19CSE453 – Natural Language Processing

Parsing

By

Ms. Kavitha C.R.

Dept. of Computer Science and Engineering
Amrita School of Engineering, Bengaluru

Amrita Vishwa Vidyapeetham

Handling Unknown words

Unknown words are **a major problem that makes the Natural Language Processing (NLP) impossible to correctly analyze the meaning of the sentence**

The aim is to provide a model that will allow the NLP to correctly diagnose unknown words and replaced by the correct words.

Compare the words in the dictionary, find similar words and unknown words

Then Unknown words must be analyzed further to correct the words

In order to understand the characteristics of the unknown words which can be used as the guidelines for solving this problem, messages from various sources including both online and offline were collected that covers all levels of language, formal, semi-formal, and non-formal.

Then save these messages as text files. These text files were used as input to analyze for the characteristics of the unknown words which is finally classified into 7 types as follows:

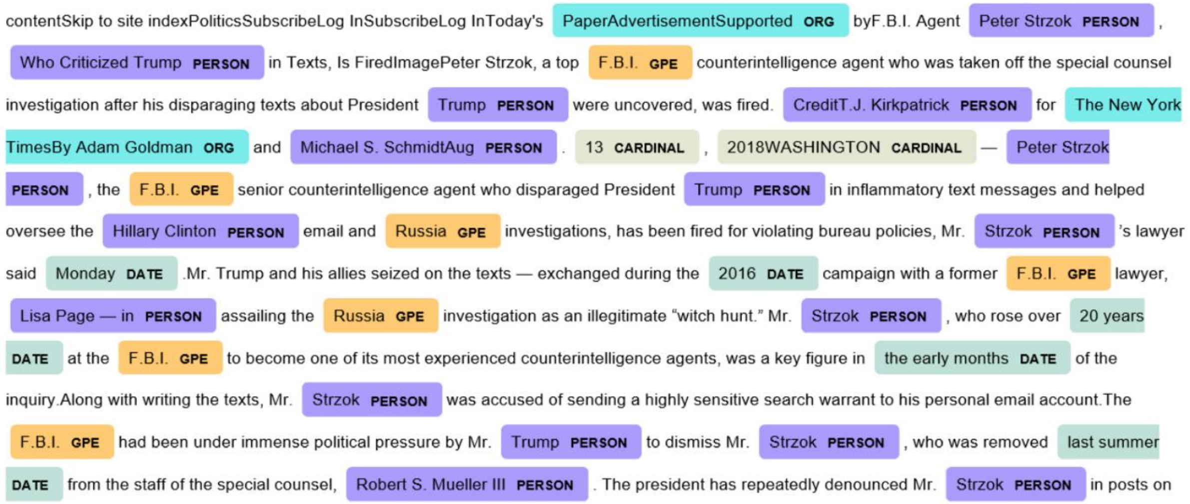
Excess of alphabets, missing of alphabets, Repetition of alphabets, Typo error, Misplacement of alphabets, Slang words and mixed type error

Named Entities

1. Named Entity Recognition is **one of the key entity detection methods in NLP**.
2. Named entity recognition is a NLP technique that can automatically scan entire articles and pull out some fundamental entities in a text and classify them into predefined categories.

Entities may be,

- •Organizations,
- •Quantities,
- •Monetary values,
- •Percentages, and more.
- •People names
- •Company names
- •Geographic locations (Both physical and political)
- •Product names
- •Dates and times
- •Amounts of money
- •Names of events



3. In simple words, Named Entity Recognition is the process of detecting the named entities such as person names, location names, company names, etc. from the text.

4. It is also known as entity identification or entity extraction or entity chunking.

For Example:

Ousted WeWork founder Adam Neumann lists his Manhattan penthouse for $37.5 million

[organization] **[person]** **[location]** **[monetary value]**

5. With the help of named entity recognition, we can extract key information to understand the text, or merely use it to extract important information to store in a database.

6. The entity detection can be seen in many applications such as

•Automated Chatbots,

•Content Analyzers,

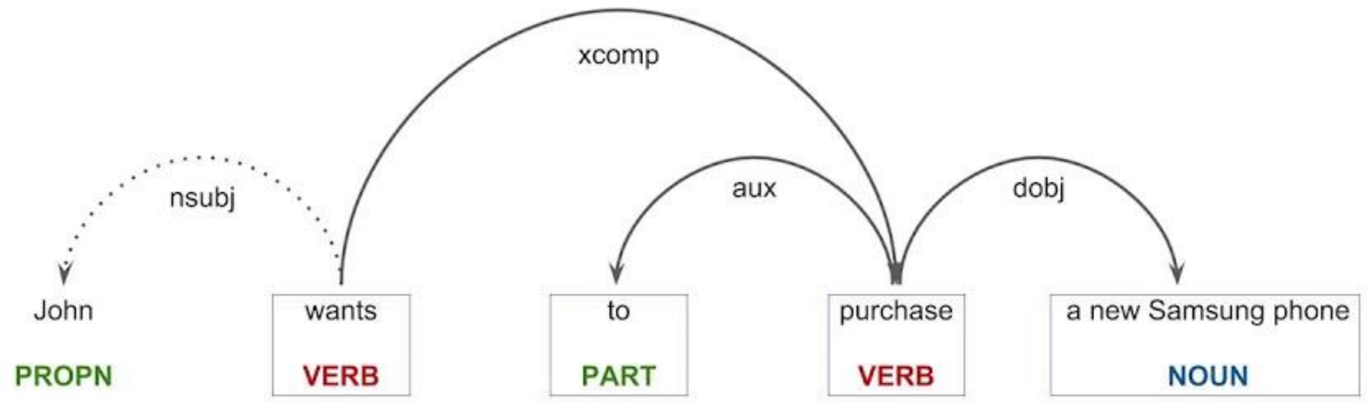
•Consumer Insights, etc.

Commonly used types of named entity are

Named Entity Type	Example
ORGANIZATION	WHO
PERSON	President Obama
LOCATION	Mount Everest
DATE	2020-07-10
TIME	12:50 P.M.
MONEY	One Million Dollars
PERCENT	98.24%
FACILITY	Washington Monument
GPE	North West America

Named Entity Extraction

- Identification of noun phrases
- Noun phrases : connected by direct subject or object relationships
- Sentence: ***John*** wants to purchase ***a new Samsung phone***



PART - particle

Multi-word Expressions

Multiword expressions (MWEs) are expressions which are made up of at least 2 words and which can be syntactically and/or semantically distinctive in nature.

