CS447: Natural Language Processing

http://courses.engr.illinois.edu/cs447

Lecture 17: CFG Parsing

Julia Hockenmaier

juliahmr@illinois.edu

3324 Siebel Center

Introduction to Syntax

Previous key concepts

NLP tasks dealing with **words**...

– POS-tagging, morphological analysis

... requiring **finite-state representations**,

– Finite-State Automata and Finite-State Transducers

... the corresponding **probabilistic models**,

– Probabilistic FSAs and Hidden Markov Models
– Estimation: relative frequency estimation, EM algorithm

... and **appropriate search algorithms**

– Dynamic programming: Viterbi

The next key concepts

NLP tasks dealing with **sentences**...

- Syntactic parsing and semantic analysis

… require (at least) **context-free representations**,

- Context-free grammars, dependency grammars, unification grammars, categorial grammars

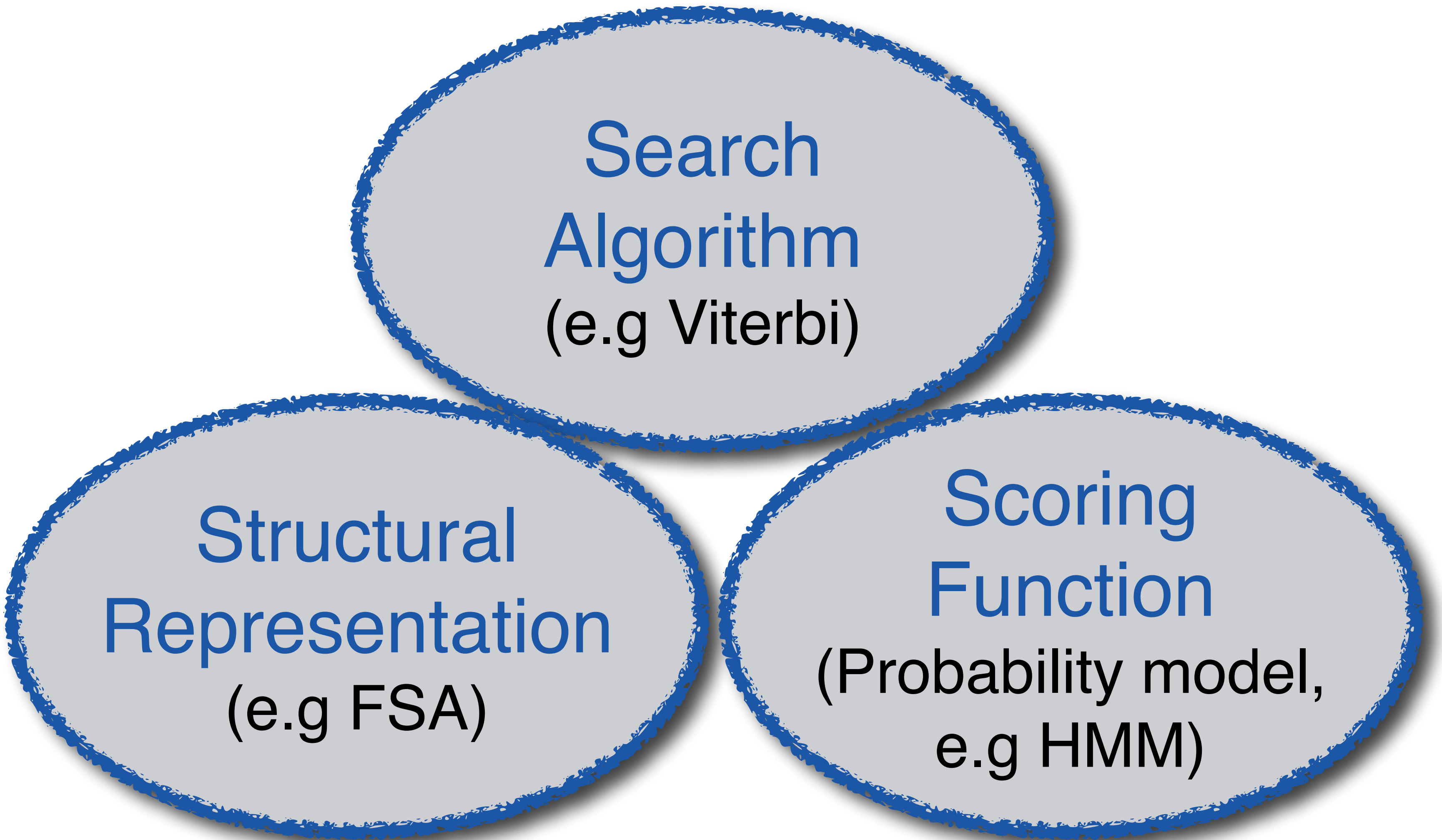
… the corresponding **probabilistic models**,

- Probabilistic Context-Free Grammars

… and appropriate **search algorithms**

- Dynamic programming: CKY parsing

Dealing with ambiguity



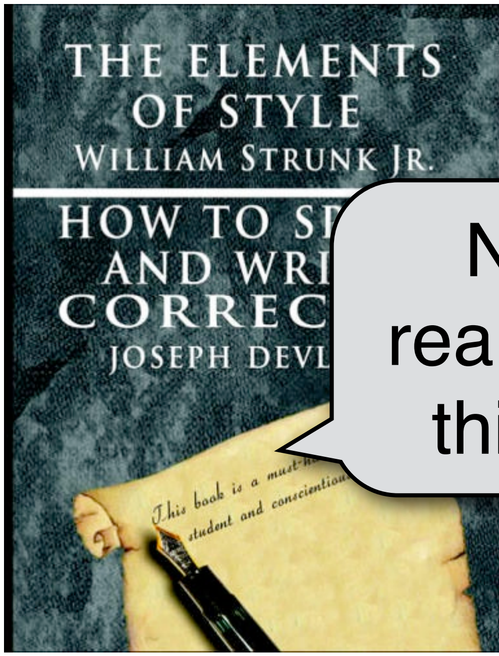
Today's lecture

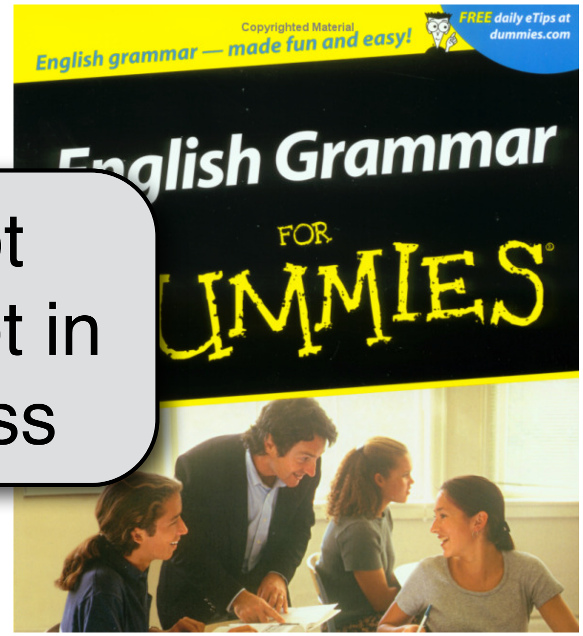
Introduction to natural language syntax ('grammar'):

Part 1: Introduction to Syntax (constituency, dependencies,…)
Part 2: Context-free Grammars for natural language
Part 3: A simple CFG for English
Part 4: The CKY parsing algorithm

Reading: Chapter 12 of Jurafsky & Martin

What is grammar?





No, not really, not in this class

Grammar formalisms:

A precise way to define and describe the structure of sentences.

There are many different formalisms out there.

What is grammar?

Grammar formalisms

(= syntacticians' programming languages)

A precise way to define and describe the structure of sentences.

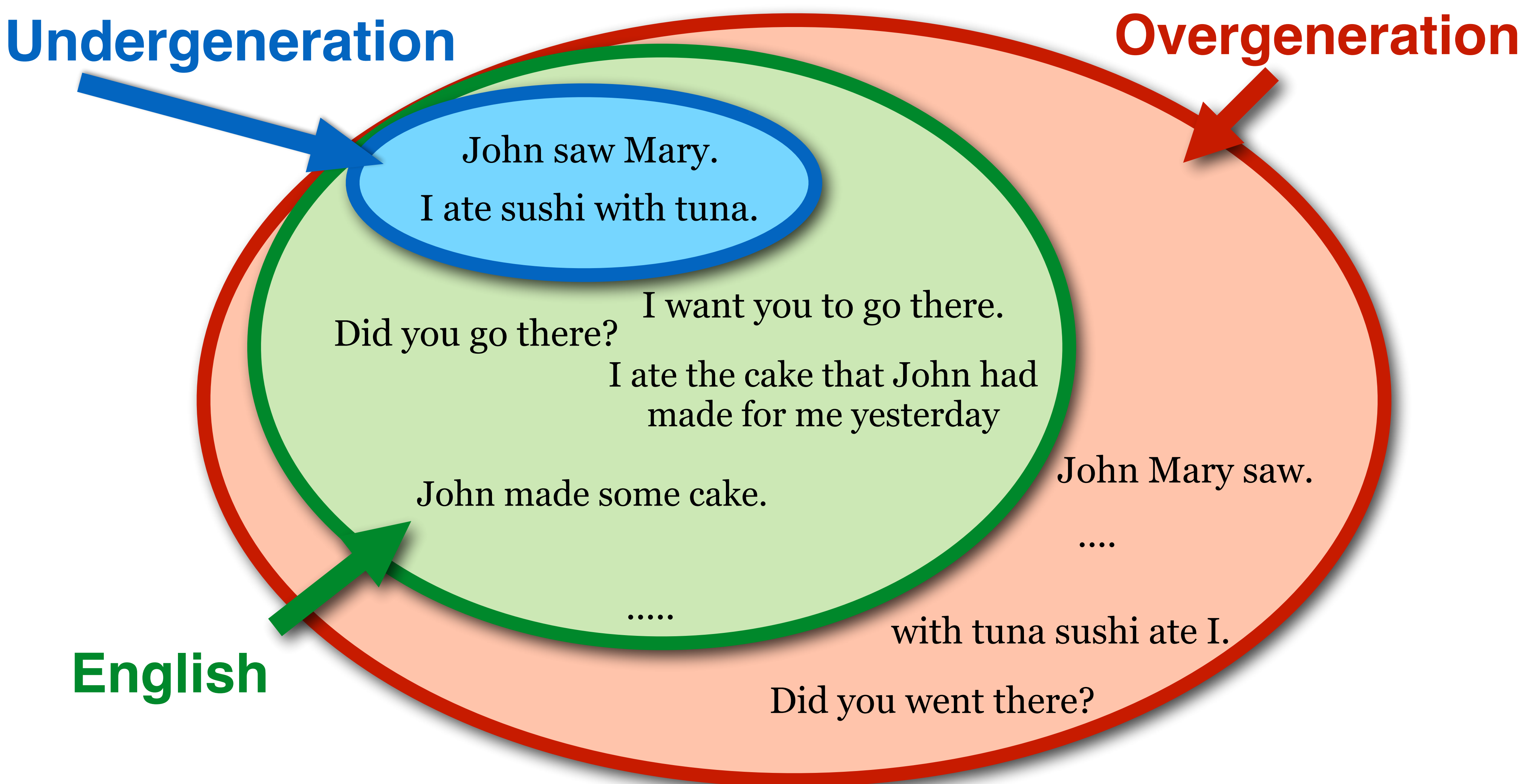
(N.B.: There are many different formalisms out there, which each define their own data structures and operations)

Specific grammars

(= syntacticians' programs)

Implementations (in a particular formalism) for a particular language (English, Chinese,....)

Can we define a program that generates all English sentences?



Can we define a program that generates all English sentences?

Challenge 1: Don't *undergenerate*!

(Your program needs to cover a lot different constructions)

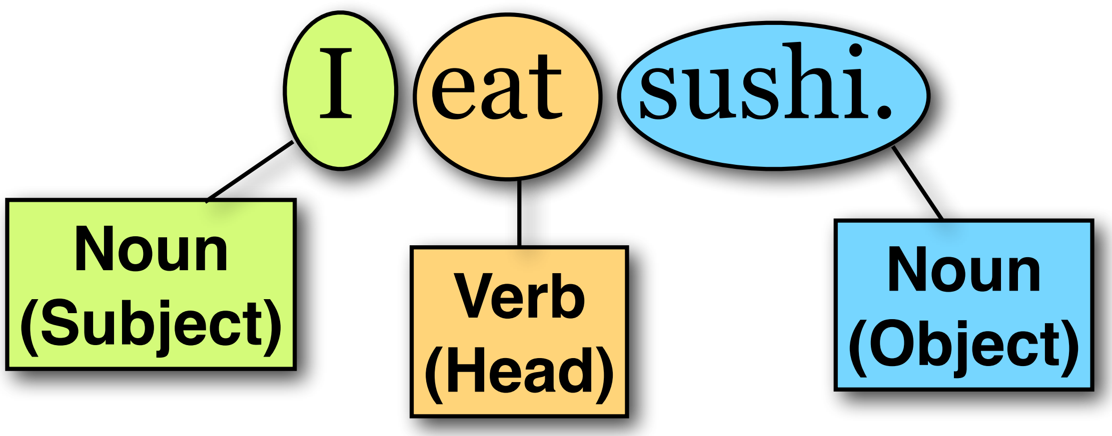
Challenge 2: Don't *overgenerate*!

(Your program should not generate word salad)

Challenge 3: Use a finite program!

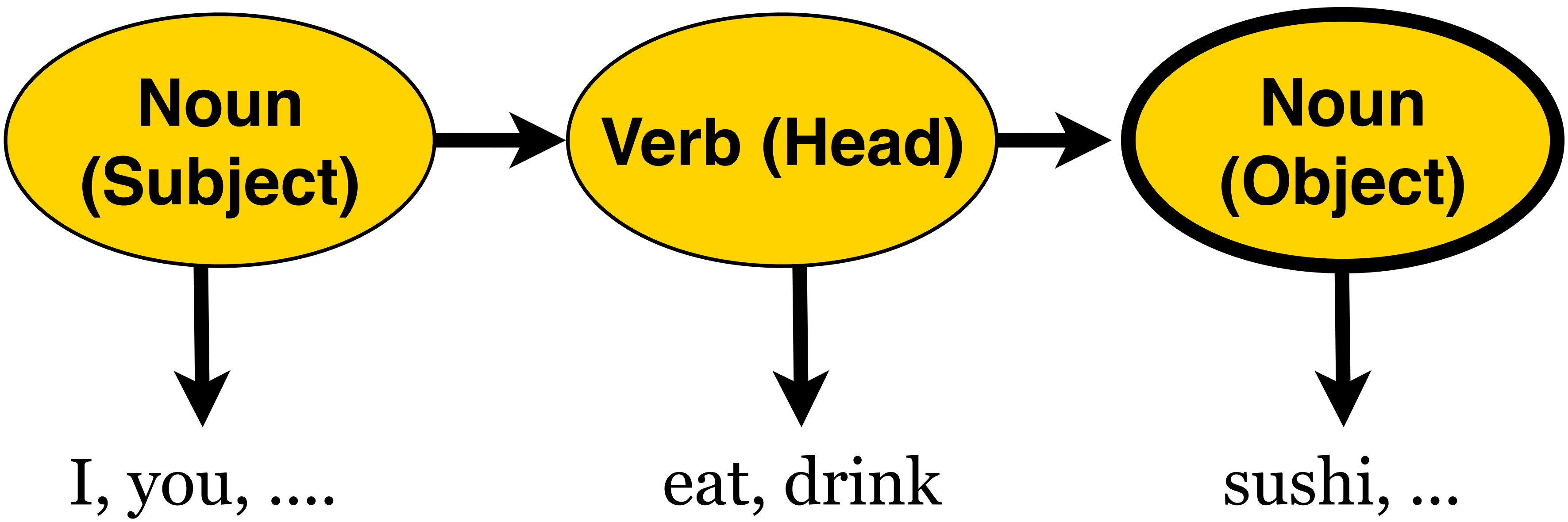
Recursion creates an infinite number of sentences
(even with a finite vocabulary),
but we need our program to be of finite size

Basic sentence structure



A finite-state-automaton (FSA)

A Hidden Markov Model (HMM)



Words take arguments

I eat sushi. ✔

I eat sushi you. ???

I sleep sushi ???

I give sushi ???

Subcategorization Violations

I drink sushi ?

Selectional Preference Violation

Subcategorization

(purely syntactic: what set of arguments do words take?)

Intransitive verbs (sleep) take only a subject.

Transitive verbs (eat) take a subject and one (direct) object.

