Parsing and Probabilistic Context Free Grammars

Instructor: Wei Xu
Ohio State University

Many slides from Ray Mooney and Michael Collins

Syntax

Syntactic Parsing

Parsing

- Given a string of terminals and a CFG, determine if the string can be generated by the CFG:
    - also return a parse tree for the string
    - also return all possible parse trees for the string

- Must search space of derivations for one that derives the given string.
    - Top-Down Parsing
    - Bottom-Up Parsing

Simple CFG for ATIS English

Grammar

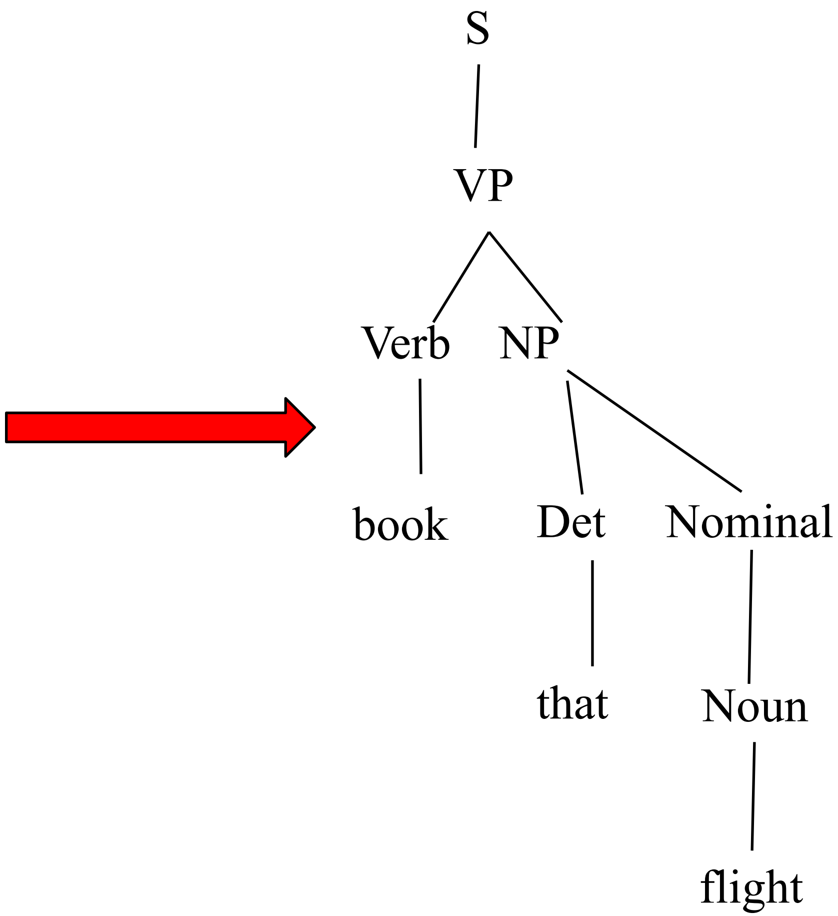
S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → VP PP
PP → Prep NP

Lexicon

Det → the | a | that | this
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | he | she | me
Proper-Noun → Houston | NWA
Aux → does
Prep → from | to | on | near | through

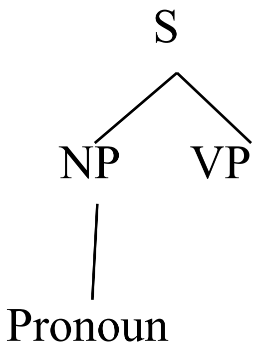
Parsing Example

book that flight

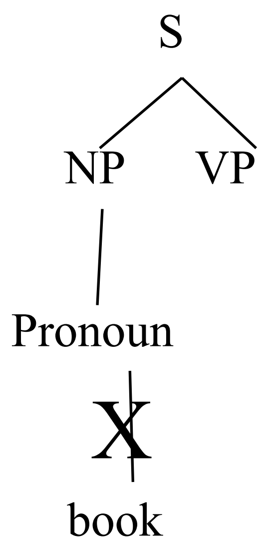


Top Down Parsing

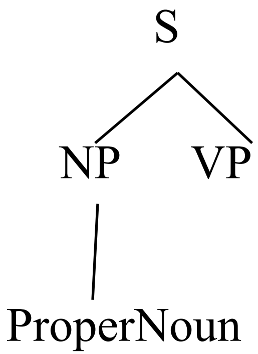
- Start searching space of derivations for the start symbol.