Moreover, they act as a single unit at some level of linguistic analysis.

Multi-word Expressions (MWEs) are word combinations with linguistic properties that cannot be predicted from the properties of the individual words or the way they have been combined.
MWEs occur frequently and are usually highly domain-dependent

MWEs can be regarded as lying at the interface of grammar and lexicon, usually being instances of well productive syntactic patterns but nevertheless showing a peculiar lexical behaviour

Examples: Idioms as “kick the bucket“
compound nouns as “telephone box“ and “post office“
proper names as “San Francisco“

Syntax Is Not Morphology

- Morphology deals with the internal structure of words
  - Syntax deals with combinations of words
  - Phrases and sentences
- Morphology is often irregular
- Syntax has its irregularities, but it is usually regular
- Syntax is mostly made up of general rules that apply across-the-board

Syntax Is Not Semantics

Semantics is about meaning; syntax is about structure alone

- A sentence can be syntactically well-formed but semantically ill-formed:

Eg: *Colorless green ideas sleep furiously*

- Some well-known linguistic theories attempt to “read” semantic representations off of syntactic representations in a compositional fashion

Constituency

- One way of viewing the structure of a sentence is as a collection of nested constituents

– **constituent**: a group of neighboring words that "go together" (or relate more closely to one another than to other words in the sentence)

- Constituents larger than a word are called *phrases*
- Phrases can contain other phrases

Noun Phrases (NPs)

- The elephant arrived.
- It arrived.
- Elephants arrived.
- The big ugly elephant arrived.
- The elephant I love to hate arrived.

Prepositional Phrases (PPs)

- I arrived on Tuesday.
- I arrived in March.
- I arrived under the leaking roof.

Every **prepositional phrase** contains a **noun phrase**.

Sentences or Clauses (Ss)

- John loves Mary.
- John loves the woman he thinks is Mary.
- Sometimes, John thinks he is Mary.
- It is patently false that sometimes John thinks he is Mary.

Constituents: Heads and dependents

There are different kinds of constituents:

Noun phrases: the man, a girl with glasses, Illinois

Prepositional phrases: with glasses, in the garden

Verb phrases: eat sushi, sleep, sleep soundly

Every phrase has a **head**:

Noun phrases: the <u>man</u>, a <u>girl</u> with glasses, <u>Illinois</u>

Prepositional phrases: <u>with</u> glasses, <u>in</u> the garden

Verb phrases: <u>eat</u> sushi, <u>sleep</u>, <u>sleep</u> soundly

The other parts are its **dependents**.

Dependents are either **arguments** or **adjuncts**

CONTEXT-FREE GRAMMARS

Context-free grammar G is a 4-tuple.

G = (V, T, S, P)

These parameters are as follows:

- **V – *Set of variables* (also called as Non-terminal symbols)**
- **T – *Set of terminal symbols* (lexicon)**

  The symbols that refer to words in a language are called terminal symbols.
  Lexicon is a set of rules that introduce these symbols.
- **S – *Designated start symbol* (one of the non-terminals, S ∈ V)**
- **P – *Set of productions* (also called as rules).**

  Each rule in P is of the form A → s, where
  s is a sequence of terminals and non-terminals. It is from (T U V)*, infinite set of strings.

A grammar G generates a language L.

The grammars are called “context-free” because there is no context in the LHS of rules—there is just one symbol.

Context-Free Rules

- S → NP VP
- NP → Det Noun
- VP → Verb NP
- Det → the|a
- Noun → boy | girl| hotdogs
- Verb → likes| hates| eats

Building Noun Phrases

- NP → Determiner NounBar
- NP →ProperNoun
- NounBar → Noun
- NounBar → AP NounBar
- NounBar → NounBar PP
- AP → Adj AP
- AP → Adj
- PP → Preposition NP

One way to look at context-free grammars is as declarative programs, Instead of specifying how the task is to be accomplished…

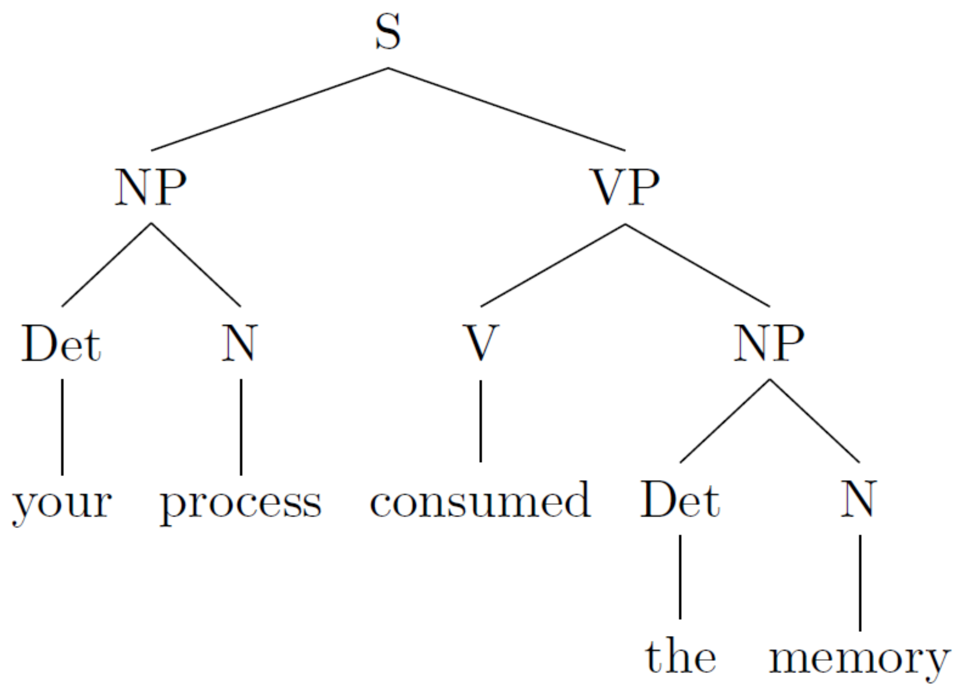
- **How sentences are to be generated**
- **How sentences are to be parsed**

- Grammatical: said of a sentence in the language
- Ungrammatical: said of a sentence not in the language
- Derivation: sequence of top-down production steps
- Parse tree: graphical representation of the derivation

A string is grammatical if there exists a derivation for it.