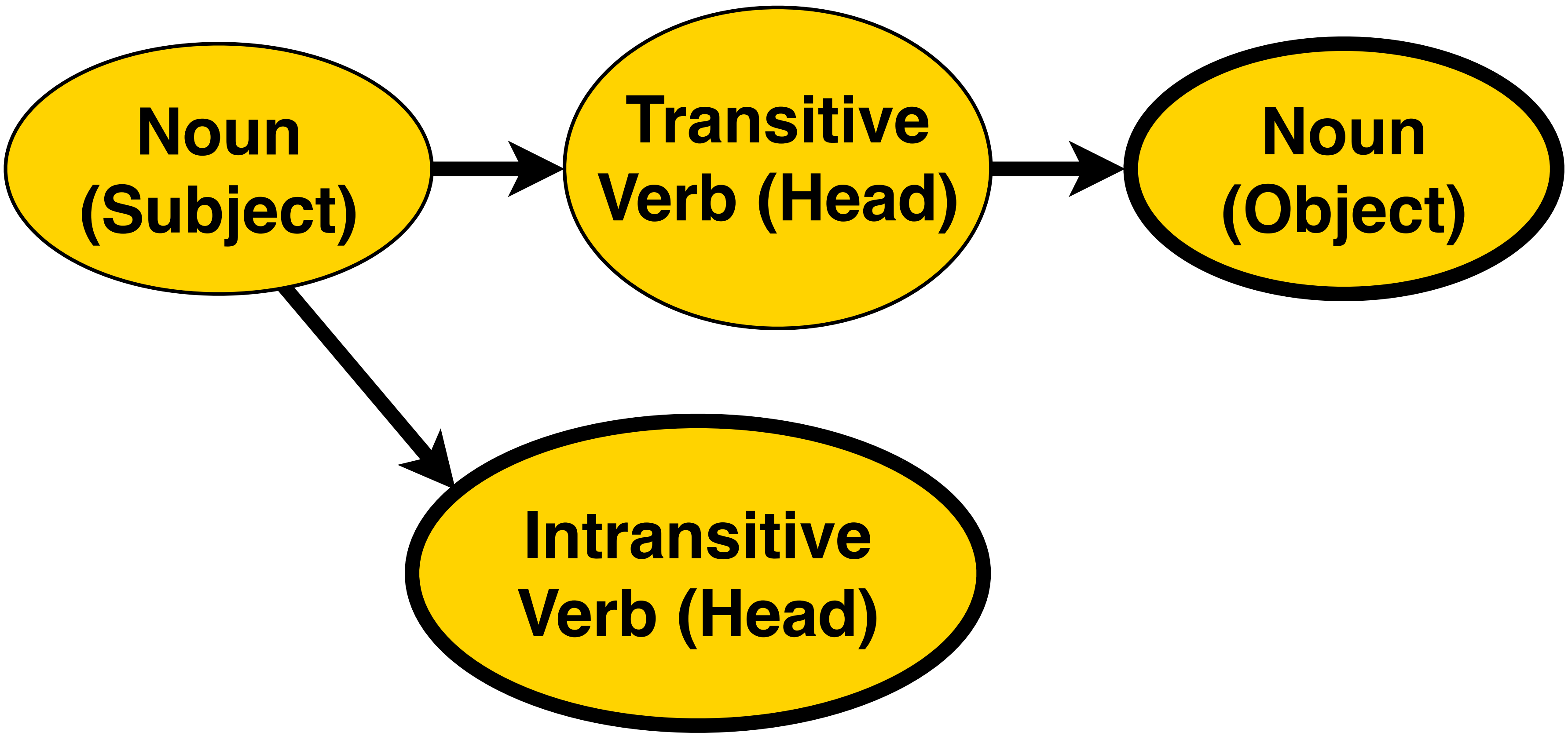
Ditransitive verbs (give) take a subject, direct object and indirect object.

Selectional preferences

(semantic: what types of arguments do words tend to take)

The object of eat should be edible.

A better FSA


Noun (Subject)
Transitive Verb (Head)
Noun (Object)
Intransitive Verb (Head)

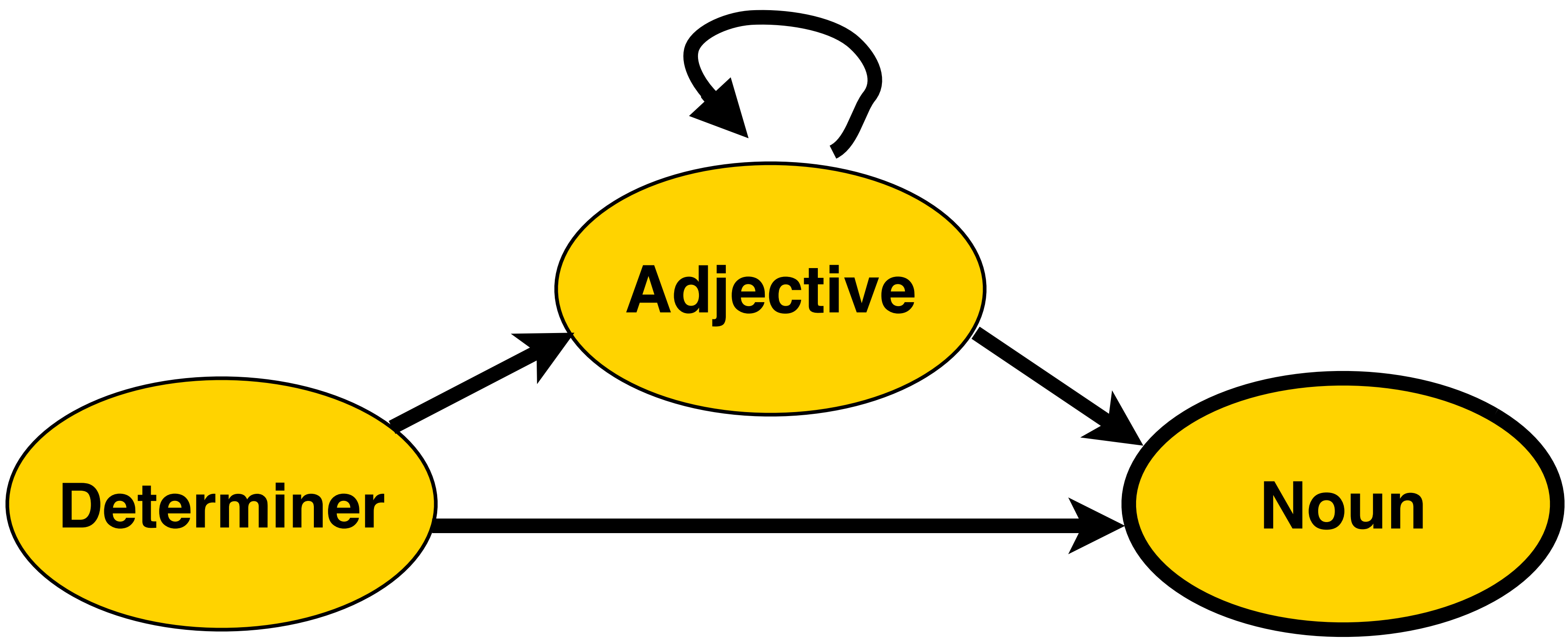
Language is recursive

the ball
the big ball
the big, red ball
the big, red, heavy ball
....

Adjectives can **modify** nouns.

The **number of modifiers (aka adjuncts)** a word can have is (in theory) **unlimited**.

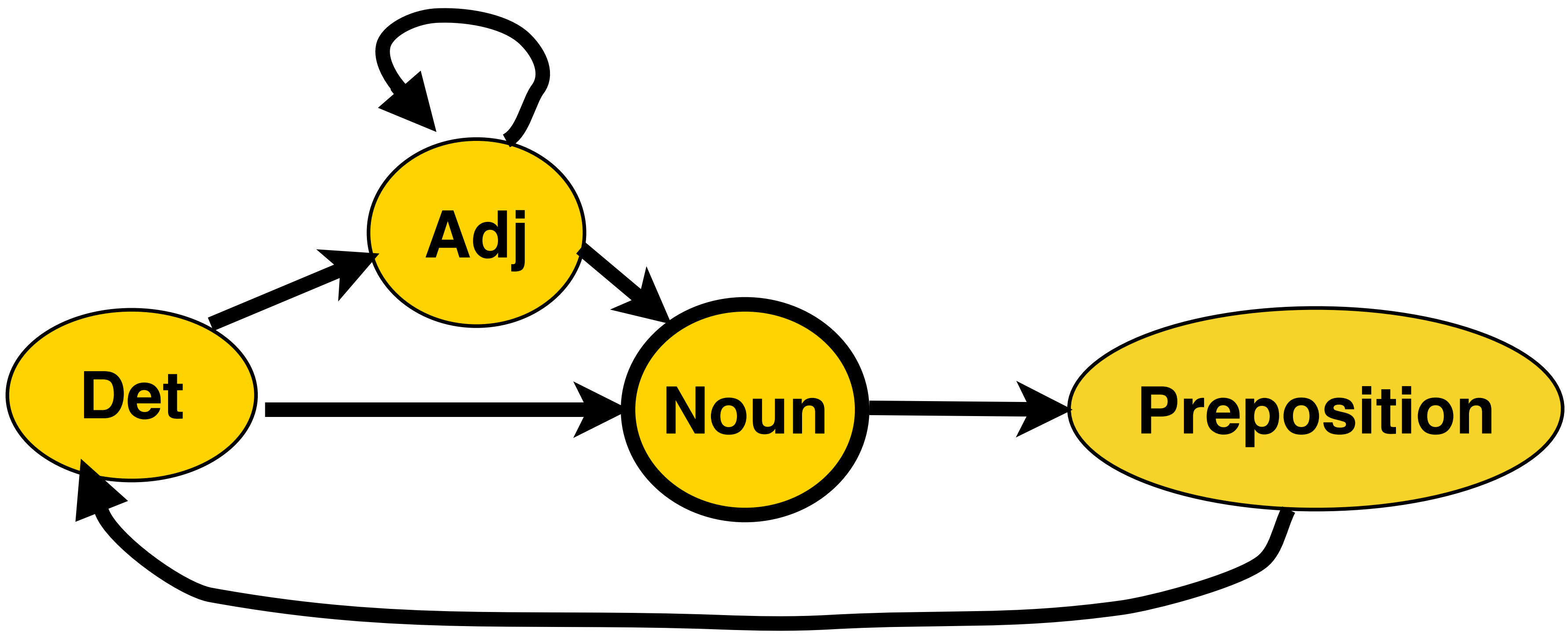
Another FSA



Recursion can be more complex

the ball
the ball in the garden
the ball in the garden behind the house
the ball in the garden behind the house next to the school
....

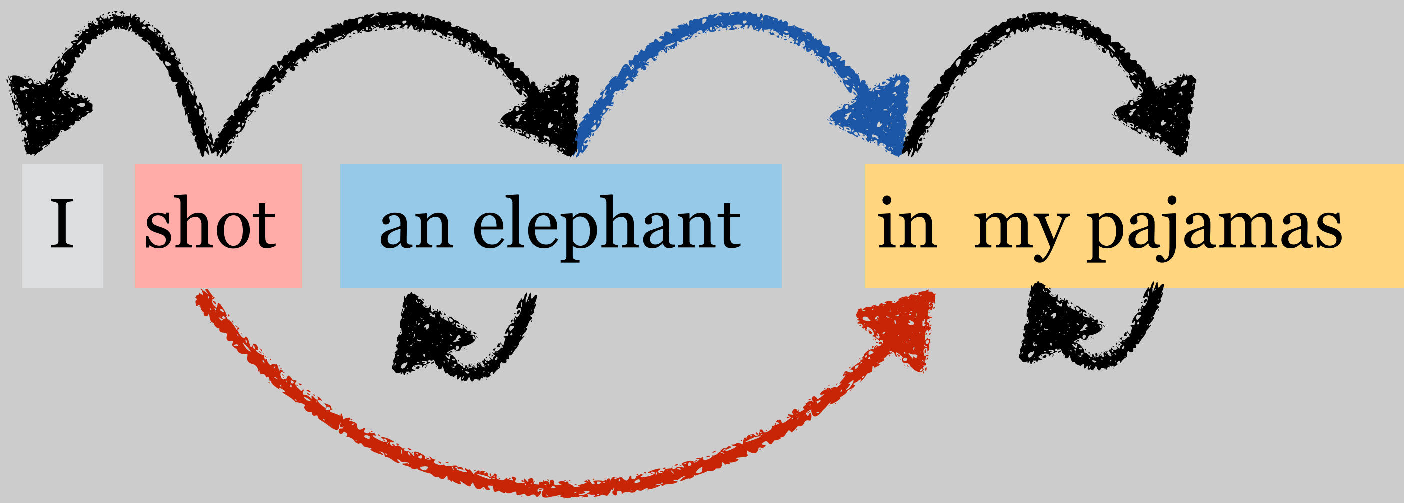
Yet another FSA



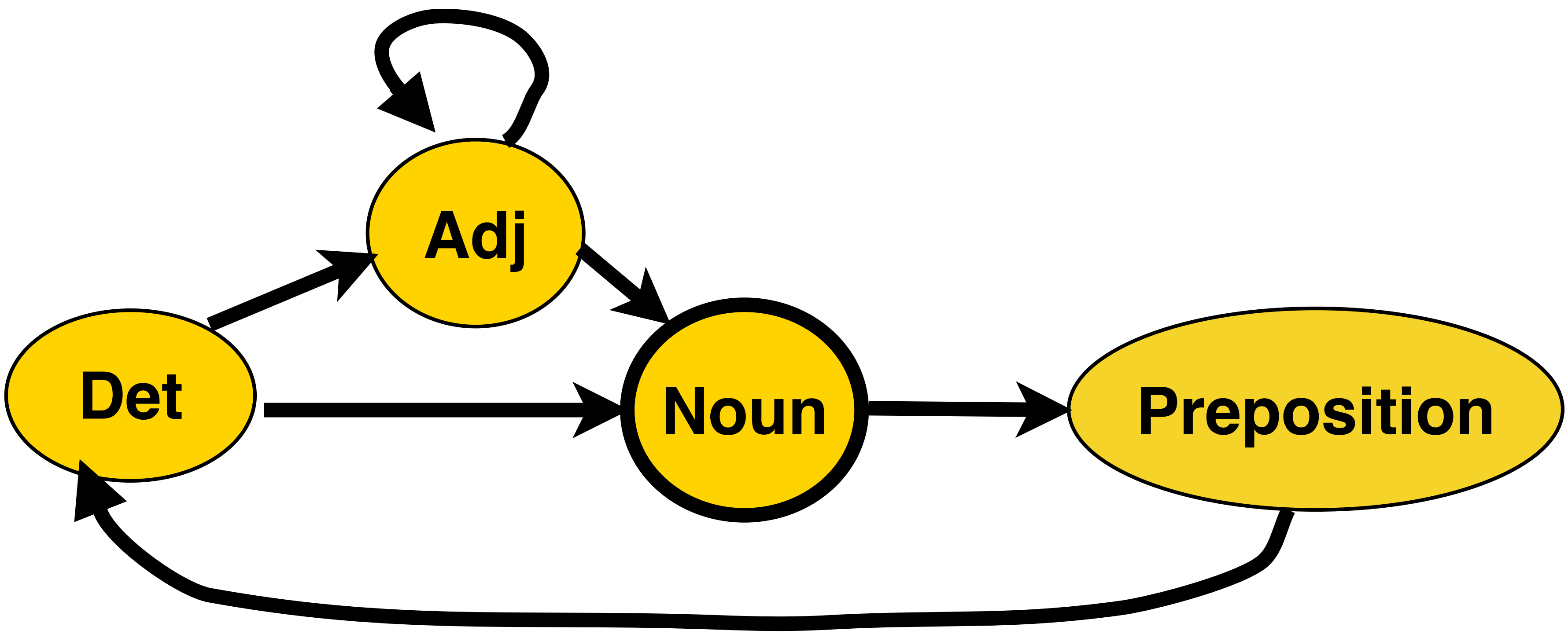
So, why do we need anything
beyond regular (finite-state) grammars?

What does this sentence mean?

There is an **attachment ambiguity**:
Does “in my pajamas” go with “shot”
or with “an elephant” ?



FSAs do not generate hierarchical structure



What is the structure of a sentence?

Sentence structure is **hierarchical**:

A sentence consists of **words** (I, eat, sushi, with, tuna)
…which form phrases or **constituents**: “sushi with tuna”

Sentence structure defines **dependencies** between words or phrases:

[*I* [*eat* [*sushi* [*with tuna*]]]]

Context-Free Grammars for natural language

Formal definitions

Context-free grammars

A CFG is a 4-tuple $\langle \mathbf{N}, \mathbf{\Sigma}, \mathbf{R}, S \rangle$ consisting of:

A finite set of **nonterminals** $\mathbf{N}$
(e.g. $\mathbf{N}$ = {S, NP, VP, PP, Noun, Verb, ...})

A finite set of **terminals** $\mathbf{\Sigma}$
(e.g. $\mathbf{\Sigma}$ = {I, you, he, eat, drink, sushi, ball, })

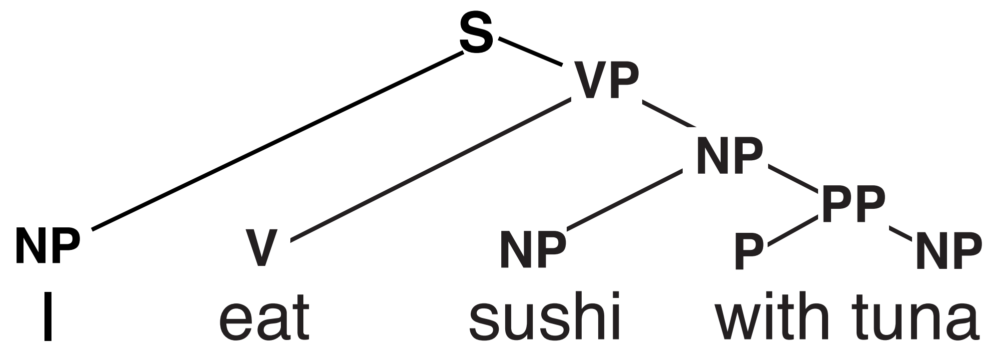
A finite set of **rules** $\mathbf{R}$
$\mathbf{R} \subseteq \{A \rightarrow \beta$ with left-hand-side (LHS) $A \in \mathbf{N}$
and right-hand-side (RHS) $\beta \in (\mathbf{N} \cup \mathbf{\Sigma})^* \}$

A unique **start symbol** $S \in \mathbf{N}$

Context-free grammars (CFGs) define **phrase structure trees**

```
NP  →  I
NP  →  sushi
NP  →  tuna
NP  →  NP PP
P   →  with
PP  →  P NP
S   →  NP VP
V   →  eat
VP  →  V  NP
```

NP: Noun Phrase
P: Preposition
S: Sentence
PP: Prepositional Phrase
V: Verb
VP: Verb Phrase



Leaf nodes (I, eat, …) correspond to the words in the sentence

Intermediate nodes (NP, VP, PP) span substrings (= the *yield* of the node), and correspond to nonterminal *constituents*

The root spans the entire sentence and is labeled with the start symbol of the grammar (here, S)

CFGs capture recursion

Language has simple and complex constituents

(simple: "the garden", complex: "the garden behind the house")

Complex constituents behave just like simple ones.

