

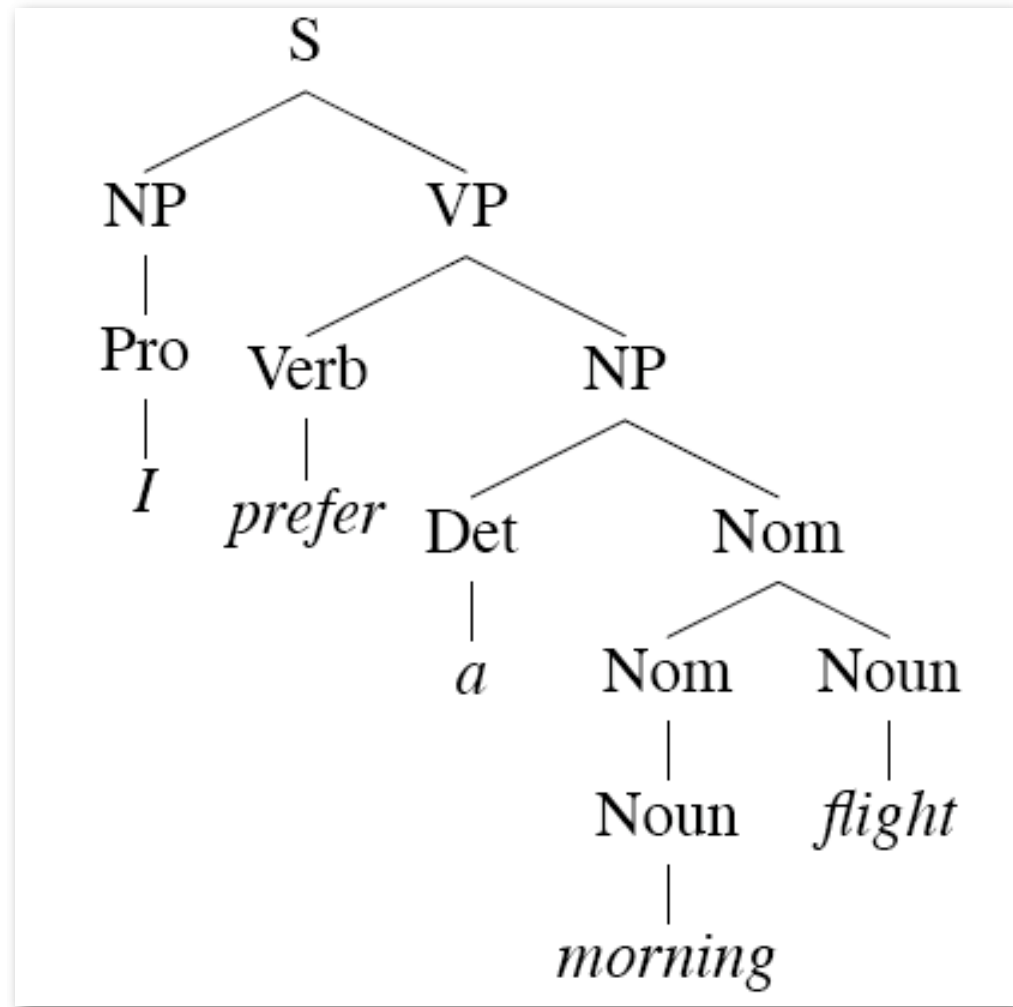
# Parsing

---

borrowing from  
Daniel Jurafsky and James Martin

# Derivations

- A derivation is a sequence of rules applied to a string that accounts for that string
  - Covers all the elements in the string
  - Covers only the elements in the string



# Parsing

- Parsing with CFGs refers to the task of assigning proper trees to input strings
- Proper here means a tree that covers **all and only the elements of the input** and **has an S at the top**
- It doesn't actually mean that the system can select the correct tree from among all the possible trees

# Parsing

- As with everything of interest, parsing involves a **search** which involves the making of choices
- We'll start with some basic (meaning bad) methods before moving on to more realistic ones

# This chunk

- Parsing with CFGs
  - Bottom-up, top-down
  - Ambiguity
  - CKY parsing
  - Earley parsing

# L0 Grammar

Grammar Rules	Examples
$S \rightarrow NP VP$	I + want a morning flight
$NP \rightarrow$	I
<i>Pronoun</i>	Los Angeles
<i>Proper-Noun</i>	a + flight
<i>Det Nominal</i>	morning + flight
$Nominal \rightarrow$	flights
<i>Nominal Noun</i>	
<i>Noun</i>	
$VP \rightarrow$	do
<i>Verb</i>	want + a flight
<i>Verb NP</i>	leave + Boston + in the morning
<i>Verb NP PP</i>	leaving + on Thursday
<i>Verb PP</i>	
$PP \rightarrow$	from + Los Angeles
<i>Preposition NP</i>	

# For Now

- Assume...

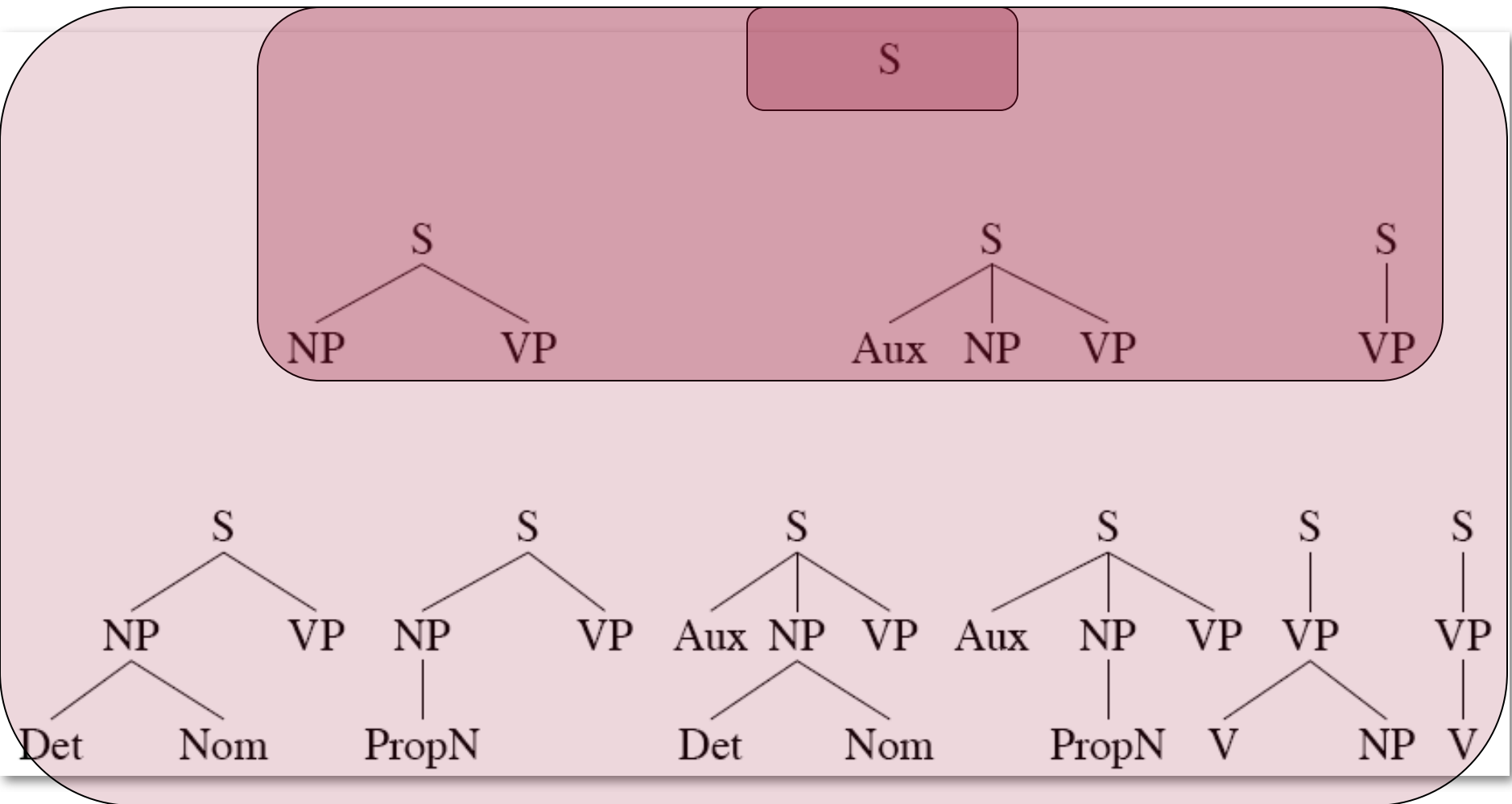
- You have all the words already in some buffer
  - The input isn't POS tagged
  - We won't worry about morphological analysis
  - All the words are known
- 
- These are all problematic in various ways, and would have to be addressed in real applications.

