Natural Language Processing

Lecture 11

Today's Outline

Homework 2

Review CFGs

Handouts and Answers to CFG Questions

Parsing with CFGs

- top down and bottom up
- search
- introduction of dynamic programming

Homework 2

Review

Context Free Grammars

- sets of terminals (either lexical items or parts of speech)
- sets of non-terminals (the constituents of the language)
- sets of rules of the form A $\rightarrow$ α, where A is a non-terminal and α is a string of zero or more terminals and non-terminals

Examples

Constituents

- S $\rightarrow$ NP VP
- NP $\rightarrow$ Det Noun
- VP $\rightarrow$ V
- ...

Parts of Speech

- Det $\rightarrow$ a
- Noun $\rightarrow$ flight
- V $\rightarrow$ left
- ...

Parsing

Parsing

- recognizing and assigning structure (previous chapters)

Syntactic parsing

- recognizing a grammatical sentence and assigning a syntactic structure

Parsing with a CFG

- assigning a correct tree (or derivation) to a sentence given some grammar
- Chapter 9: declarative CFG formalism
- Chapter 10: algorithms for using the CFG formalism to compute parse trees for a sentence

CFG Parsing

Parsing with a CFG means assigning a correct tree (or trees) to a sentence given some CFG grammar.

The leaves of the tree cover all and only the input and the tree corresponds to a valid derivation according to the grammar.

Note that *correct* here means that the tree is consistent with the input and the grammar. It doesn't mean that it's the right tree or the proper way to represent English in any more global sense of correct.

Who Cares?

Grammar Checking

Semantic Analysis

Question-Answering / Information Extraction

Speech Recognition Language Models

Parsing as Search

As with finite-state recognition and transduction, parsing can be viewed as a search. The search space corresponds to the space of trees generated by the grammar. The search is guided by the structure of the space and by the input.

- regular expression parsing
  - search space of all possible paths through the FSA
  - search space defined by the FSA
- CFG syntactic parsing
  - search space of all possible parse trees
  - search space defined by the CFG

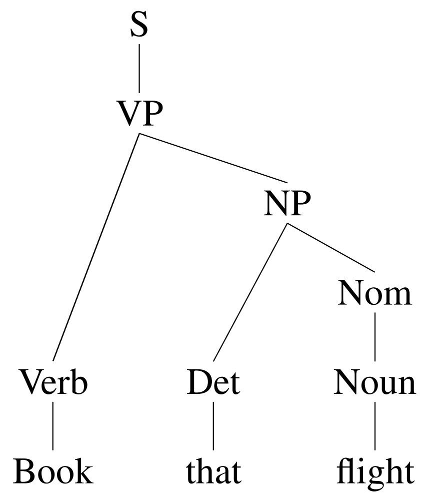
We'll start with the basic methods of parsing, see what's wrong with them, and then move on to a better method.

Parsing

A mini grammar and lexicon

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*

The parse of the sentence *Book that flight* according to the mini grammar



Parsing (continued)

What kind of constraints can be used to connect the grammar and the sentence when searching for the parse tree?

- top-down (goal-directed) strategy
  - tree should have one root (grammar constraint)
- bottom up (data-driven) strategy
  - tree should have three leaves (input sentence constraint)

A Note on the Input

For right now, we'll assume the following:

- the input is not tagged
- this input consists of unanalyzed word tokens
- all the words in the input are known
- all the word in the input are available simultaneously (i.e., they're buffered)

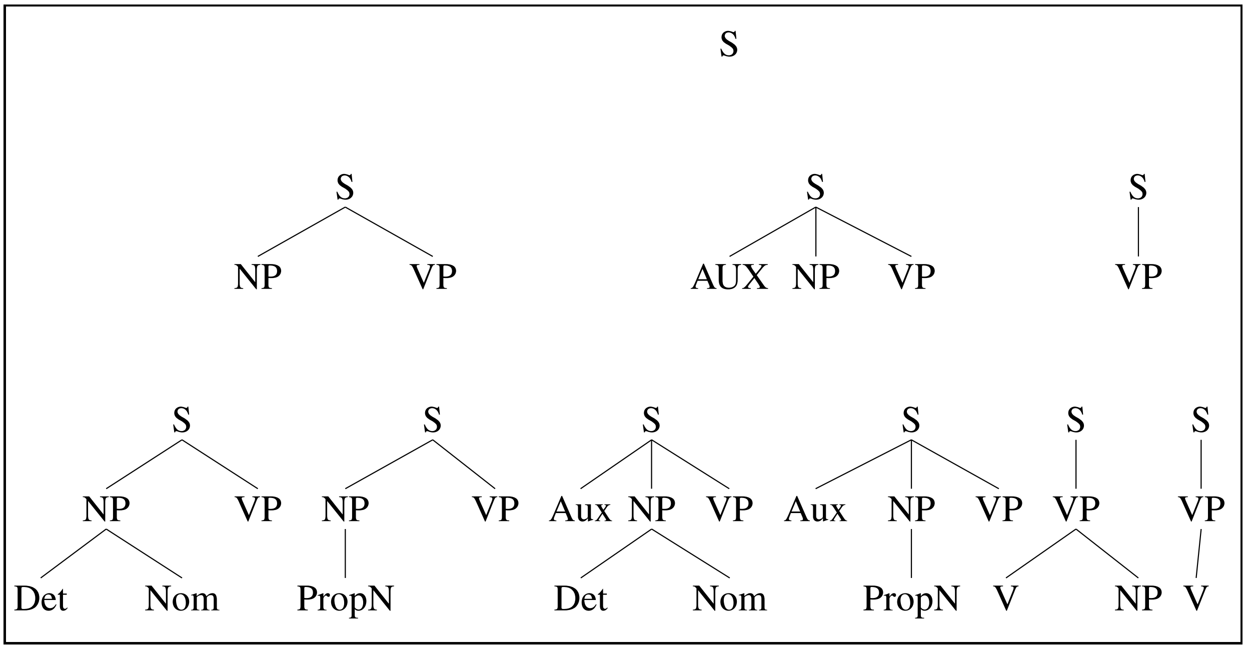
Top-Down Parsing

When the search is primarily goal or expectation-drive (by the structure of the grammar), then we're doing some kind of top-down search.

