

# Parsing with Context Free Grammars

CMSC 723 / LING 723 / INST 725

MARINE CARPUAT

[marine@cs.umd.edu](mailto:marine@cs.umd.edu)

# Today's Agenda

- Grammar-based parsing with CFGs
  - CKY algorithm
- Dealing with ambiguity
  - Probabilistic CFGs
- Strategies for improvement
  - Rule rewriting / Lexicalization

# Sample Grammar

<b>Grammar</b>	<b>Lexicon</b>
$S \rightarrow NP VP$	$Det \rightarrow that \mid this \mid a$
$S \rightarrow Aux NP VP$	$Noun \rightarrow book \mid flight \mid meal \mid money$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid prefer$
$NP \rightarrow Pronoun$	$Pronoun \rightarrow I \mid she \mid me$
$NP \rightarrow Proper-Noun$	$Proper-Noun \rightarrow Houston \mid NWA$
$NP \rightarrow Det Nominal$	$Aux \rightarrow does$
$Nominal \rightarrow Noun$	$Preposition \rightarrow from \mid to \mid on \mid near \mid through$
$Nominal \rightarrow Nominal Noun$	
$Nominal \rightarrow Nominal PP$	
$VP \rightarrow Verb$	
$VP \rightarrow Verb NP$	
$VP \rightarrow Verb NP PP$	
$VP \rightarrow Verb PP$	
$VP \rightarrow VP PP$	
$PP \rightarrow Preposition NP$	

GRAMMAR-BASED PARSING: CKY

# Grammar-based Parsing

- Problem setup
  - Input: string **and a CFG**
  - Output: parse tree assigning proper structure to input string
- “Proper structure”
  - Tree that covers all and only words in the input
  - Tree is rooted at an S
  - Derivations obey rules of the grammar
  - Usually, more than one parse tree...

# Parsing Algorithms

- Two basic (= bad) algorithms:
  - Top-down search
  - Bottom-up search
- A “real” algorithm:
  - CKY parsing

# Top-Down Search

- Observation
  - trees must be rooted with an S node
- Parsing strategy
  - Start at top with an S node
  - Apply rules to build out trees
  - Work down toward leaves

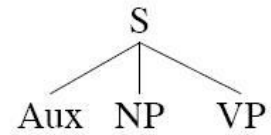
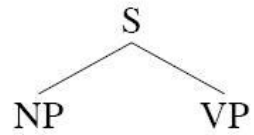
# Top-Down Search

s



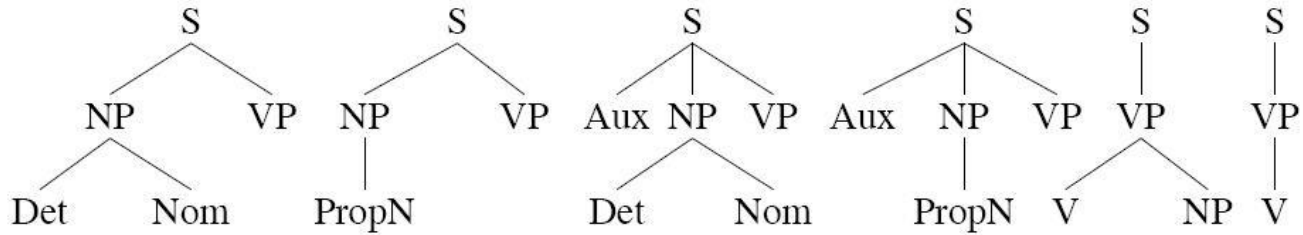
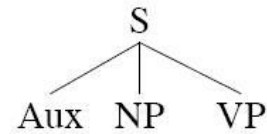
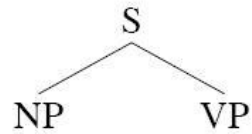
# Top-Down Search

S



# Top-Down Search

S



# Bottom-Up Search

- Observation
  - trees must cover all input words
- Parsing strategy
  - Start at the bottom with input words
  - Build structure based on grammar
  - Work up towards the root  $S$

# Bottom-Up Search

Book that flight

# Bottom-Up Search

Book that fight

Noun Det Noun  
| | |  
Book that fight

Verb Det Noun  
| | |  
Book that fight

# Bottom-Up Search

Book that fight

Noun Det Noun  
| | |  
Book that fight

Verb Det Noun  
| | |  
Book that fight

Nominal      Nominal  
|            |  
Noun   Det   Noun  
|        |     |  
Book    that   fight

                         Nominal  
                         |  
                         Verb   Det   Noun  
                         |     |     |  
                         Book   that   fight

