Lecture Plan

• Parsing as Logical Deduction
• Defining the CFG recognition problem
• Bottom up vs. top down
• Quick review of Chomsky normal form
• The CKY algorithm
  – CKY as dynamic programming
  – Complexity analysis
Algorithms for CFGs

Given a CFG G and a string s:

- **Recognition**: Is \( s \in L(G) \)?
  - Equivalently, find *some* derivation that *proves* \( s \) is in G's language.

- **Parsing**:
  - What are (all of) G's derivations of \( s \)?
  - What is the “correct” derivation of \( s \) under G?

The same core algorithms actually provide solutions to all of these!
Important Distinction

• Deterministic grammars give much faster recognition and parsing algorithms!
  – LR languages are parseable in linear time; you may have learned about these in a course on programming languages or compilers.
  – For us, the best known algorithms are $O(|N| n^{2.376})$, but these are not practical.
  – Today you will learn an $O(|N|^3 n^3)$ solution.
Parsing Strategies
Parsing as Search

Trees break into pieces (partial trees), which can be used to define a search space.
Top-Down Parsing (Recursive Descent)

\[ S \rightarrow \text{NP VP} \]
\[ S \rightarrow \text{Aux NP VP} \]
\[ S \rightarrow \text{VP} \]
\[ \text{NP} \rightarrow \text{Pronoun} \]
\[ \text{NP} \rightarrow \text{Proper-Noun} \]
\[ \text{NP} \rightarrow \text{Det Nominal} \]
\[ \text{Nominal} \rightarrow \text{Noun} \]
\[ \text{Nominal} \rightarrow \text{Nominal Noun} \]
\[ \text{Nominal} \rightarrow \text{Nominal PP} \]
\[ \text{VP} \rightarrow \text{Verb} \]
\[ \text{VP} \rightarrow \text{Verb NP} \]
\[ \text{VP} \rightarrow \text{Verb NP PP} \]
\[ \text{VP} \rightarrow \text{Verb PP} \]
\[ \text{VP} \rightarrow \text{VP PP} \]
\[ \text{PP} \rightarrow \text{Preposition NP} \]

\[ \text{Det} \rightarrow \text{that} \mid \text{this} \mid \text{a} \]
\[ \text{Noun} \rightarrow \text{book} \mid \text{flight} \mid \text{meal} \mid \text{money} \]
\[ \text{Verb} \rightarrow \text{book} \mid \text{include} \mid \text{prefer} \]
\[ \text{Pronoun} \rightarrow \text{I} \mid \text{she} \mid \text{me} \]
\[ \text{Proper-Noun} \rightarrow \text{Houston} \mid \text{NWA} \]
\[ \text{Aux} \rightarrow \text{does} \]
\[ \text{Preposition} \rightarrow \text{from} \mid \text{to} \mid \text{on} \mid \text{near} \mid \text{through} \]

\( s = \text{“Book that flight”} \)
Top-Down Parsing (Recursive Descent)

\[(S)\]

\[(S (NP) (VP))\]  \[(S \text{Aux} (NP) (VP))\]  \[(S (VP))\]

\(x = \text{“Book that flight”}\)

---

**Figure 13.1** The L1 miniature English grammar and lexicon.
Top-Down Parsing (Recursive Descent)

\[
(S) \quad (S \text{ Aux} \text{ NP} \text{ VP}) \quad (S \text{ VP}) \quad (S \text{ NP Pronoun} \text{ VP}) \quad (S \text{ NP ProperNoun} \text{ VP}) \quad (S \text{ NP Det Nominal} \text{ VP})
\]

