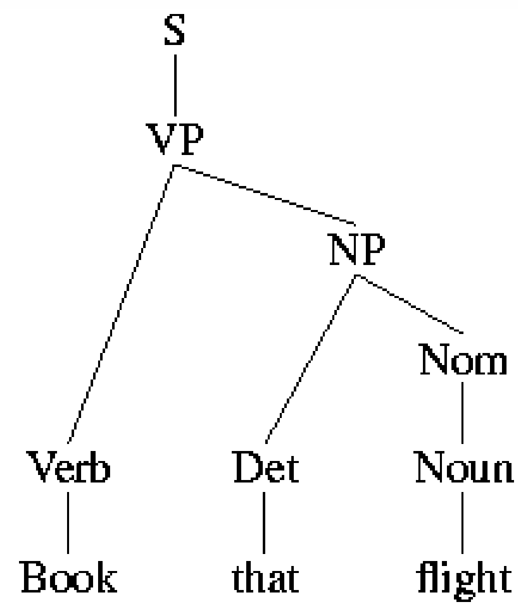
Syntactic Parsing

Background

- Syntactic parsing
  - The task of recognizing a sentence and assigning a syntactic structure to it
- Since CFGs are a declarative formalism, they do not specify how the parse tree for a given sentence should be computed.
- Parse trees are useful in applications such as
  - Grammar checking
  - Semantic analysis
  - Machine translation
  - Question answering
  - Information extraction

Parsing as Search

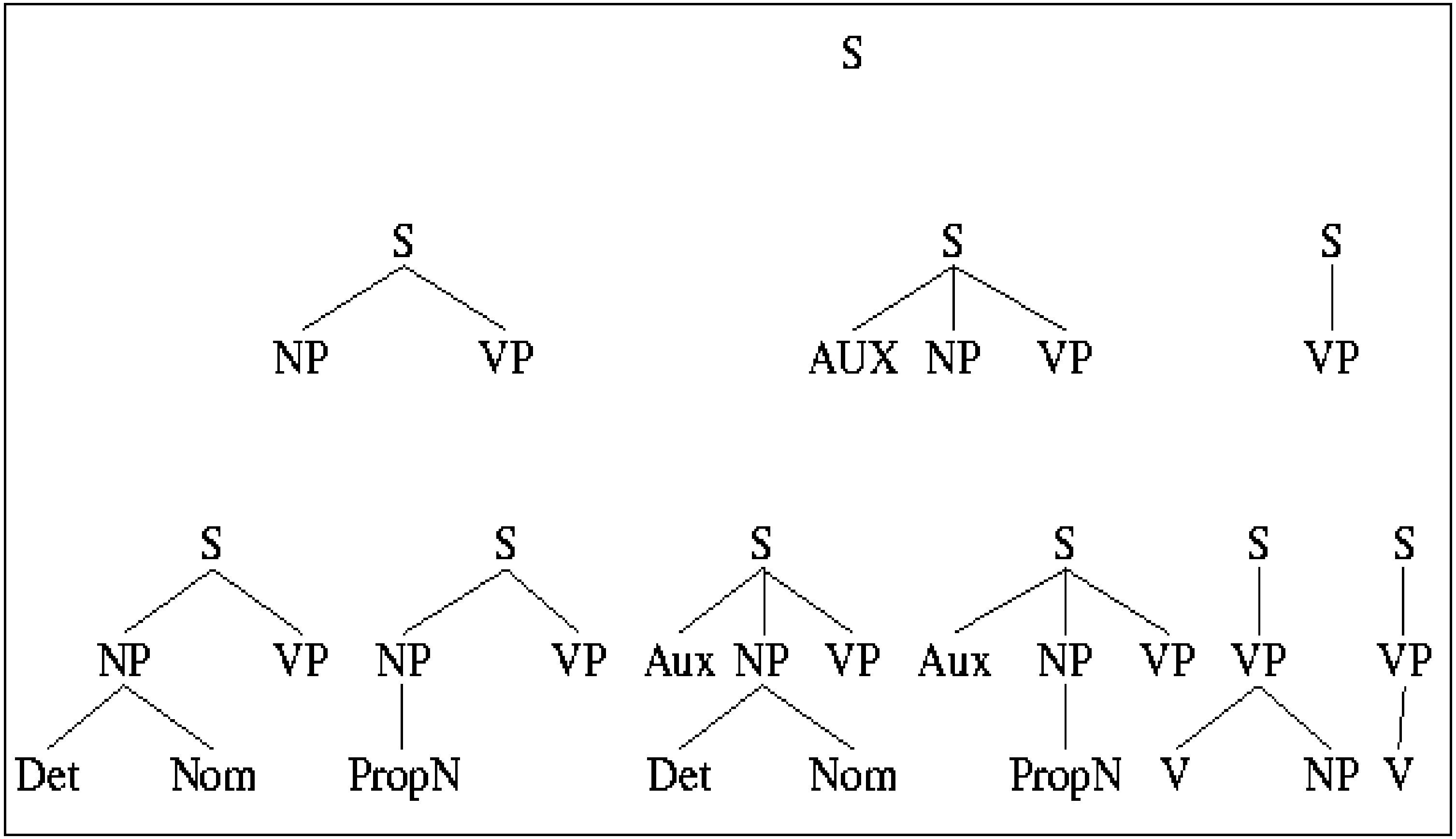
- The parser can be viewed as searching through the space of all possible parse trees to find the correct parse tree for the sentence.
- ***How can we use the grammar to produce the parse tree?***