# Bottom-Up Search

Book that flight

Noun Det Noun  
| | |  
Book that flight

Verb Det Noun  
| | |  
Book that flight

Nominal Nominal  
| |  
Noun Det Noun  
| | |  
Book that flight

Nominal  
|  
Verb Det Noun  
| | |  
Book that flight

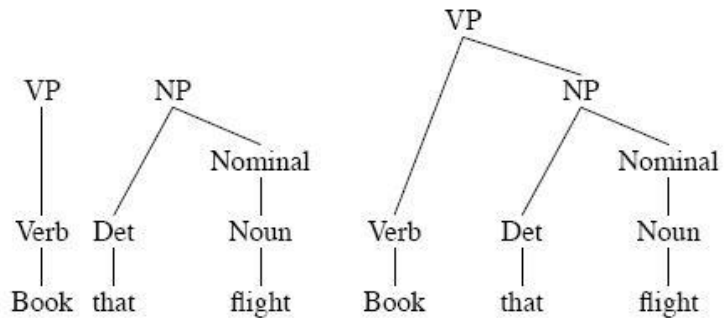
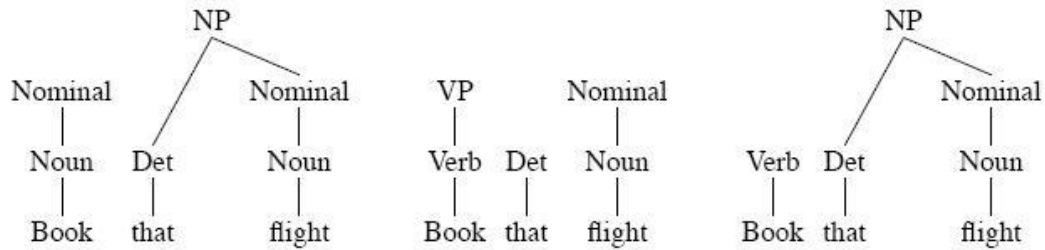
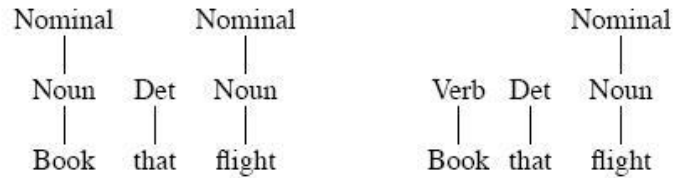
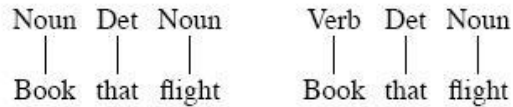
Nominal NP  
| |  
Noun Det Noun  
| | |  
Book that flight

VP Nominal  
| |  
Verb Det Noun  
| | |  
Book that flight

NP  
| |  
Verb Det Noun  
| | |  
Book that flight

# Bottom-Up Search

Book that flight





# Top-Down vs. Bottom-Up

- Top-down search
  - Only searches valid trees
  - But, considers trees that are not consistent with any of the words
- Bottom-up search
  - Only builds trees consistent with the input
  - But, considers trees that don't lead anywhere

# Parsing as Search

- Search involves controlling choices in the search space
  - Which node to focus on in building structure
  - Which grammar rule to apply
- General strategy: backtracking
  - Make a choice, if it works out then fine
  - If not, back up and make a different choice

# Shared Sub-Problems

- Observation
  - ambiguous parses still share sub-trees
- We don't want to redo work that's already been done
- Unfortunately, naïve backtracking leads to duplicate work

# Efficient Parsing with the CKY Algorithm

- Solution: Dynamic programming
- Intuition: store partial results in tables
  - Thus avoid repeated work on shared sub-problems
  - Thus efficiently store ambiguous structures with shared sub-parts
- We'll cover one example
  - CKY: roughly, bottom-up

# CKY Parsing: CNF

- CKY parsing requires that the grammar consist of binary rules in Chomsky Normal Form
  - All rules of the form:

$$A \rightarrow B C$$

$$D \rightarrow w$$

- What does the tree look like?

# CKY Parsing with Arbitrary CFGs

- What if my grammar has rules like  
 $VP \rightarrow NP PP PP$ 
  - Problem: can't apply CKY!
  - Solution: rewrite grammar into CNF
    - Introduce new intermediate non-terminals into the grammar

$A \rightarrow B C D$    $A \rightarrow X D$   
 $X \rightarrow B C$

(Where  $X$  is a symbol that doesn't occur anywhere else in the grammar)

# Sample Grammar

<b>Grammar</b>	<b>Lexicon</b>
$S \rightarrow NP VP$	$Det \rightarrow that \mid this \mid a$
$S \rightarrow Aux NP VP$	$Noun \rightarrow book \mid flight \mid meal \mid money$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid prefer$
$NP \rightarrow Pronoun$	$Pronoun \rightarrow I \mid she \mid me$
$NP \rightarrow Proper-Noun$	$Proper-Noun \rightarrow Houston \mid NWA$
$NP \rightarrow Det Nominal$	$Aux \rightarrow does$
$Nominal \rightarrow Noun$	$Preposition \rightarrow from \mid to \mid on \mid near \mid through$
$Nominal \rightarrow Nominal Noun$	
$Nominal \rightarrow Nominal PP$	
$VP \rightarrow Verb$	
$VP \rightarrow Verb NP$	
$VP \rightarrow Verb NP PP$	
$VP \rightarrow Verb PP$	
$VP \rightarrow VP PP$	
$PP \rightarrow Preposition NP$	