\[
\begin{array}{|c|c|c|}
\hline
\text{Grammar} & \text{Lexicon} \\
\hline
S \rightarrow \text{ NP VP} & \text{Det} \rightarrow \text{ that} | \text{ this} | \text{ a} \\
S \rightarrow \text{ Aux NP VP} & \text{Noun} \rightarrow \text{ book} | \text{ flight} | \text{ meal} | \text{ money} \\
S \rightarrow \text{ VP} & \text{Verb} \rightarrow \text{ book} | \text{ include} | \text{ prefer} \\
\text{NP} \rightarrow \text{ Pronoun} & \text{Pronoun} \rightarrow \text{ I} | \text{ she} | \text{ me} \\
\text{NP} \rightarrow \text{ Proper-Noun} & \text{Proper-Noun} \rightarrow \text{ Houston} | \text{ NWA} \\
\text{NP} \rightarrow \text{ Det Nominal} & \text{Aux} \rightarrow \text{ does} \\
\text{Nominal} \rightarrow \text{ Noun} & \text{Preposition} \rightarrow \text{ from} | \text{ to} | \text{ on} | \text{ near} | \text{ through} \\
\text{Nominal} \rightarrow \text{ Nominal Noun} & \\
\text{Nominal} \rightarrow \text{ Nominal PP} & \\
\text{VP} \rightarrow \text{ Verb} & \\
\text{VP} \rightarrow \text{ Verb NP} & \\
\text{VP} \rightarrow \text{ Verb NP PP} & \\
\text{VP} \rightarrow \text{ Verb PP} & \\
\text{VP} \rightarrow \text{ VP PP} & \\
\text{PP} \rightarrow \text{ Preposition NP} & \\
\hline
\end{array}
\]

\[x = \text{“Book that flight”}\]
Top-Down Parsing (Recursive Descent)

\( x = \text{“Book that flight”} \)
Top-Down Parsing (Recursive Descent)

(S)

(S (NP) (VP)) (S Aux (NP) (VP)) (S (VP))

(S (NP Pronoun) (VP)) (S (NP ProperNoun) (VP))

(S (NP Det Nominal) (VP))

(S Aux (NP Pronoun) (VP)) (S Aux (NP ProperNoun) (VP))

(S Aux (NP Det Nominal) (VP))

(S (VP (VP) (PP))) (S (VP Verb)) (S (VP Verb (NP)))

(S (VP Verb (NP) (PP))) (S (VP Verb (PP)))

x = “Book that flight”
Top-Down Parsing (Recursive Descent)

• Never wastes time exploring ungrammatical trees!
• Inefficiency: most search states (partial trees) could never lead to a derivation of our sentence.
• One other problem ...
Left Recursion!

\[(S)\]
\[(S (VP))\]
\[(S (VP (VP) (PP)))\]
\[(S (VP (VP (VP) (PP)) (PP)))\]
\[(S (VP (VP (VP (VP) (PP)) (PP)) (PP)))\]
\[(S (VP (VP (VP (VP (VP) (PP)) (PP)) (PP)) (PP)))\]
\[(S (VP (VP (VP (VP (VP (VP) (PP)) (PP)) (PP)) (PP)) (PP)))\]
\[(S (VP (VP (VP (VP (VP (VP (VP) (PP)) (PP)) (PP)) (PP)) (PP)) (PP)))\]

\[x = "Book that flight"\]
Top-Down Recognition

- Don't need to store the tree!
- Can collapse states that function the same way.
- Store unexpanded nonterminals (in sequence) only, along with the number of words “covered” so far.

Start state: \((S, 0)\)

**Scan:** If you're at \((x_{j+1} \beta, j)\), you can get to \((\beta, j + 1)\).

**Predict:** If you're at \((Z\beta, j)\) and \(Z \rightarrow \gamma\), you can get to \((\gamma\beta, j)\).

Final (“success”) state: \((\varepsilon, n)\)

- This should remind you of *generating* from a CFG.
Bottom-Up Parsing

book that flight
Bottom-Up Parsing

(Verb book) (Det that) (Noun flight)

(Noun book) (Det that) (Noun flight)
Bottom-Up Parsing

(Nominal (Noun book)) (Det that) (Nominal (Noun flight))

(Verb book) (Det that) (Noun flight)

(Noun book) (Det that) (Noun flight)

book that flight
Bottom-Up Parsing

(Verb book) (Det that) (Nominal (Noun flight))

(Nominal (Noun book)) (Det that) (Nominal (Noun flight))

