Basic Parsing with Context-Free Grammars

Some slides adapted from Julia Hirschberg and Dan Jurafsky

Announcements

- To view past videos:
  - http://globe.cvn.columbia.edu:8080/oncampus.php?c=133ae14752e27fde909fdbd64c06b337

- Usually available only for 1 week. Right now, available for all previous lectures

Homework Questions?

Evaluation

Syntactic Parsing

Syntactic Parsing

- Declarative formalisms like CFGs, FSAs define the *legal strings of a language* -- but only tell you ‘this is a legal string of the language X’
- Parsing algorithms specify how to recognize the strings of a language and assign each string one (or more) syntactic analyses

CFG: Example

the small boy likes a girl

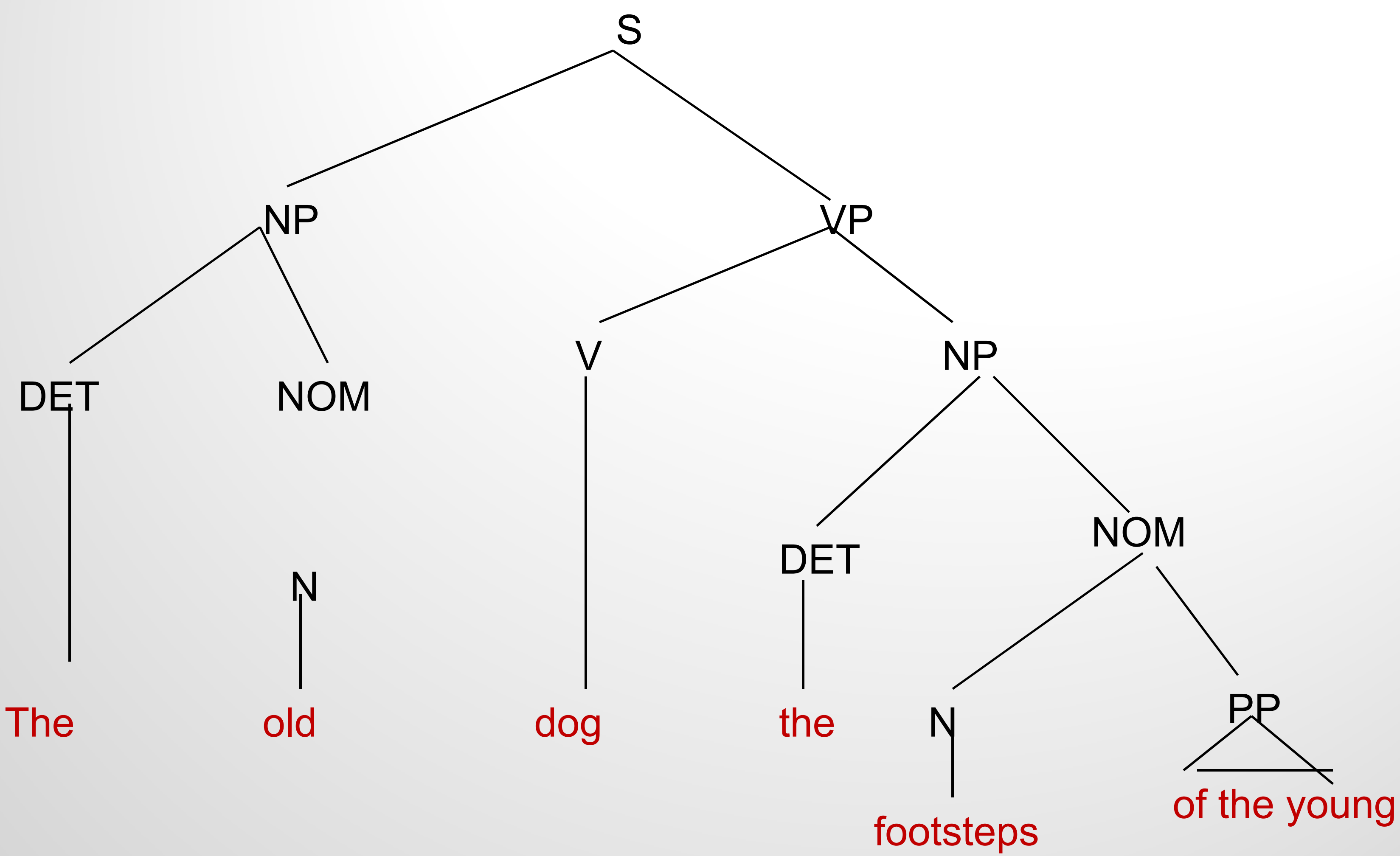
- Many possible CFGs for English, here is an example (fragment):
  - S → NP VP
  - VP → V NP
  - NP → Det N | Adj NP
  - N → boy | girl
  - V → sees | likes
  - Adj → big | small
  - DetP → a | the

