CMSC 723: Computational Linguistics I — Session #7

Syntactic Parsing with CFGs

Jimmy Lin
The iSchool
University of Maryland

Wednesday, October 14, 2009

Today's Agenda

- Words… **structure…** meaning…
- Last week: formal grammars
  - Context-free grammars
  - Grammars for English
  - Treebanks
  - Dependency grammars
- Today: parsing with CFGs
  - Top-down and bottom-up parsing
  - CKY parsing
  - Earley parsing

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…
  - Unfortunately, parsing algorithms don't help in selecting the correct tree from among all the possible trees

Parsing Algorithms

- Parsing is (surprise) a search problem
- Two basic (= bad) algorithms:
  - Top-down search
  - Bottom-up search
- Two “real” algorithms:
  - CKY parsing
  - Earley parsing
- Simplifying assumptions:
  - Morphological analysis is done
  - All the words are known

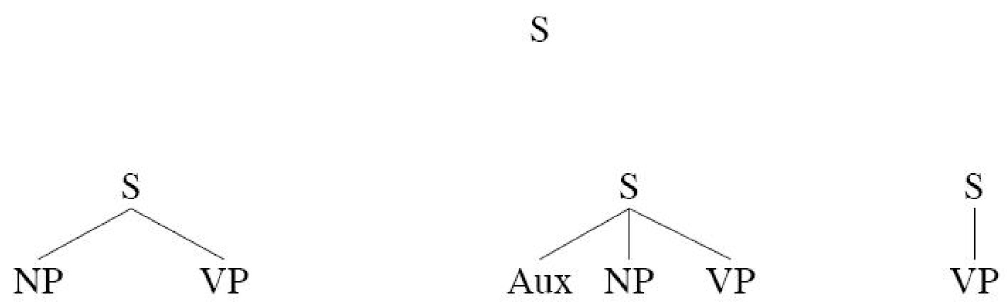
Top-Down Search

- 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

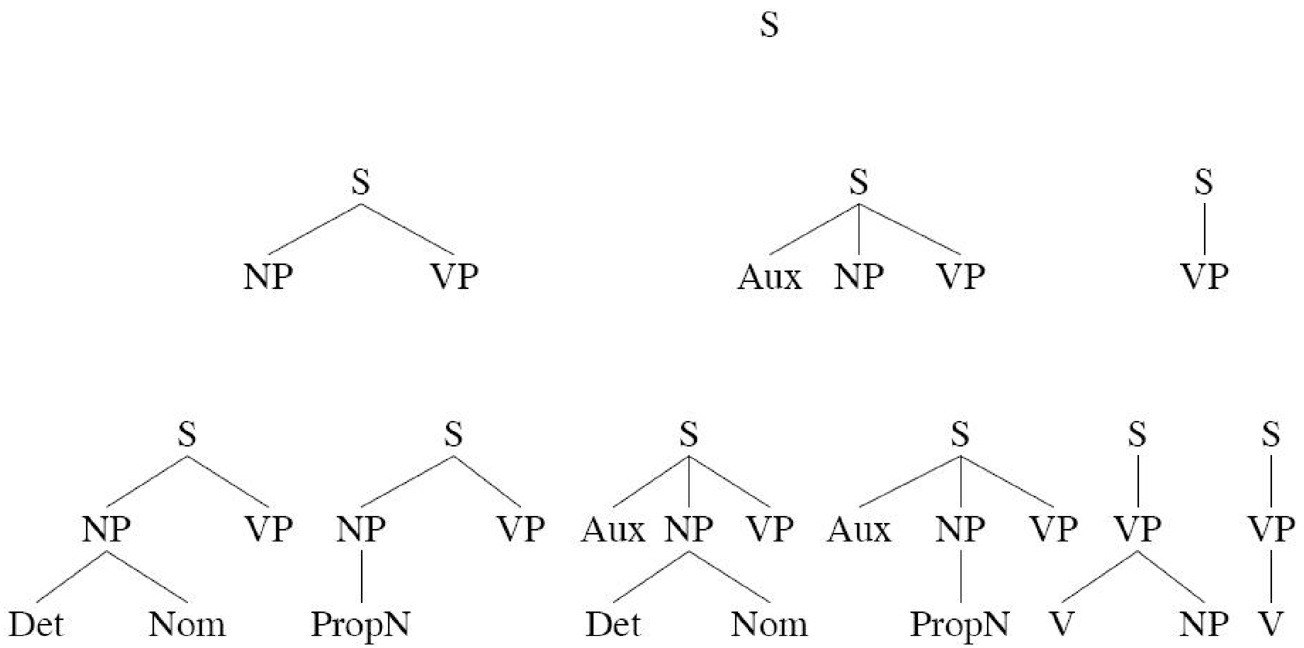
Top-Down Search

S

Top-Down Search



Top-Down Search



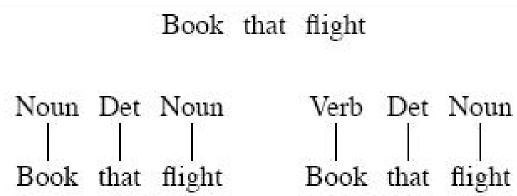
Bottom-Up Search

- 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

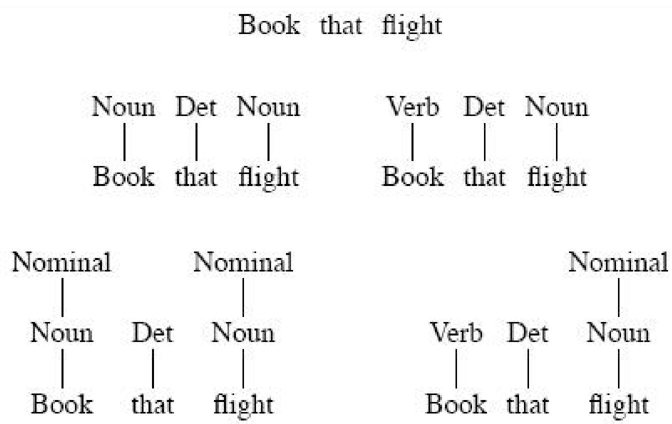
Bottom-Up Search

Book that flight

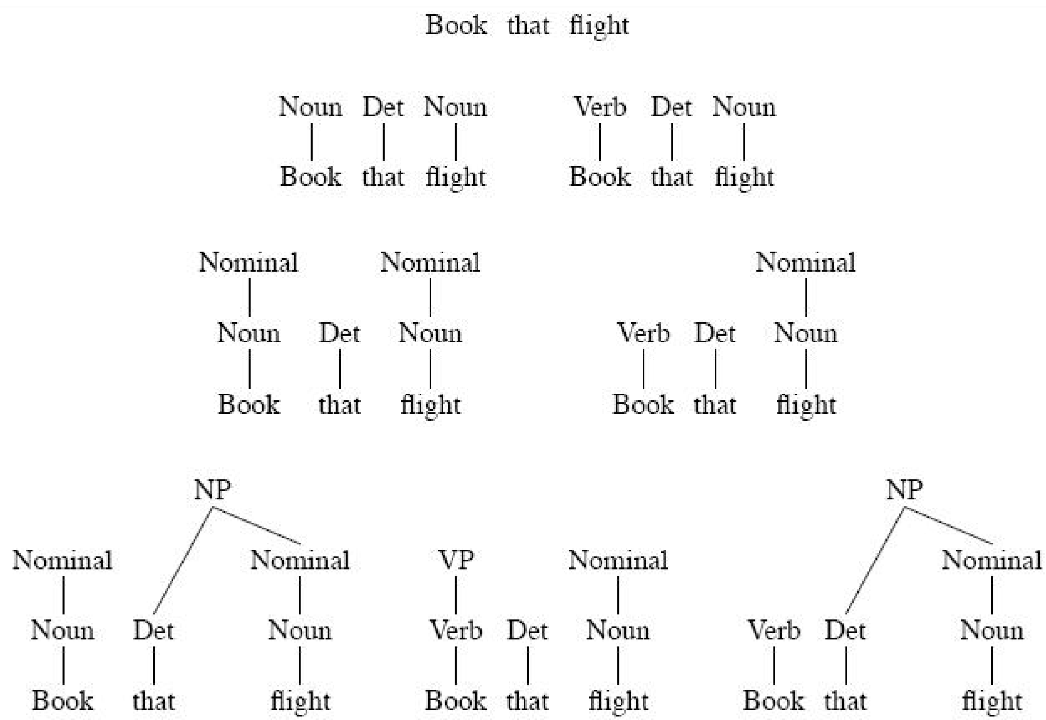
Bottom-Up Search



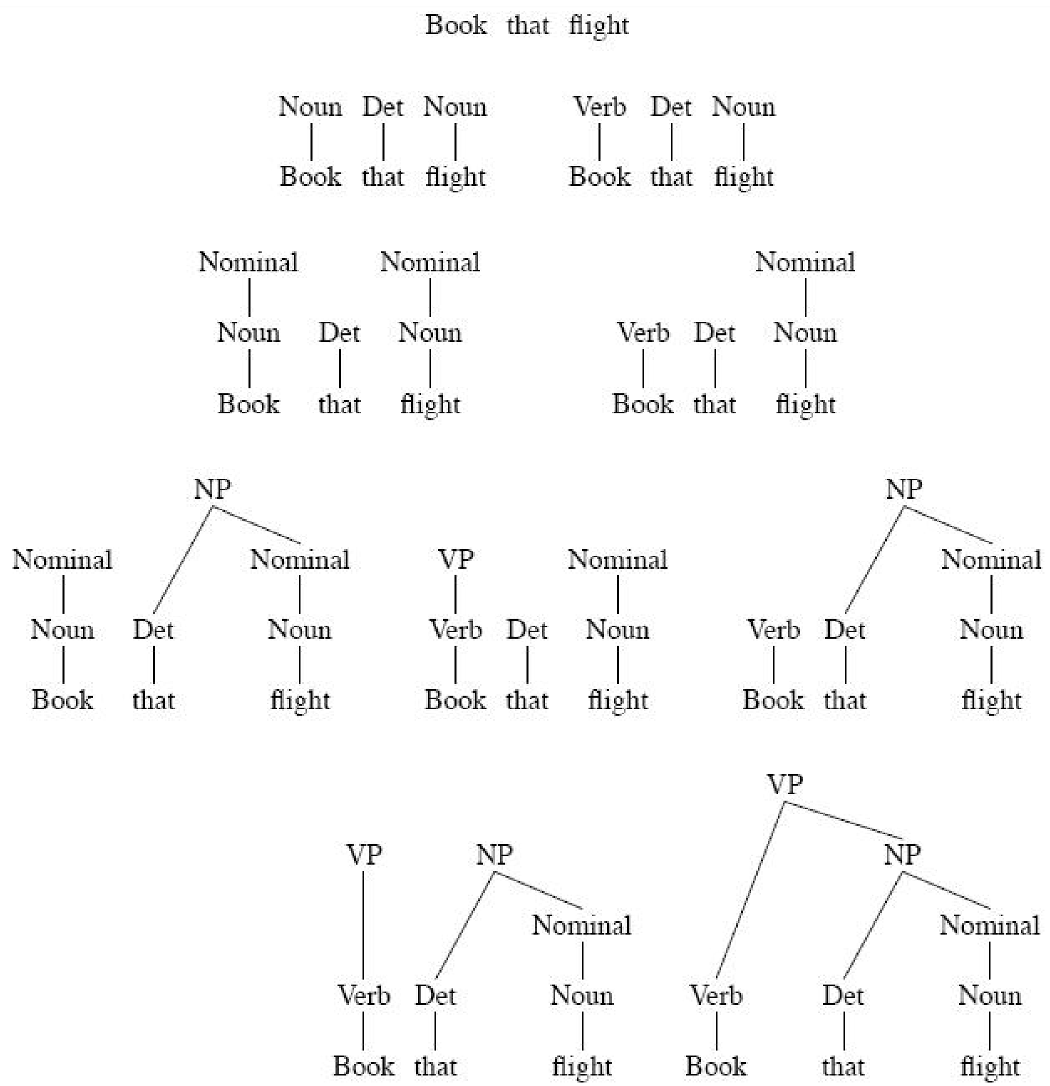
Bottom-Up Search



Bottom-Up Search



Bottom-Up Search



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

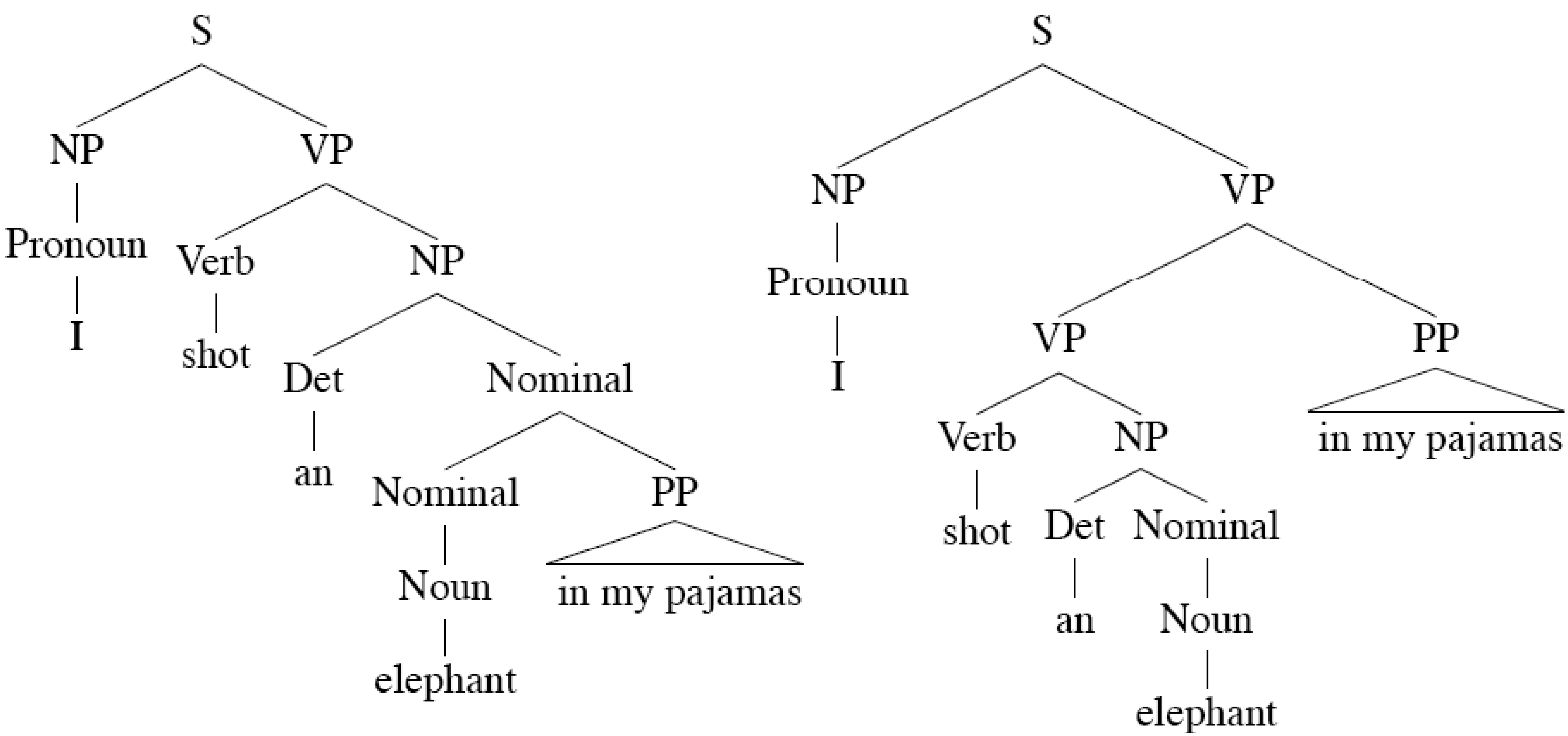
Parsing as Search

- 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, then back up and make a different choice
  - Remember DFS/BFS for NDFSA recognition?