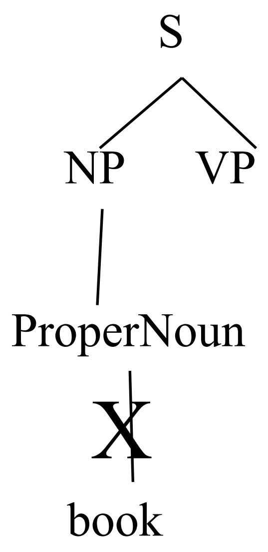
Top Down Parsing



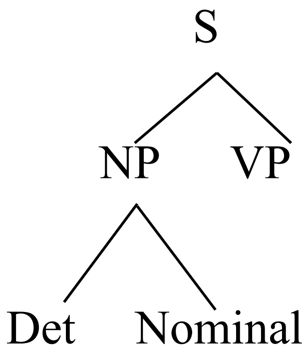
Top Down Parsing



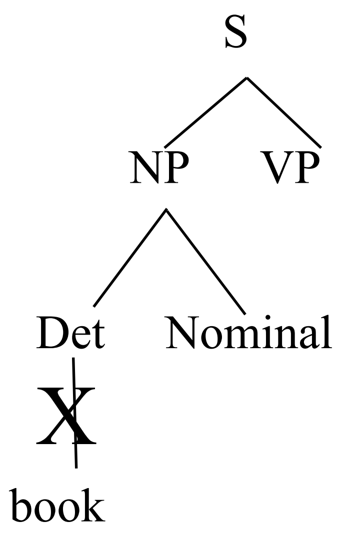
Top Down Parsing



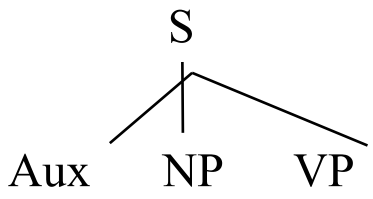
Top Down Parsing



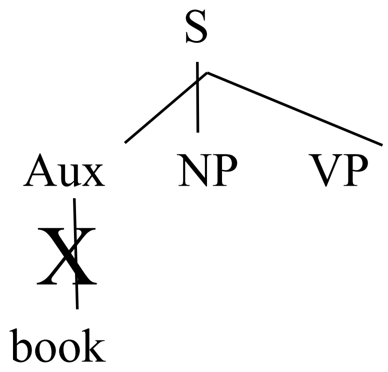
Top Down Parsing



Top Down Parsing

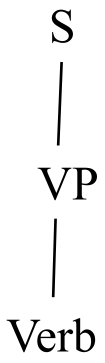


Top Down Parsing



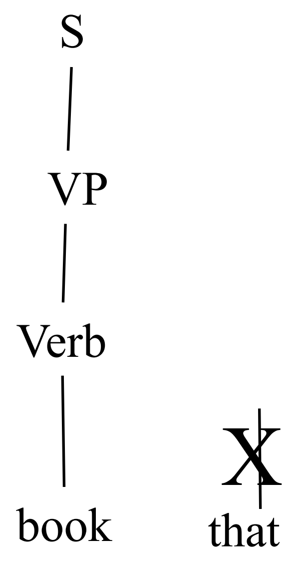
Top Down Parsing

Top Down Parsing

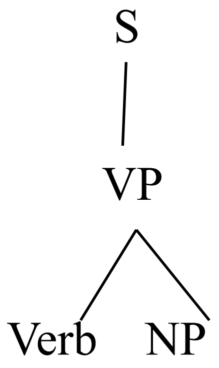


Top Down Parsing

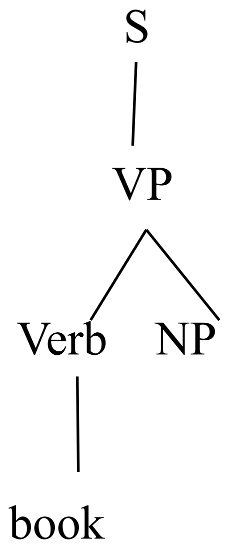
Top Down Parsing



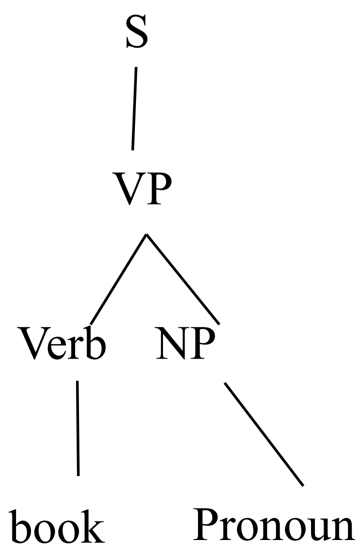
Top Down Parsing



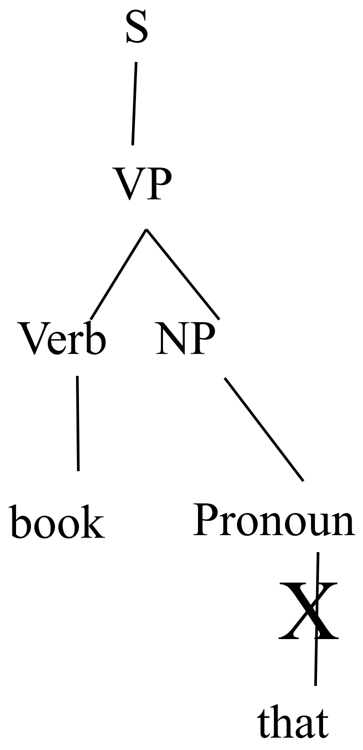
Top Down Parsing



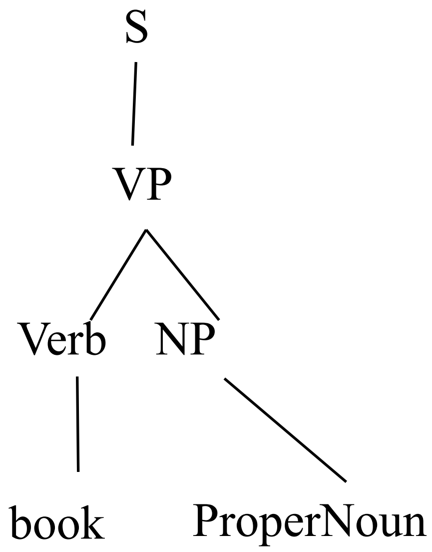
Top Down Parsing



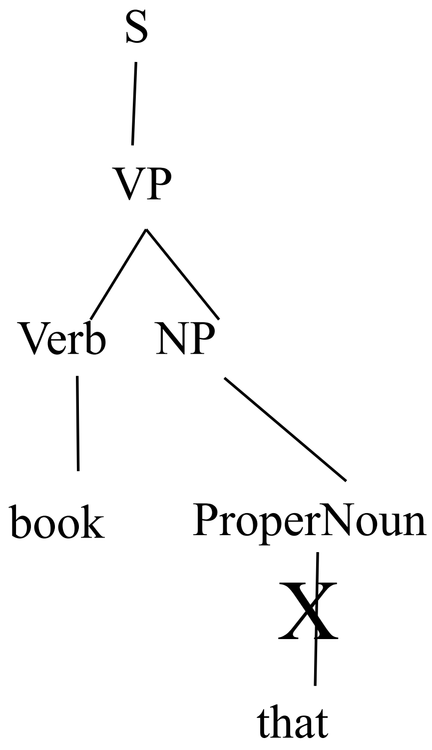
Top Down Parsing



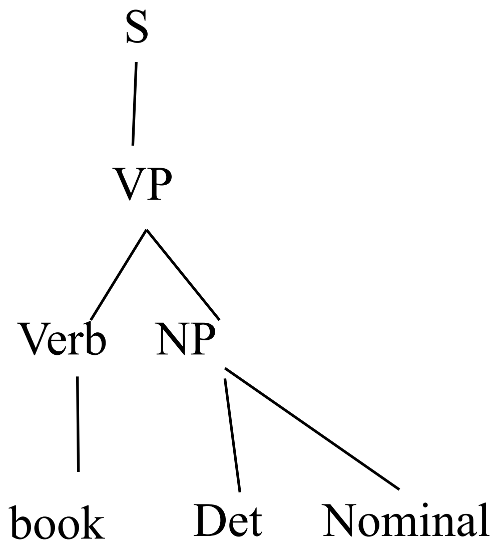
Top Down Parsing



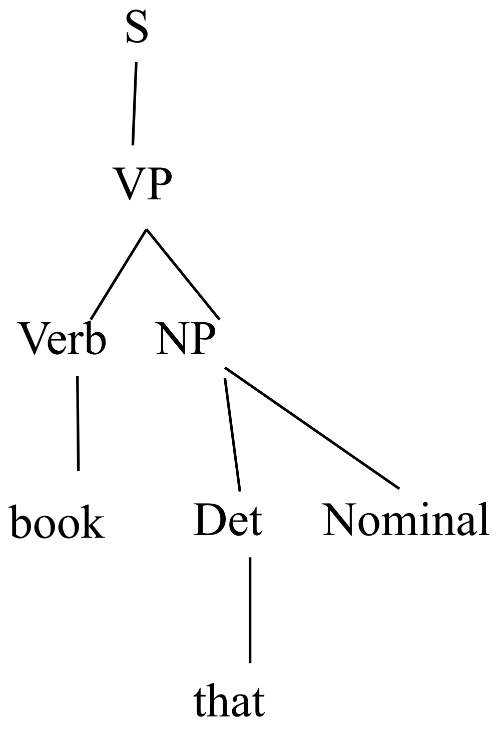
Top Down Parsing



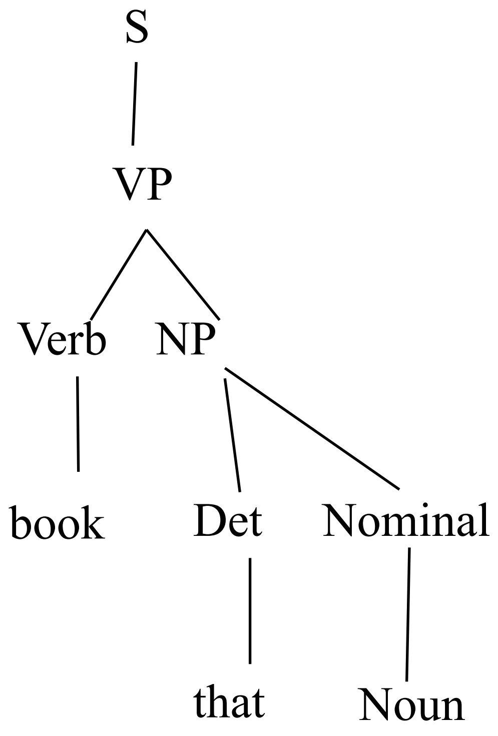
Top Down Parsing



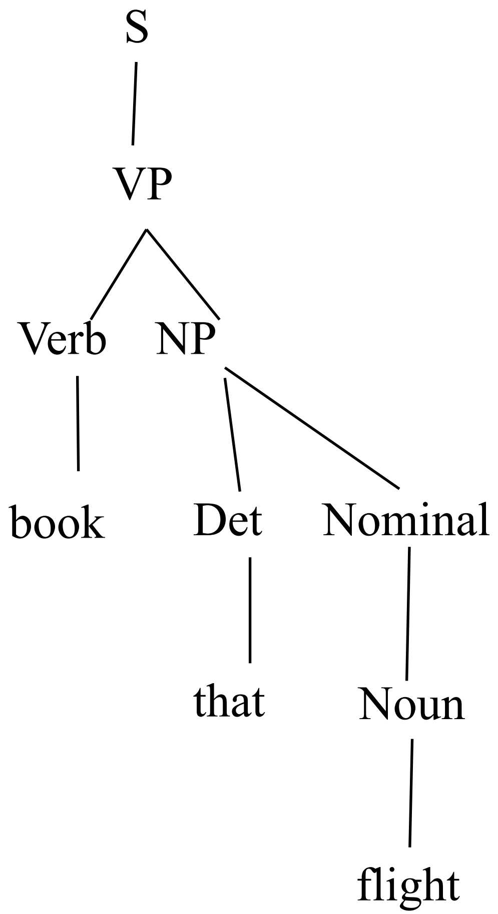
Top Down Parsing



Top Down Parsing



Top Down Parsing



Bottom Up Parsing

- Start searching space of reverse derivations from the terminal symbols in the string.

