CS395-T
Topics in Natural Language Processing

LECTURE 4

Sanda Harabagiu
Department of Computer Sciences
University of Texas at Austin
Tu, Th 11AM-12:20 PM
sanda@cs.utexas.edu
http://www.cs.utexas.edu/users/sanda/cs395T.html
http://www.engr.smu.edu/~sanda/cs395T.html

1

Using a depth-First Strategy For Parsing

- Depth-first search expands the search space incrementally by systematically exploring one state at a time
- Which state is chosen for expansion ?
  - The most recently generated one
- What happens when this search arrives at a tree that is not consistent with the input ?
  - The search continues by returning to the most recently generated, yet unexplored tree

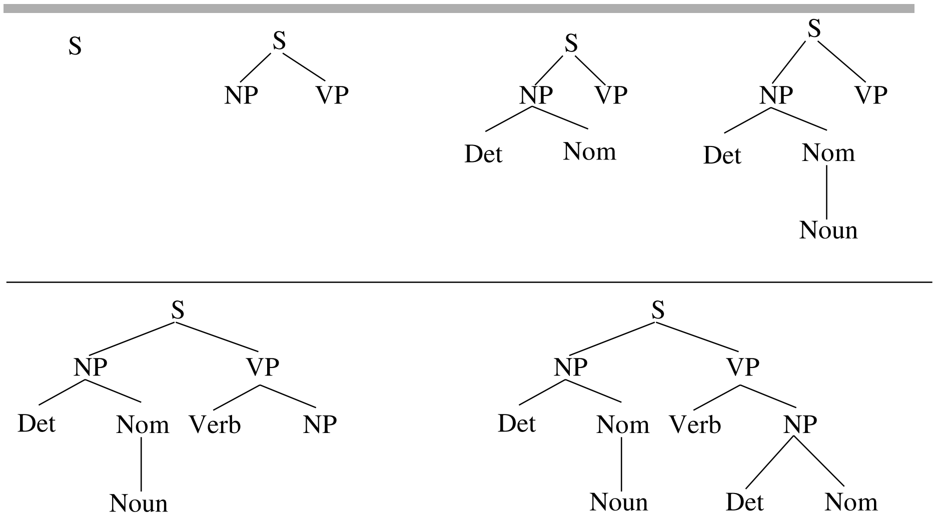
2

Example Grammar

S → NP VP	Det → that | this | a
S → Aux NP VP	Noun → book | flight | meal | money
S → VP	Verb → book | include | prefer
NP → Det Nominal	Aux → does
Nominal → Noun	
Nominal → Noun Nominal	Prep → from | to | on
NP → Proper-Noun	Proper-Noun → Houston | TWA
VP → Verb	
VP → Verb NP	Nominal → Nominal PP