$S \rightarrow NP\ VP$	$Det \rightarrow that \mid this \mid a$
$S \rightarrow Aux\ NP\ VP$	$Noun \rightarrow book \mid flight \mid meal \mid money$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid prefer$
$NP \rightarrow Det\ Nominal$	$Aux \rightarrow does$
$Nominal \rightarrow Noun$	
$Nominal \rightarrow Noun\ Nominal$	$Prep \rightarrow from \mid to \mid on$
$NP \rightarrow Proper\text{-}Noun$	$Proper\text{-}Noun \rightarrow Houston \mid TWA$
$VP \rightarrow Verb$	
$VP \rightarrow Verb\ NP$	$Nominal \rightarrow Nominal\ PP$

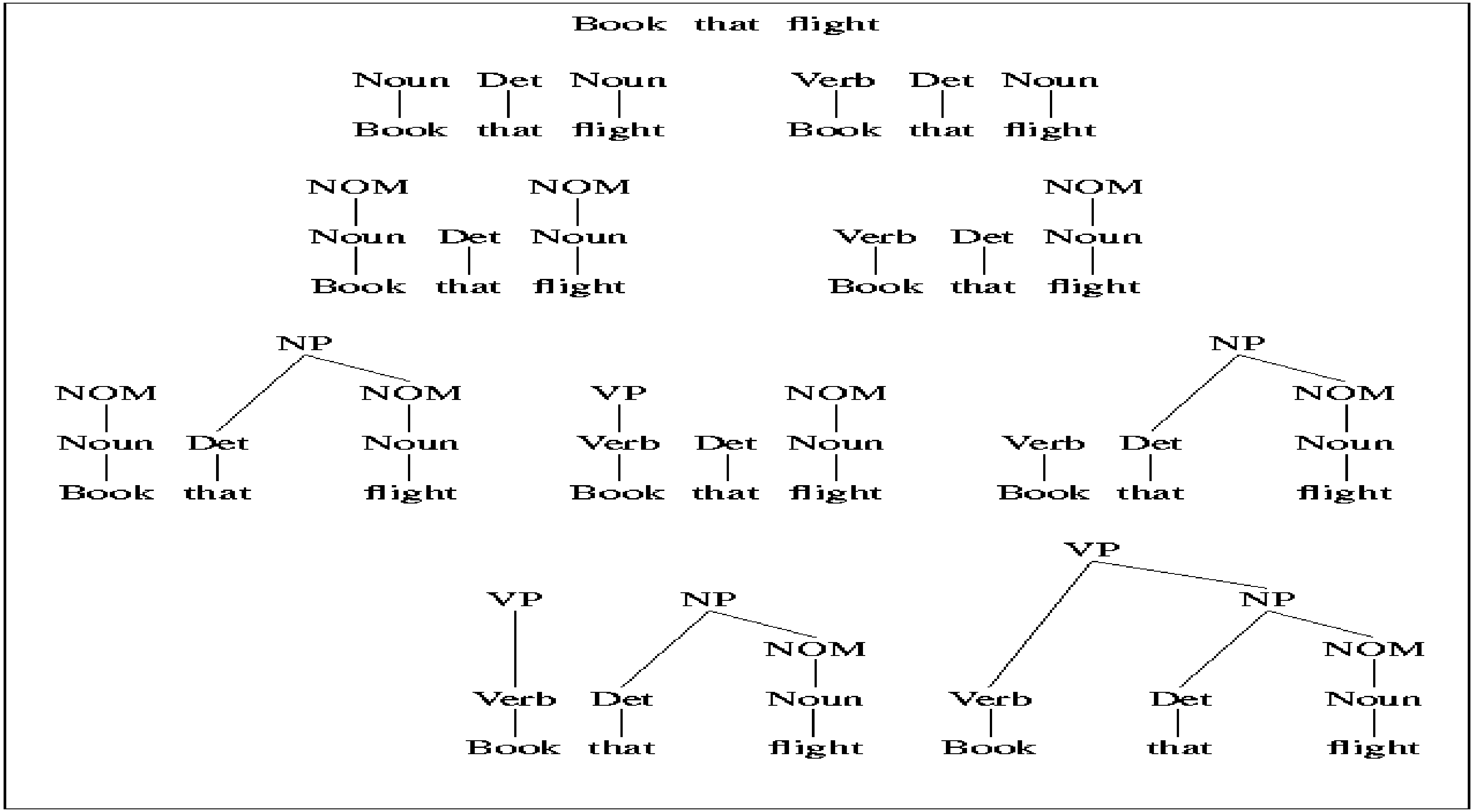
Parsing as Search

- Top-down parsing



Parsing as Search

- Bottom-up parsing



Parsing as Search

- Comparisons
  - The top-down strategy never wastes time exploring trees that cannot result in an *S*.
  - The bottom-up strategy, by contrast, trees that have no hope to leading to an *S*, or fitting in with any of their neighbors, are generated with wild abandon.
