םירצ לֶבַע

่טיל גֶּשׁ גתינע
Parsing: If \( m \in \mathcal{T}(G) \), produce the (tree) structure that is assigned by \( G \) to \( m \).

Recognition: Given a (context-free) grammar \( G \) and a string of words \( m \), determine whether \( m \in \mathcal{T}(G) \).
Flexibility: a good algorithm can be easily modified.

Efficiency:

In case of ambiguity,

Completeness: the algorithm must produce all the results.

Grammar:

Generality: the algorithm must be applicable to any

General requirements for a parsing algorithm:

Parsing
Search: Breadth-first or Depth-first

Processing vs. backtracking

Handling multiple choice: Dynamic programming vs. parallel

Island-driven)

Direction: Left-to-right vs. Right-to-left vs. mixed (e.g.,

Orientation: Top-down vs. bottom-up vs. mixed

Parameters that define different parsing algorithms:
The I cat 2 in 3 the 4 hat 5

between the input string's words:

A set of indices \( \{ 0, 1, \ldots, n \} \) is defined to point to positions

\[ u_m \ldots v_m = w_i \]

The string to recognize is \( m \).

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

Assumptions:

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

Example sentences:

\[
p \leftarrow \text{in} \\
Np \leftarrow \text{hat} \\
Np \leftarrow \text{cat} \\
Np \leftarrow \text{the} \\
\]

An example grammar
before constructing larger ones

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

If the start symbol $S$ is in the $[0,u]$ entry of the chart

Consequently, the chart is triangular. A word is recognized

$$\text{chart} \iff A \in \text{un-termin}$$

Invariant: a non-terminal $A$ is stored in the $[i, i + 1]$ entry of the chart

$$\bigl( u + 1 \bigr) \times u \times \text{dimensional matrix of size } u$$

To recognize a string of length $u$, uses a chart: a bi-

in CNF

Bottom-up, chart-based recognition algorithm for grammars

The CYK algorithm
\[\{A\} \cap [\text{chart}[i, j] = \text{chart}[i, j]} \mid \text{chart}[i, j] \in S \in \text{chart}[0, n]\]  

\text{for all rules } A \rightarrow B \cdot C \text{ do}  

\text{for all } C \in \text{chart}[k, j] \text{ do}  

\text{for all } B \in \text{chart}[i, k] \text{ do}  

\text{for } k = i+1 \text{ to } j-1 \text{ do}  

\text{for } j = j-2 \text{ down to } 0 \text{ do}  

\text{for all rules } A \rightarrow B \cdot \text{do}  

\text{for } j = i \text{ to } n \text{ do}  

\text{The CYK Algorithm}
General context-free grammars (not just CYK)

Support for ε-rules

Parsing in addition to recognition

Extensions:

The CYK algorithm
a set of goal items

a set of deduction rules

a set of axioms

A parsing scheme consists of four components:

- A parsing paradigm.
  - This is a generalization of the paradigm of parsing schemes for describing the principles behind specific parsing algorithms. We use parsing schemata, which are generalized to provide a unified framework for discussing various parsing schemes.
Given a grammar $\mathcal{G}$ and a string $w$, Parsing schema: CYK

Inference rules:

Axioms: $\forall A, \forall I \in P \rightarrow A \leftarrow A, I, + \ [I, w, +, ]$

Items: $\forall A, I \in P \rightarrow A \leftarrow A, I, [I, w, +, ]$

Goals: $[0, S, n]$

$I \in \mathbb{N}$ when $A \leftarrow A, I + I \rightarrow I + I$
The I cat 2 in 3 the a hat 5

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

CYK Parsing schema: deduction example
\[ \text{Reduce} \]
\[ \text{Shift} \]

**Inference rules:**

- **Goals:** \([ \text{goal} \] [ \text{0, 1} ] \)

- **Axioms:** \([ \text{axiom} \] [ \text{0, 1} ] \)

- **Items:** \([ \text{item} \] [ \text{axiom} \] [ \text{goal} \] [ \text{0, 1} ] \)

\((^m_n \ldots \overleftarrow{m} \leftarrow ^m_n \ldots \overleftarrow{m} \; \text{state that} \; \alpha \beta \) [ \text{Shift–Reduce} ] \)
∀ \leftarrow B \quad \frac{[\ell', \vec{g}, \bullet]}{[\vec{g}, \bullet]} \quad \text{predict}

\frac{[\ell', \vec{g}, \bullet]}{[1 + \ell, \vec{g}, \bullet]} \quad \frac{[\ell, \vec{g}, \bullet]}{[\ell, \vec{g}, \bullet + 1, \vec{m}, \bullet]} \quad \text{scan}

\text{Inference rules:}

\text{goals: } [n'] \quad [n]

\text{axioms: } S, \bullet

\text{item form: } [\ell, \vec{g}, \bullet] \quad (g, \vec{f}, \ldots, m, \vec{w}) \quad (\text{state that } S \text{ that } \vec{g}, \bullet)

