CS114 Lecture 10

Parsing and Partial Parsing

Thanks for Jurafsky & Martin & Prof. Pustejovksy for slides

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
- Partial parsing returns a set of constituents or small trees
  - Does not require a single S at the top

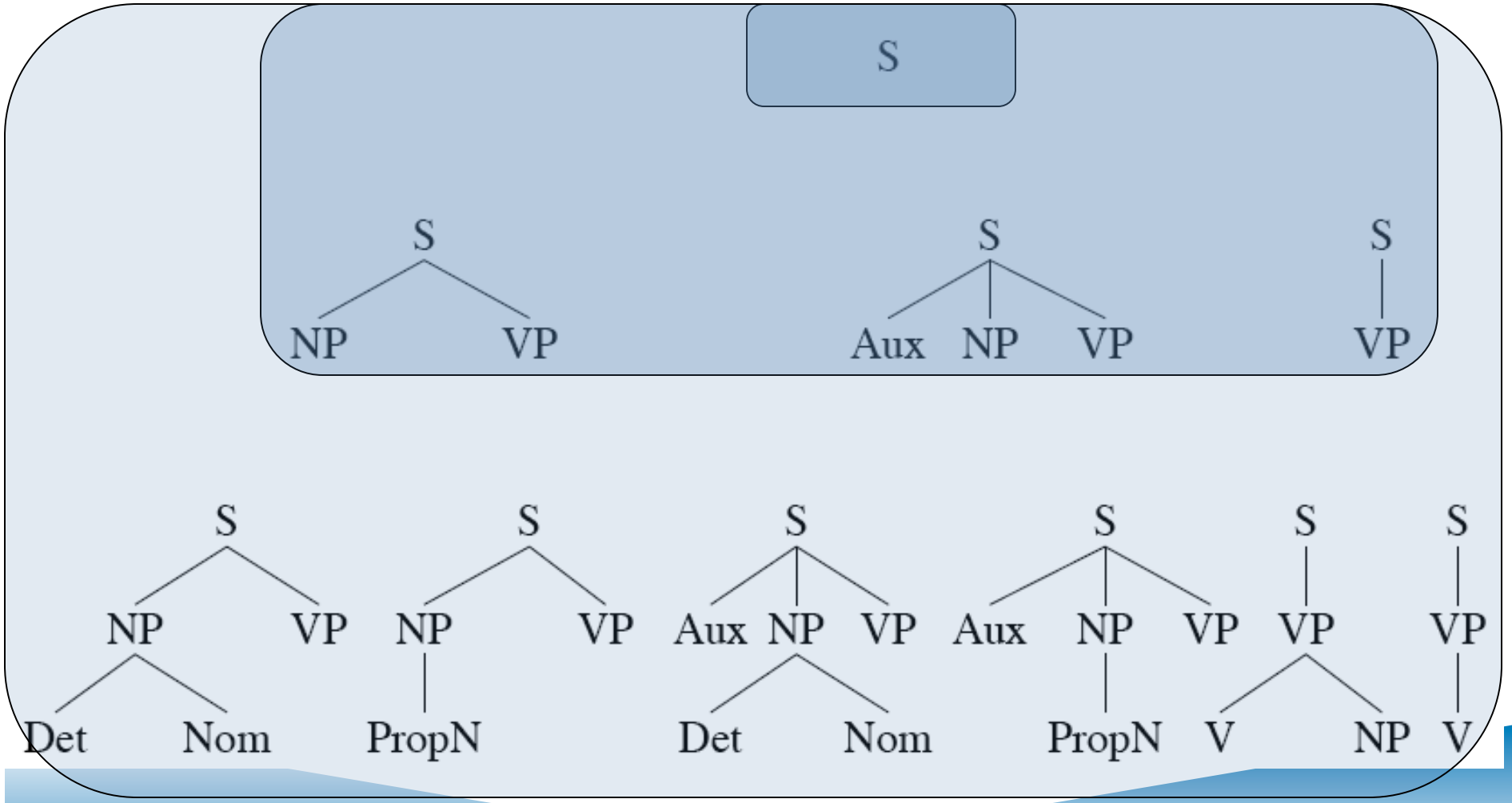
Top-Down and Bottom-Up

- Top-down
  - Only finds trees that form sentences
  - Finds trees that don't match the words
- Bottom-up
  - Only finds trees consistent with the words
  - Builds subtrees that can't combine to make sentences

