Syntactic Parsing

• **Syntactic Parsing** = assigning a syntactic structure to a sentence.
  - For CFGs: assigning a *phrase-structure tree* to a sentence.

<table>
<thead>
<tr>
<th>Grammar</th>
<th>Lexicon</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow NP \ VP$</td>
<td>$Det \rightarrow that</td>
</tr>
<tr>
<td>$S \rightarrow Aux \ NP \ VP$</td>
<td>$Noun \rightarrow book</td>
</tr>
<tr>
<td>$S \rightarrow VP$</td>
<td>$Verb \rightarrow book</td>
</tr>
<tr>
<td>$NP \rightarrow Pronoun$</td>
<td>$Pronoun \rightarrow I</td>
</tr>
<tr>
<td>$NP \rightarrow Proper-Noun$</td>
<td>$Proper-Noun \rightarrow Houston</td>
</tr>
<tr>
<td>$NP \rightarrow Det Nominal$</td>
<td>$Aux \rightarrow does$</td>
</tr>
<tr>
<td>Nominal $\rightarrow Noun$</td>
<td>$Preposition \rightarrow from</td>
</tr>
<tr>
<td>Nominal $\rightarrow Nominal Noun$</td>
<td>$Nominal \rightarrow Nominal PP$</td>
</tr>
<tr>
<td>Nominal $\rightarrow Nominal PP$</td>
<td>$VP \rightarrow Verb$</td>
</tr>
<tr>
<td>$VP \rightarrow Verb NP$</td>
<td>$VP \rightarrow Verb NP PP$</td>
</tr>
<tr>
<td>$VP \rightarrow Verb PP$</td>
<td>$VP \rightarrow VP PP$</td>
</tr>
<tr>
<td>$PP \rightarrow Preposition NP$</td>
<td>$Book$</td>
</tr>
</tbody>
</table>

Lecture 04

*Book that flight.*
Syntactic Parsing as Search

• Parsing $\equiv$ search through the space of all possible parse trees such that:
  1. The leaves of the final parse tree coincide with the words in the input sentence.
  2. The root of the parse tree is the symbol S, i.e. complete parse tree.

$\Rightarrow$ 2 search strategies:
  – **Top-Down** parsing (goal-directed search).
  – **Bottom-Up** parsing (data-directed search).
Top-Down Parsing

- Build the parse tree from the root $S$ down to the leaves:
  - Expand tree nodes $N$ by using CFG rules $N \rightarrow N_1 \ldots N_k$.
  - Grow trees downward until reaching the POS categories at the bottom of the tree.
  - Reject trees that do not match all the words in the input.
Bottom-Up Parsing

• Build the parse tree from the leaf words up to the root S:
  – Find root nodes $N_1 \ldots N_k$ in the current forest such that they match a CFG rule $N \rightarrow N_1 \ldots N_k$.
  – Reject sub-trees that cannot lead to the start symbol S.
Top-Down vs. Bottom-Up

• Top-down:
  – Only searches for trees that are complete (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 also suggests trees that make no sense globally.

• How expensive is the entire search process?
Syntactic Parsing as Search

• How to keep track of the search space and how to make choices:
  – Which node to try to expand next.
  – Which grammar rule to use to expand a node.

• Backtracking (naïve implementation of parsing):
  – Expand the search space incrementally, choose a state to expand in the search space (depth-first, breadth-first, or other strategies).
  – If strategy arrives at an inconsistent tree, backtrack to an unexplored search on the agenda.
  – Doomed because of large search space and redundant work due to shared subproblems.
Large Search Space

• Global Ambiguity:
  – coordination: old men and women
  – attachment: we saw the Eiffel Tower flying to Paris

• Local Ambiguity

Lecture 04
Shared Subproblems

• Parse the sentence:
  “a flight from Indianapolis to Houston on NWA”

• Use backtracking with a top-down, depth-first, left-to-right strategy:
  – Assume a top-down parse making choices among the various Nominal rules, in particular, between these two:
    • Nominal → Noun
    • Nominal → Nominal PP
  – Statically choosing the rules in this order leads to the following bad results, in which every part of the final tree is derived more than once:

Lecture 04
Shared Subproblems
Syntactic Parsing using Dynamic Programming

- Shared subproblems $\implies$ **dynamic programming** could help.

