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

Lecture 9: The CKY parsing algorithm

Julia Hockenmaier
juliahmr@illinois.edu
3324 Siebel Center

Last lecture's key concepts

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

Defining grammars for natural language

An example CFG

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"

Reminder: Context-free grammars

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

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

A set of terminals $\boldsymbol{\Sigma}$
(e.g. $\boldsymbol{\Sigma}$ = {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) $\beta \in (\mathbf{N} \cup \boldsymbol{\Sigma})^* \}$

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

Constituents: Heads and dependents

There are different kinds of constituents:

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

Every phrase has a **head**:

Noun phrases: the <u>man</u>, a <u>girl</u> with glasses, <u>Illinois</u>
Prepositional phrases: <u>with</u> glasses, <u>in</u> the garden
Verb phrases: <u>eat</u> sushi, <u>sleep</u>, <u>sleep</u> soundly

The other parts are its **dependents**.
Dependents are either **arguments** or **adjuncts**

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

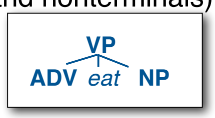
Heads:
We assume that each RHS has one head, e.g.
VP → Verb NP (Verbs are heads of VPs)
NP → Det Noun (Nouns are heads of NPs)
S → NP VP (VPs are heads of sentences)
Exception: Coordination, lists: VP → VP conj VP

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)

Chomsky Normal Form

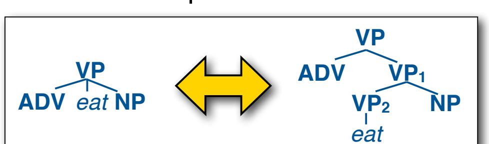
The right-hand side of a standard CFG can have an **arbitrary number of symbols** (terminals and nonterminals):

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

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)

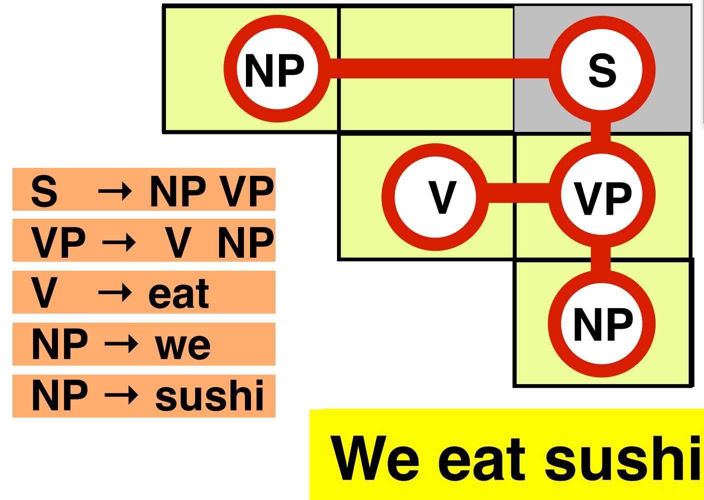
Presumes a CFG in Chomsky Normal Form:

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

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

The CKY parsing algorithm

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



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:

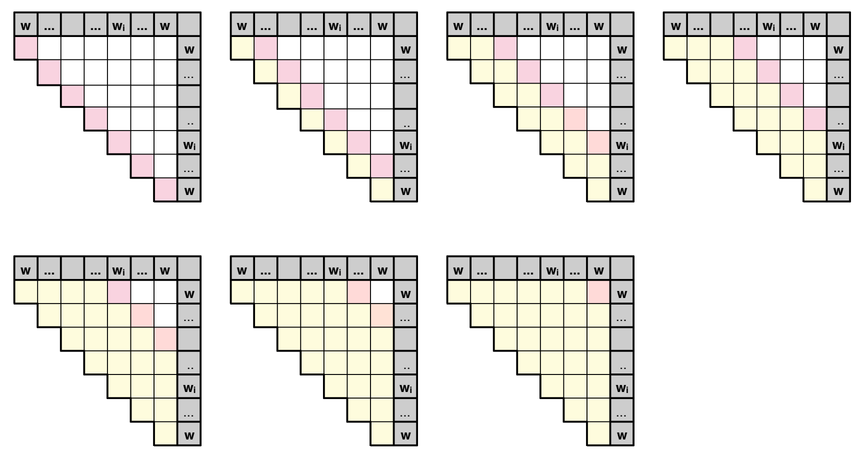
Fill in all cells chart[i][i+1], then chart[i][i+2], …,
until you reach chart[1][n] (the top right corner of the chart)