- S → NP VP
- NP → Det N
- VP → V NP
- Det → your | the
- N → process| memory
- V → consumed

A (Constituency) Parse Tree



G = (V, T, S, P)

V = {S, NP, VP, PP, Det, Noun, Verb, Aux, Pre}

T = {'a', 'ate', 'cake', 'child', 'fork', 'the', 'with'}

S = S

P = { S → NP VP

NP → Det Noun | NP PP

PP → Pre NP

VP → Verb NP

Det → 'a' | 'the'

Noun → 'cake' | 'child' | 'fork'

Pre → 'with'

Verb → 'ate'}

Derivation:

S → NP VP

→ Det Noun VP

→ the Noun VP

→ the child VP

→ the child Verb NP

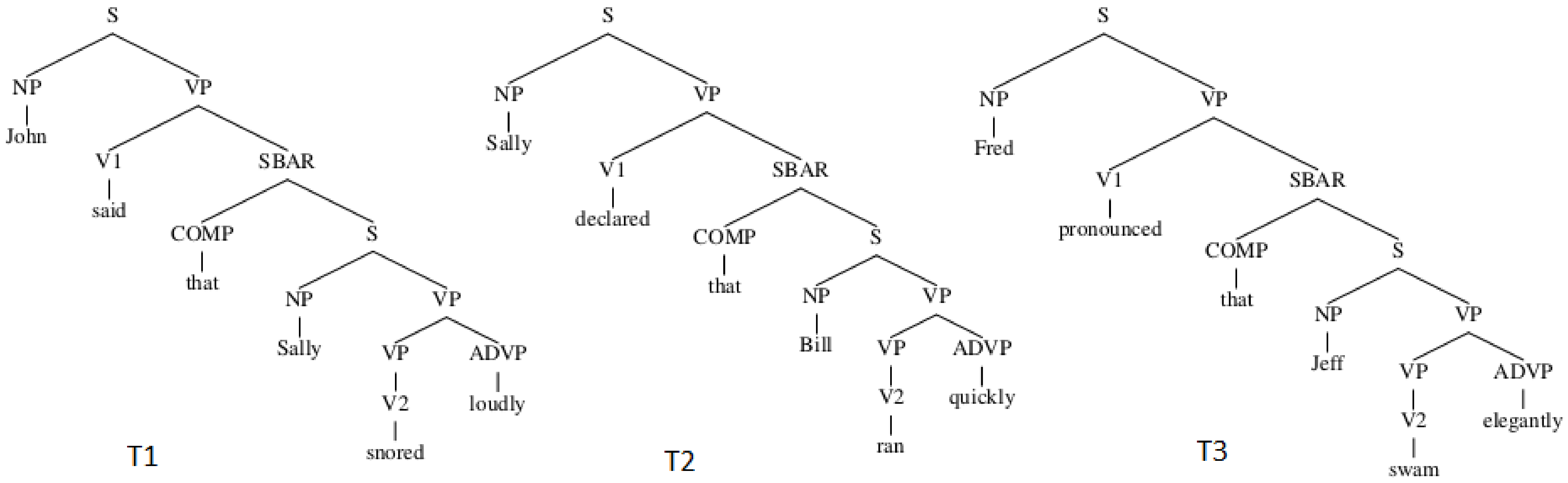
→ the child ate NP

→ the child ate Det Noun

→ the child ate a Noun

→ the child ate a cake

Example Parse Tree

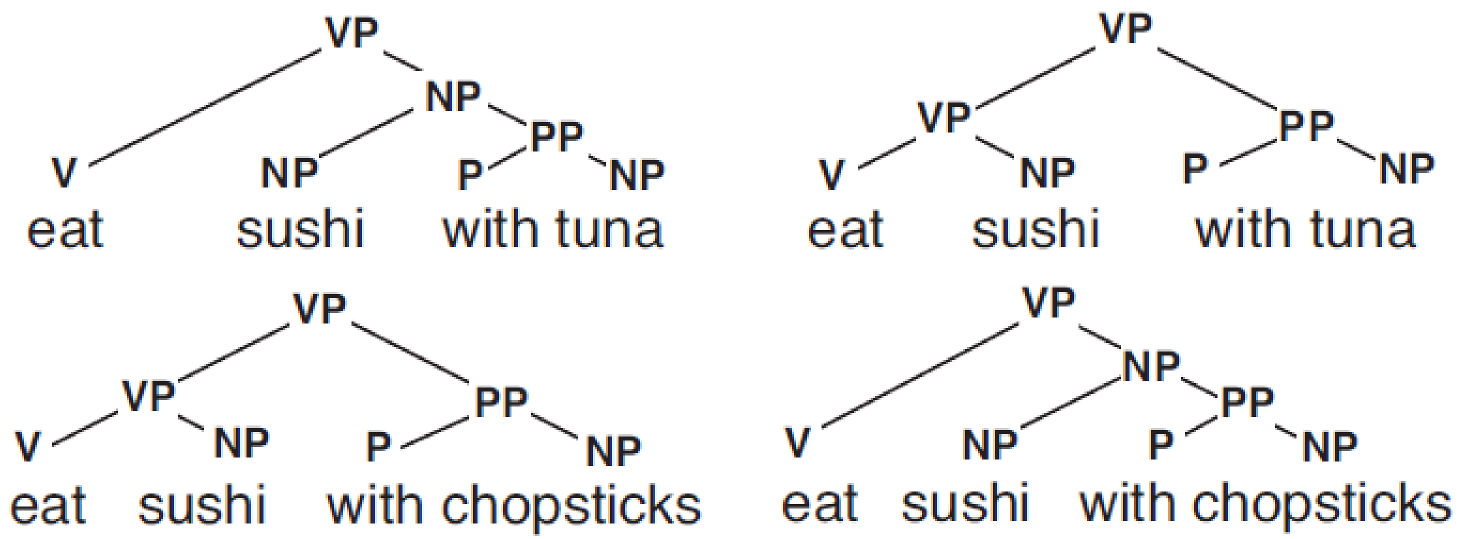


Parsing is the process of taking a string and a grammar and returning all possible parse trees for that string

That is, find all trees, whose root is the start symbol S, which cover exactly the words in the input

Grammars are ambiguous

A grammar might generate multiple trees for a sentence:

