CS447: Natural Language Processing

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

Lecture 17: Formal Grammars of English

Julia Hockenmaier
juliahmr@illinois.edu
3324 Siebel Center

Previous key concepts

NLP tasks dealing with words...

- POS-tagging, morphological analysis

… require 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: Forward, Viterbi, Forward-Backward

The next key concepts

NLP tasks dealing with sentences...

- Syntactic parsing and semantic analysis

… require (at least) context-free representations,

- Context-free grammars, unification grammars

… the corresponding probabilistic models,

- Probabilistic Context-Free Grammars, Loglinear models
- Estimation: Relative Frequency estimation, EM algorithm, etc.

… and appropriate search algorithms

- Dynamic programming: chart parsing, inside-outside algorithm

Dealing with ambiguity

Search Algorithm
(e.g Viterbi)

Structural Representation
(e.g FSA)

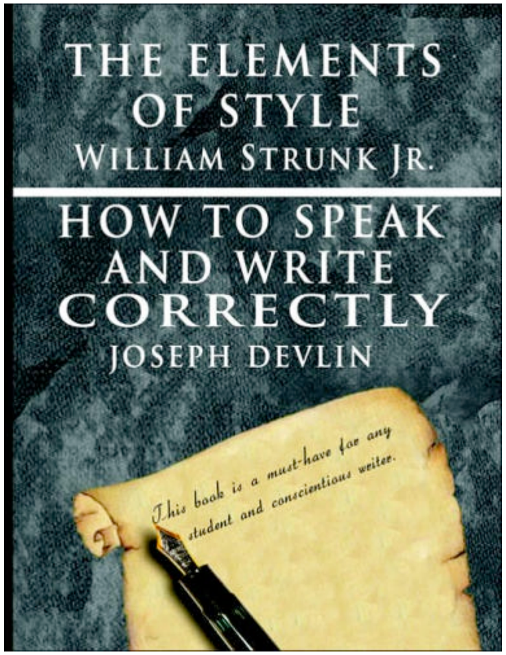
Scoring Function
(Probability model, e.g HMM)

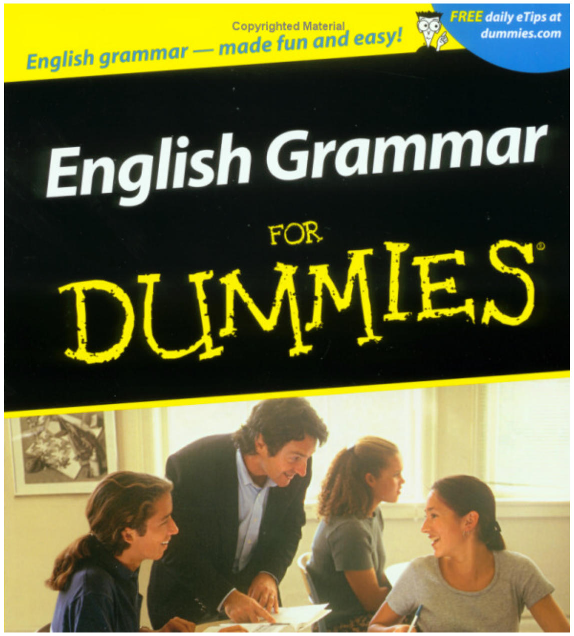
Today's lecture

Introduction to natural language syntax ('grammar'):

Constituency and dependencies
Context-free Grammars
Dependency Grammars
A simple CFG for English

What is grammar?





No, not really, not in this class

What is grammar?

Grammar formalisms
(= linguists' programming languages)

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

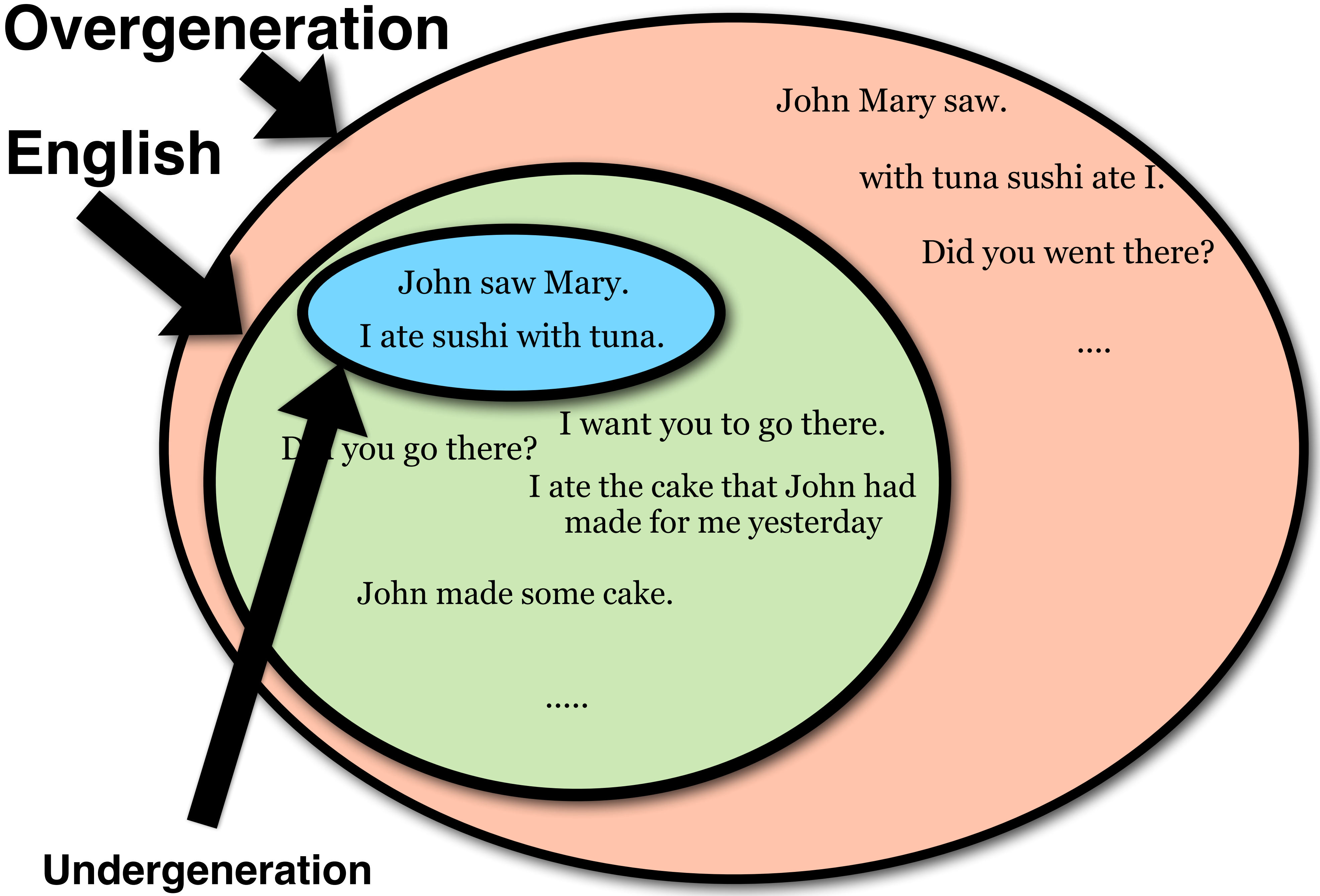
(N.B.: There are many different formalisms out there, which each define their own data structures and operations)

Specific grammars
(= linguists' programs)

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

Can we define a program that generates all English sentences?

The number of sentences is infinite.
But we need our program to be finite.



Overgeneration

English

Undergeneration

John Mary saw.

with tuna sushi ate I.

Did you went there?

....

John saw Mary.

I ate sushi with tuna.

Did you go there?

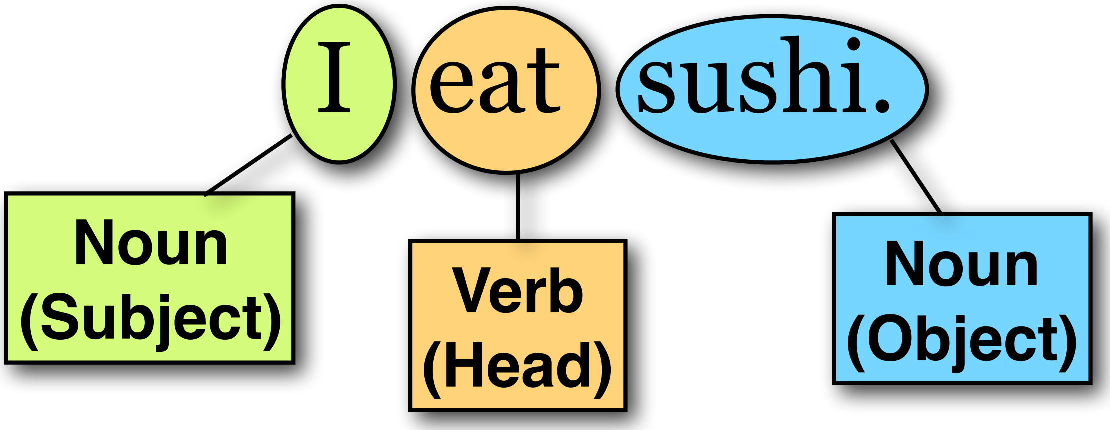
I want you to go there.