  - Spend considerable effort on *S* trees that are not consistent with the input.

Problems with the Basic Top-Down Parser

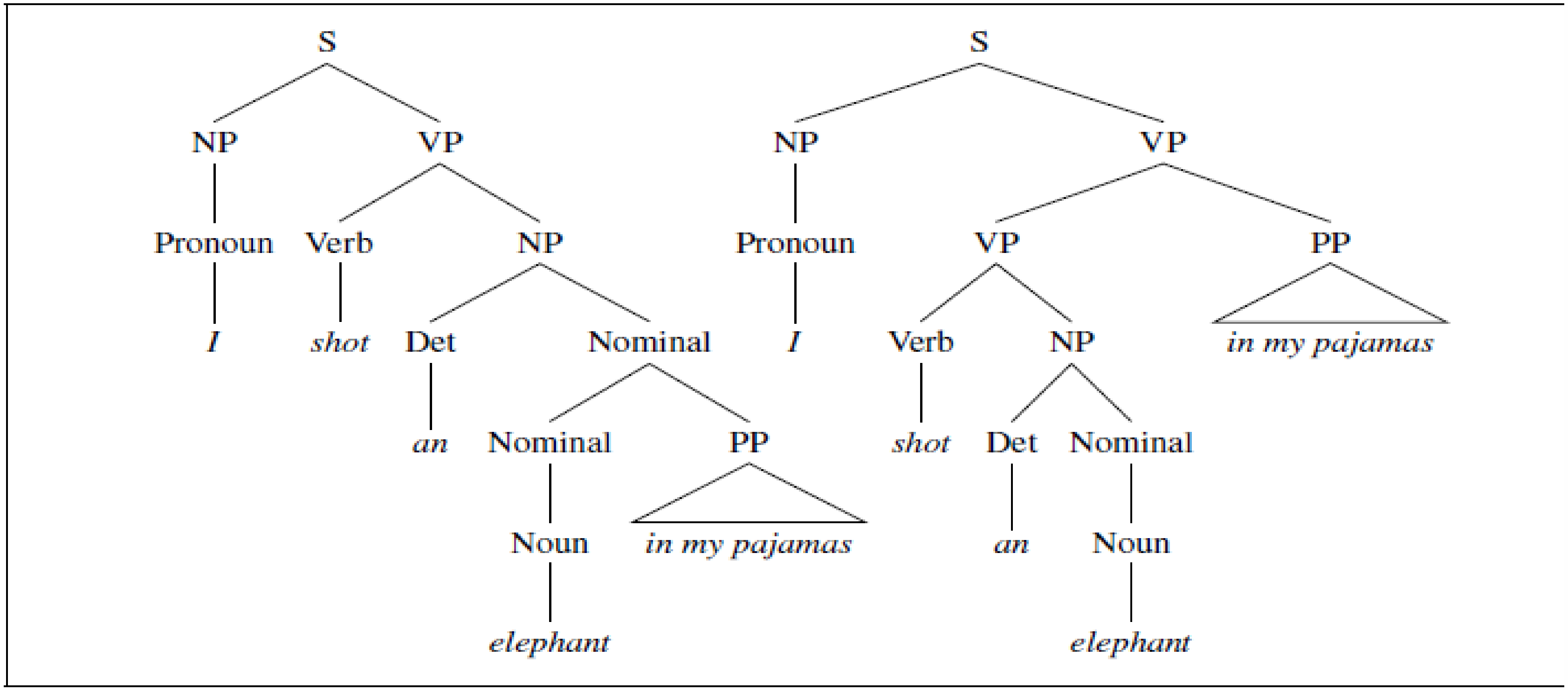
- Problems with the top-down parser
  - Left-recursion
  - Ambiguity
  - Inefficiency reparsing of subtrees
- Introducing the Earley and CYK algorithm

Ambiguity

- Common structural ambiguity
  - Attachment ambiguity
    - A sentence has an attachment ambiguity if a particular constituent can be attached to the parse tree at more than one place.
    - Various kinds of adverbial phrases are also subject to this kind of ambiguity
  - Coordination ambiguity
    - Different sets of phrases can be conjoined by a conjunction like **and**.
    - For example, the phrase old men and women can be bracketed as [old [men and women]], referring to old men and old women, or as [old men] and [women], in which case it is only the men who are old.