The primary goal is that we can start out by trying to find a tree rooted as S, since we're trying to parse sentences.

Trees are then built from the root node S to the leaves.

A Partial Top-Down Search Space



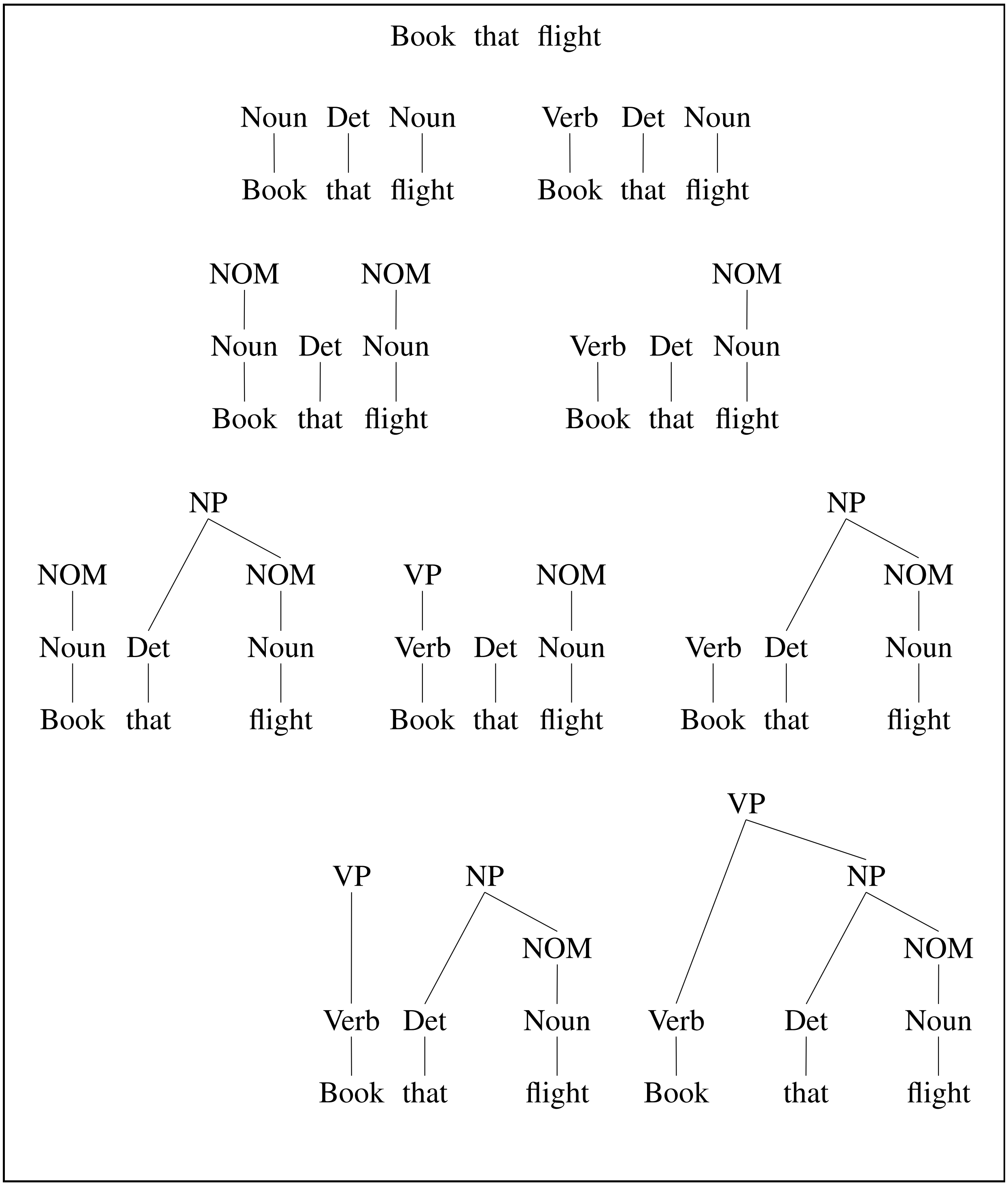
- assuming parallelism

Bottom-Up Parsing

When the search is primarily data-driven (by the input words), then we're doing some kind of bottom-up search.

The primary early consideration here is that the lowest subtrees of the final tree must hook up with the start symbol.