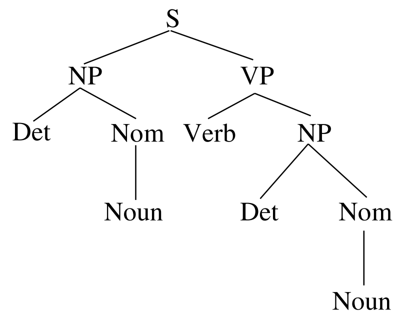
3

A Top-Down Depth-First Derivation



4

A Top-Down Depth-First Derivation



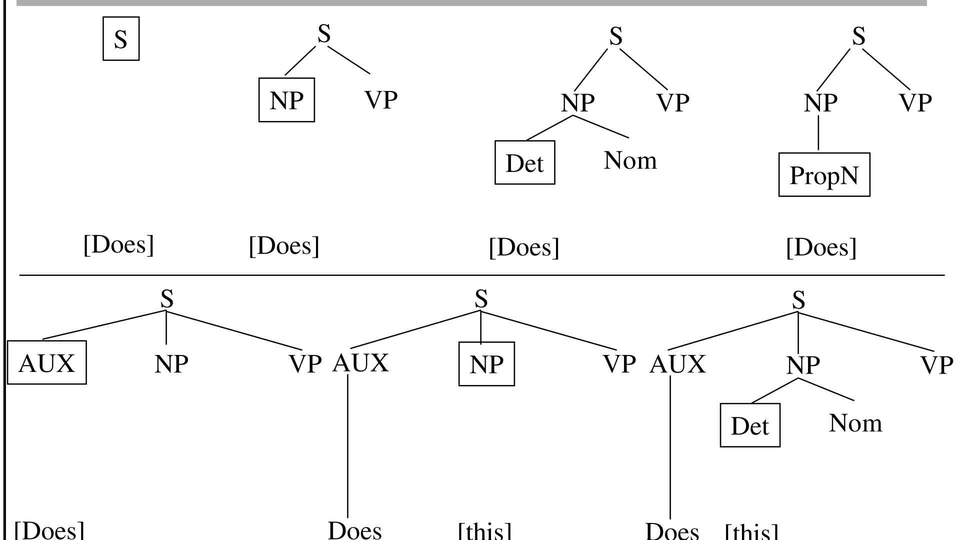
Parsing With an Agenda

- Search states = (partial tree, pointer to the next word in the sentence)
- Agenda = list of search states

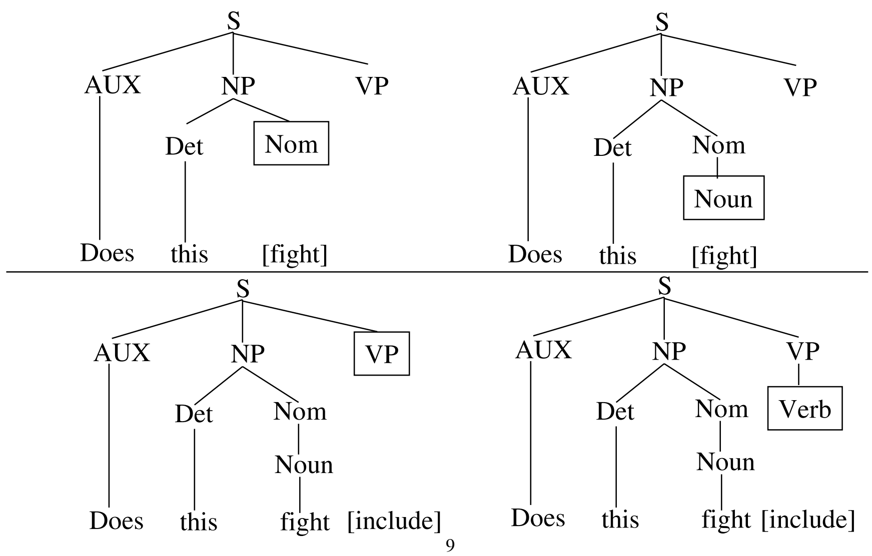
A Top-Down, Depth-First Left-to-Right Parser

function TOP-DOWN-PARSE(input, grammar) **returns** a parse tree
agenda ← *(Initial S tree, Beginning of input)*
Current-search-state ← POP(*agenda*)
loop
  if SUCCESSFUL-PARSE?(*current-search-state*) **then**
    return TREE(*current-search-state*)
  else
    if CAT(NODE-TO-EXPAND(*current-search-state*)) is a POS **then**
      if CAT(node-to-expand) ⊂ POS(CURRENT-INPUT(*current-search-state*)) **then**
        PUSH(APPLY-LEXICAL-RULE(*current-search-state*),*agent*)
      else return reject
    else PUSH(APPLY-RULES(*current-search-state,grammar*),*agenda*)
  if agenda is empty **then return** reject
  else *current-search-state* ← NEXT(*agenda*)
end