Ambiguity

PP attachment ambiguity



Ambiguous because the phrase in my pajamas can be part of the NP headed by elephant or a part of the verb phrase headed by shot.

Ambiguity

We saw the Eiffel Tower flying to Paris.

- The gerundive-VP *flying to Paris* can be
  - part of a gerundive sentence, or
  - an adjunct modifying the VP

The CYK Algorithm

- ***The membership problem***
  - Problem:
    - Given a context-free grammar **G** and a string **w**
      - **G** = (V, ∑ ,P , S) where
        - V finite set of variables
        - ∑ (the alphabet) finite set of terminal symbols
        - P finite set of rules
        - S start symbol (distinguished element of V)
        - V and ∑ are assumed to be disjoint
      - **G** is used to generate the string of a language
  - Question:
    - Is **w** in **L**(**G**)?

The CYK Algorithm

- J. Cocke
- D. Younger,
- T. Kasami

  - Independently developed an algorithm to answer this question.

Dynamic Programming

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

The CYK Algorithm Basics

- The Structure of the rules in a Chomsky Normal Form grammar
- Uses a "dynamic programming" or "table-filling algorithm"
- Based on bottom-up parsing and requires first normalizing the grammar.
- **Earley parser** is based on top-down parsing and does not require normalizing grammar but is more complex.
- More generally, **chart parsers** retain completed phrases in a chart and can combine top-down and bottom-up search.

Chomsky Normal Form

- *Normal Form* is described by a set of conditions that each rule in the grammar must satisfy
- Context-free grammar is in CNF if each rule has one of the following forms:
  - $A \rightarrow BC$ at most 2 symbols on right side
  - $A \rightarrow a$, or terminal symbol
  - $S \rightarrow \lambda$ null string

  where $B, C \in V - \{S\}$

Construct a Triangular Table

- Each row corresponds to one length of substrings
  - Bottom Row – Strings of length 1
  - Second from Bottom Row – Strings of length 2

    .

    .
  - Top Row – string ‘w’

Construct a Triangular Table

- $\mathbf{X_{i,\,i}}$ is the set of variables A such that $A \rightarrow w_i$ is a production of G

- Compare at most n pairs of previously computed sets:

  $(\mathbf{X_{i,\,i}}, \mathbf{X_{i+1,\,j}}), (\mathbf{X_{i,\,i+1}}, \mathbf{X_{i+2,\,j}}) \ldots (\mathbf{X_{i,\,j-1}}, \mathbf{X_{j,\,j}})$

Construct a Triangular Table

$X_{1, 5}$				
$X_{1, 4}$	$X_{2, 5}$			
$X_{1, 3}$	$X_{2, 4}$	$X_{3, 5}$		
$X_{1, 2}$	$X_{2, 3}$	$X_{3, 4}$	$X_{4, 5}$	
$X_{1, 1}$	$X_{2, 2}$	$X_{3, 3}$	$X_{4, 4}$	$X_{5, 5}$
w_1	w_2	w_3	w_4	w_5

Table for string ‘**w**’ that has length 5

Construct a Triangular Table

$X_{1,5}$				
$X_{1,4}$	$X_{2,5}$			
$X_{1,3}$	$X_{2,4}$	$X_{3,5}$		
$X_{1,2}$	$X_{2,3}$	$X_{3,4}$	$X_{4,5}$	
$X_{1,1}$	$X_{2,2}$	$X_{3,3}$	$X_{4,4}$	$X_{5,5}$
w_1	w_2	w_3	w_4	w_5

Looking for pairs to compare

Example CYK Algorithm

- Show the CYK Algorithm with the following example:
  - CNF grammar **G**
    - S → AB | BC
    - A → BA | a
    - B → CC | b
    - C → AB | a
  - **w** is baaba
  - Question Is **baaba** in L(G)?