Top-Down Search

- Sentences: trees rooted with S
- Start with S, expand nodes working downwards
- Try to reach the right set of words

Top Down Space



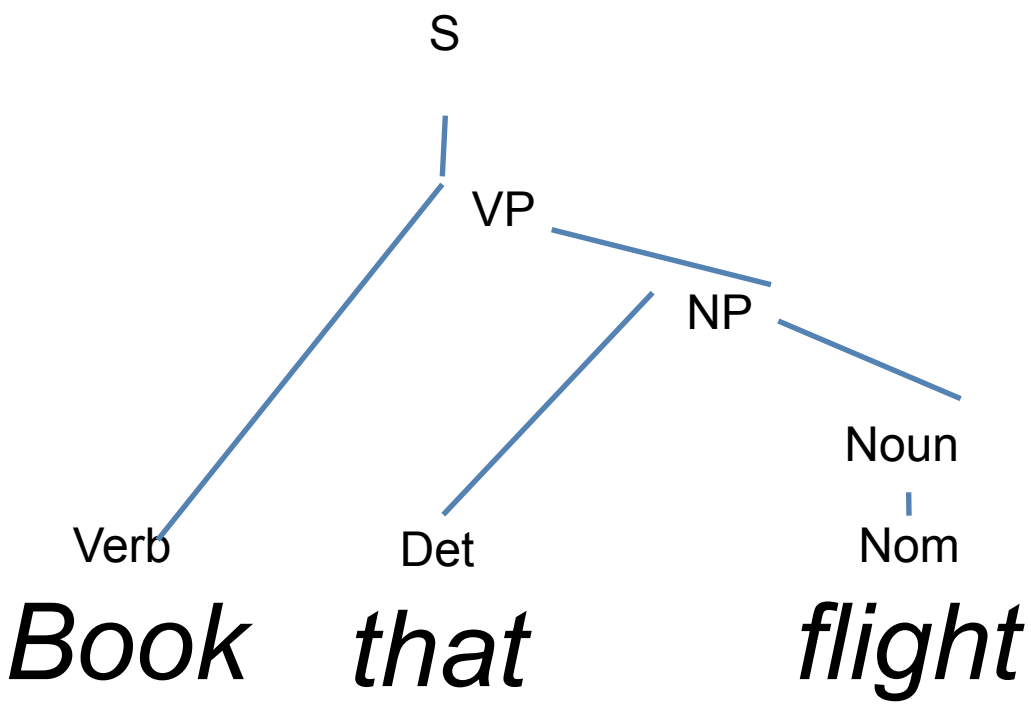
Bottom-Up Parsing

- Input must be matched exactly – very hard
- Start by combining words into small trees
- Work your way up, combining small trees into larger trees
- Try to get a single tree rooted at S

Bottom-Up Search

Grammar:

S ◊ NP VP
S◊ Aux NP VP
S ◊ VP
NP ◊ Det Nom
Nom ◊ Noun
Nom ◊ Noun Nom
Nom ◊ Nom PP
NP ◊ Proper-Noun
VP ◊ Verb
VP ◊ Verb NP
PP ◊ Prep NP