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

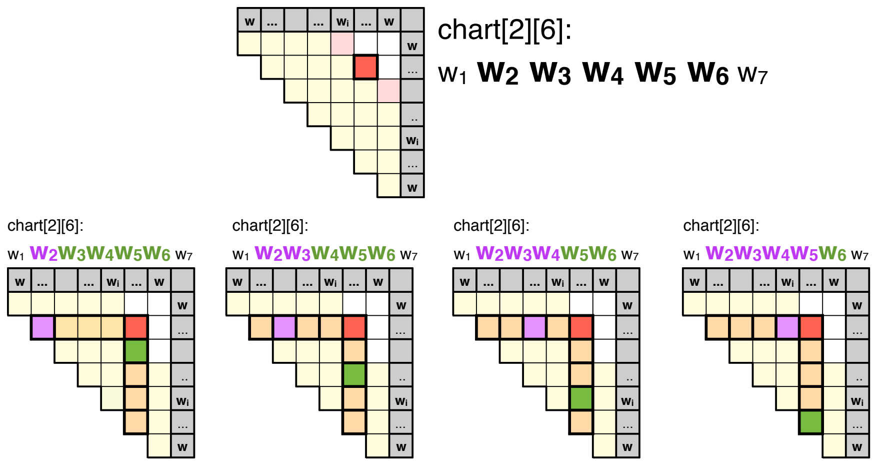
– If the grammar has a rule X → YZ, chart[i][k] contains a Y and chart[k+1][j] contains a Z, add an X to chart[i][j] with two backpointers to the Y in chart[i][k] and the Z in chart[k+1][j]

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

CKY: filling the chart

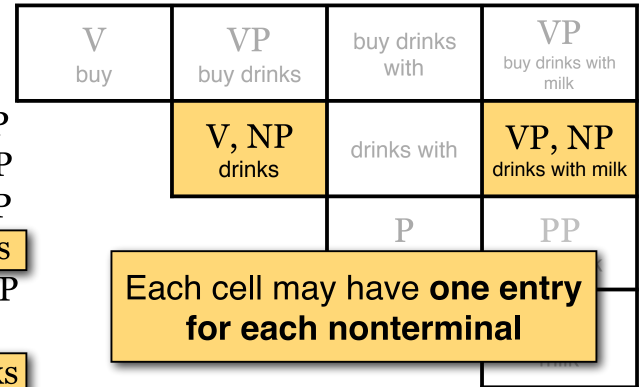


CKY: filling one cell



The CKY parsing algorithm

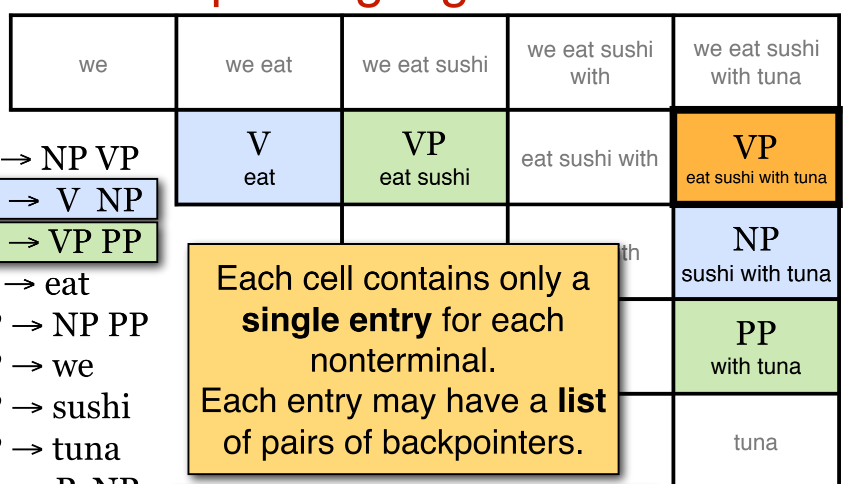
S → NP VP
VP → V NP
VP → VP PP
V → drinks
NP → NP PP
NP → we
NP → drinks
NP → milk
PP → P NP
P → with



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



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?