# Top-Down Search

- Since we're trying to find trees rooted with an S (Sentences), why not start with the rules that give us an S.
- Then we can work our way down from there to the words.



# Top Down Space



# Bottom-Up Parsing

- Of course, we also want trees that cover the input words. So we might also start with trees that link up with the words in the right way.
- Then work your way up from there to larger and larger trees.

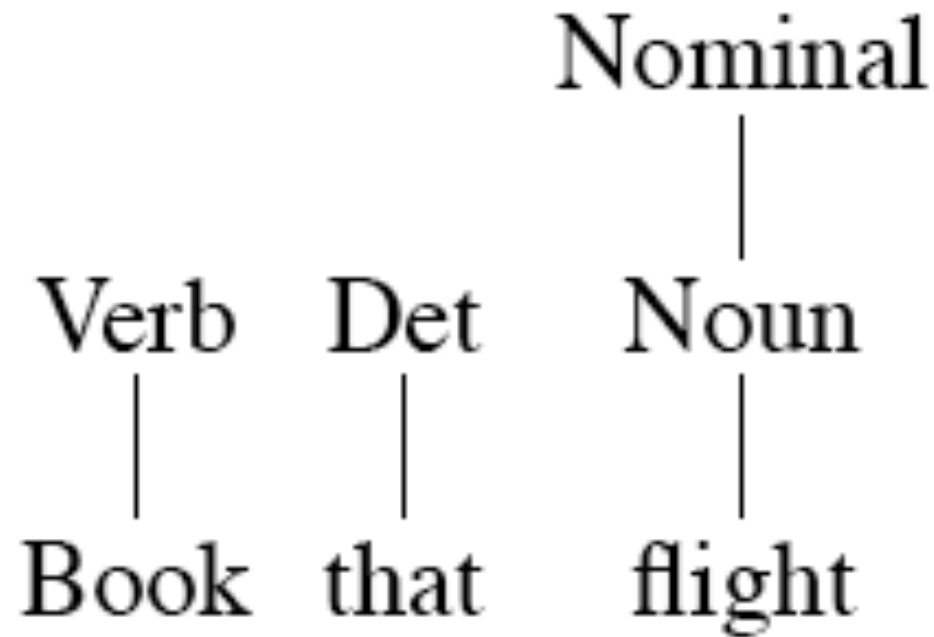
# Bottom-Up Search

Book that flight

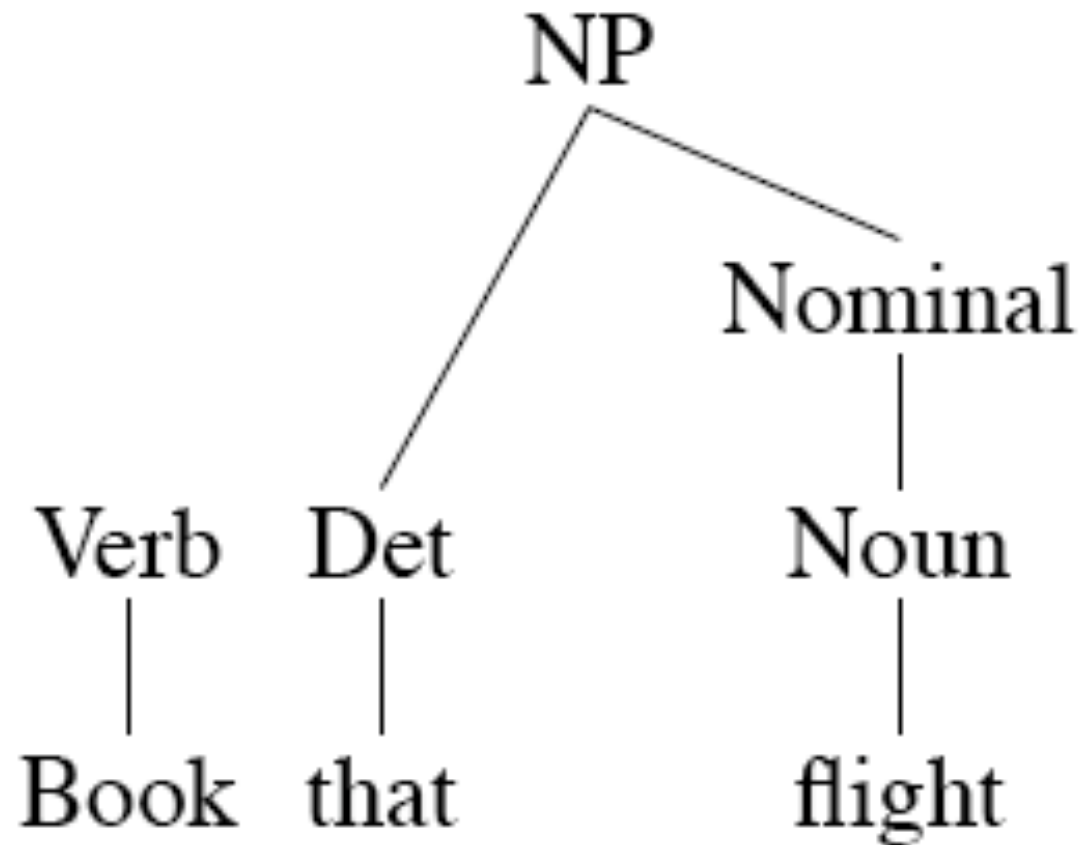