- Dynamic Programming:
  - **CKY** algorithm (bottom-up search).
    - Need to transform the CFG into Chomsky Normal Form (CNF).
    - Any CFG can be transformed into CNF automatically.
  - **Earley** algorithm (top-down search).
    - does not require a normalized grammar.
    - a single left-to-right pass that fills an array/chart of size $n + 1$.
    - more complex than CKY.
  - **Chart parsing**:
    - more general, retain completed phrases in a chart, can combine top-down and bottom-up search.
CKY Parsing: Chomsky Normal Form

- All rules should be of one of two forms:
  \[ A \rightarrow B \; C \text{ or } A \rightarrow w \]

- CNF conversion procedure:
  1. Convert terminals to dummy non-terminals:
     \[ \text{INF-VP} \rightarrow \text{to VP} \iff \text{INF-VP} \rightarrow \text{TO VP} \text{ and } \text{TO} \rightarrow \text{to} \]
  2. Convert unit productions
     \[ \text{Nominal} \rightarrow \text{Noun} \]
     \[ \text{Noun} \rightarrow \text{book} | \text{flight} \]
  3. Make all rules binary by adding new non-terminals:
     \[ \text{VP} \rightarrow \text{Verb NP PP} \iff \text{VP} \rightarrow \text{VX PP} \]
     \[ \text{VX} \rightarrow \text{Verb NP} \]
# $L_1$ Grammar

<table>
<thead>
<tr>
<th>Grammar</th>
<th>Lexicon</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow NP \ VP$</td>
<td>$Det \rightarrow that \mid this \mid a$</td>
</tr>
<tr>
<td>$S \rightarrow Aux \ NP \ VP$</td>
<td>$Noun \rightarrow book \mid flight \mid meal \mid money$</td>
</tr>
<tr>
<td>$S \rightarrow VP$</td>
<td>$Verb \rightarrow book \mid include \mid prefer$</td>
</tr>
<tr>
<td>$NP \rightarrow Pronoun$</td>
<td>$Pronoun \rightarrow I \mid she \mid me$</td>
</tr>
<tr>
<td>$NP \rightarrow Proper-Noun$</td>
<td>$Proper-Noun \rightarrow Houston \mid NWA$</td>
</tr>
<tr>
<td>$NP \rightarrow Det Nominal$</td>
<td>$Aux \rightarrow does$</td>
</tr>
<tr>
<td>Nominal $\rightarrow Noun$</td>
<td>$Preposition \rightarrow from \mid to \mid on \mid near \mid through$</td>
</tr>
<tr>
<td>Nominal $\rightarrow Nominal Noun$</td>
<td></td>
</tr>
<tr>
<td>Nominal $\rightarrow Nominal PP$</td>
<td></td>
</tr>
<tr>
<td>$VP \rightarrow Verb$</td>
<td></td>
</tr>
<tr>
<td>$VP \rightarrow Verb NP$</td>
<td></td>
</tr>
<tr>
<td>$VP \rightarrow Verb NP PP$</td>
<td></td>
</tr>
<tr>
<td>$VP \rightarrow Verb PP$</td>
<td></td>
</tr>
<tr>
<td>$VP \rightarrow VP PP$</td>
<td></td>
</tr>
<tr>
<td>$PP \rightarrow Preposition NP$</td>
<td></td>
</tr>
<tr>
<td><strong>$L_1$ Grammar</strong></td>
<td><strong>$L_1$ in CNF</strong></td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>$S \rightarrow NP\ VP$</td>
<td>$S \rightarrow NP\ VP$</td>
</tr>
<tr>
<td>$S \rightarrow Aux\ NP\ VP$</td>
<td>$S \rightarrow X1\ VP$</td>
</tr>
<tr>
<td></td>
<td>$XI \rightarrow Aux\ NP$</td>
</tr>
<tr>
<td>$S \rightarrow VP$</td>
<td>$S \rightarrow book\</td>
</tr>
<tr>
<td></td>
<td>$S \rightarrow Verb\ NP$</td>
</tr>
<tr>
<td></td>
<td>$S \rightarrow X2\ PP$</td>
</tr>
<tr>
<td></td>
<td>$S \rightarrow Verb\ PP$</td>
</tr>
<tr>
<td></td>
<td>$S \rightarrow VP\ PP$</td>
</tr>
<tr>
<td>$NP \rightarrow Pronoun$</td>
<td>$NP \rightarrow I\</td>
</tr>
<tr>
<td>$NP \rightarrow Proper-Noun$</td>
<td>$NP \rightarrow TWA\</td>
</tr>
<tr>
<td>$NP \rightarrow Det\ Nominal$</td>
<td>$NP \rightarrow Det\ Nominal$</td>
</tr>
<tr>
<td>$Nominal \rightarrow Noun$</td>
<td>$Nominal \rightarrow book\</td>
</tr>
<tr>
<td>$Nominal \rightarrow Nominal\ Noun$</td>
<td>$Nominal \rightarrow Nominal\ Noun$</td>
</tr>
<tr>
<td>$Nominal \rightarrow Nominal\ PP$</td>
<td>$Nominal \rightarrow Nominal\ PP$</td>
</tr>
<tr>
<td>$VP \rightarrow Verb$</td>
<td>$VP \rightarrow book\</td>
</tr>
<tr>
<td>$VP \rightarrow Verb\ NP$</td>
<td>$VP \rightarrow Verb\ NP$</td>
</tr>
<tr>
<td>$VP \rightarrow Verb\ NP\ PP$</td>
<td>$VP \rightarrow X2\ PP$</td>
</tr>
<tr>
<td></td>
<td>$X2 \rightarrow Verb\ NP$</td>
</tr>
<tr>
<td>$VP \rightarrow Verb\ PP$</td>
<td>$VP \rightarrow Verb\ PP$</td>
</tr>
<tr>
<td>$VP \rightarrow VP\ PP$</td>
<td>$VP \rightarrow VP\ PP$</td>
</tr>
<tr>
<td>$PP \rightarrow Preposition\ NP$</td>
<td>$PP \rightarrow Preposition\ NP$</td>
</tr>
</tbody>
</table>
CKY Parsing: Dynamic Programming