Backtracking isn’t enough!

- Ambiguity
- Shared sub-problems

Ambiguity



Or consider: I saw the man on the hill with the telescope.

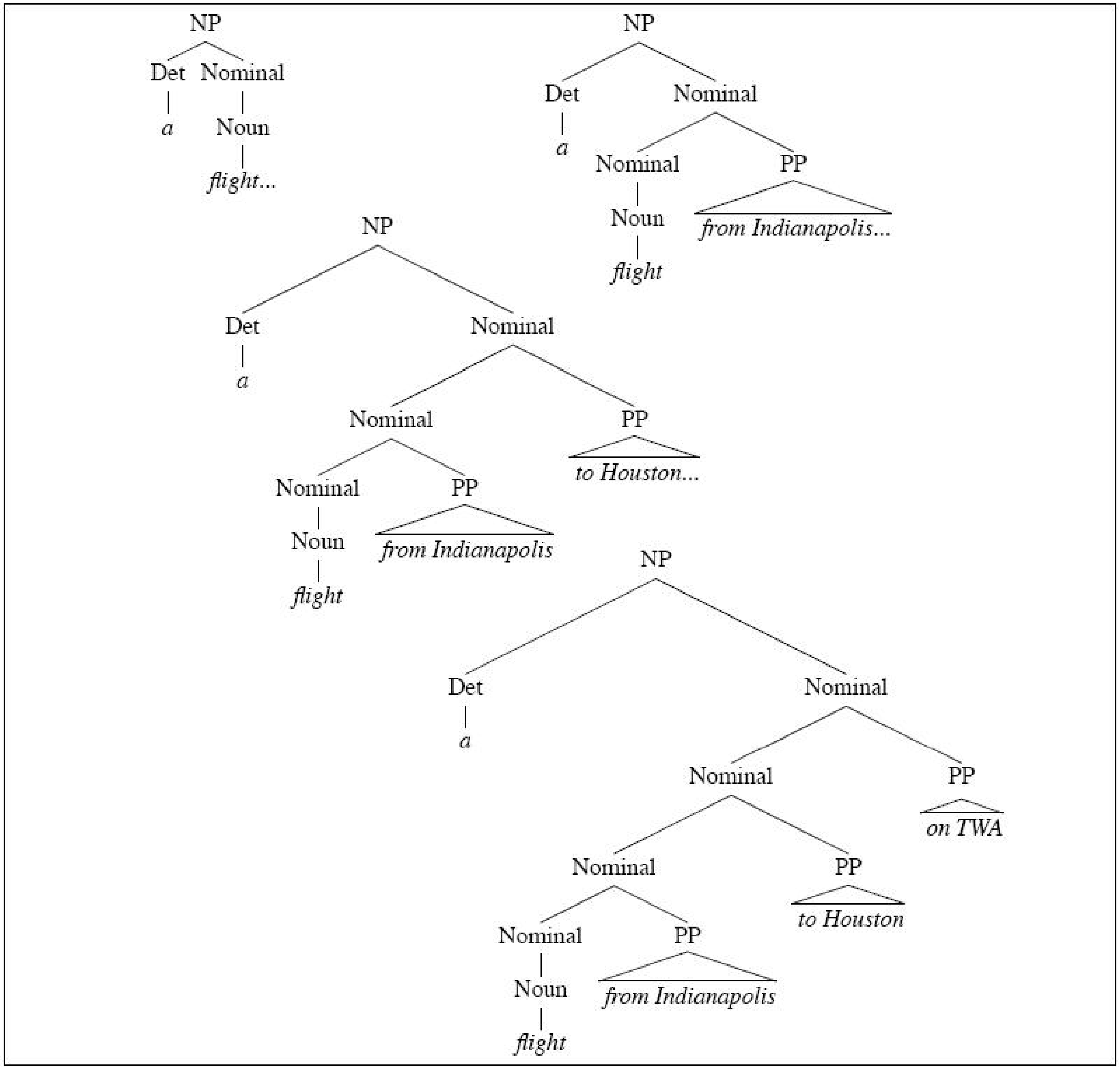
Shared Sub-Problems

- 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

Shared Sub-Problems: Example

- Example: “A flight from Indianapolis to Houston on TWA”
- Assume a top-down parse making choices among the various nominal rules:
  - Nominal → Noun
  - Nominal → Nominal PP
- Statically choosing the rules in this order leads to lots of extra work...

Shared Sub-Problems: Example



Efficient Parsing

- Dynamic programming to the rescue!
- Intuition: store partial results in tables, thereby:
  - Avoiding repeated work on shared sub-problems
  - Efficiently storing ambiguous structures with shared sub-parts
- Two algorithms:
  - CKY: roughly, bottom-up
  - Earley: roughly, top-down

CKY Parsing: CNF

- CKY parsing requires that the grammar consist of ε-free, binary rules = Chomsky Normal Form
  - All rules of the form:

    $A \rightarrow B\ C$

    $D \rightarrow w$
  - What does the tree look like?
- What if my CFG isn't in CNF?

CKY Parsing with Arbitrary CFGs

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

$$A \rightarrow B\ C\ D \quad \Rightarrow \quad \begin{array}{l} A \rightarrow X\ D \\ X \rightarrow B\ C \end{array}$$

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

- What does this mean?
  - = weak equivalence
  - The rewritten grammar accepts (and rejects) the same set of strings as the original grammar…
  - But the resulting derivations (trees) are different

Sample L_1 Grammar

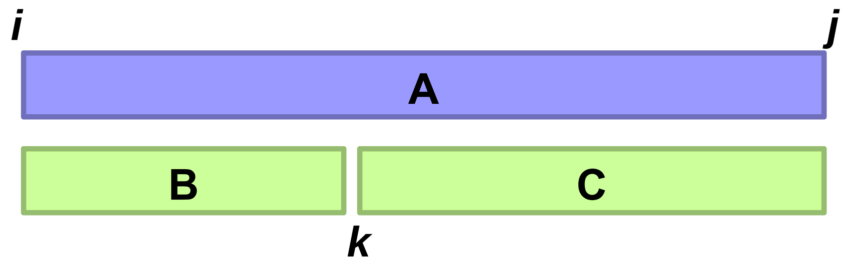
Grammar	Lexicon
$S \rightarrow NP\ VP$	$Det \rightarrow that \mid this \mid a$
$S \rightarrow Aux\ NP\ VP$	$Noun \rightarrow book \mid flight \mid meal \mid money$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid prefer$
$NP \rightarrow Pronoun$	$Pronoun \rightarrow I \mid she \mid me$
$NP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