A Top-Down Depth-First Derivation

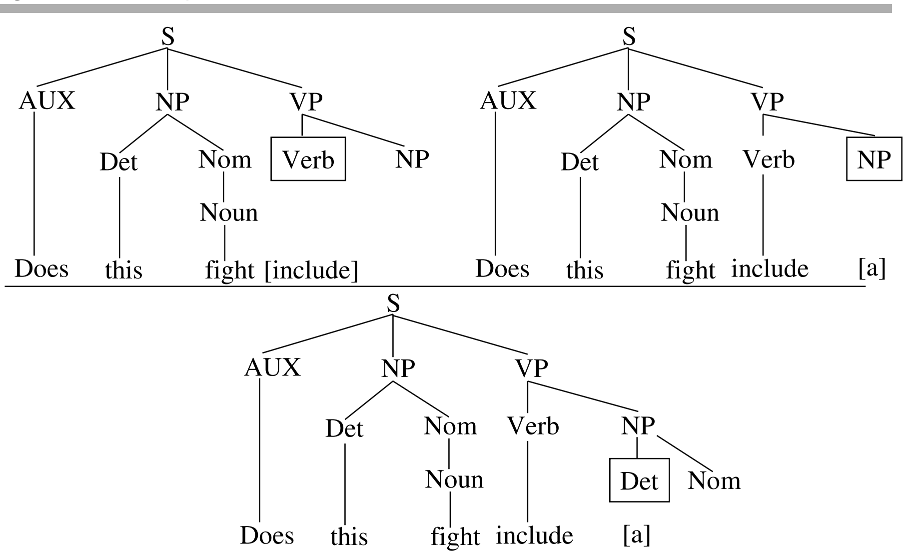


Continue Derivation



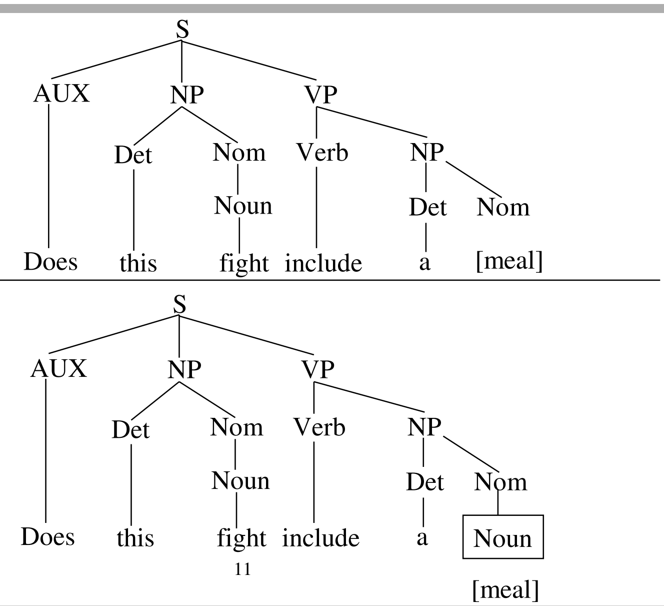
9

A Top-Down, Depth-First Left-to-Right Parser CONTINUED



10

A Top-Down, Depth-First Left-to-Right Parser CONTINUED



Adding Bottom-up Filtering

Observations:

1) The parser expands non-terminal symbols along the edge of the parse tree, down to the word at the bottom left edge of the tree.
2) When the word is incorporated in a tree, the input pointer moves on, the parser will expand the next left-most non-terminal symbol down to the new left-corner word.

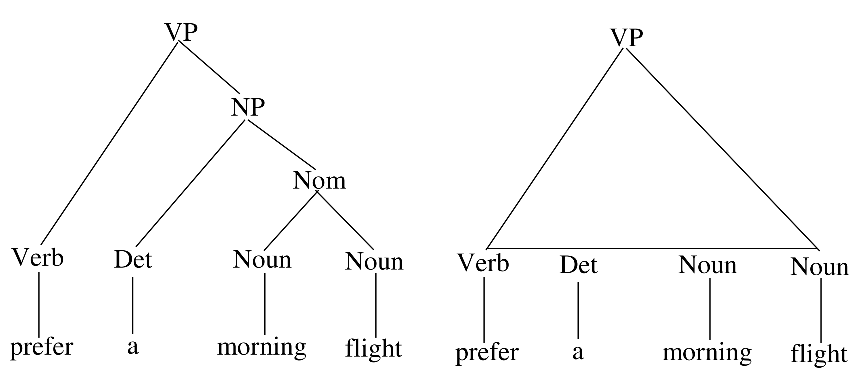
Conclusion: the input word serves as the first word in the derivation of the unexpanded node that the parser is currently processing.

