Constituency Grammars

CL1: Jordan Boyd-Graber

University of Maryland

October 14, 2013

Adapted from material by Michael Collins

Outline

1. Motivation

2. Context Free Grammars

3. Probabilistic Context Free Grammars
   - Parameterization: Defining Score Function
   - Estimation
   - Parsing

A More Grounded Syntax Theory

- A central question in linguistics is **how do we know when a sentence is grammatical**?
- Chomsky's generative grammars attempted to mathematically formalize this question
- Linguistic phrases contained a universal, hierarchical structure formalized as parse trees

A More Grounded Syntax Theory

- A central question in linguistics is **how do we know when a sentence is grammatical**?
- Chomsky's generative grammars attempted to mathematically formalize this question
- Linguistic phrases contained a universal, hierarchical structure formalized as parse trees
- Today
  - A formalization
  - Foundation of all computational syntax
  - Learnable from data

Outline

1. Motivation
2. Context Free Grammars
3. Probabilistic Context Free Grammars
   - Parameterization: Defining Score Function
   - Estimation
   - Parsing

Context Free Grammars

Definition

- N: finite set of non-terminal symbols
- Σ: finite set of terminal symbols
- R: productions of the form $X \rightarrow Y_1 \dots Y_n$, where $X \in N$, $Y \in (N \cup \Sigma)$
- S: a start symbol within N

Examples of non-terminals:

- NP for "noun phrase"
- VP for "verb phrase"
- Often correspond to multiword syntactic abstractions

Context Free Grammars

Definition

- N: finite set of non-terminal symbols
- Σ: finite set of terminal symbols
- R: productions of the form $X \rightarrow Y_1 \dots Y_n$, where $X \in N$, $Y \in (N \cup \Sigma)$
- S: a start symbol within N

Examples of terminals:

- "*dog*"
- "*play*"
- "*the*"

Context Free Grammars

Definition

- N: finite set of non-terminal symbols
- Σ: finite set of terminal symbols
- R: productions of the form $X \rightarrow Y_1 \dots Y_n$, where $X \in N$, $Y \in (N \cup \Sigma)$
- S: a start symbol within N

Examples of productions:

- $\mathrm{N} \quad \rightarrow$ "*dog*"
- $\mathrm{NP} \quad \rightarrow \mathrm{N}$
- $\mathrm{NP} \quad \rightarrow \mathrm{ADJ} \quad \mathrm{N}$

Context Free Grammars

Definition

- N: finite set of non-terminal symbols
- Σ: finite set of terminal symbols
- R: productions of the form $X \rightarrow Y_1 \dots Y_n$, where $X \in N$, $Y \in (N \cup \Sigma)$
- S: a start symbol within N

In NLP applications, by convention we use S as the start symbol

Flexibility of CFG Productions

- Unary rules: $\textsc{nn} \rightarrow$ "*man*"
- Mixing terminals and nonterminals on RHS:
  - $\textsc{np} \rightarrow$ "*Congress*" $\textsc{Vt}$ "*the*" "*pooch*"
  - $\textsc{np} \rightarrow$ "*the*" $\textsc{nn}$
- Empty terminals
  - $\textsc{np} \rightarrow \epsilon$
  - $\textsc{adj} \rightarrow \epsilon$

Derivations

- A derivation is a sequence of strings $s_1 \dots s_T$ where
- $s_1 \equiv S$, the start symbol
- $s_T \in \Sigma^*$: i.e., the final string is only terminals
- $s_i, \forall i > 1$, is derived from s_{i-1} by replacing some non-terminal X in s_{i-1} and replacing it by some β, where $x \rightarrow \beta \in R$.

Derivations

- A derivation is a sequence of strings $s_1 \dots s_T$ where
- $s_1 \equiv S$, the start symbol
- $s_T \in \Sigma^*$: i.e., the final string is only terminals
- $s_i, \forall i > 1$, is derived from s_{i-1} by replacing some non-terminal X in s_{i-1} and replacing it by some β, where $x \to \beta \in R$.
- Example: parse tree

Example Derivation

Productions

S → NP VP	NP → DET NN	VP → VZ
VP → ADVP VZ	NP → ADJP NN	NP → PRO
DET → "*the*"	DET → "*a*"	DET → "*an*"
NN → "*dot*"	NN → "*cat*"	NN → "*mouse*"
VZ → "*barked*"	VZ → "*ran*"	VZ → "*sat*"
⋮	⋮	⋮

$s_1 =$

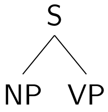
S

Example Derivation

Productions

S $\rightarrow$ NP VP	NP $\rightarrow$ DET NN	VP $\rightarrow$ VZ
VP $\rightarrow$ ADVP VZ	NP $\rightarrow$ ADJP NN	NP $\rightarrow$ PRO
DET $\rightarrow$ "*the*"	DET $\rightarrow$ "*a*"	DET $\rightarrow$ "*an*"
NN $\rightarrow$ "*dot*"	NN $\rightarrow$ "*cat*"	NN $\rightarrow$ "*mouse*"
VZ $\rightarrow$ "*barked*"	VZ $\rightarrow$ "*ran*"	VZ $\rightarrow$ "*sat*"
$\vdots$	$\vdots$	$\vdots$