I ate the cake that John had made for me yesterday

John made some cake.

.....

Basic sentence structure



I eat sushi.

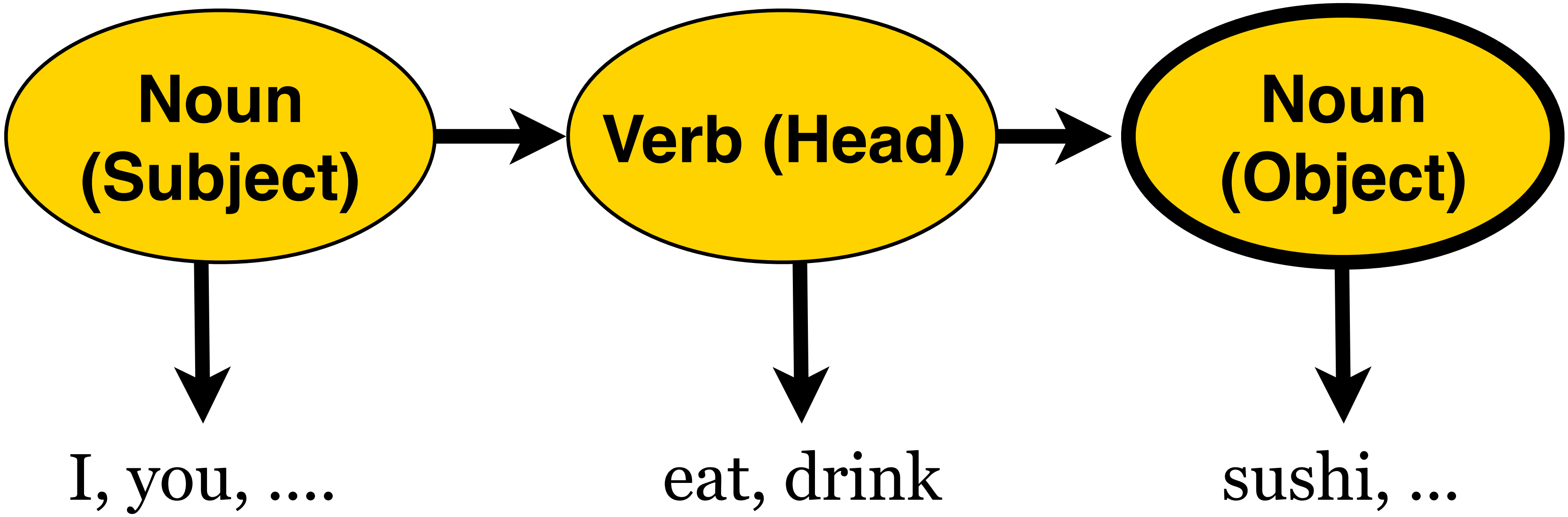
Noun (Subject)

Verb (Head)

Noun (Object)

A finite-state-automaton (FSA)

A Hidden Markov Model (HMM)



Noun (Subject)

Verb (Head)

Noun (Object)

I, you, ...

eat, drink

sushi, ...

Words take arguments

I eat sushi. ✓
I eat sushi you. ???
I sleep sushi ???
I give sushi ???
I drink sushi ?

Subcategorization

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

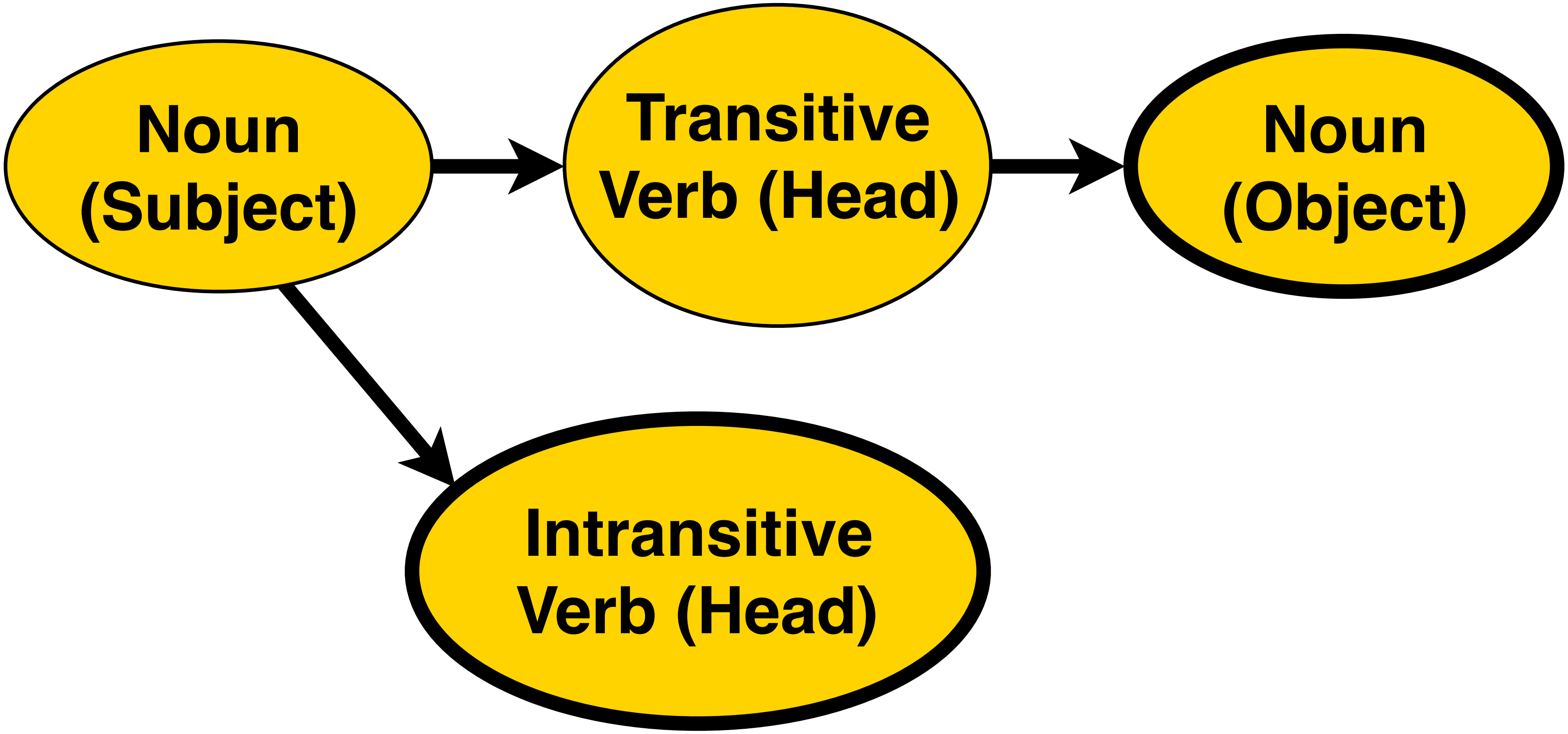
Intransitive verbs (sleep) take only a subject.
Transitive verbs (eat) take also one (direct) object.
Ditransitive verbs (give) take also one (indirect) object.

Selectional preferences

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

The object of eat should be edible.