Lesson learned

- The parser should not consider any grammar rule if the current input cannot serve as the first word along the left edge of some derivation from this rule.
- Left-corner of the tree: - first word + left edge of the derivation

13

Illustration of the left-corner notion



14

Formal Definition

- B is a left corner of A if:

$$A \overset{*}{\Rightarrow} B\alpha$$

- **Example**: *Does this flight include a meal?*

S → NP VP

S → Aux NP VP → the only rule for which <u>*does*</u> may serve as a left-corner

S → VP

15

Practical Information

- Compile all information needed to efficiently implement such a filter in a table that lists all the valid left-corner categories for each non-terminal of the grammar.

- **Example:**

Category	Left Corners
S	Det, Proper-Noun, Aux, Verb
NP	Det, Proper-Noun
Nominal	Noun
VP	Verb

16

3 More problems with the parser

- Left Recursive
- Ambiguity
- Inefficient reparsing of subtrees

17

Left recursion

Problem of depth-first search

- if infinite search space ➡ it may dive down an infinitely deeper path and never return to visit unexpanded states.

The problem exists when

- *left recursive grammars* are used

18

Left-recursive grammars

Formally: a grammar is left-recursive if it contains a non-terminal category that has a derivation that contains itself anywhere along its leftmost branch.

$$A \overset{*}{\Rightarrow} \alpha A \beta$$

and

$$\alpha \overset{*}{\Rightarrow} \varepsilon$$

Example:

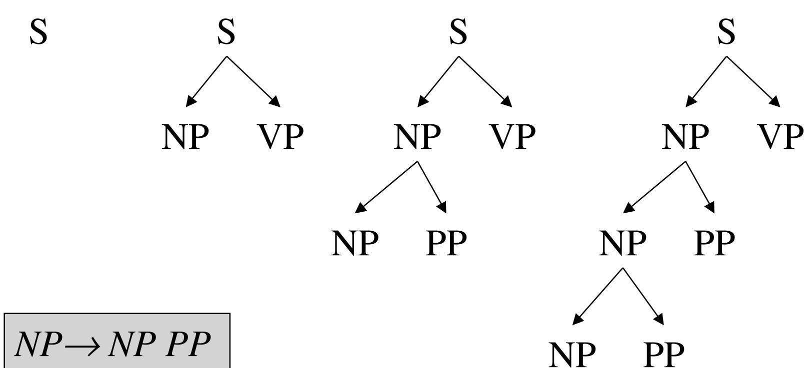
- Possessives: *(Atlanta's airport)*

$$\left\{\begin{array}{l} NP \rightarrow Det\ Nominal \\ Det \rightarrow NP\text{'s} \end{array}\right.$$

rules that induce left-recursion

19

More obvious case of left-recursion



NP→ NP PP
VP→ VP PP
S→ S and S

20

2 solutions

1) Rewrite the grammar

2) Explicitly managing the depth of the search during parsing

21

How should we rewrite rules ?

Example :

$A \rightarrow A\ \beta \mid \alpha$

Rewrite :

$\left.\begin{array}{l} A \rightarrow \alpha\ A' \\ A' \rightarrow \beta\ A' \mid \varepsilon \end{array}\right\} \Rightarrow$ changes the left recursion to right recursion.