("behind the house" can always be omitted)

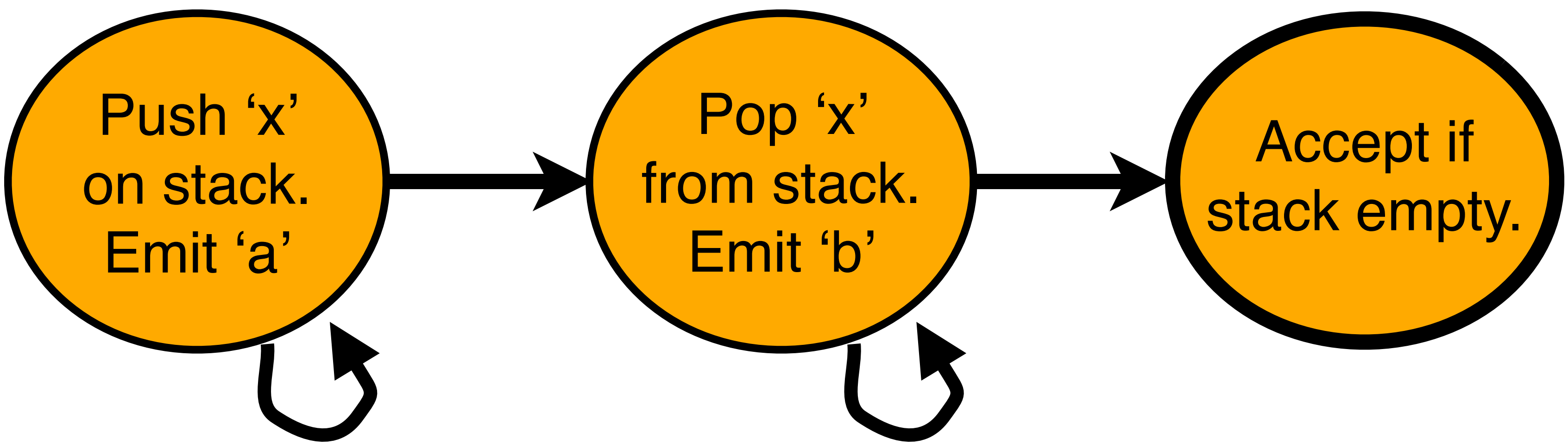
CFGs define **nonterminal categories** (e.g. NP) to capture **equivalence classes of constituents.**

Recursive rules (where the same nonterminal appears on both sides) generate recursive structures

NP	→	DT	N	(**Simple**, i.e. **non-recursive** NP)
NP	→	**NP**	PP	(**Complex**, i.e. **recursive**, NP)

CFGs are equivalent to Pushdown Automata (PDAs)

PDAs are FSAs with an additional stack:
Emit a symbol and push/pop a symbol from the stack



This is equivalent to the following CFG:

```
S  → a S b
S  → a b
```

Generating $a^n b^n$

Action	**Stack**	**String**
1. Push x on stack. Emit a.	x	a
2. Push x on stack. Emit a.	xx	aa
3. Push x on stack. Emit a.	xxx	aaa
4. Push x on stack. Emit a.	xxxx	aaaa
5. Pop x off stack. Emit b.	xxx	aaaab
6. Pop x off stack. Emit b.	xx	aaaabb
7. Pop x off stack. Emit b.	x	aaaabbb
8. Pop x off stack. Emit b		aaaabbbb

Encoding linguistic principles in a CFG

Is string α a constituent?

[Should my grammar/parse tree have a nonterminal for α?]

He talks [in class].

Substitution test:

Can α be replaced by a single word?

He talks [there].

Movement test:

Can α be moved around in the sentence?

[In class], he talks.

Answer test:

Can α be the answer to a question?

Where does he talk? - [In class].

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

> NB: this is an oversimplification. Some phrases (**John, Kim and Mary**) have multiple heads, others (I like coffee and [**you tea]**) perhaps don't even have a head

Every phrase has one **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**

> NB: some linguists think the argument-adjunct distinction isn't always clear-cut, and there are some cases that could be treated as either, or something in-between

Arguments are obligatory

Words **subcategorize** for specific sets of arguments:

Transitive verbs (sbj + obj): [John] likes [Mary]

The set/list of arguments is called a **subcat frame**

All **arguments** have to be **present**:

*[John] likes. *likes [Mary].

No argument slot can be **occupied multiple times**:

*[John] [Peter] likes [Ann] [Mary].

Words can have **multiple subcat frames**:

Transitive eat (sbj + obj): [John] eats [sushi].

Intransitive eat (sbj): [John] eats

Adjuncts (modifiers) are optional

Adverbs, PPs and adjectives can be adjuncts

Adverbs: John runs [fast].
a [very] heavy book.

PPs: John runs [in the gym].
the book [on the table]

Adjectives: a [heavy] book

There can be an arbitrary number of adjuncts:

John saw Mary.

John saw Mary [yesterday].

John saw Mary [yesterday] [in town]

John saw Mary [yesterday] [in town] [during lunch]

[Perhaps] John saw Mary [yesterday] [in town] [during lunch]

Heads, Arguments and Adjuncts in CFGs

How do we define CFGs that…
… identify heads and
… distinguish between arguments and adjuncts?

We have to make additional assumptions about the rules that we allow.

- Important: these are not formal/mathematical constraints, but aim to capture linguistic principles
- A more fleshed out version of what we will describe here is known as "X-bar Theory" (Chomsky, 1970)

Phrase structure trees that conform to these assumptions can easily be translated to dependency trees

Heads, Arguments and Adjuncts in CFGs

To identify heads:

We assume that each RHS has one head child, e.g.

```
VP → Verb NP        (Verbs are heads of VPs)
NP → Det Noun       (Nouns are heads of NPs)
S  → NP VP          (VPs are heads of sentences)
```

Exception: This does not work well for coordination:

```
VP → VP conj VP
```

We need to define for each nonterminal in our grammar (S, NP, VP, …) which nonterminals (or terminals) can be used as its head children.

Heads, Arguments and Adjuncts in CFGs

To distinguish between arguments and adjuncts, assume that each is introduced by different rules.

Argument rules:

The head has a different category from the parent:

```
S  → NP VP        (the NP is an argument of the VP [verb])
VP → Verb NP      (the NP is an argument of the verb)
```

This captures that arguments are obligatory.

Adjunct rules ("Chomsky adjunction"):

The head has the same category as the parent:

```
VP → VP PP        (the PP is an adjunct of the VP)
```

This captures that adjuncts are optional and that their number is unrestricted.

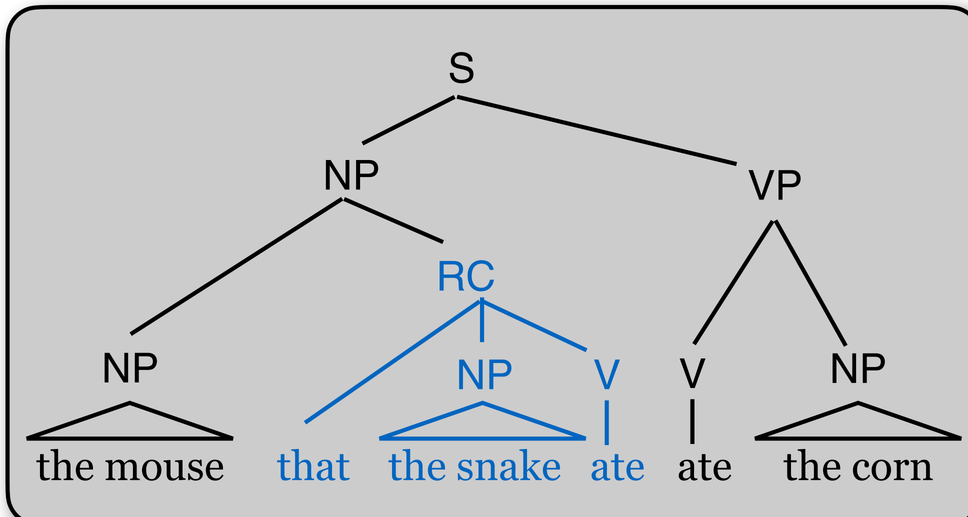
CFGs and unbounded recursion

Unbounded recursion: CFGs and center embedding

The mouse ate the corn.

The mouse that the snake ate ate the corn.

```
S    ⟶ NP  VP
VP   ⟶ V   NP
NP   ⟶ Det N
NP   ⟶ NP  RC
RC   ⟶ that NP V
Det  ⟶ the
N    ⟶ mouse|corn|snake
V    ⟶ ate
```



Unbounded recursion: CFGs and center embedding

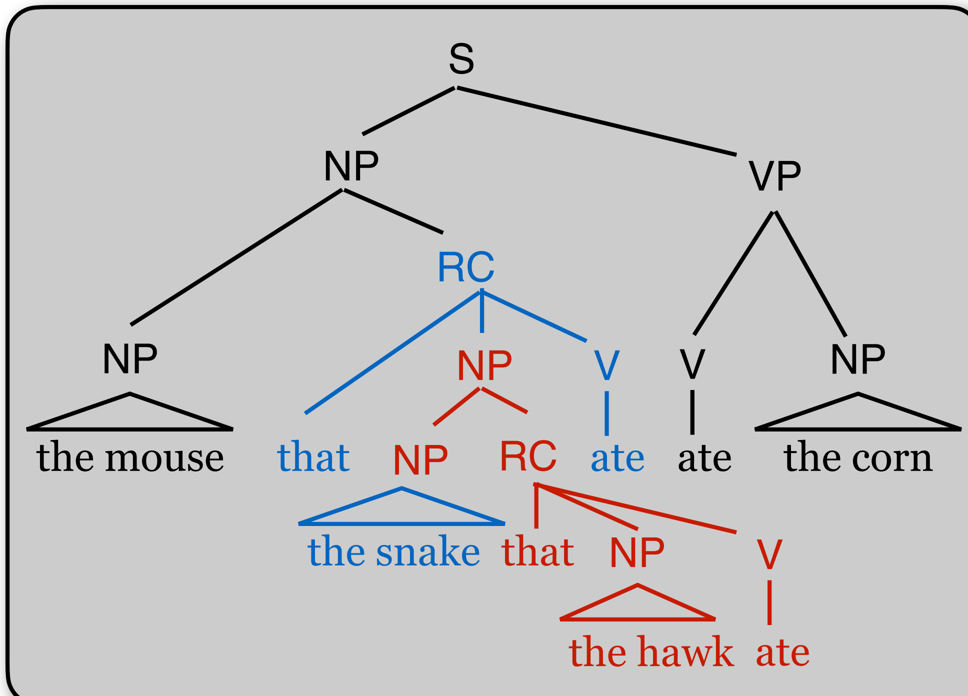
The mouse ate the corn.

The mouse that the snake ate ate the corn.

The mouse that the snake that the hawk ate ate ate the corn.

…

```
S    → NP   VP
VP   → V    NP
NP   → Det  N
NP   → NP   RC
RC   → that NP V
Det  → the
N    → mouse | corn | snake
V    → ate
```



Unbounded recursion: CFGs and center embedding

These sentences are unacceptable, but formally, they are all grammatical, because they are generated by the recursive rules required for even just one relative clause:

```
NP  ⟶  NP  RC
RC  ⟶  that NP V
```

Problem: CFGs are not able to capture **bounded recursion**. (bounded = “only embed one or two relative clauses”).

To deal with this discrepancy between what the grammar predicts to be grammatical, and what humans consider grammatical, linguists distinguish between a speaker’s **competence** (grammatical knowledge) and **performance** (processing and memory limitations)

The CKY parsing algorithm

CKY chart parsing algorithm

CKY= Cocke-Kasami-Younger (aka “CYK” 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)

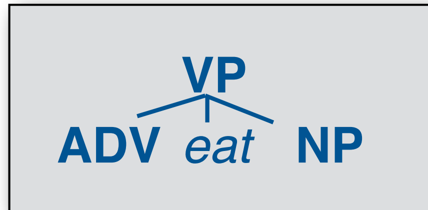
Presumes a CFG in **Chomsky Normal Form**:

Rules are all either **X → Y Z** (the RHS has two nonterminals)
or **X → w** (the RHS is a single terminal)
(with **X, Y, Z** nonterminals and **w** a terminal)

Chomsky Normal Form

The right-hand side of a standard CFG rules 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

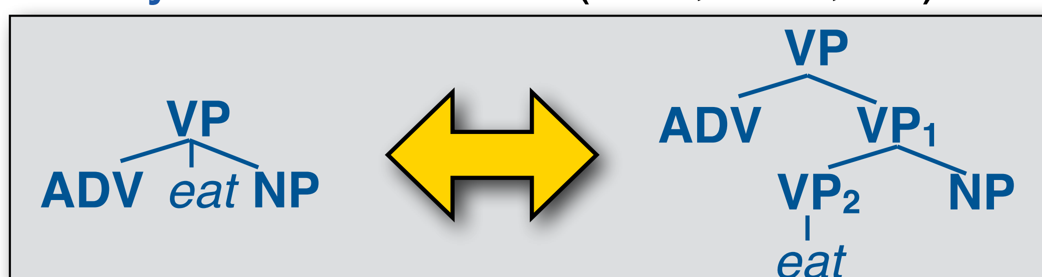
– **One terminal:** VP → eat

Any CFG can be **transformed** into an equivalent CFG in CNF by introducing new, *rule-specific* dummy non-terminals (VP_1, VP_2, …)

VP → ADVP **VP_1**

VP_1 → **VP_2** NP

VP_2 → eat



A note about ε-productions

Formally, context-free grammars are allowed to have **empty productions** (ε = the empty string):

VP → V NP   NP → DT Noun   NP → ε

These can always be **eliminated** without changing the language generated by the grammar:

VP → V NP   NP → DT Noun   NP → ε

becomes

VP → V NP   VP → V ε   NP → DT Noun

which in turn becomes

VP → V NP   VP → V   NP → DT Noun

We will assume that our grammars don't have ε-productions

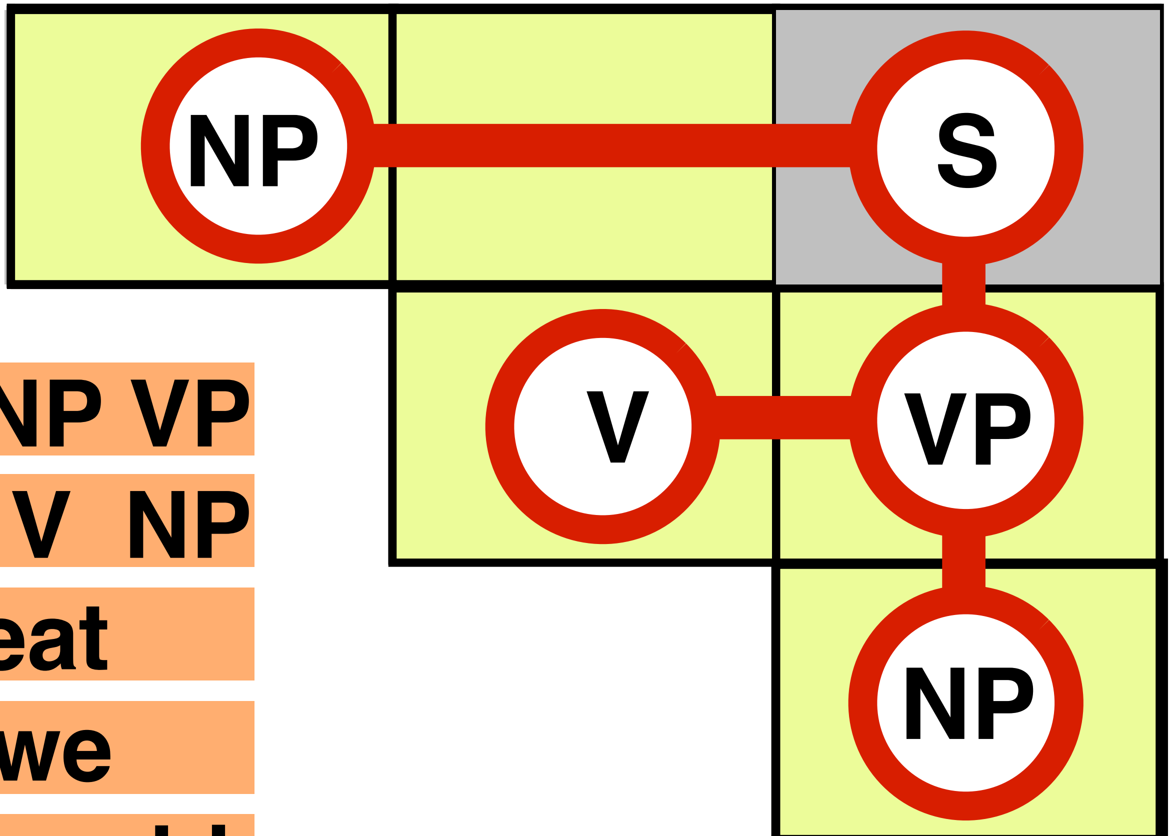
The CKY parsing algorithm

we	we eat	we eat sushi
	eat	eat sushi
		sushi

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

We eat sushi

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:

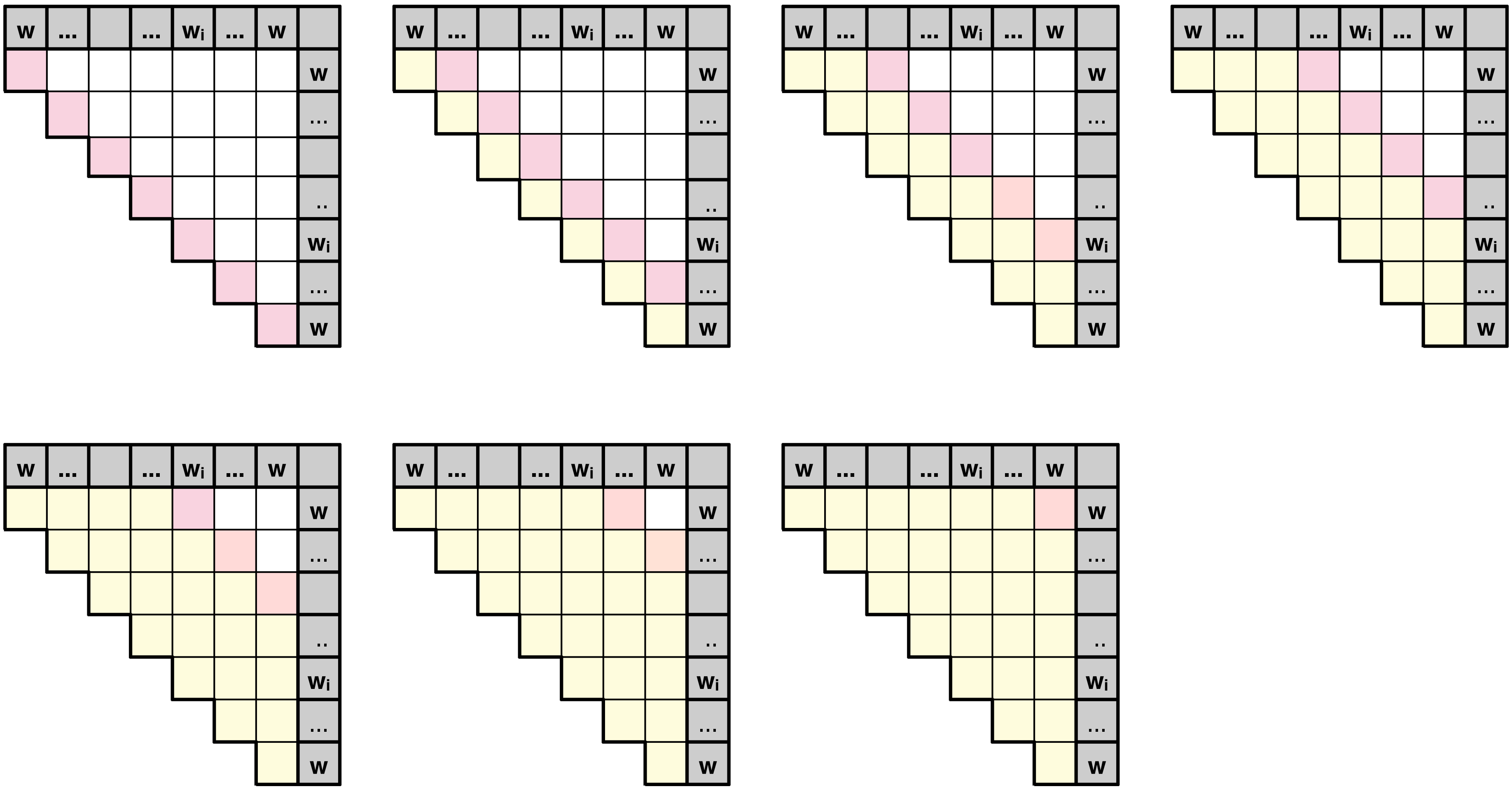
Fill in all cells $\mathrm{chart}[i][i+1]$, then $\mathrm{chart}[i][i+2]$, …,
until you reach $\mathrm{chart}[1][n]$ (the top right corner of the chart)

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

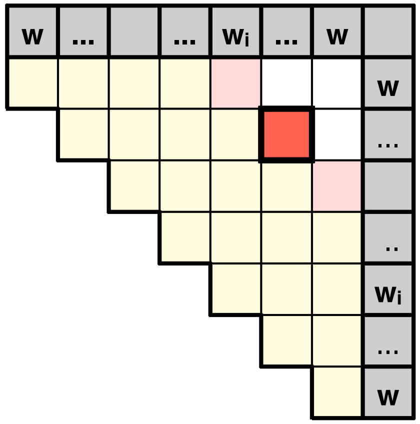
– If the grammar has a rule $X \rightarrow YZ$, $\mathrm{chart}[i][k]$ contains a Y and $\mathrm{chart}[k+1][j]$ contains a Z, add an X to $\mathrm{chart}[i][j]$ with two backpointers to the Y in $\mathrm{chart}[i][k]$ and the Z in $\mathrm{chart}[k+1][j]$

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

CKY: filling the chart



CKY: filling one cell

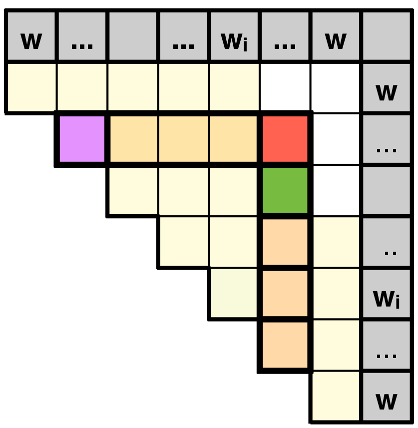


chart[2][6]:

w_1 **w_2 w_3 w_4 w_5 w_6** w_7

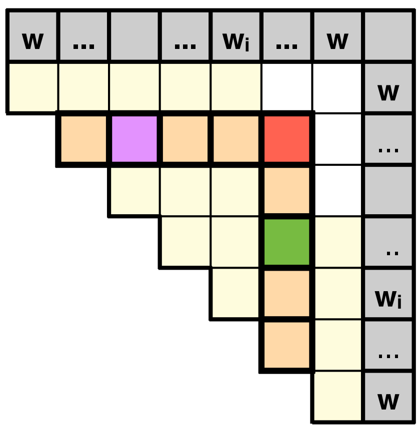
chart[2][6]:

w_1 $w_2w_3w_4w_5w_6$ w_7



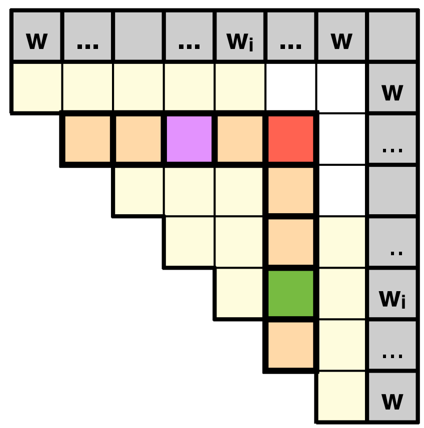
chart[2][6]:

w_1 $w_2w_3w_4w_5w_6$ w_7



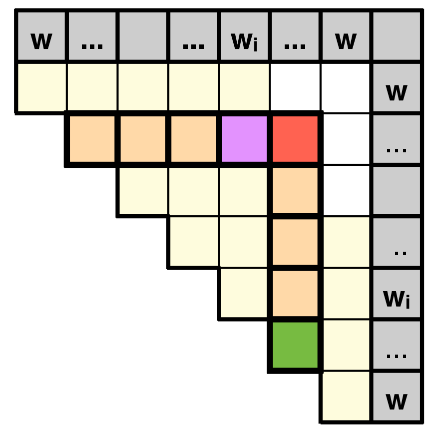
chart[2][6]:

w_1 $w_2w_3w_4w_5w_6$ w_7



chart[2][6]:

w_1 $w_2w_3w_4w_5w_6$ w_7



The CKY parsing algorithm

S → NP VP
VP → V NP
VP → VP PP
V → buy
VP → drinks
NP → NP PP
NP → we
NP → drinks
NP → milk
PP → P NP
P → with

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

Each cell may have one entry for each nonterminal

We buy drinks with milk

The CKY parsing algorithm

we	we eat	we eat sushi	we eat sushi with	we eat sushi with tuna
	V, VP eat	VP eat sushi	eat sushi with	VP eat sushi with tuna
			…th	NP sushi with tuna
				PP with tuna
				tuna

S → NP VP
VP → V NP
VP → VP PP
V → eat
VP → eat
NP → NP PP
NP → we
NP → sushi
NP → tuna
PP → P NP
P → with

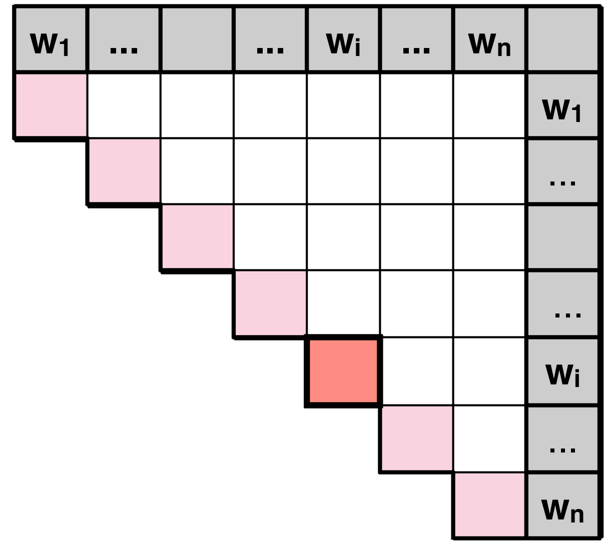
Each cell contains only a **single entry** for each nonterminal.
Each entry may have a **list** of pairs of backpointers.

We eat sushi with tuna

Cocke Kasami Younger

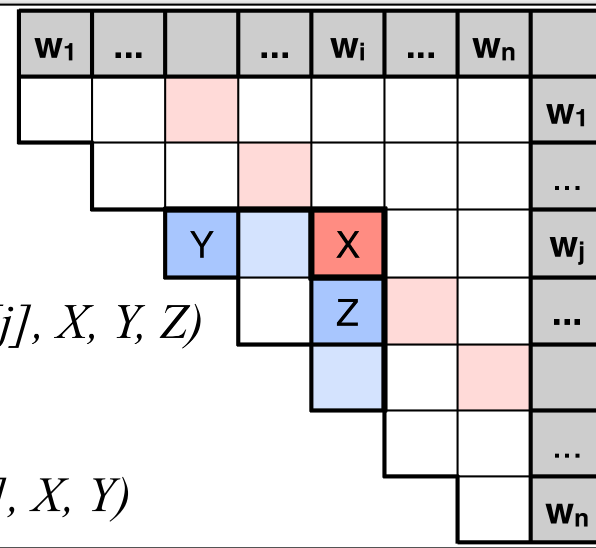
```
ckyParse(n):
    initChart(n)
    fillChart(n)
```

```
initChart(n):
  for i = 1...n:
     initCell(i,i)

initCell(i,i):
  for c in lex(word[i]):
      addToCell(cell[i][i], c)
```




```
fillChart(n):
  for span = 1...n-1:
      for i = 1...n-span:
          fillCell(i,i+span)

fillCell(i,j):
  for k = i..j-1:
     combineCells(i, k, j)
```

```
combineCells(i,k,j):
  for Y in cell[i][k]:
    for Z in cell[k +1][j]:
      for X in Nonterminals:
        if X →Y Z in Rules:
           addToCell(cell[i][j], X, Y, Z)
      for X in Nonterminals:
        if X →Y in Rules:
          addToCell(cell[i][j], X, Y)
```



Cocke Kasami Younger

addToCell(Terminal,cell) // Adding terminal nodes to the chart
cell.addEntry(Terminal) // add entry with no backpointers

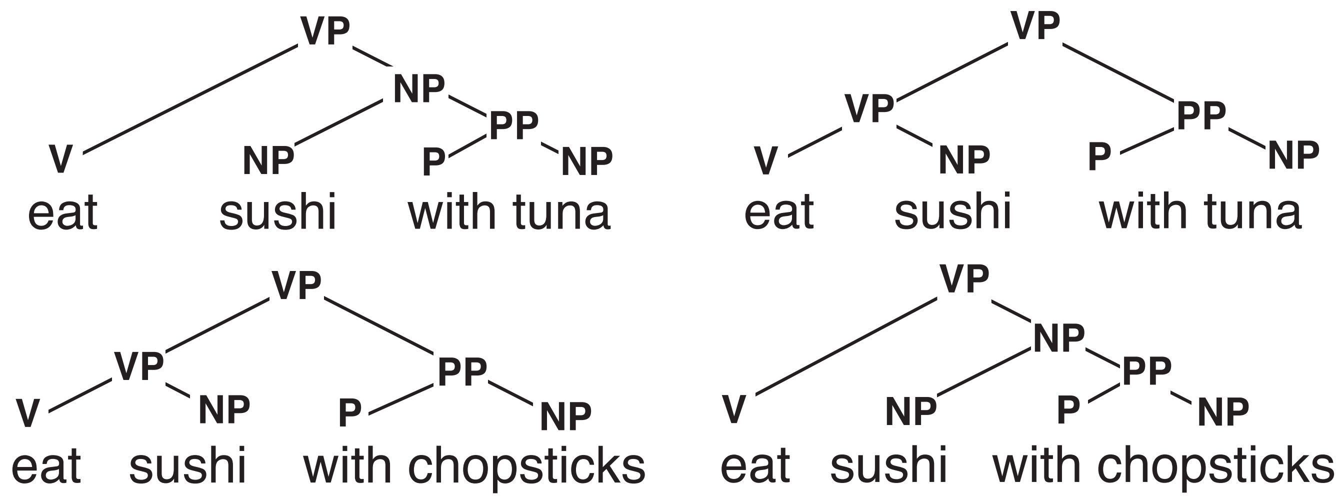
addToCell(Parent,cell,Left, Right) // For binary rules
if (cell.hasEntry(Parent)):
P = cell.getEntry(Parent)
P.addBackpointers(Left, Right) // add two backpointers to existing entry
else cell.addEntry(Parent, Left, Right) // add entry with a pair of backpointers

addToCell(Parent,cell,Child) // For unary rules
if (cell.hasEntry(Parent)):
P = cell.getEntry(Parent)
P.addBackpointer(Child) // add one backpointer to existing entry
else cell.addEntry(Parent, Child) // add entry with one backpointer

Probabilistic Context-Free Grammars (PCFGs)

Grammars are ambiguous

A grammar might generate multiple trees for a sentence:



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

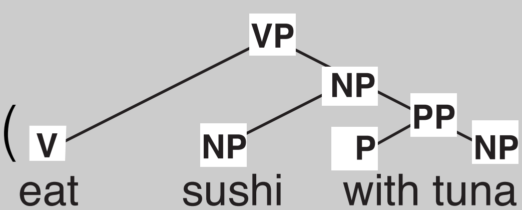
We need a model of $P(\tau \mid S)$

Computing P(τ | S)

Using Bayes' Rule:

$$
\begin{aligned}
\arg\max_{\tau} P(\tau|S) &= \arg\max_{\tau} \frac{P(\tau, S)}{P(S)} \\
&= arg\max_{\tau} P(\tau, S) \\
&= arg\max_{\tau} P(\tau) \quad \text{if} \quad \mathrm{S} = \mathrm{yield}(\tau)
\end{aligned}
$$

The **yield of a tree** is the string of terminal symbols that can be read off the leaf nodes

yield(

) = *eat sushi with tuna*

Computing P(τ)

T is the (infinite) set of all trees in the language:

$$L = \{s \in \Sigma^* | \; \exists \tau \in T : \text{yield}(\tau) = s\}$$

We need to define P(τ) such that:

$$\forall \tau \in T : \quad 0 \leq P(\tau) \leq 1$$

$$\sum_{\tau \in T} P(\tau) = 1$$

The set T is generated by a context-free grammar

S	$\longrightarrow$	NP VP	VP	$\longrightarrow$	Verb NP	NP	$\longrightarrow$	Det Noun
S	$\longrightarrow$	S conj S	VP	$\longrightarrow$	VP PP	NP	$\longrightarrow$	NP PP
S	$\longrightarrow$	.....	VP	$\longrightarrow$	.....	NP	$\longrightarrow$	.....