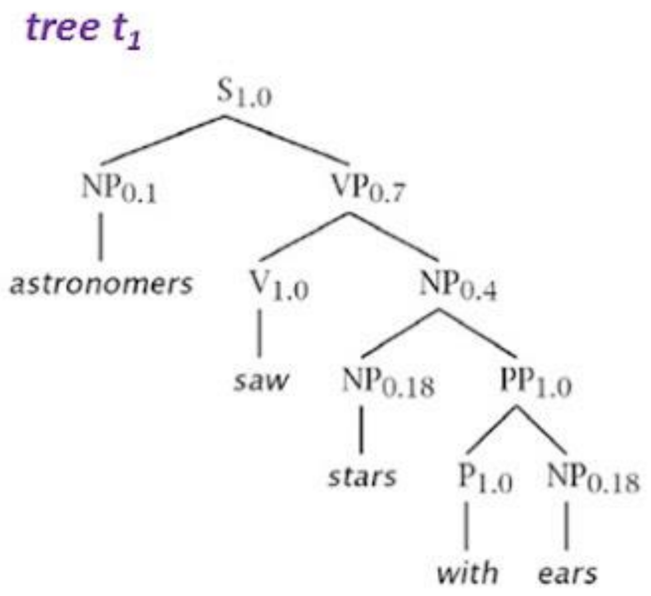


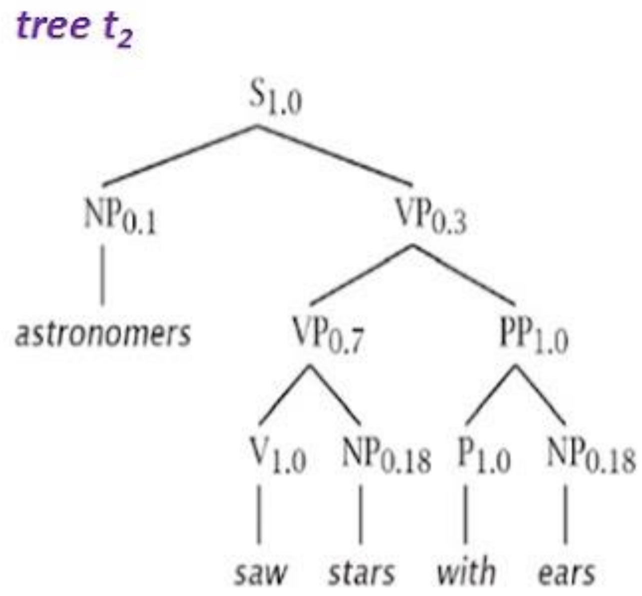
What's the most likely parse τ for sentence S ?

We need a model of $P(\tau \mid S)$

Ambiguity in the grammar

- S → NP VP
- VP → V NP
- VP → VP PP
- PP → P NP
- NP → NP PP
- NP → boy| girl| hotdogs| park| astronomers| stars| ears
- V → likes| hates| eats| sees|saw
- P → in|with





Probabilistic Context Free Grammar (PCFG)

Probabilistic Context Free Grammar (PCFG) is an extension of Context Free Grammar (CFG) with a probability for each production rule.

Ambiguity is the reason for using probabilistic version of CFG.

For instance, some sentences may have more than one underlying derivation.
That is, the sentence can be parsed in more than one ways.
In this case, the parse of the sentence become ambiguous.

To eliminate this ambiguity, we can use PCFG to find the probability of each parse of the given sentence.

A PCFG is made up of a CFG and a set of probabilities for each production rule of CFG.

A PCFG can be formally defined as follows:

A probabilistic context free grammar G is a quintuple G = (V, T, S, R, P) where

- V is set of non-terminal (variable) symbols
- T is set of terminal symbols
- S is the start symbol and
- R is the set of production rules where each rule of the form $A \rightarrow s$

A probability $P(A \rightarrow s)$ for each rule in R. The properties governing the probability are as follows:

- $P(A \rightarrow s)$ is a conditional probability of choosing a rule $A \rightarrow s$ in a left-most derivation, given that A is the non-terminal that is expanded.
- The value for each probability lies between 0 and 1.
- The **sum of all probabilities of rules with A as the left hand side non-terminal** should be equal to 1.

$$\sum_{A \rightarrow s \in R:\, A=LHS} P(A \rightarrow s) = 1$$

Example PCFG

Probabilistic Context Free Grammar G = (V, T, S, R, P)

- **V = {S, NP, VP, PP, Det, Noun, Verb, Pre}**
- **T = {‘a’, ‘ate’, ‘cake’, ‘child’, ‘fork’, ‘the’, ‘with’}**
- **S = S**
- **R = { S → NP VP**

NP → Det Noun | NP PP

PP → Pre NP

VP → Verb NP

Det → ‘a’ | ‘the’

Noun → ‘cake’ | ‘child’ | ‘fork’

Pre → ‘with’

Verb → ‘ate’ }

- P = R with associated probability as in the table below;

Rule	*Probability*	*Rule*	*Probability*
S → NP VP	1.0	Det → ‘a’ Det → ‘the’	0.5 0.5
NP → NP PP NP → Det Noun	0.6 0.4	Noun → ‘cake’ Noun → ‘child’ Noun → ‘fork’	0.4 0.3 0.3
PP → Pre NP	1.0	Pre → ‘with’	1.0
VP → Verb NP	1.0	Verb → ‘ate’	1.0

$$\sum_{A \to s \in R:\, A = NP} P(A \to s) = P(NP \to Det\ Noun) + P(NP \to NP\ PP)$$

$$= 0.4 + 0.6 = 1$$

PCFG to resolve ambiguity

Given a parse tree t, with the production rules $\alpha_1 \rightarrow \beta_1, \alpha_2 \rightarrow \beta_2, \ldots$ (ie., $\alpha_i \rightarrow \beta_i \in R$), we can find the probability of tree t using PCFG as

$$P(t) = \prod_{i=1}^{n} P(\alpha_i \rightarrow \beta_i)$$

As per the equation, the probability *P(t)* of parse tree is the product of probabilities of production rules in the tree t.