Constructing The Triangular Table

$S \rightarrow AB \mid BC$
$A \rightarrow BA \mid a$
$B \rightarrow CC \mid b$
$C \rightarrow AB \mid a$

{B}	**{A, C}**	**{A, C}**	**{B}**	**{A, C}**
b	**a**	**a**	**b**	**a**

Calculating the Bottom ROW

Constructing The Triangular Table

- $\mathbf{X_{1,2} = (X_{i,i}, X_{i+1,j}) = (X_{1,1}, X_{2,2})}$
- ➔ {B}{A,C} = {BA, BC}
- Steps:
  - Look for production rules to generate BA or BC
  - There are two: S and A
  - $\mathbf{X_{1,2} = \{S, A\}}$

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

Constructing The Triangular Table

{S, A}				
{B}	**{A, C}**	**{A, C}**	**{B}**	**{A, C}**
b	**a**	**a**	**b**	**a**

Constructing The Triangular Table

- $\mathbf{X_{2,3} = (X_{i,i}, X_{i+1,j}) = (X_{2,2}, X_{3,3})}$
- $\rightarrow$ {A, C}{A,C} = {AA, AC, CA, CC} = Y
- Steps:
  - Look for production rules to generate Y
  - There is one: B
  - $\mathbf{X_{2,3} = \{B\}}$

S $\rightarrow$ AB | BC
A $\rightarrow$ BA | a
B $\rightarrow$ CC | b
C $\rightarrow$ AB | a

Constructing The Triangular Table

{S, A}	{B}			
{B}	{A, C}	{A, C}	{B}	{A, C}
b	a	a	b	a

Constructing The Triangular Table

- $\mathbf{X_{3,4} = (X_{i,i}, X_{i+1,j}) = (X_{3,3}, X_{4,4})}$
- ➔ {A, C}{B} = {AB, CB} = Y
- Steps:
  - Look for production rules to generate Y
  - There are two: S and C
  - $\mathbf{X_{3,4} = \{S, C\}}$

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

Constructing The Triangular Table

{S, A}	{B}	{S, C}		
{B}	{A, C}	{A, C}	{B}	{A, C}
b	**a**	**a**	**b**	**a**

Constructing The Triangular Table

- $\mathbf{X_{4,5} = (X_{i,i}, X_{i+1,j}) = (X_{4,4}, X_{5,5})}$
- ➔ {B}{A, C} = {BA, BC} = Y
- Steps:
  - Look for production rules to generate Y
  - There are two: S and A
  - $\mathbf{X_{4,5} = \{S, A\}}$

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

Constructing The Triangular Table

{S, A}	**{B}**	**{S, C}**	**{S, A}**	
{B}	**{A, C}**	**{A, C}**	**{B}**	**{A, C}**
b	**a**	**a**	**b**	**a**

Constructing The Triangular Table

- $\mathbf{X_{1,3}} = \mathbf{(X_{i,i}, X_{i+1,j})\ (X_{i,i+1}, X_{i+2,j})}$
  $= (X_{1,1}, X_{2,3}), (X_{1,2}, X_{3,3})$
- ➔ {B}{B} **U** {S, A}{A, C}= {BB, SA, SC, AA, AC} = Y
- Steps:
  - Look for production rules to generate Y
  - There are NONE: S and A
  - $\mathbf{X_{1,3} = Ø}$
  - **no elements in this set (empty set)**

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

Constructing The Triangular Table

Ø				
{S, A}	**{B}**	**{S, C}**	**{S, A}**	
{B}	**{A, C}**	**{A, C}**	**{B}**	**{A, C}**
b	**a**	**a**	**b**	**a**

Constructing The Triangular Table

- $\mathbf{X_{2,4}} \quad = (\mathbf{X_{i,i}}, \mathbf{X_{i+1,j}})\ (\mathbf{X_{i,i+1}}, \mathbf{X_{i+2,j}})$
  $= (\mathbf{X_{2,2}}, \mathbf{X_{3,4}}), (\mathbf{X_{2,3}}, \mathbf{X_{4,4}})$
- ➔ {A, C}{S, C} **U** {B}{B}= {AS, AC, CS, CC, BB} = Y
- Steps:
  - Look for production rules to generate Y
  - There is one: B
  - $\mathbf{X_{2,4} = \{B\}}$

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

Constructing The Triangular Table

Ø	**{B}**			
{S, A}	**{B}**	**{S, C}**	**{S, A}**	
{B}	**{A, C}**	**{A, C}**	**{B}**	**{A, C}**
b	**a**	**a**	**b**	**a**

Constructing The Triangular Table

- $\mathbf{X_{3\,,\,5}} \quad = \mathbf{(X_{i\,,\,i}\,,X_{i+1\,,\,j})\;(X_{i\,,\,i+1}\,,X_{i+2\,,\,j})}$
  $= (X_{3\,,\,3}\;,X_{4\,,\,5})\;,\;(X_{3\,,\,4}\;,X_{5\,,\,5})$
- ➔ {A,C}{S,A} **U** {S,C}{A,C}
  = {AS, AA, CS, CA, SA, SC, CA, CC} = Y
- Steps:
  - Look for production rules to generate Y
  - There is one: B
  - $\mathbf{X_{3\,,\,5} = \{B\}}$