# CNF Conversion

## Original Grammar

$S \rightarrow NP VP$

$S \rightarrow Aux NP VP$

$S \rightarrow VP$

$NP \rightarrow Pronoun$

$NP \rightarrow Proper-Noun$

$NP \rightarrow Det Nominal$

$Nominal \rightarrow Noun$

$Nominal \rightarrow Nominal Noun$

$Nominal \rightarrow Nominal PP$

$VP \rightarrow Verb$

$VP \rightarrow Verb NP$

$VP \rightarrow Verb NP PP$

$VP \rightarrow Verb PP$

$VP \rightarrow VP PP$

$PP \rightarrow Preposition NP$

## CNF Version

$S \rightarrow NP VP$

$S \rightarrow X1 VP$

$X1 \rightarrow Aux NP$

$S \rightarrow book \mid include \mid prefer$

$S \rightarrow Verb NP$

$S \rightarrow X2 PP$

$S \rightarrow Verb PP$

$S \rightarrow VP PP$

$NP \rightarrow I \mid she \mid me$

$NP \rightarrow TWA \mid Houston$

$NP \rightarrow Det Nominal$

$Nominal \rightarrow book \mid flight \mid meal \mid money$

$Nominal \rightarrow Nominal Noun$

$Nominal \rightarrow Nominal PP$

$VP \rightarrow book \mid include \mid prefer$

$VP \rightarrow Verb NP$

$VP \rightarrow X2 PP$

$X2 \rightarrow Verb NP$

$VP \rightarrow Verb PP$

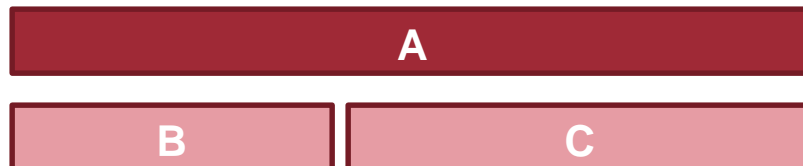
$VP \rightarrow VP PP$

$PP \rightarrow Preposition NP$



# CKY Parsing: Intuition

- Consider the rule  $D \rightarrow w$ 
  - Terminal (word) forms a constituent
  - Trivial to apply
- Consider the rule  $A \rightarrow B C$ 
  - “If there is an A somewhere in the input, then there must be a B followed by a C in the input”
  - First, precisely define span  $[i, j]$
  - If A spans from  $i$  to  $j$  in the input then there must be some  $k$  such that  $i < k < j$
  - Easy to apply: we just need to try different values for  $k$



# CKY Parsing: Table

- Any constituent can conceivably span  $[i, j]$  for all  $0 \leq i < j \leq N$ , where  $N = \text{length of input string}$ 
  - We need an  $N \times N$  table to keep track of all spans...
  - But we only need half of the table
- Semantics of table: cell  $[i, j]$  contains  $A$  iff  $A$  spans  $i$  to  $j$  in the input string
  - Of course, must be allowed by the grammar!

		TO:					
		1	2	3	4	5	6
FROM:	0	0-1	0-2	0-3	0-4	0-5	0-6
	1		1-2	1-3	1-4	1-5	1-6
	2			2-3	2-4	2-5	2-6
	3				3-4	3-5	3-6
	4					4-5	4-6
	5						5-6

# CKY Parsing: Table-Filling

- In order for A to span  $[i, j]$ 
  - $A \rightarrow B C$  is a rule in the grammar, and
  - There must be a B in  $[i, k]$  and a C in  $[k, j]$  for some  $i < k < j$
- Operationally
  - To apply rule  $A \rightarrow B C$ , look for a B in  $[i, k]$  and a C in  $[k, j]$
  - In the table: look left in the row and down in the column

TO:

	1	2	3	4	5	6
0	0-1	0-2	0-3	0-4	0-5	0-6
1		1-2	1-3	1-4	1-5	1-6
2			2-3	2-4	2-5	2-6
3				3-4	3-5	3-6
4					4-5	4-6
5						5-6

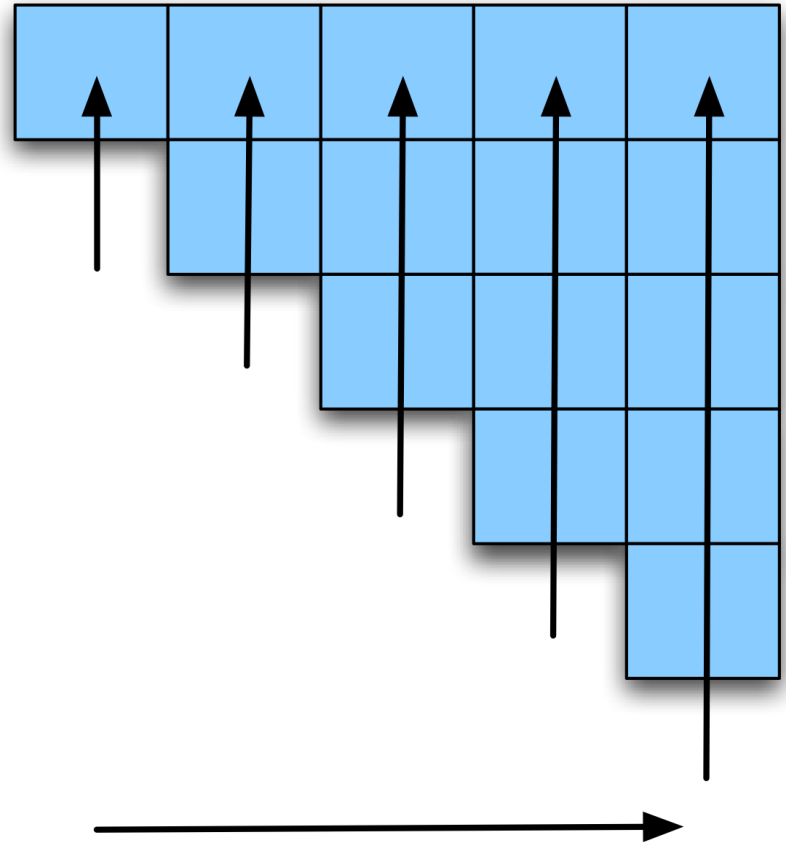
FROM:

# CKY Parsing: Canonical Ordering

- Standard CKY algorithm:
  - Fill the table a column at a time, from left to right, bottom to top
  - Whenever we're filling a cell, the parts needed are already in the table (to the left and below)
- Nice property: processes input left to right, word at a time

# CKY Parsing: Ordering Illustrated

<i>Book</i>	<i>this</i>	<i>flight</i>	<i>through</i>	<i>Houston</i>
S, VP, Verb Nominal, Noun [0,1]		S,VP,X2 [0,3]		S,VP,X2 [0,5]
	Det [1,2]	NP [1,3]		NP [1,5]
		Nominal, Noun [2,3]		Nominal [2,5]
			Prep [3,4]	PP [3,5]
				NP, Proper- Noun [4,5]



# CKY Algorithm

**function** CKY-PARSE(*words*, *grammar*) **returns** *table*

**for**  $j \leftarrow$  **from** 1 **to** LENGTH(*words*) **do**

$table[j-1, j] \leftarrow \{A \mid A \rightarrow words[j] \in grammar\}$

**for**  $i \leftarrow$  **from**  $j-2$  **downto** 0 **do**

**for**  $k \leftarrow i+1$  **to**  $j-1$  **do**

$table[i, j] \leftarrow table[i, j] \cup$

$\{A \mid A \rightarrow BC \in grammar,$

$B \in table[i, k],$

$C \in table[k, j]\}$

# CKY: Example

*Book*      *this*      *flight*      *through*      *Houston*

S, VP, Verb, Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	?
	Det [1,2]	NP [1,3]	[1,4]	?
		Nominal, Noun [2,3]	[2,4]	?
			Prep [3,4]	?
				NP, Proper- Noun [4,5]

Filling column 5

# CKY: Example

*Book this flight through Houston*

Recall our CNF grammar:

$S \rightarrow NP VP$   
 $S \rightarrow X1 VP$   
 $X1 \rightarrow Aux NP$   
 $S \rightarrow book \mid include \mid prefer$   
 $S \rightarrow Verb NP$   
 $S \rightarrow X2 PP$   
 $S \rightarrow Verb PP$   
 $S \rightarrow VP PP$   
 $NP \rightarrow I \mid she \mid me$   
 $NP \rightarrow TWA \mid Houston$   
 $NP \rightarrow Det Nominal$   
 $Nominal \rightarrow book \mid flight \mid meal \mid money$   
 $Nominal \rightarrow Nominal Noun$   
 $Nominal \rightarrow Nominal PP$   
 $VP \rightarrow book \mid include \mid prefer$   
 $VP \rightarrow Verb NP$   
 $VP \rightarrow X2 PP$   
 $X2 \rightarrow Verb NP$   
 $VP \rightarrow Verb PP$   
 $VP \rightarrow VP PP$   
 $PP \rightarrow Preposition NP$

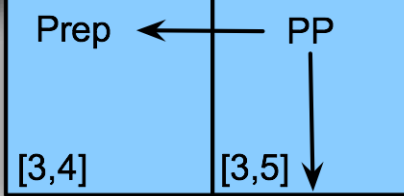
S, VP, Verb, Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	?
	Det [1,2]	NP [1,3]	[1,4]	?
		Nominal, Noun [2,3]	[2,4]	?
			Prep [3,4]	?
				NP, Proper- Noun [4,5]



# CKY: Example

*Book*      *this*      *flight*      *through*      *Houston*

S, VP, Verb, Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	[0,5]
	Det [1,2]	NP [1,3]	[1,4]	[1,5]
		Nominal, Noun [2,3]	[2,4]	[2,5]
			Prep ← PP [3,4]	[3,5]
				NP, Proper- Noun [4,5]



# CKY: Example

*Book*      *this*      *flight*      *through*      *Houston*

S, VP, Verb, Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	[0,5]
	Det [1,2]	NP [1,3]	[1,4]	[1,5]
		Nominal, Noun [2,3]	[2,4]	Nominal [2,5]
			Prep [3,4]	PP [3,5]
				NP, Proper- Noun [4,5]