$s_2 =$

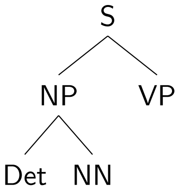


Example Derivation

Productions

S $\rightarrow$ NP VP	NP $\rightarrow$ DET NN	VP $\rightarrow$ VZ
VP $\rightarrow$ ADVP VZ	NP $\rightarrow$ ADJP NN	NP $\rightarrow$ PRO
DET $\rightarrow$ "*the*"	DET $\rightarrow$ "*a*"	DET $\rightarrow$ "*an*"
NN $\rightarrow$ "*dot*"	NN $\rightarrow$ "*cat*"	NN $\rightarrow$ "*mouse*"
VZ $\rightarrow$ "*barked*"	VZ $\rightarrow$ "*ran*"	VZ $\rightarrow$ "*sat*"
$\vdots$	$\vdots$	$\vdots$

$s_3 =$

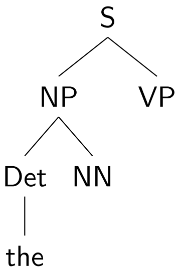


Example Derivation

Productions

S → NP VP	NP → DET NN	VP → VZ
VP → ADVP VZ	NP → ADJP NN	NP → PRO
DET → "*the*"	DET → "*a*"	DET → "*an*"
NN → "*dot*"	NN → "*cat*"	NN → "*mouse*"
VZ → "*barked*"	VZ → "*ran*"	VZ → "*sat*"
⋮	⋮	⋮

$s_4 =$

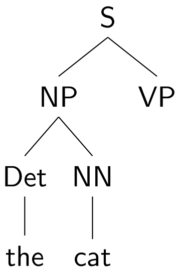


Example Derivation

Productions

S → NP VP	NP → DET NN	VP → VZ
VP → ADVP VZ	NP → ADJP NN	NP → PRO
DET → "*the*"	DET → "*a*"	DET → "*an*"
NN → "*dot*"	NN → "*cat*"	NN → "*mouse*"
VZ → "*barked*"	VZ → "*ran*"	VZ → "*sat*"
⋮	⋮	⋮

$s_5 =$

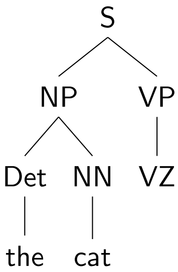


Example Derivation

Productions

S → NP VP	NP → DET NN	VP → VZ
VP → ADVP VZ	NP → ADJP NN	NP → PRO
DET → "*the*"	DET → "*a*"	DET → "*an*"
NN → "*dot*"	NN → "*cat*"	NN → "*mouse*"
VZ → "*barked*"	VZ → "*ran*"	VZ → "*sat*"
⋮	⋮	⋮

$s_6 =$

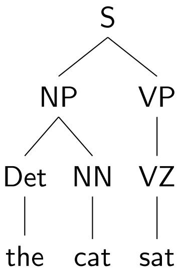


Example Derivation

Productions

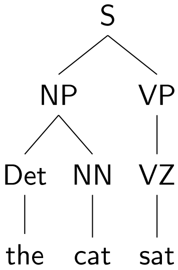
$\textsc{S} \rightarrow \textsc{NP} \quad \textsc{VP}$	$\textsc{NP} \rightarrow \textsc{Det} \quad \textsc{NN}$	$\textsc{VP} \rightarrow \textsc{VZ}$
$\textsc{VP} \rightarrow \textsc{AdvP} \quad \textsc{VZ}$	$\textsc{NP} \rightarrow \textsc{AdjP} \quad \textsc{NN}$	$\textsc{NP} \rightarrow \textsc{PRO}$
$\textsc{Det} \rightarrow$ "*the*"	$\textsc{Det} \rightarrow$ "*a*"	$\textsc{Det} \rightarrow$ "*an*"
$\textsc{NN} \rightarrow$ "*dot*"	$\textsc{NN} \rightarrow$ "*cat*"	$\textsc{NN} \rightarrow$ "*mouse*"
$\textsc{VZ} \rightarrow$ "*barked*"	$\textsc{VZ} \rightarrow$ "*ran*"	$\textsc{VZ} \rightarrow$ "*sat*"
$\vdots$	$\vdots$	$\vdots$

$s_7 =$



Example Derivation

$s_7 =$



Ambiguous Yields

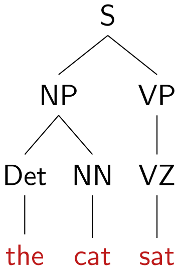
The **yield** of a parse tree is the collection of terminals produced by the parse tree. Given a yield s.