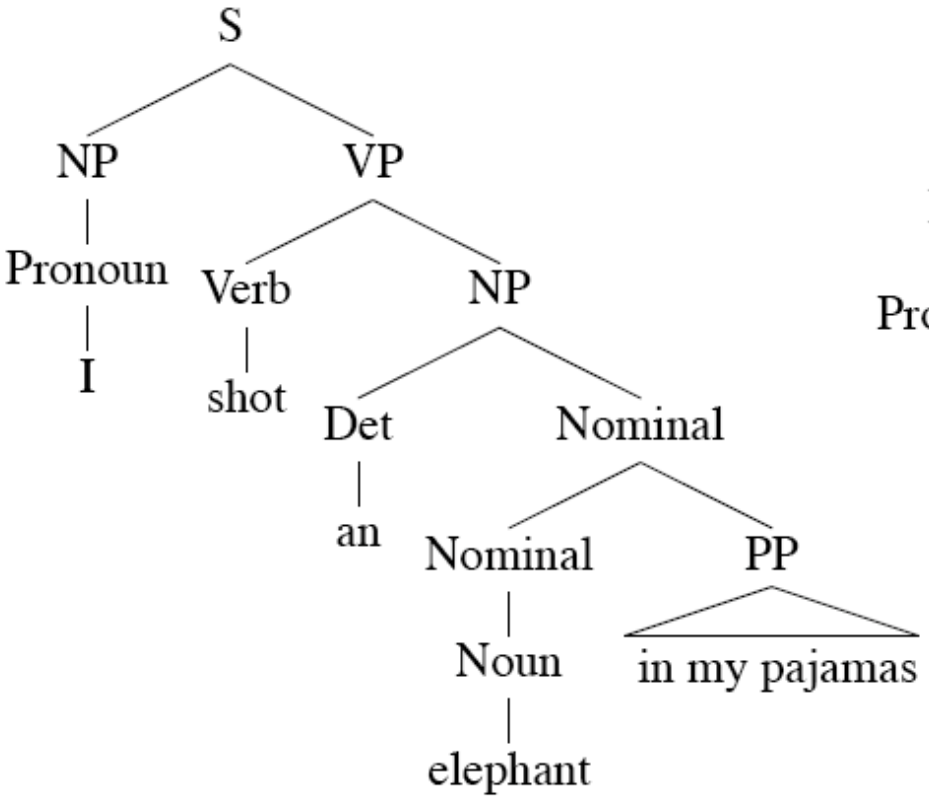
Control

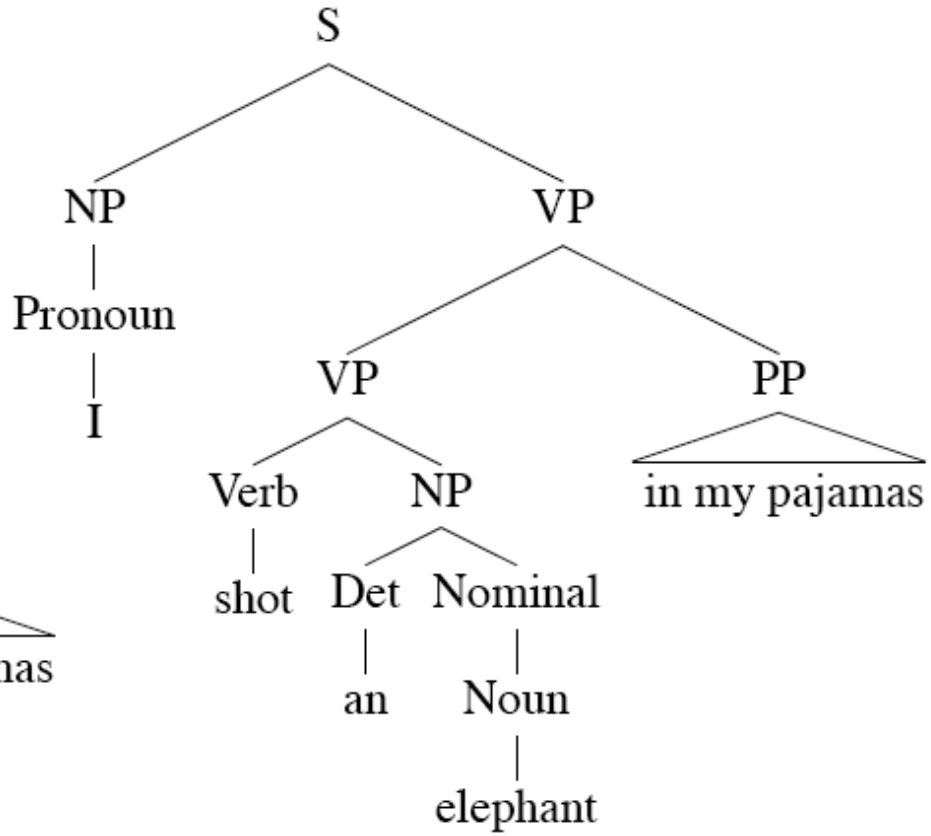
- How to explore the search space?
  - Which node to try to expand next
  - Which grammar rule to use to expand a node
  - Wrong choice leads to unsolvable problem
- One approach is called backtracking.
  - Make a choice, if it works out then fine
  - If not then back up and make a different choice