# CKY: Example

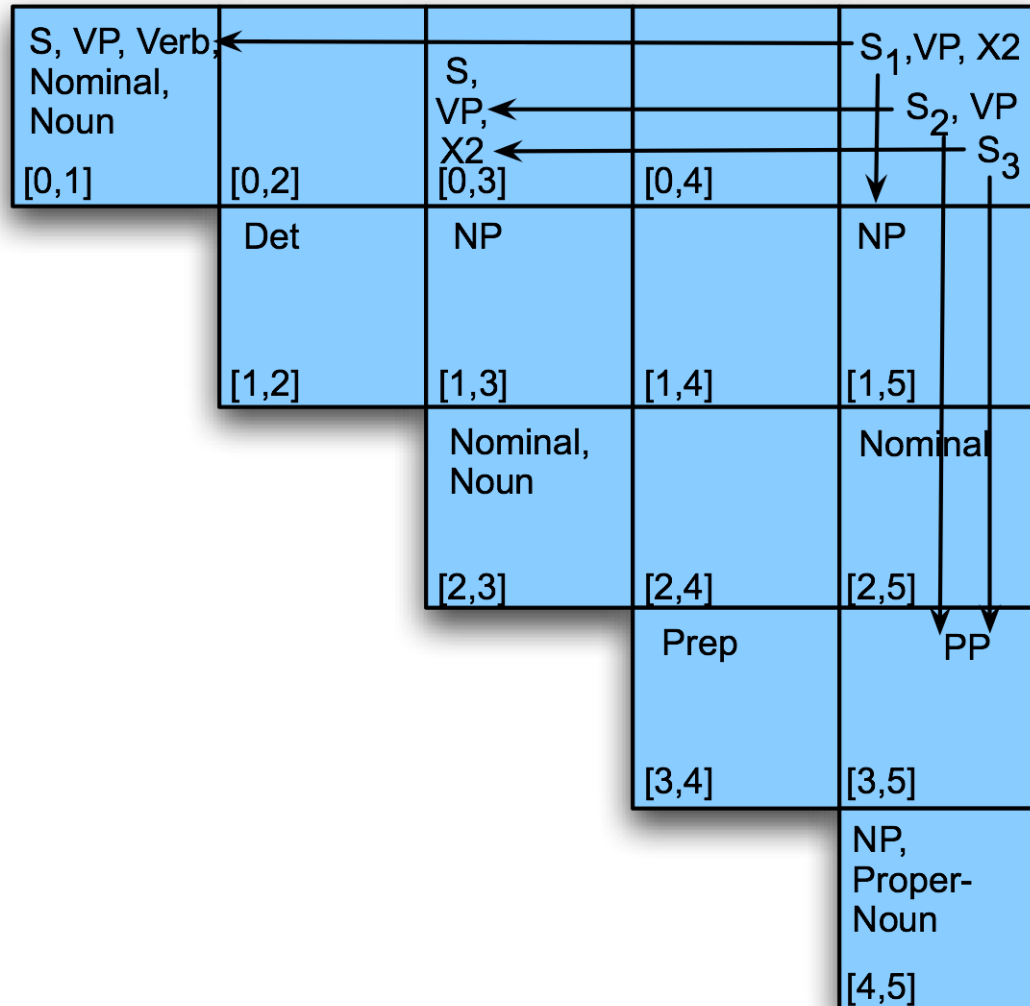
Recall our CNF grammar:

- $S \rightarrow NP VP$
- $S \rightarrow X1 VP$
- $X1 \rightarrow Aux NP$
- $S \rightarrow book \mid include \mid prefer$
- $S \rightarrow Verb NP$
- $S \rightarrow X2 PP$
- $S \rightarrow Verb PP$
- $S \rightarrow VP PP$
- $NP \rightarrow I \mid she \mid me$
- $NP \rightarrow TWA \mid Houston$
- $NP \rightarrow Det Nominal$
- $Nominal \rightarrow book \mid flight \mid meal \mid money$
- $Nominal \rightarrow Nominal Noun$
- $Nominal \rightarrow Nominal PP$
- $VP \rightarrow book \mid include \mid prefer$
- $VP \rightarrow Verb NP$
- $VP \rightarrow X2 PP$
- $X2 \rightarrow Verb NP$
- $VP \rightarrow Verb PP$
- $VP \rightarrow VP PP$
- $PP \rightarrow Preposition NP$

	<i>Book</i>	<i>this</i>	<i>flight</i>	<i>through</i>	<i>Houston</i>
S, VP, Verb, Nominal, Noun [0,1]	[0,2]	[0,3]	[0,4]	[0,5]	?
	Det ←	NP		NP	
	[1,2]	[1,3]	[1,4]	[1,5]	
		Nominal, Noun		Nominal	
		[2,3]	[2,4]	[2,5]	
			Prep	PP	
			[3,4]	[3,5]	
				NP, Proper- Noun	
				[4,5]	