# Bottom-Up Search

Verb	Det	Noun
Book	that	flight

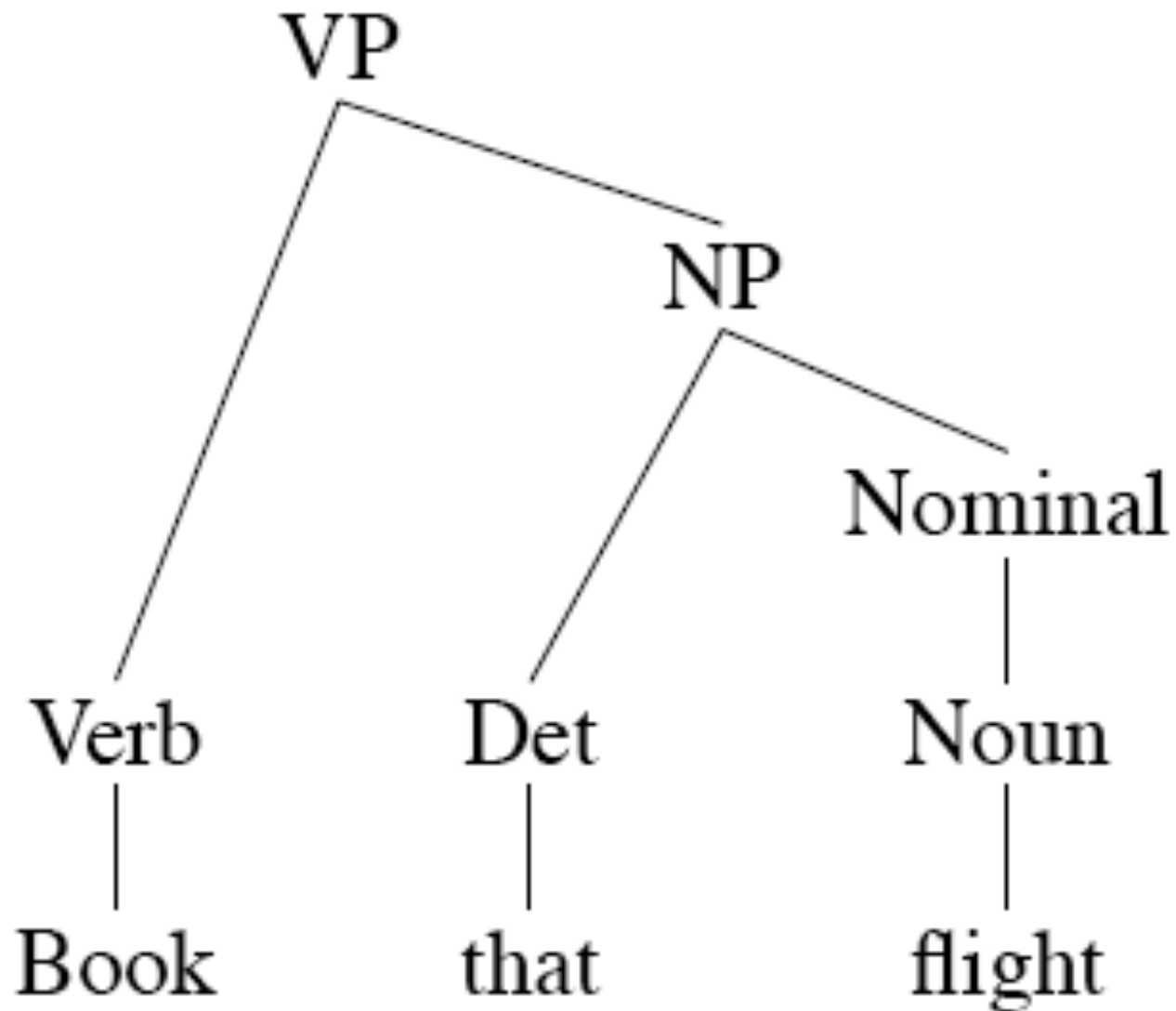
# Bottom-Up Search



# Bottom-Up Search



# Bottom-Up Search



# Top-Down and Bottom-Up

- **Top-down**

- Only searches for trees that can be answers (i.e. S's)
- But also suggests trees that are not consistent with any of the words

- **Bottom-up**

- Only forms trees consistent with the words
- But suggests trees that make no sense globally



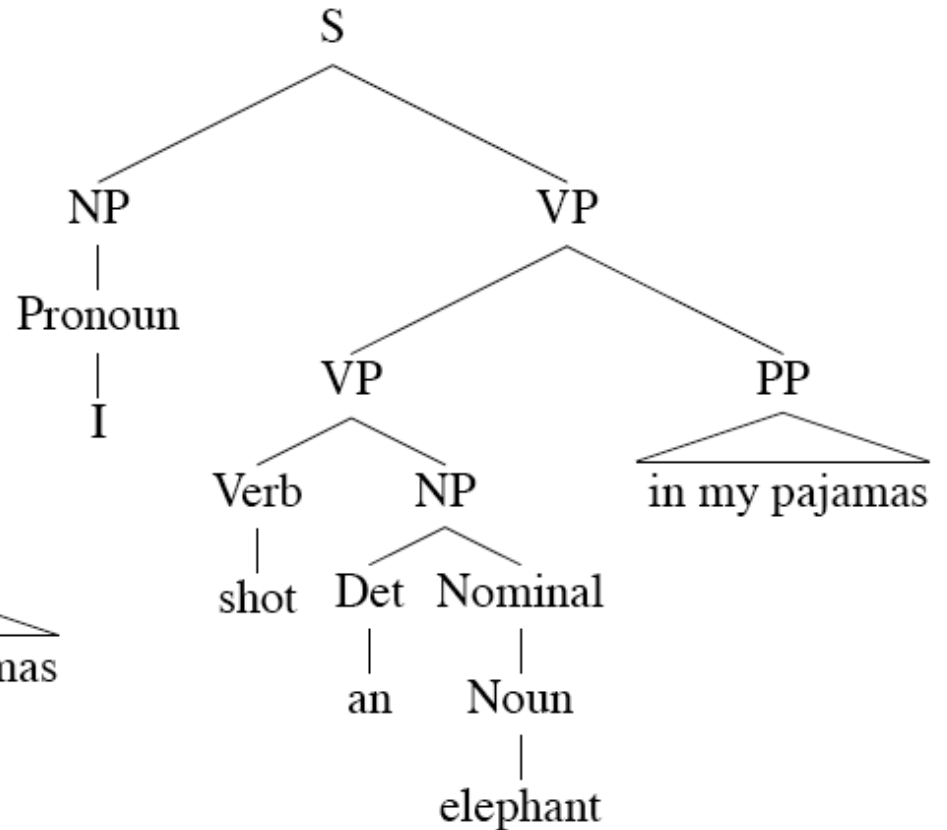
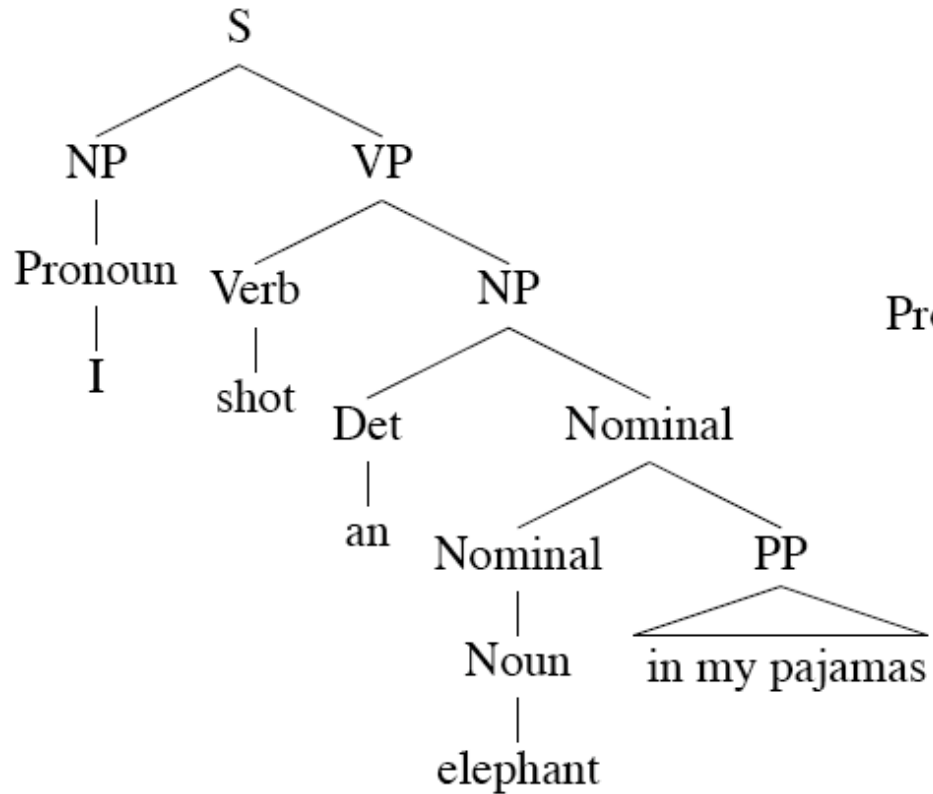
# Control