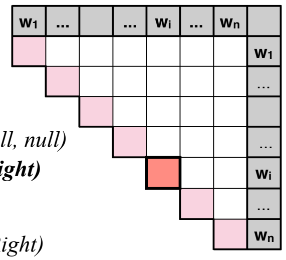
1. For each **pair of backpointers** (e.g. VP → V NP): **multiply** #trees of children
$trees(VP_{VP \rightarrow V\ NP}) = trees(V) \times trees(NP)$

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

Cocke Kasami Younger (1)

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

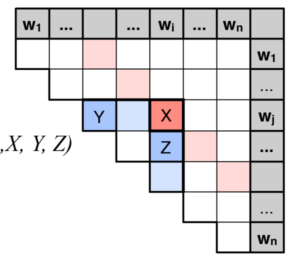
```
initChart(n):
  for i = 1...n:
     initCell(i,i)
initCell(i,i):
  for c in lex(word[i]):
     addToCell(cell[i][i], c, null, null)
addToCell(Parent,cell,Left, Right)
  if (cell.hasEntry(Parent)):
     P = cell.getEntry(Parent)
     P.addBackpointers(Left, Right)
  else cell.addEntry(Parent, Left,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

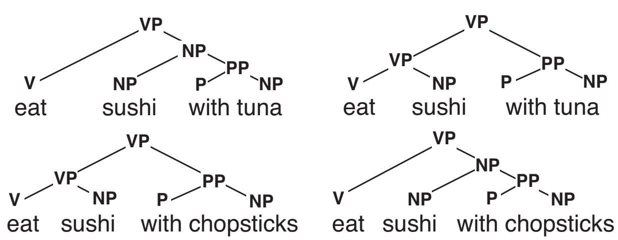
```
combineCells(i,k,j):
  for Y in cell[i][k]:
    for Z in cell[k +1][j]:
      for X in Nonterminals:
        if X →Y Z in Rules:
          addToCell(cell[i][j],X, Y, Z)
```



Dealing with ambiguity: 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 ?

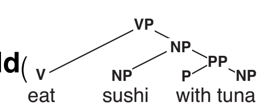
We need a model of P(τ | S)

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:

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

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	→	NP VP	VP	→	Verb NP	NP	→	Det Noun
S	→	S conj S	VP	→	VP PP	NP	→	NP PP
S	→	.....	VP	→	.....	NP	→	.....

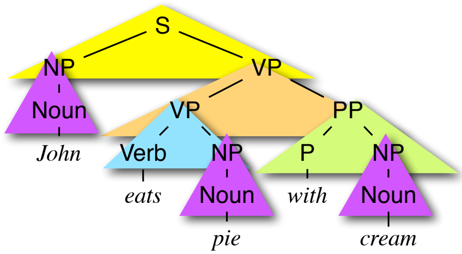
Probabilistic Context-Free Grammars

For every nonterminal X, define a probability distribution P(X → α | X) over all rules with the same LHS symbol X:

S	→	NP VP	0.8
S	→	S conj S	0.2
NP	→	Noun	0.2
NP	→	Det Noun	0.4
NP	→	NP PP	0.2
NP	→	NP conj NP	0.2
VP	→	Verb	0.4
VP	→	Verb NP	0.3
VP	→	Verb NP NP	0.1
VP	→	VP PP	0.2
PP	→	P NP	1.0

Computing P(τ) with a PCFG

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



S	→	NP VP	0.8
S	→	S conj S	0.2
NP	→	Noun	0.2
NP	→	Det Noun	0.4
NP	→	NP PP	0.2
NP	→	NP conj NP	0.2
VP	→	Verb	0.4
VP	→	Verb NP	0.3
VP	→	Verb NP NP	0.1
VP	→	VP PP	0.2
PP	→	P NP	1.0

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

$= \mathbf{0.00384}$

PCFG parsing (decoding): Probabilistic CKY

Probabilistic CKY: Viterbi

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

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

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

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

Probabilistic CKY

Input: POS-tagged sentence
`John_N eats_V pie_N with_P cream_N`

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

S	⟶	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	⟶	P NP	1.0