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

Constructing The Triangular Table

Ø	{B}	{B}		
{S, A}	{B}	{S, C}	{S, A}	
{B}	{A, C}	{A, C}	{B}	{A, C}
b	a	a	b	a

Final Triangular Table

{S, A, C}	← $X_{1,5}$			
Ø	{S, A, C}			
Ø	{B}	{B}		
{S, A}	{B}	{S, C}	{S, A}	
{B}	{A, C}	{A, C}	{B}	{A, C}
b	**a**	**a**	**b**	**a**

- Table for string ‘**w**’ that has length 5
- The algorithm populates the triangular table

Example (Result)

- Is baaba in L(G)?

Yes

We can see the S in the set X_{1n} where 'n' = 5

We can see the table

the cell X_{15} = (S, A, C) then

if S $\in$ X_{15} then baaba $\in$ L(G)

CYK Algorithm

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

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

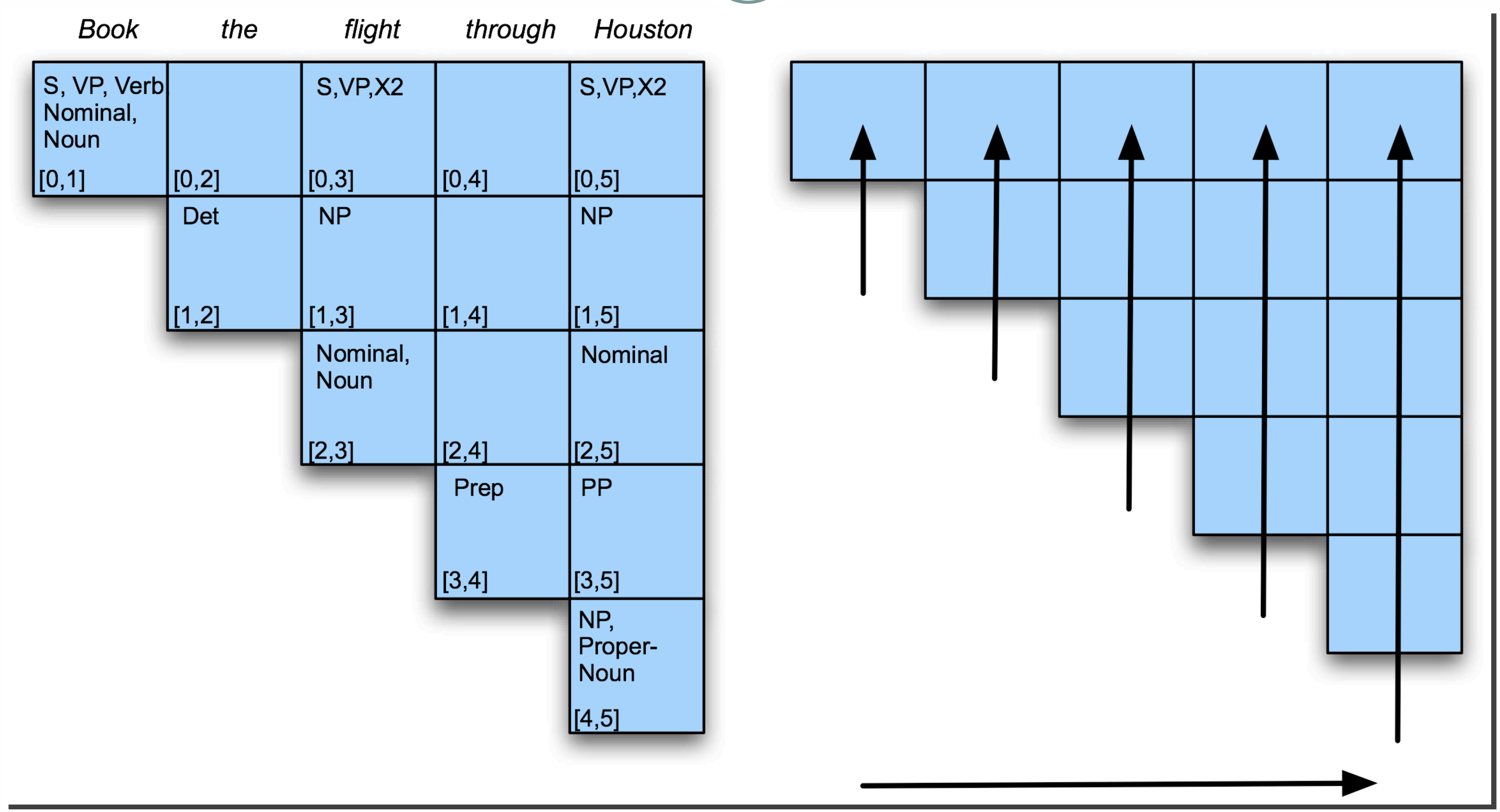
$\quad table[j-1, j] \leftarrow \{A \mid A \rightarrow words[j] \in grammar\}$