Problems

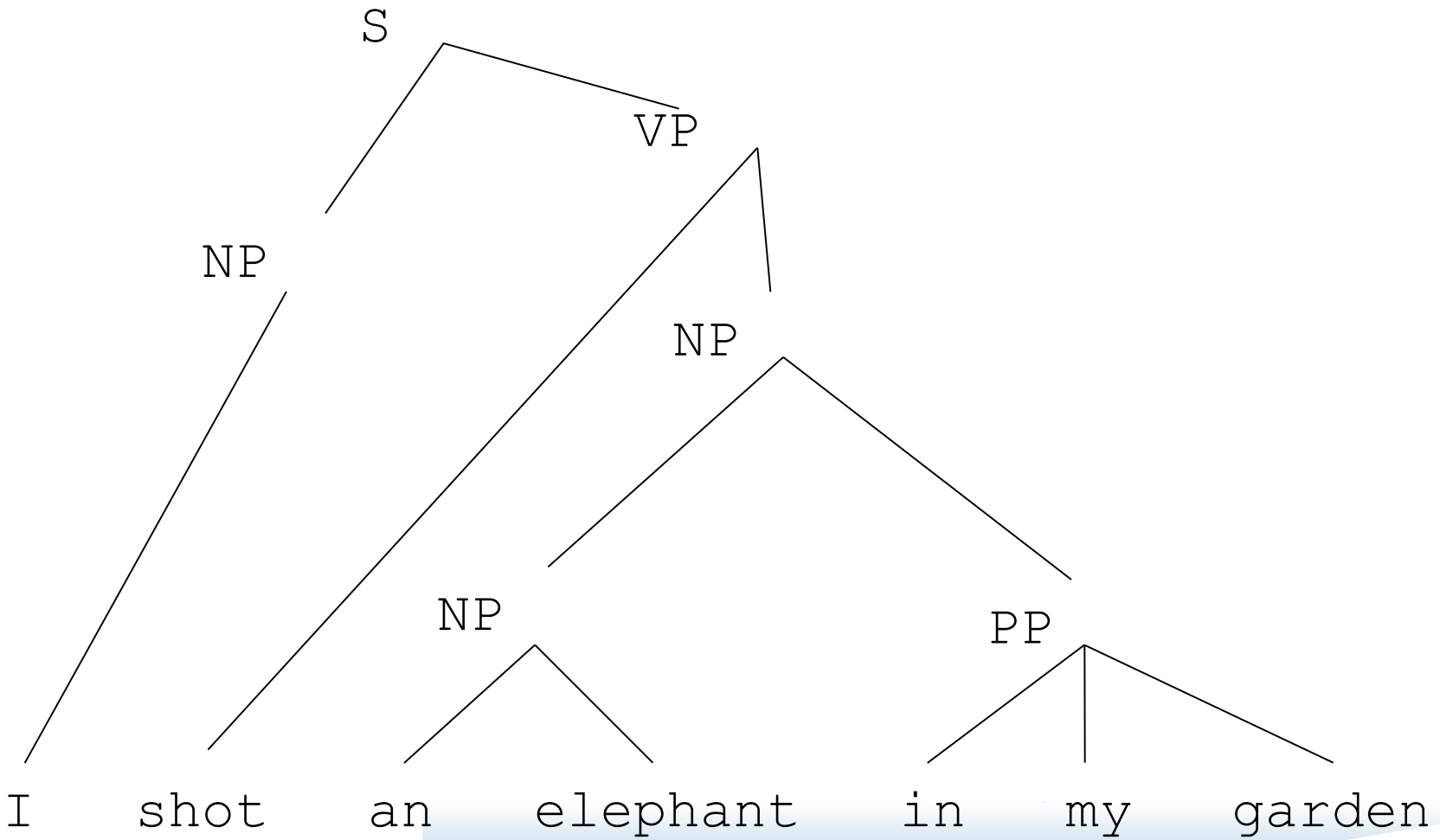
- Backtracking is not good enough
  - Ambiguity
  - Shared subproblems