- Use indeces to point at gaps between words:
  \[ \text{Book\ 1\ the\ 2\ flight\ 3\ through\ 4\ Houston\ 5} \]

- A sentence with \( n \) words \( \Rightarrow \) \( n + 1 \) positions.
  - \( \text{words}[1] = \text{“book”}, \text{words}[2] = \text{“the”}, \ldots \)

- Define a \((n + 1) \times (n + 1)\) matrix \( T \):
  - \( T[i,j] \) = the set of non-terminals that can generate the sequence of words between gaps \( i \) and \( j \).
  - \( T[0,n] \) contains \( S \Leftrightarrow \) the sentence can be generated by the CFG.

- How can we compute \( T[i,j] \)?
  - Only interested in the upper-triangular portion (i.e. \( i < j \)).
CKY: Dynamic Programming

• Recursively define the table values:
  1. \( A \in T[i-1,i] \) if and only if there is a rule \( A \rightarrow \text{words}[i] \).
  2. \( A \in T[i,j] \) if and only if \( \exists k, i < k < j \), such that:
     • \( B \in T[i,k] \) and \( C \in T[k,j] \).
     • There is a rule \( A \rightarrow B \ C \) in the CFG.

• Bottom-up computation:
  - In order to compute the set \( T[i,j] \), the sets \( T[i,k] \) and \( T[k,j] \) need to have been computed already, for all \( i < k < j \).
  \( \Rightarrow \) (at least) two possible orderings:
     • which one is more “natural”?
CKY: Bottom-Up Computation

Lecture 04
CKY Parsing

- Fill the table a column at a time, left to right, bottom to top.

```python
function CKY-PARSE(words, grammar) returns table

for j ← from 1 to LENGTH(words) do
    table[j − 1, j] ← \{A \mid A \rightarrow words[j] \in grammar\}

for i ← from j − 2 downto 0 do
    for k ← i + 1 to j − 1 do
        table[i, j] ← table[i, j] ∪
        \{A \mid A \rightarrow BC \in grammar,
         B \in table[i, k],
         C \in table[k, j]\}
```
CKY Parsing: Example

