchboard corpus

$\neq$

$$NP \rightarrow DT\ NN \quad .28$$
$$NP \rightarrow PRP \quad .25$$

PCFG probabilities based on Switchboard corpus

- No way to capture the fact that in subject position, NP → PRP should go up to 0.91
- While in object position NP → DT NN should go up to 0.66

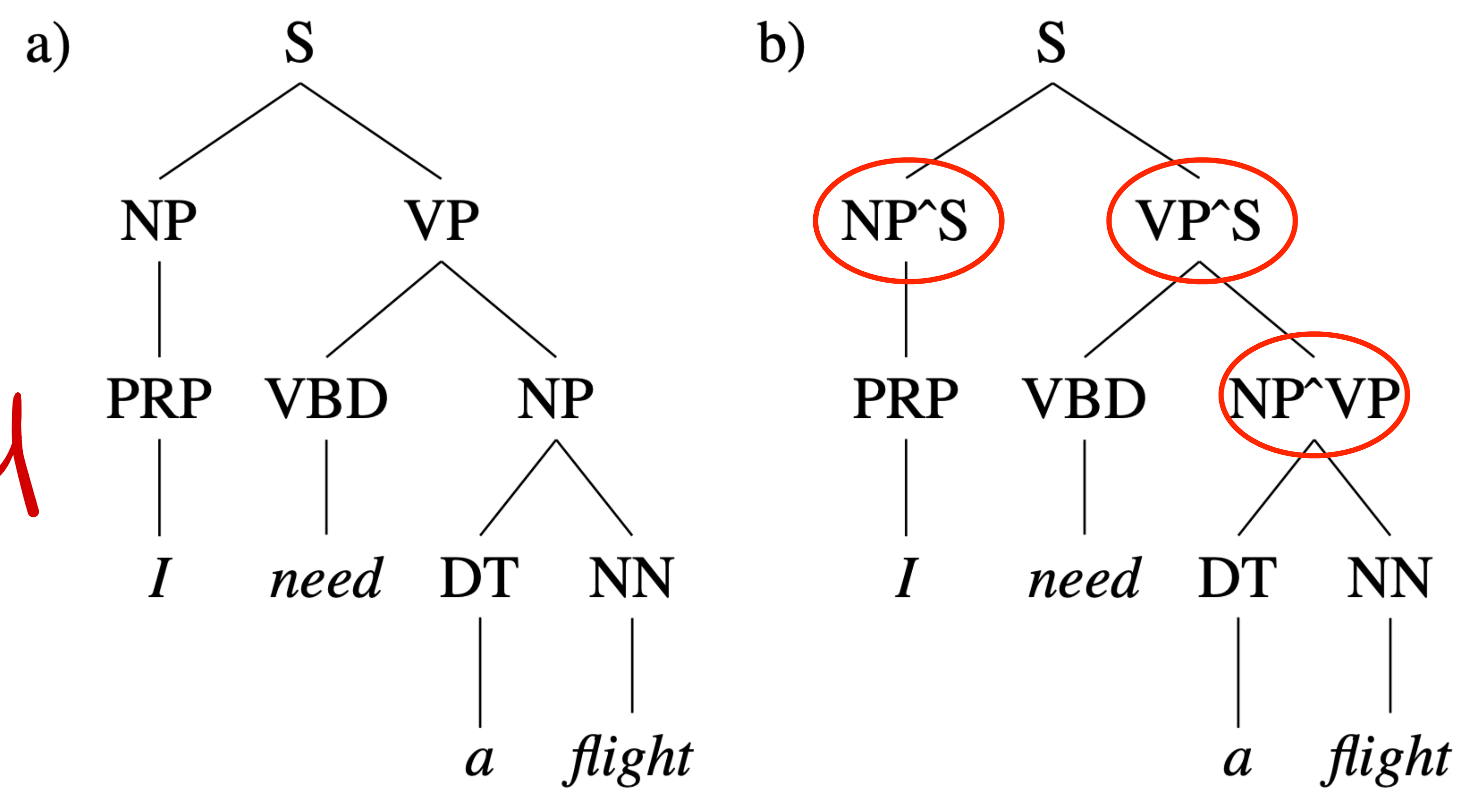
No way to capture this

- Solution: add a condition to denote whether NP is a subject or object

Solution: Parent Conditioning

- Make non-terminals more explicit by incorporating parent symbol into each symbol

