Chomsky Normal Form

- wait! how can you assume a CFG is binary-branching?
- well, we can always convert a CFG into Chomsky-Normal Form (CNF)
  - $A \rightarrow B C$
  - $A \rightarrow a$
- how to deal with epsilon-removal?
- how to do it with PCFG?
Any variants of CKY?

- bottom-up
- left-to-right

\[ O(n^3 |P|) \]

For each diff (<= n)
  - For each i (<= n)
    - For each rule \( X \rightarrow Y Z \)
      - For each split point \( k \)
        - \( \text{score}[X][i][j] = \max \)
Any variants of CKY?

- For each diff ($\leq n$)
  - For each $i$ ($\leq n$)
    - For each rule $X \rightarrow Y Z$
      - For each split point $k$
        $\text{score}[X][i][j] = \max \text{score}[X][i][j]$, 
        $\text{score}(X \rightarrow Y Z) * \text{score}[Y][i][k] * \text{score}[Z][k][j]$

what’s the difference with shift-reduce?

(S, 0, n)

bottom-up

(S, 0, n)

left-to-right

(S, 0, n)

right-to-left

$O(n^3|P|)$
Parsing as Deduction

\[(B, i, k) : a \quad (C, k, j) : b\]

\[\frac{(A, i, j) : a \times b \times \text{Pr}(A \rightarrow B C)}{A \rightarrow B C}\]
Parsing as Intersection

\[(B, i, k) : a \quad (C, k, j) : b\]

\[\begin{array}{c}
(A, i, j) : a \times b \times \Pr(A \rightarrow B \ C)
\end{array}\]

- intersection between a CFG \( G \) and an FSA \( D \):
  - define \( L(G) \) to be the set of \textit{strings} (i.e., yields) \( G \) generates
  - define \( L(G \cap D) = L(G) \cap L(D) \)
  - what does this new language generate??
  - what does the new grammar \textit{look like}?
- what about \( CFG \cap CFG \)?
Packed Forests

- a compact representation of many parses
- by sharing common sub-derivations
- polynomial-space encoding of exponentially large set

(Klein and Manning, 2001; Huang and Chiang, 2005)
Lattice vs. Forest
Forest and Deduction

\[(B, i, k) : a \quad \text{and} \quad (C, k, j) : b\]

\[\text{antecedents} \quad (B, i, k) \quad \text{and} \quad (A, i, j) \quad \text{and} \quad (C, k, j) \quad \text{consequent}\]

\[\text{tails} \quad (A, i, j) \quad \text{and} \quad (B, i, k) \quad \text{and} \quad (C, k, j) \quad \text{head} \quad (A, i, j) \quad \text{and} \quad (B, i, k) \quad \text{and} \quad (C, k, j)\]

\[f_e : a \times b \times \Pr(A \rightarrow B C)\]

\[(A, i, j) \rightarrow (B, i, k) \quad \text{and} \quad (C, k, j)\]

\[f_e : \text{fe}(a,b)\]

\[(Nederhof, 2003)\]
# Related Formalisms

<table>
<thead>
<tr>
<th>hypergraph</th>
<th>AND/OR graph</th>
<th>context-free grammar</th>
<th>deductive system</th>
</tr>
</thead>
<tbody>
<tr>
<td>vertex</td>
<td>OR-node</td>
<td>symbol</td>
<td>item</td>
</tr>
<tr>
<td>source-vertex</td>
<td>leaf OR-node</td>
<td>terminal</td>
<td>axiom</td>
</tr>
<tr>
<td>target-vertex</td>
<td>root OR-node</td>
<td>start symbol</td>
<td>goal item</td>
</tr>
<tr>
<td>hyperedge</td>
<td>AND-node</td>
<td>production</td>
<td>instantiated deduction</td>
</tr>
<tr>
<td>({u_1, u_2}, v, f)</td>
<td></td>
<td>(v \rightarrow u_1 u_2)</td>
<td>(\frac{u_1 : a \quad u_2 : b}{v : f(a, b)})</td>
</tr>
</tbody>
</table>

- **AND-node**: A node with two or more OR-nodes as its children.
- **OR-node**: A node with only one child.

## Context-Free Grammar

- Production: \(v \rightarrow u_1 u_2\)

## Deductive System

- Instantiated deduction: \(\frac{u_1 : a \quad u_2 : b}{v : f(a, b)}\)

**Diagram:***

- **AND-node**: A node with two or more OR-nodes as its children.
- **OR-node**: A node with only one child.

**Related Concepts:**

- **AND-node**
- **OR-node**
- **OR-nodes**
- **context-free grammar**
- **deductive system**
Earley Algorithm

- no binarization is needed (like CKY w/ dotted rules)
- left-to-right parsing with top-down filtering
- in CKY, some nodes will never be used (from the top)
- in Earley, try to build nodes when needed (from the left)

Predict:

\[
\frac{(A \rightarrow \alpha.B\beta, \ i, \ j)}{(B \rightarrow .\gamma, \ j, \ j)} \quad B \rightarrow \gamma \in P
\]

only try to build B when needed by A!

Scan:

\[
\frac{(A \rightarrow \alpha.a\beta, \ i, \ j)}{(A \rightarrow \alpha.a.\beta, \ i, \ j + 1)} \quad w_j = a
\]

Complete:

\[
\frac{(A \rightarrow \alpha.B\beta, \ i, \ j) \quad (B \rightarrow \gamma., \ j, \ k)}{(A \rightarrow \alpha B.\beta, \ i, \ k)}
\]
Probabilistic Earley

- probabilistic Earley due to Stolcke (1995)
- predicted item carries only the rule prob
- different items could predict the same predicted item!
- probabilities accumulated in the complete step

Predict:

\[
\frac{(A \rightarrow \alpha . B \beta, i, j) : p}{(B \rightarrow \cdot \gamma, j, j) : \Pr(B \rightarrow \gamma)}
\]

Scan:

\[
\frac{(A \rightarrow \alpha . a \beta, i, j) : p}{(A \rightarrow \alpha a . \beta, i, j + 1) : p}
\]

Complete:

\[
\frac{(A \rightarrow \alpha . B \beta, i, j) : p}{(B \rightarrow \cdot \gamma, j, k) : q}
\]

\[
(A \rightarrow \alpha B . \beta, i, k) : pq
\]
Implementation of Earley

- Earley implementation is much trickier than CKY
- still 3 loops, but in different order (like left-to-right cky)
  - outmost loop: right boundary \( j \) (left to right)
- order of actions is important: complete > predict > scan

```plaintext
for j = 0 to n
    // first do completion, from short to long
    for i = j-1 downto 0
        // look for candidate items for completion
        for each item \( x = (i, j, B \rightarrow \gamma) \) in span \([i, j]\)
            for k = i downto 0
                for each item \( y = (k, i, A \rightarrow \alpha . B \beta) \) in span \([k, i]\)
                    combine \( x \) and \( y \) to be \((k, j, A \rightarrow \alpha . B \beta)\) in span \([k, j]\)
    // then do predictions and scanning
    for i = 0 to j
        for each item \( (i, j, A \rightarrow \alpha . B \beta) \) do (repeated) prediction
        for each item \( (i, j, A \rightarrow \alpha . w_j \beta) \) do scanning
```