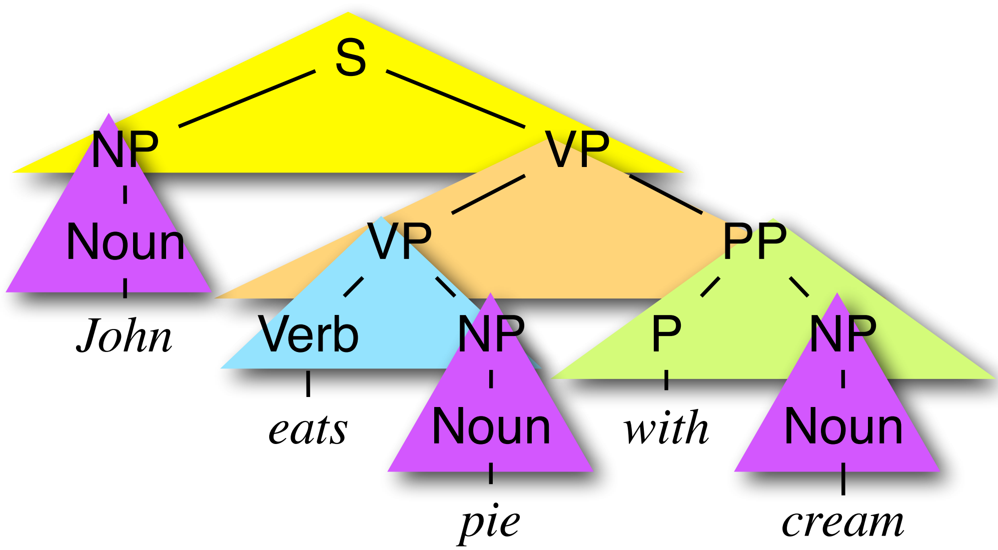
Probabilistic Context-Free Grammars

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

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

Computing $P(\tau)$ with a PCFG

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



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

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

$$= \mathbf{0.00384}$$

Learning the parameters of a PCFG

If we have a treebank (a corpus in which each sentence is associated with a parse tree), we can just **count** the number of times each rule appears, e.g.:

```
S → NP VP .      (count = 1000)
S → S conj S .   (count = 220)
PP → IN NP       (count = 700)
```

and then we divide the **count** (observed frequency) **of each rule** X → Y Z by the **sum of the frequencies of all rules with the same LHS** X to turn these counts into probabilities:

```
S → NP VP .      (p = 1000/1220)
S → S conj S .   (p = 220/1220)
PP → IN NP       (p = 700/700)
```

PCFG Decoding: CKY with Viterbi

How do we handle flat rules?

S	$\longrightarrow$	NP VP	0.8
S	$\longrightarrow$	**S conj S**	**0.2**
NP	$\longrightarrow$	Noun	0.2
NP	$\longrightarrow$	Det Noun	0.4
NP	$\longrightarrow$	NP PP	0.2
NP	$\longrightarrow$	**NP conj NP**	**0.2**
VP	$\longrightarrow$	Verb	0.3
VP	$\longrightarrow$	Verb NP	0.3
VP	$\longrightarrow$	**Verb NP NP**	**0.1**
VP	$\longrightarrow$	VP PP	0.3
PP	$\longrightarrow$	PP NP	1.0
Prep	$\longrightarrow$	P	1.0
Noun	$\longrightarrow$	N	1.0
Verb	$\longrightarrow$	V	1.0

S	$\longrightarrow$	**S ConjS**	**0.2**
ConjS	$\longrightarrow$	**conj S**	**1.0**

Binarize each flat rule by adding **a unique dummy nonterminal** (ConjS), and **setting the probability** of the new rule with the dummy nonterminal on the LHS **to 1**

How do we handle flat 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	0.3
VP	⟶	Verb NP	0.3
VP	**⟶**	**Verb NP NP**	**0.1**
VP	⟶	VP PP	0.3
PP	⟶	PP NP	1.0
Prep	⟶	P	1.0
Noun	⟶	N	1.0
Verb	⟶	V	1.0

S	⟶	NP VP	0.8
S	**⟶**	**S ConjS**	**0.2**
NP	⟶	Noun	0.2
NP	⟶	Det Noun	0.4
NP	⟶	NP PP	0.2
NP	**⟶**	**NP ConjNP**	**0.2**
VP	⟶	Verb	0.3
VP	⟶	Verb NP	0.3
VP	**⟶**	**Verb NPNP**	**0.1**
VP	⟶	VP PP	0.3
PP	⟶	PP NP	1.0
Prep	⟶	P	1.0
Noun	⟶	N	1.0
Verb	⟶	V	1.0
ConjS	**⟶**	**conj S**	**1.0**
ConjNP	**⟶**	**conj NP**	**1.0**
NPNP	**⟶**	**NP NP**	**1.0**

Probabilistic CKY: Viterbi

Like standard CKY, but with probabilities.

Finding the most likely tree is similar to Viterbi for HMMs:

Initialization:

- [*optional*] Every chart entry that corresponds to a **terminal** (entry w in `cell[i][i]`) has a Viterbi probability $P_{\text{VIT}}(\text{w}_{[i][i]}) = 1$ (*)
- Every entry for a **non-terminal** `X` in `cell[i][i]` has Viterbi probability $P_{\text{VIT}}(\text{X}_{[i][i]}) = \text{P}(\text{X} \rightarrow \text{w} \mid \text{X})$ [and a single backpointer to $\text{w}_{[i][i]}$ (*)]

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]`):

$$P_{\text{VIT}}(\text{X}_{[i][j]}) = \text{argmax}_{\text{Y,Z,k}}\, P_{\text{VIT}}(\text{Y}_{[i][k]}) \times P_{\text{VIT}}(\text{Z}_{[k+1][j]}) \times P(\text{X} \rightarrow \text{Y Z} \mid \text{X})$$

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

*this is unnecessary for simple PCFGs, but can be helpful for more complex probability models

Probabilistic CKY

Input: POS-tagged sentence

`John_N eats_V pie_N with_P cream_N`

John	eats	pie	with	cream	
					John
					eats
					pie
					with
					cream

S	$\longrightarrow$	NP VP	0.8
S	$\longrightarrow$	S ConjS	0.2
NP	$\longrightarrow$	Noun	0.2
NP	$\longrightarrow$	Det Noun	0.4
NP	$\longrightarrow$	NP PP	0.2
NP	$\longrightarrow$	NP ConjNP	0.2
VP	$\longrightarrow$	Verb	0.3
VP	$\longrightarrow$	Verb NP	0.3
VP	$\longrightarrow$	Verb NPNP	0.1
VP	$\longrightarrow$	VP PP	0.3
PP	$\longrightarrow$	PP NP	1.0
Prep	$\longrightarrow$	P	1.0
Noun	$\longrightarrow$	N	1.0
Verb	$\longrightarrow$	V	1.0
ConjS	$\longrightarrow$	conj S	1.0
ConjNP	$\longrightarrow$	conj NP	1.0
NPNP	$\longrightarrow$	NP NP	1.0

Probabilistic CKY

Input: POS-tagged sentence

`John_N eats_V pie_N with_P cream_N`

John	eats	pie	with	cream	
Noun NP 1.0 0.2					**John**
					eats
					pie
					with
					cream

S	⟶	NP VP	0.8
S	⟶	S ConjS	0.2
NP	⟶	Noun	0.2
NP	⟶	Det Noun	0.4
NP	⟶	NP PP	0.2
NP	⟶	NP ConjNP	0.2
VP	⟶	Verb	0.3
VP	⟶	Verb NP	0.3
VP	⟶	Verb NPNP	0.1
VP	⟶	VP PP	0.3
PP	⟶	PP NP	1.0
Prep	⟶	P	1.0
Noun	⟶	N	1.0
Verb	⟶	V	1.0
ConjS	⟶	conj S	1.0
ConjNP	⟶	conj NP	1.0
NPNP	⟶	NP NP	1.0

Probabilistic CKY

Input: POS-tagged sentence

`John_N eats_V pie_N with_P cream_N`

John	eats	pie	with	cream	
Noun 1.0 NP 0.2					**John**
	Verb 1.0 VP 0.3				**eats**
					pie
					with
					cream

S	$\longrightarrow$	NP VP	0.8
S	$\longrightarrow$	S ConjS	0.2
NP	$\longrightarrow$	Noun	0.2
NP	$\longrightarrow$	Det Noun	0.4
NP	$\longrightarrow$	NP PP	0.2
NP	$\longrightarrow$	NP ConjNP	0.2
VP	$\longrightarrow$	Verb	0.3
VP	$\longrightarrow$	Verb NP	0.3
VP	$\longrightarrow$	Verb NPNP	0.1
VP	$\longrightarrow$	VP PP	0.3
PP	$\longrightarrow$	PP NP	1.0
Prep	$\longrightarrow$	P	1.0
Noun	$\longrightarrow$	N	1.0
Verb	$\longrightarrow$	V	1.0
ConjS	$\longrightarrow$	conj S	1.0
ConjNP	$\longrightarrow$	conj NP	1.0
NPNP	$\longrightarrow$	NP NP	1.0

Probabilistic CKY

Input: POS-tagged sentence

`John_N eats_V pie_N with_P cream_N`

John	eats	pie	with	cream	
Noun NP 1.0 0.2					**John**
	Verb VP 1.0 0.3				**eats**
		Noun NP 1.0 0.2			**pie**
					with
					cream

S	⟶	NP VP	0.8
S	⟶	S ConjS	0.2
NP	⟶	Noun	0.2
NP	⟶	Det Noun	0.4
NP	⟶	NP PP	0.2
NP	⟶	NP ConjNP	0.2
VP	⟶	Verb	0.3
VP	⟶	Verb NP	0.3
VP	⟶	Verb NPNP	0.1
VP	⟶	VP PP	0.3
PP	⟶	PP NP	1.0
Prep	⟶	P	1.0
Noun	⟶	N	1.0
Verb	⟶	V	1.0
ConjS	⟶	conj S	1.0
ConjNP	⟶	conj NP	1.0
NPNP	⟶	NP NP	1.0

Probabilistic CKY

Input: POS-tagged sentence

`John_N eats_V pie_N with_P cream_N`

John	eats	pie	with	cream	
Noun 1.0 NP 0.2					**John**
	Verb 1.0 VP 0.3				**eats**
		Noun 1.0 NP 0.2			**pie**
			Prep 1.0		**with**
					cream

S	$\longrightarrow$	NP VP	0.8
S	$\longrightarrow$	S ConjS	0.2
NP	$\longrightarrow$	Noun	0.2
NP	$\longrightarrow$	Det Noun	0.4
NP	$\longrightarrow$	NP PP	0.2
NP	$\longrightarrow$	NP ConjNP	0.2
VP	$\longrightarrow$	Verb	0.3
VP	$\longrightarrow$	Verb NP	0.3
VP	$\longrightarrow$	Verb NPNP	0.1
VP	$\longrightarrow$	VP PP	0.3
PP	$\longrightarrow$	PP NP	1.0
Prep	$\longrightarrow$	P	1.0
Noun	$\longrightarrow$	N	1.0
Verb	$\longrightarrow$	V	1.0
ConjS	$\longrightarrow$	conj S	1.0
ConjNP	$\longrightarrow$	conj NP	1.0
NPNP	$\longrightarrow$	NP NP	1.0

Probabilistic CKY

Input: POS-tagged sentence

John_N eats_V pie_N with_P cream_N

John	eats	pie	with	cream	
Noun NP 1.0 0.2					**John**
	Verb VP 1.0 0.3				**eats**
		Noun NP 1.0 0.2			**pie**
			Prep 1.0		**with**
				Noun NP 1.0 0.2	**cream**

S	⟶	NP VP	0.8
S	⟶	S ConjS	0.2
NP	⟶	Noun	0.2
NP	⟶	Det Noun	0.4
NP	⟶	NP PP	0.2
NP	⟶	NP ConjNP	0.2
VP	⟶	Verb	0.3
VP	⟶	Verb NP	0.3
VP	⟶	Verb NPNP	0.1
VP	⟶	VP PP	0.3
PP	⟶	PP NP	1.0
Prep	⟶	P	1.0
Noun	⟶	N	1.0
Verb	⟶	V	1.0
ConjS	⟶	conj S	1.0
ConjNP	⟶	conj NP	1.0
NPNP	⟶	NP NP	1.0

Probabilistic CKY

Input: POS-tagged sentence

`John_N eats_V pie_N with_P cream_N`

John	eats	pie	with	cream	
Noun NP 1.0 **0.2**	**S** **0.8 · 0.2 · 0.3**				**John**
	Verb **VP** 1.0 **0.3**				**eats**
		Noun NP 1.0 0.2			**pie**
			Prep 1.0		**with**
				Noun NP 1.0 0.2	**cream**

S ⟶	NP VP	0.8
S ⟶	S ConjS	0.2
NP ⟶	Noun	0.2
NP ⟶	Det Noun	0.4
NP ⟶	NP PP	0.2
NP ⟶	NP ConjNP	0.2
VP ⟶	Verb	0.3
VP ⟶	Verb NP	0.3
VP ⟶	Verb NPNP	0.1
VP ⟶	VP PP	0.3
PP ⟶	PP NP	1.0
Prep ⟶	P	1.0
Noun ⟶	N	1.0
Verb ⟶	V	1.0
ConjS ⟶	conj S	1.0
ConjNP ⟶	conj NP	1.0
NPNP ⟶	NP NP	1.0

Probabilistic CKY

Input: POS-tagged sentence

`John_N eats_V pie_N with_P cream_N`

John	eats	pie	with	cream	
Noun NP 1.0 0.2	S $0.8 \cdot 0.2 \cdot 0.3$				**John**
	Verb VP **1.0** 0.3	**VP** $1 \cdot 0.3 \cdot 0.2$ $= 0.06$			**eats**
		Noun **NP** 1.0 **0.2**			**pie**
			Prep 1.0		**with**
				Noun NP 1.0 0.2	**cream**

S	$\longrightarrow$	NP VP	0.8
S	$\longrightarrow$	S ConjS	0.2
NP	$\longrightarrow$	Noun	0.2
NP	$\longrightarrow$	Det Noun	0.4
NP	$\longrightarrow$	NP PP	0.2
NP	$\longrightarrow$	NP ConjNP	0.2
VP	$\longrightarrow$	Verb	0.3
VP	$\longrightarrow$	Verb NP	0.3
VP	$\longrightarrow$	Verb NPNP	0.1
VP	$\longrightarrow$	VP PP	0.3
PP	$\longrightarrow$	PP NP	1.0
Prep	$\longrightarrow$	P	1.0
Noun	$\longrightarrow$	N	1.0
Verb	$\longrightarrow$	V	1.0
ConjS	$\longrightarrow$	conj S	1.0
ConjNP	$\longrightarrow$	conj NP	1.0
NPNP	$\longrightarrow$	NP NP	1.0

Probabilistic CKY

Input: POS-tagged sentence

`John_N eats_V pie_N with_P cream_N`

John	eats	pie	with	cream	
Noun NP 1.0 0.2	S $0.8 \cdot 0.2 \cdot 0.3$				**John**
	Verb VP 1.0 0.3	VP $1 \cdot 0.3 \cdot 0.2$ $= 0.06$			**eats**
		Noun NP 1.0 0.2			**pie**
			Prep 1.0		**with**
				Noun NP 1.0 0.2	**cream**

S	⟶	NP VP	0.8
S	⟶	S ConjS	0.2
NP	⟶	Noun	0.2
NP	⟶	Det Noun	0.4
NP	⟶	NP PP	0.2
NP	⟶	NP ConjNP	0.2
VP	⟶	Verb	0.3
VP	⟶	Verb NP	0.3
VP	⟶	Verb NPNP	0.1
VP	⟶	VP PP	0.3
PP	⟶	PP NP	1.0
Prep	⟶	P	1.0
Noun	⟶	N	1.0
Verb	⟶	V	1.0
ConjS	⟶	conj S	1.0
ConjNP	⟶	conj NP	1.0
NPNP	⟶	NP NP	1.0

Probabilistic CKY

Input: POS-tagged sentence

`John_N eats_V pie_N with_P cream_N`

<table>
<tr><th>John</th><th>eats</th><th>pie</th><th>with</th><th>cream</th><th></th></tr>
<tr><td>Noun NP
1.0 0.2</td><td>S
0.8 · 0.2 · 0.3</td><td></td><td></td><td></td><td>John</td></tr>
<tr><td></td><td>Verb VP
1.0 0.3</td><td>VP
1 · 0.3 · 0.2
= 0.06</td><td></td><td></td><td>eats</td></tr>
<tr><td></td><td></td><td>Noun NP
1.0 0.2</td><td></td><td></td><td>pie</td></tr>
<tr><td></td><td></td><td></td><td>Prep
1.0</td><td>PP
1 · 1 · 0.2</td><td>with</td></tr>
<tr><td></td><td></td><td></td><td></td><td>Noun NP
1.0 0.2</td><td>cream</td></tr>
</table>