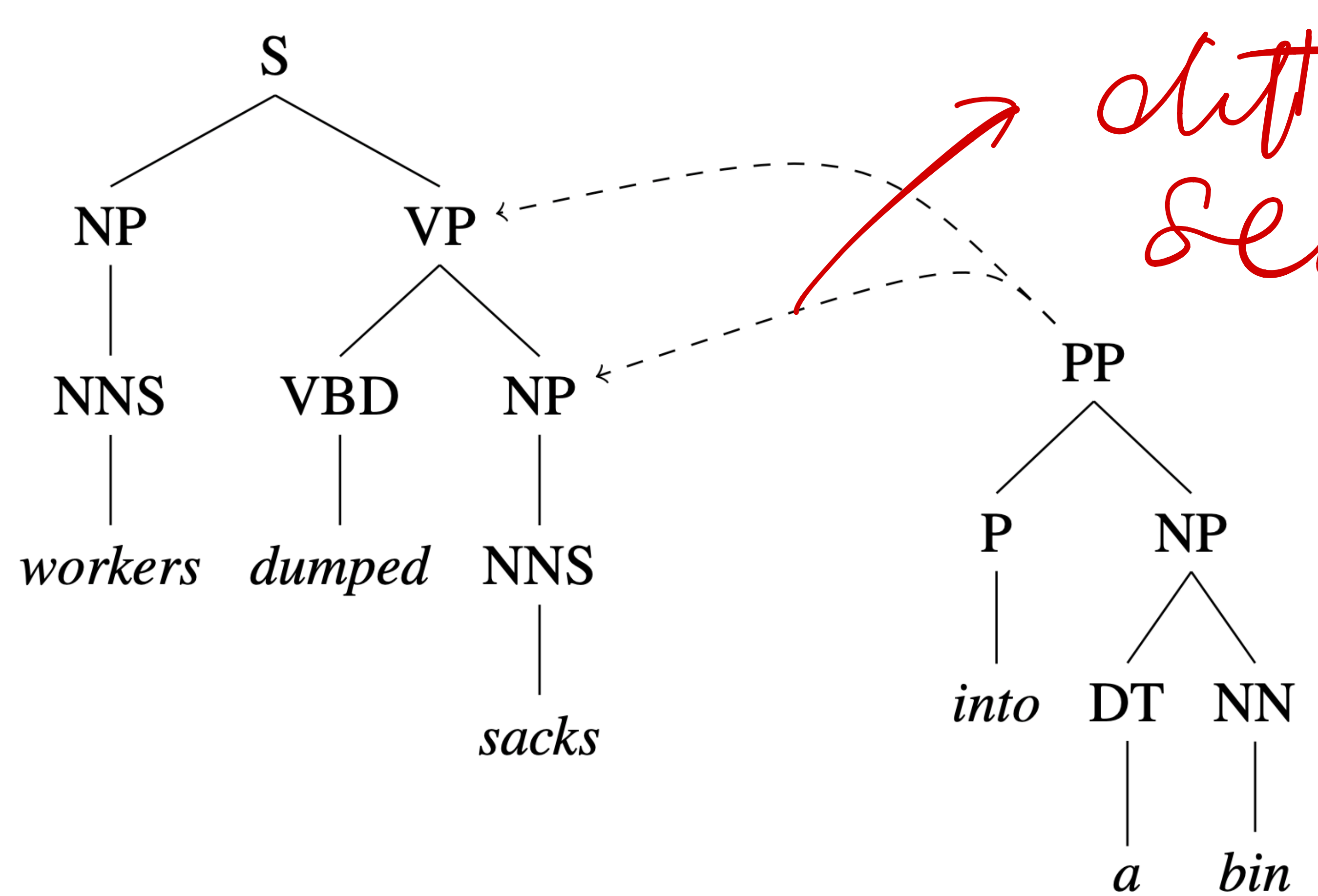
has a parent attached to it.



