CS447: Natural Language Processing
http://courses.engr.illinois.edu/cs447

Lecture 15: Formal Grammars of English

Julia Hockenmaier
juliahmr@illinois.edu
3324 Siebel Center

Lecture 15:
Introduction to Syntactic Parsing

Part 1: 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

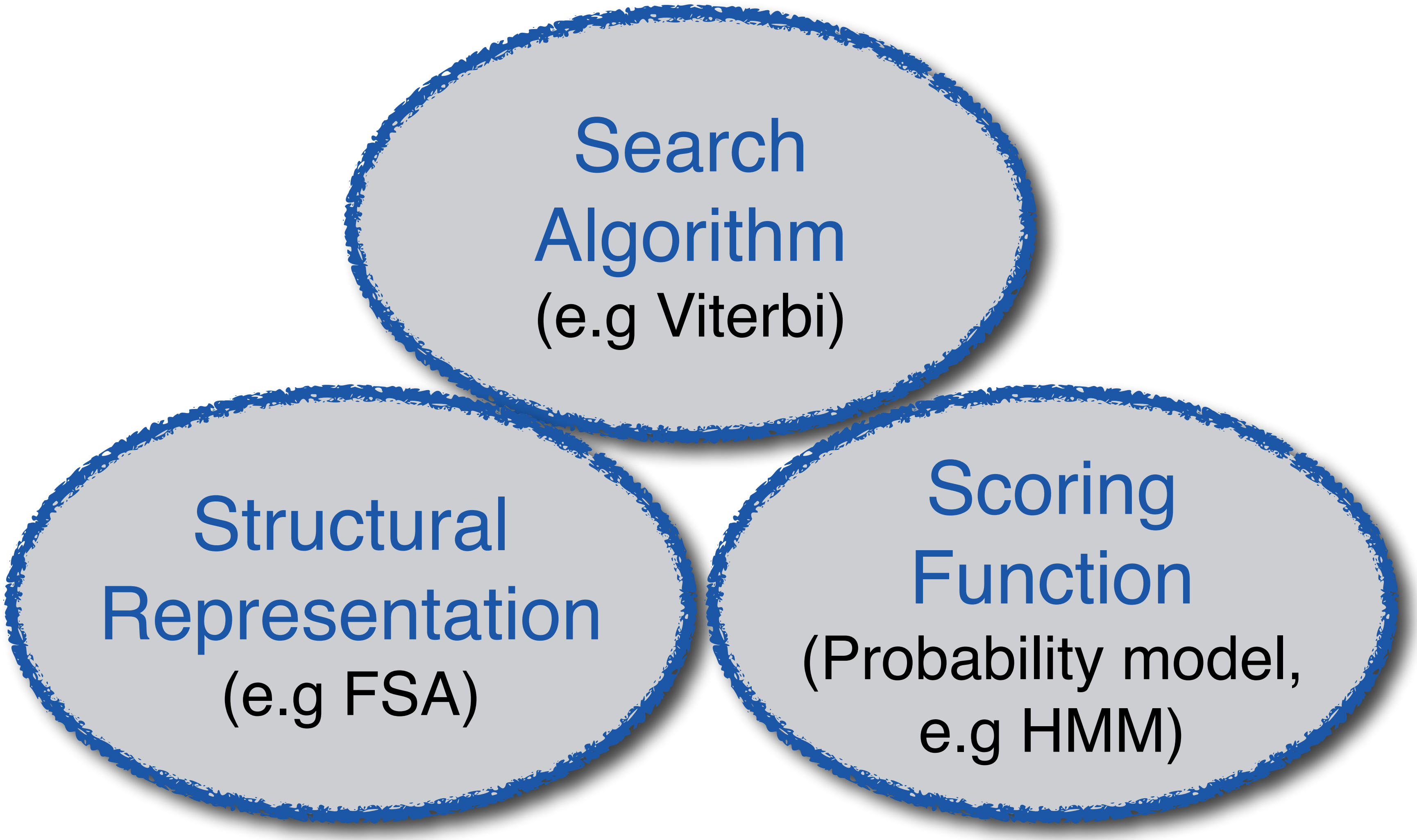
… the corresponding **probabilistic models**,

- Probabilistic Context-Free Grammars

… and appropriate **search algorithms**

- Dynamic programming: CKY parsing

Dealing with ambiguity



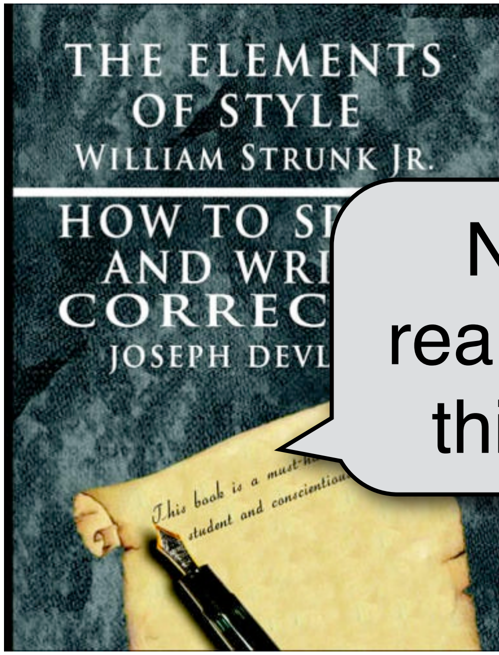
Today's lecture

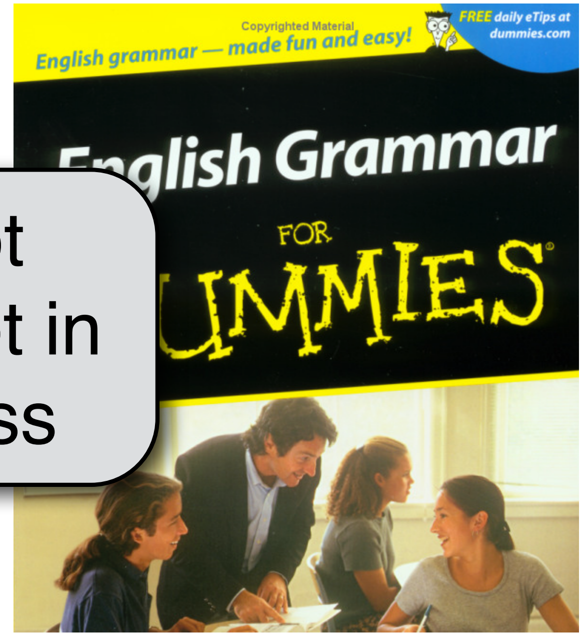
Introduction to natural language syntax ('grammar'):

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.

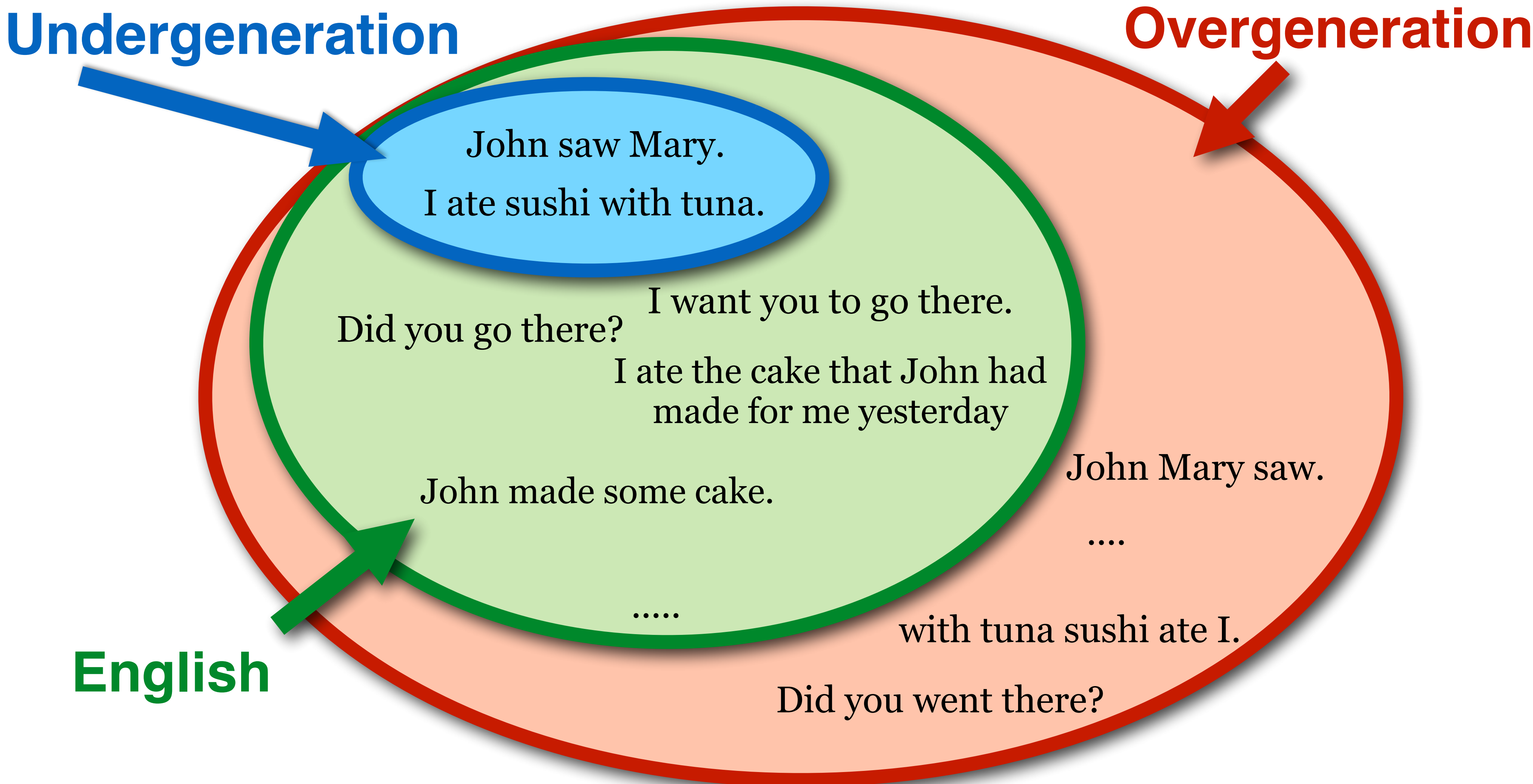
(N.B.: There are many different formalisms out there, which each define their own data structures and operations)