Ambiguity

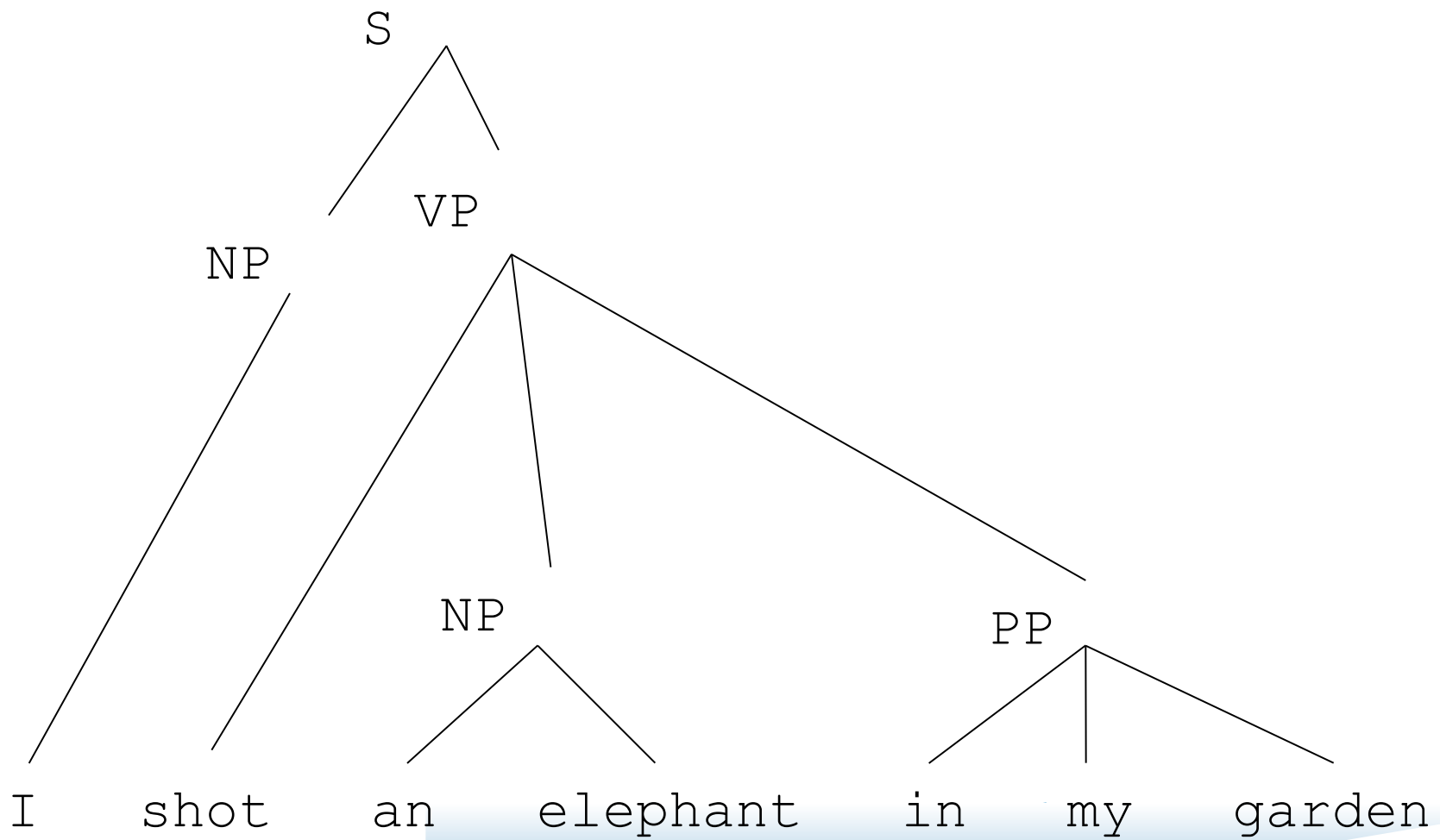

S
NP
VP
Pronoun
Verb
NP
I
shot
Det
Nominal
an
Nominal
PP
Noun
in my pajamas
elephant


S
NP
VP
Pronoun
VP
PP
I
Verb
NP
in my pajamas
shot
Det
Nominal
an
Noun
elephant

The elephant is in the garden



I fired from the garden



Shared Sub-Problems

- No matter what kind of search (top-down or bottom-up or mixed) that we choose.
  - Redoing work is wasted effort
  - Naïve backtracking means duplicated work.
  - Dynamic Programming...

Parsing so far

- Goals

- Define a language

- A grammar defines a language L over token set (terminals)

- A parser determines whether a particular string of tokens is in the language

- Determine the constituent structure of a string of tokens (assuming it's in the language)

- Determine the meaning (we're not there yet)

Parsing so far

- Information
  - Categories (nonterminals) of the tokens
  - Which adjacent nonterminals form tokens (rules)
  - What additional conditions must be met in order for a constituent to be formed (e.g. features, words)