<table>
<thead>
<tr>
<th></th>
<th>Book</th>
<th>the</th>
<th>flight</th>
<th>through</th>
<th>Houston</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0,1]</td>
<td>S, VP, Verb Nominal, Noun</td>
<td>[0,2]</td>
<td>S, VP, X2</td>
<td>[0,3]</td>
<td>S, VP, X2</td>
</tr>
<tr>
<td>[1,2]</td>
<td>Det</td>
<td>[1,3]</td>
<td>NP</td>
<td>[1,4]</td>
<td>NP</td>
</tr>
<tr>
<td>[4,5]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
S → NP VP
S → X1 VP
X1 → Aux NP
S → book | include | prefer
S → Verb NP
S → X2 NP
X2 → Verb NP
S → VP PP
NP → I | he | she | me
NP → Houston | NWA
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → VP PP
VP → X2 PP
PP → Prep NP
Lecture 04

S → NP VP
S → X1 VP
X1 → Aux NP
S → book | include | prefer
S → Verb NP
S → X2 NP
X2 → Verb NP
S → VP PP
NP → I | he | she | me
NP → Houston | NWA
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → VP PP
VP → X2 PP
PP → Prep NP
Lecture 04

S → NP VP
S → X1 VP
X1 → Aux NP
S → book | include | prefer
S → Verb NP
S → X2 NP
X2 → Verb NP
S → VP PP
NP → I | he | she | me
NP → Houston | NWA
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → VP PP
VP → X2 PP
PP → Prep NP
S → NP VP
S → X1 VP
X1 → Aux NP
S → book | include | prefer
S → Verb NP
S → X2 NP
X2 → Verb NP
S → VP PP
NP → I | he | she | me
NP → Houston | NWA
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → VP PP
VP → X2 PP
PP → Prep NP
Lecture 04

S → NP VP
S → X1 VP
X1 → Aux NP
S → book | include | prefer
S → Verb NP
S → X2 NP
X2 → Verb NP
S → VP PP
NP → I | he | she | me
NP → Houston | NWA
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → VP PP
VP → X2 PP
PP → Prep NP
CKY Parsing

- How do we change the algorithm to output the parse trees?
- Time complexity:
  - for computing the table?
  - for computing all parses?

```latex
\begin{verbatim}
function CKY-PARSE(words, grammar) returns table

    for j ← from 1 to LENGTH(words) do
        table[j - 1, j] ← \{ A \mid A \to \text{words}[j] \in grammar \}
    for i ← from j - 2 downto 0 do
        for k ← i + 1 to j - 1 do
            table[i, j] ← table[i, j] \cup
                \{ A \mid A \to BC \in grammar,
                \quad B \in table[i, k],
                \quad C \in table[k, j] \}\n\end{verbatim}
```
CKY Parsing

- The parse trees correspond to the CNF grammar, not the original CFG:
  \[
  \Rightarrow \text{complicates subsequent syntax-direct semantic analysis.}
  \]

- Post-processing of the parse tree:
  - For binary productions:
    - delete the new dummy non-terminals and promote their daughters to restore the original tree.
  - For unit productions:
    - alter the basic CKY algorithm to handle them directly.
    - homework exercise 13.3
CKY Parsing

• Does CKY solve ambiguity?
  – Book the flight through Houston.

  Use probabilistic CKY parsing, output highest probability tree.

• Will probabilistic CKY solve all ambiguity?
  – One morning I shot an elephant in my pajamas.
  – How he got into my pajamas I don’t know.
Shallow Parsing: Chunking

- **Chunking** = find all non-recursive major types of phrases:
  - \([_{NP} \text{The morning flight}] \ [_{PP} \text{from}] \ [_{NP} \text{Denver}] \ [_{VP} \text{has arrived}]\)
  - \([_{NP} \text{The morning flight}] \text{ from } [_{NP} \text{Denver}] \text{ has arrived}\)

- Chunking can be approached as **Sequence Labeling**.

- Evaluation:

  \[
  \text{Precision (P)} = \frac{\text{# correct chunks found}}{\text{total \# chunks found}}
  \]

  \[
  \text{Recall (R)} = \frac{\text{# correct chunks found}}{\text{total \# actual chunks}}
  \]

  \[
  F = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}
  \]

  \[
  F_1 = \frac{2PR}{P + R}
  \]

Currently, best NP chunking system obtains \(F_1=96\%\).