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)

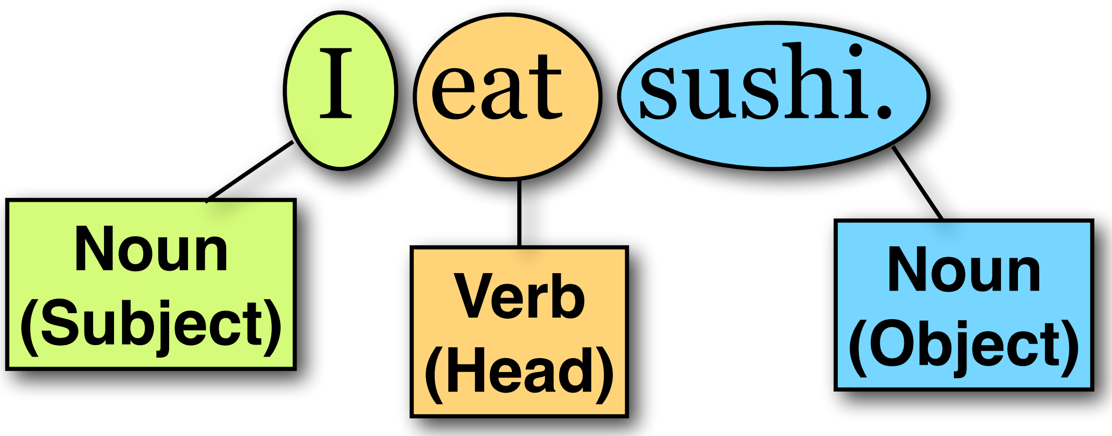
Challenge 2: Don't *overgenerate*!

(Your program should not generate word salad)

Challenge 3: Use a finite program!

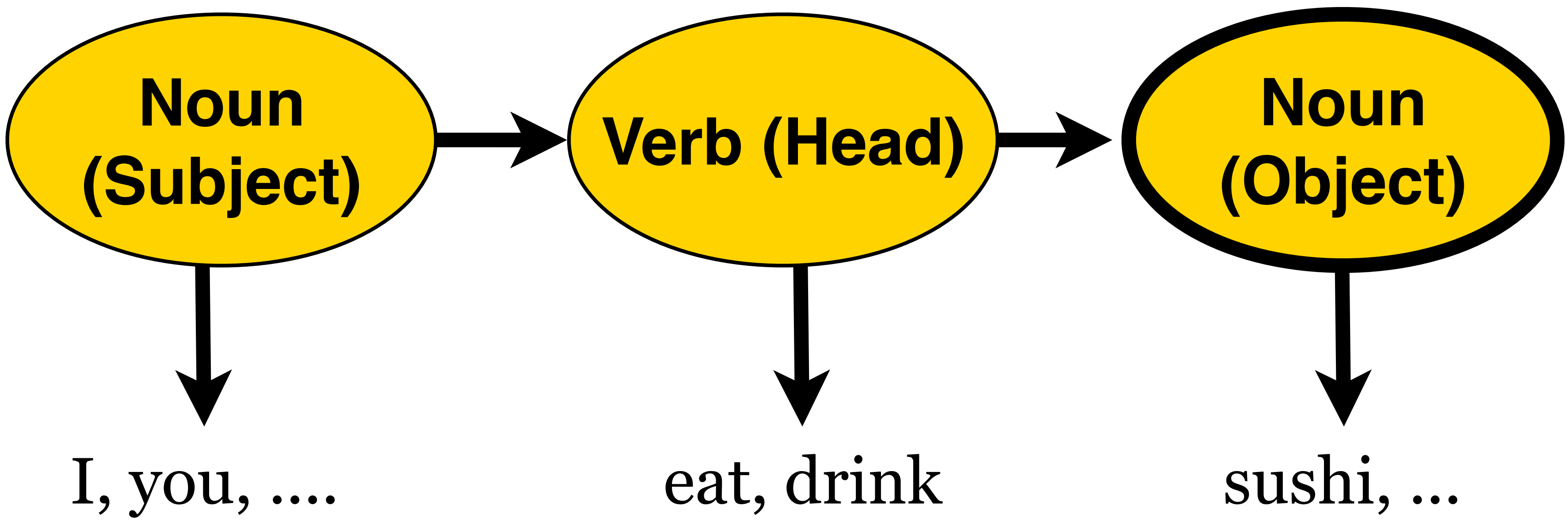
Recursion creates an infinite number of sentences
(even with a finite vocabulary),
but we need our program to be of finite size

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.

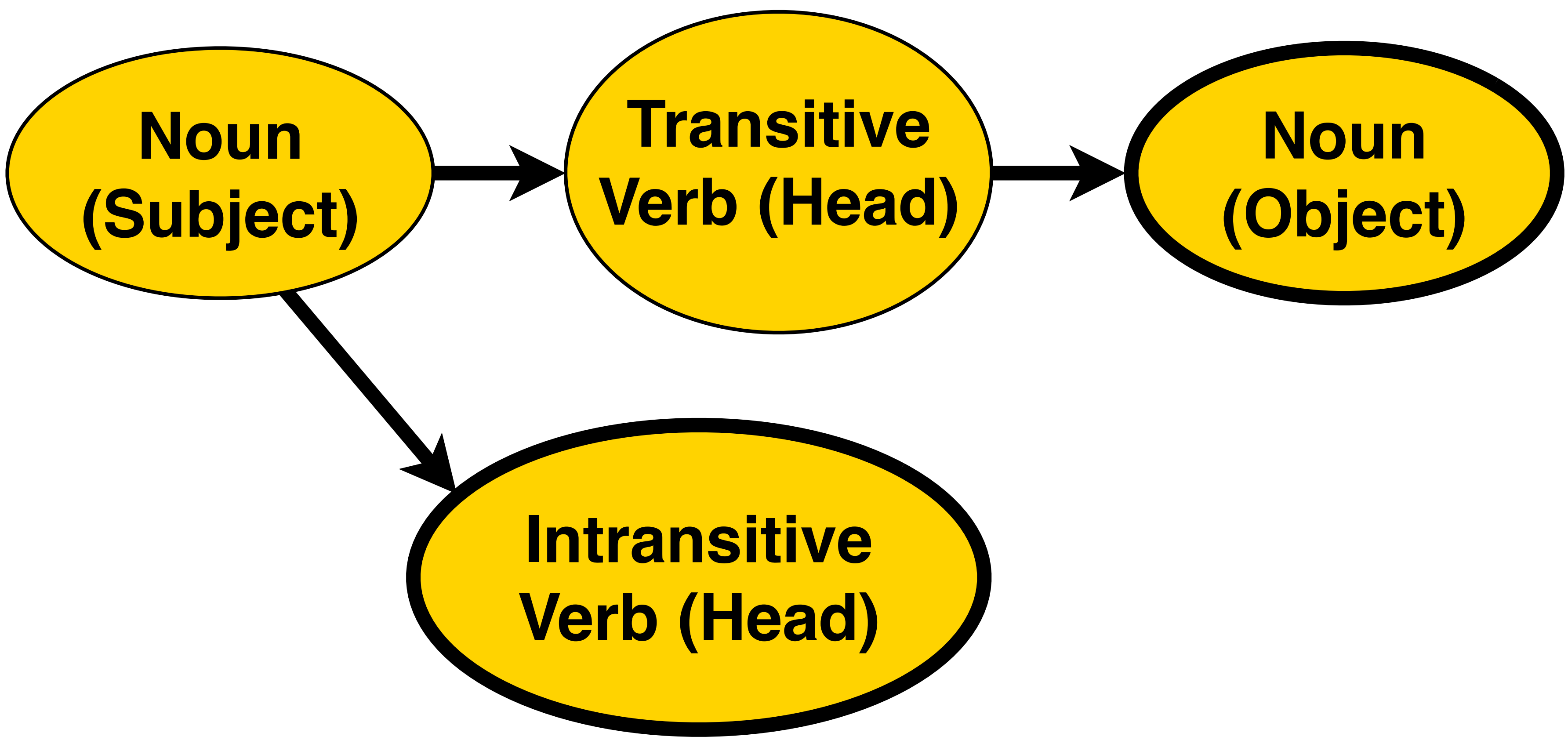
Ditransitive verbs (give) take a subject, direct object and indirect object.