  - *big the small girl sees a boy
  - John likes a girl
  - I like a girl
  - I sleep
  - The old dog the footsteps of the young

Modified CFG

S → NP VP	VP → V
S → Aux NP VP	*VP -> V PP*
S -> VP	*PP -> Prep NP*
NP → Det Nom	N → old | dog | footsteps | young | flight
NP → PropN	V → dog | include | prefer | book
NP -> Pronoun	
Nom -> Adj Nom	Aux → does
Nom → N	Prep → from | to | on | of
Nom → N Nom	PropN → Bush | McCain | Obama
Nom → Nom PP	Det → that | this | a| the
VP → V NP	Adj -> old | green | red

Parse Tree for 'The old dog the footsteps of the young' for Prior CFG



Parsing as a Form of Search

- Searching FSAs
  - Finding the right path through the automaton
  - Search space defined by structure of FSA
- Searching CFGs
  - Finding the right parse tree among all possible parse trees
  - Search space defined by the grammar
- Constraints provided by *the input sentence* and *the automaton or grammar*

Top-Down Parser

- Builds from the root S node to the leaves
- Expectation-based
- Common search strategy
  - Top-down, left-to-right, backtracking
  - Try first rule with LHS = S
  - Next expand all constituents in these trees/rules
  - Continue until leaves are POS
  - Backtrack when candidate POS does not match input string

Rule Expansion

- “The old dog the footsteps of the young.”
  - Where does backtracking happen?
  - What are the computational disadvantages?
  - What are the advantages?

Bottom-Up Parsing

- Parser begins with words of input and builds up trees, applying grammar rules whose RHS matches

  Det N V Det N Prep Det N
  The old dog the footsteps of the young.

  Det Adj N Det N Prep Det N
  The old dog the footsteps of the young.

  Parse continues until an S root node reached or no further node expansion possible

Det N V Det N Prep Det N
The old dog the footsteps of the young.
Det Adj N Det N Prep Det N

Bottom-up parsing

- When does disambiguation occur?
- What are the computational advantages and disadvantages?

What's right/wrong with….

- Top-Down parsers – they never explore illegal parses (e.g. which can't form an S) -- but waste time on trees that can never match the input
- Bottom-Up parsers – they never explore trees inconsistent with input -- but waste time exploring illegal parses (with no S root)
- For both: find a control strategy -- how explore search space efficiently?
  - Pursuing all parses in parallel or backtrack or …?
  - Which rule to apply next?
  - Which node to expand next?

Some Solutions

Dynamic Programming Approaches – Use a chart to represent partial results

- CKY Parsing Algorithm
  - Bottom-up
  - Grammar must be in Normal Form
  - The parse tree might not be consistent with linguistic theory
- Early Parsing Algorithm
  - Top-down
  - Expectations about constituents are confirmed by input
  - A POS tag for a word that is not predicted is never added
- Chart Parser

Earley Parsing

- Allows arbitrary CFGs
- Fills a table in a single sweep over the input words
  - 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.