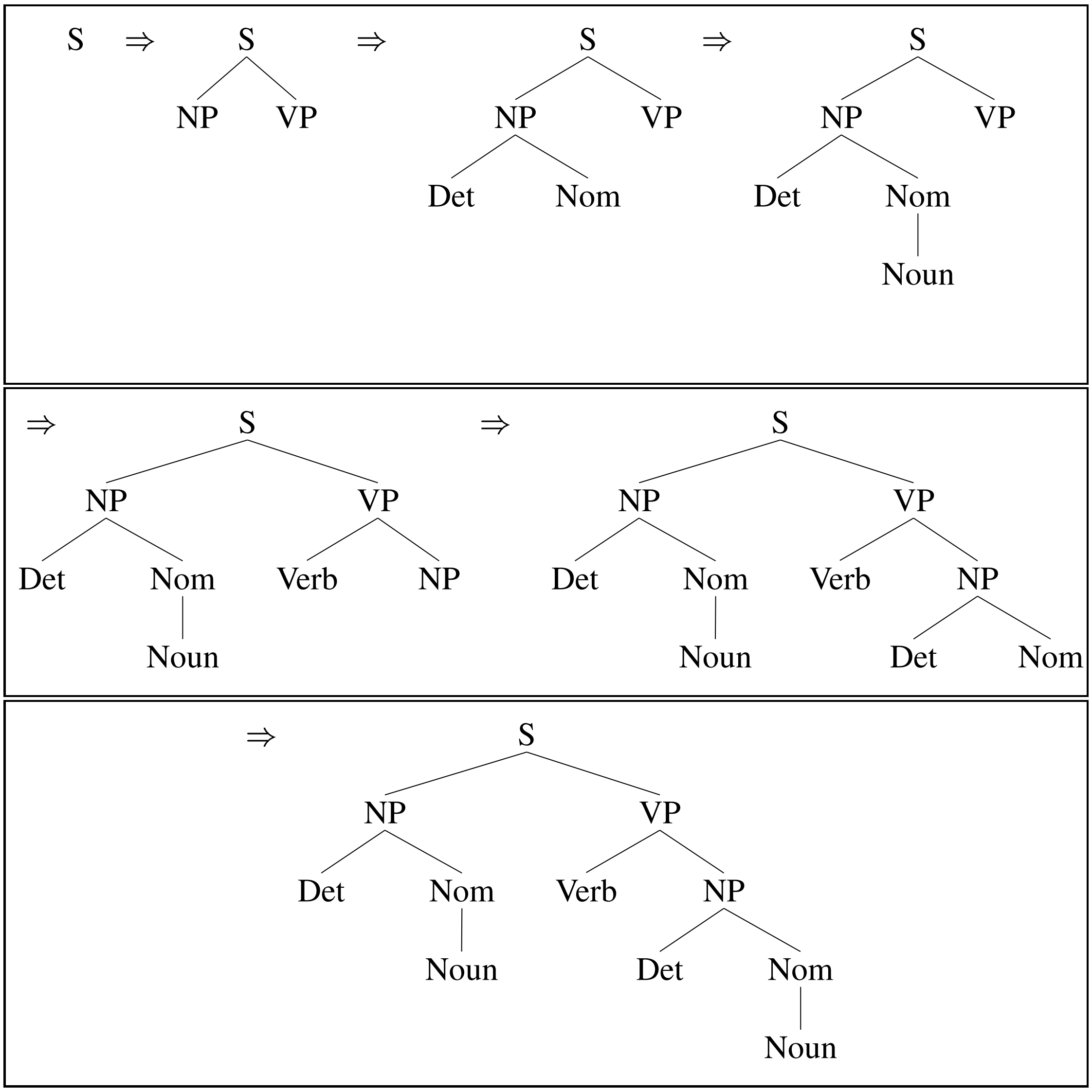
A Partial Bottom-Up Space



Search Control Issues

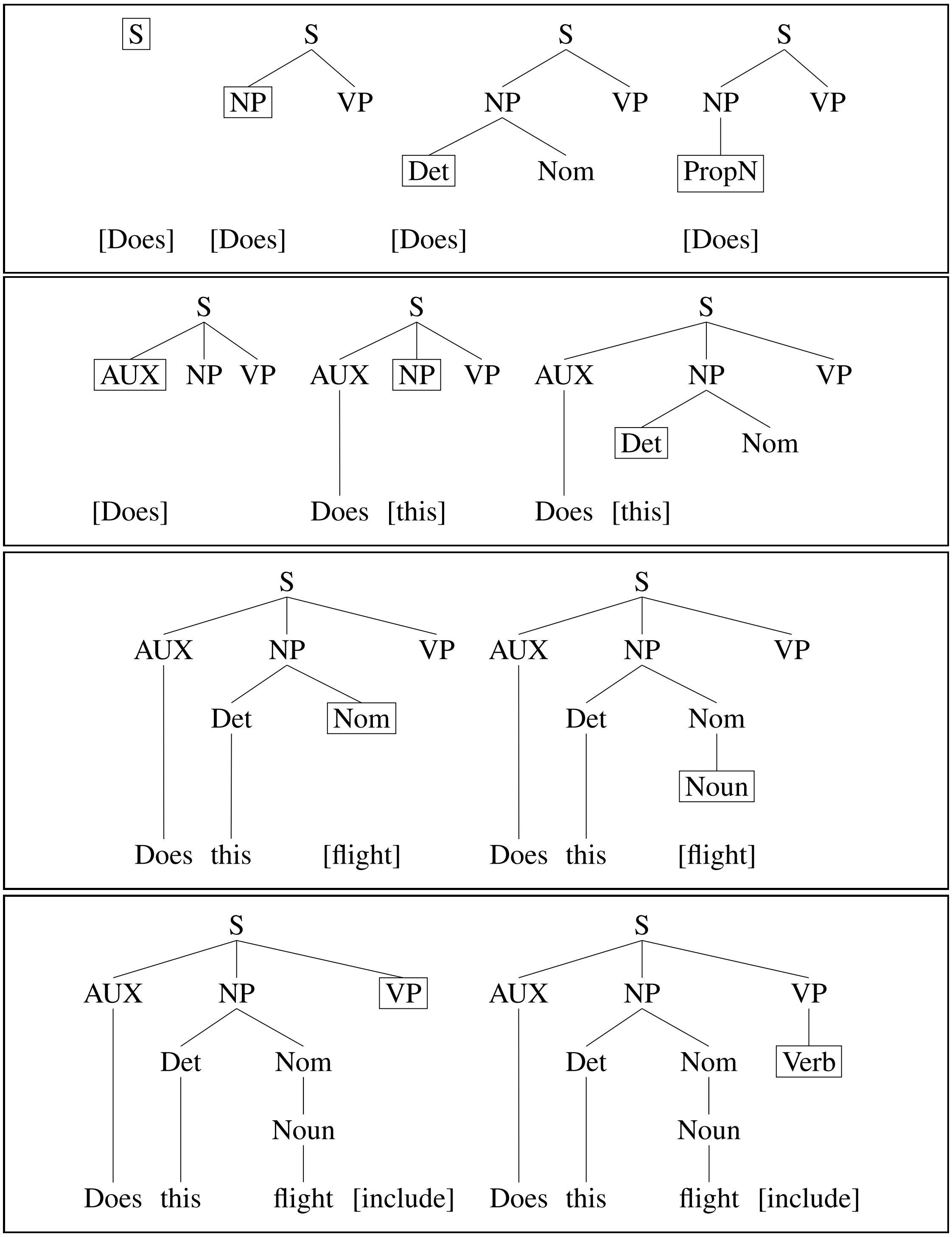
This discussion left out a few issues. Such as . . .