L_1 Grammar: CNF Conversion

$\mathscr{L}_1$ **Grammar**	$\mathscr{L}_1$ **in CNF**
$S \rightarrow NP\ VP$	$S \rightarrow NP\ VP$
$S \rightarrow Aux\ NP\ VP$	$S \rightarrow X1\ VP$
	$X1 \rightarrow Aux\ NP$
$S \rightarrow VP$	$S \rightarrow book \mid include \mid prefer$
	$S \rightarrow Verb\ NP$
	$S \rightarrow X2\ PP$
	$S \rightarrow Verb\ PP$
	$S \rightarrow VP\ PP$
$NP \rightarrow Pronoun$	$NP \rightarrow I \mid she \mid me$
$NP \rightarrow Proper\text{-}Noun$	$NP \rightarrow TWA \mid Houston$
$NP \rightarrow Det\ Nominal$	$NP \rightarrow Det\ Nominal$
$Nominal \rightarrow Noun$	$Nominal \rightarrow book \mid flight \mid meal \mid money$
$Nominal \rightarrow Nominal\ Noun$	$Nominal \rightarrow Nominal\ Noun$
$Nominal \rightarrow Nominal\ PP$	$Nominal \rightarrow Nominal\ PP$
$VP \rightarrow Verb$	$VP \rightarrow book \mid include \mid prefer$
$VP \rightarrow Verb\ NP$	$VP \rightarrow Verb\ NP$
$VP \rightarrow Verb\ NP\ PP$	$VP \rightarrow X2\ PP$
	$X2 \rightarrow Verb\ NP$
$VP \rightarrow Verb\ PP$	$VP \rightarrow Verb\ PP$
$VP \rightarrow VP\ PP$	$VP \rightarrow VP\ PP$
$PP \rightarrow Preposition\ NP$	$PP \rightarrow Preposition\ NP$

CKY Parsing: Intuition

- Consider the rule D → w
  - Terminal (word) forms a constituent
  - Trivial to apply
- Consider the rule A → 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* = length of input string
  - We need an *N* × *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:

FROM:	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

CKY Parsing: Table-Filling

- So let's fill this table…
  - And look at the cell [*0*, *N*]: which means?
- But how?

TO:

FROM:

	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

CKY Parsing: Table-Filling

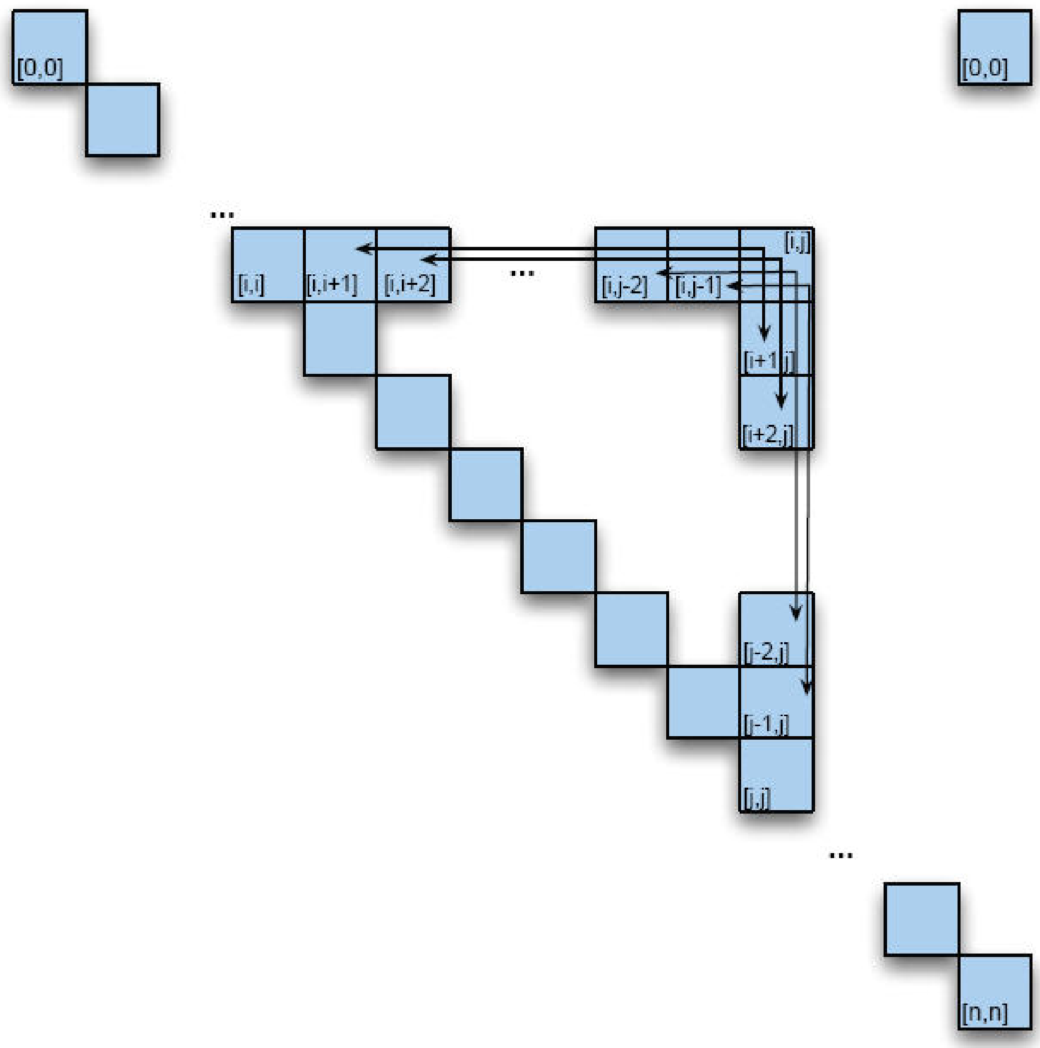
- In order for A to span [*i*, *j*]:
  - A $\rightarrow$ B C is a rule in the grammar, and
  - There must be a B in [*i*, *k*] and a C in [*k*, *j*] for some *i*<*k*<*j*
- Operationally:
  - To apply rule A $\rightarrow$ B C, look for a B in [*i*, *k*] and a C in [*k*, *j*]
  - In the table: look left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

TO:

FROM:	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

CKY Parsing: Rule Application

note: mistake in book (Figure 13.11, p 441), should be [0,n]