A better FSA

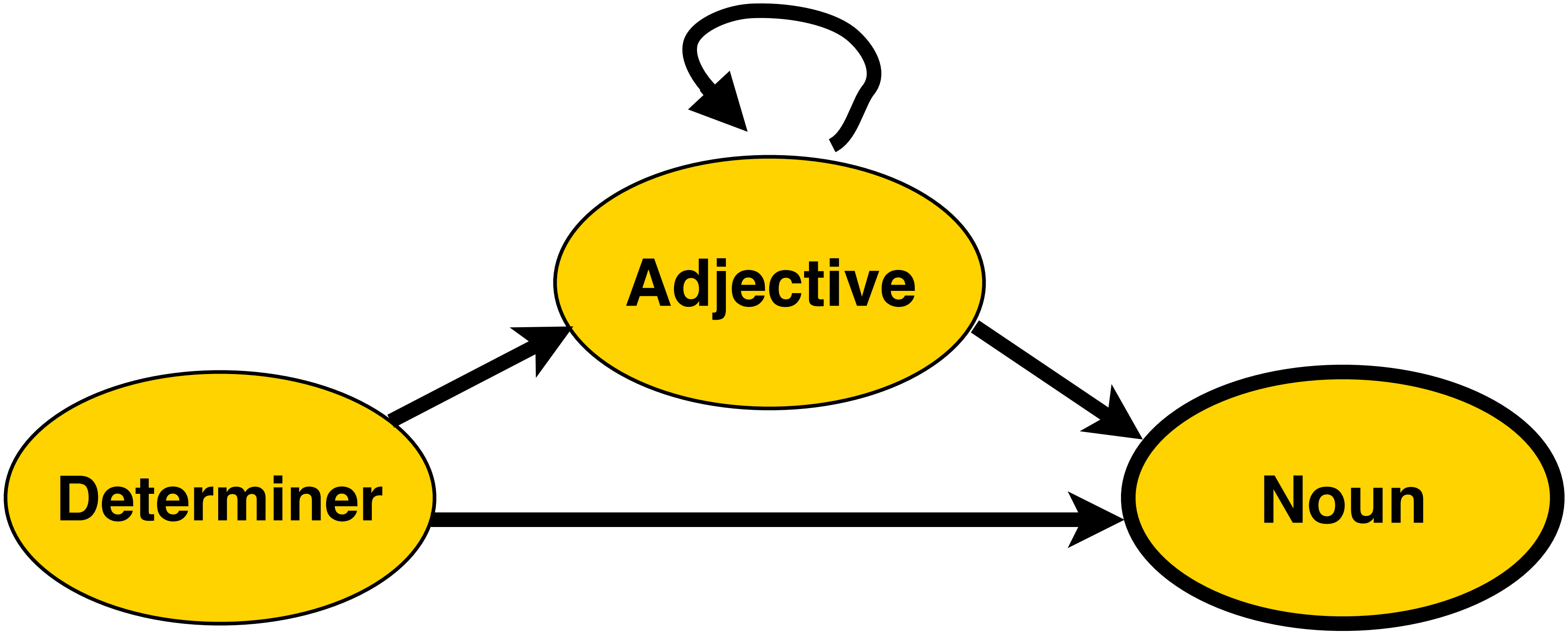

Noun (Subject)
Transitive Verb (Head)
Noun (Object)
Intransitive Verb (Head)

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**.

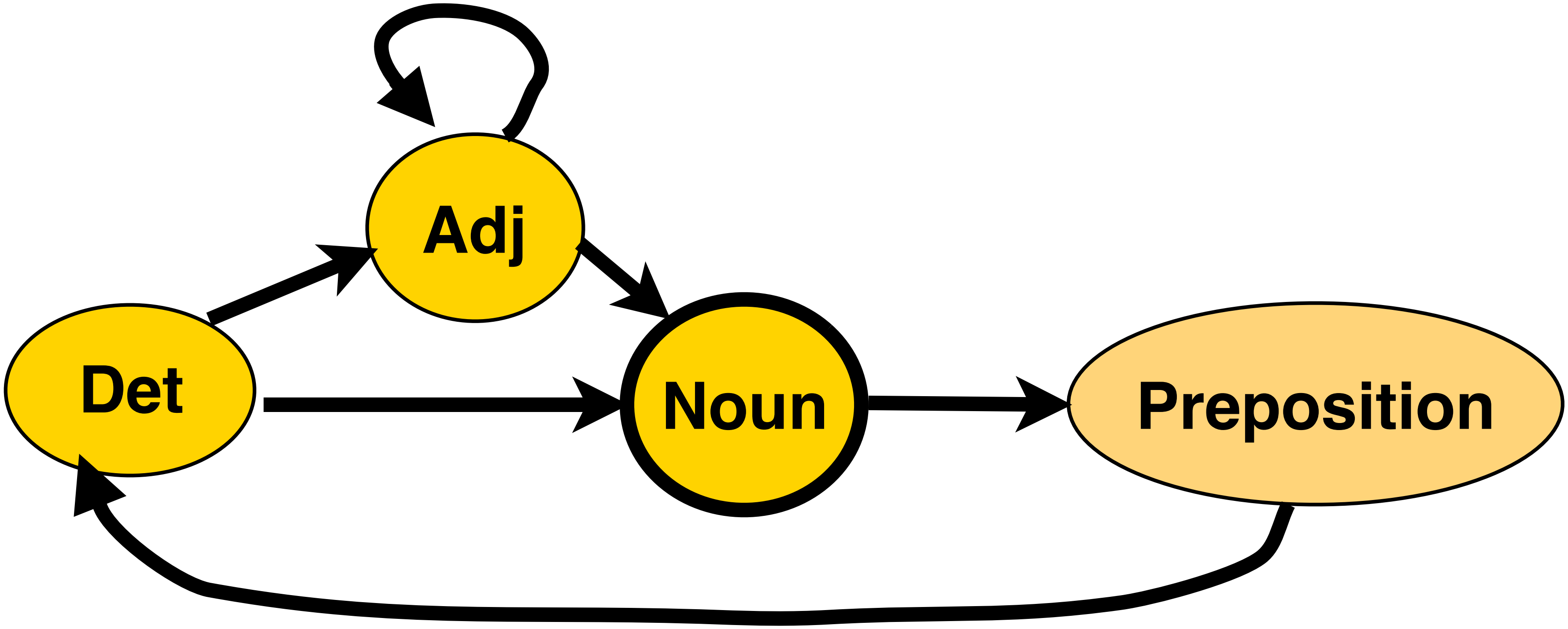
Another FSA



Recursion can be more complex

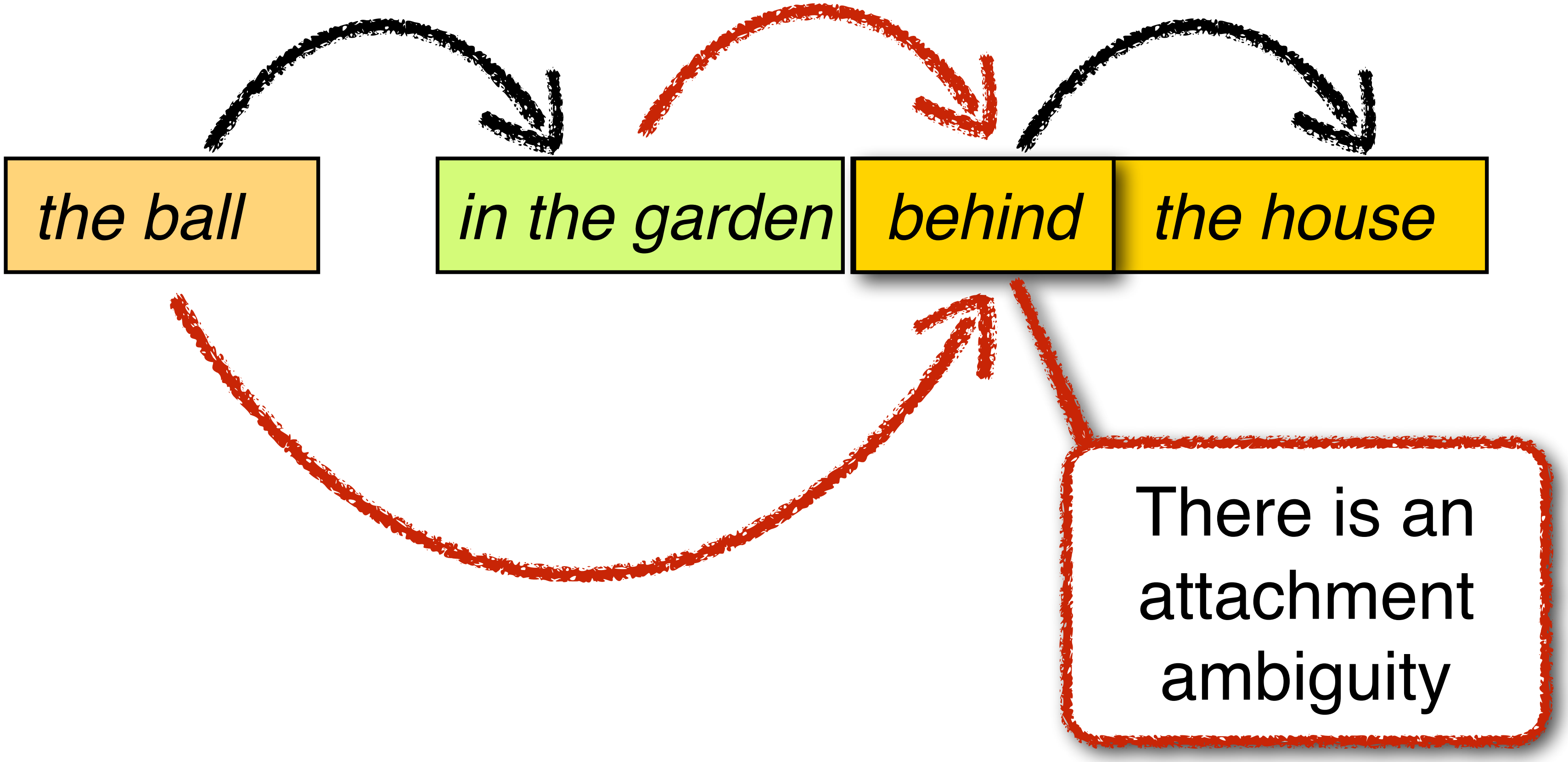
the ball
the ball in the garden
the ball in the garden behind the house
the ball in the garden behind the house next to the school
....