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

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

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

Example


Book
the
flight
through
Houston
S, VP, Verb Nominal, Noun
[0,1]
[0,2]
S,VP,X2
[0,3]
[0,4]
S,VP,X2
[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]

The Earley Algorithm

- Chart entries represent three type of constituents
  - predicted constituents (top-down predictions)
  - Scan in-progress constituents (we're in the midst of …)
  - completed constituents (we've found …)
- Progress in parse represented by Dotted Rules
  - Position of • indicates type of constituent
  - $_0$ Book $_1$ that $_2$ flight $_3$

    S --> • VP, [0,0] (predicting VP)

    NP --> Det • Nom, [1,2] (finding NP)

    VP --> V NP •, [0,3] (found VP)
  - [x,y] tells us where the state begins (x) and where the dot lies (y) wrt the input

The Earley Algorithm

$_0$ Book $_1$ that $_2$ flight $_3$

S --> • VP, [0,0]

- First 0 means S constituent begins at the start of the input
- Second 0 means the dot here too

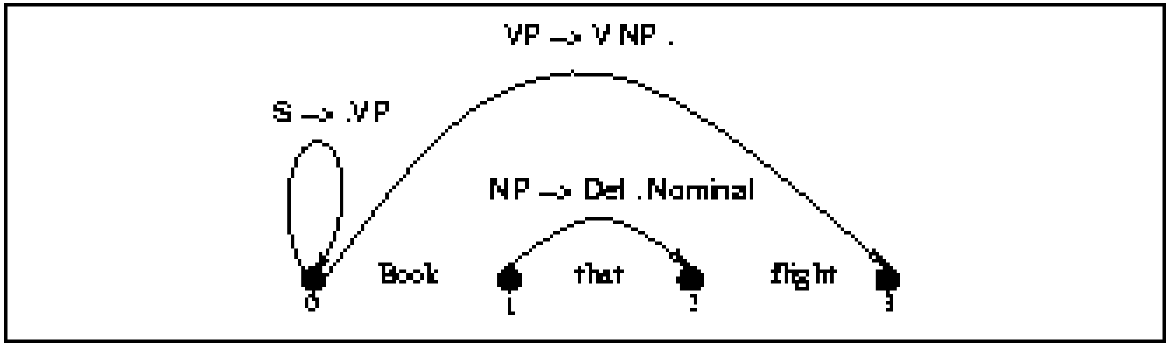
NP --> Det • Nom, [1,2]

- the NP begins at position 1
- the dot is at position 2
- Det has been successfully parsed
- Nom predicted next

The Earley Algorithm

VP --> V NP •, [0,3]

- Successful VP parse of entire input
- Graphical representation



Successful Parse

- Final answer is found by looking at last entry in chart
- If entry resembles S --> α • [0,N] then input parsed successfully
- But …
  - note that chart will also contain a record of all possible parses of input string, given the grammar
  - not just the successful one(s)

Parsing Procedure for the Earley Algorithm

- Move through each set of states in order, applying one of three operators to each state
  - predictor: add top-down predictions
  - scanner: read input and add corresponding state
  - completer: move dot to right when new constituent found
- No backtracking and no states removed: keep complete history of parse

Predictor

- New states represent top-down expectations
- Applied when non part-of-speech non-terminals are to the right of a dot

  S --> • VP [0,0]

- Adds new states to end of ***current*** chart
  - One new state for each expansion of the non-terminal in the grammar

    VP --> • V [0,0]

    VP --> • V NP [0,0]

Scanner

- New states for predicted part of speech.
- Applicable when part of speech is to the right of a dot

  VP --> • V NP [0,0]
- Looks at current word in input
- If match, adds state(s) to ***next*** chart

  VP --> V • NP [0,1]
  - i.e., we've ***found*** a piece of this constituent!

Completer

- We've found a constituent, so tell everyone waiting for this
- Applied when dot has reached right end of rule

  NP --> Det Nom • [1,3]
- Find all states w/dot at 1 and expecting an NP

  VP --> V • NP [0,1]
- Adds new (completed) state(s) to ***current*** chart

  VP --> V NP • [0,3]

CFG for Fragment of English

S → NP VP	Det → that | this | a
S → Aux NP VP	N → book | flight | meal | money
S → VP	V → book | include | prefer
NP → Det Nom	Aux → does
Nom → N	
Nom → N Nom	Prep → from | to | on
NP → PropN	PropN → Houston | TWA
VP → V	Nom → Nom PP
VP → V NP	PP → Prep NP

Book that flight (Chart [0])