- Of course, in both cases we left out how to keep track of the search space and how to make choices
  - Which node to try to expand next
  - Which grammar rule to use to expand a node
- One approach is called backtracking.
  - Make a choice, if it works out then fine
  - If not then back up and make a different choice

# Problems

- Even with the best filtering, backtracking methods are doomed because of two inter-related problems
  - Ambiguity
  - Shared subproblems

# Ambiguity

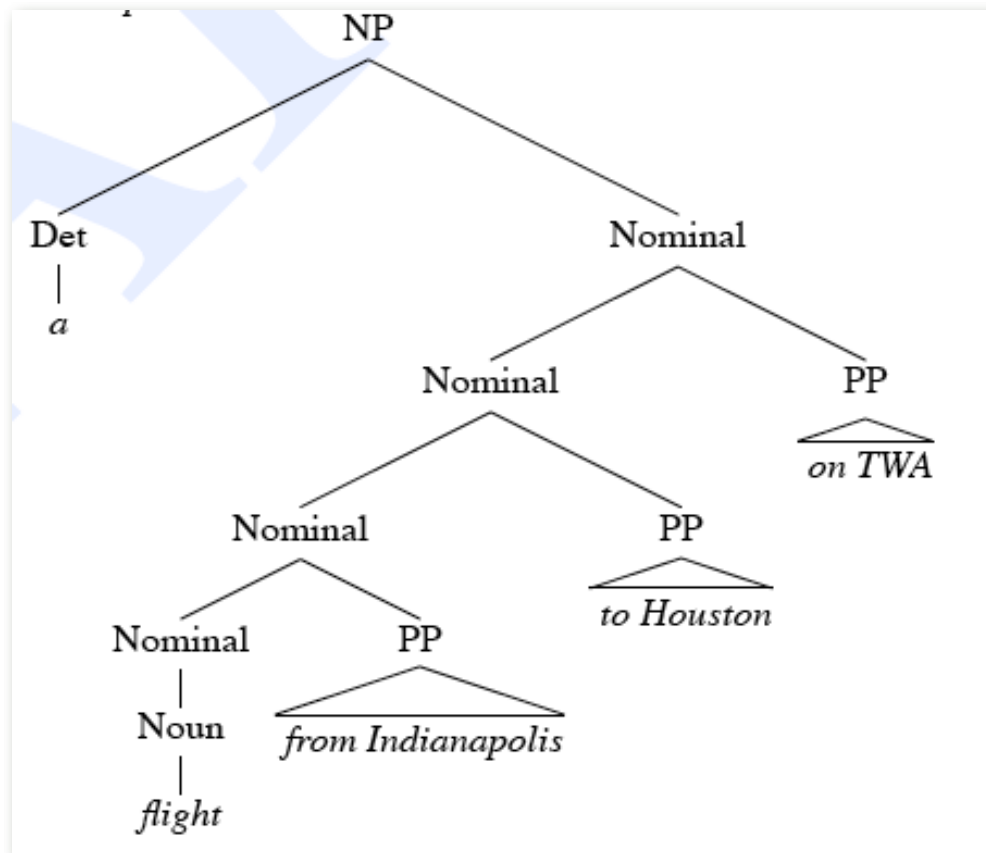


# Shared Sub-Problems

- No matter what kind of search (top-down or bottom-up or mixed) that we choose.
  - We don't want to redo work we've already done.
  - Unfortunately, naïve backtracking will lead to duplicated work.

# Shared Sub-Problems

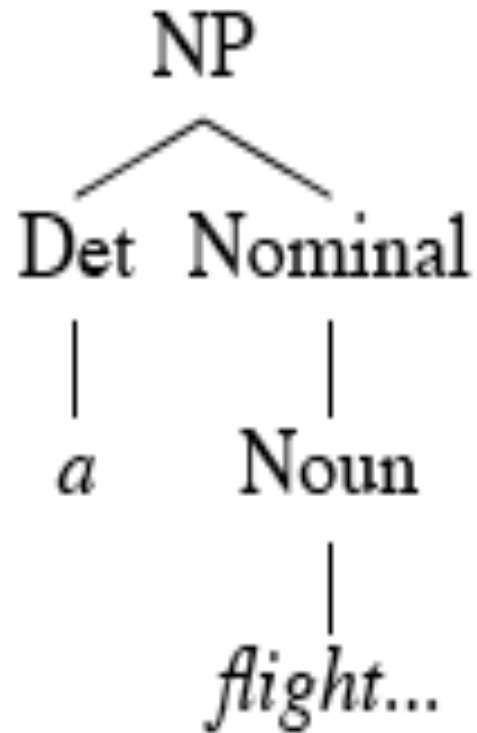
- Consider
  - A flight from Indianapolis to Houston on TWA