- NP^S represents subject position (left)
- NP^VP denotes object position (right)

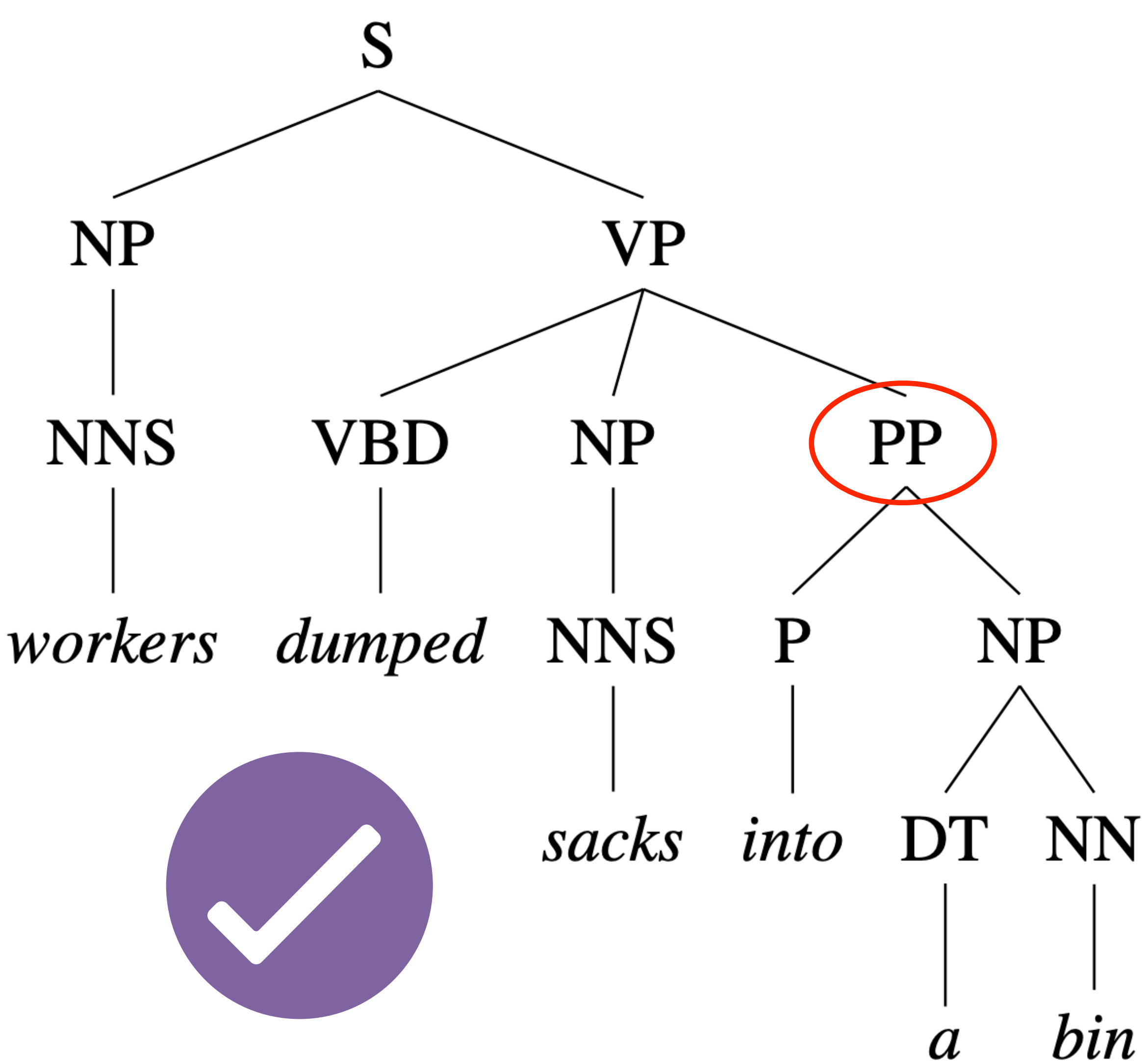
PCFG Problem 2:
Lack of Lexical Conditioning

- Lack of sensitivity to words in tree
- Prepositional phrase (PP) attachment ambiguity
  - *Worker dumped sacks into a bin*