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)

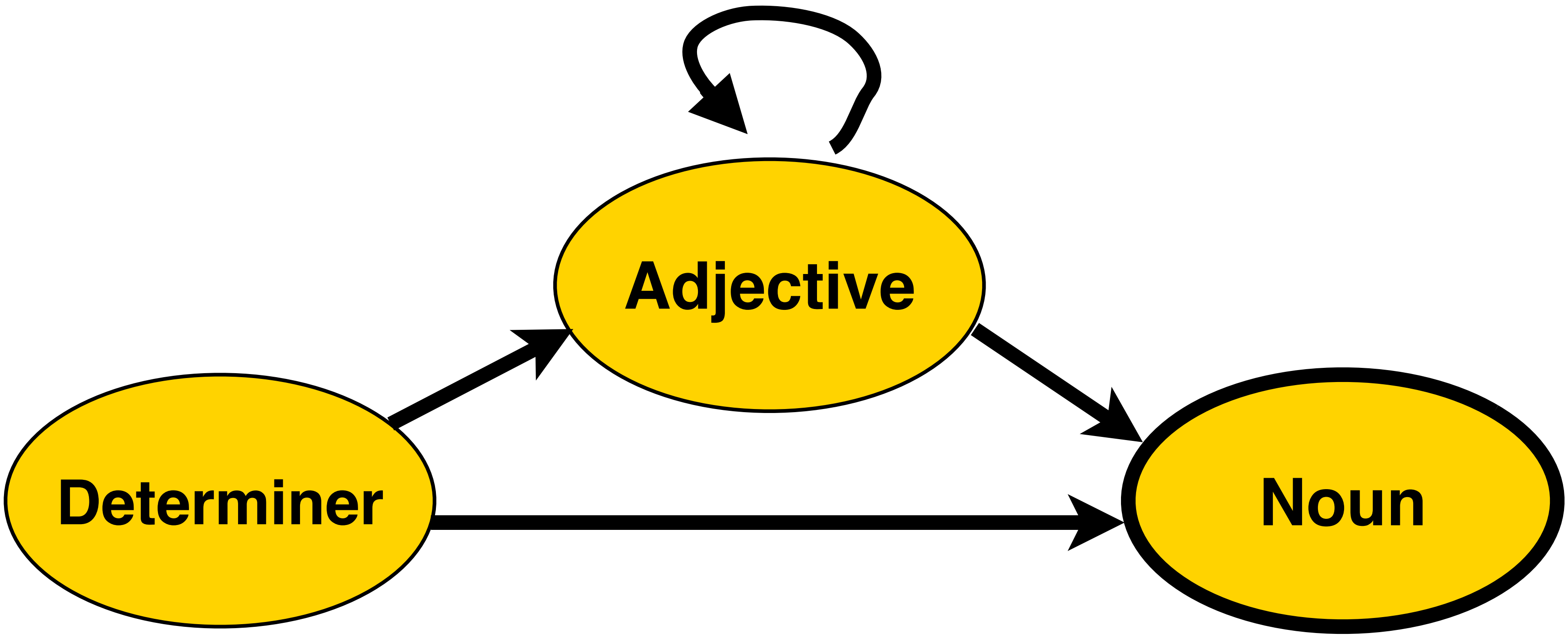
Language is recursive

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**.

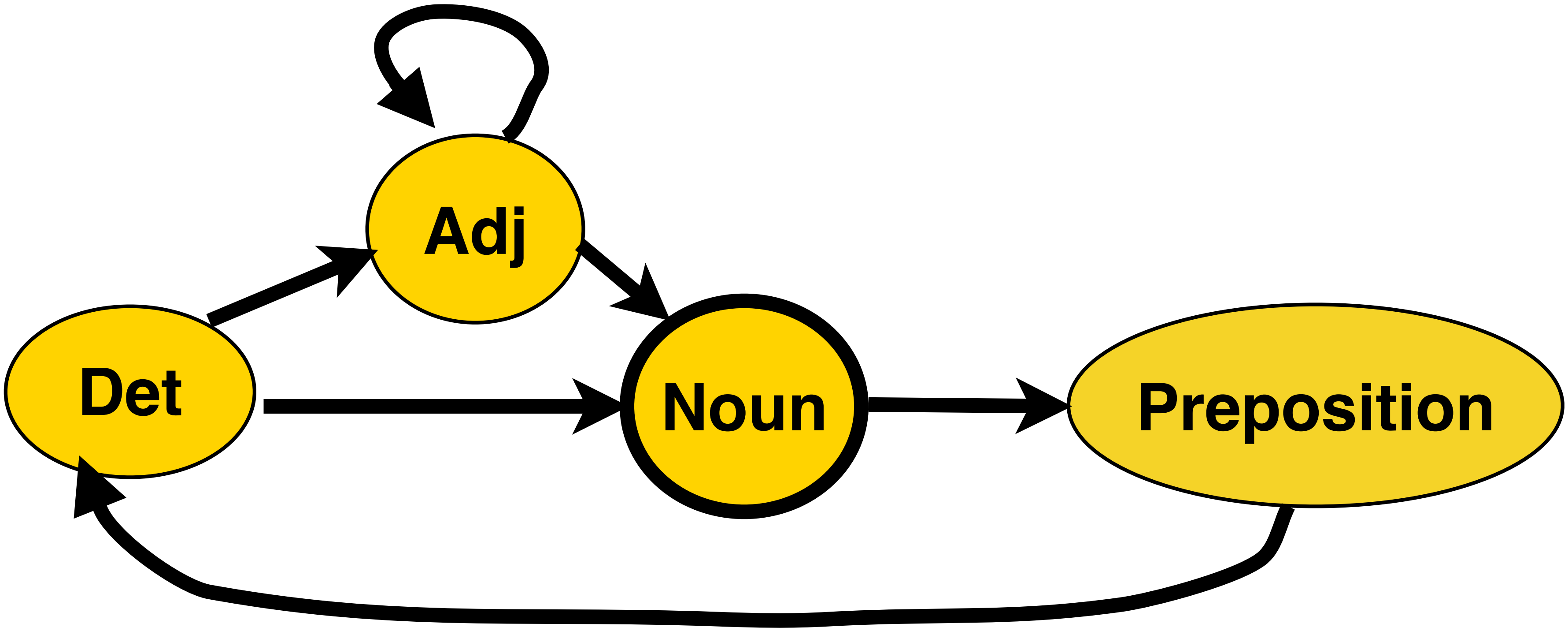
Another FSA



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
....