S	⟶	NP VP	0.8
S	⟶	S ConjS	0.2
NP	⟶	Noun	0.2
NP	⟶	Det Noun	0.4
NP	⟶	NP PP	0.2
NP	⟶	NP ConjNP	0.2
VP	⟶	Verb	0.3
VP	⟶	Verb NP	0.3
VP	⟶	Verb NPNP	0.1
VP	⟶	VP PP	0.3
PP	⟶	PP NP	1.0
Prep	⟶	P	1.0
Noun	⟶	N	1.0
Verb	⟶	V	1.0
ConjS	⟶	conj S	1.0
ConjNP	⟶	conj NP	1.0
NPNP	⟶	NP NP	1.0

Probabilistic CKY

Input: POS-tagged sentence

`John_N eats_V pie_N with_P cream_N`

John	eats	pie	with	cream	
Noun NP 1.0 **0.2**	S 0.8 · 0.2 · 0.3	**S** **0.8 · 0.2 · 0.06**			**John**
	Verb VP 1.0 0.3	**VP** **1 · 0.3 · 0.2** **= 0.06**			**eats**
		Noun NP 1.0 0.2			**pie**
			Prep 1.0	PP 1 · 1 · 0.2	**with**
				Noun NP 1.0 0.2	**cream**

S	⟶	NP VP	0.8
S	⟶	S ConjS	0.2
NP	⟶	Noun	0.2
NP	⟶	Det Noun	0.4
NP	⟶	NP PP	0.2
NP	⟶	NP ConjNP	0.2
VP	⟶	Verb	0.3
VP	⟶	Verb NP	0.3
VP	⟶	Verb NPNP	0.1
VP	⟶	VP PP	0.3
PP	⟶	PP NP	1.0
Prep	⟶	P	1.0
Noun	⟶	N	1.0
Verb	⟶	V	1.0
ConjS	⟶	conj S	1.0
ConjNP	⟶	conj NP	1.0
NPNP	⟶	NP NP	1.0

Probabilistic CKY

Input: POS-tagged sentence

`John_N eats_V pie_N with_P cream_N`

John	eats	pie	with	cream	
Noun NP 1.0 0.2	S 0.8 · 0.2 · 0.3	S 0.8 · 0.2 · 0.06			**John**
	Verb VP 1.0 0.3	VP 1 · 0.3 · 0.2 = 0.06			**eats**
		Noun NP 1.0 0.2			**pie**
			Prep 1.0	PP 1 · 1 · 0.2	**with**
				Noun NP 1.0 0.2	**cream**

```
S      ⟶ NP VP       0.8
S      ⟶ S ConjS     0.2
NP     ⟶ Noun        0.2
NP     ⟶ Det Noun    0.4
NP     ⟶ NP PP       0.2
NP     ⟶ NP ConjNP   0.2
VP     ⟶ Verb        0.3
VP     ⟶ Verb NP     0.3
VP     ⟶ Verb NPNP   0.1
VP     ⟶ VP PP       0.3
PP     ⟶ PP NP       1.0
Prep   ⟶ P           1.0
Noun   ⟶ N           1.0
Verb   ⟶ V           1.0
ConjS  ⟶ conj S      1.0
ConjNP ⟶ conj NP     1.0
NPNP   ⟶ NP NP       1.0
```

Probabilistic CKY

Input: POS-tagged sentence

`John_N eats_V pie_N with_P cream_N`

John	eats	pie	with	cream	
Noun NP 1.0 0.2	S 0.8 · 0.2 · 0.3	S 0.8 · 0.2 · 0.06			**John**
	Verb VP 1.0 0.3	VP 1 · 0.3 · 0.2 = 0.06			**eats**
		Noun NP 1.0 0.2			**pie**
			Prep 1.0	PP 1 · 1 · 0.2	**with**
				Noun NP 1.0 0.2	**cream**

S	⟶	NP VP	0.8
S	⟶	S ConjS	0.2
NP	⟶	Noun	0.2
NP	⟶	Det Noun	0.4
NP	⟶	NP PP	0.2
NP	⟶	NP ConjNP	0.2
VP	⟶	Verb	0.3
VP	⟶	Verb NP	0.3
VP	⟶	Verb NPNP	0.1
VP	⟶	VP PP	0.3
PP	⟶	PP NP	1.0
Prep	⟶	P	1.0
Noun	⟶	N	1.0
Verb	⟶	V	1.0
ConjS	⟶	conj S	1.0
ConjNP	⟶	conj NP	1.0
NPNP	⟶	NP NP	1.0

Probabilistic CKY

Input: POS-tagged sentence

`John_N eats_V pie_N with_P cream_N`

John	eats	pie	with	cream	
Noun NP 1.0 0.2	S 0.8·0.2·0.3	S 0.8·0.2·0.06			**John**
	Verb VP 1.0 0.3	VP 1·0.3·0.2 = 0.06			**eats**
		Noun NP 1.0 0.2			**pie**
			Prep 1.0	PP 1·1·0.2	**with**
				Noun NP 1.0 0.2	**cream**

S	⟶	NP VP	0.8
S	⟶	S ConjS	0.2
NP	⟶	Noun	0.2
NP	⟶	Det Noun	0.4
NP	⟶	NP PP	0.2
NP	⟶	NP ConjNP	0.2
VP	⟶	Verb	0.3
VP	⟶	Verb NP	0.3
VP	⟶	Verb NPNP	0.1
VP	⟶	VP PP	0.3
PP	⟶	PP NP	1.0
Prep	⟶	P	1.0
Noun	⟶	N	1.0
Verb	⟶	V	1.0
ConjS	⟶	conj S	1.0
ConjNP	⟶	conj NP	1.0
NPNP	⟶	NP NP	1.0

Probabilistic CKY

Input: POS-tagged sentence

`John_N eats_V pie_N with_P cream_N`

John	eats	pie	with	cream	
Noun NP 1.0 0.2	S $0.8 \cdot 0.2 \cdot 0.3$	S $0.8 \cdot 0.2 \cdot 0.06$			**John**
	Verb VP 1.0 0.3	VP $1 \cdot 0.3 \cdot 0.2$ $= 0.06$			**eats**
		Noun **NP** 1.0 **0.2**		**NP** $\mathbf{0.2 \cdot 0.2 \cdot 0.2}$ $\mathbf{= 0.008}$	**pie**
			Prep 1.0	**PP** $\mathbf{1 \cdot 1 \cdot 0.2}$	**with**
				Noun NP 1.0 0.2	**cream**

S	⟶	NP VP	0.8
S	⟶	S ConjS	0.2
NP	⟶	Noun	0.2
NP	⟶	Det Noun	0.4
NP	⟶	NP PP	0.2
NP	⟶	NP ConjNP	0.2
VP	⟶	Verb	0.3
VP	⟶	Verb NP	0.3
VP	⟶	Verb NPNP	0.1
VP	⟶	VP PP	0.3
PP	⟶	PP NP	1.0
Prep	⟶	P	1.0
Noun	⟶	N	1.0
Verb	⟶	V	1.0
ConjS	⟶	conj S	1.0
ConjNP	⟶	conj NP	1.0
NPNP	⟶	NP NP	1.0

Probabilistic CKY

Input: POS-tagged sentence

`John_N eats_V pie_N with_P cream_N`

John	eats	pie	with	cream	
Noun NP 1.0 0.2	S $0.8 \cdot 0.2 \cdot 0.3$	S $0.8 \cdot 0.2 \cdot 0.06$			**John**
	Verb VP 1.0 0.3	VP $1 \cdot 0.3 \cdot 0.2$ $= 0.06$			**eats**
		Noun NP 1.0 0.2		NP $0.2 \cdot 0.2 \cdot 0.2$ $= 0.008$	**pie**
			Prep 1.0	PP $1 \cdot 1 \cdot 0.2$	**with**
				Noun NP 1.0 0.2	**cream**

S	$\longrightarrow$	NP VP	0.8
S	$\longrightarrow$	S ConjS	0.2
NP	$\longrightarrow$	Noun	0.2
NP	$\longrightarrow$	Det Noun	0.4
NP	$\longrightarrow$	NP PP	0.2
NP	$\longrightarrow$	NP ConjNP	0.2
VP	$\longrightarrow$	Verb	0.3
VP	$\longrightarrow$	Verb NP	0.3
VP	$\longrightarrow$	Verb NPNP	0.1
VP	$\longrightarrow$	VP PP	0.3
PP	$\longrightarrow$	PP NP	1.0
Prep	$\longrightarrow$	P	1.0
Noun	$\longrightarrow$	N	1.0
Verb	$\longrightarrow$	V	1.0
ConjS	$\longrightarrow$	conj S	1.0
ConjNP	$\longrightarrow$	conj NP	1.0
NPNP	$\longrightarrow$	NP NP	1.0

Probabilistic CKY

Input: POS-tagged sentence

`John_N eats_V pie_N with_P cream_N`

John	eats	pie	with	cream	
Noun NP 1.0 0.2	S $0.8 \cdot 0.2 \cdot 0.3$	S $0.8 \cdot 0.2 \cdot 0.06$			**John**
	Verb VP 1.0 0.3	VP $1 \cdot 0.3 \cdot 0.2$ $= 0.06$			**eats**
		Noun NP 1.0 0.2		NP $0.2 \cdot 0.2 \cdot 0.2$ $= 0.008$	**pie**
			Prep 1.0	PP $1 \cdot 1 \cdot 0.2$	**with**
				Noun NP 1.0 0.2	**cream**

S	$\longrightarrow$	NP VP	0.8
S	$\longrightarrow$	S ConjS	0.2
NP	$\longrightarrow$	Noun	0.2
NP	$\longrightarrow$	Det Noun	0.4
NP	$\longrightarrow$	NP PP	0.2
NP	$\longrightarrow$	NP ConjNP	0.2
VP	$\longrightarrow$	Verb	0.3
VP	$\longrightarrow$	Verb NP	0.3
VP	$\longrightarrow$	Verb NPNP	0.1
VP	$\longrightarrow$	VP PP	0.3
PP	$\longrightarrow$	PP NP	1.0
Prep	$\longrightarrow$	P	1.0
Noun	$\longrightarrow$	N	1.0
Verb	$\longrightarrow$	V	1.0
ConjS	$\longrightarrow$	conj S	1.0
ConjNP	$\longrightarrow$	conj NP	1.0
NPNP	$\longrightarrow$	NP NP	1.0

Probabilistic CKY

Input: POS-tagged sentence

`John_N eats_V pie_N with_P cream_N`

John	eats	pie	with	cream	
Noun NP 1.0 0.2	S 0.8 · 0.2 · 0.3	S 0.8 · 0.2 · 0.06			**John**
	Verb VP 1.0 0.3	VP 1 · 0.3 · 0.2 = 0.06			**eats**
		Noun NP 1.0 0.2		NP 0.2 · 0.2 · 0.2 = 0.008	**pie**
			Prep 1.0	PP 1 · 1 · 0.2	**with**
				Noun NP 1.0 0.2	**cream**

S	$\longrightarrow$	NP VP	0.8
S	$\longrightarrow$	S ConjS	0.2
NP	$\longrightarrow$	Noun	0.2
NP	$\longrightarrow$	Det Noun	0.4
NP	$\longrightarrow$	NP PP	0.2
NP	$\longrightarrow$	NP ConjNP	0.2
VP	$\longrightarrow$	Verb	0.3
VP	$\longrightarrow$	Verb NP	0.3
VP	$\longrightarrow$	Verb NPNP	0.1
VP	$\longrightarrow$	VP PP	0.3
PP	$\longrightarrow$	PP NP	1.0
Prep	$\longrightarrow$	P	1.0
Noun	$\longrightarrow$	N	1.0
Verb	$\longrightarrow$	V	1.0
ConjS	$\longrightarrow$	conj S	1.0
ConjNP	$\longrightarrow$	conj NP	1.0
NPNP	$\longrightarrow$	NP NP	1.0

Probabilistic CKY

Input: POS-tagged sentence

John_N eats_V pie_N with_P cream_N

John	eats	pie	with	cream	
Noun NP 1.0 0.2	S $0.8 \cdot 0.2 \cdot 0.3$	S $0.8 \cdot 0.2 \cdot 0.06$			**John**
	Verb VP 1.0 0.3	VP $1 \cdot 0.3 \cdot 0.2$ $= 0.06$			**eats**
		Noun NP 1.0 0.2		NP $0.2 \cdot 0.2 \cdot 0.2$ $= 0.008$	**pie**
			Prep 1.0	PP $1 \cdot 1 \cdot 0.2$	**with**
				Noun NP 1.0 0.2	**cream**

S	⟶	NP VP	0.8
S	⟶	S ConjS	0.2
NP	⟶	Noun	0.2
NP	⟶	Det Noun	0.4
NP	⟶	NP PP	0.2
NP	⟶	NP ConjNP	0.2
VP	⟶	Verb	0.3
VP	⟶	Verb NP	0.3
VP	⟶	Verb NPNP	0.1
VP	⟶	VP PP	0.3
PP	⟶	PP NP	1.0
Prep	⟶	P	1.0
Noun	⟶	N	1.0
Verb	⟶	V	1.0
ConjS	⟶	conj S	1.0
ConjNP	⟶	conj NP	1.0
NPNP	⟶	NP NP	1.0

Probabilistic CKY

Input: POS-tagged sentence

John_N eats_V pie_N with_P cream_N

John	eats	pie	with	cream	
Noun NP 1.0 0.2	S 0.8·0.2·0.3	S 0.8·0.2·0.06			**John**
	Verb VP **1.0** 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**
		Noun NP 1.0 0.2		**NP** **0.2·0.2·0.2** **= 0.008**	**pie**
			Prep 1.0	**PP** **1·1·0.2**	**with**
				Noun NP 1.0 0.2	**cream**

S	⟶	NP VP	0.8
S	⟶	S ConjS	0.2
NP	⟶	Noun	0.2
NP	⟶	Det Noun	0.4
NP	⟶	NP PP	0.2
NP	⟶	NP ConjNP	0.2
VP	⟶	Verb	0.3
VP	⟶	Verb NP	0.3
VP	⟶	Verb NPNP	0.1
VP	⟶	VP PP	0.3
PP	⟶	PP NP	1.0
Prep	⟶	P	1.0
Noun	⟶	N	1.0
Verb	⟶	V	1.0
ConjS	⟶	conj S	1.0
ConjNP	⟶	conj NP	1.0
NPNP	⟶	NP NP	1.0

Probabilistic CKY

Input: POS-tagged sentence

`John_N eats_V pie_N with_P cream_N`

John	eats	pie	with	cream	
Noun NP 1.0 0.2	S 0.8 · 0.2 · 0.3	S 0.8 · 0.2 · 0.06		**S** 0.2 · 0.0036 · 0.8	**John**
	Verb VP 1.0 0.3	VP 1 · 0.3 · 0.2 = 0.06		**VP** 0.0036	**eats**
		Noun NP 1.0 0.2		NP 0.2 · 0.2 · 0.2 = 0.008	**pie**
			Prep 1.0	PP 1 · 1 · 0.2	**with**
				Noun NP 1.0 0.2	**cream**

S	⟶	NP VP	0.8
S	⟶	S ConjS	0.2
NP	⟶	Noun	0.2
NP	⟶	Det Noun	0.4
NP	⟶	NP PP	0.2
NP	⟶	NP ConjNP	0.2
VP	⟶	Verb	0.3
VP	⟶	Verb NP	0.3
VP	⟶	Verb NPNP	0.1
VP	⟶	VP PP	0.3
PP	⟶	PP NP	1.0
Prep	⟶	P	1.0
Noun	⟶	N	1.0
Verb	⟶	V	1.0
ConjS	⟶	conj S	1.0
ConjNP	⟶	conj NP	1.0
NPNP	⟶	NP NP	1.0

Probabilistic CKY

Input: POS-tagged sentence

`John_N eats_V pie_N with_P cream_N`

John	eats	pie	with	cream	
Noun NP 1.0 0.2	S 0.8 · 0.2 · 0.3	S 0.8 · 0.2 · 0.06		S 0.2 · 0.0036 · 0.8	**John**
	Verb VP 1.0 0.3	VP 1 · 0.3 · 0.2 = 0.06		VP 0.0036	**eats**
		Noun NP 1.0 0.2		NP 0.2 · 0.2 · 0.2 = 0.008	**pie**
			Prep 1.0	PP 1 · 1 · 0.2	**with**
				Noun NP 1.0 0.2	**cream**

S	$\longrightarrow$	NP VP	0.8
S	$\longrightarrow$	S ConjS	0.2
NP	$\longrightarrow$	Noun	0.2
NP	$\longrightarrow$	Det Noun	0.4
NP	$\longrightarrow$	NP PP	0.2
NP	$\longrightarrow$	NP ConjNP	0.2
VP	$\longrightarrow$	Verb	0.3
VP	$\longrightarrow$	Verb NP	0.3
VP	$\longrightarrow$	Verb NPNP	0.1
VP	$\longrightarrow$	VP PP	0.3
PP	$\longrightarrow$	PP NP	1.0
Prep	$\longrightarrow$	P	1.0
Noun	$\longrightarrow$	N	1.0
Verb	$\longrightarrow$	V	1.0
ConjS	$\longrightarrow$	conj S	1.0
ConjNP	$\longrightarrow$	conj NP	1.0
NPNP	$\longrightarrow$	NP NP	1.0