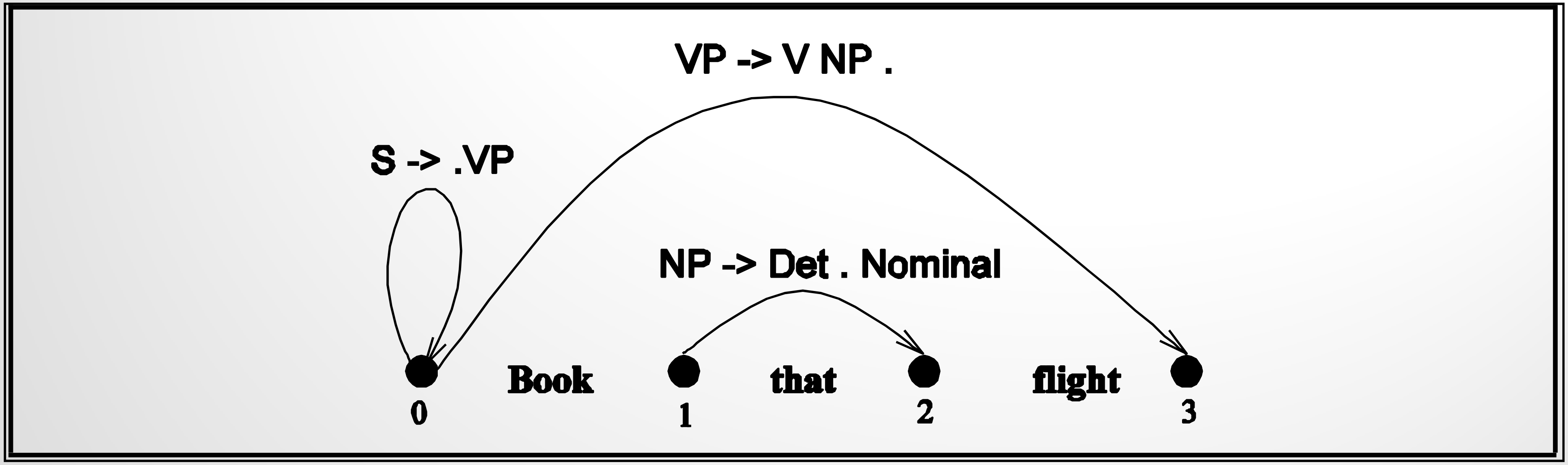
S -> · VP	A VP is predicted
NP -> Det · Nominal	An NP is in progress
VP -> V NP ·	A VP has been found

States/Locations

- It would be nice to know where these things are in the input so…

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

Graphically



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+1 and is complete.
- If that's the case you're done.
  - S -> α · [0,n+1]

Earley Algorithm

- March through chart left-to-right.
- At each step, apply 1 of 3 operators
  - Predictor
    - Create new states representing top-down expectations
  - Scanner
    - Match word predictions (rule with word after dot) to words
  - Completer
    - When a state is complete, see what rules were looking for that completed constituent

Predictor

- Given a state
  - With a non-terminal to right of dot (not a part-of-speech category)
  - Create a new state for each expansion of the non-terminal
  - Place these new states into same chart entry as generated state, beginning and ending where generating state ends.
  - So predictor looking at
    - S -> . VP [0,0]
  - results in
    - VP -> . Verb [0,0]
    - VP -> . Verb NP [0,0]

Scanner

- Given a state
  - With a non-terminal to right of dot that is a part-of-speech category
  - If the next word in the input matches this POS
  - Create a new state with dot moved over the non-terminal
  - So scanner looking at VP -> . Verb NP [0,0]
  - If the next word, “book”, can be a verb, add new state:
    - VP -> Verb . NP [0,1]
  - Add this state to chart entry following current one
  - Note: Earley algorithm uses top-down input to disambiguate POS! Only POS predicted by some state can get added to chart!

Completer

- Applied to a state when its dot has reached right end of role.
- Parser has discovered a category over some span of input.
- Find and advance all previous states that were looking for this category
  - copy state, move dot, insert in current chart entry
- Given:
  - NP -> Det Nominal . [1,3]
  - VP -> Verb. NP [0,1]
- Add
  - VP -> Verb NP . [0,3]

How do we know we are done?

- Find an S state in the final column that spans from 0 to n+1 and is complete.

- If that's the case you're done.
  - S -> α · [0,n+1]

Earley

- More specifically…

1. Predict all the states you can upfront
2. Read a word
   1. Extend states based on matches
   2. Add new predictions
   3. Go to 2
3. Look at N+1 to see if you have a winner

Example

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

CFG for Fragment of English

S → NP VP	VP → V
S → Aux NP VP	PP -> Prep NP
NP → Det Nom	N → old | dog | footsteps | young
NP →PropN	V → dog | include | prefer
Nom -> Adj Nom	Aux → does
Nom → N	Prep →from | to | on | of
Nom → N Nom	PropN → Bush | McCain | Obama
Nom → Nom PP	Det → that | this | a| the
VP → V NP	Adj -> old | green | red

Example

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	$NP \rightarrow \bullet Proper\text{-}Noun$	[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