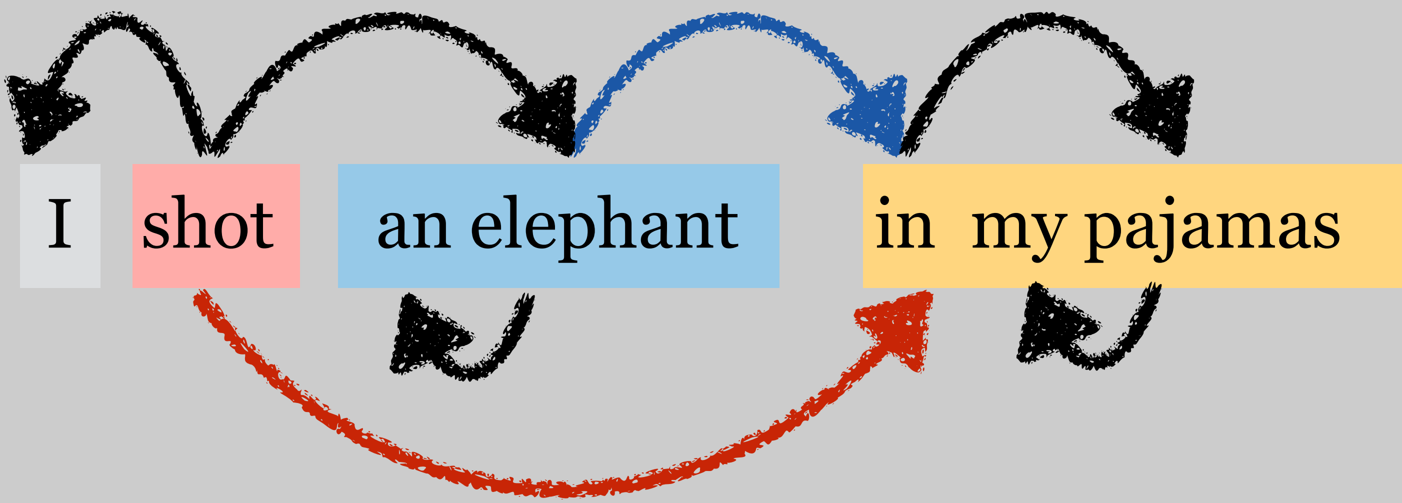
Yet another FSA



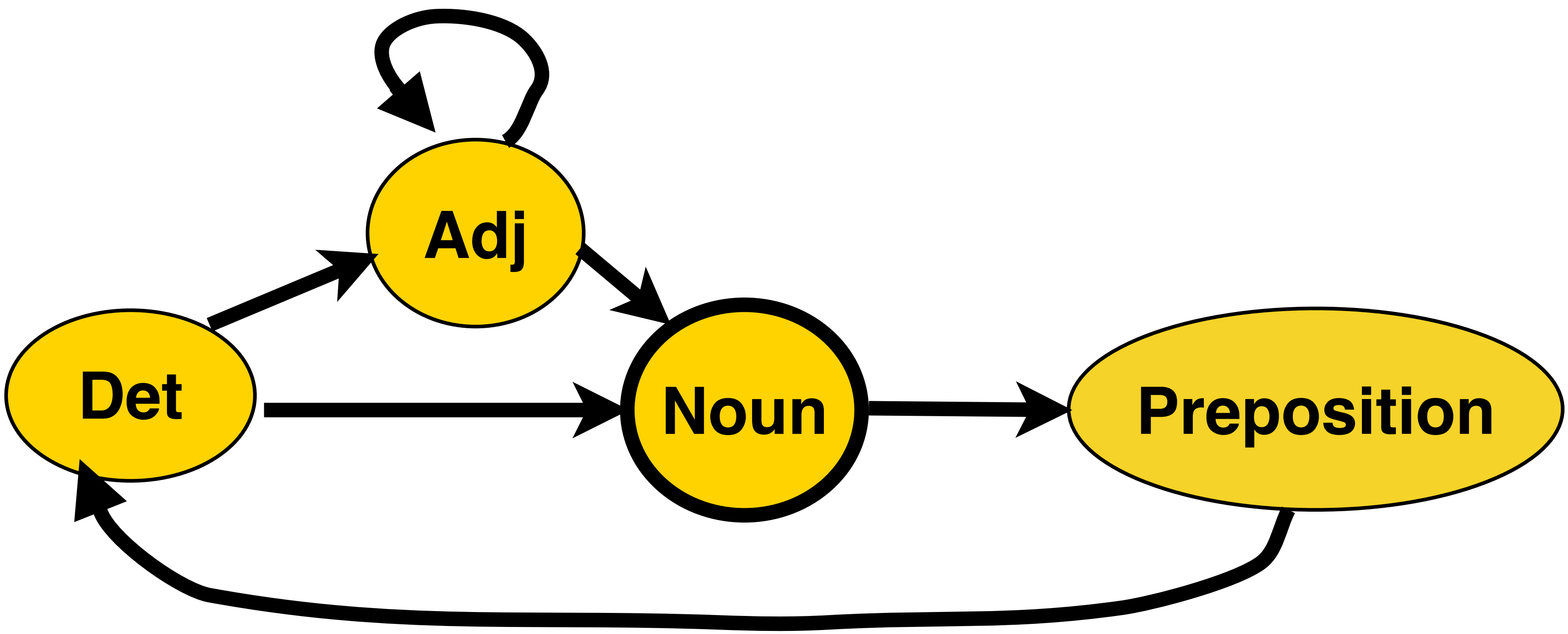
So, why do we need anything
beyond regular (finite-state) grammars?

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



Strong vs. weak generative capacity

Formal language theory:

– defines language as string sets
– is only concerned with generating these strings
  (*weak* generative capacity)

Formal/Theoretical syntax (in linguistics):

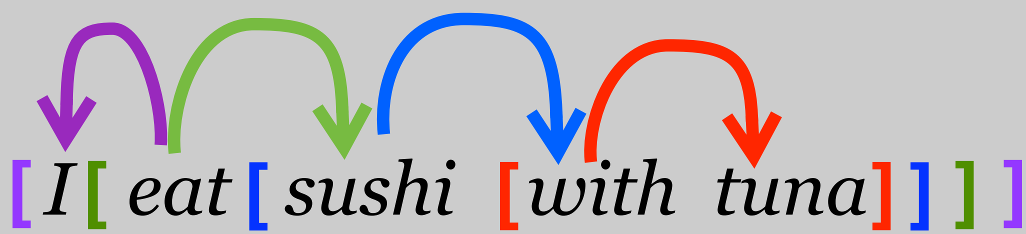
– defines language as sets of strings with (hidden) structure
– is also concerned with generating the right *structures* for these strings
  (*strong* generative capacity)

What is the structure of a sentence?

Sentence structure is **hierarchical**:

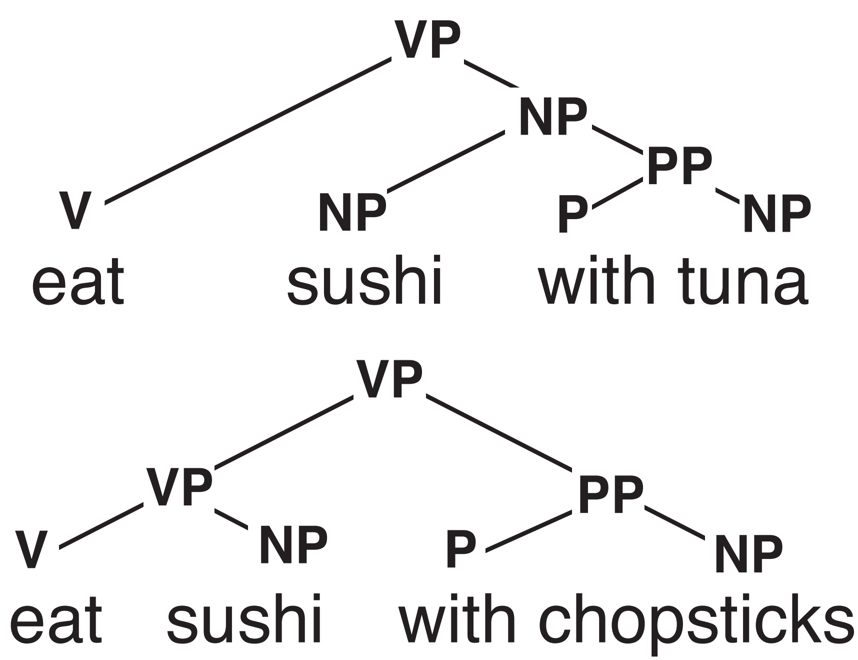
A sentence consists of **words** (I, eat, sushi, with, tuna)
…which form phrases or **constituents**: “sushi with tuna”