Probabilistic CKY

Input: POS-tagged sentence

`John_N eats_V pie_N with_P cream_N`

John	eats	pie	with	cream	
Noun NP 1.0 0.2	S $0.8 \cdot 0.2 \cdot 0.3$	S $0.8 \cdot 0.2 \cdot 0.06$		S $0.2 \cdot 0.0036 \cdot 0.8$	**John**
	Verb VP 1.0 0.3	VP $1 \cdot 0.3 \cdot 0.2$ $= 0.06$		VP 0.0036	**eats**
		Noun NP 1.0 0.2		NP $0.2 \cdot 0.2 \cdot 0.2$ $= 0.008$	**pie**
			Prep 1.0	PP $1 \cdot 1 \cdot 0.2$	**with**
				Noun NP 1.0 0.2	**cream**

S	$\longrightarrow$	NP VP	0.8
S	$\longrightarrow$	S ConjS	0.2
NP	$\longrightarrow$	Noun	0.2
NP	$\longrightarrow$	Det Noun	0.4
NP	$\longrightarrow$	NP PP	0.2
NP	$\longrightarrow$	NP ConjNP	0.2
VP	$\longrightarrow$	Verb	0.3
VP	$\longrightarrow$	Verb NP	0.3
VP	$\longrightarrow$	Verb NPNP	0.1
VP	$\longrightarrow$	VP PP	0.3
PP	$\longrightarrow$	PP NP	1.0
Prep	$\longrightarrow$	P	1.0
Noun	$\longrightarrow$	N	1.0
Verb	$\longrightarrow$	V	1.0
ConjS	$\longrightarrow$	conj S	1.0
ConjNP	$\longrightarrow$	conj NP	1.0
NPNP	$\longrightarrow$	NP NP	1.0

Probabilistic CKY

Input: POS-tagged sentence

`John_N eats_V pie_N with_P cream_N`

John	eats	pie	with	cream	
Noun NP 1.0 0.2	S $0.8 \cdot 0.2 \cdot 0.3$	S $0.8 \cdot 0.2 \cdot 0.06$		S $0.2 \cdot 0.0036 \cdot 0.8$	**John**
	Verb VP 1.0 0.3	VP $1 \cdot 0.3 \cdot 0.2$ $= 0.06$		VP 0.0036	**eats**
		Noun NP 1.0 0.2		NP $0.2 \cdot 0.2 \cdot 0.2$ $= 0.008$	**pie**
			Prep 1.0	PP $1 \cdot 1 \cdot 0.2$	**with**
				Noun NP 1.0 0.2	**cream**

S	$\longrightarrow$	NP VP	0.8
S	$\longrightarrow$	S ConjS	0.2
NP	$\longrightarrow$	Noun	0.2
NP	$\longrightarrow$	Det Noun	0.4
NP	$\longrightarrow$	NP PP	0.2
NP	$\longrightarrow$	NP ConjNP	0.2
VP	$\longrightarrow$	Verb	0.3
VP	$\longrightarrow$	Verb NP	0.3
VP	$\longrightarrow$	Verb NPNP	0.1
VP	$\longrightarrow$	VP PP	0.3
PP	$\longrightarrow$	PP NP	1.0
Prep	$\longrightarrow$	P	1.0
Noun	$\longrightarrow$	N	1.0
Verb	$\longrightarrow$	V	1.0
ConjS	$\longrightarrow$	conj S	1.0
ConjNP	$\longrightarrow$	conj NP	1.0
NPNP	$\longrightarrow$	NP NP	1.0

Extracting the final tree

Input: POS-tagged sentence

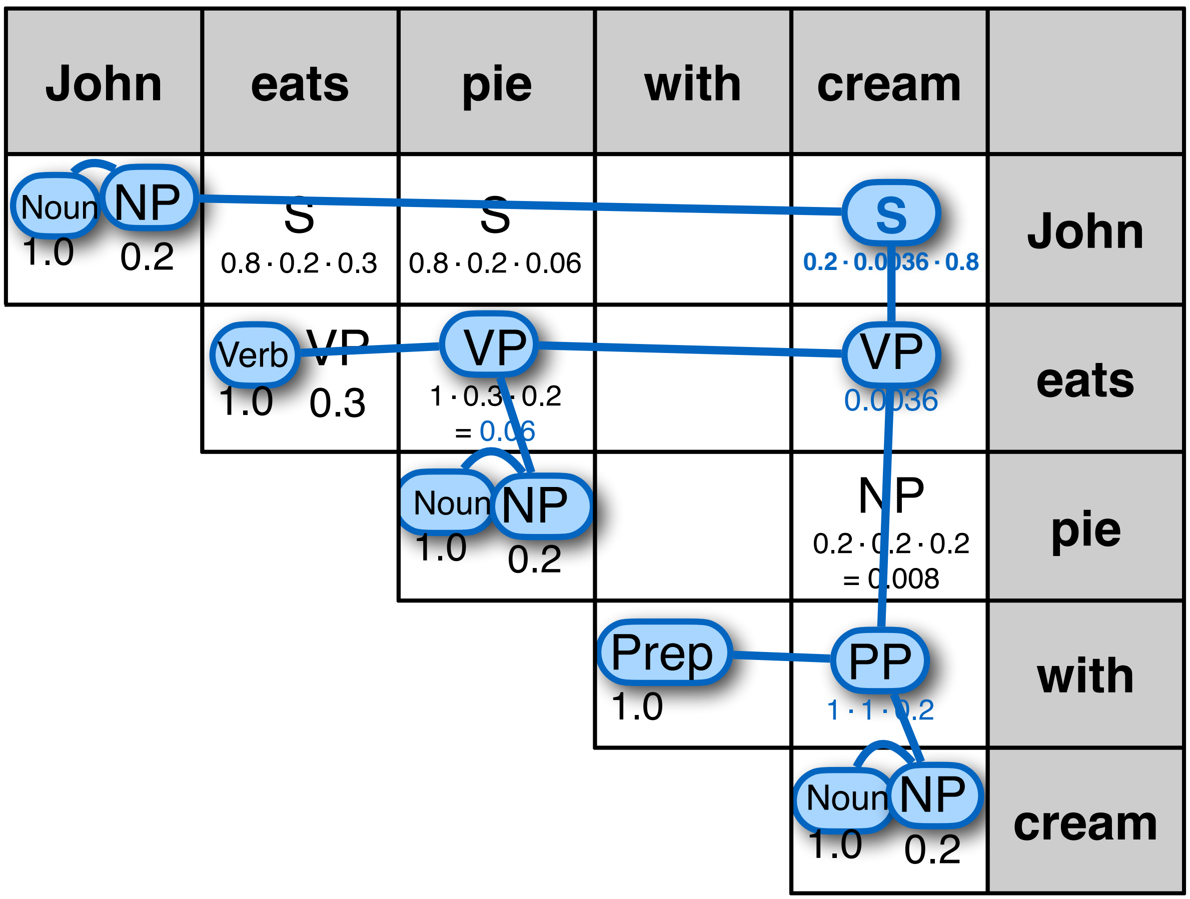
`John_N eats_V pie_N with_P cream_N`

John	eats	pie	with	cream	
Noun NP 1.0 0.2	S $0.8 \cdot 0.2 \cdot 0.3$	S $0.8 \cdot 0.2 \cdot 0.06$		$\boxed{S}$ $0.2 \cdot 0.0036 \cdot 0.8$	**John**
	Verb VP 1.0 0.3	VP $1 \cdot 0.3 \cdot 0.2 = 0.06$		VP 0.0036	**eats**
		Noun NP 1.0 0.2		NP $0.2 \cdot 0.2 \cdot 0.2 = 0.008$	**pie**
			Prep 1.0	PP $1 \cdot 1 \cdot 0.2$	**with**
				Noun NP 1.0 0.2	**cream**

Extracting the final tree

Input: POS-tagged sentence

`John_N eats_V pie_N with_P cream_N`




```
(S (NP (N John))
   (VP (VP (Verb eats)
           (NP (Noun pie)))
       (PP (Prep with)
           (NP (Noun cream)))))
```

Shortcomings of PCFGs

How well can a PCFG model the distribution of trees?

PCFGs make **independence assumptions**:

Only the label of a node determines what children it has.

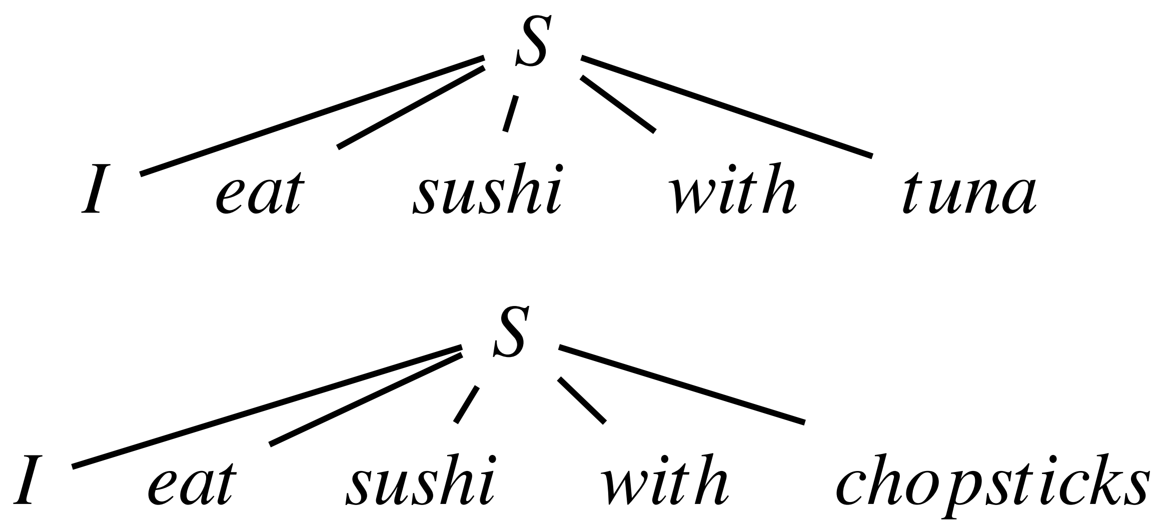
Factors that influence these assumptions:

Shape of the trees:
A corpus with **flat trees** (i.e. few nodes/sentence)
results in a model with few independence assumptions.

Labeling of the trees:
A corpus with **many node labels** (nonterminals)
results in a model with few independence assumptions.

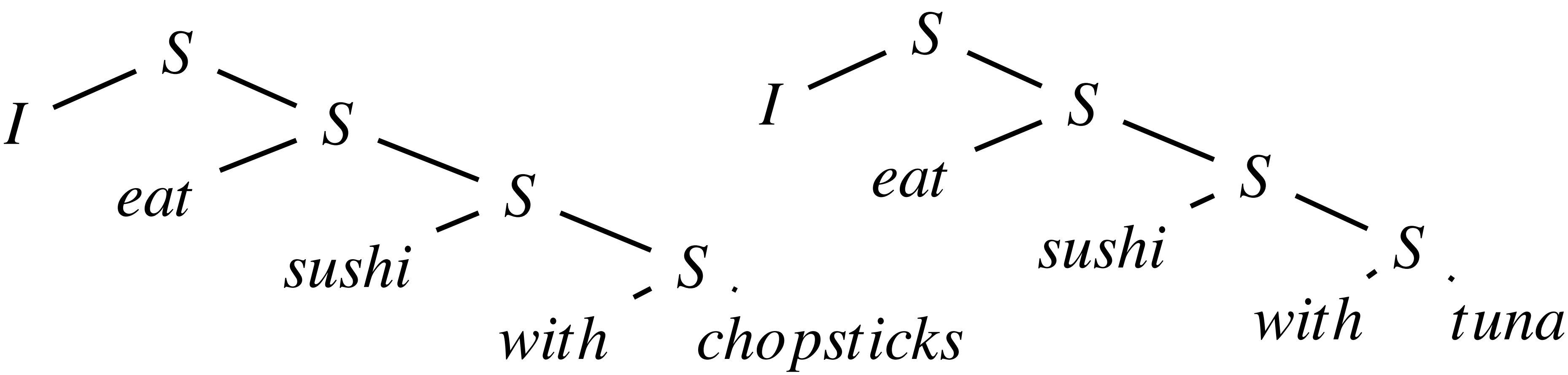
Example 1: flat trees

S -> Pierre Vinken



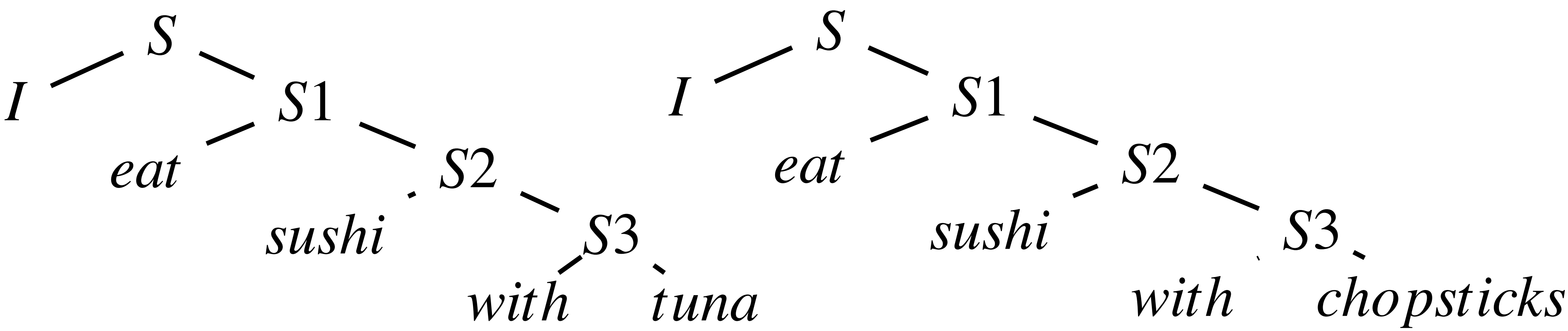
What sentences would a PCFG estimated from this corpus generate?

Example 2: deep trees, few labels



What sentences would a PCFG estimated from this corpus generate?

Example 3: deep trees, many labels



What sentences would a PCFG estimated from this corpus generate?

Two ways to improve performance

… change the (internal) grammar:

- **Parent annotation/state splits:**
  Not all NPs/VPs/DTs/… are the same.
  It matters where they are in the tree

… change the probability model:

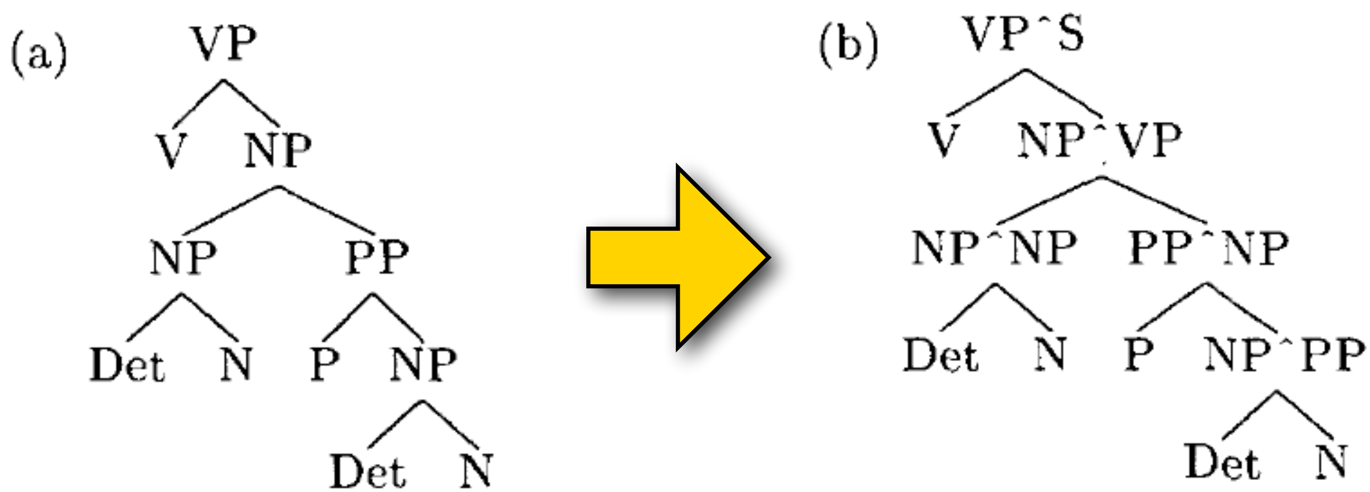
- **Lexicalization:**
  Words matter!
- **Markovization:**
  Generalizing the rules

The parent transformation

PCFGs assume the expansion of any nonterminal is independent of its parent.

- But this is not true: NP subjects more likely to be modified than objects.

We can change the grammar by adding the name of the parent node to each nonterminal



Lexicalization

PCFGs can't distinguish between
"eat sushi with chopsticks" and *"eat sushi with tuna"*.

We need to take words into account!

$P(VP_{eat} \rightarrow VP\ PP_{with\ chopsticks} \mid VP_{eat})$

vs. $P(VP_{eat} \rightarrow VP\ PP_{with\ tuna} \mid VP_{eat})$

Problem: sparse data ($PP_{with\ fatty|white|...\ tuna....}$)
Solution: only take **head words** into account!