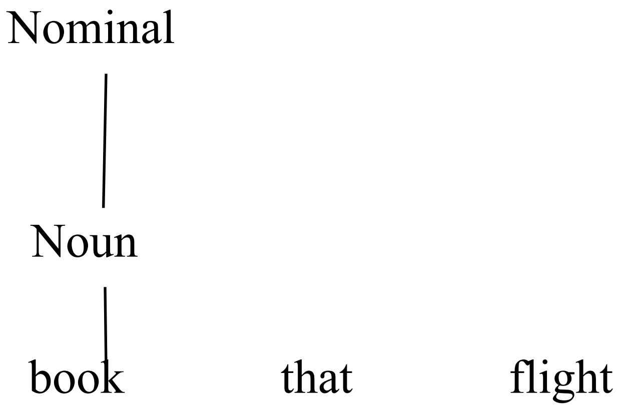
book that flight

Bottom Up Parsing

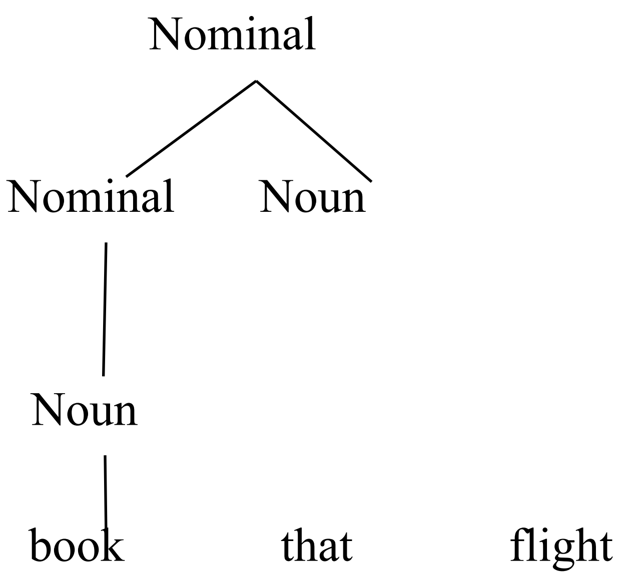
Noun

book that flight

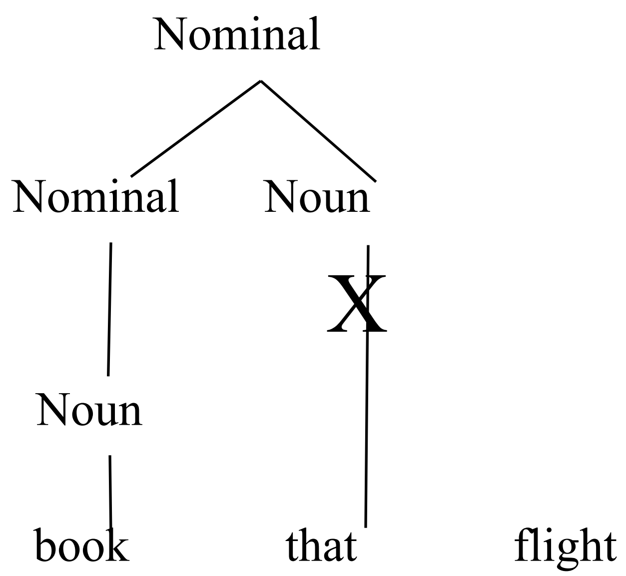
Bottom Up Parsing



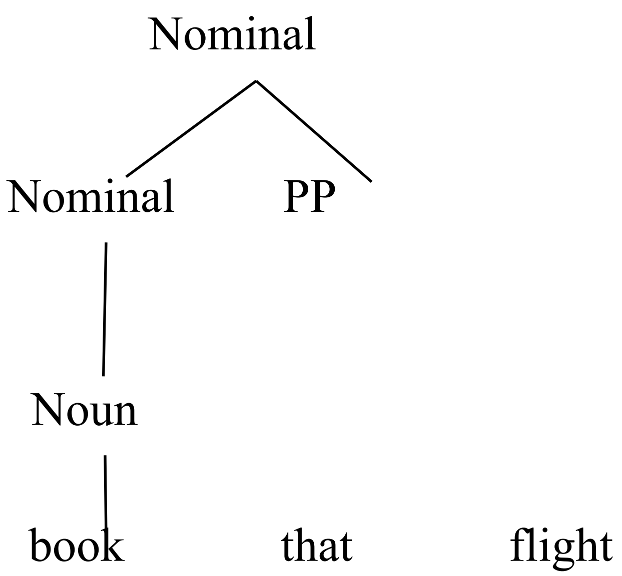
Bottom Up Parsing



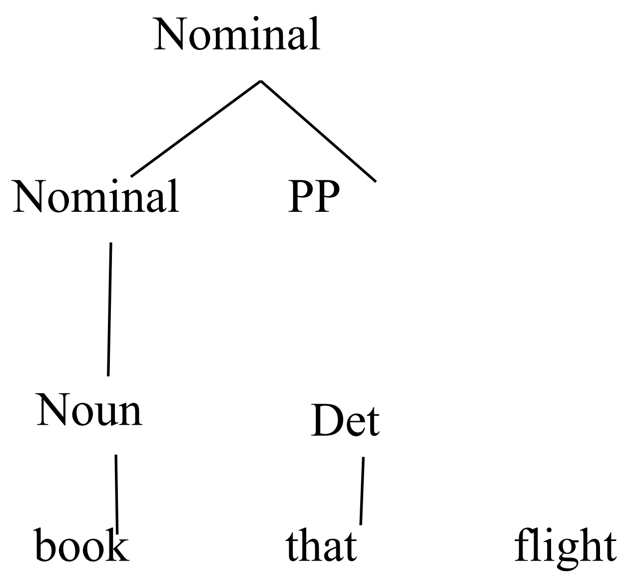
Bottom Up Parsing



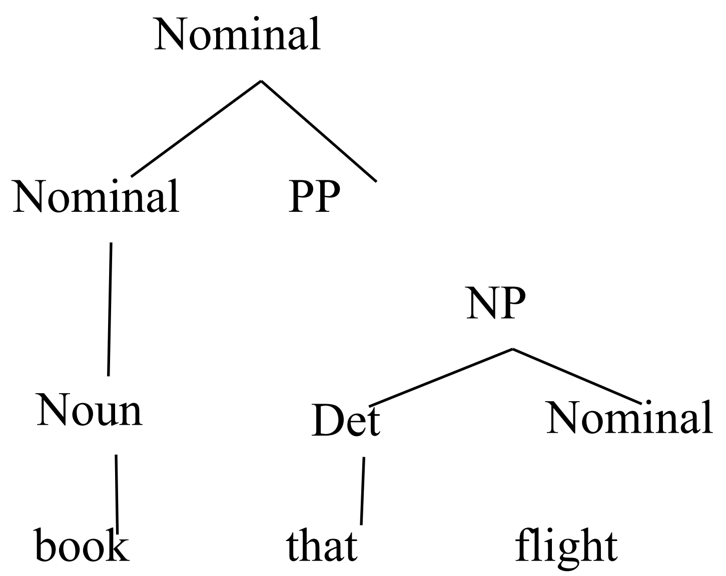
Bottom Up Parsing



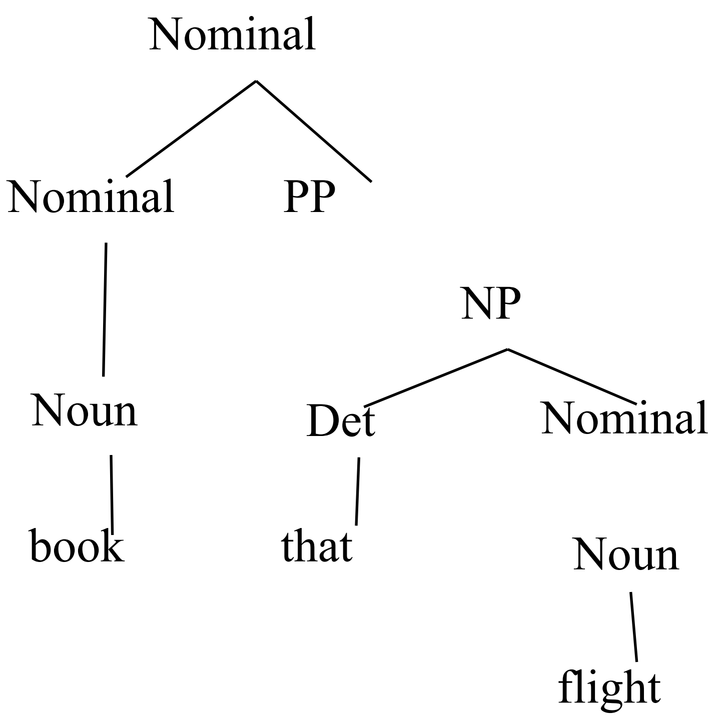
Bottom Up Parsing



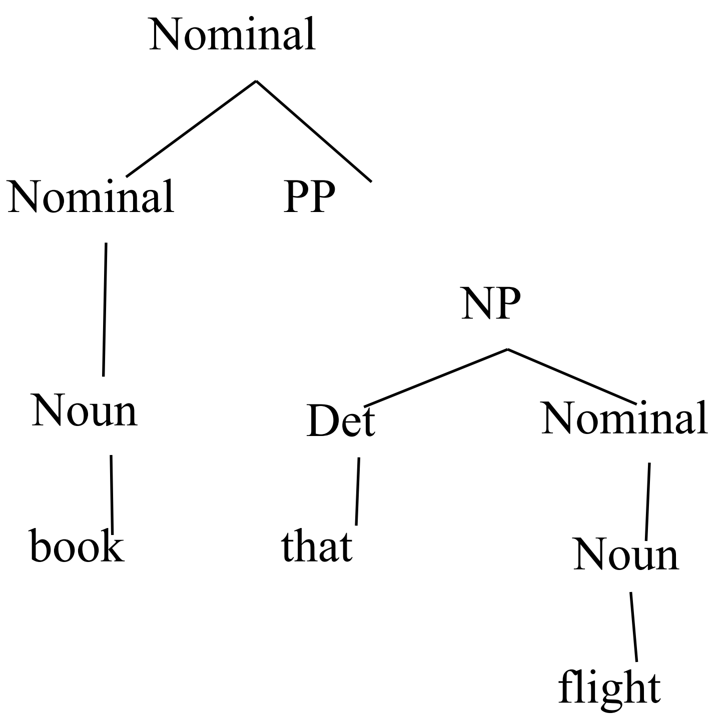
Bottom Up Parsing



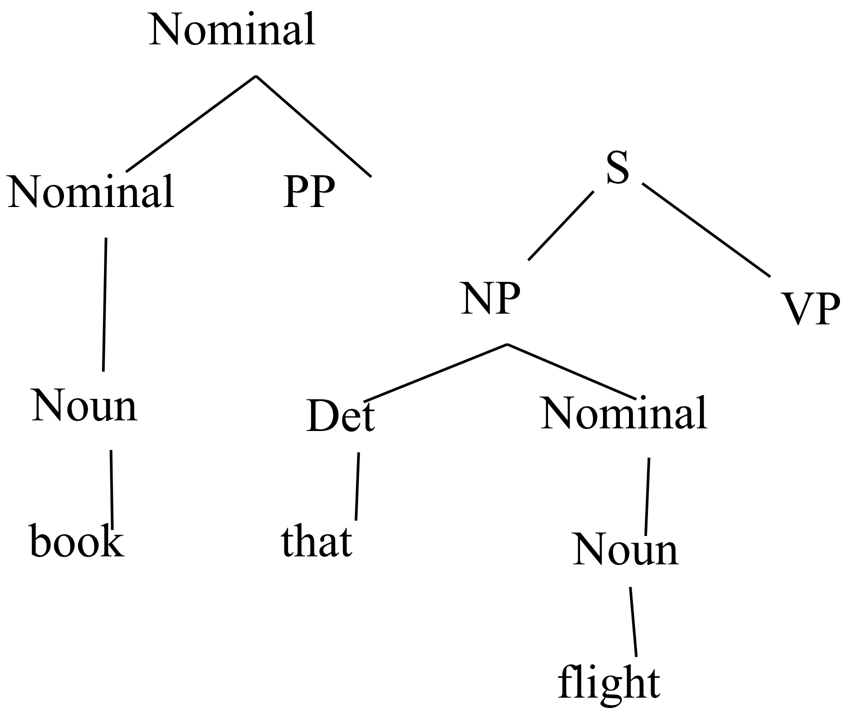
Bottom Up Parsing



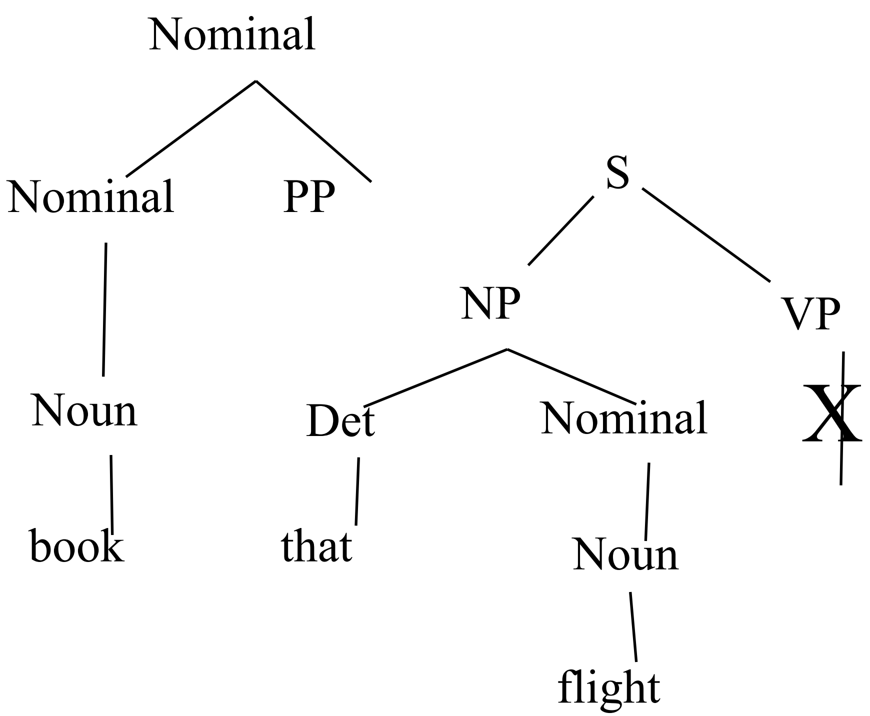
Bottom Up Parsing



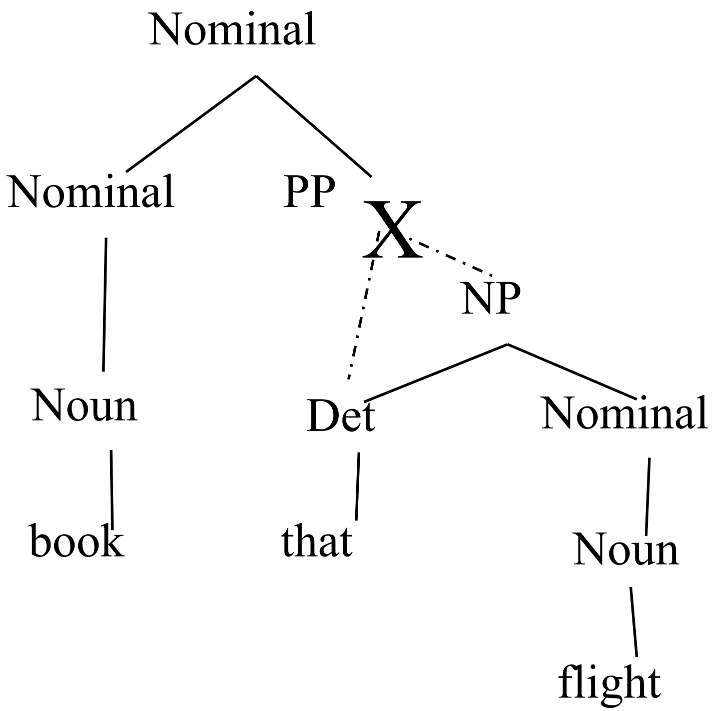
Bottom Up Parsing



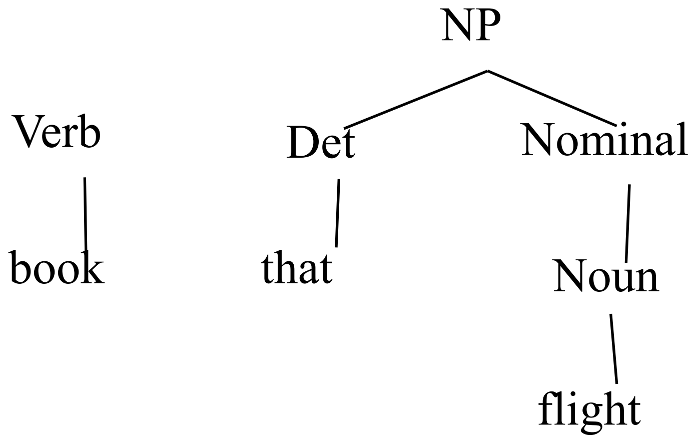