22

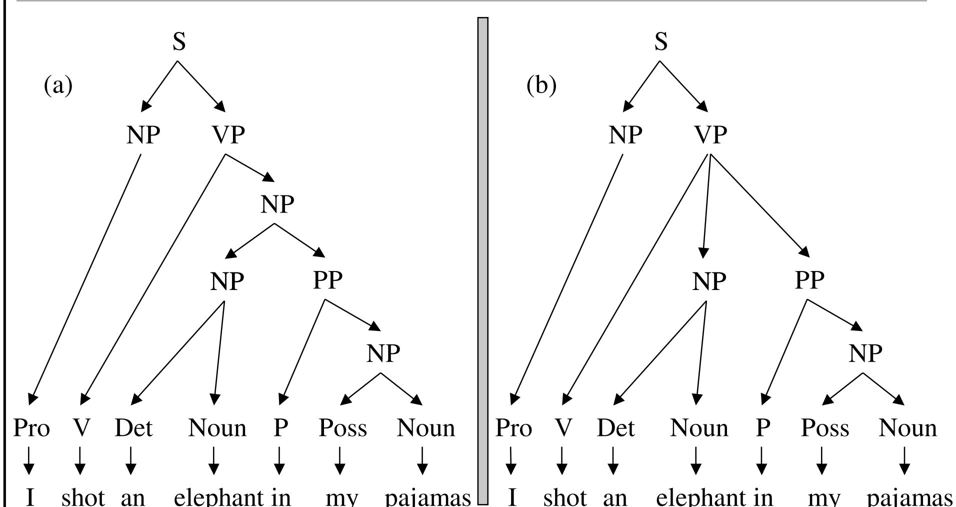
Ambiguity

- Lexical ambiguity ⇒ POS taggers
- Structural ambiguity ⇒ Parsing problems

3 forms
- attachment ambiguity
- coordination ambiguity
- noun-phrase bracketing ambiguity

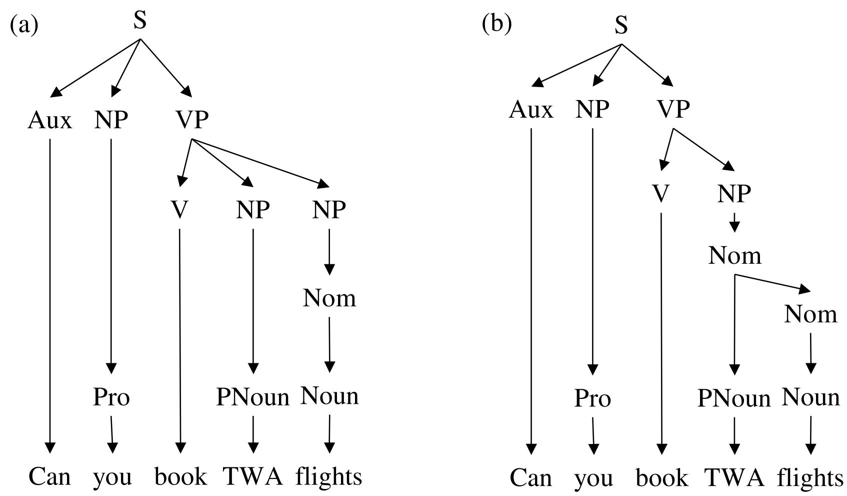
23

Attachment ambiguity



24

Other examples



Coordination ambiguity

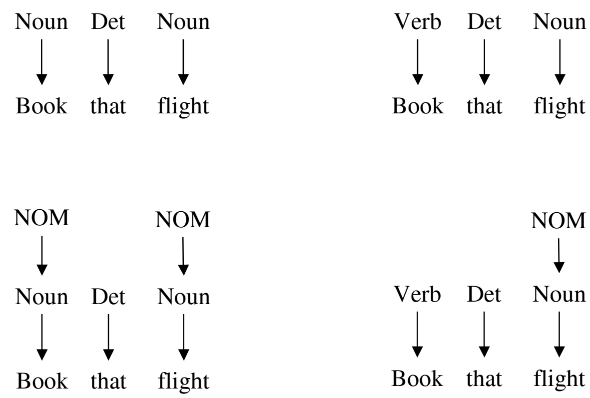
Old men and women

[Old [man and women]]

[Old man] and [women]

Combination of ambiguity

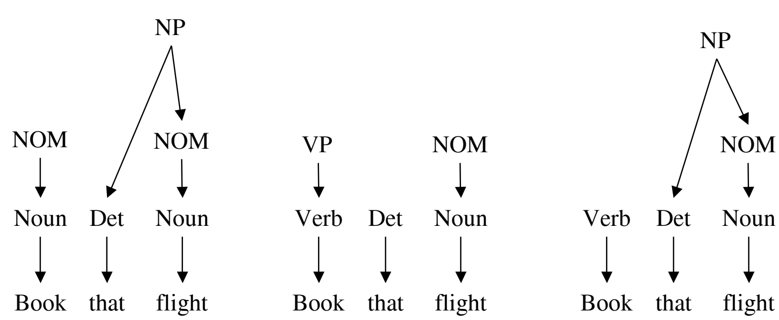
Book that flight



27

Combination of ambiguity (cont'd)