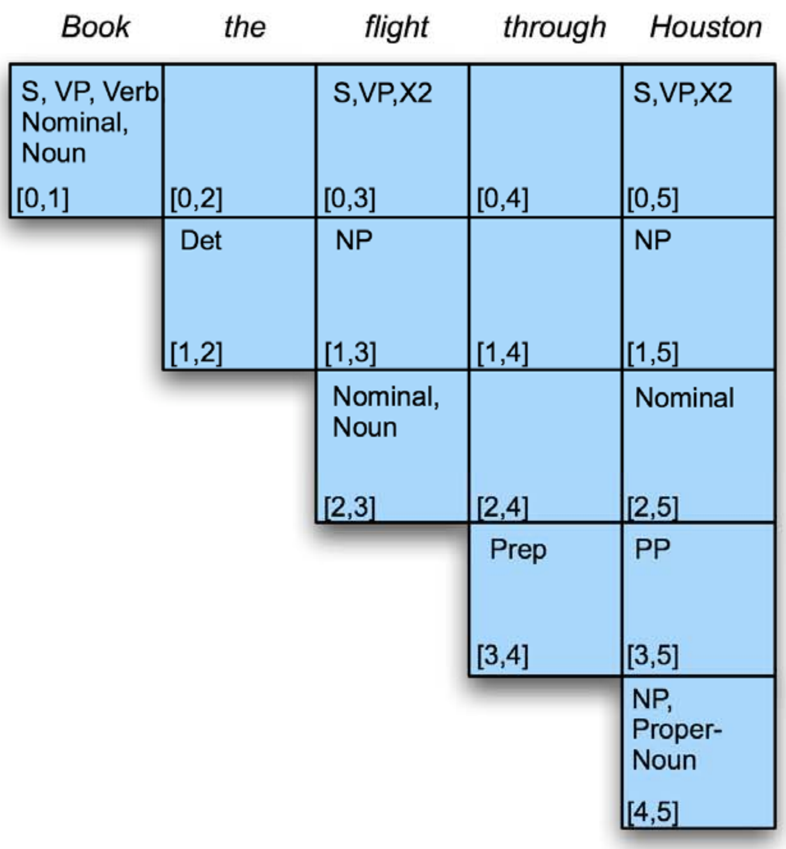
CKY Parsing: Cell Ordering

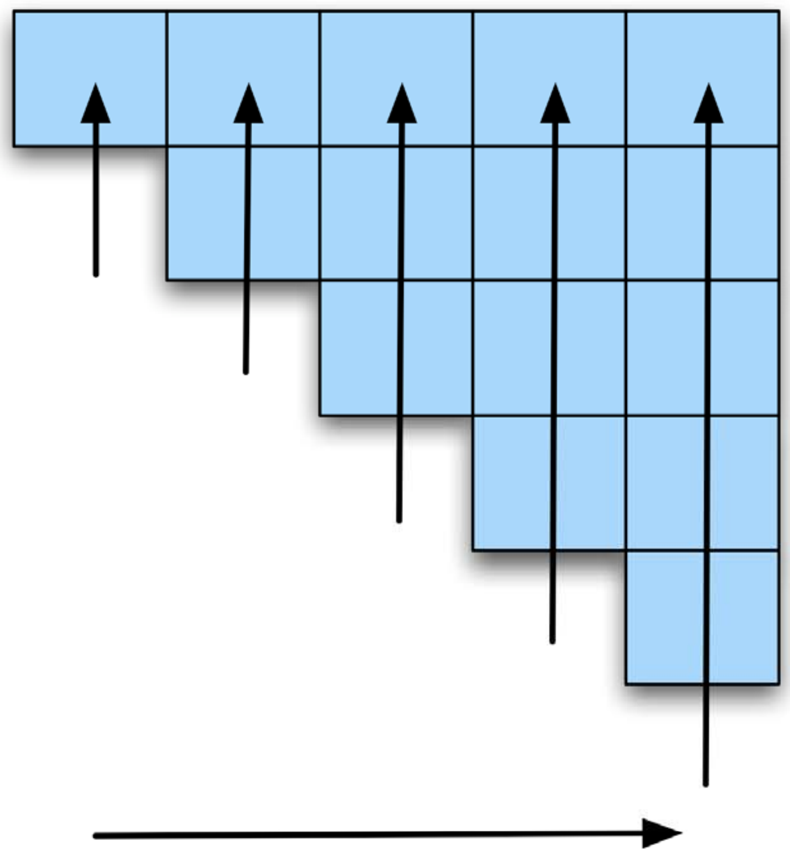
- CKY = exercise in filling the table representing spans
  - Need to establish a systematic order for considering each cell
  - For each cell [*i*, *j*] consider all possible values for *k* and try applying each rule
- What constraints do we have on the ordering of the cells?

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





CKY Algorithm

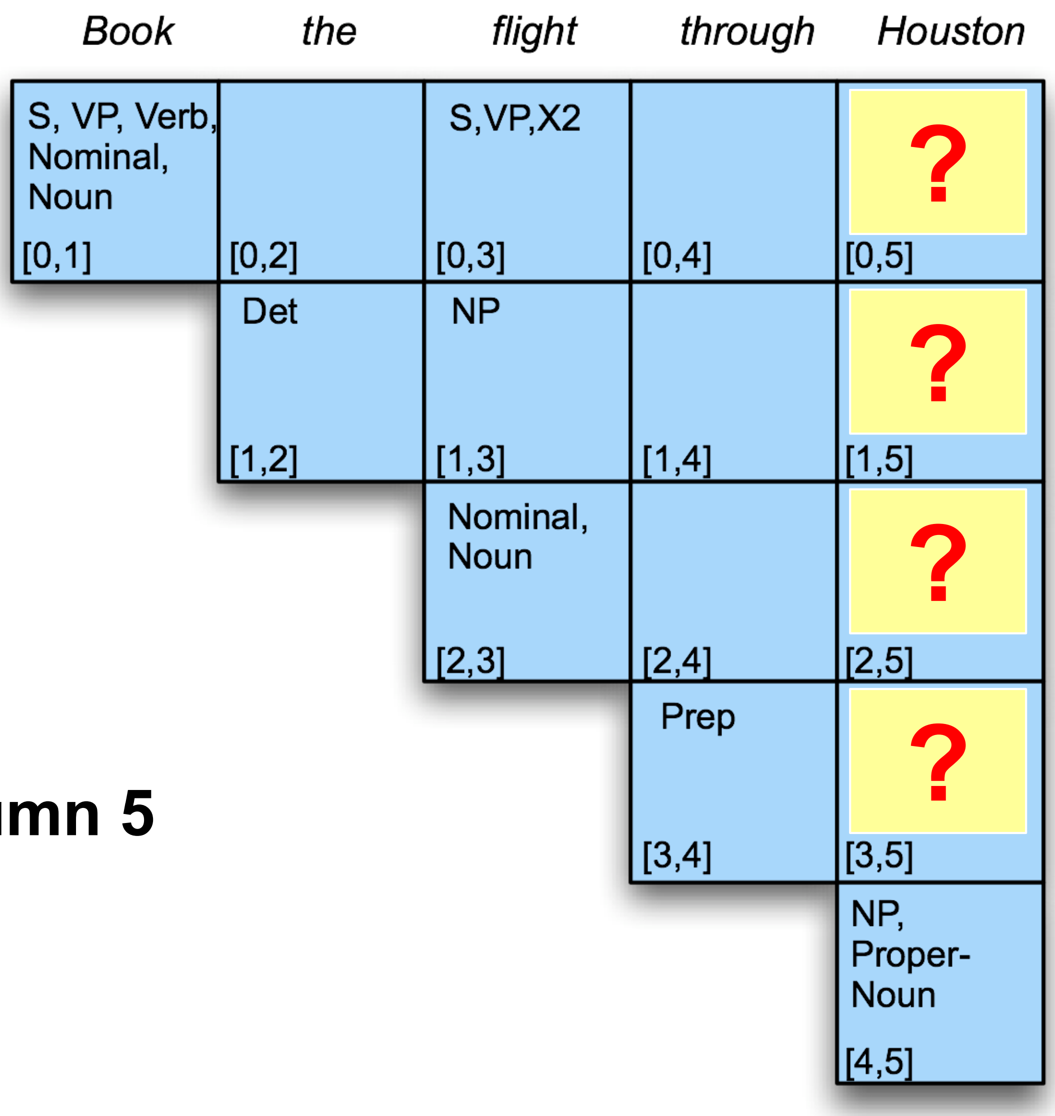
function CKY-PARSE(*words, grammar*) **returns** *table*

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 Parsing: Recognize or Parse