Search Control Issues (cont.)

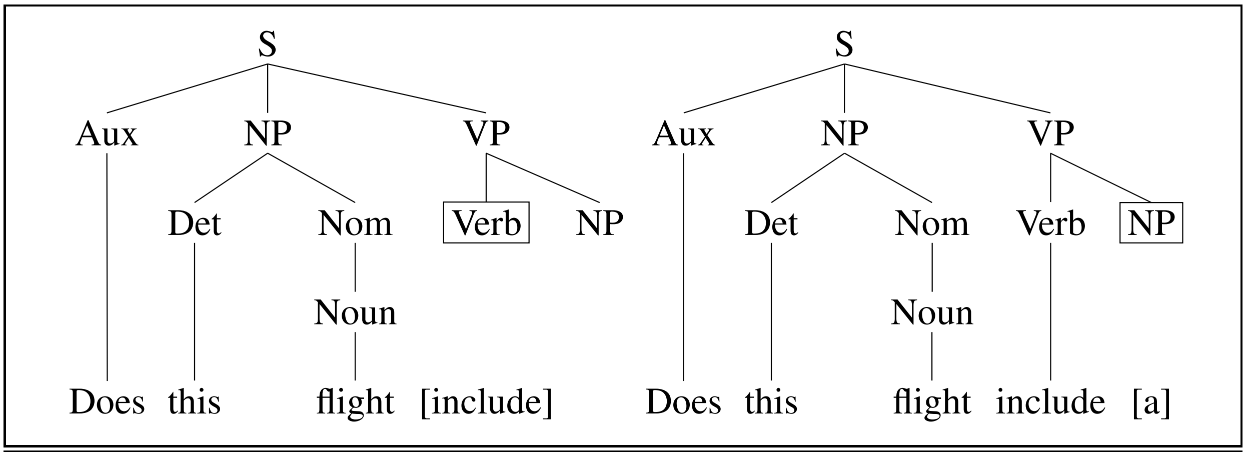


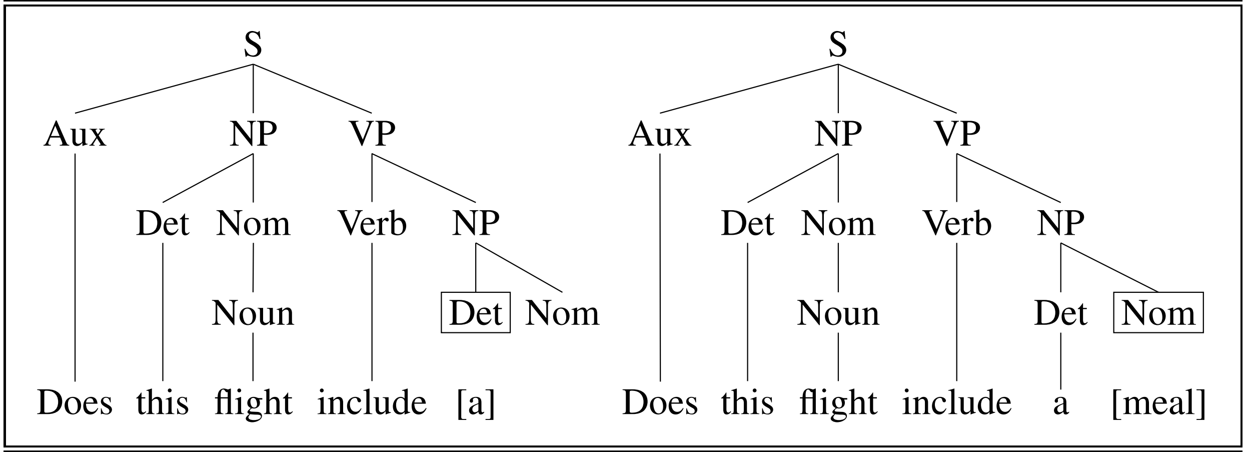
- non-parallel strategies (e.g., depth-first)
- which leaf node to expand next (e.g., leftmost)
- which of the applicable grammar rules to try (e.g., order in the grammar)