(Verb book) (Det that) (Noun flight)

(Noun book) (Det that) (Noun flight)

book that flight
Bottom-Up Parsing

(Nominal (Noun book)) (NP (Det that) (Nominal (Noun flight)))

(Verb book) (Det that) (Nominal (Noun flight))

(Nominal (Noun book)) (Det that) (Nominal (Noun flight))

(Verb book) (Det that) (Noun flight)

(Noun book) (Det that) (Noun flight)

book that flight
Bottom-Up Parsing

• Never generates trees that are inconsistent with the sentence.
• Generates partial trees that have no hope of getting to S.
Shift-Reduce (A Bottom-Up Method)

- Don't need to store the tree!
- Can collapse states that function the same way.
- Store only the nodes that are parentless.

Start state: \((\varepsilon, 0)\)

**Shift:** If you're at \((\alpha, j)\), you can get to \((\alpha x_{j+1}, j + 1)\).

**Reduce:** If you're at \((\alpha \gamma, j)\) and \(Z \rightarrow \gamma\), you can get to \((\alpha Z, j)\).

Final ("success") state: \((S, n)\)
Implementing Recognizers as Search

Agenda = \{ \text{state}_0 \} \\
\textbf{while}(\text{Agenda not empty}) \\
\quad s = \textbf{pop} \text{ a state from Agenda} \\
\quad \textbf{if} \ s \text{ is a success-state} \ \textbf{return} \ s \ // \ \text{valid parse tree} \\
\quad \textbf{else if} \ s \text{ is not a failure-state:} \\
\quad \quad \text{generate new states from } s \\
\quad \quad \textbf{push} \text{ new states onto Agenda} \\
\textbf{return} \ \text{nil} \ // \ \text{no parse}!
**Parsing as Search**

- Basically, you need a strategy for deciding what to pursue next (how to **prioritize** the agenda).
  - Examples: depth-first, breadth-first, bottom-up, top-down, left-to-right, right-to-left, ...

- **The problems:**
  - Memory!
  - Ambiguity leads to lots of bad paths

- In a depth-first model, this results in lots of wasted effort in **backtracking**.

- One solution: **greedy** parsing. Only works if you have a really smart agenda!
a flight

from Indianapolis...

to Houston...

from Indianapolis

on TWA
to Houston

from Indianapolis

flight
Ambiguity Redux

• A sentence may have many parses.
• Even if a sentence has only one parse, finding it may be difficult, because there are many misleading paths you could follow.
  – Bottom-up: fragments that can never have a home in any S
  – Top-down: fragments that never get you to x
• What to do when there are many parses ... how to choose? Return them all?
So Far ...

• Two ways to think of parsing as search
• Problems:
  – Left-recursion (in top-down parsers)
  – Efficiency (backtracking, memory)
  – Ambiguity
CKY

- Kasami, 1965
- Younger, 1967
- Cocke and Schwartz, 1970
Chomsky Normal Form

• $G = (\Sigma, N, S, R)$
• $\Sigma$: Vocabulary of terminal symbols
• $N$: set of nonterminal symbols
• $S \in N$: special start symbol
• $R$: Production rules of the form $X \rightarrow \alpha$ where $X \in N$ (a nonterminal symbol) and $\alpha \in N^2 \cup \Sigma$
CKY Schema

- Data structure: chart.
- In each cell, store the possible phrase types.
- Visit each cell once.
  - Start with narrow spans, progress to longer ones.
Figure 13.8 \( \mathcal{L}_1 \) Grammar and its conversion to CNF. Note that although they aren’t shown here all the original lexical entries from \( \mathcal{L}_1 \) carry over unchanged as well.
CKY Schema

<table>
<thead>
<tr>
<th>$\mathcal{L}_1$ Grammar</th>
<th>$\mathcal{L}_1$ in CNF</th>
</tr>
</thead>
<tbody>
<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>$S \rightarrow VP$</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 Nominal$</td>
<td>$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$</td>
</tr>
<tr>
<td>$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>