Example

Chart[1]	S12	*Verb* → *book* •	[0,1]	Scanner
	S13	*VP* → *Verb* •	[0,1]	Completer
	S14	*VP* → *Verb* • *NP*	[0,1]	Completer
	S15	*VP* → *Verb* • *NP PP*	[0,0]	Predictor
	S16	*VP* → *Verb* • *PP*	[0,0]	Predictor
	S17	*S* → *VP* •	[0,1]	Completer
	S18	*VP* → *VP* • *PP*	[0,1]	Completer
	S19	*NP* → • *Pronoun*	[1,1]	Predictor
	S20	*NP* → • *Proper-Noun*	[1,1]	Predictor
	S21	*NP* → • *Det Nominal*	[1,1]	Predictor
	S22	*PP* → • *Prep NP*	[1,1]	Predictor

Example

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

Details

- What kind of algorithms did we just describe
  - Not parsers – recognizers
    - The presence of an S state with the right attributes in the right place indicates a successful recognition.
    - But no parse tree… no parser
    - That's how we solve (not) an exponential problem in polynomial time

Converting Earley from Recognizer to Parser

- With the addition of a few pointers we have a parser
- Augment the “Completer” to point to where we came from.

Augmenting the chart with structural information

Chart[1]

S8	*Verb*	*book*	[0,1]	Scanner	
S9	*VP*	*Verb*	[0,1]	Completer	S8
S10	*S*	*VP*	[0,1]	Completer	S9
S11	*VP*	*Verb NP*	[0,1]	Completer	S8
S12	*NP*	*Det NOMINAL*	[1,1]	Predictor	
S13	*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

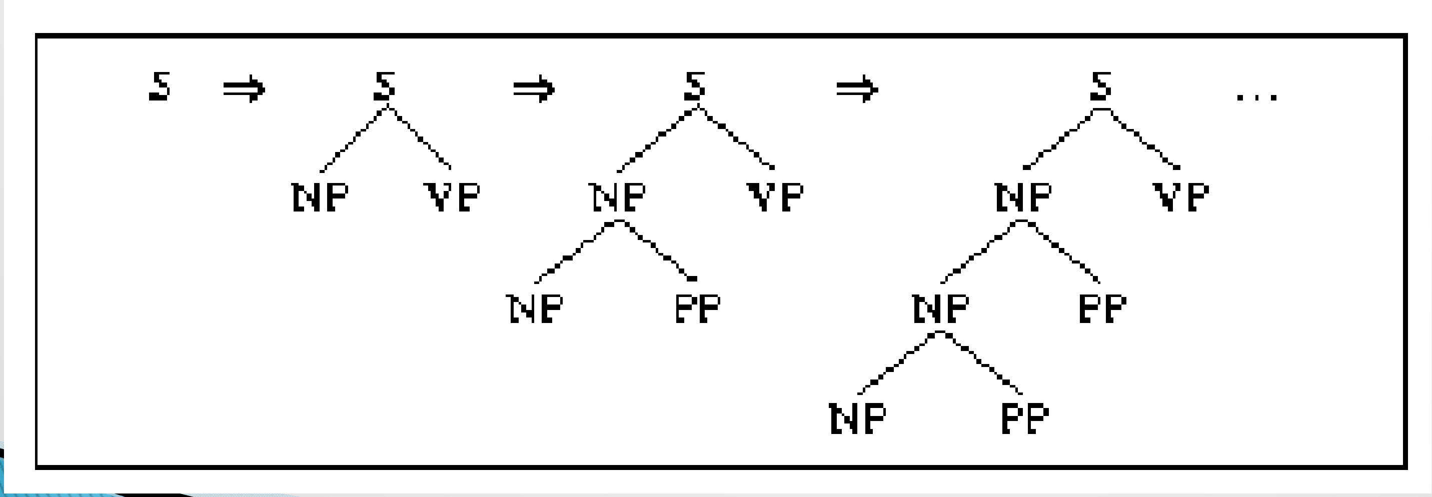
Retrieving Parse Trees from Chart

- All the possible parses for an input are in the table
- We just need to read off all the backpointers from every complete S in the last column of the table
- Find all the S -> X . [0,N+1]
- Follow the structural traces from the Completer
- Of course, this won't be polynomial time, since there could be an exponential number of trees
- We can at least represent ambiguity efficiently

Left Recursion vs. Right Recursion

- Depth-first search will never terminate if grammar is *left recursive* (e.g. NP --> NP PP)

$$(A \xrightarrow{*} \alpha AB, \alpha \xrightarrow{*} \varepsilon)$$



- Solutions:
  - Rewrite the grammar (automatically?) to a *weakly equivalent* one which is not left-recursive

    e.g. The man {on the hill with the telescope…}

    NP → NP PP (wanted: Nom plus a sequence of PPs)

