Introduction to Parsing:
Top-Down vs. Bottom-Up
Structural Ambiguities
Parsing algorithms defined:

• The process of finding a derivation (i.e. sequence of productions) leading from the START symbol to the TERMINAL symbols
  – Shows how a particular sentence *could be* generated by the rules of the grammar
• If sentence is structurally ambiguous, more than one possible derivation is produced
• Can solve both the recognition and analysis problems
  – Is this sentence derived from this grammar?
  – Give the derivation(s) that can derive this sentence.
• Parsing algorithms give a strategy for finding a derivation by making choices among the derivation rules and deciding when the derivation is complete or not.
Top-down Parser

• Goal-driven

• At each stage, the parser looks at goal of a non-terminal symbol (starting with S) and then sees which rules can be applied
  – Typically progresses from top-to-bottom, left-to-right
  – Non-deterministic (can be rewritten in more than one way)

• When rules derive lexical elements (words), check with the input to see if the right sentence is being derived

• An algorithm may include a backtracking mechanism
  – When it is determined that the wrong rule has been used, it backs up and tries another rule
Example Grammar

- The flight grammar from the text has multiple rules for S:

\[
\begin{align*}
S & \rightarrow NP \ VP \\
S & \rightarrow Aux \ NP \ VP \\
S & \rightarrow VP \\
NP & \rightarrow Pronoun \\
NP & \rightarrow Proper-Noun \\
NP & \rightarrow Det \ Nominal \\
Nominal & \rightarrow Noun \\
Nominal & \rightarrow Nominal \ Noun \\
Nominal & \rightarrow Nominal \ PP \\
VP & \rightarrow Verb \\
VP & \rightarrow Verb \ NP \\
VP & \rightarrow Verb \ NP \ PP \\
VP & \rightarrow Verb \ PP \\
VP & \rightarrow VP \ PP \\
PP & \rightarrow Preposition \ NP
\end{align*}
\]

\[
\begin{align*}
Det & \rightarrow that \ | \ this \ | \ a \\
Noun & \rightarrow book \ | \ flight \ | \ meal \ | \ money \\
Verb & \rightarrow book \ | \ include \ | \ prefer \\
Pronoun & \rightarrow I \ | \ she \ | \ me \\
Proper-Noun & \rightarrow Houston \ | \ TWA \\
Aux & \rightarrow does \\
Preposition & \rightarrow from \ | \ to \ | \ on \ | \ near \ | \ through
\end{align*}
\]
Example Derivation

- Derivation for “Book that flight” (from the text)
  - The Start symbol
    \[
    \text{S}
    \]
  - Can derive 3 rules as follows:
    \[
    \begin{align*}
    \text{S} & \quad \text{NP} \quad \text{VP} \\
    \text{S} & \quad \text{AUX} \quad \text{NP} \quad \text{VP} \\
    \text{S} & \quad \text{VP}
    \end{align*}
    \]
  - Each non-terminal can derive additional rules
    \[
    \begin{align*}
    \text{S} & \quad \text{NP} \quad \text{VP} \\
    \text{S} & \quad \text{NP} \quad \text{VP} \\
    \text{S} & \quad \text{NP} \quad \text{VP} \\
    \text{S} & \quad \text{VP} \quad \text{NP} \quad \text{VP} \\
    \text{S} & \quad \text{VP} \quad \text{NP} \quad \text{V} \quad \text{NP} \quad \text{V}
    \end{align*}
    \]
  - Only the last two trees can derive the word “book” as first in the input
Top-down Parsing Demo

• NLTK parsing demos
  – Top-down parsing using a recursive descent algorithm
    • Top down parsing with back-tracking
    • Must not have left-recursion in the grammar rules

  nltk.app.rdparserr()

• Described in NLTK book, Chapter 8, Analyzing Sentence Structure
Bottom-up Parser

- Data-driven

- Looks at words in input string first, checks / assigns their category(ies), and tries to combine them into acceptable structures in the grammar

- Involves scanning the derivation so far for sub-strings which match the right-hand-side of grammar / production rules and using the rule that would show their derivation from the non-terminal symbol of that rule
Bottom-up Derivation

- Starts with input text

- derive the text from rules, in this case, two possible lexical rules

- Each of those can be derived from nonterminals
Bottom-Up Derivation

• Only the rightmost tree can continue the derivation here:

• And only one succeeds: \( S \rightarrow VP \)
Bottom-up Parsing

• Algorithm called shift/reduce parsing
  – Scans the input from left to right and keeps a “stack” of the partial parse tree so far
  – Chooses shift or reduce operations
    • The shift operation looks at the next input and shifts it onto the stack
    • The reduce operation looks at N symbols on the stack and if they match the RHS of a grammar rule, reduces the stack by replacing those symbols with the nonterminal
• Also must either incorporate back-tracking or must keep multiple possible parses
Bottom-up Parsing Demo

• NLTK parsing demos
  – Bottom-up parsing using a shift-reduce algorithm
  • Instead of back-tracking or multiple parses, this NLTK implementation requires outside intervention to apply the correct rule when there is a choice

        nltk.app.srparsen()

• Described in NLTK book, Chapter 8, Analyzing Sentence Structure
Parsing issues

• Top-down
  – Only searches for trees that can be answers (i.e. S’s)
  – But also suggests trees that are not consistent with any of the words

• Bottom-up
  – Only forms trees consistent with the words
  – But suggest trees that make no sense globally

• Note that in the “book that flight” example, there was local ambiguity between “book” being a verb or a noun that was resolved at the end of the parse

• But examples with structural ambiguity will not be resolved, resulting in more than one possible derivation
Structural Ambiguity