Yet another FSA



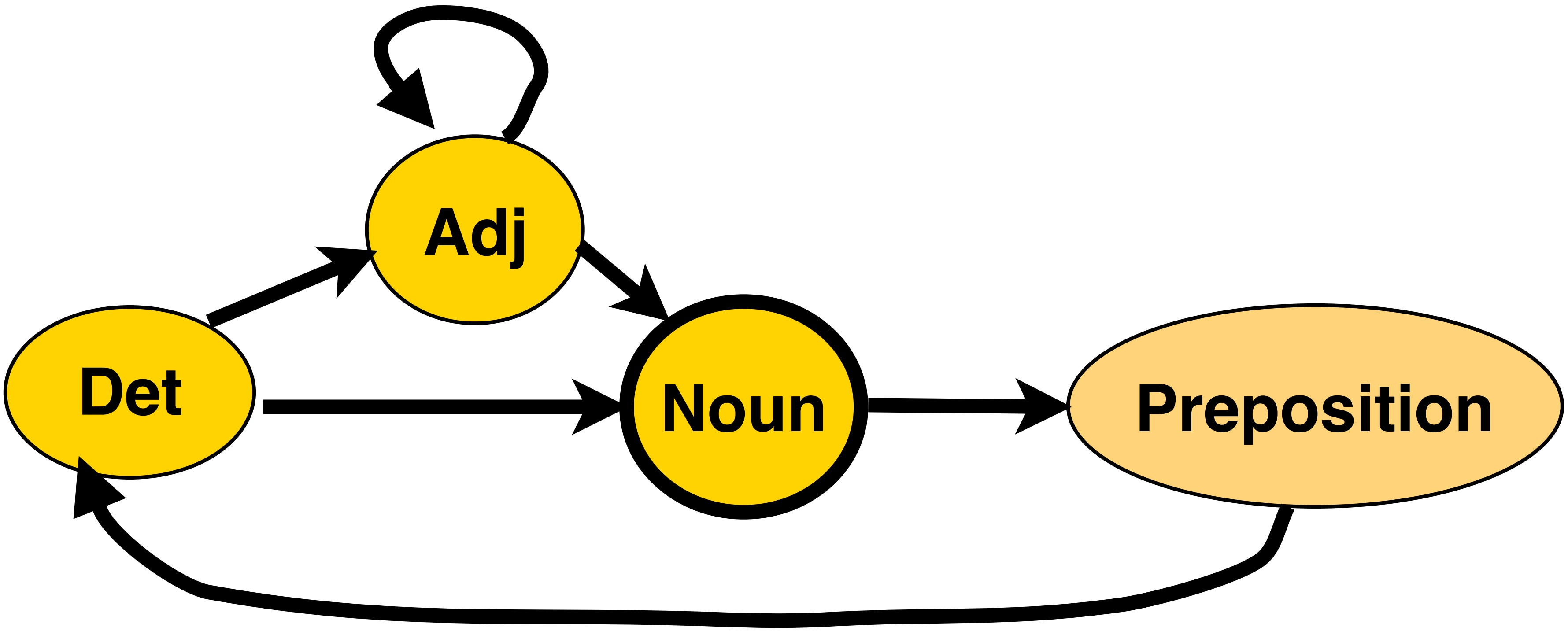
So, why do we need anything
beyond regular (finite-state) grammars?

What does this mean?



There is an attachment ambiguity

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):

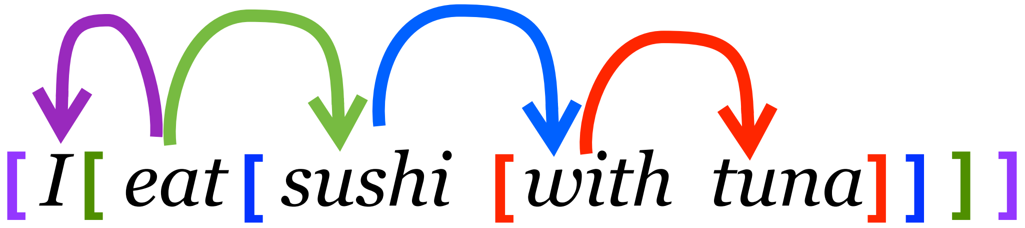
- defines language as sets of strings with (hidden) structure
- is also concerned with generating the right *structures* (*strong* generative capacity)

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”

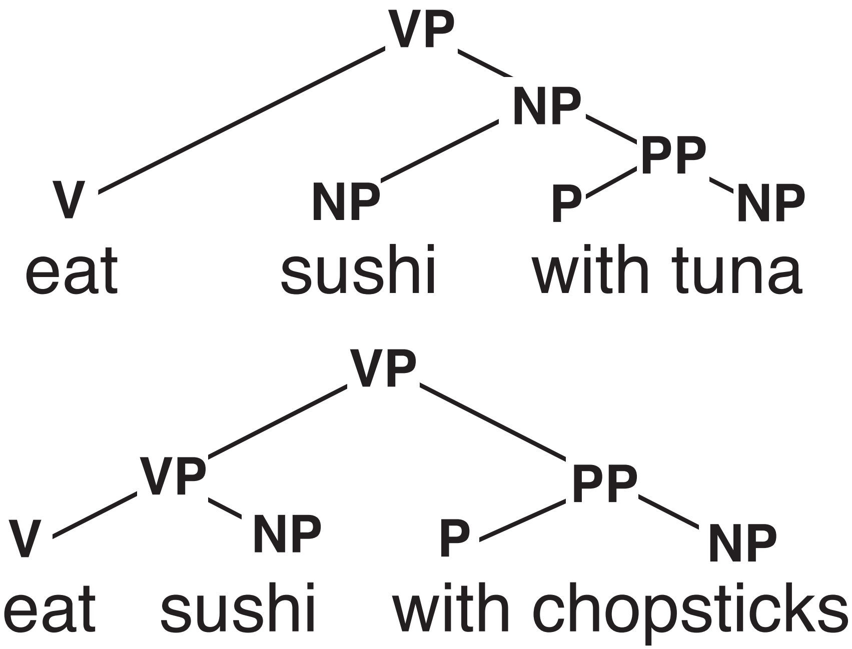
Sentence structure defines **dependencies** between words or phrases:



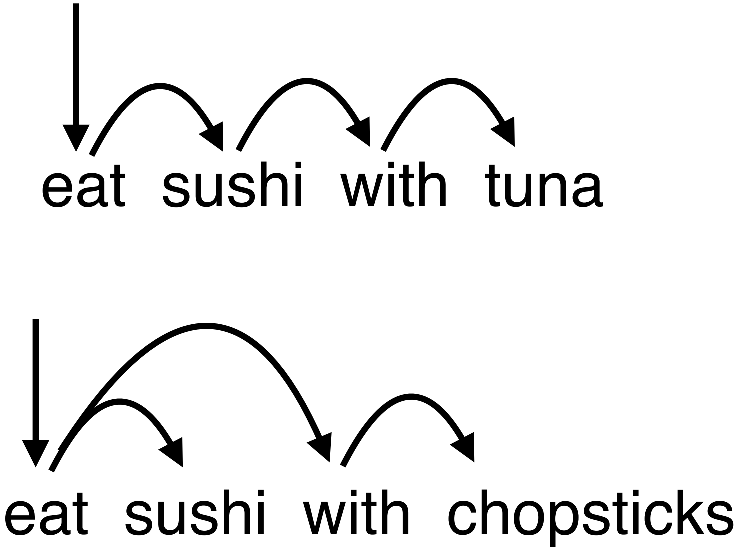
[*I* [*eat* [*sushi* [*with tuna*]]]]