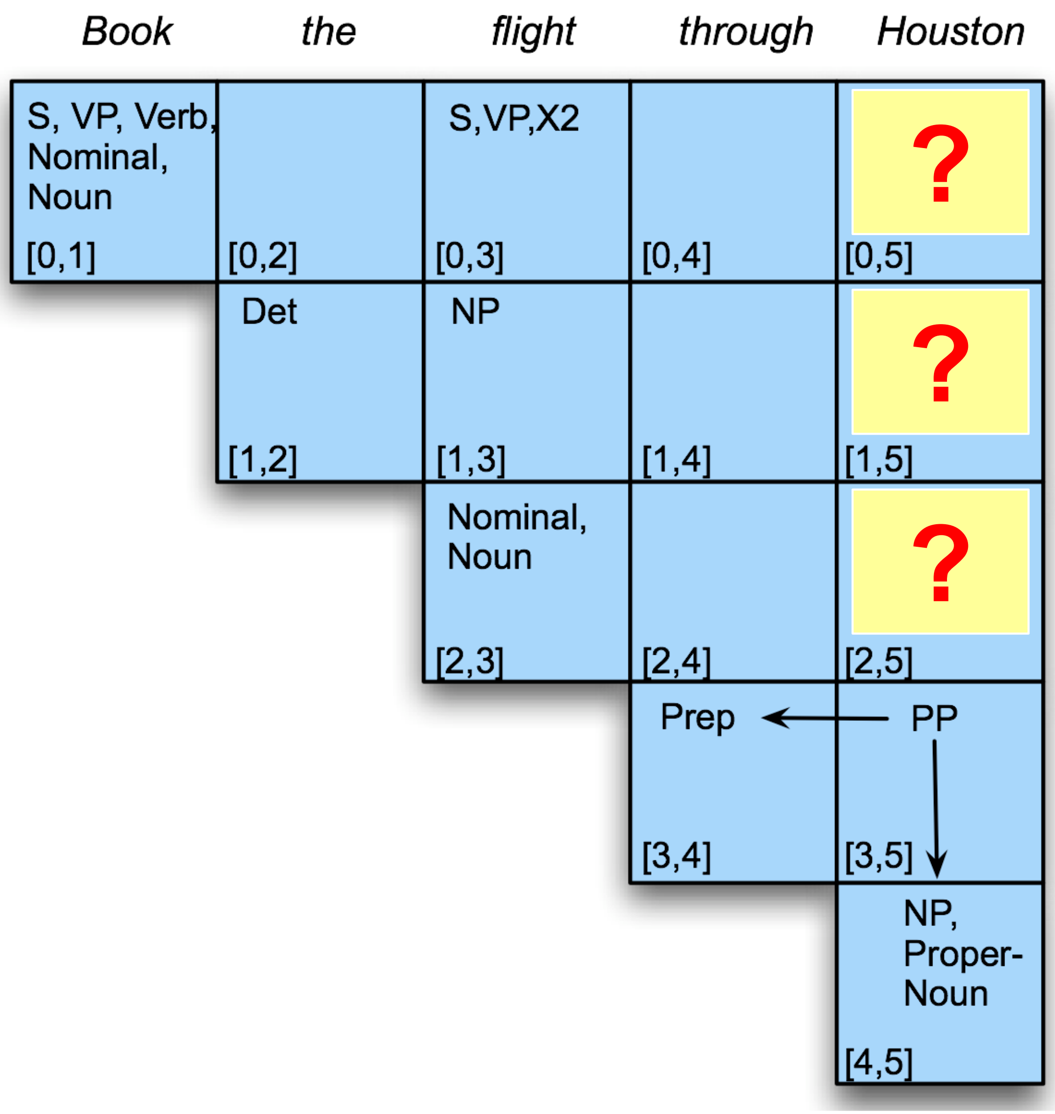
- Is this really a parser?
- Recognizer to parser: add backpointers!