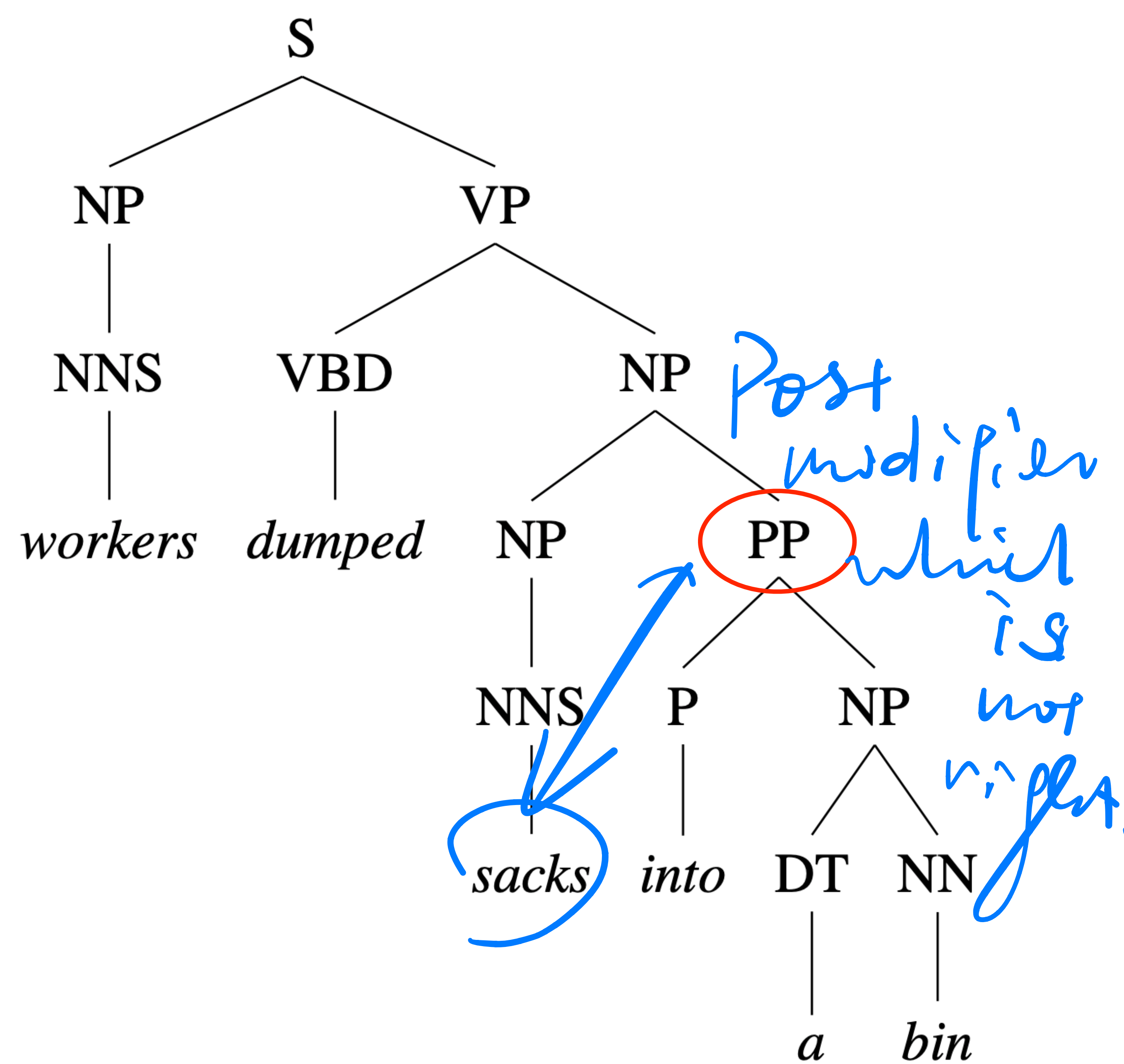


different sensitivity

PP Attachment Ambiguity



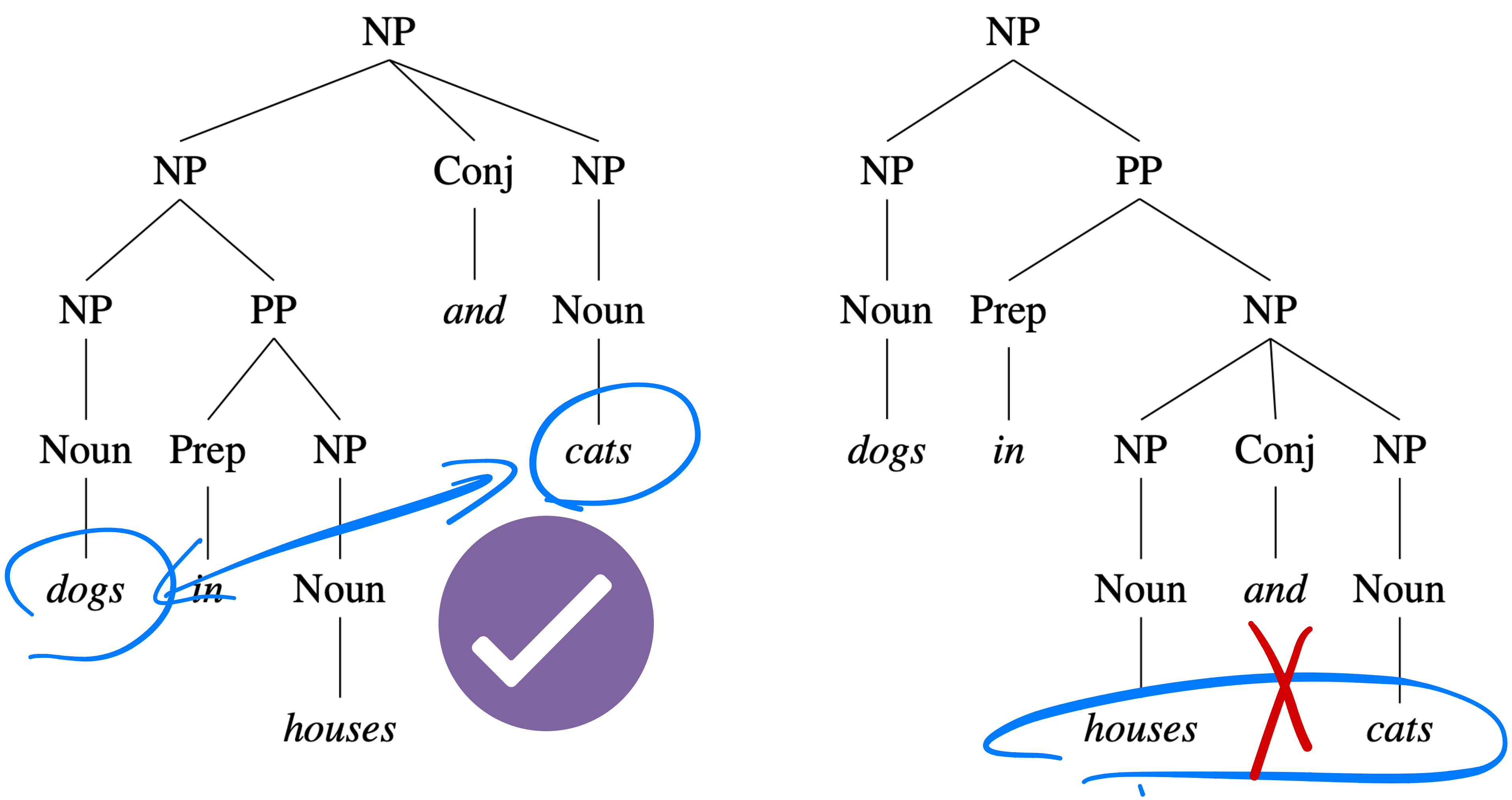
"into a bin" describes the resulting location of the sacks



sacks to be dumped are the ones which are already "into a bin"