# CKY: Example

*Book this flight through Houston*



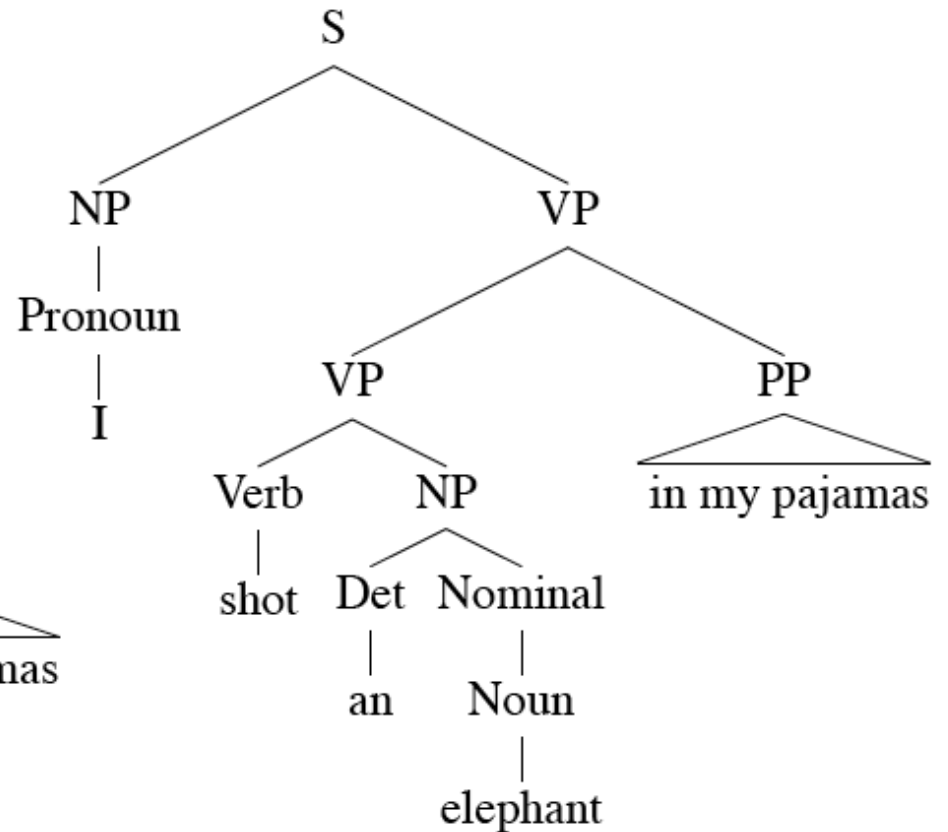
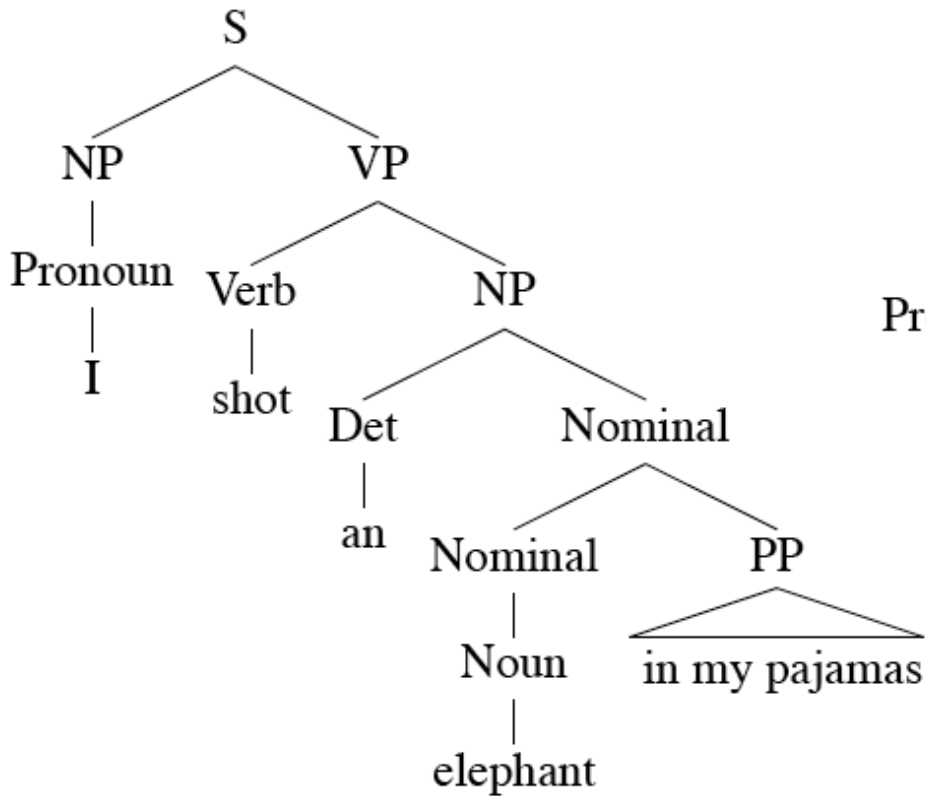
# CKY Parsing: Recognize or Parse

- Recognizer
  - Output is binary
  - Can the complete span of the sentence be covered by an S symbol?
- Parser
  - Output is a parse tree
  - From recognizer to parser: add backpointers!

# Ambiguity

- CKY can return multiple parse trees
  - Plus: compact encoding with shared sub-trees
  - Plus: work deriving shared sub-trees is reused
  - Minus: algorithm doesn't tell us which parse is correct!

