**Parsing**

**Recognition:** Given a (context-free) grammar $G$ and a string of words $w$, determine whether $w \in L(G)$.

** Parsing:** If $w \in L(G)$, produce the (tree) structure that is assigned by $G$ to $w$.

---

**An example grammar**

$D \rightarrow \text{the}$

$N \rightarrow \text{cat}$

$N \rightarrow \text{hat}$

$P \rightarrow \text{in}$

$NP \rightarrow D N$

$PP \rightarrow P NP$

$NP \rightarrow NP PP$

---

**Example sentences:**

- the cat in the hat
- the cat in the hat in the hat
A bottom-up recognition algorithm

Assumptions:

• The grammar is given in Chomsky Normal Form: each rule is either of the form $A \rightarrow BC$ (where $A$, $B$, $C$ are non-terminals) or of the form $A \rightarrow a$ (where $a$ is a terminal).

• The string to recognize is $w = w_1 \cdots w_n$.

• A set of indices $\{0, 1, \ldots, n\}$ is defined to point to positions between the input string’s words:

  0 the 1 cat 2 in 3 the 4 hat 5

---

The CYK algorithm

for $j := 1$ to $n$ do
  for all rules $A \rightarrow w_j$ do
    $\text{chart}[j-1,j] := \text{chart}[j-1,j] \cup \{A\}$
  for $i := j-2$ downto $0$ do
    for $k := i+1$ to $j-1$ do
      for all $B \in \text{chart}[i,k]$ do
        for all $C \in \text{chart}[k,j]$ do
          for all rules $A \rightarrow BC$ do
            $\text{chart}[i,j] := \text{chart}[i,j] \cup \{A\}$
      if $S \in \text{chart}[0,n]$ then accept else reject

---

The CYK algorithm

Bottom-up, chart-based recognition algorithm for grammars in CNF

To recognize a string of length $n$, uses a chart: a bi-dimensional matrix of size $n \times (n + 1)$

Invariant: a non-terminal $A$ is stored in the $[i,j]$ entry of the chart iff $A \Rightarrow w_{i+1} \cdots w_j$

Consequently, the chart is triangular. A word $w$ is recognized iff the start symbol $S$ is in the $[0,n]$ entry of the chart

The idea: build all constituents up to the $i$-th position before constructing the $i+1$ position; build smaller constituents before constructing larger ones

---

The CYK algorithm

Extensions:

• Parsing in addition to recognition

• Support for $\epsilon$-rules

• General context-free grammars (not just CNF)
### Parsing schemata

To provide a unified framework for discussing various parsing algorithms we use parsing schemata, which are generalized schemes for describing the principles behind specific parsing algorithms. This is a generalization of the parsing as deduction paradigm.

A parsing schema consists of four components:

- a set of items
- a set of axioms
- a set of deduction rules
- a set of goal items

---

### Parsing schema: CYK

Given a grammar \( G = (\Sigma, V, S, P) \) and a string \( w = w_1 \cdots w_n \):

**Items:** \([i, A, j] \) for \( A \in V \) and \( 0 \leq i, j \leq n \) (state that \( A \Rightarrow w_{i+1} \cdots w_j \))

**Axioms:** \([i, A, i+1] \) when \( A \rightarrow w_{i+1} \in P \)

**Goals:** \([0, S, n] \)

**Inference rules:**

\[
\begin{align*}
[i, B, j] & \quad [j, C, k] \\
\ldots & \\
[i, A, k] & \quad A \rightarrow B \ C
\end{align*}
\]

---

### CYK parsing schema: deduction example

\[
\begin{align*}
D & \rightarrow \text{the} \\
N & \rightarrow \text{cat} \\
N & \rightarrow \text{hat} \\
P & \rightarrow \text{in}
\end{align*}
\]

\[
\begin{align*}
NP & \rightarrow D \ N \\
PP & \rightarrow P \ N P \\
NP & \rightarrow NP \ PP
\end{align*}
\]

0 the 1 cat 2 in 3 the 4 hat 5

\[
\begin{align*}
[0, D, 1] & \quad [1, N, 2] & \quad [2, P, 3] & \quad [3, D, 4] & \quad [4, N, 5] \\
[0, NP, 2] & \quad [3, NP, 5] & \quad [2, PP, 5]
\end{align*}
\]

---

### Parsing: bottom-up schema (Shift–Reduce)

**Items:** \([\alpha \bullet, j] \) (state that \( \alpha w_{j+1} \cdots w_n \Rightarrow w_1 \cdots w_n \))

**Axioms:** \([\bullet, 0] \)

**Goals:** \([S \bullet, n] \)

**Inference rules:**

- **Shift**
  \[
  \begin{align*}
  \frac{[\alpha \bullet, j]}{[\alpha w_{j+1} \bullet, j + 1]}
  \end{align*}
  \]