Parsing / Decoding

Given, a yield s and a grammar G, determine the set of parse trees that could have produced that sequence of terminals: $T_G(s)$.

Example Derivation

$s_7 =$



Ambiguous Yields

The **yield** of a parse tree is the collection of terminals produced by the parse tree. Given a yield s.

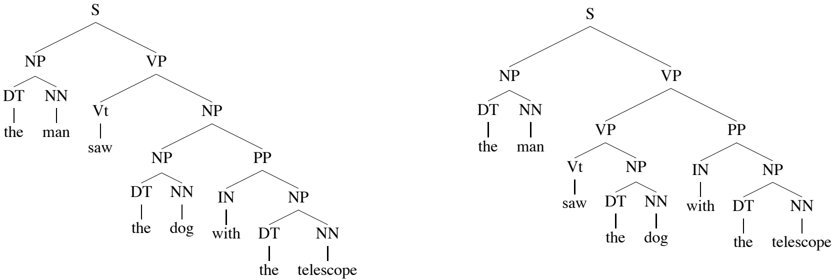
Parsing / Decoding

Given, a yield s and a grammar G, determine the set of parse trees that could have produced that sequence of terminals: $T_G(s)$.

Ambiguity

Example sentence: "The man saw the dog with the telescope"

- Grammatical: $T_G(s) > 0$
- Ambiguous: $T_G(s) > 1$

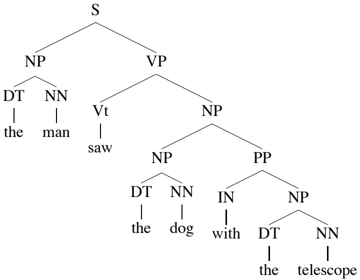


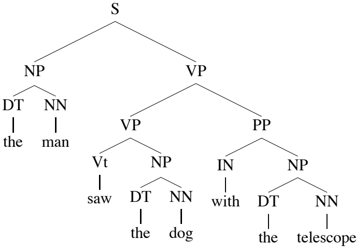
- Which should we prefer?

Ambiguity

Example sentence: "The man saw the dog with the telescope"

- Grammatical: $T_G(s) > 0$
- Ambiguous: $T_G(s) > 1$





- Which should we prefer?
- One is more *probable* than the other
- We can formalize this by adding a notion of probability to our context-free grammar

Outline

1. Motivation
2. Context Free Grammars
3. Probabilistic Context Free Grammars
   - Parameterization: Defining Score Function
   - Estimation
   - Parsing

Goals

- What we want is a probability distribution over possible parse trees $t \in T_G(s)$

$$\forall t, p(t) \geq 0 \qquad \sum_{t \in T_G(s)} p(t) = 1 \tag{1}$$

- Rest of this lecture:
  - How do we define the function $p(t)$ (paramterization)
  - How do we learn $p(t)$ from data (estimation)
  - Given a sentence, how do we find the possible parse trees (parsing / decoding)

Parametrization

- For every production $\alpha \to \beta$, we assume we have a function $q(\alpha \to \beta)$
- We consider it a **conditional probability** of β (LHS) being derived from α (RHS)

$$\sum_{\alpha \to \beta \in R : \alpha = X} q(\alpha \to \beta) = 1 \tag{2}$$

- The total probability of a tree $t \equiv \{\alpha_1 \to \beta_1 \ldots \alpha_n \to \beta_n\}$ is

$$p(t) = \prod_{i=1}^{n} q(\alpha_i \to \beta_i) \tag{3}$$

Estimation

- Get a bunch of grad students to make parse trees for a million sentences
- Mitch Markus: Penn Treebank (Wall Street Journal)
- To compute the conditional probability of a rule,

$$q(\textsc{np} \rightarrow \textsc{Det} \ \textsc{adj} \ \textsc{nn}) \approx \frac{\text{Count}(\textsc{np} \rightarrow \textsc{Det} \ \textsc{adj} \ \textsc{nn})}{\text{Count}(\textsc{np})}$$

- Where "Count" is the number of times that derivation appears in the sentences

Estimation

- Get a bunch of grad students to make parse trees for a million sentences
- Mitch Markus: Penn Treebank (Wall Street Journal)
- To compute the conditional probability of a rule,

$$q(\text{NP} \rightarrow \text{Det ADJ NN}) \approx \frac{\text{Count}(\text{NP} \rightarrow \text{Det ADJ NN})}{\text{Count}(\text{NP})}$$

- Where "Count" is the number of times that derivation appears in the sentences
- Why no smoothing?