CKY: Example



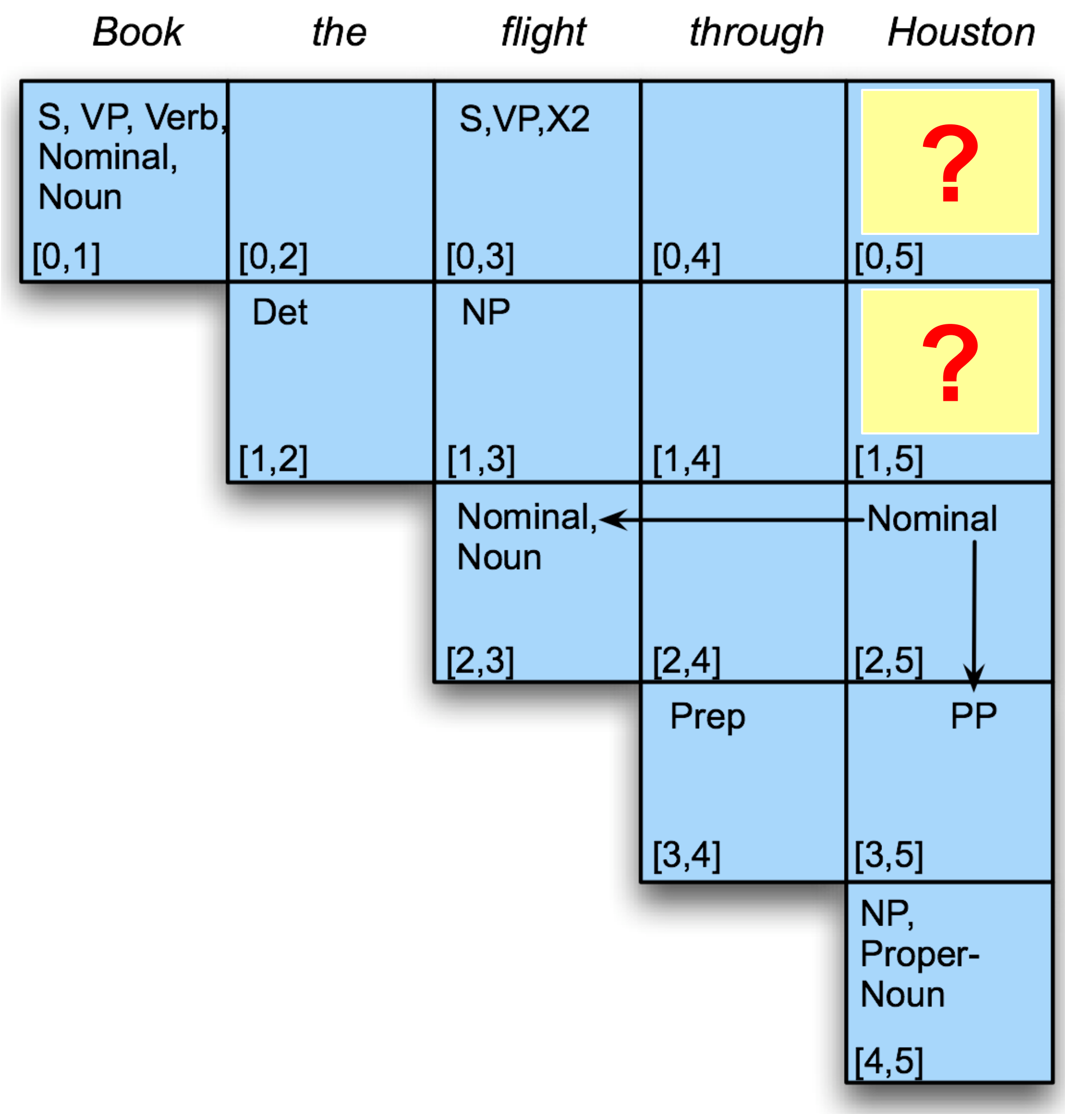
Filling column 5

CKY: Example



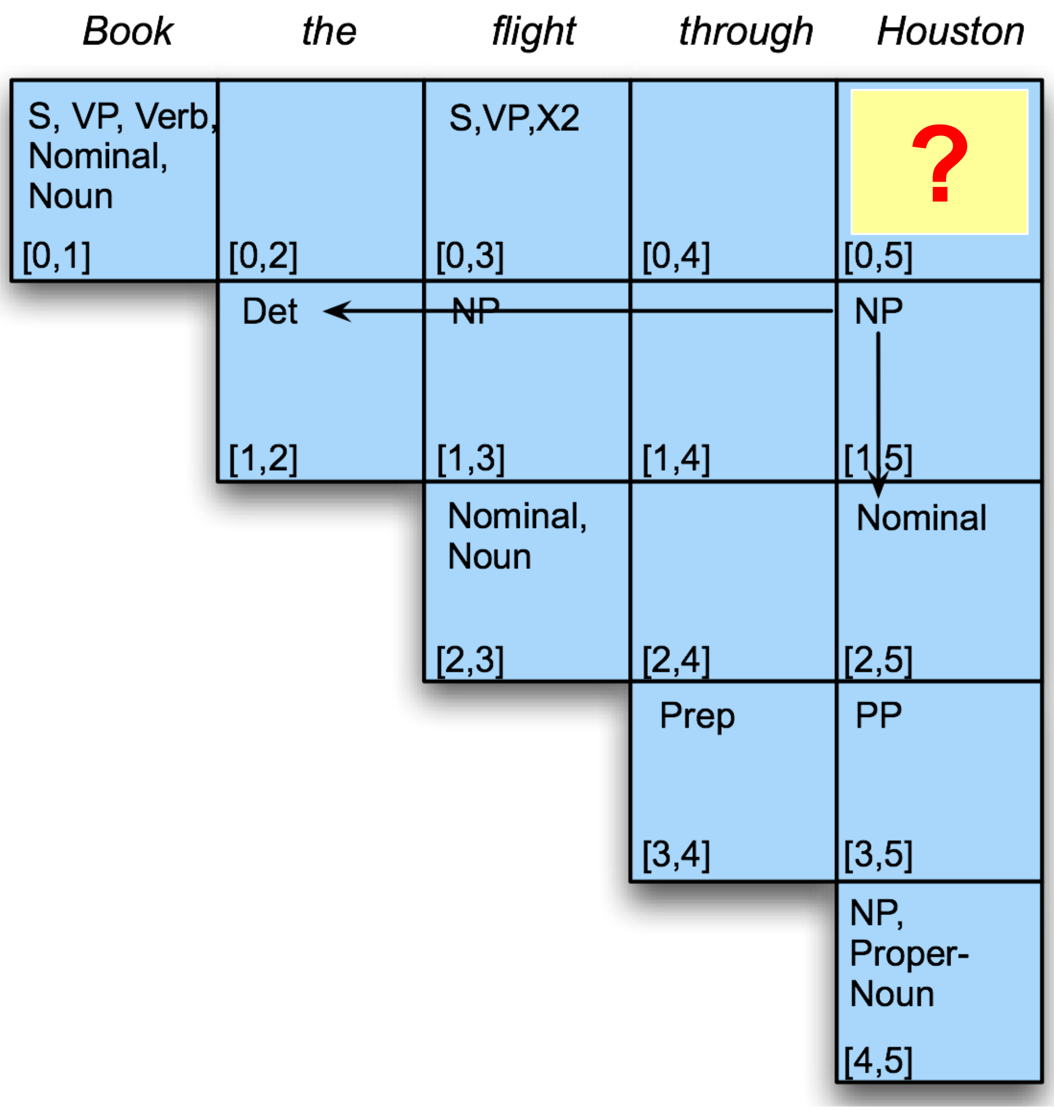
Book
the
flight
through
Houston
S, VP, Verb, Nominal, Noun
[0,1]
[0,2]
S,VP,X2
[0,3]
[0,4]
?
[0,5]
Det
[1,2]
NP
[1,3]
[1,4]
?
[1,5]
Nominal, Noun
[2,3]
[2,4]
?
[2,5]
Prep
[3,4]
PP
[3,5]
NP, Proper-Noun
[4,5]

CKY: Example

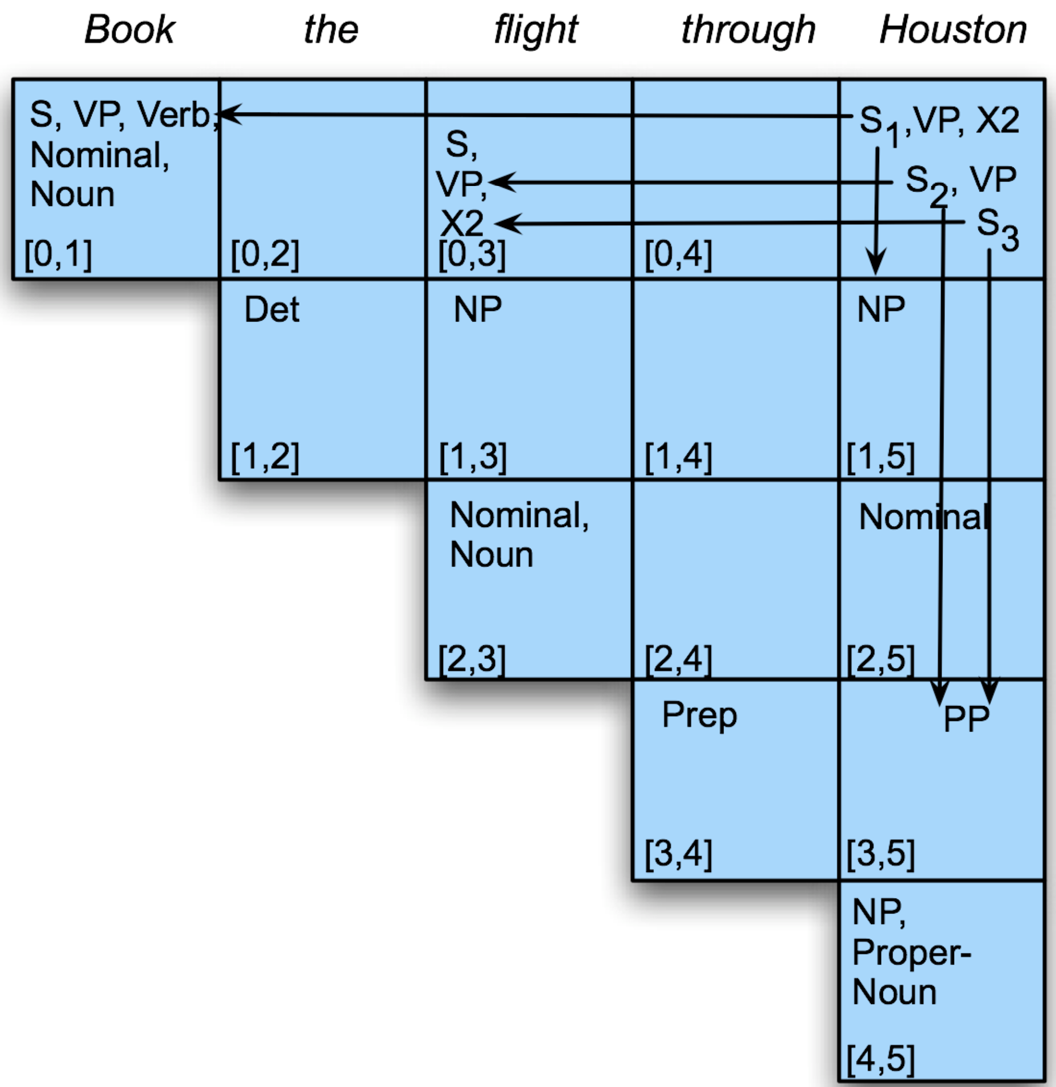


Book
the
flight
through
Houston
S, VP, Verb, Nominal, Noun
[0,1]
[0,2]
S,VP,X2
[0,3]
[0,4]
?
[0,5]
Det
[1,2]
NP
[1,3]
[1,4]
?
[1,5]
Nominal, Noun
[2,3]
[2,4]
Nominal
[2,5]
Prep
[3,4]
PP
[3,5]
NP, Proper-Noun
[4,5]

CKY: Example


Book
the
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]
NP
[1,5]
Nominal, Noun
[2,3]
[2,4]
Nominal
[2,5]
Prep
[3,4]
PP
[3,5]
NP, Proper-Noun
[4,5]

CKY: Example



Book
the
flight
through
Houston
S, VP, Verb, Nominal, Noun
[0,1]
[0,2]
S, VP, X2
[0,3]
[0,4]
S1,VP, X2
S2, VP
S3
Det
[1,2]
NP
[1,3]
[1,4]
NP
[1,5]
Nominal, Noun
[2,3]
[2,4]
Nominal
[2,5]
Prep
[3,4]
PP
[3,5]
NP, Proper-Noun
[4,5]

CKY: Algorithmic Complexity

- What's the asymptotic complexity of CKY?

CKY: Analysis

- Since it’s bottom up, CKY populates the table with a lot of “phantom constituents”
  - Spans that are constituents, but cannot really occur in the context in which they are suggested
- Conversion of grammar to CNF adds additional non-terminal nodes
  - Leads to weak equivalence wrt original grammar
  - Additional terminal nodes not (linguistically) meaningful: but can be cleaned up with post processing
- Is there a parsing algorithm for arbitrary CFGs that combines dynamic programming and top-down control?