Parsing Types

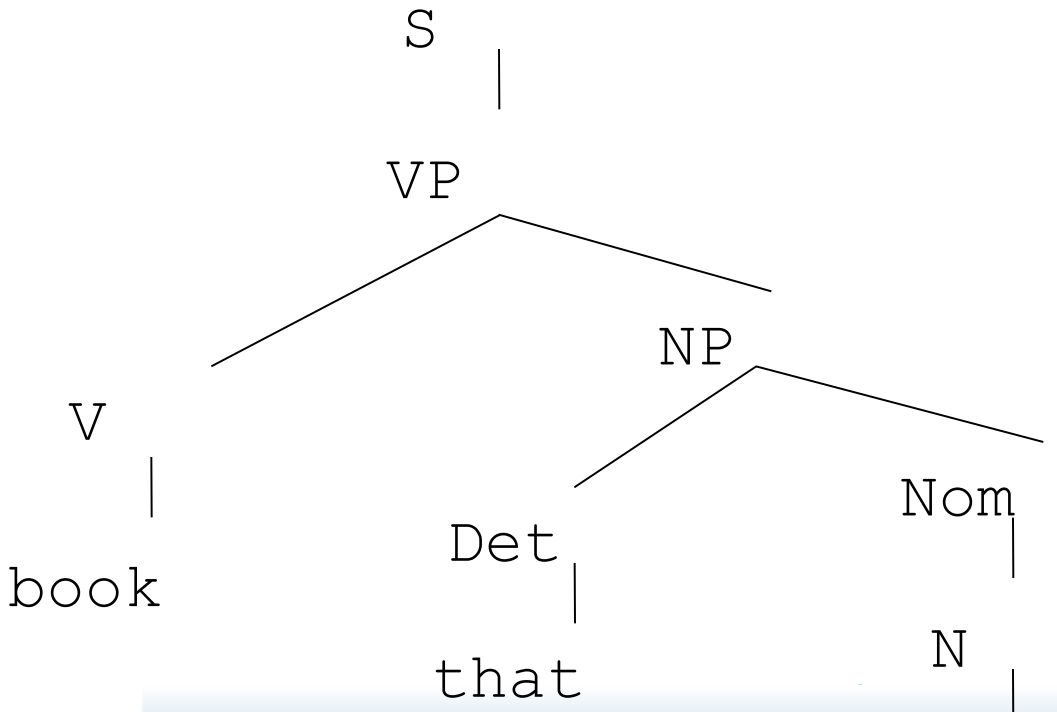
- Next:
  - Chunking
  - Partial parsing
- We'll come back to Parsing as Search
  - CFGS
    - Top down, bottom up
    - Earley's algorithm
  - Probabalistic CFGs
  - Unification Grammars

Motivation: Parsing is hard

- Example (wsj_0001):
  - Pierre Vinken, 61 years old, will join the board as a nonexecutive director Nov. 29.
  - Mr. Vinken is chairman of Elsevier N.V., the Dutch publishing group.
  - Pierre/NNP Vinken/NNP ,/, 61/CD years/NNS old/JJ ,/, will/MD join/VB the/DT board/NN as/IN a/DT nonexecutive/JJ director/NN Nov./NNP 29/CD ./.
  - Mr./NNP Vinken/NNP is/VBZ chairman/NN of/IN Elsevier/NNP N.V./NNP ,/, the/DT Dutch/NNP publishing/ VBG group/NN ./.

Parsing Problem

- Given grammar G and sentence A **discover all valid parse trees** for G that **exactly cover** A



Slide modified from Steven Bird's

Shallow (Chunk) Parsing

Goal: divide a sentence into a sequence of chunks.

- Chunks are non-overlapping regions of a text

[I] saw [a tall man] in [the park].

- Chunks are non-recursive
  - Cannot contain other chunks
- Chunks are non-exhaustive
  - Not all words are included in chunks

Slide modified from Steven Bird's

Chunk Parsing Examples

- Noun-phrase chunking:

[I] saw [a tall man] in [the park].

- Verb-phrase chunking:

The man who [was in the park] [saw me].