Dynamic Programming

- Like for dependency parsing, we build a chart to consider all possible subtrees
- First, however, we'll just consider whether a sentence is grammatical or not
- Build up a chart with all possible derivations of spans
- Then see entry with start symbol over the entire sentence: those are all grammatical parses

CYK Algorithm (deterministic)

Assumptions

Assumes binary grammar (not too difficult to extend) and no recursive rules

Given sentence $\vec{w}$ of length N, grammar (N, Σ, R, S)
Initialize array $C[s, t, n]$ as array of booleans, all false ($\perp$)

CYK Algorithm (deterministic)

Assumptions

Assumes binary grammar (not too difficult to extend) and no recursive rules

Given sentence $\vec{w}$ of length N, grammar (N, Σ, R, S)
Initialize array $C[s, t, n]$ as array of booleans, all false ($\perp$)
for $i = 0 \dots N$ **do**
  for For each production $r_j \equiv N_a \rightarrow w_i$ **do**
    set $C[i, i, a] \leftarrow \top$

CYK Algorithm (deterministic)

Assumptions

Assumes binary grammar (not too difficult to extend) and no recursive rules

Given sentence $\vec{w}$ of length N, grammar (N, Σ, R, S)
Initialize array $C[s, t, n]$ as array of booleans, all false ($\perp$)
for $i = 0 \ldots N$ **do**
  for For each production $r_j \equiv N_a \rightarrow w_i$ **do**
    set $C[i, i, a] \leftarrow \top$
for $l = 2 \ldots n$ (length of span) **do**
  for $s = 1 \ldots N - l + 1$ (start of span) **do**
    for $k = 1 \ldots l - 1$ (pivot within span) **do**
      for each production $r \equiv \alpha \rightarrow \beta\gamma$ **do**
        if $\neg C[s, s + l, \alpha]$ **then**
          $C[s, s + l, \alpha] \leftarrow C[s, s + k - 1, \beta] \wedge C[s + k, s + l, \gamma]$