Bottom Up Parsing



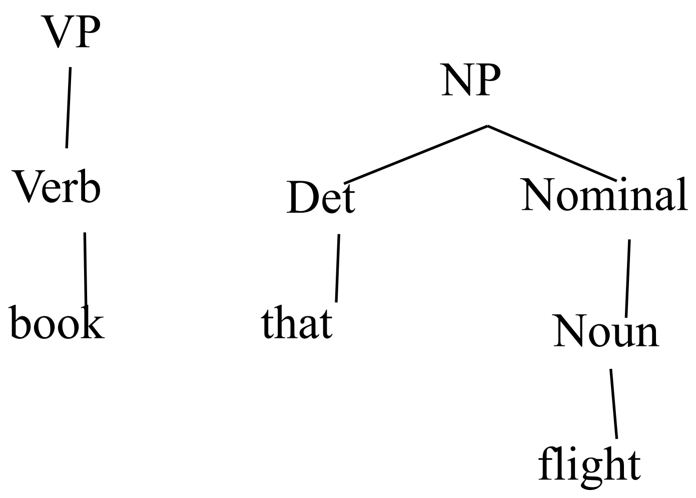
Bottom Up Parsing



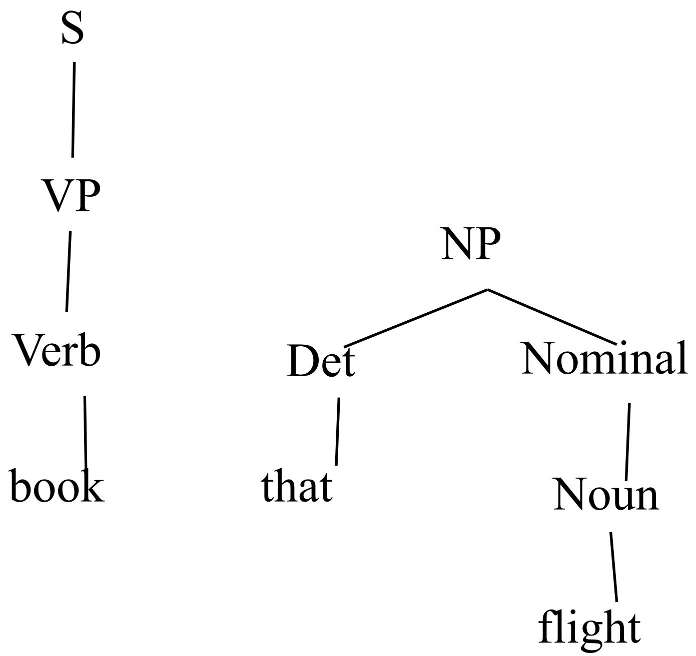
Bottom Up Parsing



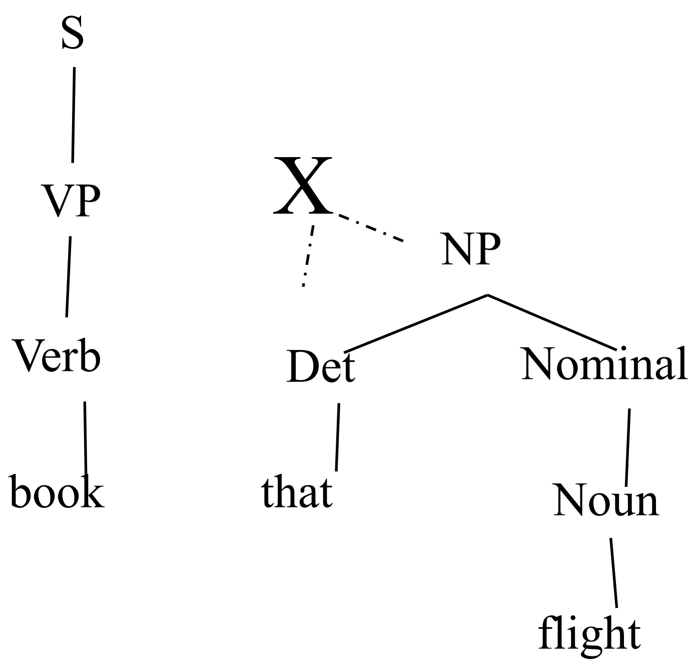
Bottom Up Parsing



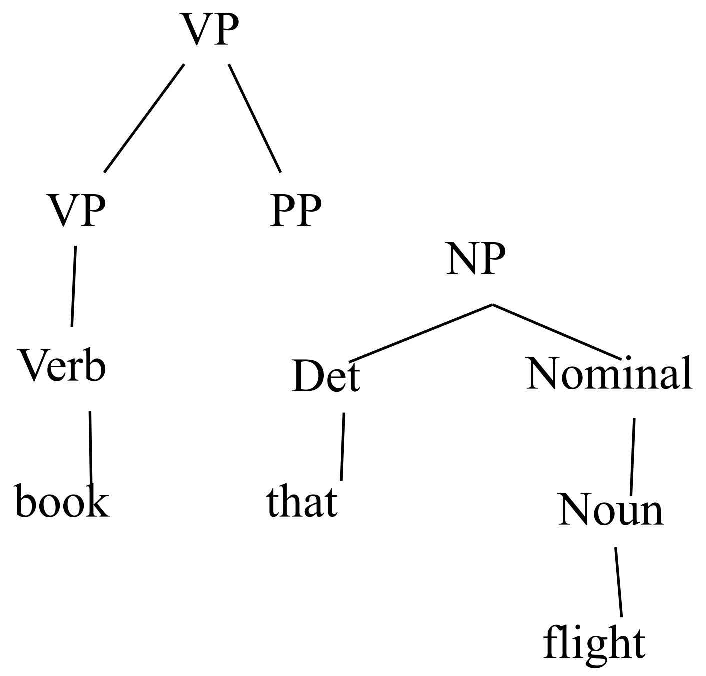
Bottom Up Parsing



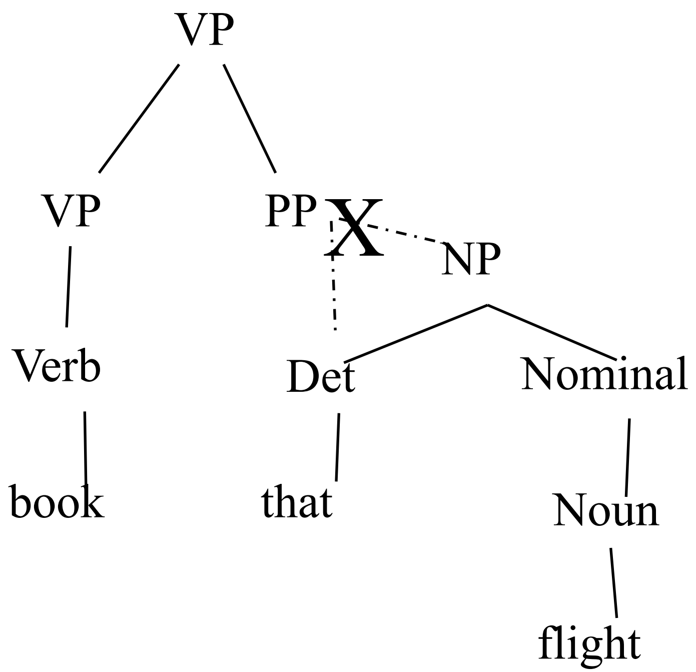
Bottom Up Parsing



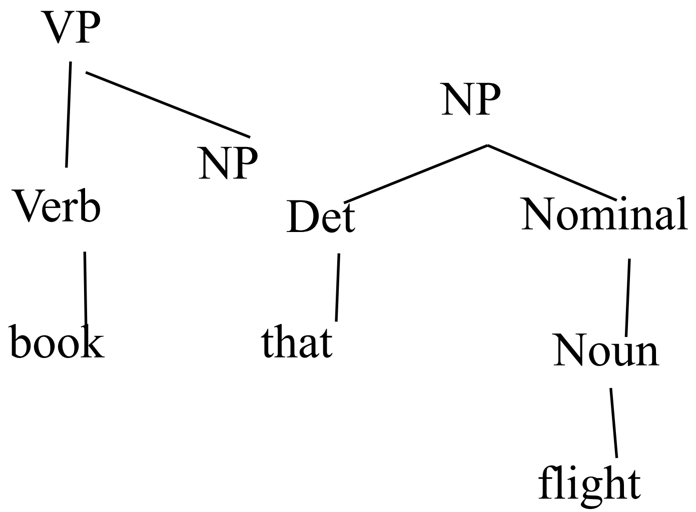
Bottom Up Parsing



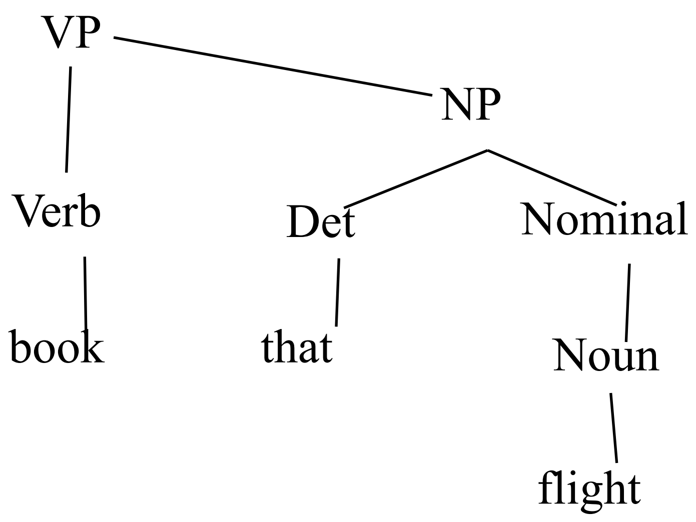
Bottom Up Parsing



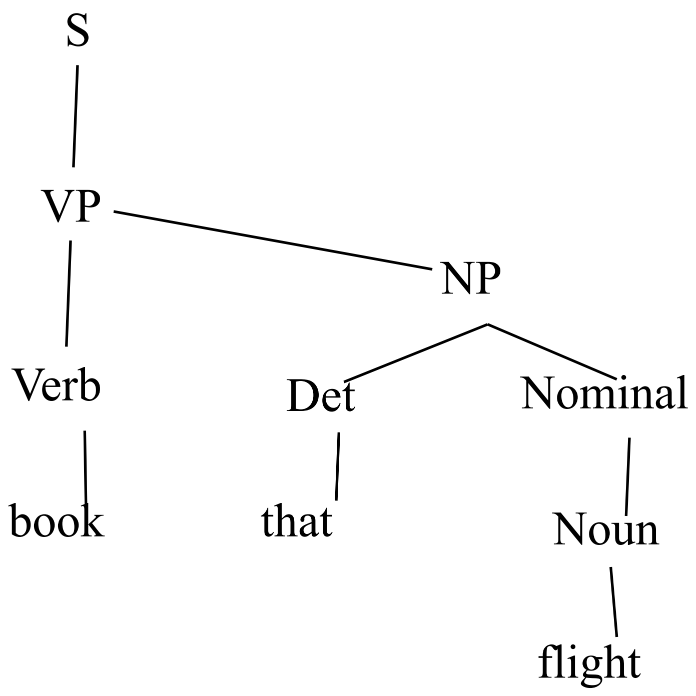
Bottom Up Parsing



Bottom Up Parsing



Bottom Up Parsing



Top Down vs. Bottom Up

- Top down never explores options that will not lead to a full parse, but can explore many options that never connect to the actual sentence.