Assumption: each constituent has one head word.

Lexicalized PCFGs

At the root (start symbol S), generate the head word of the sentence, w_S, with $P(w_S)$

Lexicalized rule probabilities:
Every nonterminal is lexicalized: X_{w_x}
Condition rules $X_{w_x} \rightarrow \alpha Y \beta$ on the lexicalized LHS X_{w_x}
$P(X_{w_x} \rightarrow \alpha Y \beta \mid X_{w_x})$

Word-word dependencies:
For each nonterminal Y in RHS of a rule $X_{w_x} \rightarrow \alpha Y \beta$,
condition w_Y (the head word of Y) on X and w_X:
$P(w_Y \mid Y, X, w_X)$

Dealing with unknown words

A lexicalized PCFG assigns zero probability
to any word that does not appear in the training data.

Solution:

Training: Replace rare words in training data
with a token ‘UNKNOWN’.

Testing: Replace unseen words with ‘UNKNOWN’

Markov PCFGs (Collins parser)

The RHS of each CFG rule consists of:
one head H_X, *n* left sisters L_i and *m* right sisters R_i:

$$X \longrightarrow \underbrace{L_n \ldots L_1}_{\text{left sisters}} \; H_X \; \underbrace{R_1 \ldots R_m}_{\text{right sisters}}$$

Replace rule probabilities with a generative process:
For each nonterminal X

- generate its head H_X (nonterminal or terminal)
- then generate its left sisters $L_{1..n}$ and a STOP symbol conditioned on H_X
- then generate its right sisters $R_{1...n}$ and a STOP symbol conditioned on H_X

Penn Treebank Parsing

The Penn Treebank

The first publicly available syntactically annotated corpus

Wall Street Journal (50,000 sentences, 1 million words)

also Switchboard, Brown corpus, ATIS

The annotation:

- POS-tagged (Ratnaparkhi's MXPOST)
- Manually annotated with phrase-structure trees
- Richer than standard CFG: *Traces* and other *null elements* used to represent non-local dependencies (designed to allow extraction of predicate-argument structure),
  although these are typically removed when we do parsing
  [more on non-local dependencies and traces later in the semester]

The standard data set for English phrase-structure parsers

The Treebank label set

48 preterminals (tags):

- 36 POS tags, 12 other symbols (punctuation etc.)
- Simplified version of Brown tagset (87 tags)
  (cf. Lancaster-Oslo/Bergen (LOB) tag set: 126 tags)

14 nonterminals:

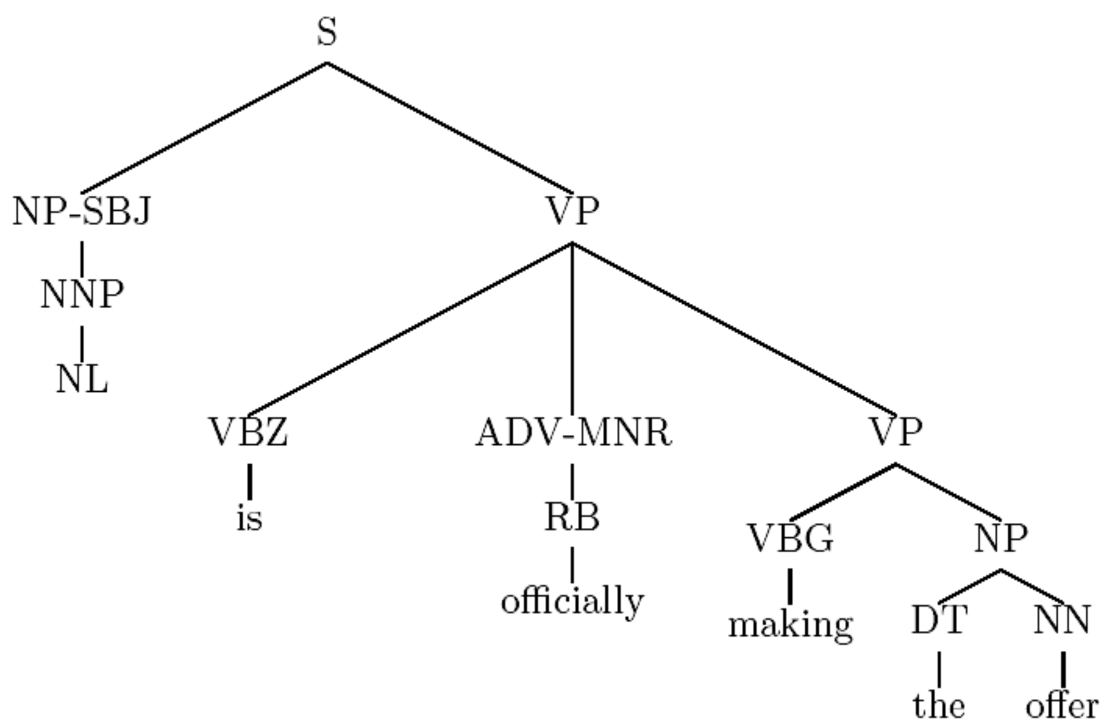
Standard inventory (S, NP, VP, PP, ADJP, ADVP, SBAR,…)

Many nonterminals have function tags indicating their syntactic roles (NP-SBJ: subject NP) or what role they play

(e.g. PP-LOC: locative PP, i.e. indicating a location [“in NYC”]
PP-DIR: directional PP, indicating a direction [“to NYC”],
ADVP-MNR: manner adverb [“slowly”]).

For historical reasons, these function tags are typically removed before parsing.

A simple example



Relatively flat structures:

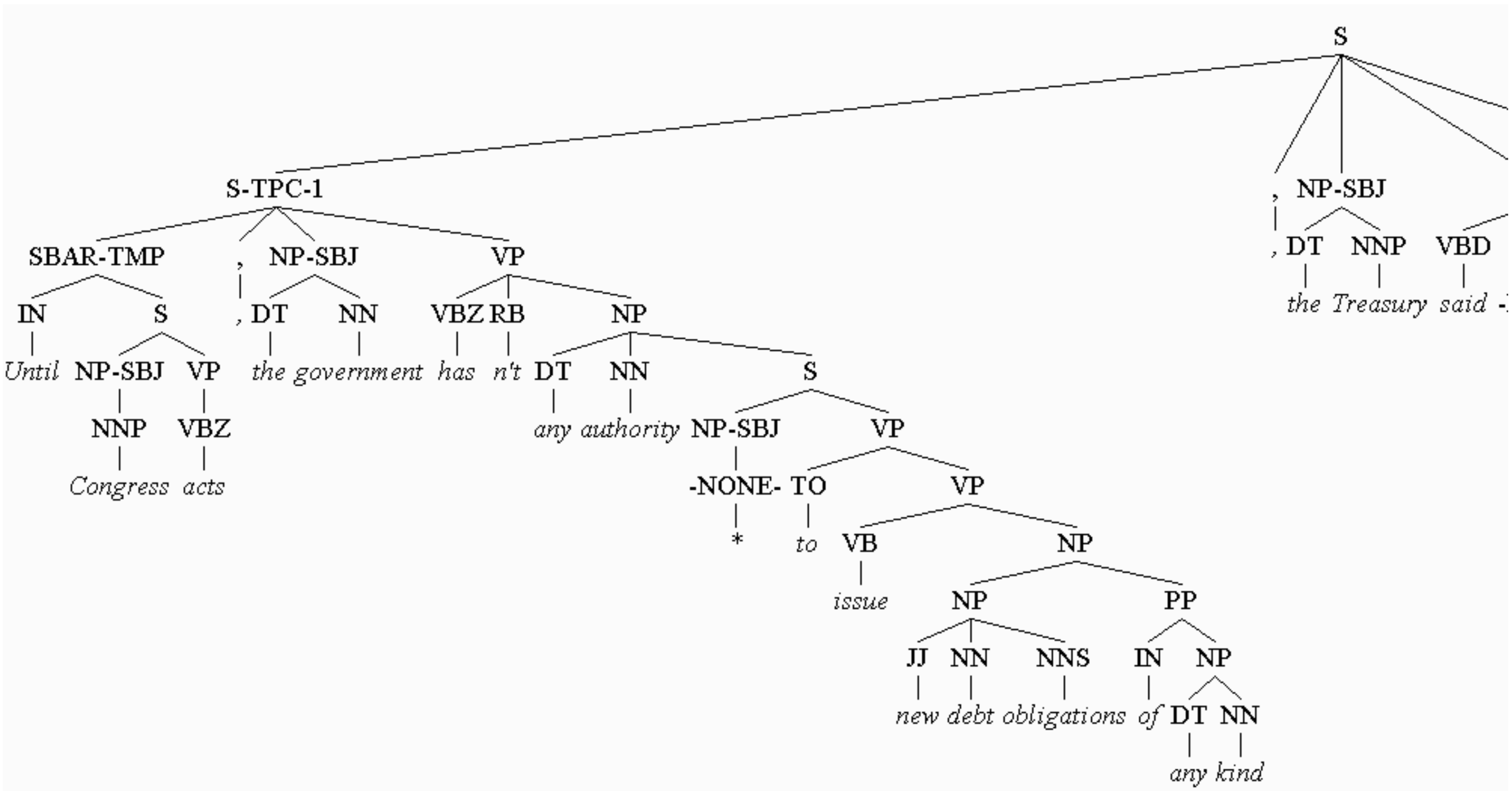
– There is no noun level

– VP arguments and adjuncts appear at the same level

Function tags, e.g. -SBJ (subject), -MNR (manner)

A more realistic (partial) example

Until Congress acts, the government hasn't any authority to issue new debt obligations of any kind, the Treasury said .... .



The Penn Treebank CFG

The Penn Treebank uses very flat rules, e.g.:

```
NP → DT JJ NN
NP → DT JJ NNS
NP → DT JJ NN NN
NP → DT JJ JJ NN
NP → DT JJ CD NNS
NP → RB DT JJ NN NN
NP → RB DT JJ JJ NNS
NP → DT JJ JJ NNP NNS
NP → DT NNP NNP NNP NNP JJ NN
NP → DT JJ NNP CC JJ JJ NN NNS
NP → RB DT JJS NN NN SBAR
NP → DT VBG JJ NNP NNP CC NNP
NP → DT JJ NNS , NNS CC NN NNS NN
NP → DT JJ JJ VBG NN NNP NNP FW NNP
NP → NP JJ , JJ `` SBAR '' NNS
```

Basic PCFGs don't work well on the Penn Treebank

– Many of these rules appear only once.
– But many of these rules are very similar.
  Can we generalize by not treating each rule as atomic?

Summary

The Penn Treebank has a large number of very flat rules.

Accurate parsing requires modifications to basic PCFG models:
— Generalizing across similar rules ("Markov PCFGs")
— Modeling word-word dependencies
(although this does not help as much as people used to think)
— Refining the nonterminals to capture more context

How much of this transfers to other treebanks or languages?

Appendix:
A context-free grammar
for a fragment of
English

Noun phrases (NPs)

Simple NPs:

[He] sleeps. (pronoun)

[John] sleeps. (proper name)

[A student] sleeps. (determiner + noun)

[A tall student] sleeps. (det + adj + noun)

[Snow] falls. (noun)

Complex NPs:

[The student in the back] sleeps. (NP + PP)

[The student who likes MTV] sleeps. (NP + Relative Clause)

The NP fragment

```
NP → Pronoun
NP → ProperName
NP → Det Noun
NP → Noun
NP → NP PP
NP → NP RelClause
Noun → AdjP Noun
Noun → N
N          → {class,… student, snow, …}
Det        → {a, the, every,… }
Pronoun    → {he, she,…}
ProperName → {John, Mary,…}
```

Adjective phrases (AdjP) and prepositional phrases (PP)

```
AdjP → Adj
AdjP → Adv AdjP
Adj  → {big, small, red,…}
Adv  → {very, really,…}

PP → P NP
P  → {with, in, above,…}
```

The verb phrase (VP)

He [eats].
He [eats sushi].
He [gives John sushi].
He [gives sushi to John].
He [eats sushi with chopsticks].
He [somtimes eats].

```
VP → V
VP → V NP
VP → V NP NP
VP → V NP PP
VP → VP PP
VP → AdvP VP
V → {eats, sleeps gives,…}
```

Capturing subcategorization

He [eats]. ✔
He [eats sushi]. ✔
He [gives John sushi]. ✔
He [eats sushi with chopsticks]. ✔
*He [eats John sushi]. ???

$VP \rightarrow V_{intrans}$

$VP \rightarrow V_{trans}\ NP$

$VP \rightarrow V_{ditrans}\ NP\ NP$

$VP \rightarrow VP\ PP$

$V_{intrans} \rightarrow \{eats, sleeps\}$
$V_{trans} \rightarrow \{eats\}$
$V_{ditrans} \rightarrow \{gives\}$

Sentences

[He eats sushi].
[Sometimes, he eats sushi].
[In Japan, he eats sushi].

```
S → NP VP
S → AdvP S
S → PP S
```

Capturing agreement

[He eats sushi]. ✔
*[I eats sushi]. ???
*[They eats sushi]. ???

$S \rightarrow NP_{3sg}\ VP_{3sg}$

$S \rightarrow NP_{1sg}\ VP_{1sg}$

$S \rightarrow NP_{3pl}\ VP_{3pl}$

We would need features to capture agreement:
(number, person, case,…)

Complex VPs

In English, simple tenses have separate forms:

Present tense: the girl **eats** sushi

Simple past tense: the girl **ate** sushi

Complex tenses, progressive aspect and passive voice consist of auxiliaries and participles:

Past perfect tense: the girl **has eaten** sushi

Future perfect tense: the girl **will have eaten** sushi

Passive voice: the sushi **is/was/will be/... eaten** by the girl

Progressive aspect: the girl **is/was/will be eating** sushi

VPs redefined

He [has [eaten sushi]].
The sushi [was [eaten by him]].

$\texttt{VP} \rightarrow \texttt{V}_{\texttt{have}}\ \texttt{VP}_{\texttt{pastPart}}$

$\texttt{VP} \rightarrow \texttt{V}_{\texttt{be}}\ \texttt{VP}_{\texttt{pass}}$

$\texttt{VP}_{\texttt{pastPart}} \rightarrow \texttt{V}_{\texttt{pastPart}}\ \texttt{NP}$

$\texttt{VP}_{\texttt{pass}} \rightarrow \texttt{V}_{\texttt{pastPart}}\ \texttt{PP}$

$\texttt{V}_{\texttt{have}} \rightarrow \{\texttt{has}\}$

$\texttt{V}_{\texttt{pastPart}} \rightarrow \{\texttt{eaten, seen}\}$

We would need even more nonterminals (e.g. $VP_{pastpart}$)!

N.B.: We call $VP_{pastPart}$, VP_{pass}, etc. `untensed’ VPs

Subordination

He says [he eats sushi].
He says [that [he eats sushi]].

$\texttt{VP} \rightarrow \texttt{V}_{\texttt{comp}}\ \texttt{S}$

$\texttt{VP} \rightarrow \texttt{V}_{\texttt{comp}}\ \texttt{SBAR}$

$\texttt{SBAR} \rightarrow \texttt{COMP}\ \texttt{S}$

$\texttt{V}_{\texttt{comp}} \rightarrow \{\texttt{says, think, believes}\}$

$\texttt{COMP} \rightarrow \{\texttt{that}\}$

Coordination

[He eats sushi] but [she drinks tea]
[John] and [Mary] eat sushi.
He [eats sushi] and [drinks tea]
He [sells and buys] shares
He eats [at home or at a restaurant]

```
S    → S conj S
NP   → NP conj NP
VP   → VP conj VP
V    → V conj V
PP   → PP conj PP
```

Relative clauses

Relative clauses modify noun phrases:

the girl [that eats sushi] (`NP → NP RelClause`)

Relative clauses lack an NP that is understood to be filled by the NP they modify:

'the girl that eats sushi' implies 'the girl eats sushi'

Subject relative clauses lack a subject: 'the girl that eats sushi'

`RelClause → RelPron VP` [sentence w/o sbj = VP]

Object relative clauses lack an object: 'the sushi that the girl eats'
Define "slash categories" `S-NP,VP-NP` that are missing object NPs

```
RelClause → RelPron S-NP
S-NP      → NP VP-NP
VP-NP     → Vtrans
VP-NP     → VP-NP PP
```

Yes/No questions

Yes/no questions consist of an auxiliary, a subject and an (untensed) verb phrase:

does she eat sushi?

have you eaten sushi?

$\texttt{YesNoQ} \rightarrow \texttt{Aux NP VP}_{\texttt{inf}}$

$\texttt{YesNoQ} \rightarrow \texttt{Aux NP VP}_{\texttt{pastPart}}$

Wh-questions

Subject wh-questions consist of an wh-word,
an auxiliary and an (untensed) verb phrase:

Who has eaten the sushi?

$\texttt{WhQ} \rightarrow \texttt{WhPron Aux VP}_{\texttt{pastPart}}$

Object wh-questions consist of an wh-word,
an auxiliary, an NP and an (untensed) verb phrase
that is missing an object.

What does Mary eat?

$\texttt{WhQ} \rightarrow \texttt{WhPron Aux NP VP}_{\texttt{inf}}\texttt{-NP}$