Chart Parsing

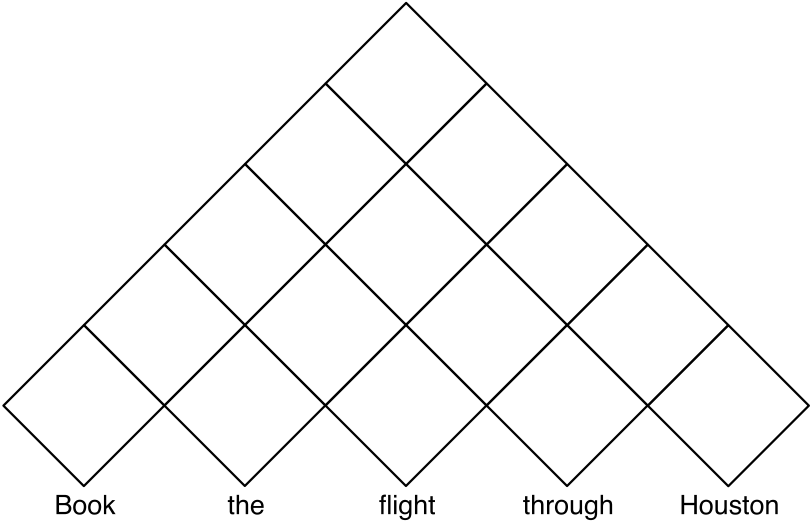


Chart Parsing

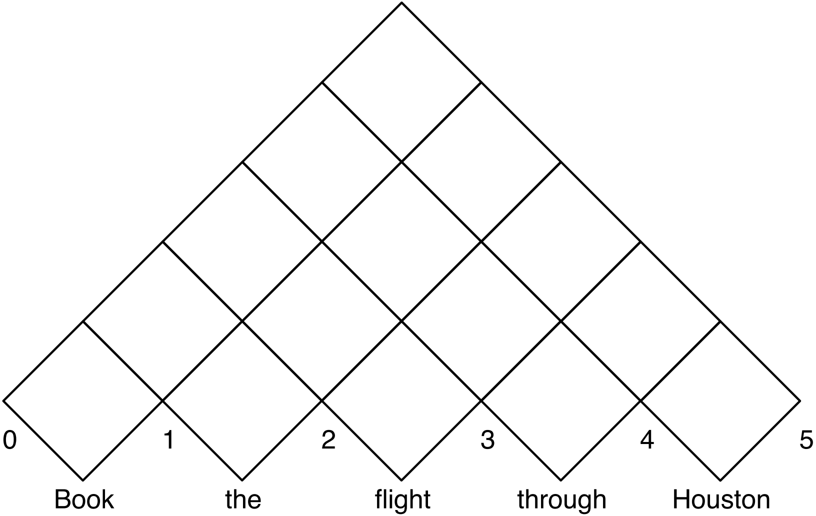


Chart Parsing

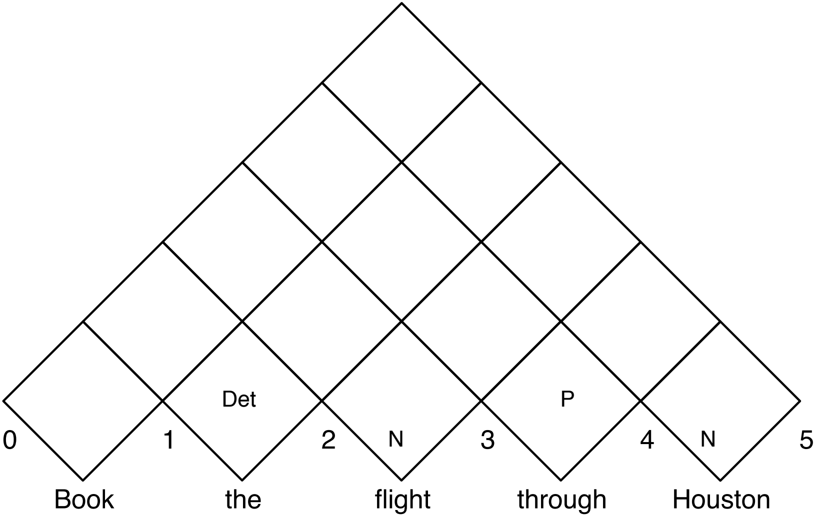


Chart Parsing

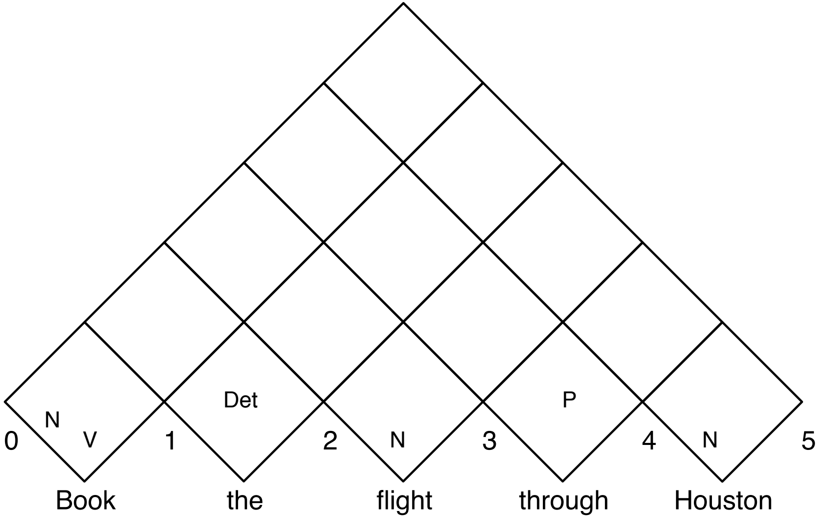


Chart Parsing

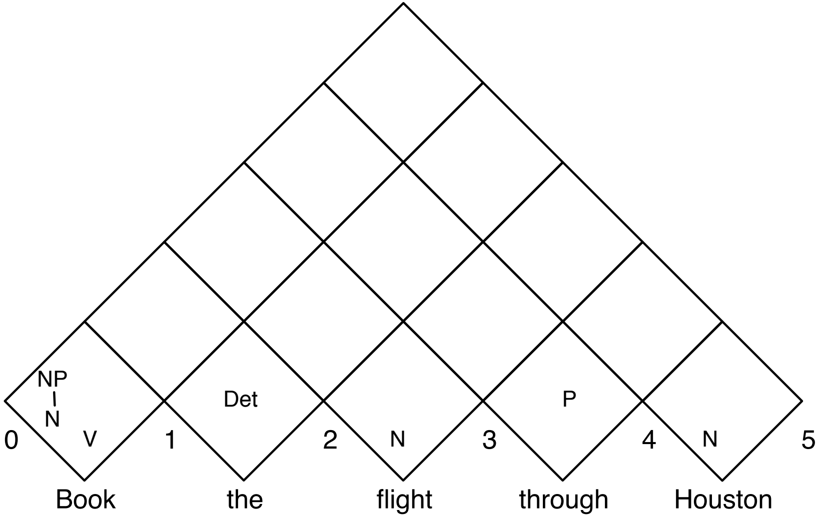


Chart Parsing

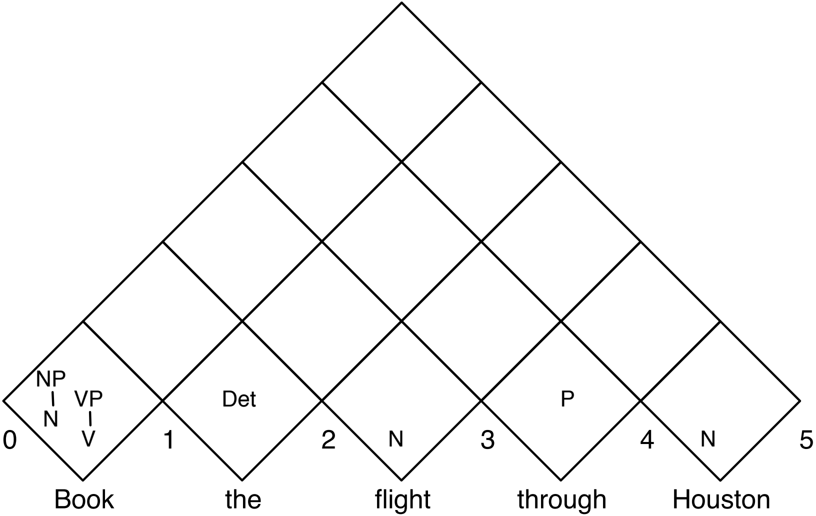


Chart Parsing