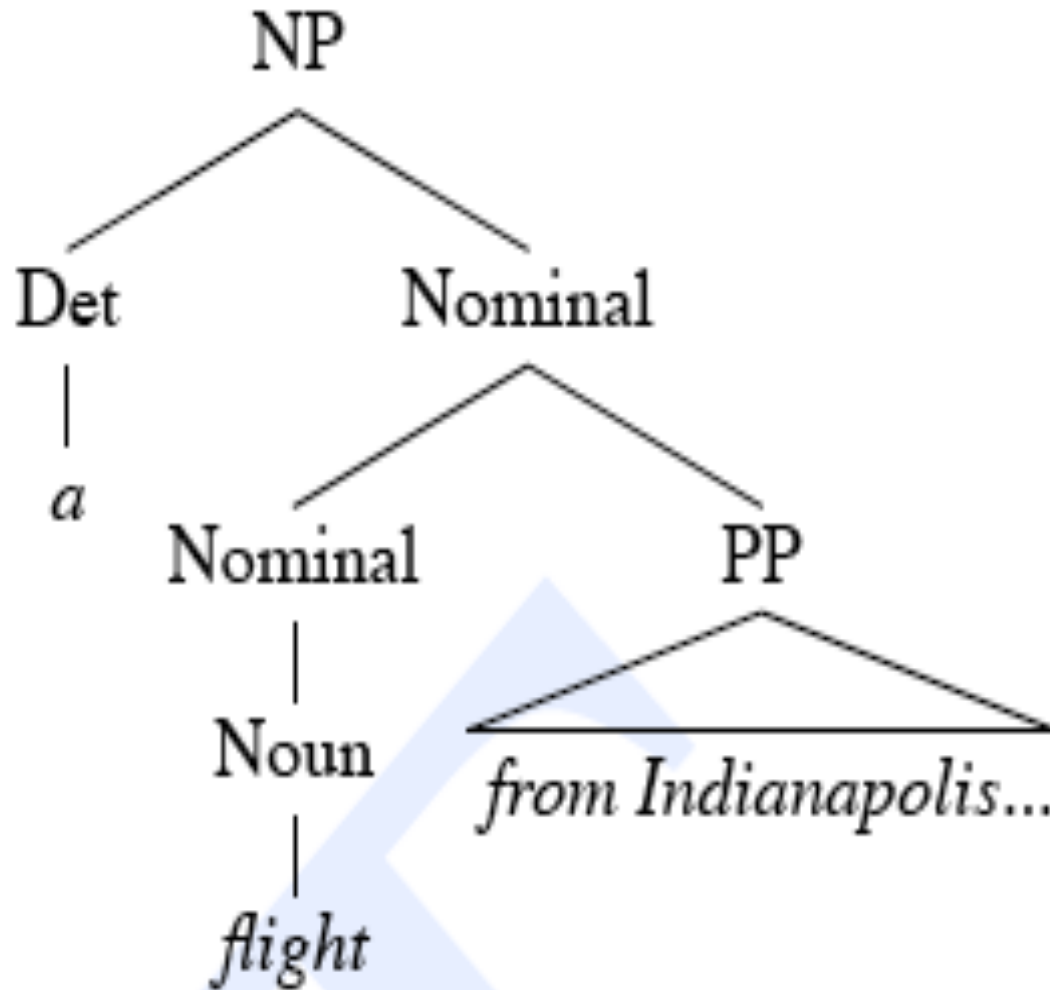
# Shared Sub-Problems

- Assume a top-down parse making choices among the various Nominal rules.
- In particular, between these two
  - Nominal -> Noun
  - Nominal -> Nominal PP
- Statically choosing the rules in this order leads to the following bad results...