Two ways to represent structure

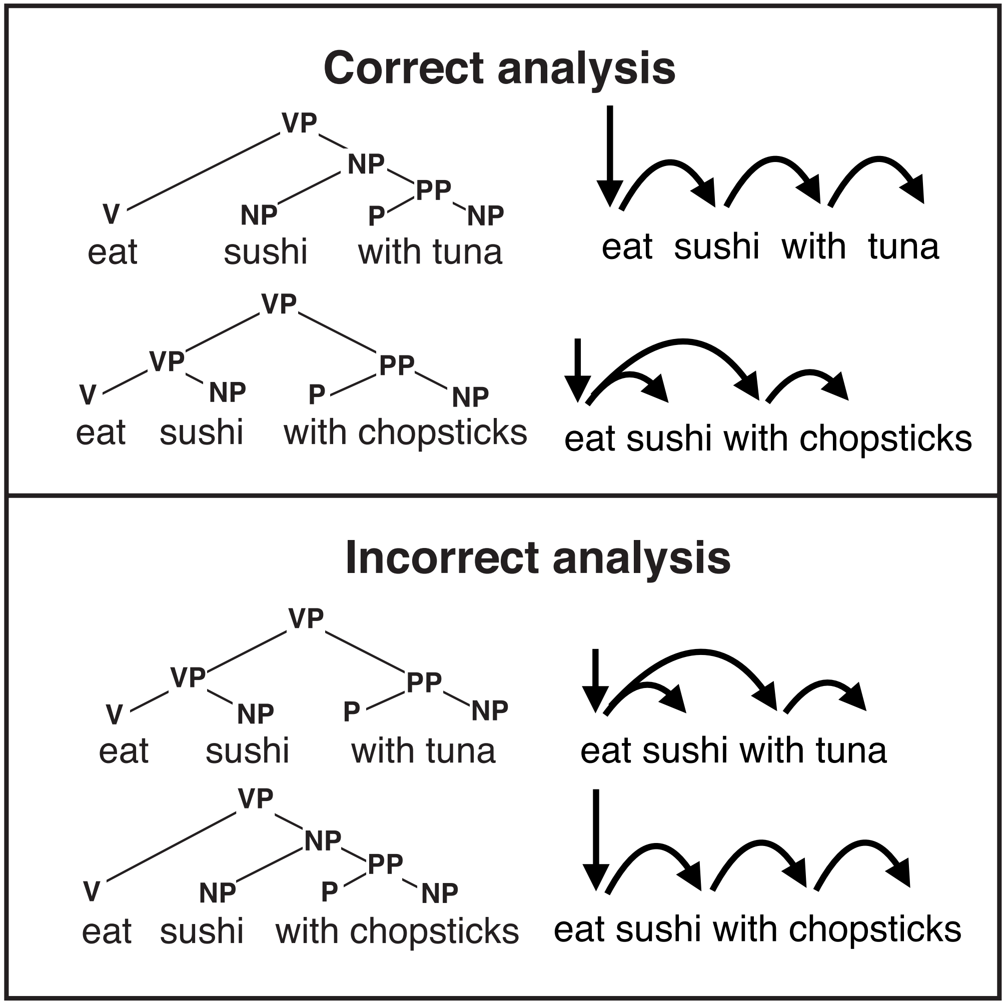
Phrase structure trees



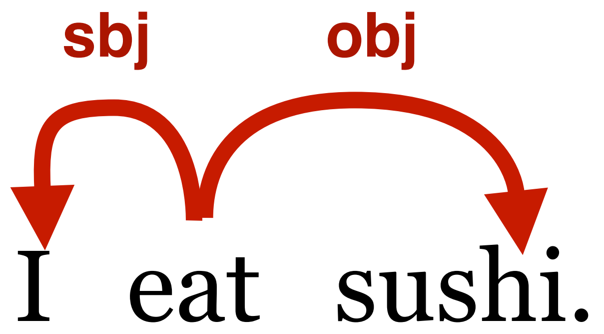
Dependency trees



Structure (syntax) corresponds to meaning (semantics)

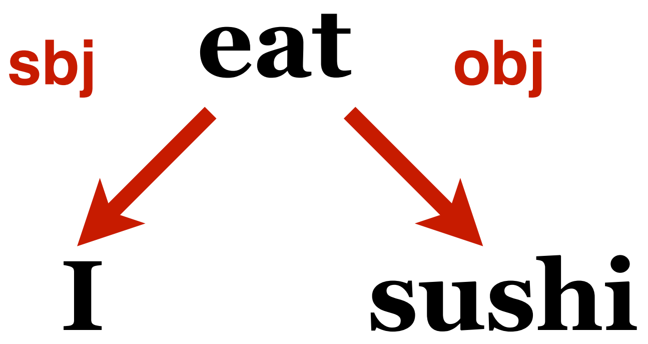


This is a dependency tree:



sbj obj

I eat sushi.



sbj **eat** obj

I **sushi**

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**.

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.

Context-free grammars

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

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

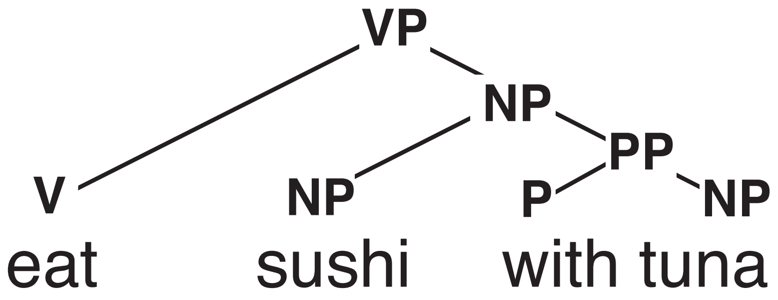
A set of **terminals** $\mathbf{\Sigma}$
(e.g. $\mathbf{\Sigma}$ = {I, you, he, eat, drink, sushi, ball, })

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

Context-free grammars (CFGs) define phrase structure trees

DT → {the, a}
N → {ball, garden, house, sushi }
P → {in, behind, with}
NP → DT N
NP → NP PP
PP → P NP



N: noun
P: preposition
NP: “noun phrase”
PP: “prepositional phrase”

Context-free grammars (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 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.
....

CFGs and center embedding

Formally, these sentences are all grammatical, because they can be generated by the CFG that is required for the first sentence:

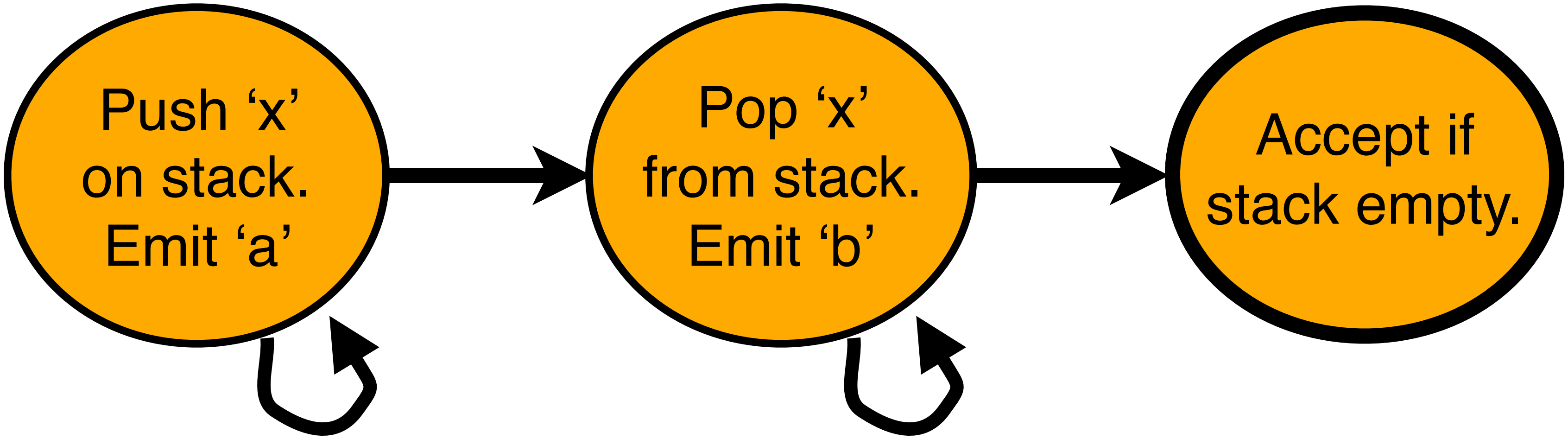
S → NP VP
NP → NP RelClause
RelClause → that NP ate

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 model predicts to be grammatical, and what humans consider grammatical, linguists distinguish between a speaker’s **competence** (grammatical knowledge) and **performance** (processing and memory limitations)

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


Push ‘x’ on stack. Emit ‘a’

Pop ‘x’ from stack. Emit ‘b’

Accept if stack empty.

This is equivalent to the following CFG:

S → a X b S → a b
X → a X b X → 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

Defining grammars for natural language

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 man, a girl with glasses, Illinois
Prepositional phrases: with glasses, in the garden
Verb phrases: eat sushi, sleep, sleep soundly

The other parts are its **dependents**.

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

Is string α a constituent?

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].

Arguments are obligatory

Words subcategorize for specific sets of arguments:

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

All arguments have to be present:

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

No argument 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 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

Heads:

We assume that each RHS has one head, 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: Coordination, lists: VP → VP conj VP

Arguments:

The head has a different category from the parent:

VP → Verb NP (the NP is an argument of the verb)

Adjuncts:

The head has the same category as the parent:

VP → VP PP (the PP is an adjunct)

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)

Complex NPs:

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

Det → {a, the, every}
Pronoun → {he, she,...}
ProperName → {John, Mary,...}
Noun → AdjP Noun
Noun → N
NP → NP PP
NP → NP RelClause

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 [eats sushi with chopsticks].