Earley Parsing

- Dynamic programming algorithm (surprise)
- Allows arbitrary CFGs
- Top-down control
  - But, compare with naïve top-down search
- Fills a chart in a single sweep over the input
  - Chart is an array of length $N + 1$, where N = number of words
  - Chart entries represent states:
    - Completed constituents and their locations
    - In-progress constituents
    - Predicted constituents

Chart Entries: States

- Charts are populated with states
- Each state contains three items of information:
  - A grammar rule
  - Information about progress made in completing the sub-tree represented by the rule
  - Span of the sub-tree

Chart Entries: State Examples

- S → • VP [0,0]
  - A VP is predicted at the start of the sentence
- NP → Det • Nominal [1,2]
  - An NP is in progress; the Det goes from 1 to 2
- VP → V NP • [0,3]
  - A VP has been found starting at 0 and ending at 3

Earley in a nutshell

- Start by predicting S
- Step through chart:
  - New predicted states are created from current states
  - New incomplete states are created by advancing existing states as new constituents are discovered
  - States are completed when rules are satisfied
- Termination: look for $S \rightarrow \alpha \bullet [0, N]$

Earley Algorithm

function EARLEY-PARSE(*words, grammar*) **returns** *chart*

ENQUEUE(($\gamma \rightarrow \bullet S$, [0,0]), *chart[0]*)
for $i \leftarrow$ **from** 0 **to** LENGTH(*words*) **do**
  for each *state* **in** *chart*[i] **do**
    if INCOMPLETE?(*state*) **and**
        NEXT-CAT(*state*) is not a part of speech **then**
      PREDICTOR(*state*)
    elseif INCOMPLETE?(*state*) **and**
        NEXT-CAT(*state*) is a part of speech **then**
      SCANNER(*state*)
    else
      COMPLETER(*state*)
  end
end
return(*chart*)

Earley Algorithm

procedure PREDICTOR(($A \rightarrow \alpha \bullet B\, \beta,\ [i,j]$))
  for each $(B \rightarrow \gamma)$ **in** GRAMMAR-RULES-FOR(B, *grammar*) **do**
    ENQUEUE(($B \rightarrow \bullet\, \gamma,\ [j,j]$), *chart*[$j$])
  end

procedure SCANNER(($A \rightarrow \alpha \bullet B\, \beta,\ [i,j]$))
  if B $\subset$ PARTS-OF-SPEECH(*word[j]*) **then**
    ENQUEUE(($B \rightarrow word[j],\ [j, j+1]$), *chart[j+1]*)

procedure COMPLETER(($B \rightarrow \gamma \bullet,\ [j,k]$))
  for each $(A \rightarrow \alpha \bullet B\, \beta,\ [i,j])$ **in** *chart*[j] **do**
    ENQUEUE(($A \rightarrow \alpha\, B \bullet \beta,\ [i,k]$), *chart[k]*)
  end

Earley Example

- Input: Book that flight
- Desired end state: $S \rightarrow \alpha \bullet [0,3]$
  - Meaning: S spanning from 0 to 3, completed rule

Earley: Chart[0]

S0	$\gamma \rightarrow \bullet S$	[0,0]	Dummy start state
S1	$S \rightarrow \bullet NP\ VP$	[0,0]	Predictor
S2	$S \rightarrow \bullet Aux\ NP\ VP$	[0,0]	Predictor
S3	$S \rightarrow \bullet VP$	[0,0]	Predictor
S4	$NP \rightarrow \bullet Pronoun$	[0,0]	Predictor
S5	$NP \rightarrow \bullet Proper\text{-}Noun$	[0,0]	Predictor
S6	$NP \rightarrow \bullet Det\ Nominal$	[0,0]	Predictor
S7	$VP \rightarrow \bullet Verb$	[0,0]	Predictor
S8	$VP \rightarrow \bullet Verb\ NP$	[0,0]	Predictor
S9	$VP \rightarrow \bullet Verb\ NP\ PP$	[0,0]	Predictor
S10	$VP \rightarrow \bullet Verb\ PP$	[0,0]	Predictor
S11	$VP \rightarrow \bullet VP\ PP$	[0,0]	Predictor

Note that given a grammar, these entries are the same for all inputs; they can be pre-loaded…

Earley: Chart[1]

S12	$Verb \rightarrow book\,\bullet$	[0,1]	Scanner
S13	$VP \rightarrow Verb\,\bullet$	[0,1]	Completer
S14	$VP \rightarrow Verb \bullet NP$	[0,1]	Completer
S15	$VP \rightarrow Verb \bullet NP\ PP$	[0,1]	Completer
S16	$VP \rightarrow Verb \bullet PP$	[0,1]	Completer
S17	$S \rightarrow VP\,\bullet$	[0,1]	Completer
S18	$VP \rightarrow VP \bullet PP$	[0,1]	Completer
S19	$NP \rightarrow \bullet\, Pronoun$	[1,1]	Predictor
S20	$NP \rightarrow \bullet\, Proper\text{-}Noun$	[1,1]	Predictor
S21	$NP \rightarrow \bullet\, Det\ Nominal$	[1,1]	Predictor
S22	$PP \rightarrow \bullet\, Prep\ NP$	[1,1]	Predictor

Earley: Chart[2] and Chart[3]

S23	$Det \rightarrow that \bullet$	[1,2]	Scanner
S24	$NP \rightarrow Det \bullet Nominal$	[1,2]	Completer
S25	$Nominal \rightarrow \bullet Noun$	[2,2]	Predictor
S26	$Nominal \rightarrow \bullet Nominal\ Noun$	[2,2]	Predictor
S27	$Nominal \rightarrow \bullet Nominal\ PP$	[2,2]	Predictor
S28	$Noun \rightarrow flight \bullet$	[2,3]	Scanner
S29	$Nominal \rightarrow Noun \bullet$	[2,3]	Completer
S30	$NP \rightarrow Det\ Nominal \bullet$	[1,3]	Completer
S31	$Nominal \rightarrow Nominal \bullet Noun$	[2,3]	Completer
S32	$Nominal \rightarrow Nominal \bullet PP$	[2,3]	Completer
S33	$VP \rightarrow Verb\ NP \bullet$	[0,3]	Completer
S34	$VP \rightarrow Verb\ NP \bullet PP$	[0,3]	Completer
S35	$PP \rightarrow \bullet Prep\ NP$	[3,3]	Predictor
S36	$S \rightarrow VP \bullet$	[0,3]	Completer
S37	$VP \rightarrow VP \bullet PP$	[0,3]	Completer

Earley: Recovering the Parse

As with CKY, add backpointers…

Chart[1]	S12	*Verb* $\rightarrow$ *book* $\bullet$	[0,1]	Scanner
Chart[2]	S23	*Det* $\rightarrow$ *that* $\bullet$	[1,2]	Scanner
Chart[3]	S28	*Noun* $\rightarrow$ *flight* $\bullet$	[2,3]	Scanner
	S29	*Nominal* $\rightarrow$ *Noun* $\bullet$	[2,3]	(S28)
	S30	*NP* $\rightarrow$ *Det Nominal* $\bullet$	[1,3]	(S23, S29)
	S33	*VP* $\rightarrow$ *Verb NP* $\bullet$	[0,3]	(S12, S30)
	S36	*S* $\rightarrow$ *VP* $\bullet$	[0,3]	(S33)

Earley: Efficiency

- For such a simple example, there seems to be a lot of useless stuff…
- Why?

Back to Ambiguity

- Did we solve it?
- No: both CKY and Earley return multiple parse trees…
  - Plus: compact encoding with shared sub-trees
  - Plus: work deriving shared sub-trees is reused
  - Minus: neither algorithm tells us which parse is correct

Ambiguity

- Why don’t humans usually encounter ambiguity?
- How can we improve our models?

What we covered today..

- Parsing is (surprise) a search problem
- Two important issues:
  - Ambiguity
  - Shared sub-problems
- Two basic (= bad) algorithms:
  - Top-down search
  - Bottom-up search
- Two “real” algorithms:
  - CKY parsing
  - Earley parsing