# Shared Sub-Problems

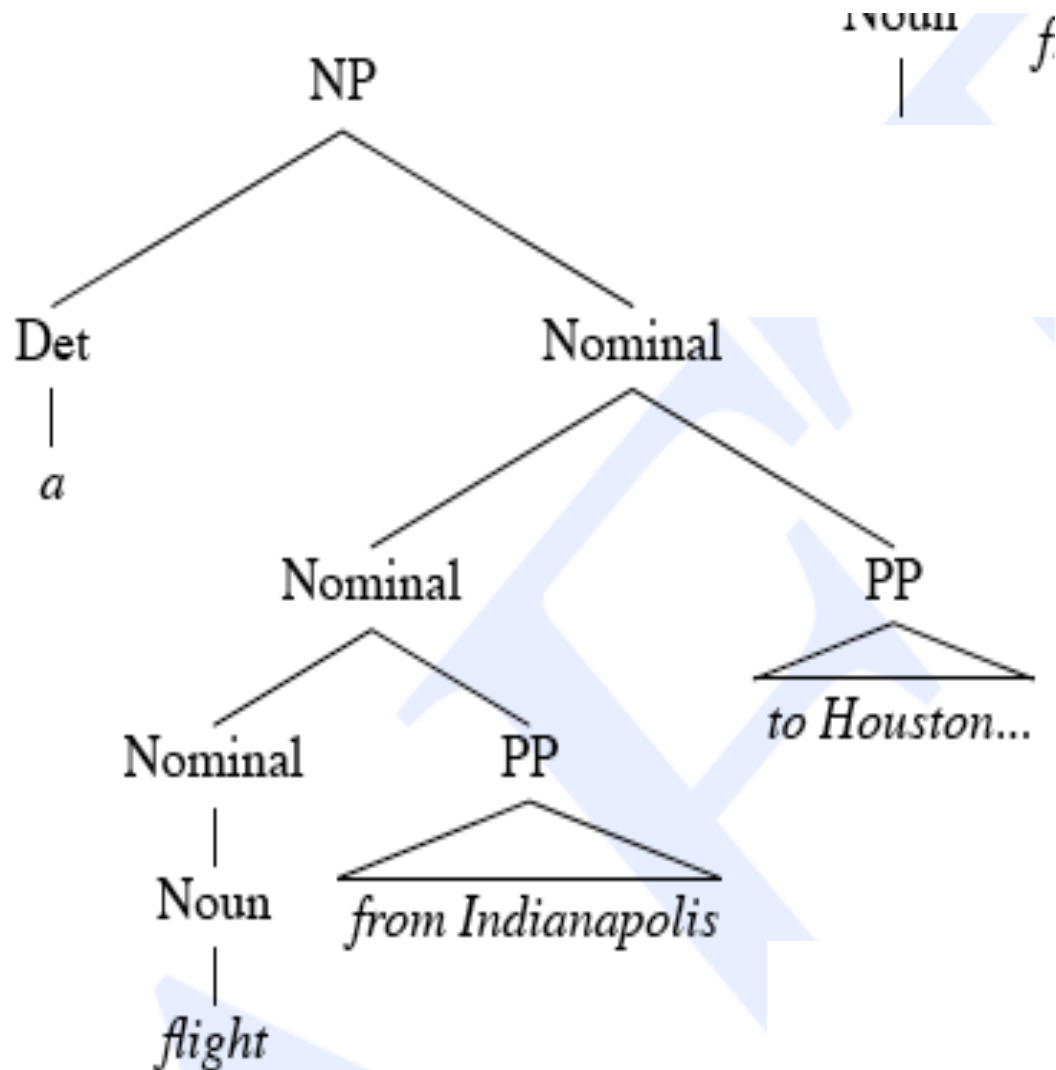


# Shared Sub-Problems

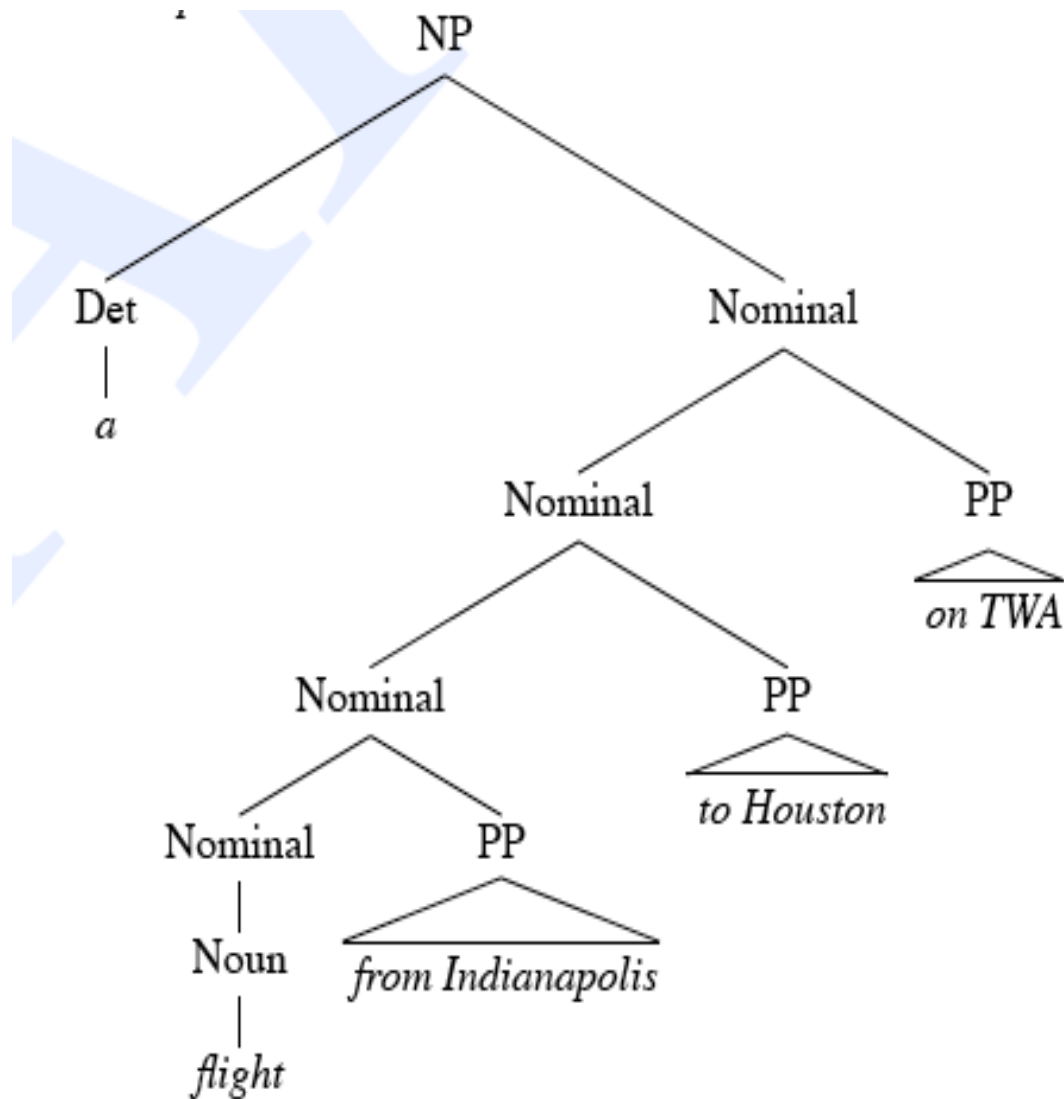




# Shared Sub-Problems



# Shared Sub-Problems



# Dynamic Programming

- DP search methods fill tables with partial results and thereby
  - Avoid doing avoidable repeated work
  - Solve exponential problems in polynomial time
  - Efficiently store ambiguous structures with shared sub-parts.
- We'll cover two approaches that roughly correspond to top-down and bottom-up approaches.
  - CKY
  - Earley

# CKY Parsing

- First we'll limit our grammar to epsilon-free, binary rules (more later)
- Consider the rule  $A \rightarrow BC$ 
  - If there is an  $A$  somewhere in the input then there must be a  $B$  followed by a  $C$  in the input.
  - If the  $A$  spans from  $i$  to  $j$  in the input then there must be some  $k$  st.  $i < k < j$ 
    - Ie. The  $B$  splits from the  $C$  someplace.

# Problem

- What if your grammar isn't binary?
  - As in the case of the TreeBank grammar?
- Convert it to binary... any arbitrary CFG can be rewritten into Chomsky-Normal Form automatically.
- What does this mean?
  - The resulting grammar accepts (and rejects) the same set of strings as the original grammar.
  - **But** the resulting derivations (trees) are different.

# Problem

- More specifically, we want our rules to be of the form

$$A \rightarrow B C$$

Or

$$A \rightarrow w$$

That is, rules can expand to either 2 non-terminals or to a single terminal.

# Binarization Intuition

- Eliminate chains of unit productions.
- Introduce new intermediate non-terminals into the grammar that distribute rules with **length > 2** over several rules.
  - So...  $S \rightarrow A B C$  turns into  
 $S \rightarrow X C$  and  
 $X \rightarrow A B$   
Where  $X$  is a symbol that doesn't occur anywhere else in the the grammar.

# Sample L1 Grammar

## Grammar

$S \rightarrow NP VP$   
 $S \rightarrow Aux NP VP$   
 $S \rightarrow VP$   
 $NP \rightarrow Pronoun$   
 $NP \rightarrow Proper-Noun$   
 $NP \rightarrow Det Nominal$   
 $Nominal \rightarrow Noun$   
 $Nominal \rightarrow Nominal Noun$   
 $Nominal \rightarrow Nominal PP$   
 $VP \rightarrow Verb$   
 $VP \rightarrow Verb NP$   
 $VP \rightarrow Verb NP PP$   
 $VP \rightarrow Verb PP$   
 $VP \rightarrow VP PP$   
 $PP \rightarrow Preposition NP$

## Lexicon

$Det \rightarrow that \mid this \mid a$   
 $Noun \rightarrow book \mid flight \mid meal \mid money$   
 $Verb \rightarrow book \mid include \mid prefer$   
 $Pronoun \rightarrow I \mid she \mid me$   
 $Proper-Noun \rightarrow Houston \mid NWA$   
 $Aux \rightarrow does$   
 $Preposition \rightarrow from \mid to \mid on \mid near \mid through$



# CNF Conversion

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

# CKY

- So let's build a table so that an  $A$  spanning from  $i$  to  $j$  in the input is placed in cell  $[i,j]$  in the table.
- So a non-terminal spanning an entire string will sit in cell  $[0, n]$ 
  - Hopefully an  $S$
- If we build the table bottom-up, we'll know that the parts of the  $A$  must go from  $i$  to  $k$  and from  $k$  to  $j$ , for some  $k$ .