    NP → Nom PP

    NP → Nom

    Nom → Det N

    …becomes…

    NP → Nom NP’

    Nom → Det N

    NP’ → PP NP’ (wanted: a sequence of PPs)

    NP’ → e

    - *Not so obvious what these rules mean…*

- Harder to detect and eliminate *non-immediate left recursion*
    - NP --> Nom PP
    - Nom --> NP

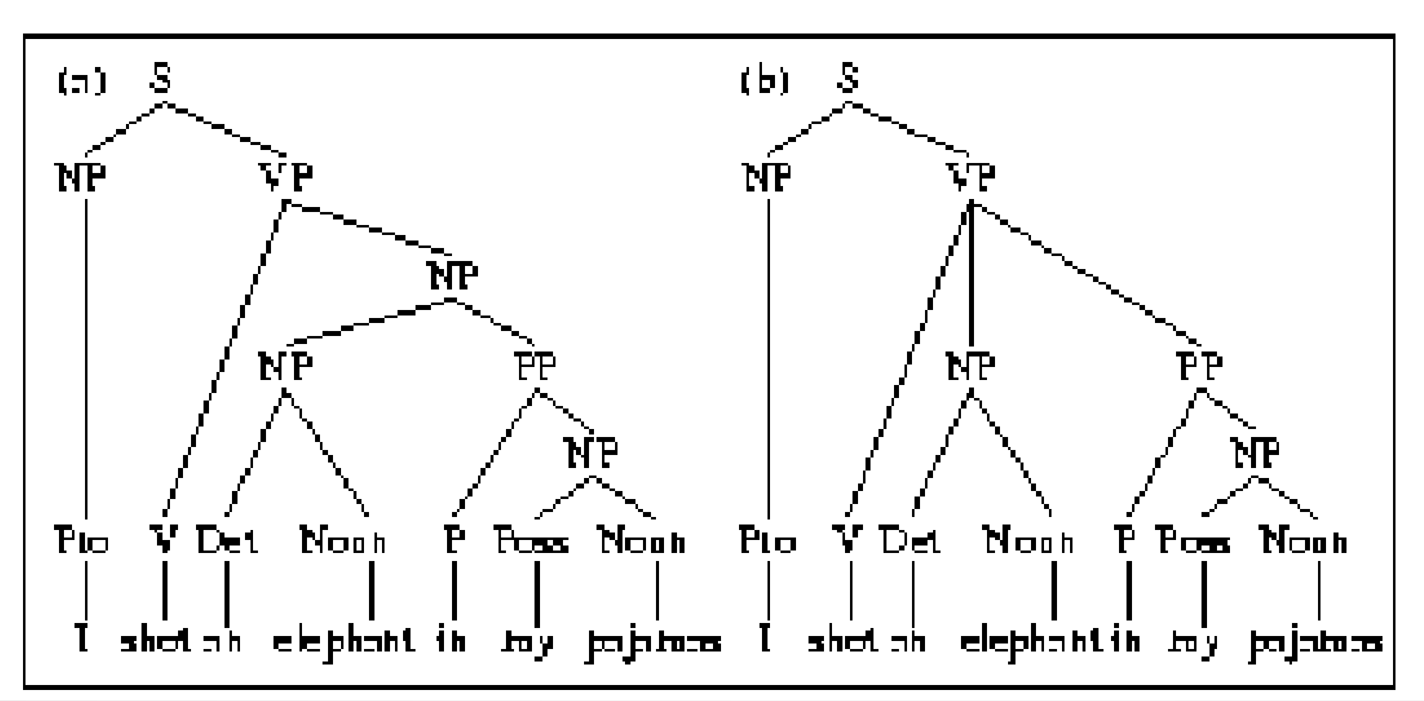
- Fix depth of search explicitly

- Rule ordering: non-recursive rules first
  - NP --> Det Nom
  - NP --> NP PP

Another Problem: Structural ambiguity

- Multiple legal structures
  - Attachment (e.g. I saw a man on a hill with a telescope)
  - Coordination (e.g. younger cats and dogs)
  - NP bracketing (e.g. Spanish language teachers)

NP vs. VP Attachment



- Solution?
  - Return all possible parses and disambiguate using “other methods”

Summing Up

- Parsing is a search problem which may be implemented with many control strategies
  - Top-Down or Bottom-Up approaches each have problems
    - Combining the two solves some but not all issues
  - Left recursion
  - Syntactic ambiguity
- Next time: Making use of statistical information about syntactic constituents
  - Read Ch 14