Coordination Ambiguity

- *dogs in houses and cats*



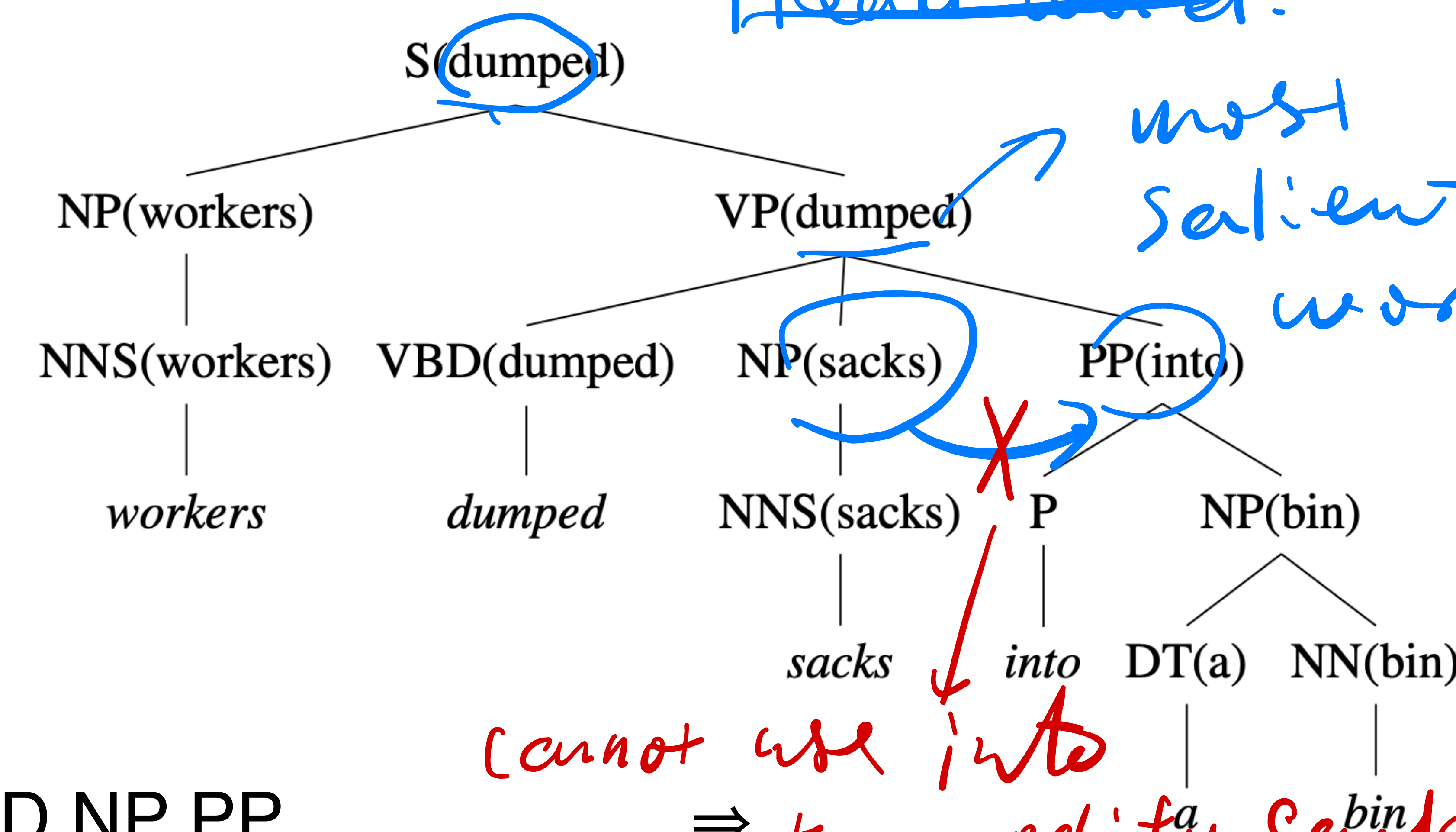
- *dogs* is semantically a better conjunct for *cats* than *houses* (dogs can't fit into houses!)

Solution: Head Lexicalisation

Non Terminal

- Record head word with parent symbols
  - the most salient child of a constituent, usually the noun in a NP, verb in a VP etc

~~Head word~~.



most salient word.

cannot use into to modify sacks

- VP → VBD NP PP ⇒

  VP(dumped) → VBD(dumped) NP(sacks) PP(into)

Head Lexicalisation

- Incorporate head words into productions, such that the most important links between words is captured
  - rule captures correlations between head tokens of phrases
  - VP(dumped) / NP(sacks) for PP(into)
- Grammar symbol inventory expands massively!
  - Many of the productions much too specific, seen very rarely
  - Learning more involved to avoid sparsity problems (e.g., zero probabilities)

low frequency

A Final Word

- PCFGs widely used, and there are efficient parsers available.
  - Collins parser, Berkeley parser, Stanford parser
  - all use some form of lexicalisation
- But there are other grammar formalisms
  - Lexical function grammar
  - Head-driven phrase structure grammar
  - Next lecture: dependency grammar

Required Reading

- J&M3 Ch. 14 – 14.6 (skip 14.6.1)