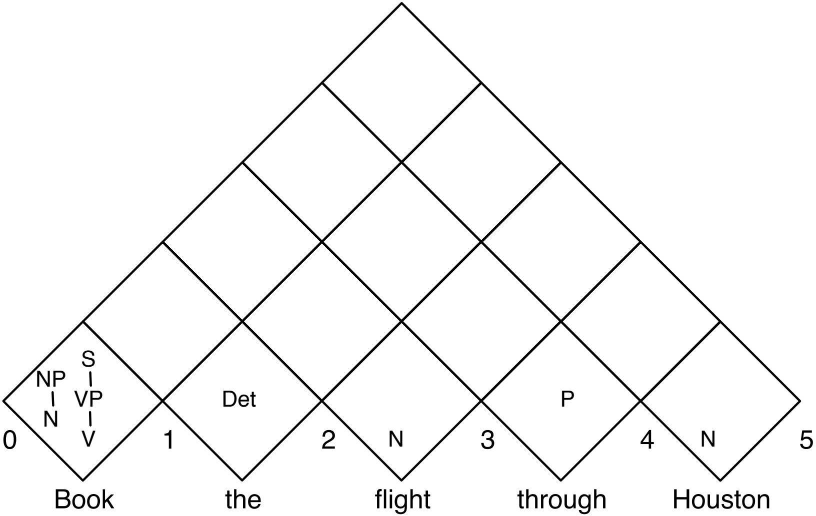


Chart Parsing

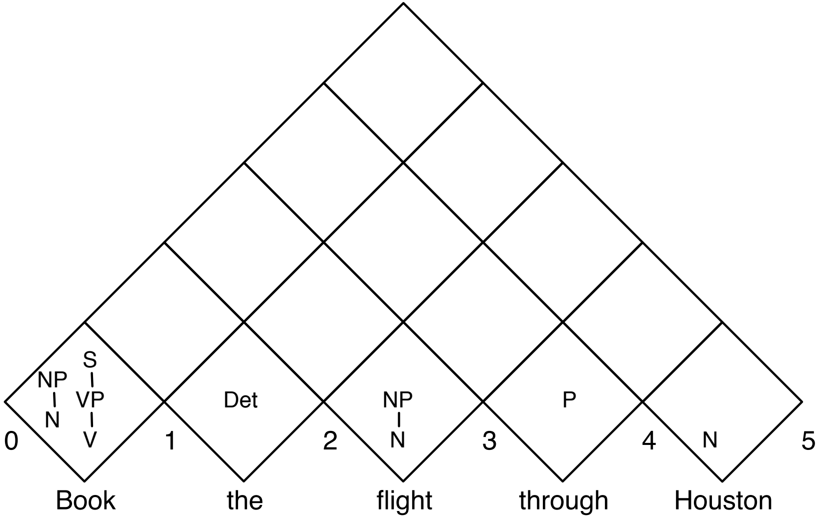


Chart Parsing

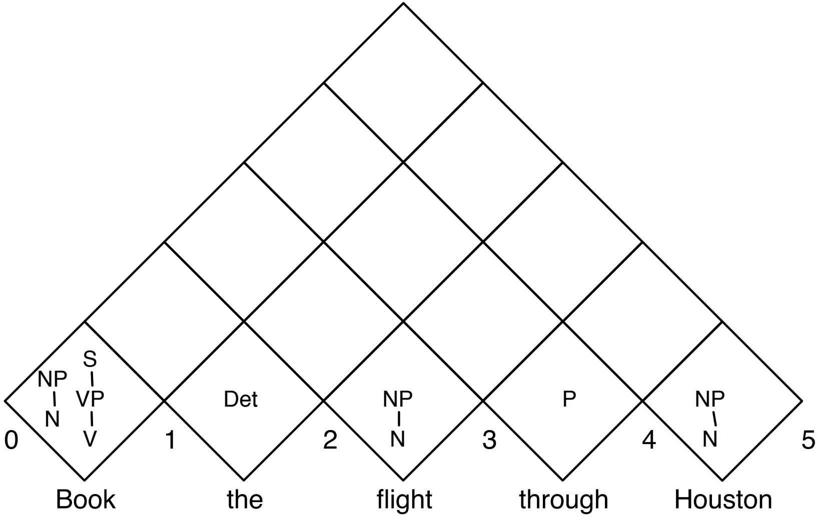


Chart Parsing

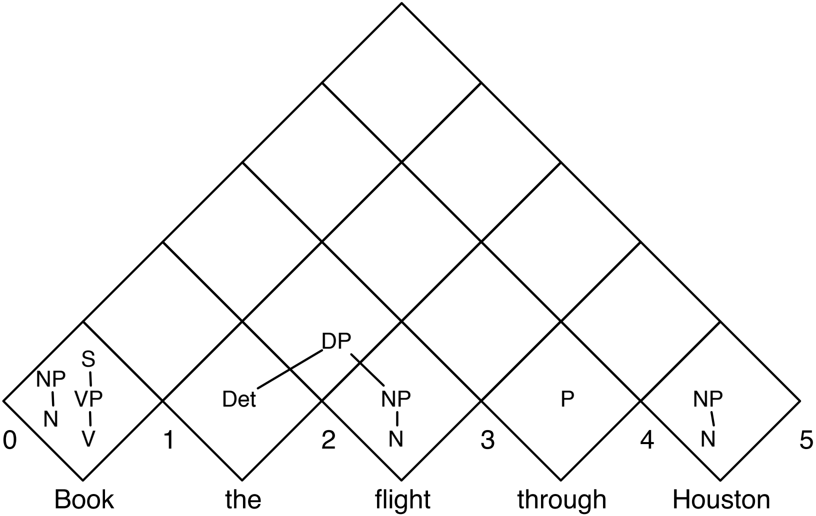


Chart Parsing

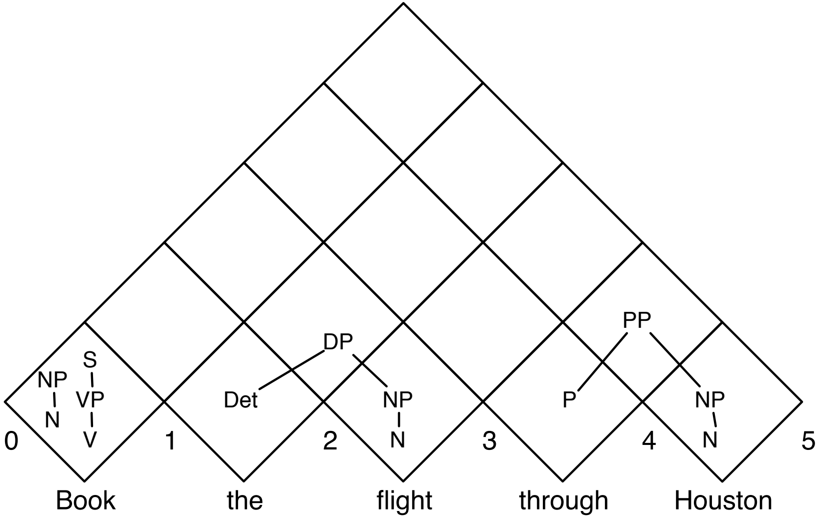


Chart Parsing

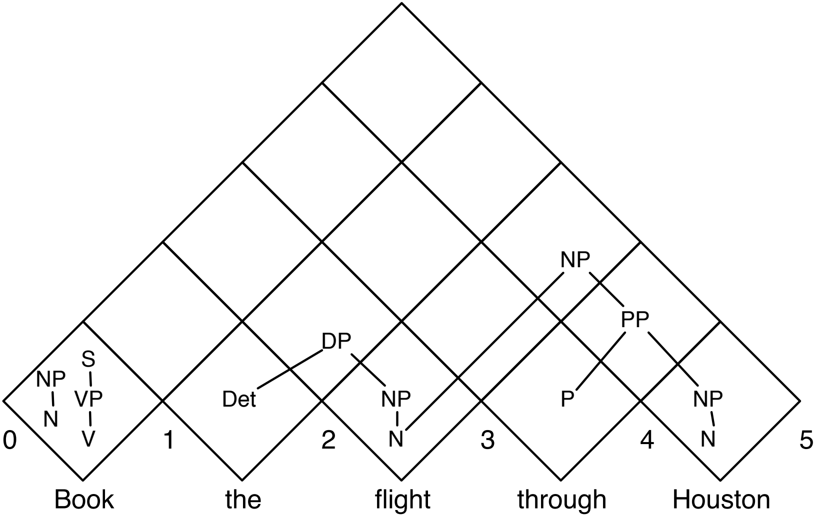


Chart Parsing