- Bottom up never explores options that do not connect to the actual sentence but can explore options that can never lead to a full parse.
- Relative amounts of wasted search depend on how much the grammar branches in each direction.

Syntax

CYK Algorithm

Dynamic Programming Parsing

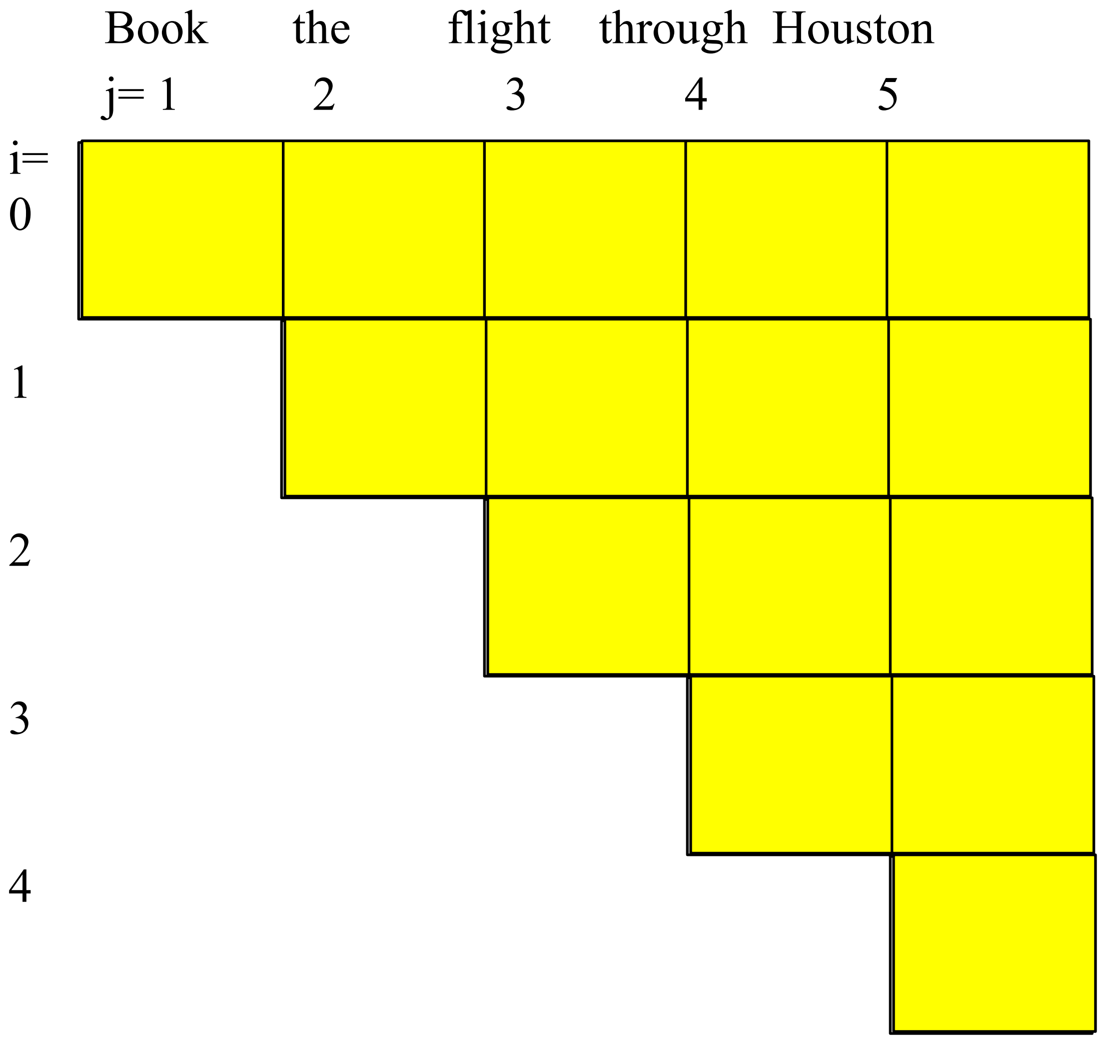
- CKY (Cocke-Kasami-Younger) algorithm based on bottom-up parsing and requires first normalizing the grammar.
  - First grammar must be converted to **Chomsky normal form (CNF)** in which productions must have either exactly 2 non-terminal symbols on the RHS or 1 terminal symbol (lexicon rules).
  - Parse bottom-up storing phrases formed from all substrings in a triangular table (chart).