# Ambiguity



# PROBABILISTIC CONTEXT-FREE GRAMMARS



# Simple Probability Model

- A derivation (tree) consists of the bag of grammar rules that are in the tree
  - The probability of a tree is the product of the probabilities of the rules in the derivation.

$$P(T, S) = \prod_{node \in T} P(rule(n))$$

# Rule Probabilities

- What's the probability of a rule?
- Start at the top...
  - A tree should have an  $S$  at the top. So given that we know we need an  $S$ , we can ask about the probability of each particular  $S$  rule in the grammar:  $P(\text{particular rule} \mid S)$
- In general we need  $P(\alpha \rightarrow \beta \mid \alpha)$  for each rule in the grammar

# Training the Model

- We can get the estimates we need from a treebank

$$P(\alpha \rightarrow \beta | \alpha) = \frac{\text{Count}(\alpha \rightarrow \beta)}{\sum_{\gamma} \text{Count}(\alpha \rightarrow \gamma)} = \frac{\text{Count}(\alpha \rightarrow \beta)}{\text{Count}(\alpha)}$$

For example, to get the probability for a particular *VP* rule:

1. count all the times the rule is used
2. divide by the number of *VPs* overall.

# Parsing (Decoding)

How can we get the best (most probable) parse for a given input?

1. Enumerate all the trees for a sentence
2. Assign a probability to each using the model
3. Return the argmax



# Examples

- These trees consist of the following rules.

Rules	P	Rules	P
S → VP	.05	S → VP	.05
VP → Verb NP	.20	VP → Verb NP NP	.10
NP → Det Nominal	.20	NP → Det Nominal	.20
Nominal → Nominal Noun	.20	NP → Nominal	.15
Nominal → Noun	.75	Nominal → Noun	.75
		Nominal → Noun	.75
Verb → book	.30	Verb → book	.30
Det → the	.60	Det → the	.60
Noun → dinner	.10	Noun → dinner	.10
Noun → flights	.40	Noun → flights	.40

$$P(T_{left}) = .05 * .20 * .20 * .20 * .75 * .30 * .60 * .10 * .40 = \mathbf{2.2 \times 10^{-6}}$$

$$P(T_{right}) = .05 * .10 * .20 * .15 * .75 * .75 * .30 * .60 * .10 * .40 = \mathbf{6.1 \times 10^{-7}}$$

# Dynamic Programming

- Of course, as with normal parsing we don't really want to do it that way...
- Instead, we need to exploit dynamic programming
  - For the parsing (as with CKY)
  - And for computing the probabilities and returning the best parse (as with Viterbi and HMMs)

# Probabilistic CKY

- Store probabilities of constituents in the table
  - $\text{table}[i,j,A]$  = probability of constituent  $A$  that spans positions  $i$  through  $j$  in input
- If  $A$  is derived from the rule  $A \rightarrow B C$  :
  - $\text{table}[i,j,A] = P(A \rightarrow B C \mid A) * \text{table}[i,k,B] * \text{table}[k,j,C]$
  - Where
    - $P(A \rightarrow B C \mid A)$  is the rule probability
    - $\text{table}[i,k,B]$  and  $\text{table}[k,j,C]$  are already in the table given the way that CKY operates
- Only store the MAX probability over all the  $A$  rules.



# Probabilistic CKY

**function** **PROBABILISTIC-CKY**(*words*, *grammar*) **returns** most probable parse  
and its probability

```
for  $j \leftarrow$  from 1 to LENGTH(words) do
  for all {  $A \mid A \rightarrow \text{words}[j] \in \text{grammar}$  }
     $\text{table}[j-1, j, A] \leftarrow P(A \rightarrow \text{words}[j])$ 
  for  $i \leftarrow$  from  $j-2$  downto 0 do
    for  $k \leftarrow i+1$  to  $j-1$  do
      for all {  $A \mid A \rightarrow BC \in \text{grammar},$ 
        and  $\text{table}[i, k, B] > 0$  and  $\text{table}[k, j, C] > 0$  }
        if ( $\text{table}[i, j, A] < P(A \rightarrow BC) \times \text{table}[i, k, B] \times \text{table}[k, j, C]$ ) then
           $\text{table}[i, j, A] \leftarrow P(A \rightarrow BC) \times \text{table}[i, k, B] \times \text{table}[k, j, C]$ 
           $\text{back}[i, j, A] \leftarrow \{k, B, C\}$ 
    return BUILD_TREE( $\text{back}[1, \text{LENGTH}(\text{words}), S]$ ),  $\text{table}[1, \text{LENGTH}(\text{words}), S]$ 
```

# Problems with PCFGs

- The probability model we're using is just based on the bag of rules in the derivation...
  1. Doesn't take the actual words into account in any useful way.
  2. Doesn't take into account *where* in the derivation a rule is used
  3. *Doesn't work terribly well*