VP → V
VP → V NP
VP → V NP PP
VP → VP PP

V → {eats, sleeps gives,...}

Capturing subcategorization

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

$\mathrm{VP} \rightarrow \mathrm{V}_{\mathrm{intrans}}$
$\mathrm{VP} \rightarrow \mathrm{V}_{\mathrm{trans}}\ \mathrm{NP}$
$\mathrm{VP} \rightarrow \mathrm{V}_{\mathrm{ditrans}}\ \mathrm{NP}\ \mathrm{NP}$
$\mathrm{VP} \rightarrow \mathrm{VP}\ \mathrm{PP}$
$\mathrm{V}_{\mathrm{intrans}} \rightarrow \{\mathrm{eats}, \mathrm{sleeps}\}$
$\mathrm{V}_{\mathrm{trans}} \rightarrow \{\mathrm{eats}\}$
$\mathrm{V}_{\mathrm{trans}} \rightarrow \{\mathrm{gives}\}$

Sentences

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

S → NP VP
S → AdvP S
S → PP S

He says [he eats sushi].
VP → Vcomp S
Vcomp → {says, think, believes}

Sentences redefined

[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 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: the girl will have eaten sushi
passive voice: the sushi was eaten by the girl
progressive: the girl is/was/will be eating sushi

VPs redefined

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

$\mathrm{VP} \rightarrow \mathrm{V}_{\mathrm{have}}\ \mathrm{VP}_{\mathrm{pastPart}}$
$\mathrm{VP} \rightarrow \mathrm{V}_{\mathrm{be}}\ \mathrm{VP}_{\mathrm{pass}}$
$\mathrm{VP}_{\mathrm{pastPart}} \rightarrow \mathrm{V}_{\mathrm{pastPart}}\ \mathrm{NP}$
$\mathrm{VP}_{\mathrm{pass}} \rightarrow \mathrm{V}_{\mathrm{pastPart}}\ \mathrm{PP}$
$\mathrm{V}_{\mathrm{have}} \rightarrow \{\mathrm{has}\}$
$\mathrm{V}_{\mathrm{pastPart}} \rightarrow \{\mathrm{eaten, seen}\}$

We need more nonterminals (e.g. $\mathrm{VP}_{\mathrm{pastpart}}$).
N.B.: We call $\mathrm{VP}_{\mathrm{pastPart}}$, $\mathrm{VP}_{\mathrm{pass}}$, etc. \`untensed’ VPs

Coordination

[He eats sushi] and [she drinks tea]
[John] and [Mary] eat sushi.
He [eats sushi] and [drinks tea]

S → S conj S
NP → NP conj NP
VP → VP conj VP

He says [he eats sushi].
VP → V_{comp} S
V_{comp} → {says, think, believes}

Relative clauses

Relative clauses modify a noun phrase:
the girl [that eats sushi]

Relative clauses lack a noun phrase, which is understood to be filled by the NP they modify:
'the girl that eats sushi' implies 'the girl eats sushi'

There are subject and object relative clauses:
subject: 'the girl that eats sushi'
object: 'the sushi that the girl eats'

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?

$\text{YesNoQ} \rightarrow \text{Aux NP VP}_{\text{inf}}$
$\text{YesNoQ} \rightarrow \text{Aux NP VP}_{\text{pastPart}}$

Wh-questions

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

Who has eaten the sushi?

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

What does Mary eat?

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:

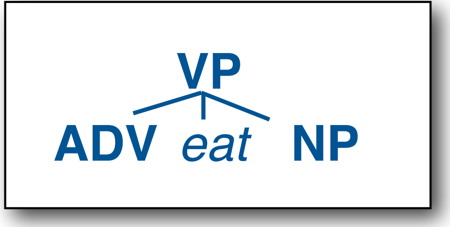
Rules are all either **A → B C** or **A → a**

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

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

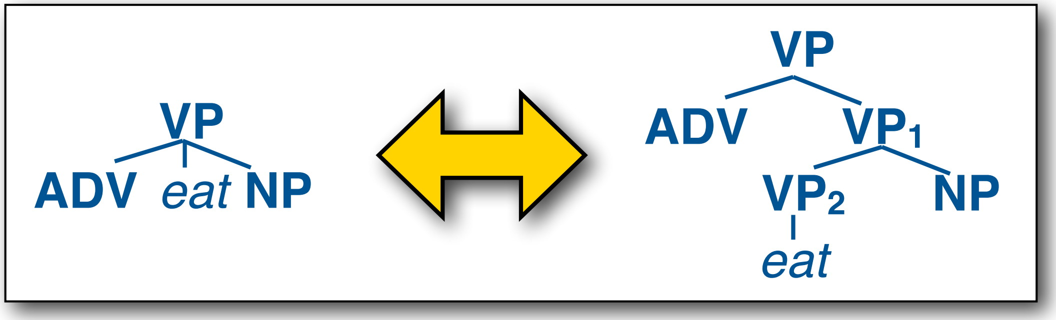


A CFG in **Chomsky Normal Form** (CNF) allows only two kinds of right-hand sides:

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

Any CFG can be transformed into an equivalent CNF:

VP → ADVP **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

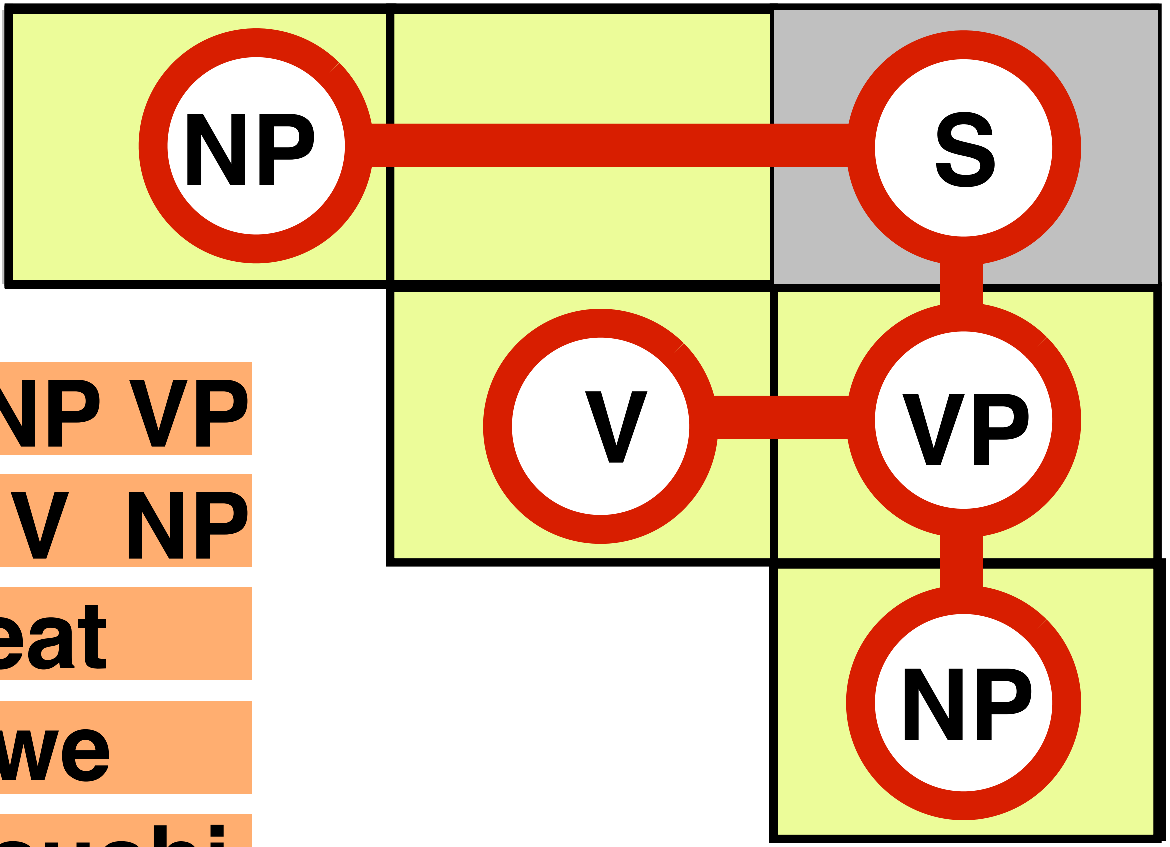
VP → V NP VP → V ε NP → DT Noun

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



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*×*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 $X \rightarrow w^{(i)}$, add an entry X to chart[i][i]