- **Reduce**
  \[
  \begin{align*}
  \frac{[\alpha \gamma \bullet, j]}{[\alpha B \bullet, j]} & \quad B \rightarrow \gamma
  \end{align*}
  \]
Bottom-up deduction: example

Input: 0 the 1 cat 2 in 3 the 4 hat 5

Parsing: top-down schema

Item form: \([\bullet \beta, j]\) (state that \(S \rightarrow w_{j+1} \ldots w_{j}\))

Axioms: \([\bullet, S, 0]\)

Goals: \([\bullet, n]\)

Inference rules:

\[
\begin{align*}
\text{Scan} & \quad \frac{[\bullet \omega_{j+1} \beta, j]}{[\bullet \beta, j+1]} \\
\text{Predict} & \quad \frac{[\bullet \beta, j]}{[\bullet \gamma \beta, j]} \quad B \rightarrow \gamma
\end{align*}
\]

Top-down deduction: example

Input: 0 the 1 cat 2 in 3 the 4 hat 5

Output:

\[
\begin{align*}
[\bullet \text{NP}, 0] & \quad \text{axiom} \\
[\bullet \text{NP PP}, 0] & \quad \text{predict } \text{NP} \rightarrow \text{NP PP} \\
[\bullet \text{D N PP}, 0] & \quad \text{predict } \text{NP} \rightarrow \text{D N} \\
[\bullet \text{the } \text{N PP}, 0] & \quad \text{predict } \text{D} \rightarrow \text{the} \\
[\bullet \text{NP PP}, 1] & \quad \text{scan} \\
[\bullet \text{cat PP}, 1] & \quad \text{predict } \text{NP} \rightarrow \text{cat} \\
[\bullet \text{PP}, 2] & \quad \text{scan} \\
[\bullet \text{P NP}, 2] & \quad \text{predict } \text{PP} \rightarrow \text{P NP} \\
[\bullet \text{in NP}, 2] & \quad \text{predict } \text{NP} \rightarrow \text{in} \\
[\bullet \text{NP}, 3] & \quad \text{scan} \\
[\bullet \text{D N}, 3] & \quad \text{predict } \text{NP} \rightarrow \text{D N} \\
[\bullet \text{the } \text{N}, 3] & \quad \text{predict } \text{D} \rightarrow \text{the} \\
[\bullet \text{N}, 4] & \quad \text{scan} \\
[\bullet \text{hat}, 4] & \quad \text{predict } \text{NP} \rightarrow \text{hat} \\
[\bullet, 5] & \quad \text{scan}
\end{align*}
\]
**Top-down parsing: algorithm**

Parse(\(\beta, j\)) ::
  if \(\beta = w_{j+1} \cdot \beta'\) then return parse(\(\beta', j + 1\))
  else if \(\beta = B \cdot \beta'\) then
    for every rule \(B \rightarrow \gamma \in P\)
    if Parse(\(\gamma \cdot \beta', j\)) then return true
    return false

if Parse(\(S, 0\)) then accept else reject

---

**Top-down vs. Bottom-up parsing**

Two inherent constraints:

1. The root of the tree is \(S\)
2. The yield of the tree is the input word

---

**An example grammar**

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

- \(Det \rightarrow \text{that} | \text{this} | \text{a}\)
- \(Noun \rightarrow \text{book} | \text{flight} | \text{meal}\)
- \(Verb \rightarrow \text{book} | \text{include} | \text{includes}\)
- \(Prep \rightarrow \text{from} | \text{to} | \text{on}\)
- \(Proper-Noun \rightarrow \text{Houston} | \text{TWA}\)
- \(Aux \rightarrow \text{does}\)

---

**An example derivation tree**

```
  S
 / \ 
VP NP
 / \ /
Verb Det Noun
  book that flight
```
An example derivation tree

S

NP
Nominal
Det Noun Prep Proper-Noun Verb Det Noun
the flight from Houston includes a meal

NP
Nominal
Aux Det Noun Prep Proper-Noun Verb Det Noun
does the flight from Houston includes a meal

Top-down vs. Bottom-up parsing

When expanding the top-down search space, which local trees are created?

Top-down vs. Bottom-up parsing

To reduce “blind” search, add bottom-up filtering.

Observation: when trying to Parse(β, j), where $β = B_γ$, the parser succeeds only if $B \Rightarrow w_{j+1} β$.