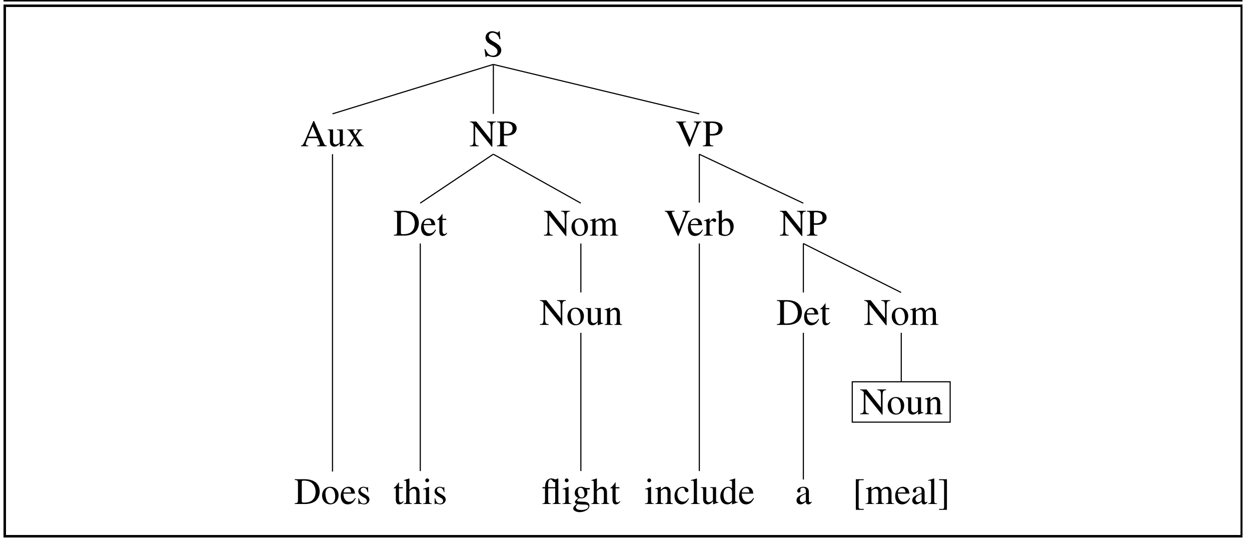
Top-Down Depth-First Left-to-Right



The Search Continued







T-D Depth-First L-R 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), agenda)
        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
```

Compare with ND-Recognize!

```
function ND-RECOGNIZE(tape, machine) returns accept or reject

  agenda ← {(Initial state of machine, beginning of tape)}
  current-search-state ← NEXT(agenda)
  loop
    if ACCEPT-STATE?(current-search-state) returns true then
       return accept
    else
       agenda ← agenda ∪ GENERATE-NEW-STATES(current-search-state)
    if agenda is empty then
       return reject
    else
       current-search-state ← NEXT(agenda)
  end

function GENERATE-NEW-STATES(current-state) returns a set of search-states

  current-node ← the node the current search-state is in
  index ← the point on the tape the current search-state is looking at
  return a list of search states from transition table as follows:
    (transition-table[current-node,ε], index)
    ∪
    (transition-table[current-node, tape[index]], index + 1)