ATIS English Grammar Conversion

Original Grammar	Chomsky Normal Form
S → NP VP	S → NP VP
S → Aux NP VP	S → X1 VP
	X1 → Aux NP
S → VP	S → book | include | prefer
	S → Verb NP
	S → VP PP
NP → Pronoun	NP → I | he | she | me
NP → Proper-Noun	NP → Houston | NWA
NP → Det Nominal	NP → Det Nominal
Nominal → Noun	Nominal → book | flight | meal | money
Nominal → Nominal Noun	Nominal → Nominal Noun
Nominal → Nominal PP	Nominal → Nominal PP
VP → Verb	VP → book | include | prefer
VP → Verb NP	VP → Verb NP
VP → VP PP	VP → VP PP
PP → Prep NP	PP → Prep NP

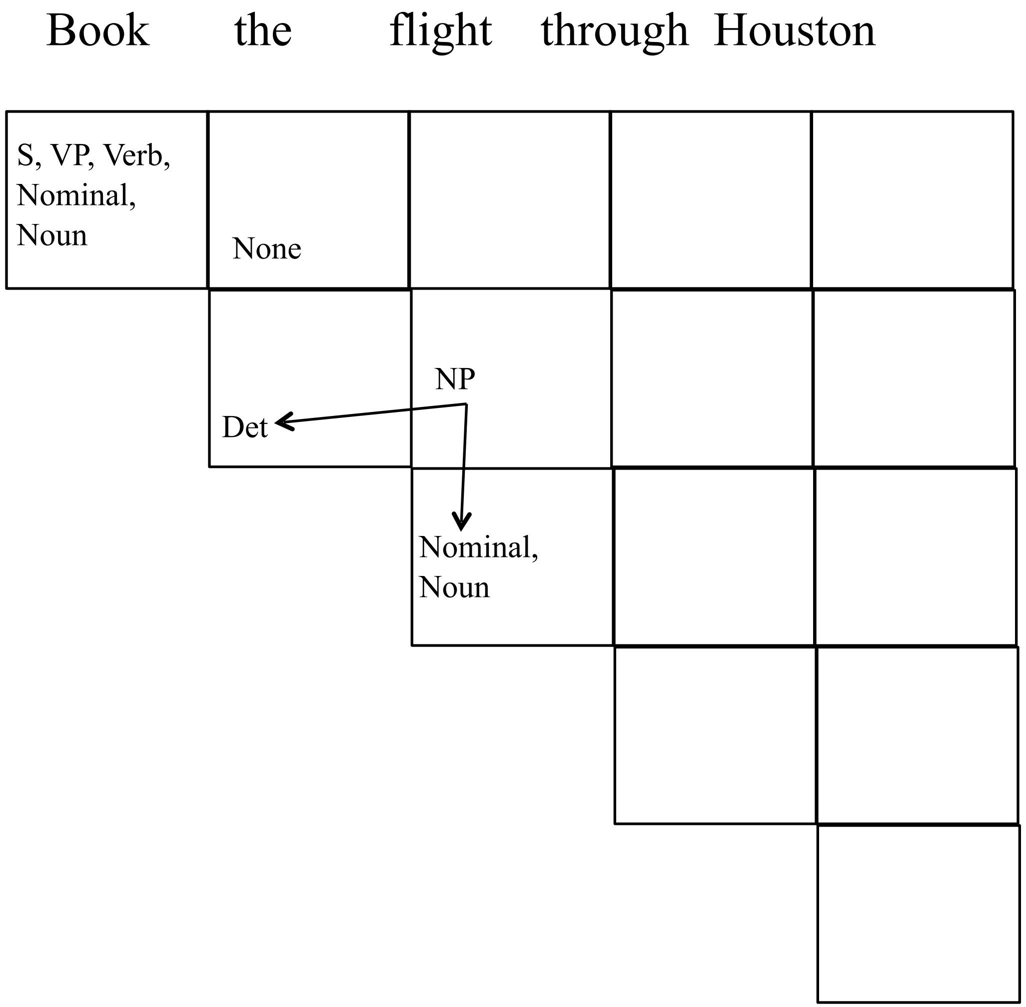
Note that, although not shown here, original grammar contain all the lexical entires.

CKY Parser



Cell[i,j] contains all constituents (non-terminals) covering words i +1 through j

CKY Parser


Book
the
flight
through
Houston
S, VP, Verb, Nominal, Noun
None
NP
Det
Nominal, Noun

CKY Parser

Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun	None	VP		
	Det	NP		
		Nominal, Noun		

CKY Parser

Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun	None	S VP		
	Det	NP		
		Nominal, Noun		

CKY Parser

Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun	None	S VP		
	Det	NP		
		Nominal, Noun		

CKY Parser

Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun	None	S VP	None	
	Det	NP	None	
		Nominal, Noun	None	
			Prep	

CKY Parser

Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun	None	S VP	None	
	Det	NP	None	
		Nominal, Noun	None	
			Prep	PP
				NP ProperNoun

CKY Parser

Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun	None	S VP	None	
	Det	NP	None	
		Nominal, Noun	None	Nominal
			Prep	PP
				NP ProperNoun

CKY Parser

Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun	None	S VP	None	
	Det	NP	None	NP
		Nominal, Noun	None	Nominal
			Prep	PP
				NP ProperNoun

CKY Parser

Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun	None	S VP	None	VP
	Det	NP	None	NP
		Nominal, Noun	None	Nominal
			Prep	PP
				NP ProperNoun

CKY Parser

Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun	None	S VP	None	S VP
	Det	NP	None	NP
		Nominal, Noun	None	Nominal
			Prep	PP
				NP ProperNoun

CKY Parser

Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun	None	S VP	None	VP S VP
	Det	NP	None	NP
		Nominal, Noun	None	Nominal
			Prep	PP
				NP ProperNoun

CKY Parser

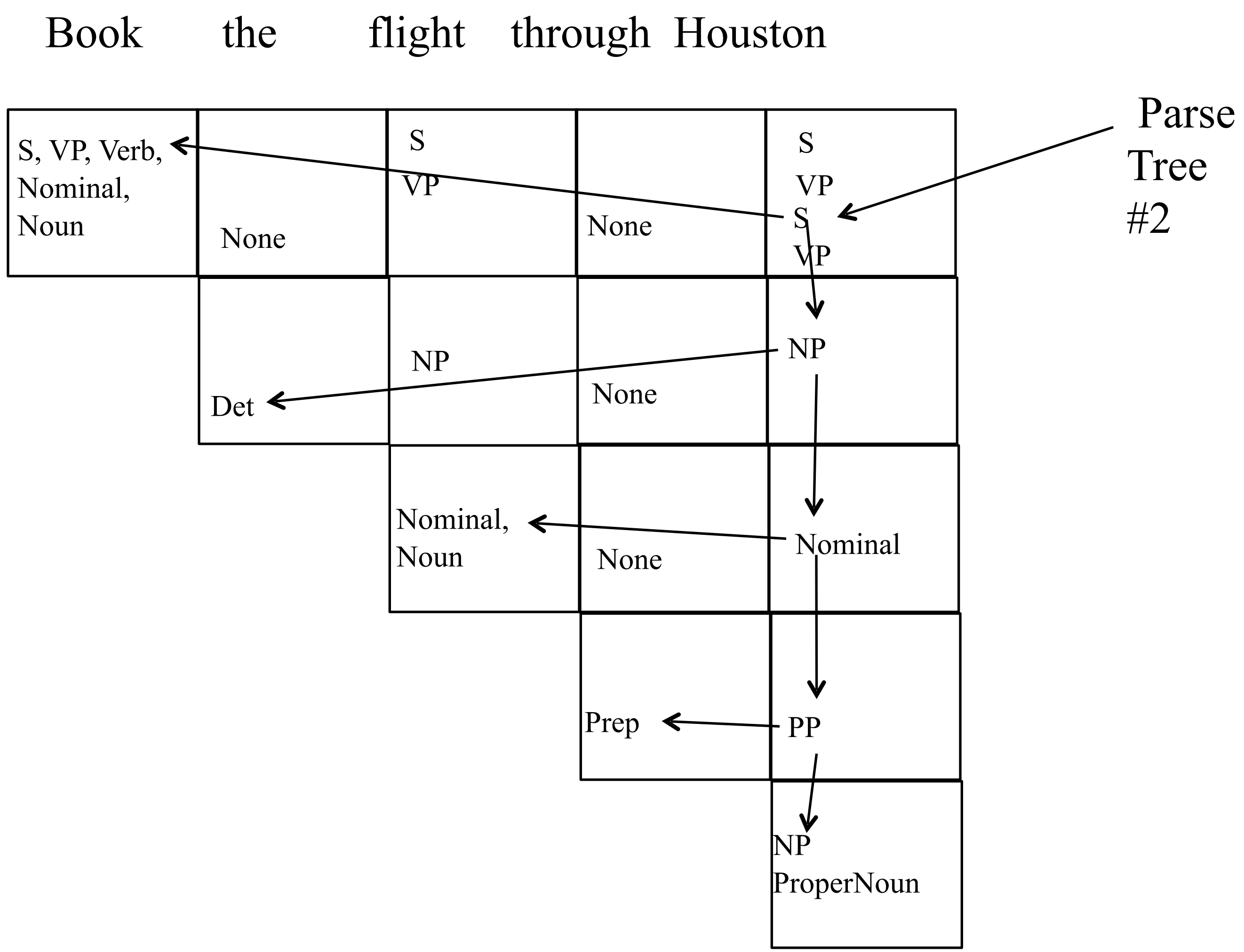
Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun	None	S VP	None	S VP S VP
	Det	NP	None	NP
		Nominal, Noun	None	Nominal
			Prep	PP
				NP ProperNoun

CKY Parser

Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun	None	S VP	None	S VP S VP
	Det	NP	None	NP
		Nominal, Noun	None	Nominal
			Prep	PP
				NP ProperNoun

Parse Tree #1

CKY Parser


Book the flight through Houston
S, VP, Verb, Nominal, Noun
None
S
VP
None
S
VP
S
VP
Parse Tree #2
Det
NP
None
NP
Nominal, Noun
None
Nominal
Prep
PP
NP
ProperNoun

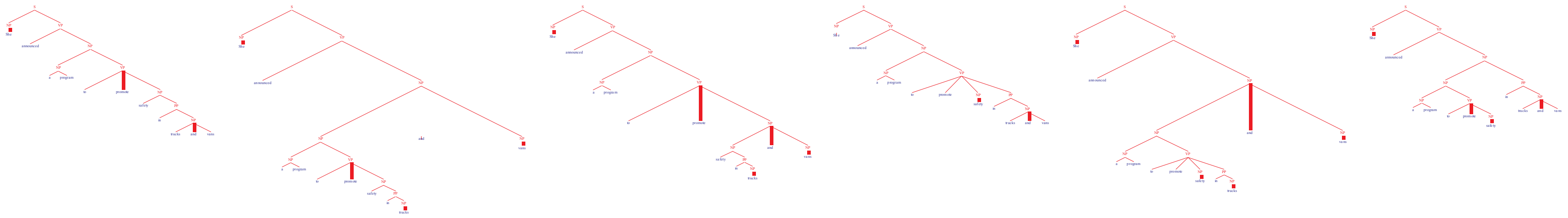
The Problem with Parsing: Ambiguity

INPUT:

She announced a program to promote safety in trucks and vans

+

POSSIBLE OUTPUTS:



And there are more...