Find the probability of the parse tree t_1 given below:

$$P(t) = \prod_{i=1}^{n} P(\alpha_i \rightarrow \beta_i)$$

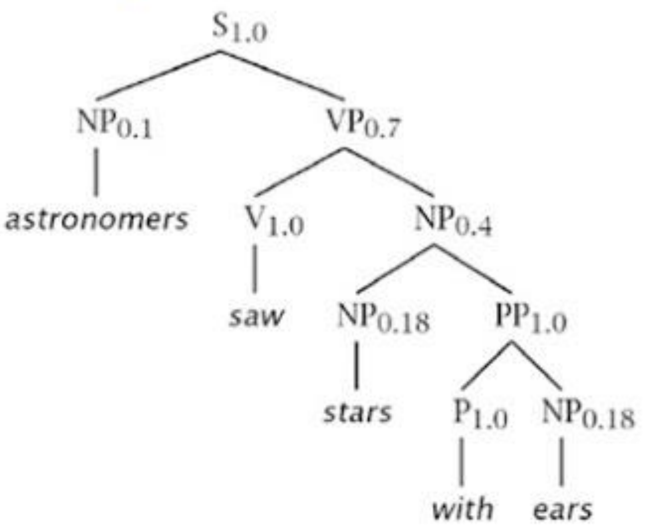
$$= P(S \rightarrow NP\ VP) * P(NP \rightarrow astronomers) * P(VP \rightarrow V\ NP)$$
$$* P(V \rightarrow saw) * P(NP \rightarrow NP\ PP) * P(NP \rightarrow stars)$$
$$* P(PP \rightarrow P\ NP) * P(P \rightarrow with) * P(NP \rightarrow ears)$$

= 1.0 * 0.1 * 0.7 * 1.0 * 0.4 * 0.18 * 1.0 * 1.0 * 0.18

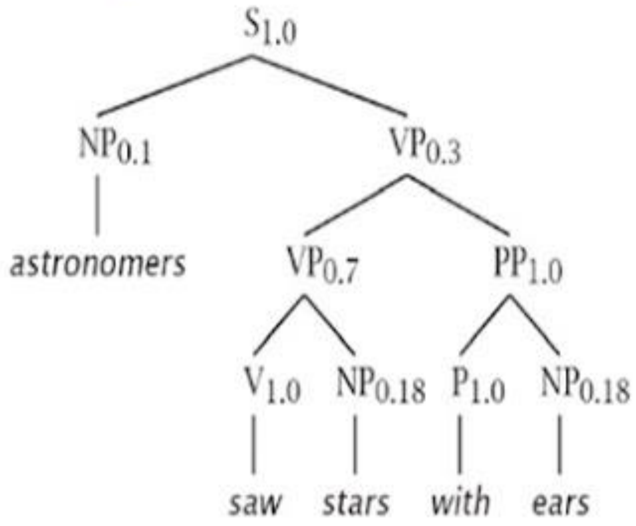
= 0.0009072

The probability of the parse tree *t* is calculated as 0.0009072

tree t_1



tree t_2



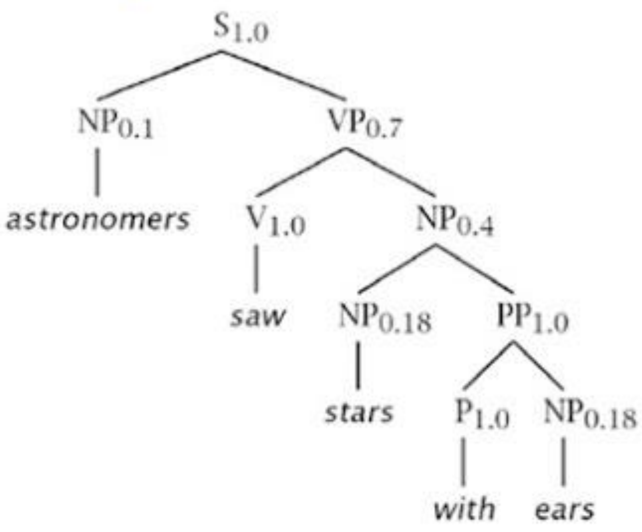
Probability of a sentence

Probability of a sentence is the sum of probabilities of all parse trees that can be derived from the sentence under PCFG.

$$\sum_{i=1}^{n} P(t_i)$$

Example: PCFG to resolve ambiguity

tree t_1



<u>*Probability of tree t_1*</u>

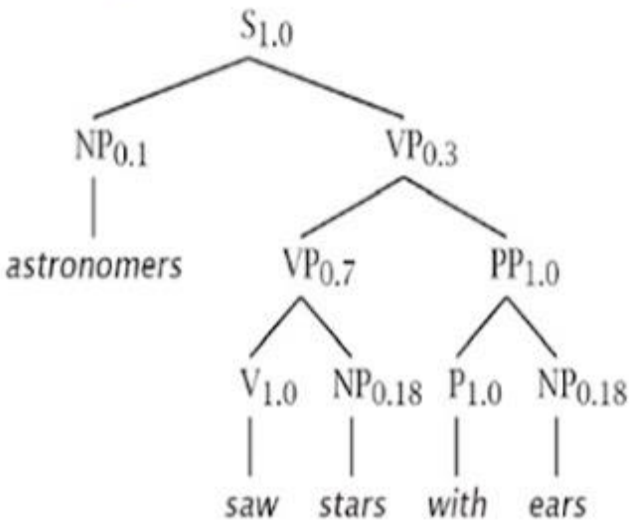
$$P(t_1) = \prod_{i=1}^{n} P(\alpha_i \rightarrow \beta_i)$$

$$= P(S \rightarrow NP\ VP) * P(NP \rightarrow astronomers) * P(VP \rightarrow V\ NP)$$
$$* P(V \rightarrow saw) * P(NP \rightarrow NP\ PP) * P(NP \rightarrow stars)$$
$$* P(PP \rightarrow P\ NP) * P(P \rightarrow with) * P(NP \rightarrow ears)$$

= 1.0 * 0.1 * 0.7 * 1.0 * 0.4 * 0.18 * 1.0 * 1.0 * 0.18

= 0.0009072

tree t_2



<u>*Probability of tree t_2*</u>

$P(t_2)$ = 1.0 * 0.1 * 0.3 * 0.7 * 1.0 * 0.18 * 1.0 * 1.0 * 0.18

= 0.0006804

Probability of the sentence "astronomers saw the stars with ears"

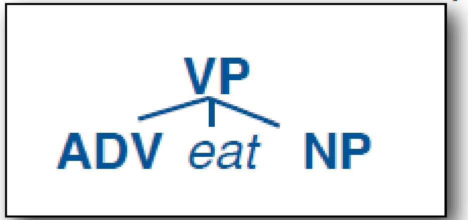
$$\sum_{i=1}^{n} P(t_i) = P(t_1) + P(t_2) = 0.0009072 + 0.0006804 = 0.001588$$

The probability of the parse tree t_1 is greater than the probability of parse tree t_2. Hence, t_1 is the more probable of the two parses.