- Seed chart with top-down predictions for S from grammar

$\gamma \rightarrow \bullet S$	[0,0]	Dummy start state
S → • NP VP	[0,0]	Predictor
S → • Aux NP VP	[0,0]	Predictor
S → • VP	[0,0]	Predictor
NP → • Det Nom	[0,0]	Predictor
NP → • PropN	[0,0]	Predictor
VP → • V	[0,0]	Predictor
VP → • V NP	[0,0]	Predictor

Parsing by Earley Algorithm

- When dummy start state is processed, it's passed to Predictor, which produces states representing every possible expansion of S, and adds these and every expansion of the left corners of these trees to bottom of Chart[0]

- When VP --> • V, [0,0] is reached, Scanner called, which consults first word of input, Book, and adds first state to Chart[1], VP --> Book •, [0,0]

Chart[1]

V→ book •	[0,1]	Scanner
VP → V •	[0,1]	Completer
VP → V • NP	[0,1]	Completer
S → VP •	[0,1]	Completer
NP → • Det Nom	[1,1]	Predictor
NP → • PropN	[1,1]	Predictor

V--> book • passed to Completer, which finds 2 states in Chart[0] whose left corner is V and adds them to Chart[1], moving dots to right

Parsing by Earley Algorithm

Chart[0]

γ	$\rightarrow$	$\bullet\, S$	[0,0]	Dummy start state
S	$\rightarrow$	$\bullet\, NP\ VP$	[0,0]	Predictor
NP	$\rightarrow$	$\bullet\, Det\ NOMINAL$	[0,0]	Predictor
NP	$\rightarrow$	$\bullet\, Proper\text{-}Noun$	[0,0]	Predictor
S	$\rightarrow$	$\bullet\, Aux\ NP\ VP$	[0,0]	Predictor
S	$\rightarrow$	$\bullet\, VP$	[0,0]	Predictor
VP	$\rightarrow$	$\bullet\, Verb$	[0,0]	Predictor
VP	$\rightarrow$	$\bullet\, Verb\ NP$	[0,0]	Predictor

Chart[1]

$Verb$	$\rightarrow$	$book\, \bullet$	[0,1]	Scanner
VP	$\rightarrow$	$Verb \bullet$	[0,1]	Completer
S	$\rightarrow$	$VP \bullet$	[0,1]	Completer
VP	$\rightarrow$	$Verb \bullet NP$	[0,1]	Completer
NP	$\rightarrow$	$\bullet\, Det\ NOMINAL$	[1,1]	Predictor
NP	$\rightarrow$	$\bullet\, Proper\text{-}Noun$	[1,1]	Predictor

Chart[2]

Det	$\rightarrow$	$that \bullet$	[1,2]	Scanner
NP	$\rightarrow$	$Det \bullet NOMINAL$	[1,2]	Completer
$NOMINAL$	$\rightarrow$	$\bullet\, Noun$	[2,2]	Predictor
$NOMINAL$	$\rightarrow$	$\bullet\, Noun\ NOMINAL$	[2,2]	Predictor

Chart[3]

$Noun$	$\rightarrow$	$flight \bullet$	[2,3]	Scanner
$NOMINAL$	$\rightarrow$	$Noun \bullet$	[2,3]	Completer
$NOMINAL$	$\rightarrow$	$Noun \bullet NOMINAL$	[2,3]	Completer
NP	$\rightarrow$	$Det\ NOMINAL \bullet$	[1,3]	Completer
VP	$\rightarrow$	$Verb\ NP \bullet$	[0,3]	Completer
S	$\rightarrow$	$VP \bullet$	[0,3]	Completer
$NOMINAL$	$\rightarrow$	$\bullet\, Noun$	[3,3]	Predictor
$NOMINAL$	$\rightarrow$	$\bullet\, Noun\ NOMINAL$	[3,3]	Predictor

The Earley Algorithm

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

  ENQUEUE((γ → • S, [0,0]), chart[0])
  for i ← 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)

 procedure PREDICTOR((A → α • B β, [i, j]))
    for each (B → γ) in GRAMMAR-RULES-FOR(B, grammar) do
        ENQUEUE((B → • γ, [j, j]), chart[j])
    end

 procedure SCANNER((A → α • B β, [i, j]))
    if B ⊂ PARTS-OF-SPEECH(word[j]) then
       ENQUEUE((B → word[j], [j, j+1]), chart[j+1])

 procedure COMPLETER((B → γ •, [j,k]))
    for each (A → α • B β, [i, j]) in chart[j] do
        ENQUEUE((A → α B • β, [i,k]), chart[k])
    end

 procedure ENQUEUE(state, chart-entry)
    if state is not already in chart-entry then
       PUSH(state, chart-entry)
    end
```

THANK YOU