Figure 13.6 $\mathcal{L}_1$ Grammar and its conversion to CNF. Note that although they aren't shown here all the original lexical entries from $\mathcal{L}_1$ carry over unchanged as well.
CKY Algorithm (Recognition)

**input:**  \( x, G \) (in CNF)

**output:**  true iff \( x \) is in \( L(G) \)

initialize all cells of \( C \) to \( \emptyset \)

for \( i = 1 \ldots n \)
  \( C[i-1, i] = \{A \mid A \rightarrow x_i\} \)

for \( \ell = 2 \ldots n \)
  for \( i = 0 \ldots n - \ell \)
    \( k = i + \ell \)
    for \( j = i + 1 \ldots k - 1 \)
      for each \( X \rightarrow YZ \) s.t. \( Y \in C[i, j] \) & \( Z \in C[j, k] \)
        \( C[i, k] = C[i, k] \cup \{X\} \)

return true if \( S \in C[0, n] \)
Visualizing CKY
How do we fill in $C(0,2)$?
How do we fill in $C(0, 2)$?

Put together $C(0, 1)$ and $C(1, 2)$. 
How do we fill in $C(0,3)$?
How do we fill in \( C(0,3) \)?

One way …
Visualizing CKY

How do we fill in $C(0,3)$?

One way …
Another way.
Visualizing CKY

How do we fill in $C(0,n)$?
Visualizing CKY

How do we fill in $C(0,n)$?

$n - 1$ ways!
Ordering in CKY
CKY Algorithm for CF Recognition

input: \( x, G \) (in CNF)
output: true iff \( x \) is in \( L(G) \)
initialize all cells of \( C \) to \( \emptyset \)
for \( i = 1 \ldots n \)
    \( C[i-1, i] = \{ A \mid A \rightarrow x_i \} \)
for \( \ell = 2 \ldots n \)
    for \( i = 0 \ldots n - \ell \)
        \( k = i + \ell \)
        for \( j = i + 1 \ldots k - 1 \)
            for each \( X \rightarrow YZ \) s.t. \( Y \in C[i, j] \) & \( Z \in C[j, k] \)
                \( C[i, k] = C[i, k] \cup \{ X \} \)
return true if \( S \in C[0, n] \)
CKY Algorithm

**input:** \( x, G \) (in CNF)

**output:** true iff \( x \) is in \( L(G) \)

initialize all cells of \( C \) to \( \emptyset \)

for \( i = 1 \ldots n \)

  for each \( A \) s.t. \( A \rightarrow x_i \)
    \( C[A, i-1, i] = 1 \)

for \( \ell = 2 \ldots n \)

  for \( i = 0 \ldots n - \ell \)
    \( k = i + \ell \)
    for \( j = i + 1 \ldots k - 1 \)
      for each \( X \rightarrow YZ \)
        \( C[X, i, k] = C[X, i, k] \lor (C[Y, i, j] \land C[Z, j, k]) \)

return \( C[S, 0, n] \)
CKY Operations

<table>
<thead>
<tr>
<th>given ...</th>
<th>for every</th>
<th>add</th>
<th>to</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>A → x_i</td>
<td>A</td>
<td>C[i - 1, i]</td>
</tr>
<tr>
<td>Y ∈ C[i, j]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z ∈ C[j, k]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X → Y Z</td>
<td>X</td>
<td>C[i, k]</td>
</tr>
</tbody>
</table>
Visualizing CKY Operations

B → x_i

size of N

nonterminal

length of sentence

length of sentence

O(n)

O(n^3)

X → Y Z

k j

j i

start

end
CKY Algorithm Asymptotics

• Space: $O(n^2)$
• Runtime: $O(n^3)$
• Both also depend on *grammar size*
Parsing as Logical Deduction
(Weighted) Logic Programming

- Parsing algorithms can be expressed as (weighted) logic programs

\[
\frac{I_1 \quad I_2 \quad \cdots \quad I_k}{I} \quad \phi
\]