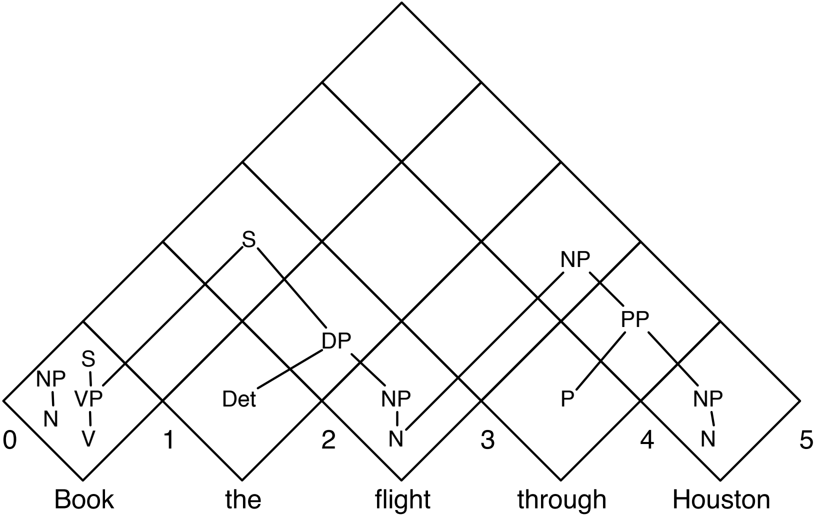


Chart Parsing

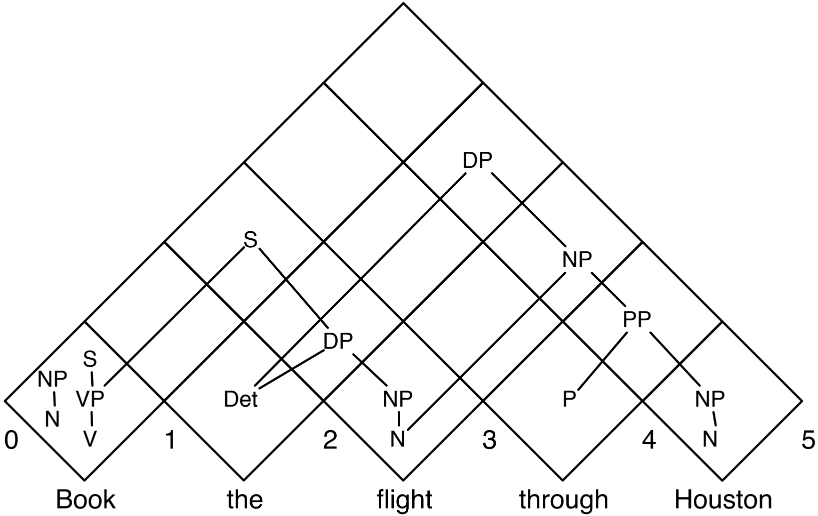
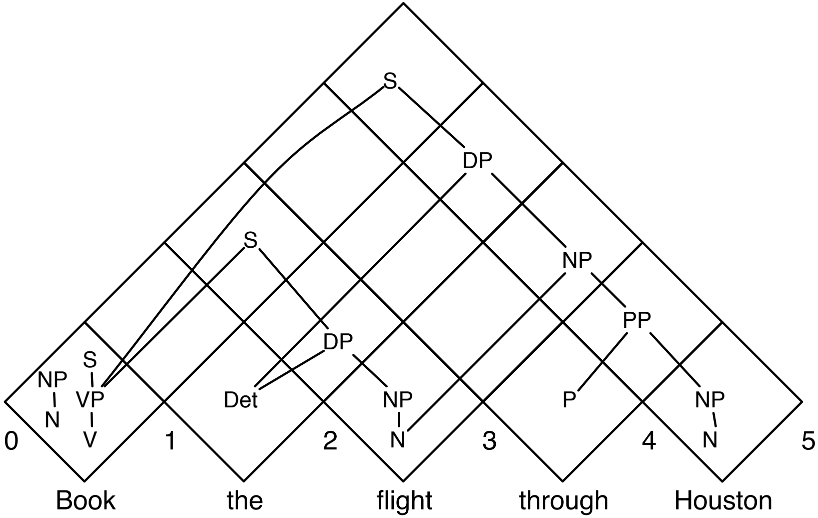


Chart Parsing



How to detal with PCFG ambiguity

- In addition to keeping track of non-terminals in cell, also include **max** probability of forming non-terminal from sub-trees

  $$C[s, s+k, \alpha] \leftarrow \max(C[s, s+k, \alpha], C[s, s+l-1, \beta] \cdot C[s+l, s+k, \gamma])$$

- The score associated with S in the top of the chart is the best overall parse-tree (**given the yield**)

Recap

- Hierarchical syntax model: context free grammar
- Probabilistic interpretation: learn from data to solve ambiguity
- In class:
  - Work through example to resolve ambiguity
  - Morphology HW results