Chomsky Normal Form

The right-hand side of a standard CFG can have an **arbitrary number of symbols** (terminals and nonterminals):

VP → ADV eat NP



A CFG in **Chomsky Normal Form** (CNF) allows only two kinds of right-hand sides:

- **Two nonterminals:** VP → ADV VP
- **One terminal:** VP → eat

Any CFG can be transformed into an equivalent CNF:

VP → ADVP $\mathbf{VP_1}$
$\mathbf{VP_1}$ → $\mathbf{VP_2}$ NP
$\mathbf{VP_2}$ → eat

Cocke Kasami Younger (CKY) 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)

Presumes a CFG in Chomsky Normal Form:

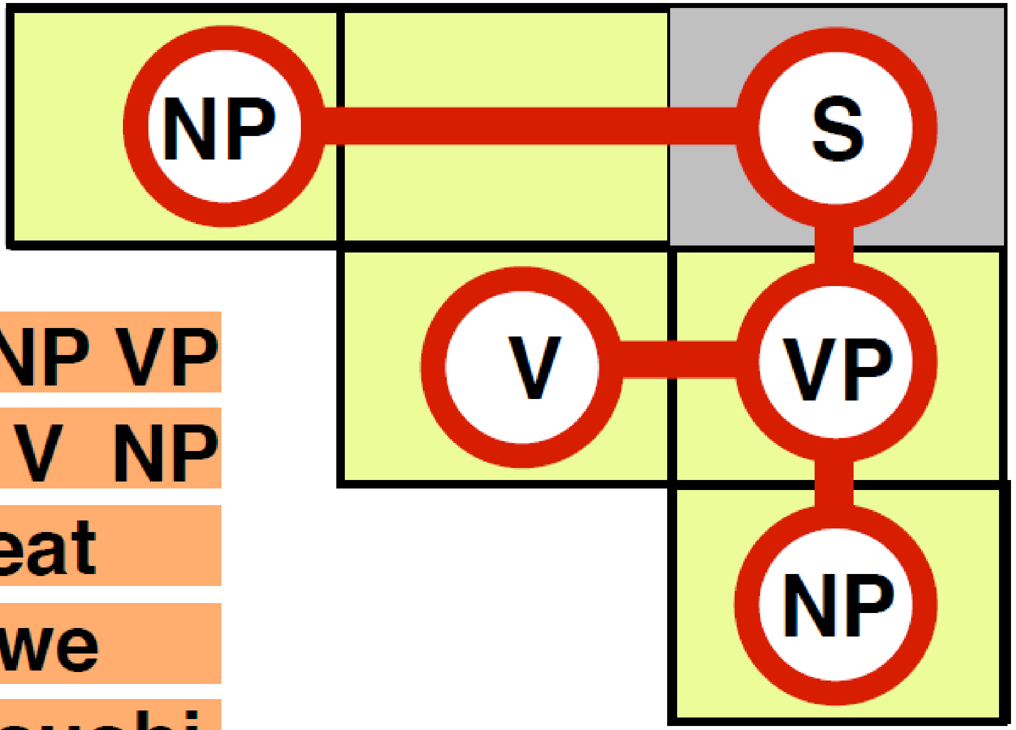
Rules are all either **A → B C** or **A → a**

(with **A,B,C** nonterminals and **a** a terminal)

Example

The CKY parsing algorithm

To recover the parse tree, each entry needs **pairs** of backpointers.



S → NP VP
VP → V NP
V → eat
NP → we
NP → sushi

We eat sushi

CKY Algorithm

1. Create the chart

(an $n \times n$ upper triangular matrix for an sentence with n words)

– Each cell $\mathrm{chart}[i][j]$ corresponds to the substring $w^{(i)} \ldots w^{(j)}$

2. Initialize the chart (fill the diagonal cells $\mathrm{chart}[i][i]$):

For all rules $X \rightarrow w^{(i)}$, add an entry X to $\mathrm{chart}[i][i]$

3. Fill in the chart:

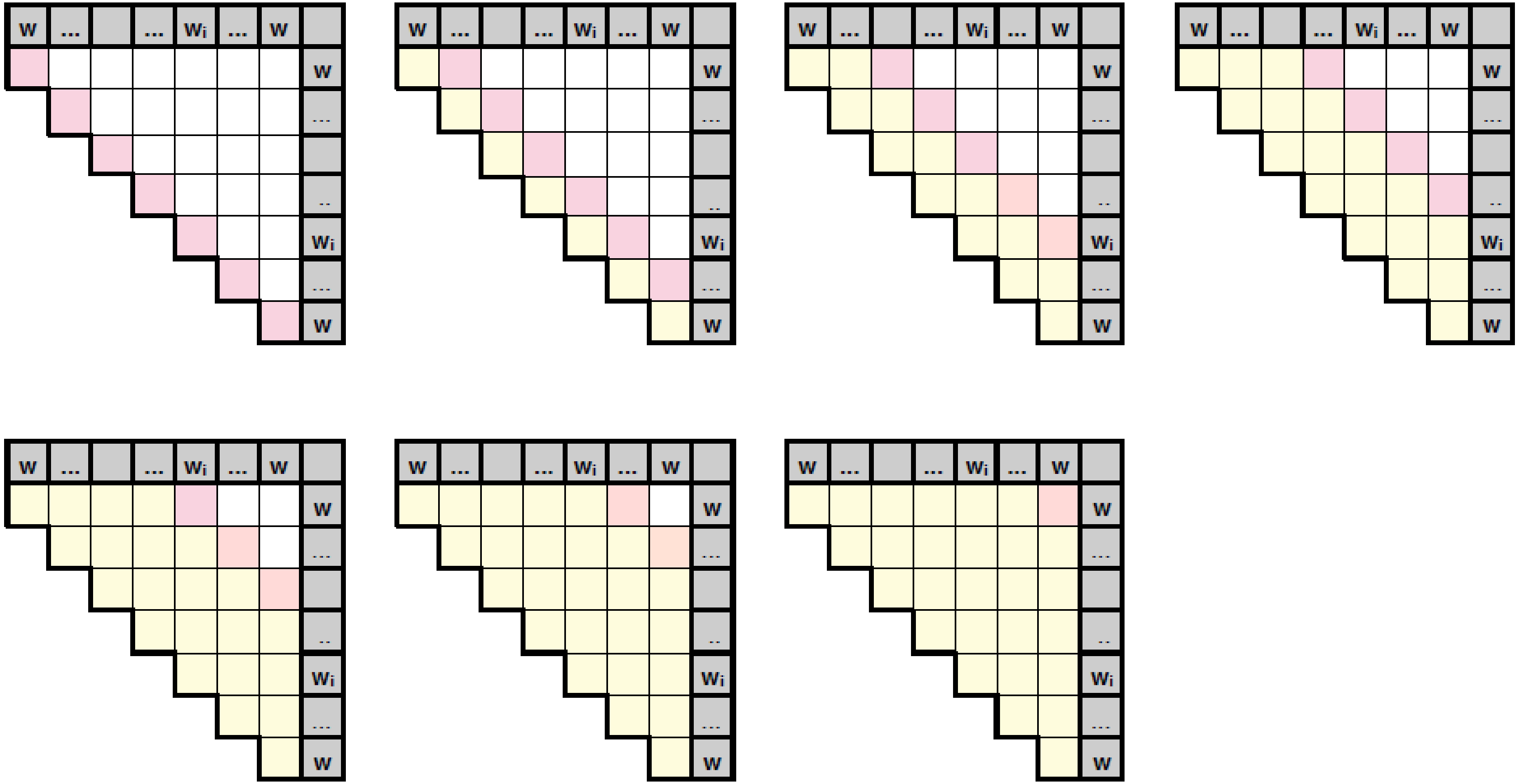
Fill in all cells $\mathrm{chart}[i][i+1]$, then $\mathrm{chart}[i][i+2]$, …,
until you reach $\mathrm{chart}[1][n]$ (the top right corner of the chart)