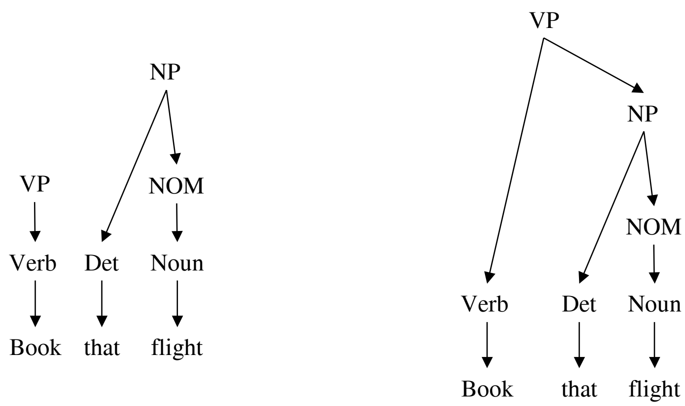
Book that flight



28

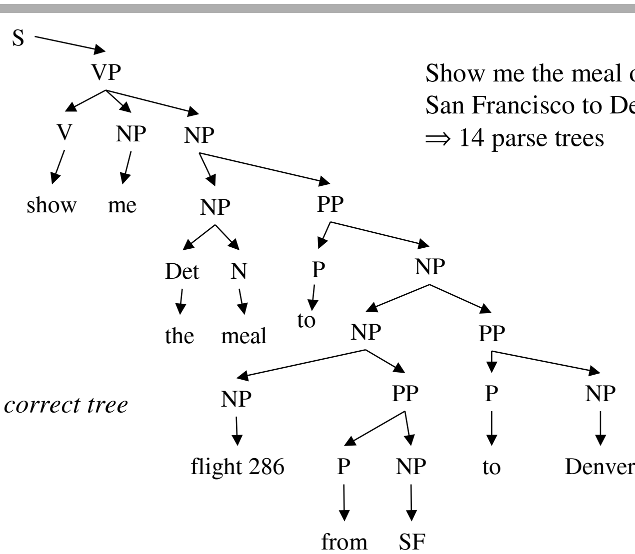
Combination of ambiguity (cont'd)

Book that flight



29

Exponential number of parses



Show me the meal on Flight 286 from San Francisco to Denver

$\Rightarrow$ 14 parse trees

The correct tree

30

Exponential growth

$$C(n) = \frac{1}{n+1}\binom{2n}{n}$$ $\rightarrow$ the catalan numbers

Number of PPs	Number of NP Parses
2	2
3	5
4	14
5	132
6	469
7	1430
8	4867

31

Repeated parses of subtrees

The parser often builds valid trees for portions of the input, then discards them during bracketing.

$\Rightarrow$ later it finds that it has to rebuild them again.

Example:

the process involved in finding the parse for the NP:

"a flight from Indianapolis to Houston on TWA"

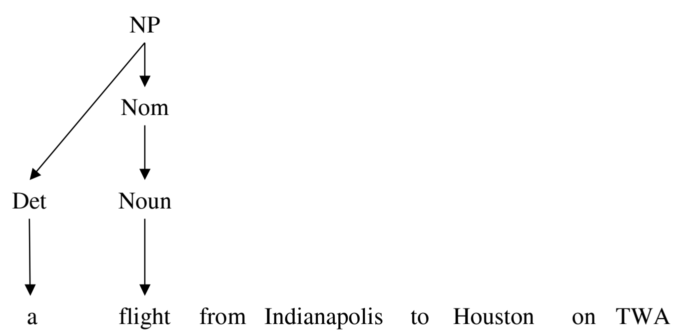
32

Repeated parses of subtrees (cont'd)

a flight	4
from Indianapolis	3
to Houston	2
on TWA	1
a flight form Indianapolis	3
a flight form Indianapolis to Houston	2
a flight form Indianapolis to Houston on TWA	1