IMPROVING OUR PARSER

# Improved Approaches

There are two approaches to overcoming these shortcomings

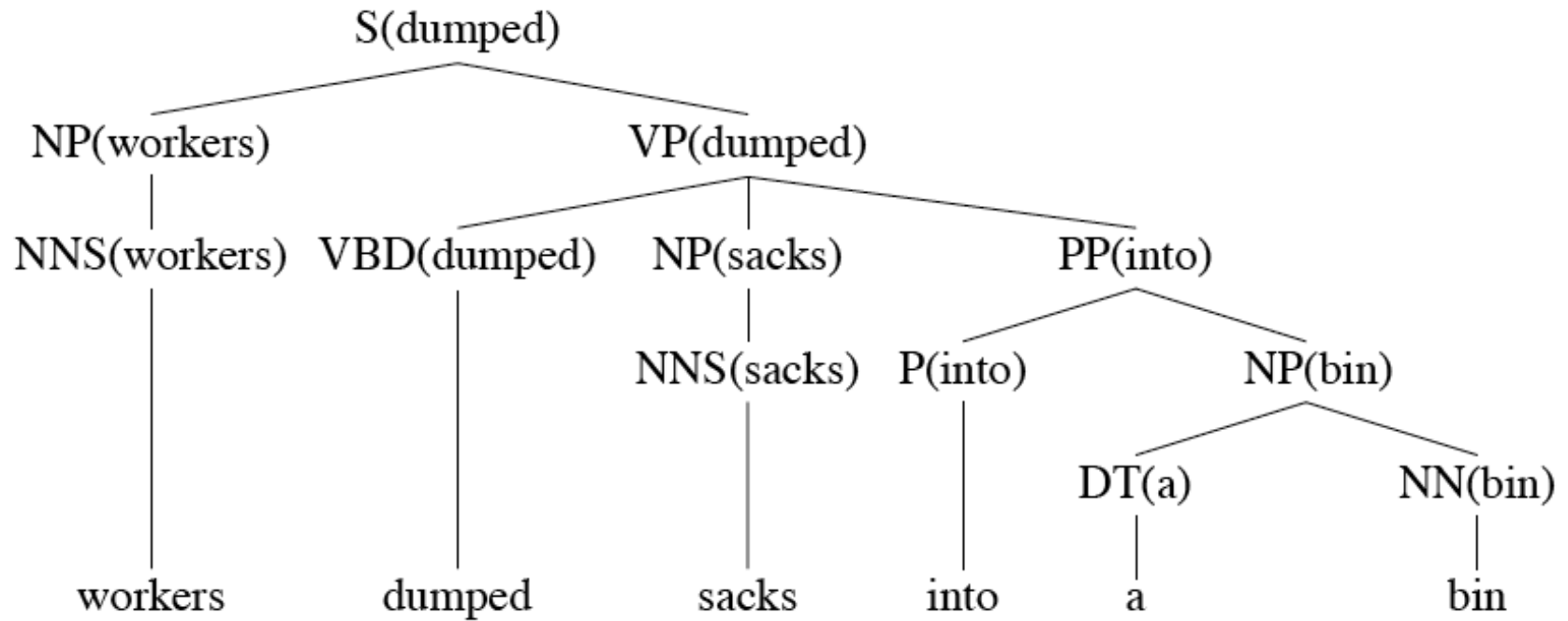
1. Rewrite the grammar to better capture the dependencies among rules
2. Integrate lexical dependencies into the model

# Solution 2:

## Lexicalized Grammars

- Lexicalize the grammars with heads
- Compute the rule probabilities on these lexicalized rules
- Run Prob CKY as before

# Lexicalized Grammars: Example



# How can we learn probabilities for lexicalized rules?

- We used to have
  - $VP \rightarrow V NP PP$
  - $P(\text{rule}|VP) = \text{count of this rule} / \text{number of VPs in a treebank}$
- Now we have fully lexicalized rules...
  - $VP(\text{dumped}) \rightarrow V(\text{dumped}) NP(\text{sacks}) PP(\text{into})$
  - $P(r|VP \wedge \text{dumped is the verb} \wedge \text{sacks is the head of the NP} \wedge \text{into is the head of the PP})$

# We need to make independence assumptions

- Strategies: exploit independence and collect the statistics we can get
  - Many many ways to do this...
- Let's consider one generative story: given a rule we'll
  1. Generate the head
  2. Generate the stuff to the left of the head
  3. Generate the stuff to the right of the head



# From the generative story to rule probabilities...

The rule probability for

$$P(VP(dumped, VBD) \rightarrow \\ VBD(dumped, VBD) NP(sacks, NNS) PP(into, P))$$

Can be estimated as

$$\begin{aligned} P_H(VBD|VP, dumped) &\times P_L(STOP|VP, VBD, dumped) \\ &\times P_R(NP(sacks, NNS)|VP, VBD, dumped) \\ &\times P_R(PP(into, P)|VP, VBD, dumped) \\ &\times P_R(STOP|VP, VBD, dumped) \end{aligned}$$

# Framework

- That's just one simple model
  - "Collins Model 1"
- Other assumptions that might lead to better models
  - make sure that you can get the counts you need
  - make sure they can get exploited efficiently during decoding

# Today's Agenda

- Grammar-based parsing with CFGs
  - CKY algorithm
- Dealing with ambiguity
  - Probabilistic CFGs
- Strategies for improvement
  - Lexicalization

# Today's Agenda

- Grammar-based parsing with CFGs
  - CKY algorithm
- Dealing with ambiguity
  - Probabilistic CFGs
- Strategies for improvement
  - Lexicalization
- Tools for parsing English, Chinese, French, ... with PCFGs <http://nlp.stanford.edu/software/lex-parser.shtml>