- Question answering:
  - **What [Spanish explorer] discovered [the Mississippi River]?**

Slide modified from Steven Bird's

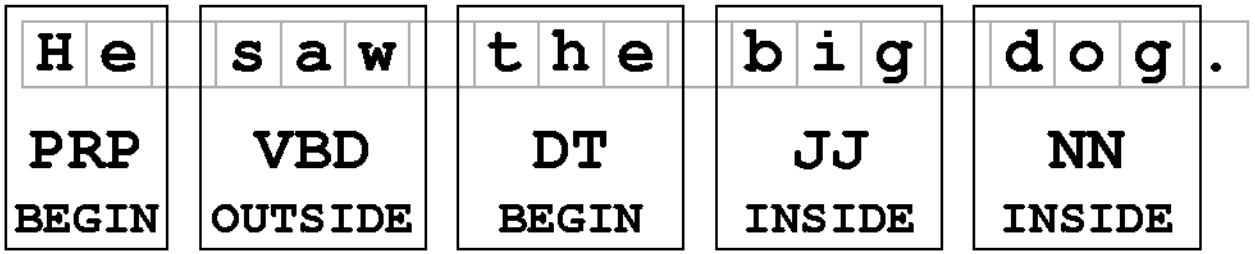
Shallow Parsing: Motivation

- Locating information
  - e.g. index a document collection on its noun phrases
- Ignoring information
  - Generalize in order to study higher-level patterns
    - e.g. phrases involving “gave” in Penn treebank:
      - gave NP; gave up NP in NP; gave NP up; gave NP help; gave NP to NP
  - Sometimes a full parse has too much structure

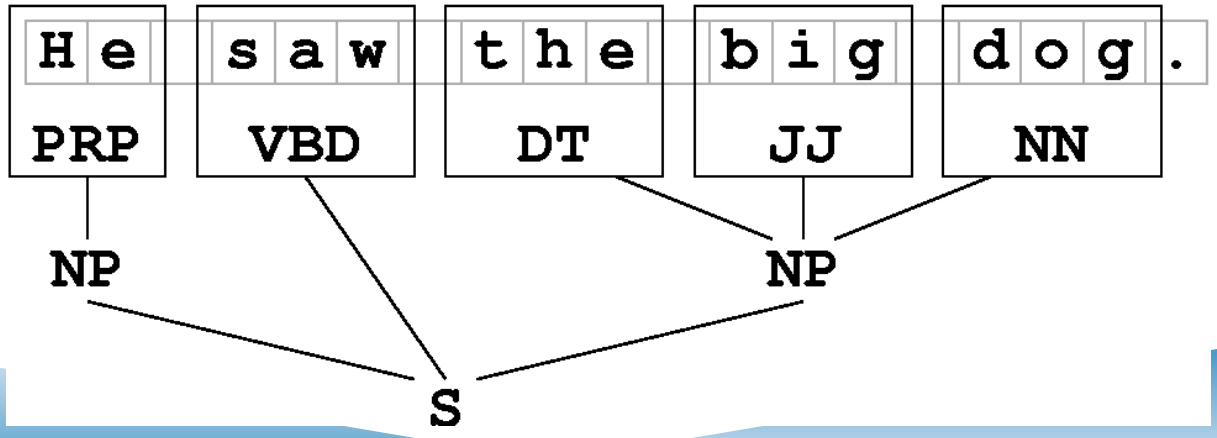
Slide modified from Steven Bird's

Representation

- BIO (or IOB)



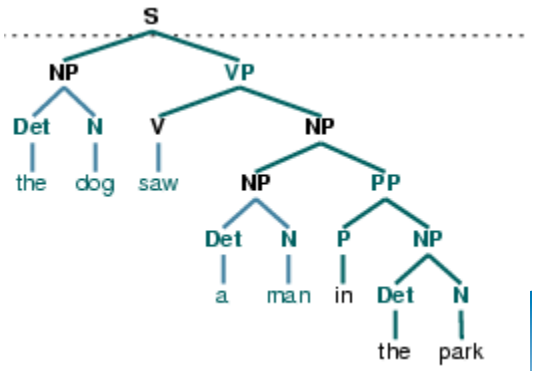
Trees



Slide modified from Steven Bird's

Comparison with Full Syntactic

- Parsing often an intermediate stage
  - later stages draw on the parse for their own purposes
- Full parsing sufficient, often not necessary
  - Often more information than we need
- Shallow parsing is an easier problem
- Less structure, no recursion



Slide modified from Steven Bird's

Chunks and Constituency

Constituents: ***[[a tall man] [in [the park]]].***

Chunks: ***[a tall man] in [the park].***

- A constituent is part of some higher unit in the hierarchical syntactic parse
- Chunks are *not constituents*
  - Constituents are recursive
  - Chunks do not cross major constituent boundaries (why?)