Syntax

Probabilistic Context Free Grammars (PCFG)

Overview

- Probabilistic Context-Free Grammars (PCFGs)
- The CKY Algorithm for parsing with PCFGs

A Probabilistic Context-Free Grammar (PCFG)

S	$\Rightarrow$	NP	VP	1.0
VP	$\Rightarrow$	Vi		0.4
VP	$\Rightarrow$	Vt	NP	0.4
VP	$\Rightarrow$	VP	PP	0.2
NP	$\Rightarrow$	DT	NN	0.3
NP	$\Rightarrow$	NP	PP	0.7
PP	$\Rightarrow$	P	NP	1.0

Vi	$\Rightarrow$	sleeps	1.0
Vt	$\Rightarrow$	saw	1.0
NN	$\Rightarrow$	man	0.7
NN	$\Rightarrow$	woman	0.2
NN	$\Rightarrow$	telescope	0.1
DT	$\Rightarrow$	the	1.0
IN	$\Rightarrow$	with	0.5
IN	$\Rightarrow$	in	0.5

- Probability of a tree t with rules

$$\alpha_1 \to \beta_1, \alpha_2 \to \beta_2, \ldots, \alpha_n \to \beta_n$$

is $p(t) = \prod_{i=1}^{n} q(\alpha_i \to \beta_i)$ where $q(\alpha \to \beta)$ is the probability for rule $\alpha \to \beta$.

DERIVATION	RULES USED	PROBABILITY
S		

DERIVATION	RULES USED	PROBABILITY
S		1.0
	S $\rightarrow$ NP VP	
NP VP		

DERIVATION	RULES USED	PROBABILITY
S		1.0
	S $\rightarrow$ NP VP	
NP VP		0.3
	NP $\rightarrow$ DT NN	
DT NN VP		

DERIVATION	RULES USED	PROBABILITY
S		1.0
	S $\rightarrow$ NP VP	
NP VP		0.3
	NP $\rightarrow$ DT NN	
DT NN VP		1.0
	DT $\rightarrow$ the	
the NN VP		

DERIVATION	RULES USED	PROBABILITY
S		1.0
	S $\rightarrow$ NP VP	
NP VP		0.3
	NP $\rightarrow$ DT NN	
DT NN VP		1.0
	DT $\rightarrow$ the	
the NN VP		0.1
	NN $\rightarrow$ dog	
the dog VP		

DERIVATION	RULES USED	PROBABILITY
S	S $\rightarrow$ NP VP	1.0
NP VP	NP $\rightarrow$ DT NN	0.3
DT NN VP	DT $\rightarrow$ the	1.0
the NN VP	NN $\rightarrow$ dog	0.1
the dog VP	VP $\rightarrow$ Vi	0.4
the dog Vi		

DERIVATION	RULES USED	PROBABILITY
S	$S \rightarrow NP\ VP$	1.0
NP VP	$NP \rightarrow DT\ NN$	0.3
DT NN VP	$DT \rightarrow$ the	1.0
the NN VP	$NN \rightarrow$ dog	0.1
the dog VP	$VP \rightarrow Vi$	0.4
the dog Vi	$Vi \rightarrow$ laughs	0.5
the dog laughs		

Properties of PCFGs

- Assigns a probability to each *left-most derivation*, or parse-tree, allowed by the underlying CFG

Properties of PCFGs

- Assigns a probability to each *left-most derivation*, or parse-tree, allowed by the underlying CFG
- Say we have a sentence s, set of derivations for that sentence is $\mathcal{T}(s)$. Then a PCFG assigns a probability $p(t)$ to each member of $\mathcal{T}(s)$. i.e., *we now have a ranking in order of probability.*

Properties of PCFGs

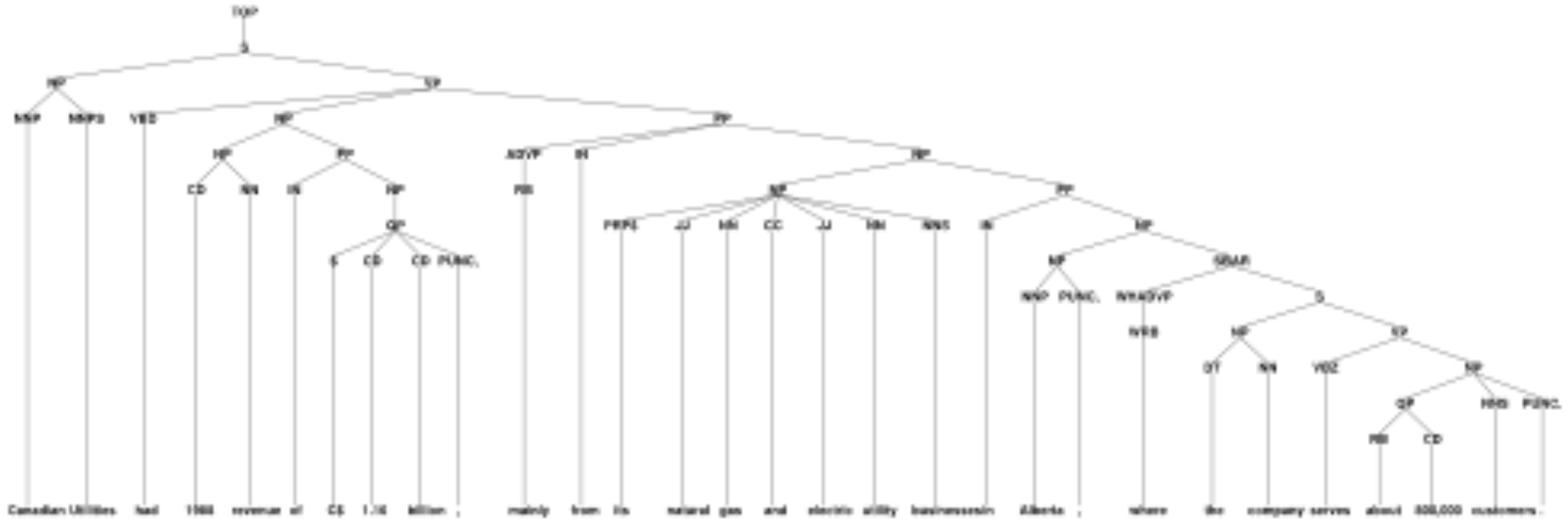
- Assigns a probability to each *left-most derivation*, or parse-tree, allowed by the underlying CFG
- Say we have a sentence s, set of derivations for that sentence is $\mathcal{T}(s)$. Then a PCFG assigns a probability $p(t)$ to each member of $\mathcal{T}(s)$. i.e., *we now have a ranking in order of probability.*
- The most likely parse tree for a sentence s is

$$\arg\max_{t \in \mathcal{T}(s)} p(t)$$

Data for Parsing Experiments: Treebanks

- Penn WSJ Treebank = 50,000 sentences with associated trees
- Usual set-up: 40,000 training sentences, 2400 test sentences

An example tree:



Deriving a PCFG from a Treebank

- Given a set of example trees (a treebank), the underlying CFG can simply be **all rules seen in the corpus**
- Maximum Likelihood estimates:

$$q_{ML}(\alpha \rightarrow \beta) = \frac{\text{Count}(\alpha \rightarrow \beta)}{\text{Count}(\alpha)}$$

where the counts are taken from a training set of example trees.

Parsing with a PCFG

- Given a PCFG and a sentence s, define $\mathcal{T}(s)$ to be the set of trees with s as the yield.
- Given a PCFG and a sentence s, how do we find

$$\arg\max_{t \in \mathcal{T}(s)} p(t)$$

Chomsky Normal Form

A context free grammar $G = (N, \Sigma, R, S)$ in Chomsky Normal Form is as follows

- N is a set of non-terminal symbols
- Σ is a set of terminal symbols
- R is a set of rules which take one of two forms:
  - $X \rightarrow Y_1 Y_2$ for $X \in N$, and $Y_1, Y_2 \in N$
  - $X \rightarrow Y$ for $X \in N$, and $Y \in \Sigma$
- $S \in N$ is a distinguished start symbol

A Dynamic Programming Algorithm

- Given a PCFG and a sentence s, how do we find

$$\max_{t \in \mathcal{T}(s)} p(t)$$

- Notation:

$$n = \text{number of words in the sentence}$$
$$w_i = i\text{'th word in the sentence}$$
$$N = \text{the set of non-terminals in the grammar}$$
$$S = \text{the start symbol in the grammar}$$

- Define a dynamic programming table

$$\pi[i, j, X] = \text{maximum probability of a constituent with non-terminal } X \text{ spanning words } i \ldots j \text{ inclusive}$$

- Our goal is to calculate $\max_{t \in \mathcal{T}(s)} p(t) = \pi[1, n, S]$

A Dynamic Programming Algorithm

- Base case definition: for all $i = 1 \ldots n$, for $X \in N$

$$\pi[i, i, X] = q(X \to w_i)$$

(note: define $q(X \to w_i) = 0$ if $X \to w_i$ is not in the grammar)

- Recursive definition: for all $i = 1 \ldots n$, $j = (i + 1) \ldots n$, $X \in N$,

$$\pi(i, j, X) = \max_{\substack{X \to Y Z \in R, \\ s \in \{i \ldots (j-1)\}}} \left(q(X \to Y Z) \times \pi(i, s, Y) \times \pi(s + 1, j, Z) \right)$$

split point

An Example

$$\pi(i,j,X) = \max_{\substack{X \to Y\,Z \in R,\\ s \in \{i \ldots (j-1)\}}} \left(q(X \to Y\,Z) \times \pi(i,s,Y) \times \pi(s+1,j,Z)\right)$$

the dog saw the man with the telescope

The Full Dynamic Programming Algorithm

$O(n^3|N|^3)$

Input: a sentence $s = x_1 \ldots x_n$, a PCFG $G = (N, \Sigma, S, R, q)$.
Initialization:
For all $i \in \{1 \ldots n\}$, for all $X \in N$,

$$\pi(i, i, X) = \begin{cases} q(X \rightarrow x_i) & \text{if } X \rightarrow x_i \in R \\ 0 & \text{otherwise} \end{cases}$$

Algorithm:

- For $l = 1 \ldots (n-1)$
  - For $i = 1 \ldots (n-l)$
    - Set $j = i + l$
    - For all $X \in N$, calculate