function ACCEPT-STATE?(search-state) returns true or false

  current-node ← the node search-state is in
  index ← the point on the tape search-state is looking at
  if index is at the end of the tape and current-node is an accept state of machine
then
    return true
  else
    return false
```

Top-Down vs. Bottom-Up

There are advantages and disadvantages to both.

Top-Down

- only searches in the space of reasonable answers
- suggests hypotheses that are not consistent with the data
- has problems with left-recursion (infinite spaces)

Bottom-Up

- only forms hypotheses consistent with the data
- suggests hypotheses that make no sense globally
- also has problems with infinite spaces

A Hybrid Approach

Neither top-down nor bottom-up adequately exploit all the constraints.

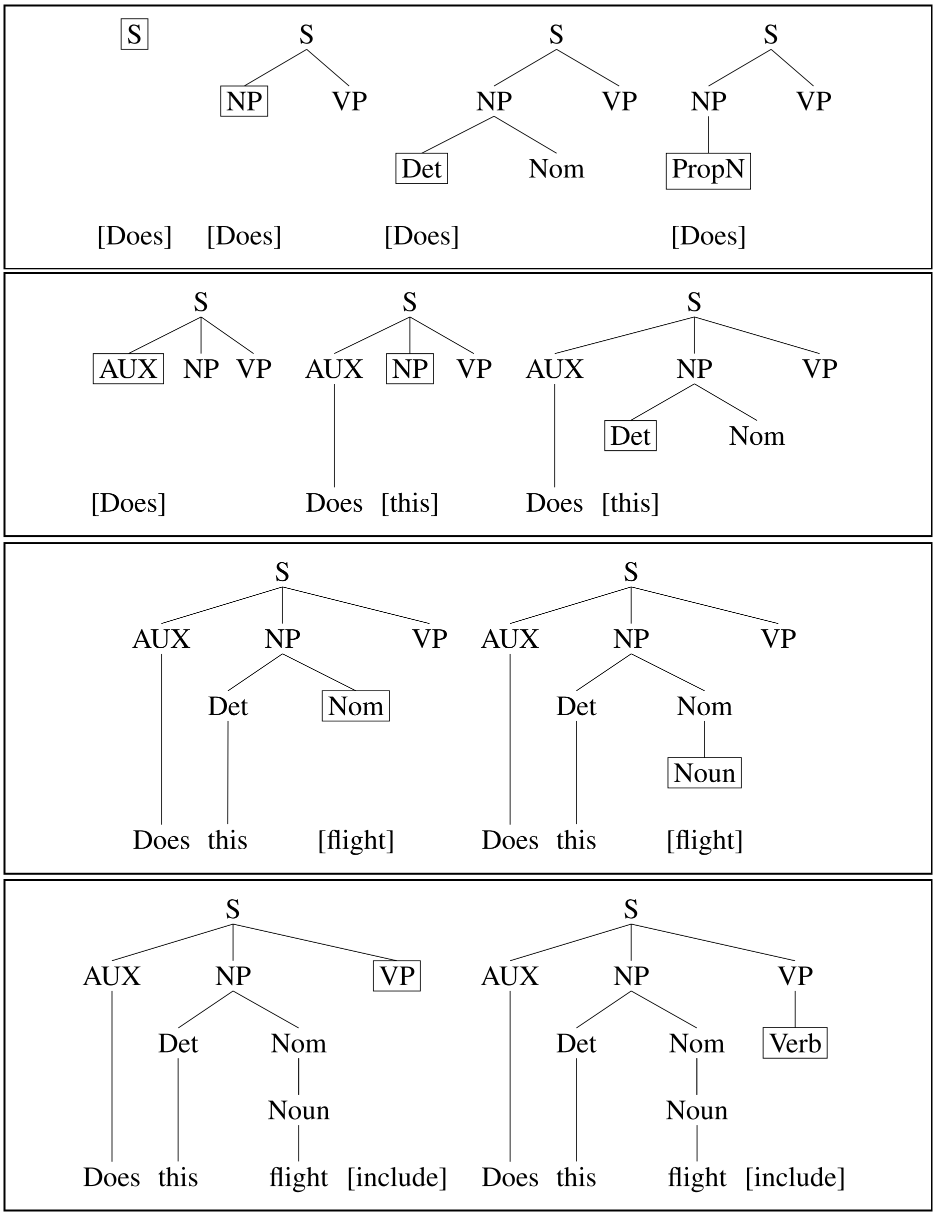
There are many way to combine top-down expectations with bottom-up data to get a more efficient search.

The most popular methods use one method as the basic search control strategy to generate trees.

They then use constraints from the other method to dynamically filter out "bad" structures.

We'll explore top-down parsing with bottom-up filtering.

Adding Bottom-Up Filtering



Left-Corner Observation: in a successful parse, the current input word is first in the derivation of the unexpanded node

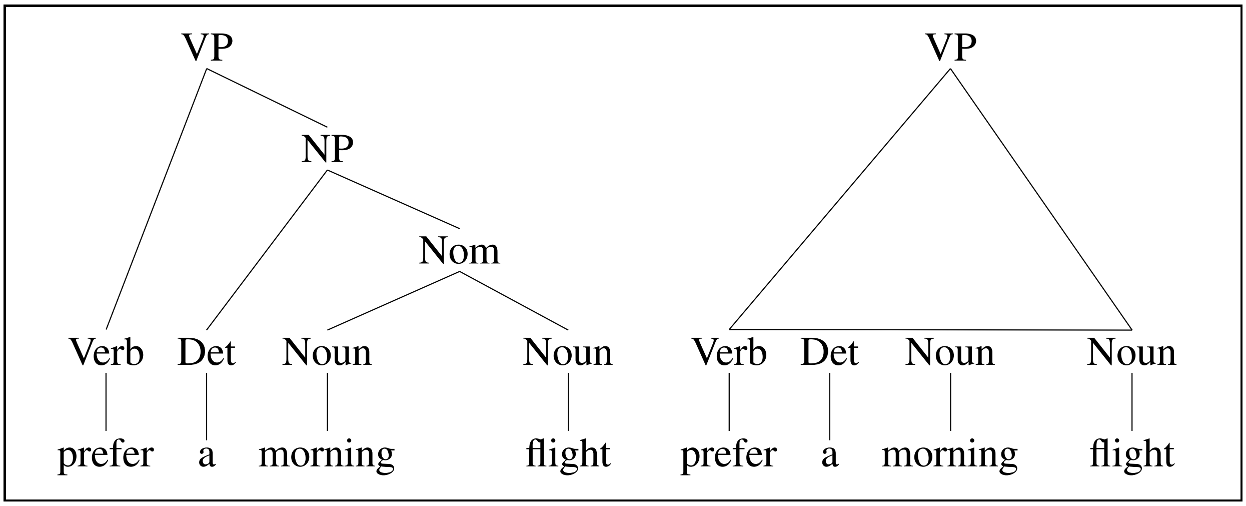
Bottom-Up Left-Corner Filtering

Don't consider any expansion where the current input cannot serve as the left-corner of that expansion.

Left-Corner of a tree:

- the first word along the left edge of a derivation
- B is a left corner of A if A derives Bα

Left-Corners



Verb and *prefer* are both left-corners of VP

Example

Consider a top-down parser parsing the following input:

- *Does this flight include a meal?*

Recall that the grammar contains the following rules:

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*

Using the left corner filter:

- only the second S rule is viable
- parser with filtering avoids previous backtracking

Knowledge for Left-Corner Filtering

Given our mini 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*

Here are the left corners for each non-terminal:

- S: Det, Proper-Noun, Aux, Verb
- NP: Det, Proper-Noun
- Nominal: Noun
- VP: Verb

Remaining Problems

Left recursive grammars

Ambiguity

Doing way too much reparsing

Left Recursion

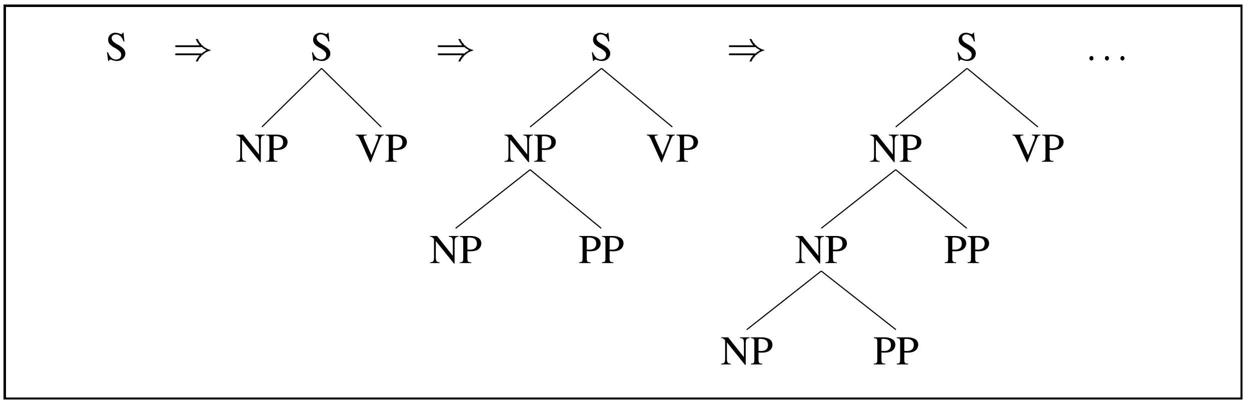
In top-down, depth-first, left-to-right parsers, a left recursive grammar can cause the search to never return due to an infinite expansion of a tree.

Example left-recursive rules:

- A derives A B
  - NP $\rightarrow$ NP PP
  - VP $\rightarrow$ VP PP
  - S $\rightarrow$ S and S
- A derives α A B and α derives epsilon
  - NP $\rightarrow$ Det Nom
  - Det $\rightarrow$ NP ' s

Left Recursion (continued)

NP $\rightarrow$ NP PP



Some (poor) solutions

- rewrite the grammar to a weakly equivalent one
  - might not get a correct or useful parse tree
- limit the depth during search
  - limit is arbitrary

Ambiguity

Chapter 8 discussed POS Ambiguity.

Here we are concerned with Structural Ambiguity.

Given a grammar, Global Ambiguity potentially leads to multiple parses for the same input (if we force it to).

- *I saw a woman with a telescope.*

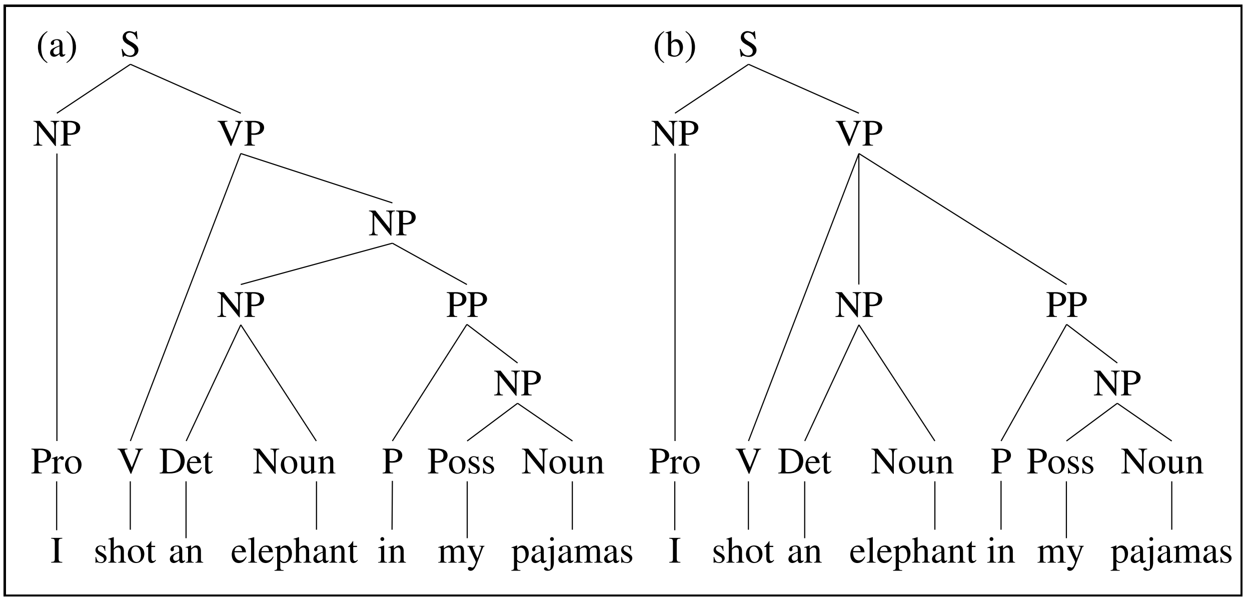
Local Ambiguity, in contrast, leads to hypotheses that are locally reasonable but eventually lead nowhere and result in inefficient backtracking. Filtering helps a little.

- *Book that flight.*

Common Structural Ambiguities