Definition: A word $w$ is a left-corner of a non-terminal $B$ iff $B \Rightarrow wβ$ for some $β$. 
Top-down parsing with bottom-up filtering

Parse($\beta$, $j$) ::
  if $\beta = w_{j+1} \cdot \beta'$ then return parse($\beta', j + 1$)
  else if $\beta = B \cdot \beta'$ then
    if $w_{j+1}$ is a left-corner of $B$ then
      for every rule $B \rightarrow \gamma \in P$
        if Parse($\gamma \cdot \beta', j$) then return true
      return false
  if Parse($\mathcal{S}, 0$) then accept else reject

Top-down vs. Bottom-up parsing

Even with bottom-up filtering, top-down parsing suffers from the following problems:

- Left recursive rules can cause non-termination: $NP \rightarrow NP PP$.
- Even when parsing terminates, it might take exponentially many steps.
- Constituents are computed over and over again

Top-down parsing: repeated generation of sub-trees

```
NP
  /\Nominal
  |      
Det | Noun
   |   
a flight from Chicago to Houston on TWA
```

Top-down parsing: repeated generation of sub-trees

```
NP
  /\Nominal
  |      
Det | Noun
   |   
a flight from Chicago to Houston on TWA
```

```
NP
  /\Nominal
  |      
Det | Noun
   |   
a flight from Chicago to Houston on TWA
```
Top-down parsing: repeated generation of sub-trees

Reduplication:

<table>
<thead>
<tr>
<th>Constituent</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>a flight</td>
<td>4</td>
</tr>
<tr>
<td>from Chicago</td>
<td>3</td>
</tr>
<tr>
<td>to Houston</td>
<td>2</td>
</tr>
<tr>
<td>on TWA</td>
<td>1</td>
</tr>
<tr>
<td>a flight from Chicago</td>
<td>3</td>
</tr>
<tr>
<td>a flight from Chicago to Houston</td>
<td>2</td>
</tr>
<tr>
<td>a flight from Chicago to Houston on TWA</td>
<td>1</td>
</tr>
</tbody>
</table>

Top-down vs. Bottom-up parsing

When expanding the bottom-up search space, which local trees are created?
Top-down vs. Bottom-up parsing

Bottom-up parsing suffers from the following problems:

- All possible analyses of every substring are generated, even when they can never lead to an S, or can never combine with their neighbors
- \( \varepsilon \)-rules can cause performance degradation
- Reduplication of effort

---

Earley's parsing algorithm

Basic concepts:

Dotted rules: if \( A \rightarrow \alpha \beta \) is a grammar rule then \( A \rightarrow \alpha \bullet \beta \) is a dotted rule.

Edges: if \( A \rightarrow \alpha \bullet \beta \) is a dotted rule and \( i, j \) are indices into the input string then \([i, A \rightarrow \alpha \bullet \beta, j]\) is an edge. An edge is passive (or complete) if \( \beta = \varepsilon \), active otherwise.

Actions: The algorithm performs three operations: scan, predict and complete.

- Dynamic programming: partial results are stored in a chart
- Combines top-down predictions with bottom-up scanning
- No reduplication of computation
- Left-recursion is correctly handled
- \( \varepsilon \)-rules are handled correctly
- Worst-case complexity: \( O(n^3) \)
**Earley’s parsing algorithm**

**rightsisters:** given an active edge $A \rightarrow \alpha \cdot B\beta$, return all dotted rules $B \rightarrow \cdot \gamma$

**leftsisters:** given a complete edge $B \rightarrow \gamma \cdot$, return all dotted edges $A \rightarrow \alpha \cdot B\beta$

**combination:**  

$[i, A \rightarrow \alpha \cdot B\beta, k] \cdot [k, B \rightarrow \gamma \cdot, j] = [i, A \rightarrow \alpha B \cdot \beta, j]$

**Parsing: Earley deduction**

**Inference rules:**

**Scan**

$[i, A \rightarrow \alpha \cdot w_{j+1}\beta, j]$

$[i, A \rightarrow \alpha w_{j+1} \cdot \beta, j + 1]$

**Predict**

$[i, A \rightarrow \alpha \cdot B\beta, j]$

$[j, B \rightarrow \cdot \gamma, j]$

$B \rightarrow \gamma$

**Complete**

$[i, A \rightarrow \alpha \cdot B\beta, k] \cdot [k, B \rightarrow \gamma \cdot, j]$

$[i, A \rightarrow \alpha B \cdot \beta, j]$

**Parse:**

1. `enteredge([0, S' \rightarrow \cdot S, 0])`
2. for $j := 1$ to $n$ do
   1. for every rule $A \rightarrow w_j$ do
     1. `enteredge([j-1, A \rightarrow w_j \cdot, j])`
3. if $S' \rightarrow S \cdot \in C[0,n]$ then accept else reject
Earley's parsing algorithm

```plaintext
derived(i,edge,j) ::
    if edge \notin C[i,j] then /* occurs check */
        C[i,j] := C[i,j] \cup \{edge\}
    if edge is active then /* predict */
        for edge' \in rightsisters(edge) do
            derived([j,edge',j])
    if edge is passive then /* complete */
        for edge' \in leftsisters(edge) do
            for k such that edge' \in C[k,i] do
                derived([k,edge'*edge,j])
```