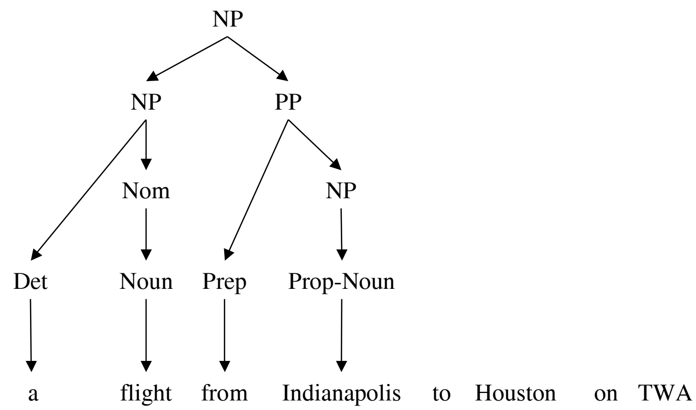
33

Repeated parses of subtrees (cont'd)



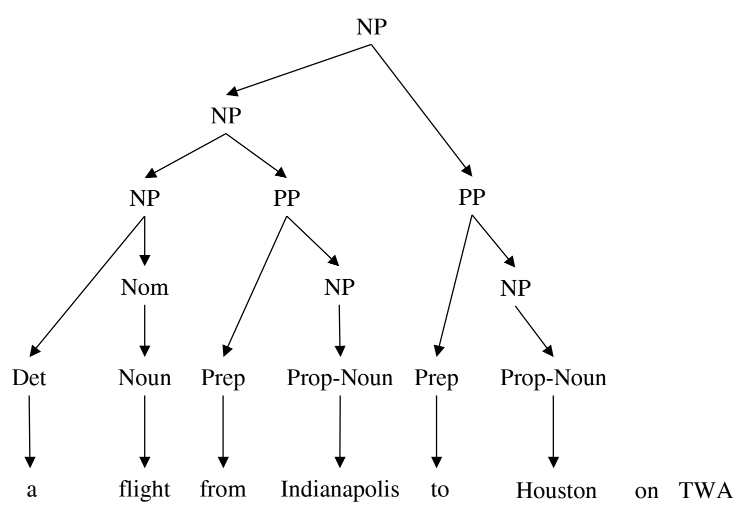
34

Repeated parses of subtrees (cont'd)



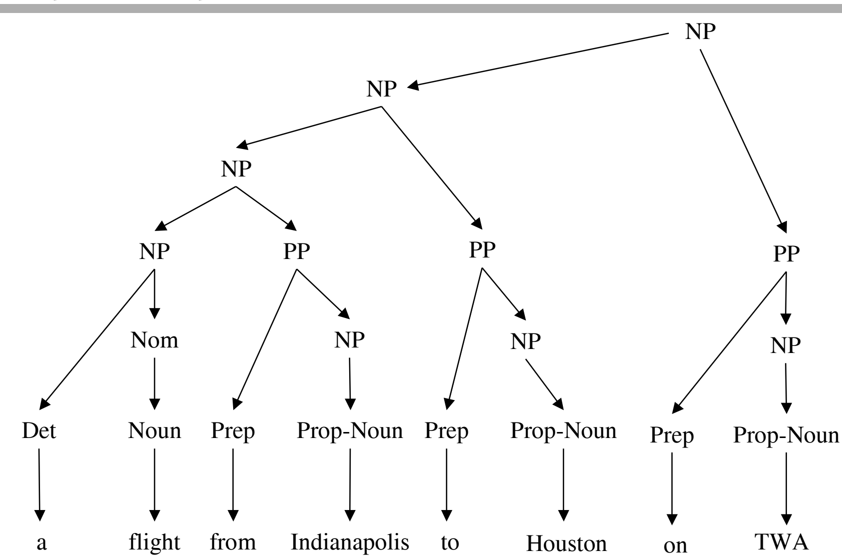
35

Repeated parses of subtrees (cont'd)



36

Repeated parses of subtrees (cont'd)



37

Chart-Parsing

- Chart ➔ an array that has N+1 entries (the sentence has N words !)
- For each word, the chart contains a list of states representing the partial parse trees that have been generated so far
- **Advantage**: Each possible sub-tree is represented only once, and thus can be shared by all the parses that need it

38

The Representation of A State

- 3 kinds of information:
  1) A sub-tree corresponding to a single grammar rule
  2) Information about the **progress** made in completing this sub-tree
  3) The position of the sub-tree with respect to the input

39

How Do We Represent Progress ?