– To fill $\mathrm{chart}[i][j]$, consider all binary splits $w^{(i)} \ldots w^{(k)} | w^{(k+1)} \ldots w^{(j)}$

– If the grammar has a rule $X \rightarrow YZ$, $\mathrm{chart}[i][k]$ contains a Y and $\mathrm{chart}[k+1][j]$ contains a Z, add an X to $\mathrm{chart}[i][j]$ with two backpointers to the Y in $\mathrm{chart}[i][k]$ and the Z in $\mathrm{chart}[k+1][j]$

4. Extract the parse trees from the S in $\mathrm{chart}[1][n]$.

CKY: filling the chart



The CKY parsing algorithm

V buy	VP buy drinks	buy drinks with	VP buy drinks with milk
	V, NP drinks	drinks with	VP, NP drinks with milk
		P	PP

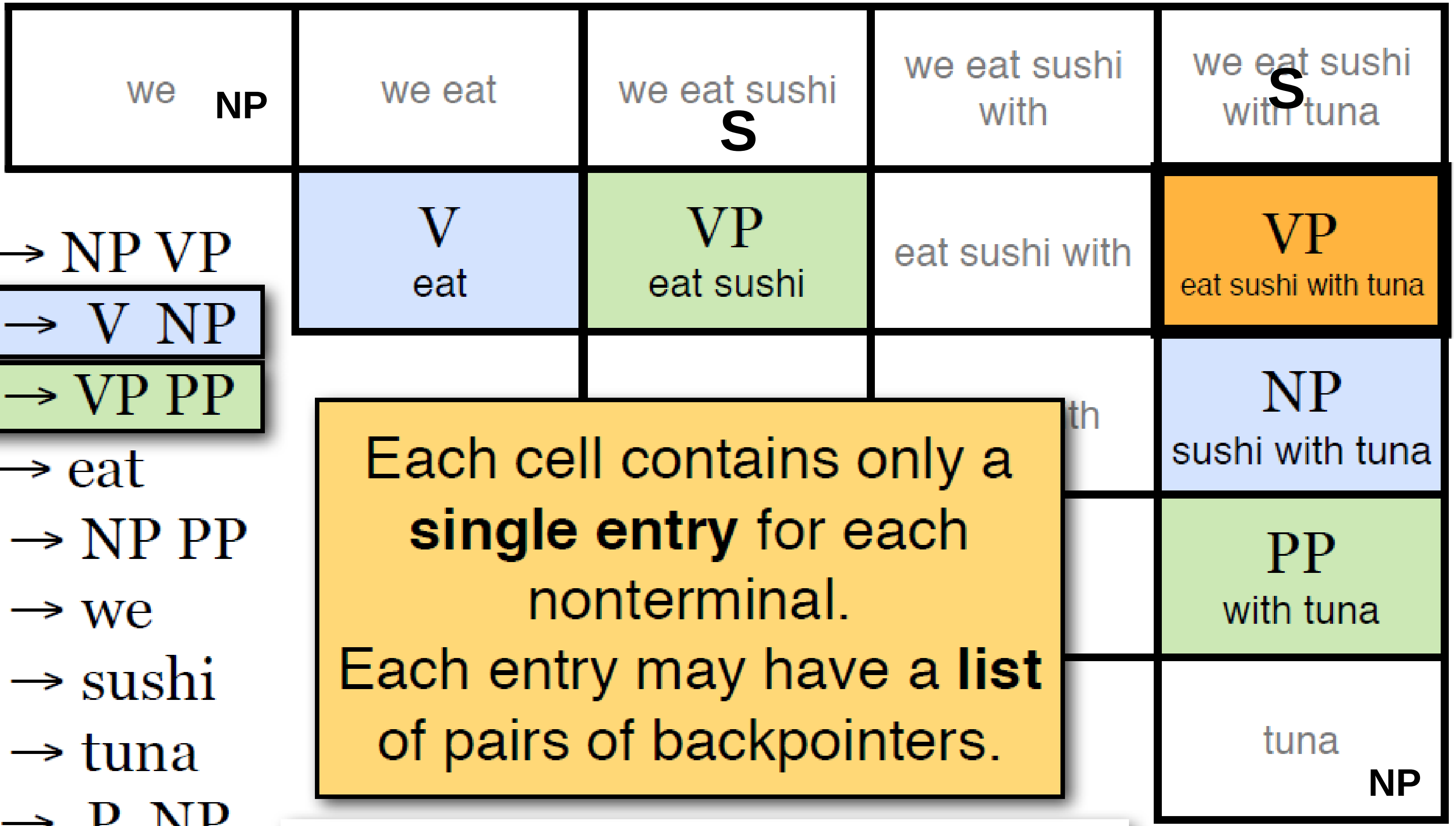
$S \rightarrow NP\ VP$
$VP \rightarrow V\ NP$
$VP \rightarrow VP\ PP$
$\boxed{V \rightarrow drinks}$
$NP \rightarrow NP\ PP$
$NP \rightarrow we$
$\boxed{NP \rightarrow drinks}$
$NP \rightarrow milk$
$PP \rightarrow P\ NP$
$P \rightarrow with$
V → buy