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

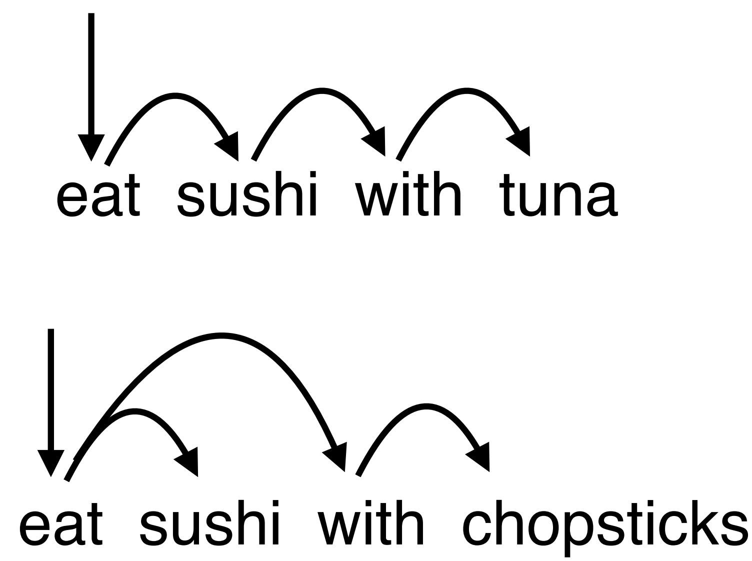


Two ways to represent structure

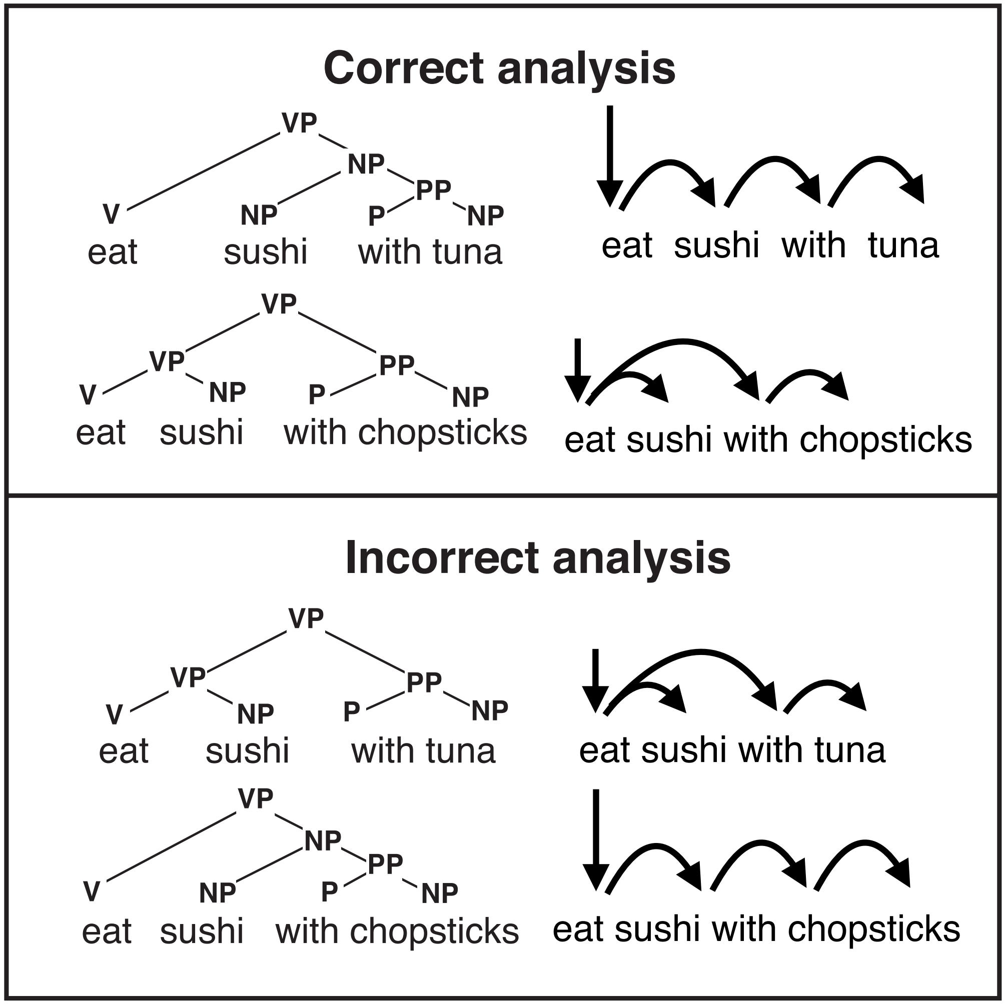
Phrase structure trees



Dependency trees



Structure (syntax) corresponds to meaning (semantics)



Dependency grammar

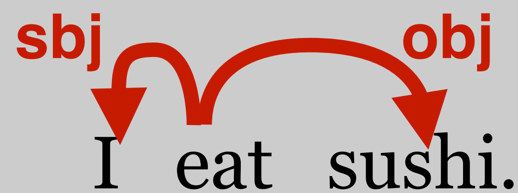
DGs describe the structure of sentences as a **directed acyclic graph**.

The **nodes** of the graph are the **words**

The **edges** of the graph are the **dependencies**.

Edge labels indicate different **dependency types**.

Typically, the graph is assumed to be a **tree**.



Note: the relationship between DG and CFGs:

If a CFG phrase structure tree is translated into DG, the resulting dependency graph has no crossing edges.

Lecture 15:
Introduction to Syntactic Parsing

Part 2:
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, })

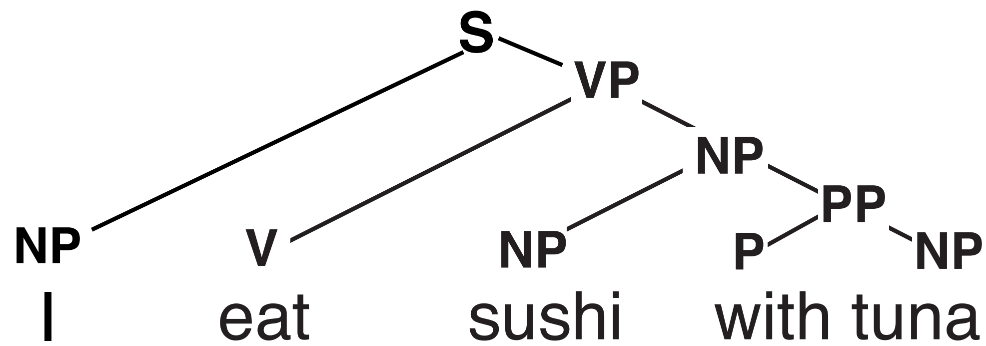
A finite set of **rules** $\mathbf{R}$
$\mathbf{R} \subseteq \{A \rightarrow \beta$ with left-hand-side (LHS) $A \in \mathbf{N}$
and right-hand-side (RHS) $\beta \in (\mathbf{N} \cup \mathbf{\Sigma})^* \}$

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)

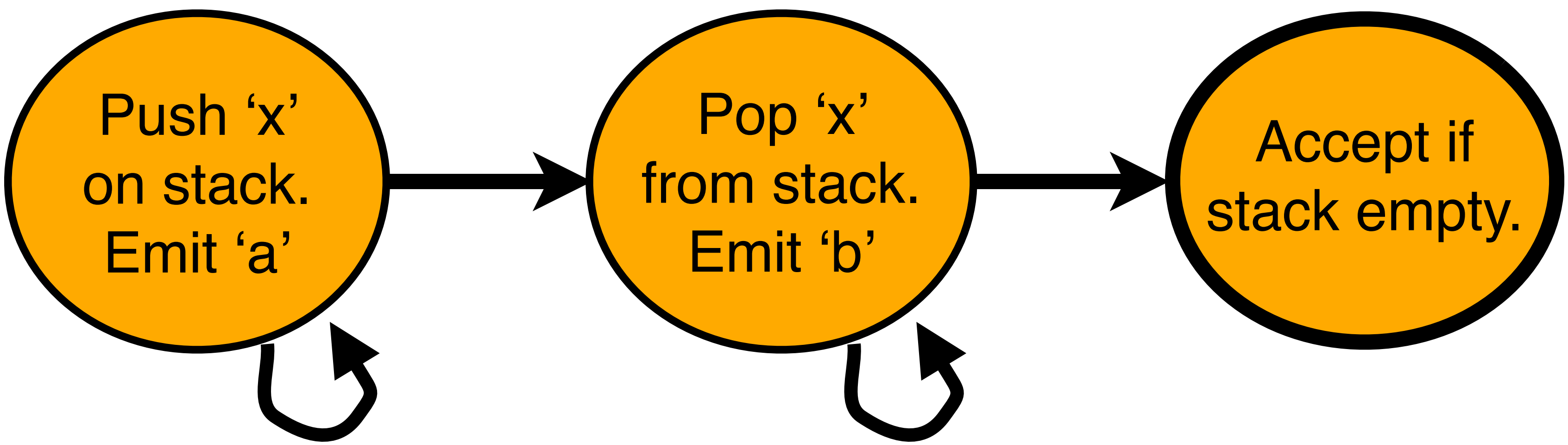
CFGs define **nonterminal categories** (e.g. NP)
to capture **equivalence classes of constituents.**

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

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: 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

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.

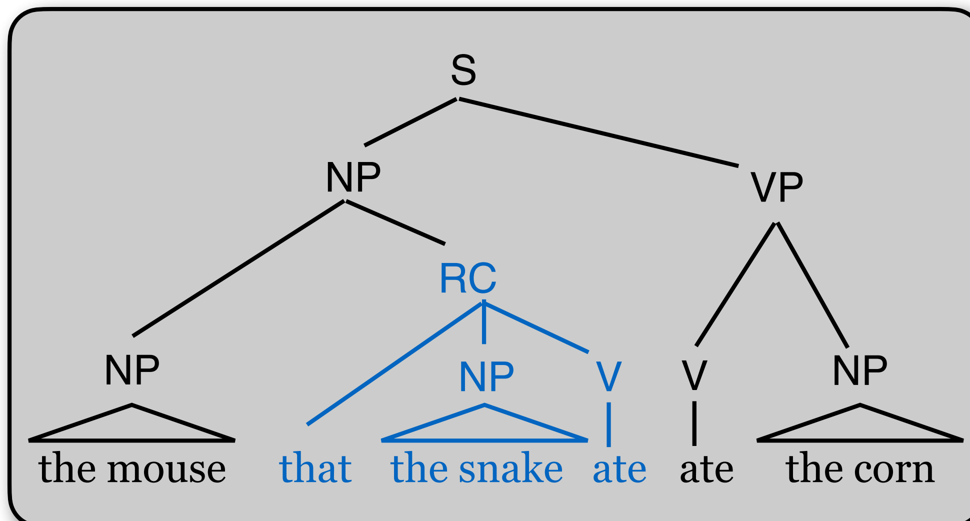
CFGs and unbounded recursion

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

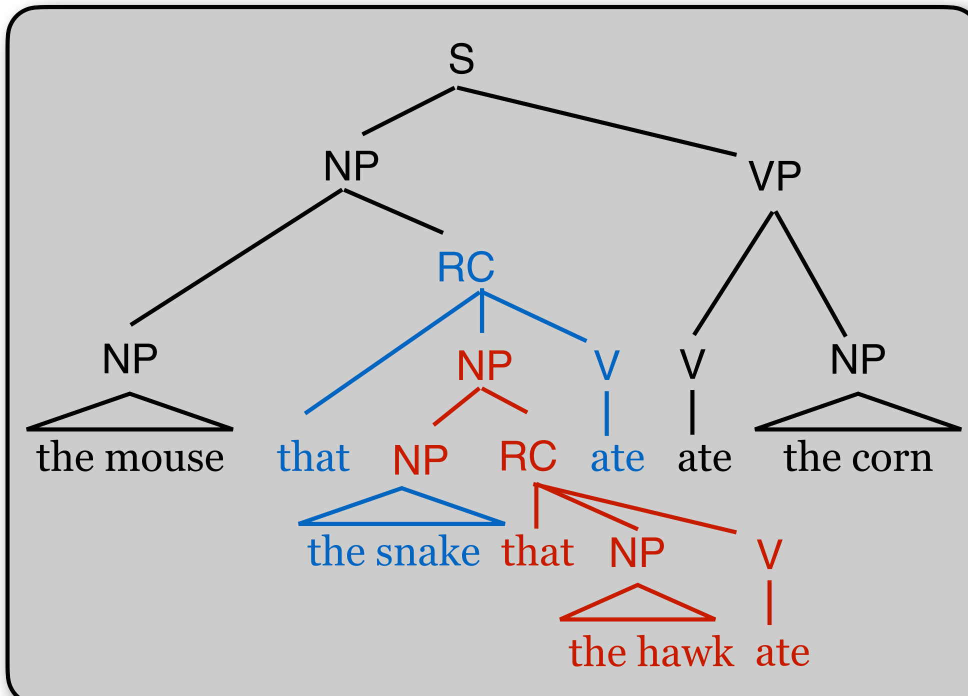
The mouse ate the corn.