- Graphically: use a dot within the right-hand side of a state's grammar rule

  (dotted rule)

- A state position with respect to the input ?
  - 2 numbers [where the state begins]
    [where the dot lies]

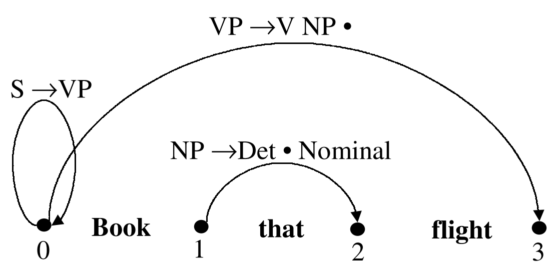
40

Example

- Sentence: *Book that flight*.
  - State 0   S → • VP, [0,0]
  - State 1   NP → Det • Nominal, [1,2]
  - State 2   VP → V NP • , [0,3]

41

Another Representation



42

The Early Parser

- Scan through the N+1 sets of states in the chart in a left → right fashion
  - At each step one of three operators is applied, depending on the state's status
    - Results → addition of new states to the end of either the current or the next set of states in the chart
- Successful parse: the presence of a state

$$S \rightarrow \alpha\bullet, [0,N]$$

in the list of the last chart entry

43

What Oprators?

•Predictor	???
•Scanner	???
•Completer	???

- The **Predictor** creates new states representing top-down expectations.
  - applied to any state that has a non-terminal to the right of the dot that is not a PoS category.
- What happenes? ⟹ create a new state for each alternative expansion of that non-terminal provided by the grammar.

44

Predictor Example

- The state

  S → • VP, [0,0]
- results in states

  VP → • Verb, [0,0]
- and

  VP → • Verb NP, [0,0]

45

Scanner

- If a state has a PoS category at the right of the dot!
- The scanner is called to examine the input and incorporate a state corresponding to the predicted PoS in the chart.

Example:

- State 0 VP → • Verb NP, [0,0]

  Book that flight

  verb
- State 1 VP → Verb • NP, [0,1]

46

Completer

- Applied to a state if the dot has reached the right end of the rule ⟹ Meaning that the parser has successfully discovered a particular grammatical category over some span of the input.
- **Example:** NP → Det Nominal •, [1,3]

  (looks for states ending at 1, expecting a NP)

➢ finds the state VP → Verb • NP, [0,1]

(created by the scanner)

➢ addition of a new complete state

VP → Verb NP •, [0,3]

47

The Early-Parse

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*)

48

The Early-Parse

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

procedure Enqueue(*state, chart-entry*)
    if *state* is not already in *chart-entry* **then**
        Push(*state, chart-entry*)
    end

49

Chart Content

Chart[0]

$\gamma \rightarrow \bullet\ S$	[0.0]	Dummy start stare
$S \rightarrow \bullet\ NP\ VP$	[0.0]	Predictor
$NP \rightarrow \bullet\ Det\ NOMINAL$	[0.0]	Predictor
$NP \rightarrow \bullet$ *Proper-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

50

Chart Content

Chart[1]

Verb → book •	[0,1]	Scanner
VP → Verb •	[0,1]	Completer
S → VP •	[0,1]	Completer
VP → Verb • NP	[0,1]	Completer
NP → • Det NOMINAL	[1,1]	Predictor
NP → • Proper-Noun	[1,1]	Predictor

Chart[2]

Det → that •	[1,2]	Scanner
NP → Det • Nominal	[1,2]	Completer
NOMINAL → • Noun	[2,2]	Predictor
NOMINAL → • Noun Nominal	[2,2]	Predictor

51

Chart Content

Chart[3]

Noun → flight •	[2.3]	Scanner
NOMINAL → Noun •	[2.3]	Completer
NOMINAL → Noun • NOMINAL	[2.3]	Completer
NP → Det NOMINAL •	[1.3]	Completer
VP → Verb NP •	[0.3]	Completer
S → VP•	[0.3]	Completer
NOMINAL → • Noun	[3.3]	Predictor
NOMINAL → • Noun NOMINAL	[3.3]	Predictor

52

[redacted]