Attachment ambiguity

- a constituent can be attached to the parse tree at more than one place
- PP attachment ambiguity



- gerundive VP attachment ambiguity
  - *We saw the Eiffel Tower flying to Paris*

Coordination ambiguity

- *old (men and women)* vs. *(old men) and women*

Why is Ambiguity Problematic?

There are potentially an exponential number of parses for a sentence, so returning all structurally valid parses isn't always a good idea.

Some solutions:

- exploit regularities in the search space to derive common subparts only once (e.g., use dynamic programming to increase efficiency)
- heuristic search strategies
- incorporate semantics into the disambiguation process

Invariants

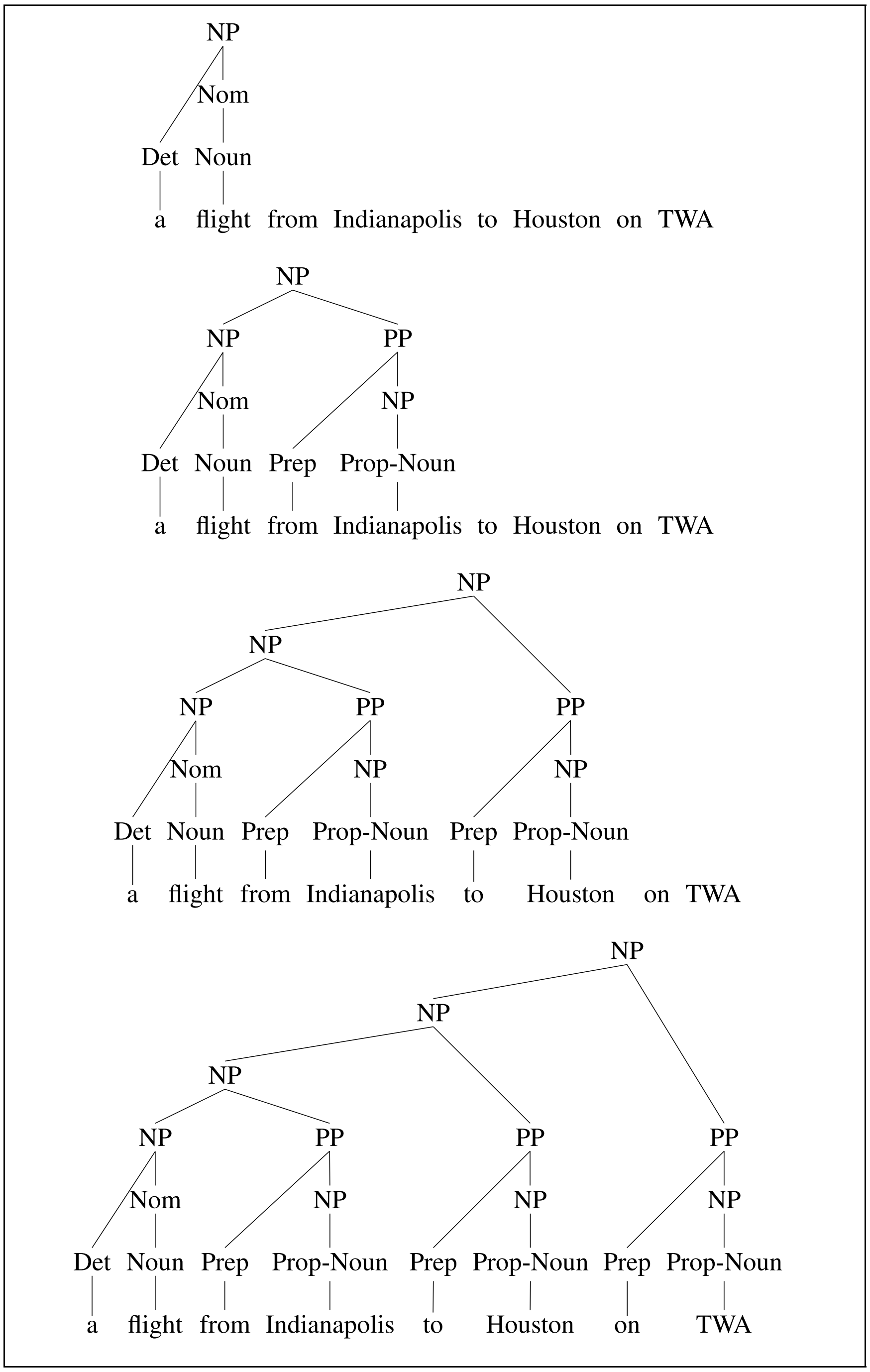
Despite all the ambiguity and backtracking there are invariants to be taken advantage of.

Consider parsing the following NP with the following rules:

- *a flight from Indianapolis to Houston on TWA*
- NP $\rightarrow$ Det Nominal
- NP $\rightarrow$ NP PP
- NP $\rightarrow$ ProperNoun

What happens with a top-down parser?

Invariants (continued)



Reuse via Dynamic Programming

Our current algorithm builds valid trees, discards them during backtracking, then rebuilds them.

- the subtree for *a flight* was derived 4 times!

Dynamic programming is one answer to problems that have sub-problems that get solved again and again.

We want an algorithm that fills a table with solutions to sub-problems that:

- does not do repeated work
- does top-down search with bottom-up filtering
- solves the left-recursion problem
- solves an exponential problem in $O(N^3)$ time

For Next Time

Finish Chapter 10

Dynamic Program/Earley Algorithm

Acknowledgements

Some of this material was taken from the course web pages of James Martin.