(Weighted) Logic Programming

• Parsing algorithms can be expressed as (weighted) logic programs

\[
\begin{array}{cccc}
I_1 & I_2 & \cdots & I_k \\
\hline
I & & & \\
\end{array}
\phi
\]

\[
\begin{array}{cccc}
I_1 : w_1 & I_2 : w_2 & \cdots & I_k : w_k \\
\hline
I : w \otimes \bigotimes_{i=1}^{k} w_i & \phi
\end{array}
\]

Understanding Logic Programs

$\Phi$
Understanding Logic Programs

Items

\[ \square \]

Axioms

\[ A \subseteq \square \]

Inference rules

\[ \frac{I_1 \quad I_2 \quad \cdots \quad I_k}{I} \]

Goals

\[ G \subseteq \square \]
CKY Algorithm

Item form
\[ [N, N_1, N_1] \]
CKY Algorithm

**Item form**
\[ N, N_1, N_1 \]

**Goals**
\[ [S, 1, |x| + 1] \]
CKY Algorithm

**Item form**
\[ \mathcal{N}, \mathcal{N}_1, \mathcal{N}_1 \]

**Goals**
\[ [S, 1, |x| + 1] \]

**Axioms**
\[ [X, i, i + 1] \]

\((X \rightarrow x_i) \in \mathcal{N}\)
CKY Algorithm

**Item form**
\[ \mathbb{N}, \mathbb{N}_1, \mathbb{N}_1 \]

**Goals**
\[ [S, 1, |x| + 1] \]

**Axioms**
\[ [X, i, i + 1] \]

\[ (X \rightarrow x_i) \in \mathbb{N} \]

**Inference rules**
\[ [X, i, k] \quad [Y, k, j] \]
\[ [Z, i, j] \]

\[ (Z \rightarrow X Y) \in \mathbb{R} \]
Top-Down Parsing

Item form
\[(N \cup \Sigma)^*, N\]

Goals
\[[\varepsilon, |x|]\]

Axioms
\[[S, 0]\]

Inference rules
\[
\begin{align*}
[x_{j+1}\beta, j] & \quad \frac{[\beta, j + 1]}{[\beta, j + 1]} \\
[Z\beta, j] & \quad \frac{[\gamma\beta, j]}{(Z \rightarrow \gamma) \in R}
\end{align*}
\]

“Scan”

“Predict”
CKY Algorithm: Discussion

- Top-down or bottom-up?
- Parsing as deduction: visual notation \(\approx\) Horn clauses, used to “prove” \(x \in L(G)\).
Dynamic Programming

• Examples: WFSA best path, Viterbi, Earley, CKY

• “Solve sub-problems, store results in a table.”
  – Reuse work you’ve done already (runtime)
  – Pack sub-problems efficiently (memory)

• For parsing, many sub-parses can be used repeatedly in many larger sub-parses! Don’t build the same sub-parse more than once.
DP Execution Model

• Clever ordering to ensure items are built exactly once
  – Viterbi: “left to right”
  – CKY: “narrow to wide”

• Memoization

• Agenda

DP logic ⊥ execution model
Implementing Recognizers as Search

Agenda = \{ \text{state}_0 \}\}

\textbf{while}(Agenda not empty)

\hspace{1em} s = \textbf{pop} a state from Agenda

\hspace{1em} \textbf{if} s is a success-state \textbf{return} s // valid parse
tree

\hspace{1em} \textbf{else if} s is not a failure-state:

\hspace{2em} generate new states from s

\hspace{2em} push new states onto Agenda

\textbf{return} nil // no parse!
CKY as Search

Agenda = [{ [A, i − 1, i] | A → x_i}]

while(Agenda not empty)
    [N, i, j] = pop from Agenda
    if [N, i, j] = [S, 0, n] return true
else:
    for each visited [Y, j, k] and X → N Y
        push [X, i, k] onto Agenda
    for each visited [Z, k, i] and X → Z N
        push [X, k, i] onto Agenda
return false
What Else?

• So far, this only does recognition.
• Can we recover parse trees? (Need to store some more information.)
  – All trees
  – “Best” tree