# CKY

- Meaning that for a rule like  $A \rightarrow B C$  we should look for a  $B$  in  $[i,k]$  and a  $C$  in  $[k,j]$ .
- In other words, if we think there might be an  $A$  spanning  $i,j$  in the input... AND  $A \rightarrow B C$  is a rule in the grammar THEN
- There must be a  $B$  in  $[i,k]$  and a  $C$  in  $[k,j]$  for some  $i < k < j$

# CKY

- So to fill the table loop over the cell[i,j] values in some systematic way
  - What constraint should we put on that systematic search?
  - For each cell, loop over the appropriate k values to search for things to add.

# 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  
         $table[i, j] \leftarrow table[i, j] \cup$   
           $\{A \mid A \rightarrow BC \in grammar,$   
             $B \in table[i, k],$   
             $C \in table[k, j]\}$ 
```

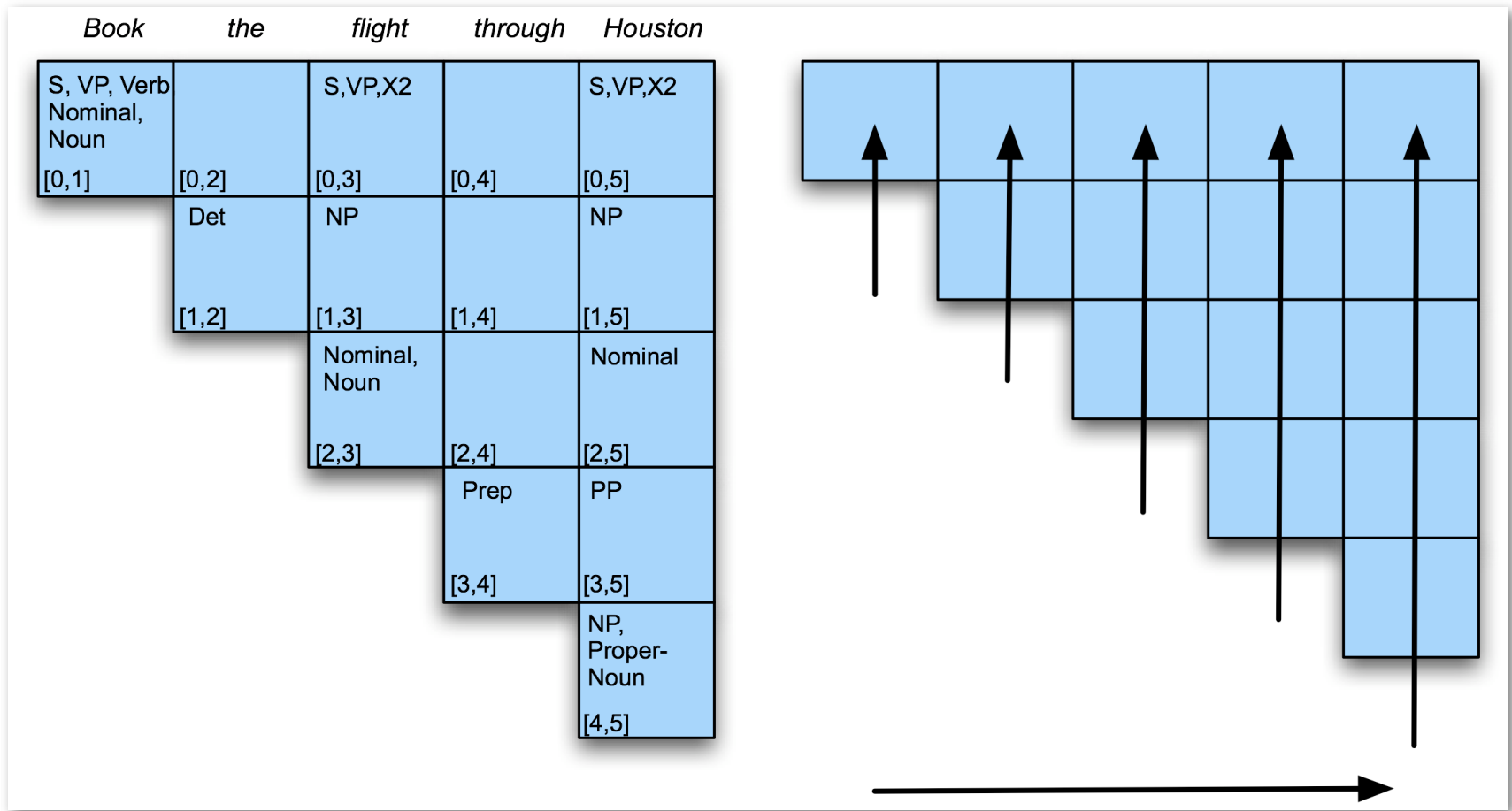
# CKY Parsing

- Is that really a parser?

# Note

- We arranged the loops to fill the table a column at a time, from left to right, bottom to top.
  - This assures us that whenever we're filling a cell, the parts needed to fill it are already in the table (to the left and below)
  - It's somewhat natural in that it processes the input a left to right a word at a time
    - Known as online

# Example





# Example

<i>Book</i>	<i>the</i>	<i>flight</i>	<i>through</i>	<i>Houston</i>
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]	[3,5]
				NP, Proper- Noun [4,5]

Filling column 5

# Example

<i>Book</i>	<i>the</i>	<i>flight</i>	<i>through</i>	<i>Houston</i>
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]	[2,5]
			Prep [3,4]	PP [3,5]
				NP, Proper- Noun [4,5]

Diagram illustrating the syntactic structure of the sentence "Book the flight through Houston". The structure is shown as a grid of cells, each containing a phrase structure rule (PSR) and its corresponding index range. Arrows indicate the flow of the derivation from the root node (S) down to the terminal words.

# Example

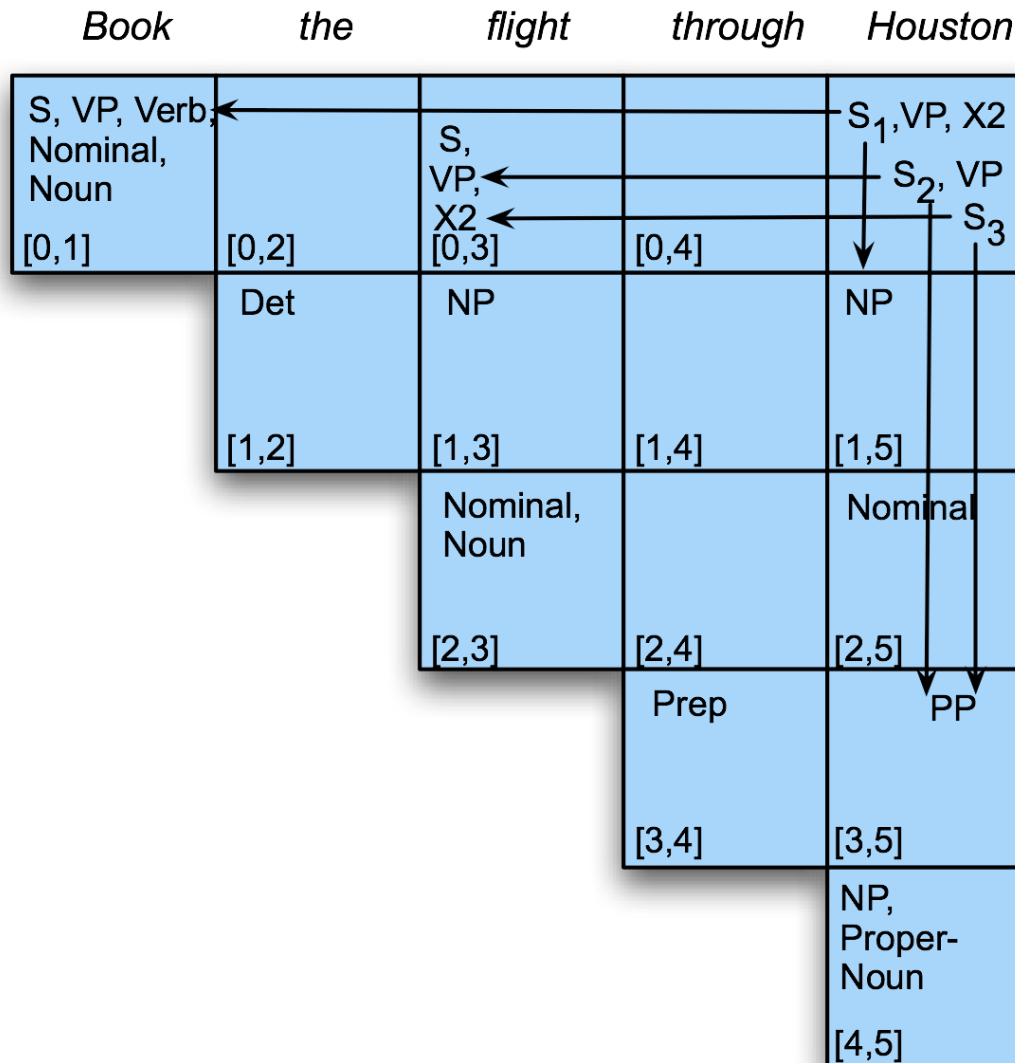
<i>Book</i>	<i>the</i>	<i>flight</i>	<i>through</i>	<i>Houston</i>
S, VP, Verb, Nominal, Noun [0,1]		S,VP,X2 [0,3]		
	Det [1,2]	NP [1,3]		NP [1,5]
		Nominal, Noun [2,3]		Nominal [2,5]
			Prep [3,4]	PP [3,5]
				NP, Proper- Noun [4,5]

Diagram illustrating the parsing of the sentence "Book the flight through Houston" using a shift-reduce parser. The table shows the state of the parser at each step, with the current state (S, VP, Verb, Nominal, Noun) and the current input (Book, the, flight, through, Houston). The arrows indicate the transitions between states.

# Example

<i>Book</i>	<i>the</i>	<i>flight</i>	<i>through</i>	<i>Houston</i>
S, VP, Verb, Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	[0,5]
	Det ←	NP		NP
	[1,2]	[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]

# Example



# CKY Notes

- Since it's bottom up, CKY populates the table with a lot of phantom constituents.
  - Segments that by themselves are constituents but cannot really occur in the context in which they are being suggested.
  - To avoid this we can switch to a top-down control strategy
  - Or we can add some kind of filtering that blocks constituents where they can not happen in a final analysis.

# Earley Parsing

- Allows arbitrary CFGs
- Top-down control
- Fills a table in a single sweep over the input
  - Table is length  $N+1$ ;  $N$  is number of words
  - Table entries represent
    - Completed constituents and their locations
    - In-progress constituents
    - Predicted constituents

# States

- The table-entries are called states and are represented with dotted-rules.

$S \rightarrow \cdot VP$

A VP is predicted

$NP \rightarrow Det \cdot Nominal$

An NP is in progress

$VP \rightarrow V NP \cdot$

A VP has been found



# States/Locations

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

# Earley

- As with most dynamic programming approaches, the answer is found by looking in the table in the right place.
- In this case, there should be an S state in the final column that spans from 0 to N and is complete. That is,
  - $S \rightarrow \alpha \bullet [0, N]$
- If that's the case you're done.

# Earley

- So sweep through the table from 0 to N...
  - New predicted states are created by starting top-down from S
  - New incomplete states are created by advancing existing states as new constituents are discovered
  - New complete states are created in the same way.

# Earley

- More specifically...
  1. **Predict** all the states you can upfront
  2. Read a word
    1. Extend states based on matches
    2. Generate new predictions
    3. Go to step 2
  3. When you're out of words, look at the chart to see if you have a winner

# Core Earley Code