Each cell may have **one entry for each nonterminal**

We buy drinks with milk

The CKY parsing algorithm

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

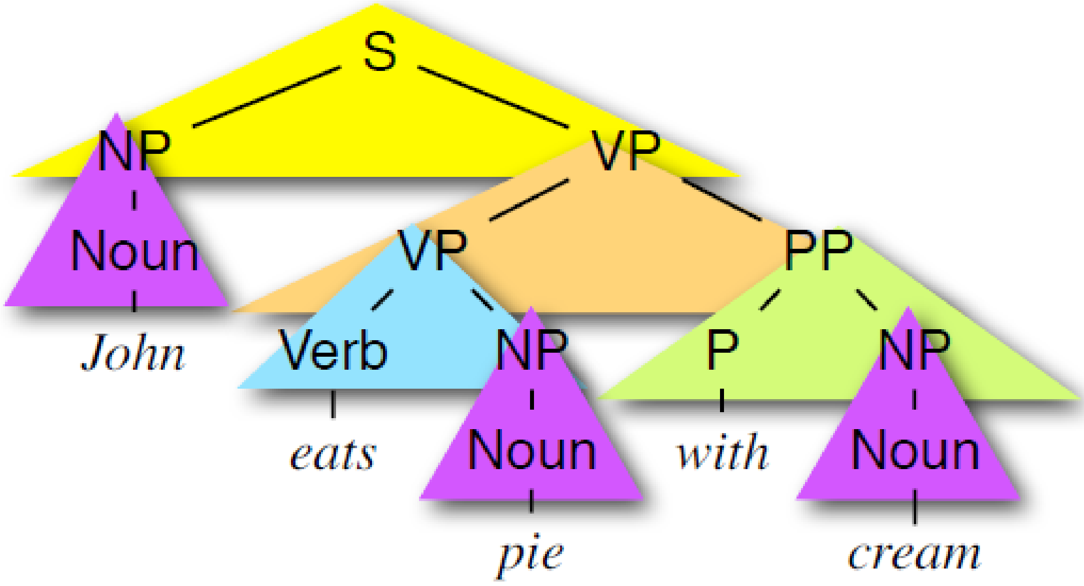


Each cell contains only a **single entry** for each nonterminal.
Each entry may have a **list** of pairs of backpointers.

We eat sushi with tuna

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

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

$= \mathbf{0.00384}$

Probabilistic CKY: Viterbi

Like standard CKY, but with probabilities.
Finding the most likely tree $\text{argmax}_\tau\, P(\tau, s)$ is similar to Viterbi for HMMs:

Initialization: every chart entry that corresponds to a **terminal** (entries X in `cell[i][i]`) has a Viterbi probability $P_{\text{VIT}}(X_{[i][i]}) = 1$

Recurrence: For every entry that corresponds to a **non-terminal** `X` in `cell[i][j]`, keep only the highest-scoring pair of backpointers to any pair of children (`Y` in `cell[i][k]` and `Z` in `cell[k+1][j]`):
$P_{\text{VIT}}(X_{[i][j]}) = \text{argmax}_{Y,Z,k}\, P_{\text{VIT}}(Y_{[i][k]}) \times P_{\text{VIT}}(Z_{[k+1][j]}) \times P(X \rightarrow Y\ Z \mid X)$

Final step: Return the Viterbi parse for the start symbol S in the top `cell[1][n]`.

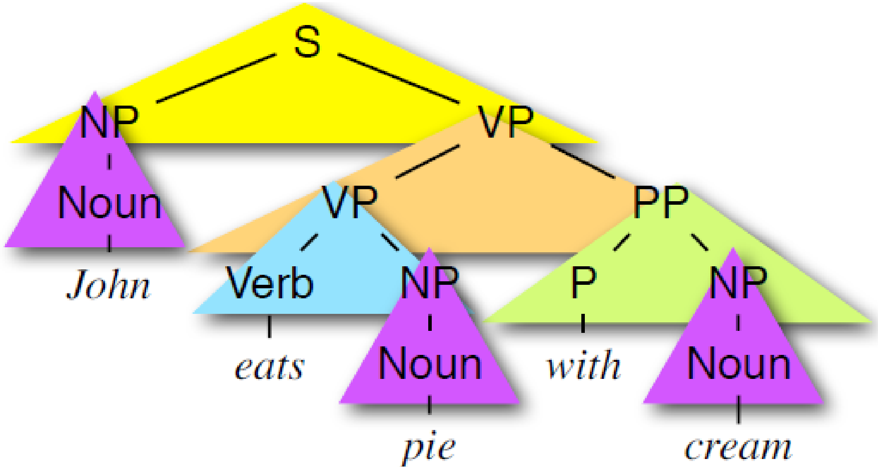
Probabilistic CKY

Input: POS-tagged sentence

John_N eats_V pie_N with_P cream_N

John	eats	pie	with	cream	
N 1.0 NP 0.2	S 0.8 · 0.2 · 0.3	S 0.8 · 0.2 · 0.06		S 0.2 · 0.0036 · 0.8	**John**
	V 1.0 VP 0.3	VP 1 · 0.3 · 0.2 = 0.06		VP max(1.0 · 0.008 · 0.3, 0.06 · 0.2 · 0.3)	**eats**
		N 1.0 NP 0.2		NP 0.2 · 0.2 · 0.2 = 0.008	**pie**
			P 1.0	PP 1 · 1 · 0.2	**with**
				N 1.0 NP 0.2	**cream**

S	$\rightarrow$	NP VP	0.8
S	$\rightarrow$	S conj S	0.2
NP	$\rightarrow$	Noun	0.2
NP	$\rightarrow$	Det Noun	0.4
NP	$\rightarrow$	NP PP	0.2
NP	$\rightarrow$	NP conj NP	0.2
VP	$\rightarrow$	Verb	0.3
VP	$\rightarrow$	Verb NP	0.3
VP	$\rightarrow$	Verb NP NP	0.1
VP	$\rightarrow$	VP PP	0.3
PP	$\rightarrow$	P NP	1.0



Thank You