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)

Lecture 15:
Introduction to Syntactic Parsing

Part 3:
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]].

$VP \rightarrow V_{have}\ VP_{pastPart}$

$VP \rightarrow V_{be}\ VP_{pass}$

$VP_{pastPart} \rightarrow V_{pastPart}\ NP$

$VP_{pass} \rightarrow V_{pastPart}\ PP$

$V_{have} \rightarrow \{has\}$

$V_{pastPart} \rightarrow \{eaten,\ seen\}$

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]].

VP → V_{comp} S

VP → V_{comp} SBAR

SBAR → COMP S

V_{comp} → {says, think, believes}

COMP → {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?

`WhQ → WhPron Aux VP`$_{pastPart}$

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?

`WhQ → WhPron Aux NP VP`$_{inf}$`-NP`

Lecture 15:
Introduction to Syntactic Parsing

Part 4: The CKY parsing algorithm

CKY chart 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)

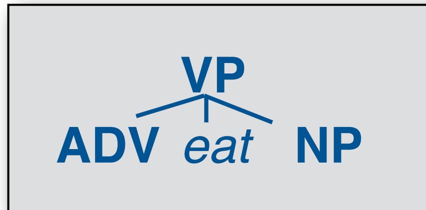
Presumes a CFG in **Chomsky Normal Form**:

Rules are all either **A → B C** (RHS = two nonterminals)
or **A → a** (RHS = a single terminal)
(with **A, B, C** nonterminals and **a** 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:

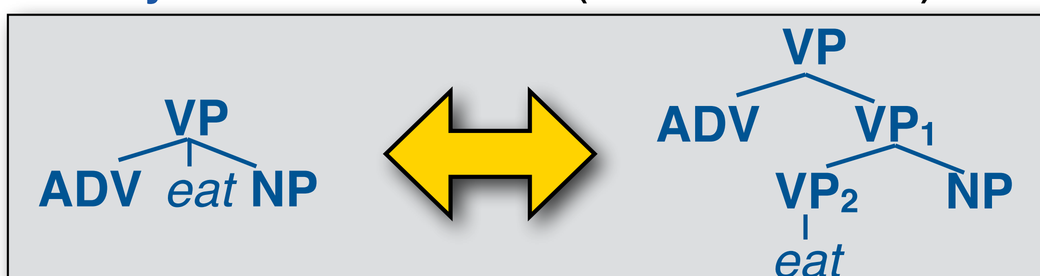
- **Two nonterminals:** VP → ADV VP
- **One terminal:** VP → eat

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 $\mathbf{VP_1}$

$\mathbf{VP_1}$ → $\mathbf{VP_2}$ NP

$\mathbf{VP_2}$ → 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

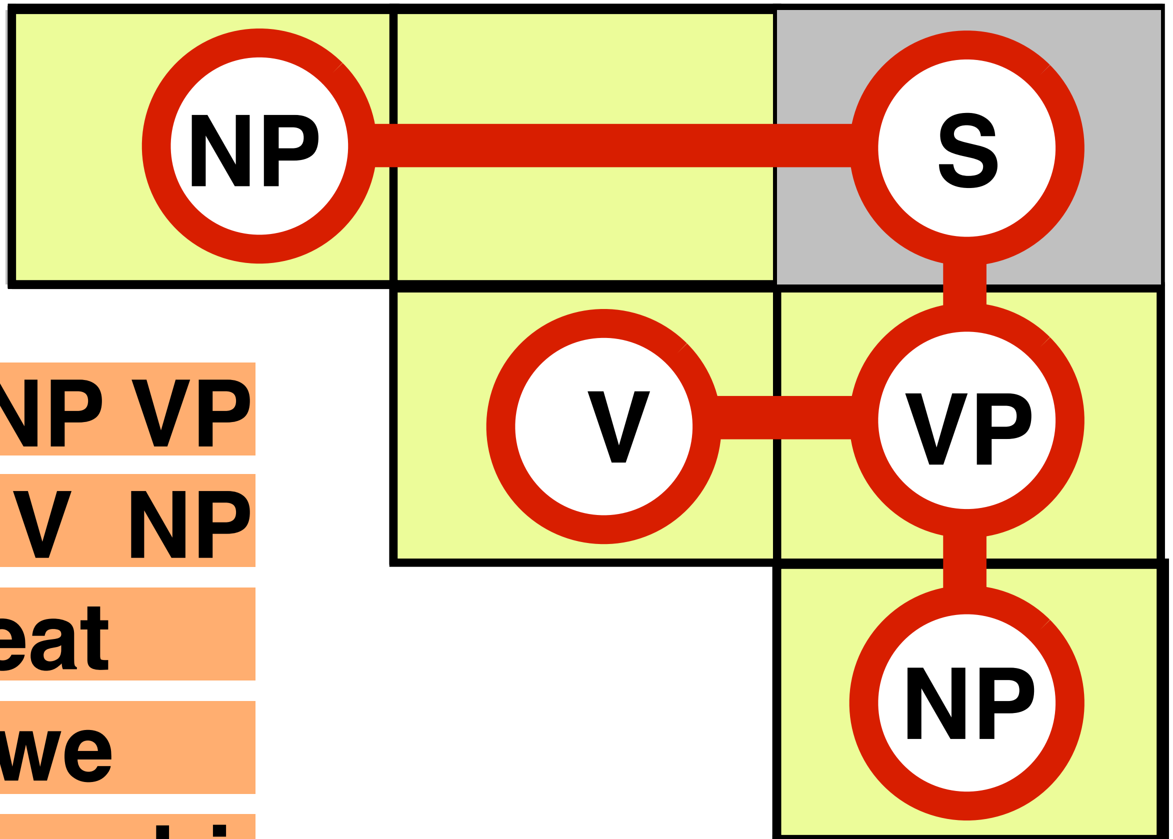
we	we eat	we eat sushi
	eat	eat sushi
		sushi

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

2. Initialize the chart (fill the diagonal cells chart[i][i]):

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

3. Fill in the chart:

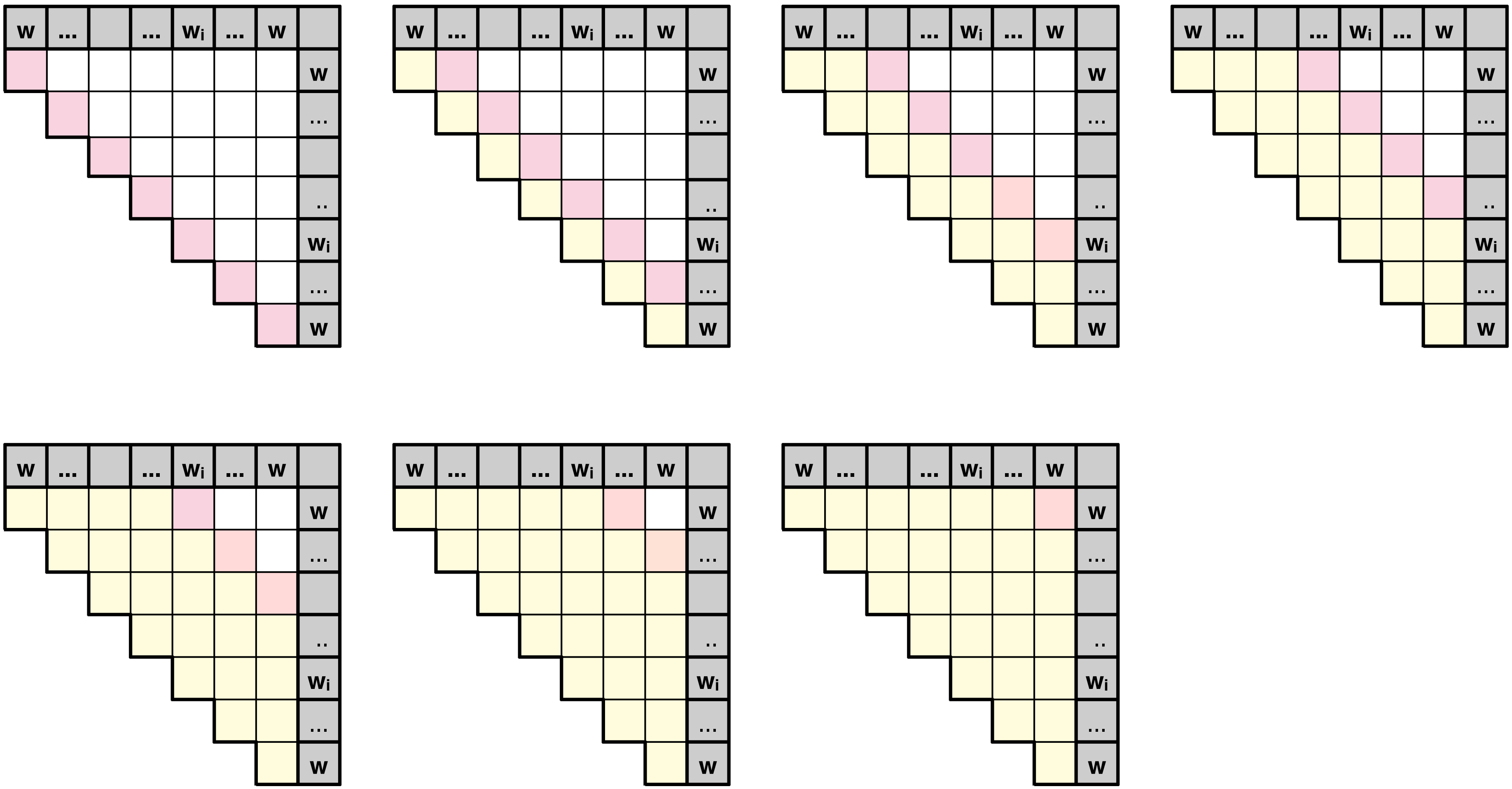
Fill in all cells chart[i][i+1], then chart[i][i+2], …,
until you reach chart[1][n] (the top right corner of the chart)