3. Fill in the chart:

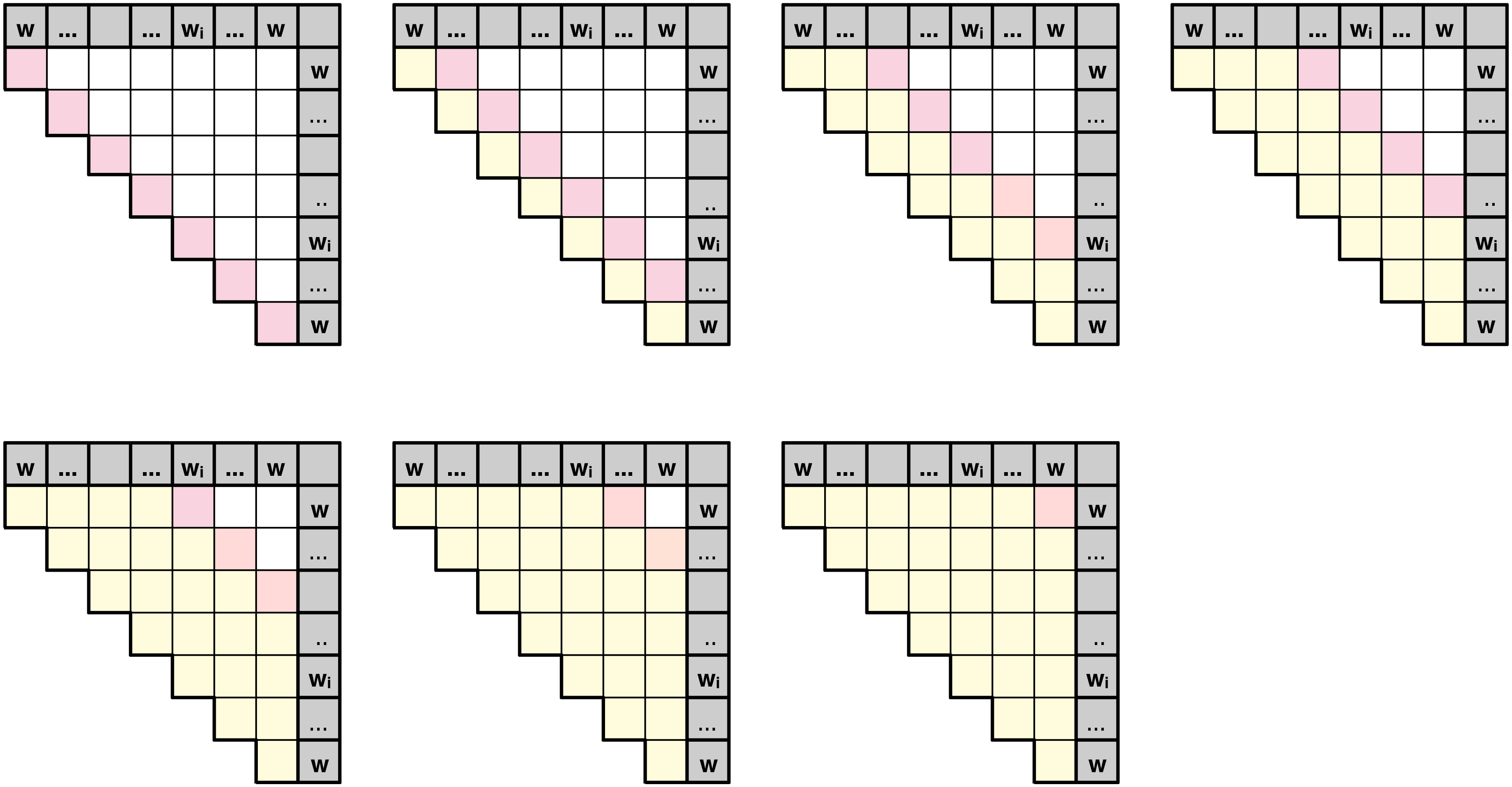
Fill in all cells chart[i][i+1], then chart[i][i+2], …,
until you reach chart[1][n] (the top right corner of the chart)

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

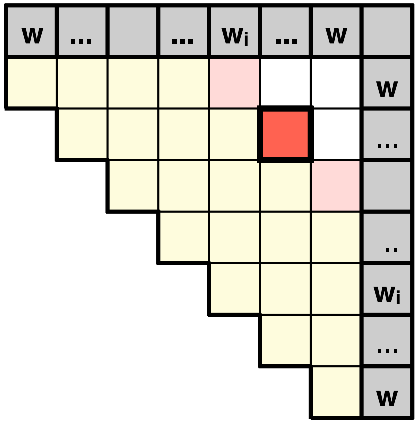
– If the grammar has a rule $X \rightarrow YZ$, chart[i][k] contains a Y and chart[k+1][j] contains a Z, add an X to chart[i][j] with two backpointers to the Y in chart[i][k] and the Z in chart[k+1][j]

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

CKY: filling the chart



CKY: filling one cell

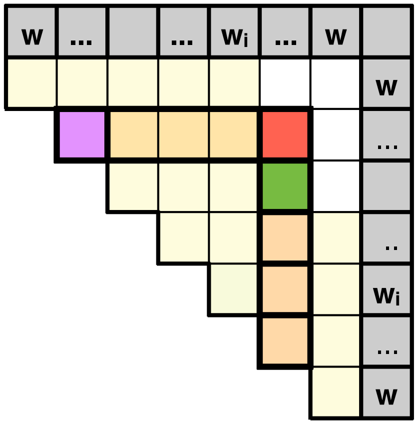


chart[2][6]:

w_1 $\mathbf{W_2}$ $\mathbf{W_3}$ $\mathbf{W_4}$ $\mathbf{W_5}$ $\mathbf{W_6}$ w_7

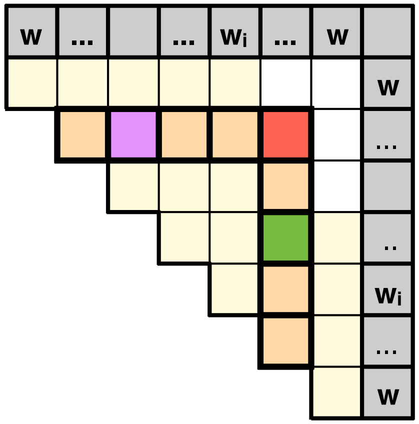
chart[2][6]:

w_1 $W_2W_3W_4W_5W_6$ w_7



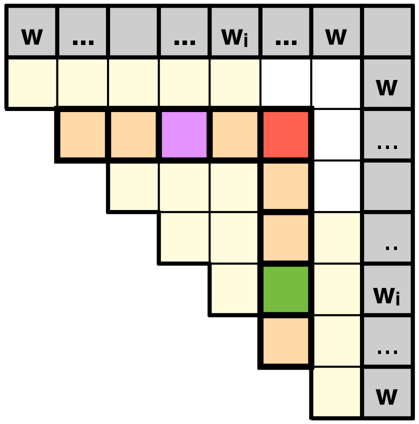
chart[2][6]:

w_1 $W_2W_3W_4W_5W_6$ w_7



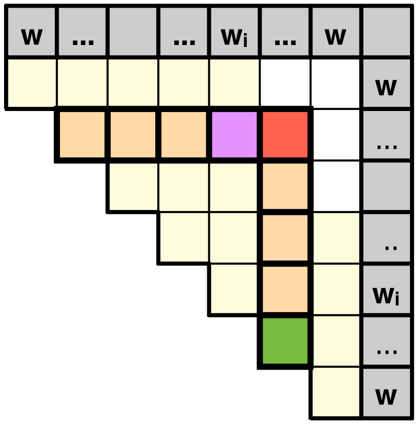
chart[2][6]:

w_1 $W_2W_3W_4W_5W_6$ w_7



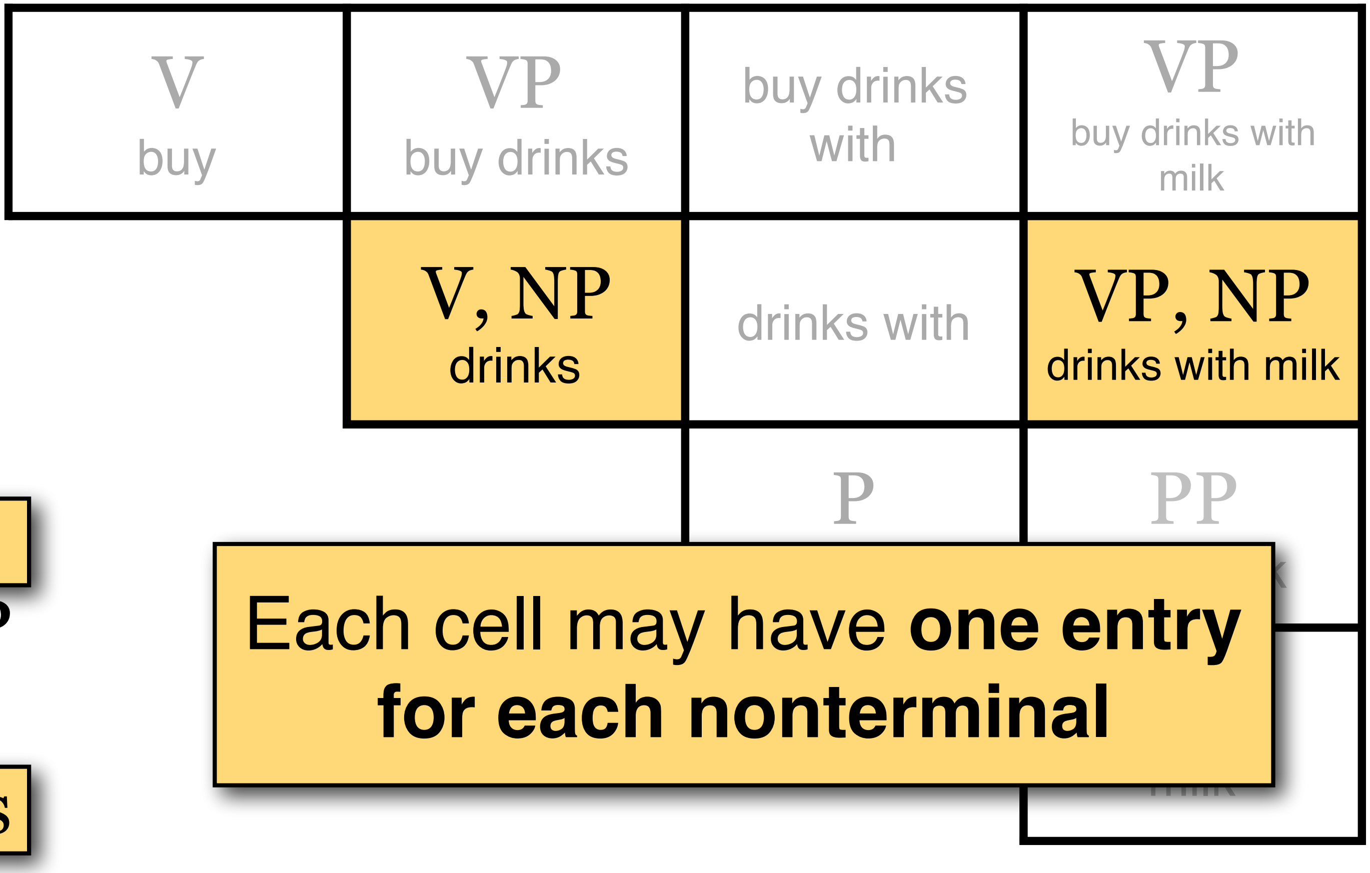
chart[2][6]:

w_1 $W_2W_3W_4W_5W_6$ w_7



The CKY parsing algorithm

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



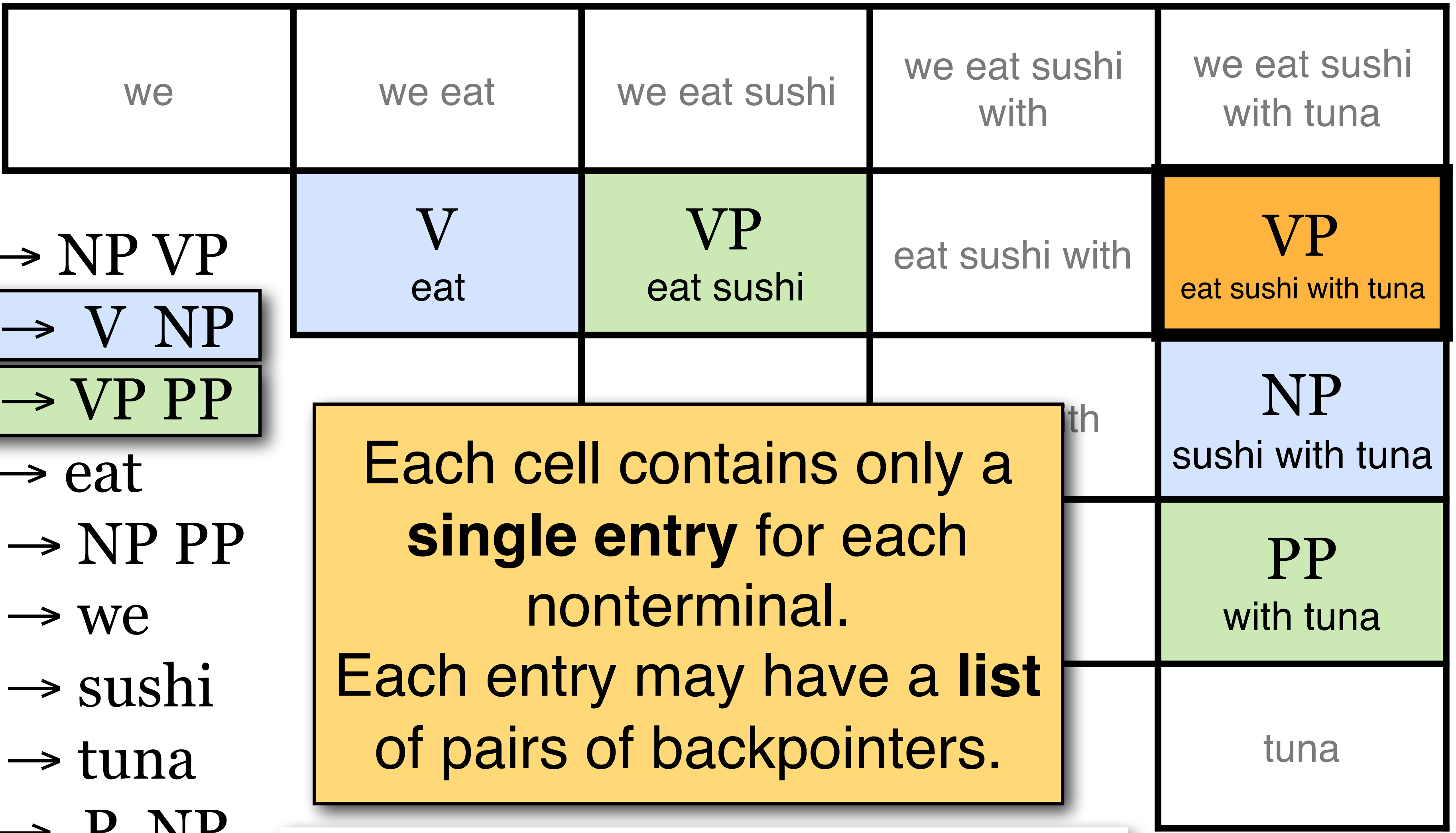
V buy	VP buy drinks	buy drinks with	VP buy drinks with milk
	V, NP **drinks**	drinks with	**VP, NP** **drinks with milk**
		P	PP

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

We buy drinks with milk

The CKY parsing algorithm

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

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 could 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 ⟶ NP VP
NP ⟶ NP PP
NP ⟶ sushi
NP ⟶ I
NP ⟶ chopsticks
NP ⟶ you
VP ⟶ VP PP
VP ⟶ Verb NP
Verb ⟶ eat
PP ⟶ Prep NP
Prep ⟶ with

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

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

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

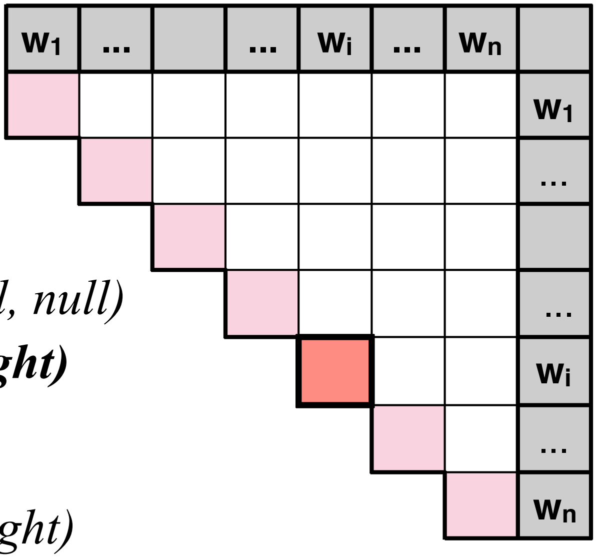
2. For each **list of pairs of backpointers** (e.g.VP → V NP and VP → VP PP): **sum** #trees

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

Cocke Kasami Younger (1)

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

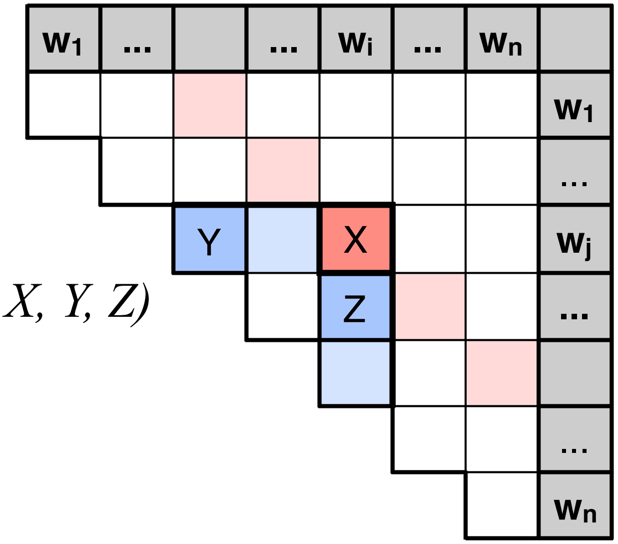
```
initChart(n):
    for i = 1...n:
        initCell(i,i)
initCell(i,i):
    for c in lex(word[i]):
        addToCell(cell[i][i], c, null, null)
addToCell(Parent,cell,Left, Right)
    if (cell.hasEntry(Parent)):
        P = cell.getEntry(Parent)
        P.addBackpointers(Left, Right)
    else cell.addEntry(Parent, Left, Right)
```



```
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 $X \rightarrow Y\ Z$ in Rules:
                    addToCell(cell[i][j],X, Y, Z)
```



Today's key concepts

Natural language syntax

- Constituents
- Dependencies
- Context-free grammar
- Arguments and modifiers
- Recursion in natural language

Today’s reading

Textbook:

Jurafsky and Martin, Chapter 12, sections 1-7