```
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 Code

```
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  $B \subset \text{PARTS-OF-SPEECH}(word[j])$  then  
    ENQUEUE( $(B \rightarrow word[j], [j, j+1])$ ,  $chart[j+1]$ )  
  end  
  
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
```

# Example

- Book that flight
- We should find... an S from 0 to 3 that is a completed state...

# 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	$NP \rightarrow \bullet Proper-Noun$	[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.



# 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	$NP \rightarrow \bullet Proper-Noun$	[1,1]	Predictor
S21	$NP \rightarrow \bullet Det Nominal$	[1,1]	Predictor
S22	$PP \rightarrow \bullet Prep NP$	[1,1]	Predictor

# Charts[2] and [3]

S23	<i>Det</i> → <i>that</i> •	[1,2]	Scanner
S24	<i>NP</i> → <i>Det</i> • <i>Nominal</i>	[1,2]	Completer
S25	<i>Nominal</i> → • <i>Noun</i>	[2,2]	Predictor
S26	<i>Nominal</i> → • <i>Nominal Noun</i>	[2,2]	Predictor
S27	<i>Nominal</i> → • <i>Nominal PP</i>	[2,2]	Predictor
S28	<i>Noun</i> → <i>flight</i> •	[2,3]	Scanner
S29	<i>Nominal</i> → <i>Noun</i> •	[2,3]	Completer
S30	<i>NP</i> → <i>Det Nominal</i> •	[1,3]	Completer
S31	<i>Nominal</i> → <i>Nominal</i> • <i>Noun</i>	[2,3]	Completer
S32	<i>Nominal</i> → <i>Nominal</i> • <i>PP</i>	[2,3]	Completer
S33	<i>VP</i> → <i>Verb NP</i> •	[0,3]	Completer
S34	<i>VP</i> → <i>Verb NP</i> • <i>PP</i>	[0,3]	Completer
S35	<i>PP</i> → • <i>Prep NP</i>	[3,3]	Predictor
S36	<i>S</i> → <i>VP</i> •	[0,3]	Completer
S37	<i>VP</i> → <i>VP</i> • <i>PP</i>	[0,3]	Completer

# Efficiency

- For such a simple example, there seems to be a lot of useless stuff in there.
- Why?
  - It's predicting things that aren't consistent with the input
  - That's the flipside to the CKY problem.

# Details

- As with CKY that isn't a parser until we add the backpointers so that each state knows where it came from.

# Back to Ambiguity

- Did we solve it?

# Ambiguity

- No...

- Both CKY and Earley will result in multiple **S** structures for the **[0,N]** table entry.
- They both efficiently store the sub-parts that are shared between multiple parses.
- And they obviously avoid re-deriving those sub-parts.
- But neither can tell us which one is right.

# Ambiguity

- In most cases, humans don't notice incidental ambiguity (lexical or syntactic). It is resolved on the fly and never noticed.
- We can model that with probabilities.