Slide modified from Steven Bird's

Chunk Parsing in NLTK

- Chunk parsers usually ignore lexical content
  - Only need to look at part-of-speech tags
- Possible steps in chunk parsing
  - Chunking, unchunking
  - Chinking
  - Merging, splitting

Slide modified from Steven Bird's

Chunking

- Define a regular expression that matches the sequences of tags in a chunk

  A simple noun phrase chunk regexp:

(Note that <NN.*> matches any tag starting with NN)

<DT>? <JJ>* <NN.*>

- Chunk all matching subsequences:

the/DT little/JJ cat/NN sat/VBD on/IN the/DT mat/NN

[the/DT little/JJ cat/NN] sat/VBD on/IN [the/DT mat/NN]

- If matching subsequences overlap, first 1 gets priority

Unchunking

- Remove any chunk with a given pattern
  - e.g., unChunkRule('<NN|DT>+', 'Unchunk NNDT')
  - Combine with Chunk Rule <NN|DT|JJ>+
- Chunk all matching subsequences:
  - Input:

the/DT little/JJ cat/NN sat/VBD on/IN the/DT mat/NN

  - Apply chunk rule

[the/DT little/JJ cat/NN] sat/VBD on/IN [the/DT mat/NN]

  - Apply unchunk rule

[the/DT little/JJ cat/NN] sat/VBD on/IN the/DT mat/NN

Slide modified from Steven Bird's

Chinking

- A *chink* is a subsequence of the text that is not a chunk.
- Define a regular expression that matches the sequences of tags in a chink

A simple chink regexp for finding NP chunks:
(<VB.?>|<IN>)+

- First apply chunk rule to chunk everything
  - Input: the/DT little/JJ cat/NN sat/VBD on/IN the/DT mat/NN
  - ChunkRule('<.*>+', 'Chunk everything')

[the/DT little/JJ cat/NN sat/VBD on/IN the/DT mat/NN]

  - Apply Chink rule above:

[the/DT little/JJ cat/NN] sat/VBD on/IN [the/DT mat/NN]

Chunk ***Chink*** ***Chunk***

Slide modified from Steven Bird's

Merging

- Combine adjacent chunks into a single chunk
  - Define a regular expression that matches the sequences of tags on both sides of the point to be merged
- Example:
  - Merge a chunk ending in JJ with a chunk starting with NN

    MergeRule('<JJ>', '<NN>', 'Merge adjs and nouns')

[the/DT little/JJ] [cat/NN] sat/VBD on/IN the/DT mat/NN

[the/DT little/JJ cat/NN] sat/VBD on/IN the/DT mat/NN

- Splitting is the opposite of merging

Cascaded Chunking

- Goal: create chunks that include other chunks
- Examples:
  - PP consists of preposition + NP
  - VP consists of verb followed by PPs or NPs

Slide modified from Steven Bird's

Cascaded Chunking

```
# find NP chunks
>>> rule = ChunkRule(r'<DT>?<JJ>*<NN.*>', 'Chunk NPs')
>>> parser = RegexpChunkParser([rule], chunk_node='NP',
...             top_node='S', TREE='NP-CHUNKS', SUBTOKENS='WORDS')
>>> text_tok = Token(WORDS=s.leaves())
>>> parser.parse(text_tok)

# find VP chunks
>>> rule = ChunkRule(r'<VB.*><.*>*', 'Chunk VPs')
>>> parser = RegexpChunkParser([rule], chunk_node='VP',
...             top_node='S', SUBTOKENS='NP-CHUNKS')
>>> parser.parse( text_tok )
>>> print text_tok['TREE']
(S:
  (NP: <the/DT> <little/JJ> <cat/NN>)
  (VP: <sat/VBD> <on/IN> (NP: <the/DT> <mat/NN>)))
```

Next Assignment

- Use the nltk chunker and write chunking rules and apply it to the Treebank data.
- Evaluate your performance on the "tagged" treebank data (which is already chunked).
- There are 200 sentences. Develop the grammar using the first 150 sentences and test on the last 50 sentences.

Example

- Write rules using POS to produce the chunks
- Section 7.3, NLTK book
- Evaluate performance
- Definitely > 70% f-measure, shoot for >90%

[A/DT form/NN]

of/IN

[asbestos/NN]

once/RB used/VBN to/TO make/VB

[Kent/NNP]

[cigarette/NN filters/NNS]

has/VBZ caused/VBN

[a/DT high/JJ percentage/NN]

of/IN

[cancer/NN deaths/NNS]

.....