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

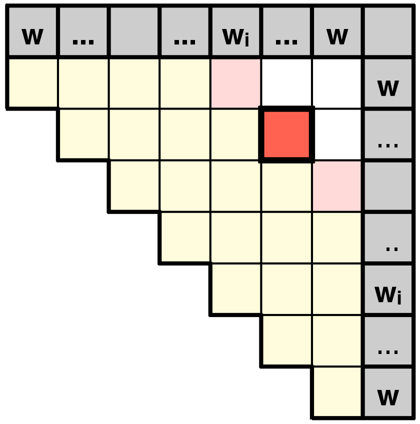
– If the grammar has a rule X → YZ, chart[i][k] contains a Y and chart[k+1][j] contains a Z, add an X to chart[i][j] with two backpointers to the Y in chart[i][k] and the Z in chart[k+1][j]

4. Extract the parse trees from the S in chart[1][n].

CKY: filling the chart



CKY: filling one cell

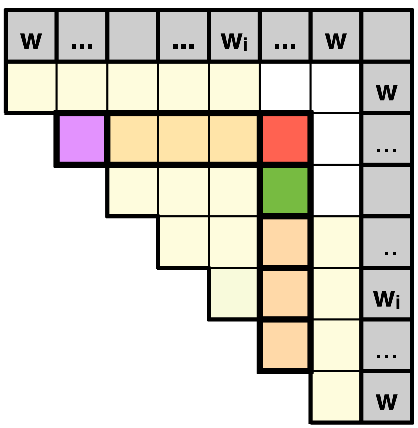


chart[2][6]:

w_1 **w_2 w_3 w_4 w_5 w_6** w_7

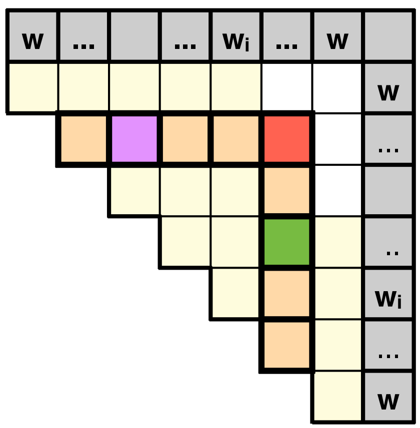
chart[2][6]:

w_1 $w_2 w_3 w_4 w_5 w_6$ w_7



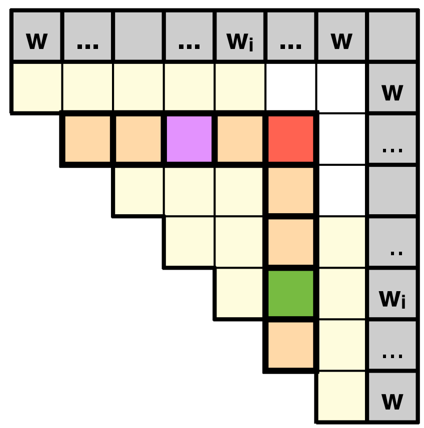
chart[2][6]:

w_1 $w_2 w_3 w_4 w_5 w_6$ w_7



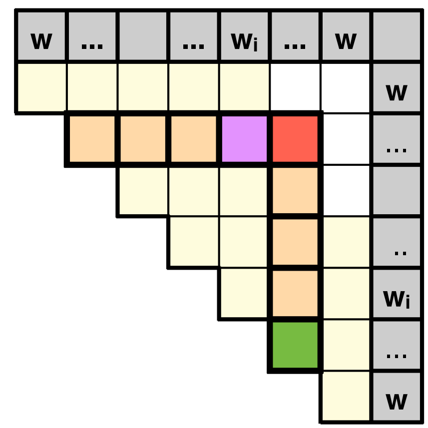
chart[2][6]:

w_1 $w_2 w_3 w_4 w_5 w_6$ w_7



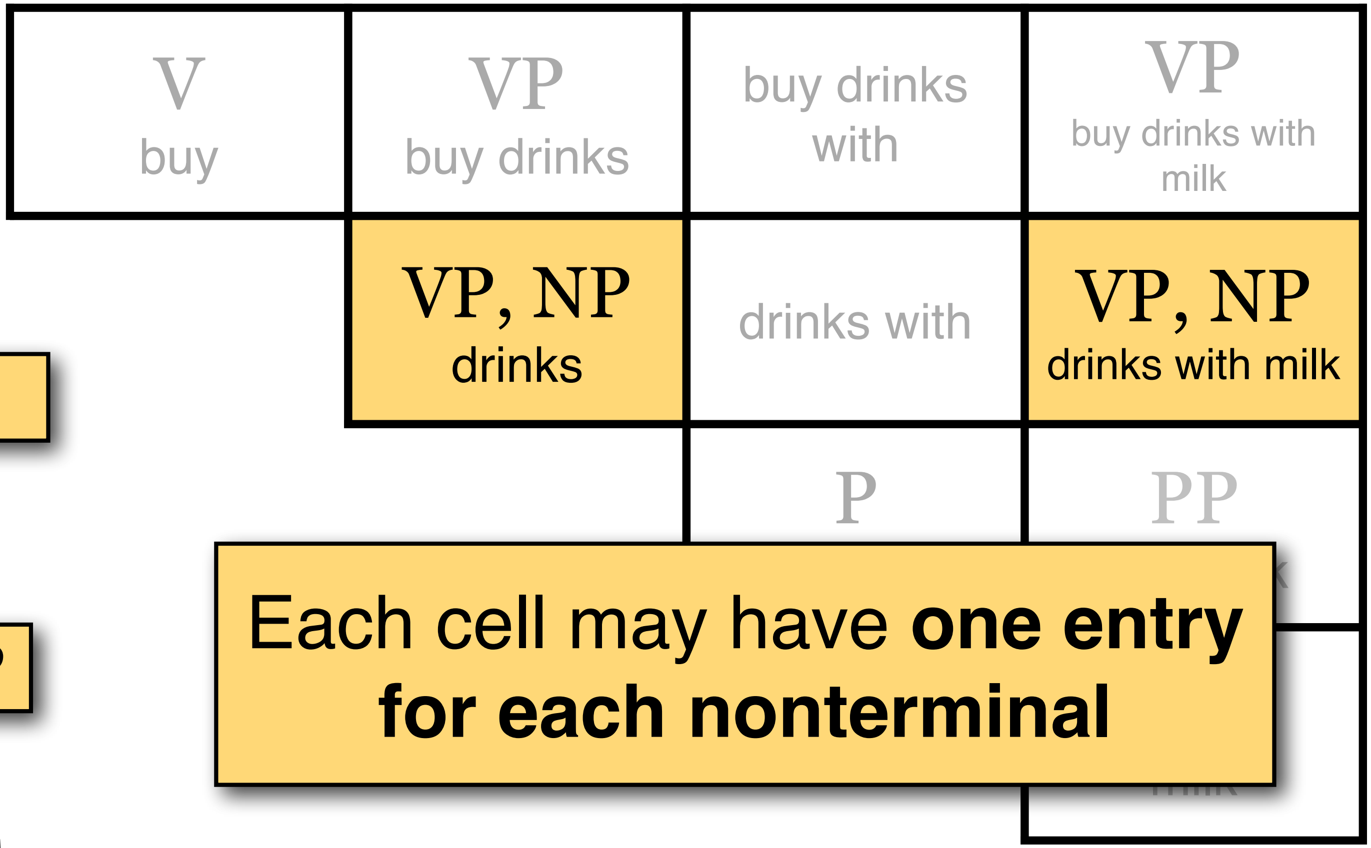
chart[2][6]:

w_1 $w_2 w_3 w_4 w_5 w_6$ 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

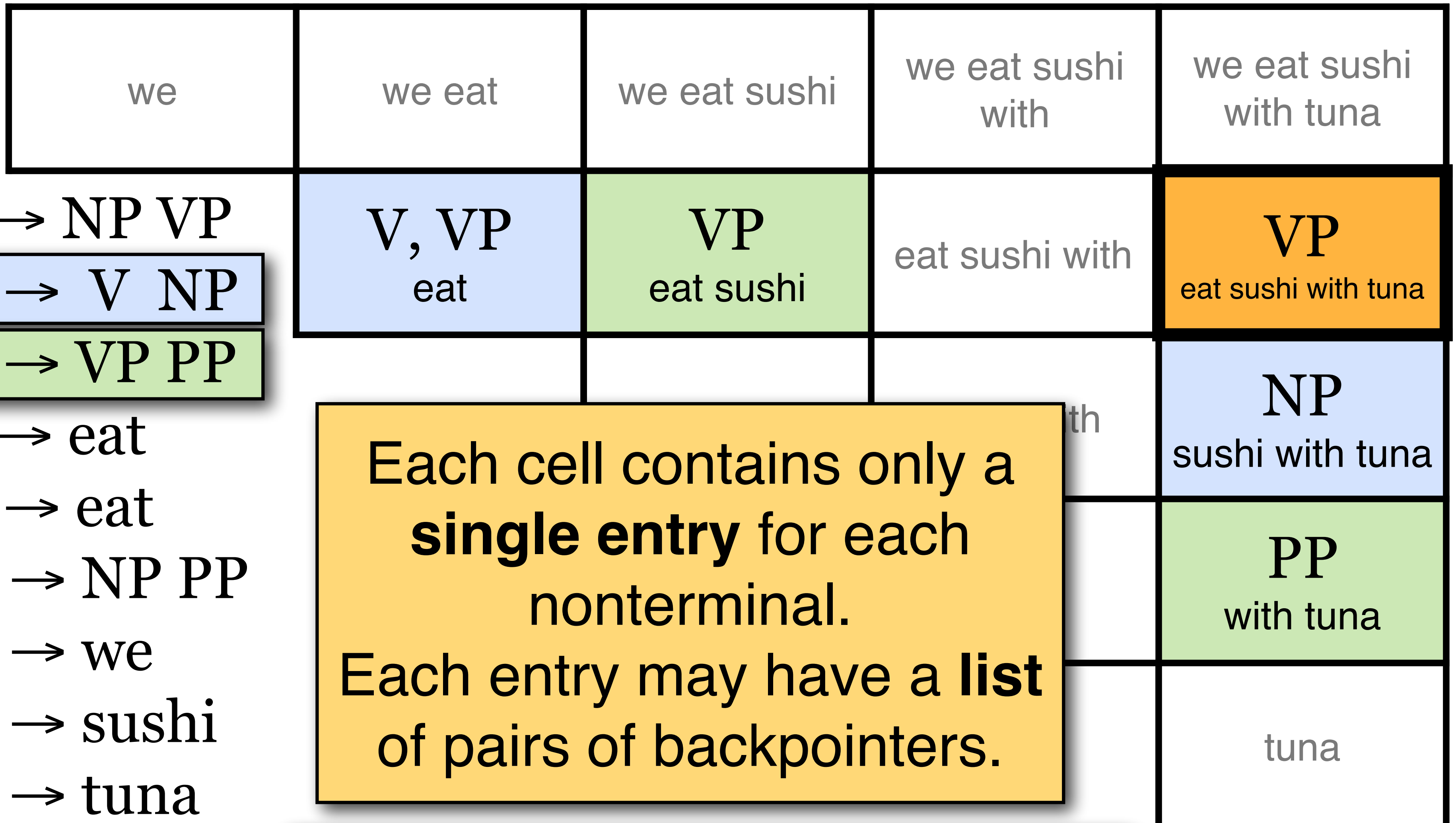


Each cell may have **one entry for each nonterminal**

We buy drinks with milk

The CKY parsing algorithm

S → NP VP
VP → V NP
VP → VP PP
V → eat
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

What are the terminals in NLP?

Are the "terminals": words or POS tags?

For toy examples (e.g. on slides), it's typically the words

With POS-tagged input, we may either treat the POS tags as the terminals, or we assume that the unary rules in our grammar are of the form

POS-tag → word

(so POS tags are the only nonterminals that can be rewritten as words; some people call POS tags "preterminals")

Additional unary rules

In practice, we may allow other unary rules, e.g.

NP → Noun

(where Noun is also a nonterminal)

In that case, we apply all unary rules to the entries in chart[i][j] *after* we've checked all binary splits (chart[i][k], chart[k+1][j])

Unary rules are fine as long as there are **no "loops"** that lead to an infinite chain of unary productions, e.g.:

X → Y and Y → **X**

or: **X** → Y and Y → Z and Z → **X**

CKY so far…

Each entry in a cell chart[i][j] is associated with a nonterminal X.

If there is a rule X → YZ in the grammar,
and there is a pair of cells chart[i][k], chart[k+1][j]
with a Y in chart[i][k] and a Z in chart[k+1][j],
we can add an entry X to cell chart[i][j], and associate one pair of backpointers with the X in cell chart[i][k]

Each entry might have multiple pairs of backpointers.

When we extract the parse trees at the end,
we can get **all possible trees**.

We will need probabilities to find the single best tree!

Exercise: CKY parser

I eat sushi with chopsticks with you

S	$\longrightarrow$	NP	VP
NP	$\longrightarrow$	NP	PP
NP	$\longrightarrow$	*sushi*	
NP	$\longrightarrow$	*I*	
NP	$\longrightarrow$	*chopsticks*	
NP	$\longrightarrow$	*you*	
VP	$\longrightarrow$	VP	PP
VP	$\longrightarrow$	Verb	NP
Verb	$\longrightarrow$	*eat*	
PP	$\longrightarrow$	Prep	NP
Prep	$\longrightarrow$	*with*	

How do you count the **number of parse trees** for a sentence?

1. For each **pair of backpointers** (e.g. $VP \rightarrow V\ NP$): **multiply** #trees of children

$$\text{trees}(VP_{VP \rightarrow V\ NP}) = \text{trees}(V) \times \text{trees}(NP)$$

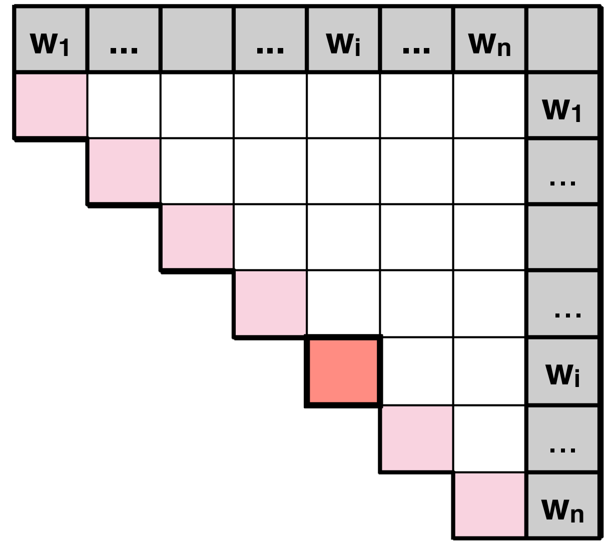
2. For each **list of pairs of backpointers** (e.g. $VP \rightarrow V\ NP$ and $VP \rightarrow VP\ PP$): **sum** #trees

$$\text{trees}(VP) = \text{trees}(VP_{VP \rightarrow V\ NP}) + \text{trees}(VP_{VP \rightarrow VP\ PP})$$

Cocke Kasami Younger

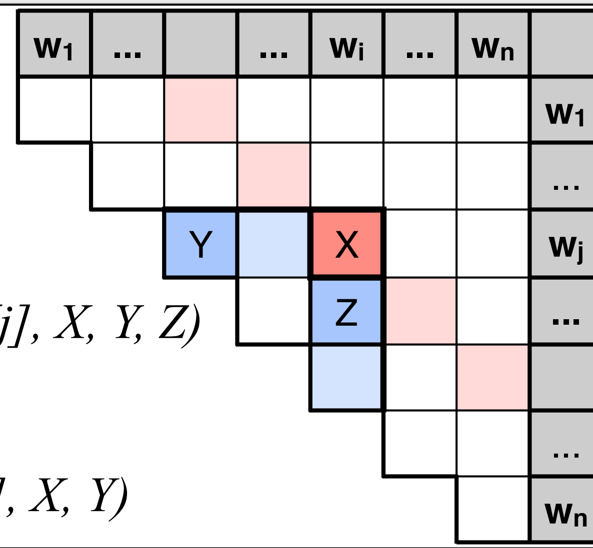
```
ckyParse(n):
    initChart(n)
    fillChart(n)
```

```
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