$O(n^2)$ for l, i choices

$$\pi(i, j, X) = \max_{\substack{X \rightarrow Y Z \in R, \\ s \in \{i \ldots (j-1)\}}} \left(q(X \rightarrow Y Z) \times \pi(i, s, Y) \times \pi(s+1, j, Z)\right)$$

and

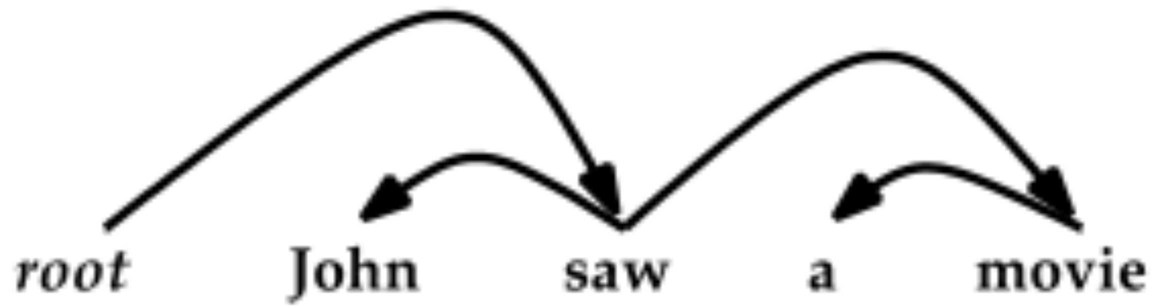
$$bp(i, j, X) = \arg \max_{\substack{X \rightarrow Y Z \in R, \\ s \in \{i \ldots (j-1)\}}} \left(q(X \rightarrow Y Z) \times \pi(i, s, Y) \times \pi(s+1, j, Z)\right)$$

Summary

- PCFGs augments CFGs by including a probability for each rule in the grammar.
- The probability for a parse tree is the product of probabilities for the rules in the tree
- To build a PCFG-parsed parser:
  1. Learn a PCFG from a treebank
  2. Given a test data sentence, use the CKY algorithm to compute the highest probability tree for the sentence under the PCFG

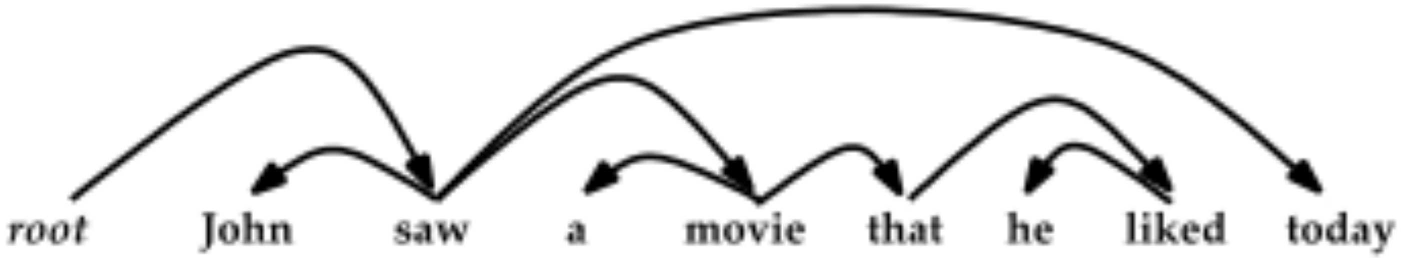
Dependency Parsing

Unlabeled Dependency Parses



- `root` is a special *root* symbol
- Each dependency is a pair (h, m) where h is the index of a head word, m is the index of a modifier word. In the figures, we represent a dependency (h, m) by a directed edge from h to m.
- Dependencies in the above example are $(0, 2)$, $(2, 1)$, $(2, 4)$, and $(4, 3)$. (We take 0 to be the root symbol.)

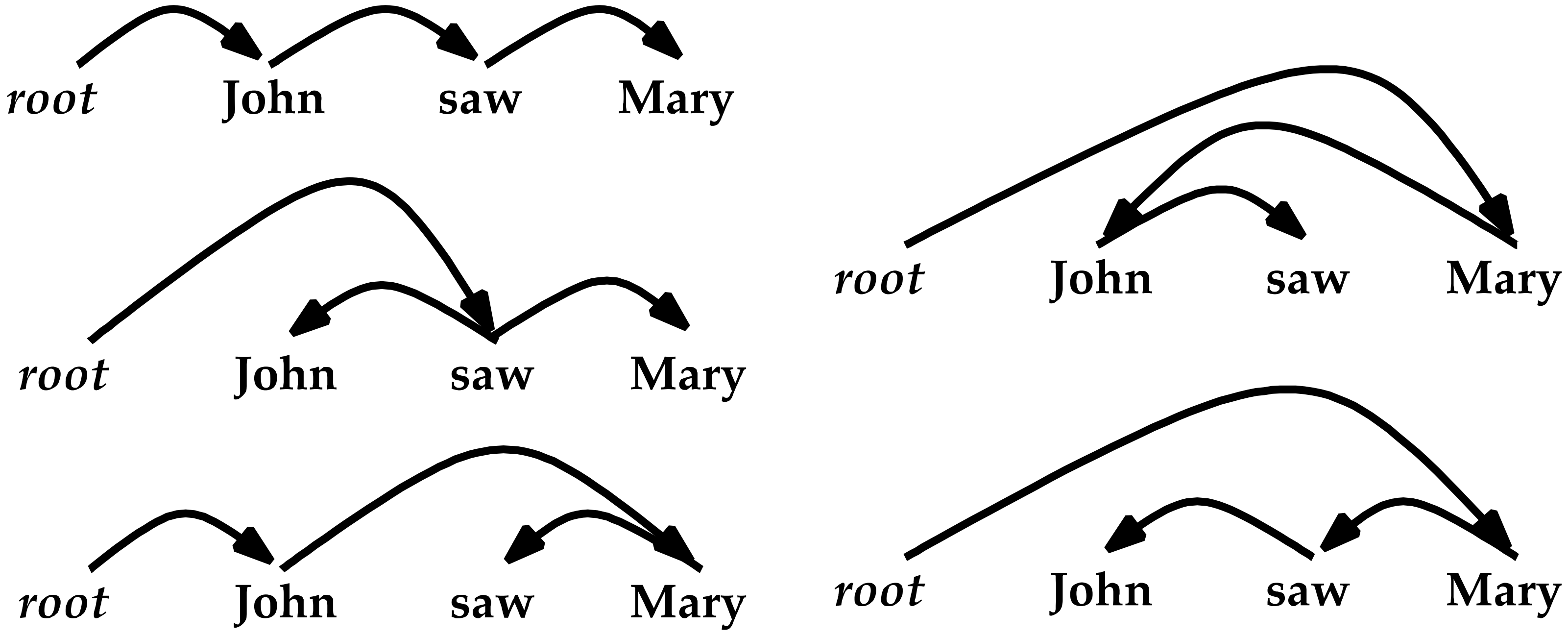
Conditions on Dependency Structures



- The dependency arcs form a *directed tree*, with the `root` symbol at the root of the tree.
  (Definition: A directed tree rooted at *root* is a tree, where for every word w other than the root, there is a directed path from *root* to w.)

- There are no "crossing dependencies".
  Dependency structures with no crossing dependencies are sometimes referred to as **projective** structures.

All Dependency Parses for *John saw Mary*



Dependency Parsing Resources

- CoNLL 2006 conference had a "shared task" with dependency parsing of 12 languages (Arabic, Chinese, Czech, Danish, Dutch, German, Japanese, Portuguese, Slovene, Spanish, Swedish, Turkish). 19 different groups developed dependency parsing systems. (See also CoNLL 2007).

- PhD thesis on the topic: Ryan McDonald, *Discriminative Training and Spanning Tree Algorithms for Dependency Parsing*, University of Pennsylvania.

- For some languages, e.g., Czech, there are "dependency banks" available which contain training data in the form of sentences paired with dependency structures

- For other languages, we can extract dependency structures from treebanks

Efficiency of Dependency Parsing

- PCFG parsing is $O(n^3G^3)$ where n is the length of the sentence, G is the number of non-terminals in the grammar
- Lexicalized PCFG parsing is $O(n^5G^3)$ where n is the length of the sentence, G is the number of non-terminals in the grammar.
- Unlabeled dependency parsing is $O(n^3)$.

GLMs for Dependency parsing

- x is a sentence
- $\mathbf{GEN}(x)$ is set of all dependency structures for x
- $\mathbf{f}(x, y)$ is a feature vector for a sentence x paired with a dependency parse y

GLMs for Dependency parsing

- To run the perceptron algorithm, we must be able to efficiently calculate

$$\arg\max_{y \in \mathbf{GEN}(x)} \mathbf{w} \cdot \mathbf{f}(x, y)$$

- Local feature vectors: define

$$\mathbf{f}(x, y) = \sum_{(h,m) \in y} \mathbf{g}(x, h, m)$$

where $\mathbf{g}(x, h, m)$ maps a sentence x and a dependency (h, m) to a local feature vector

- Can then use dynamic programming to calculate

$$\arg\max_{y \in \mathbf{GEN}(x)} \mathbf{w} \cdot \mathbf{f}(x, y) = \arg\max_{y \in \mathbf{GEN}(x)} \sum_{(h,m) \in y} \mathbf{w} \cdot \mathbf{g}(x, h, m)$$

Definition of Local Feature Vectors

- Features from McDonald et al. (2005):
  - Note: define w_i to be the i'th word in the sentence, t_i to be the part-of-speech (POS) tag for the i'th word.
  - *Unigram* features: Identity of w_h. Identity of w_m. Identity of t_h. Identity of t_m.
  - *Bigram* features: Identity of the 4-tuple $\langle w_h, w_m, t_h, t_m \rangle$. Identity of sub-sets of this 4-tuple, e.g., identity of the pair $\langle w_h, w_m \rangle$.
  - *Contextual features:* Identity of the 4-tuple $\langle t_h, t_{h+1}, t_{m-1}, t_m \rangle$. Similar features which consider t_{h-1} and t_{m+1}, giving 4 possible feature types.
  - *In-between features:* Identity of triples $\langle t_h, t, t_m \rangle$ for any tag t seen between words h and m.

Results from McDonald (2005)

Method	Accuracy
Collins (1997)	91.4%
1st order dependency	90.7%
2nd order dependency	91.5%

- Accuracy is percentage of correct unlabeled dependencies
- Collins (1997) is result from a lexicalized context-free parser, with dependencies extracted from the parser's output
- 1st order dependency is the method just described.
  2nd order dependency is a model that uses richer representations.
- Advantages of the dependency parsing approaches: simplicity, efficiency ($O(n^3)$ parsing time).