\text{Parsing top-down schema}
Input: 0 the 1 cat 2 in 3 the 4 hat 5

Top-down deduction example
scan
\[ \text{predict N hat} \]
scan
\[ \text{predict D the} \]
\[ \text{N D} \]
\[ \text{predict Np} \]
\[ \text{scan} \]
\[ \text{predict P in} \]
\[ \text{predict Pp p Np} \]
\[ \text{scan} \]
\[ \text{predict N cat} \]
\[ \text{scan} \]
\[ \text{predict D the} \]
\[ \text{N D} \]
\[ \text{predict Np} \]
\[ \text{predict Np Np} \]
axiom
\[ [5', \bullet] \]
\[ [\text{that} \bullet] \]
\[ [\text{N} \bullet] \]
\[ [\text{the N} \bullet] \]
\[ [\text{D N} \bullet] \]
\[ [\text{Np} \bullet] \]
\[ [\text{in Np} \bullet] \]
\[ [\text{p Np} \bullet] \]
\[ [\text{p Pp} \bullet] \]
\[ [\text{cat Pp I} \bullet] \]
\[ [\text{N Pp I} \bullet] \]
\[ [\text{the N Pp O} \bullet] \]
\[ [\text{D N Pp O} \bullet] \]
\[ [\text{N Pp O} \bullet] \]
\[ [\text{N Pp O} \bullet] \]
\text{Top-down parsing algorithm:}

\begin{align*}
\text{if } \text{parse}(S, 0) \text{ then accept else reject} \\
\quad \text{return false} \\
\text{if } \text{parse}(\_ \_ \_ , \_ \_ \_ ) \text{ then return true} \\
\quad \text{for every rule } B \in P \\
\quad \text{else if } B = B \cdot B \text{ then return } \text{parse}(\_ \_ \_ , \_ \_ \_ ) \cdot \_ \_ \_ + 1 \\
\quad \text{else return } \text{parse}(\_ \_ \_ , \_ \_ \_ ) \\
\text{Top-down parsing: algorithm}
\end{align*}
2. The yield of the tree is the input word.

1. The root of the tree is $S$.

Two Inherent Constraints:

**Top-down vs. Bottom-up Parsing**
An example grammar:

```
pp ← prep NP
Nominal ← Nominal PP
Nominal ← Nominal NN
Nominal ← NN
NP ← Proper-Noun
NN ← Det Nominal
VP ← Verb NP
VP ← Verb VP
S ← VP S
S ← NP VP
```

```
Aux does ← does
Proper-Noun ← Proper-Noun TWA
Prep from to on ← from to on includes
Verb book includes ← book includes
Nominal ← book flight meal
Det that this a ← that this a
```
An example derivation tree
An example derivation tree
An example derivation tree
Top-down vs. Bottom-up Parsing

are created?

When expanding the top-down search space, which local trees
Definition: A word \( w \) is a left-corner of a non-terminal \( B \) iff

\[
B * \not\Rightarrow \quad w / B
\]

the parser succeeds only if \( B \not* \Rightarrow \quad \gamma / B \), where

\[
\forall \gamma = \gamma / B
\]

Observation: When trying to parse \( \gamma / B \)), where

To reduce "blind" search, add bottom-up filtering.

**Top-down vs. Bottom-up Parsing**
If parse$(S', 0)$ then accept else reject

return false  
if parse$(S', 0)$ then return true

for every rule $B \in \mathcal{P}$ do
  if $w_{i+1}$ is a left-corner of $B$ then
    if $B = B'$ then
      if $m = 0$ then
        if parse$(B', 0)$ then
          parse$(S', 0)$
        else
          return false
        end
      end
    end
  end
end

Top-down parsing with bottom-up filling
 Constituents are computed over and over again.

Even when parsing terminates, it might take exponentially many steps.

NP \leftarrow NP_{pp}.

Let recursive rules can cause non-termination.

The following problems:

Even with bottom-up filling, top-down parsing suffers from

**Top-down vs. Bottom-up Parsing**
sub-trees
Top-down parsing: repeated generation of
sub-trees
Top-down Parsing: repeated generation of
sub-trees
Top-down Parsing: Repeated Generation of
Constituent

Reduplication:

sub-trees

Top-down Parsing: Repeated Generation of
trees are created?

When expanding the bottom-up search space, which local

Top-down vs. Bottom-up Parsing
Reduplication of effort

-e-rules can cause performance degradation

with their neighbors

when they can never lead to an $S$, or can never combine

All possible analyses of every substring are generated, even

Bottom-up parsing suffers from the following problems:

Top-down vs. Bottom-up parsing
\( O(n^3) \) \( \bullet \)

Worst-case complexity:  

\- Rules are handled correctly  

\- Left-recursion is correctly handled  

No redundancy of computation  

\- Combines top-down predictions with bottom-up scanning  

Dynamic programming: partial results are stored in a chart  

Earley's parsing algorithm
predicit and complete.

Actions: The algorithm performs three operations: scan,

is passive (or complete) if \( g = e \), active otherwise.

the input string then \( [i, \] \( A \cdot \alpha \leftarrow g \cdot \alpha \) is a dotted rule and \( i, \) are indices into

Edges: If \( A \cdot \alpha \leftarrow g \cdot \alpha \) is a dotted rule rule.

Dotted rules: If \( A \cdot \alpha \leftarrow g \cdot \alpha \) is a grammar rule then \( A \)

Basic concepts:

Earley’s parsing algorithm
Earley's parsing algorithm

complete: when a complete edge is added to the chart

predict: when an active edge is added to the chart, predict all possible edges that can follow it

scan: read an input word and add a corresponding complete edge to the chart.
\[
[f \cdot g, x] = [g, y] \cdot [y, B, k] \cdot [k, \lambda] \cdot [\lambda, A, g] = [g, x] \\
\]

**Combination:**

- **edges**  
  \[g \cdot x \leftarrow A \cdot g \]

**Leftsisters:**  
Given a complete edge \( B \leftarrow \gamma \), return all dotted rules \( B \leftarrow g \).

**Rightsisters:**  
Given an active edge \( A \leftarrow g \cdot X \), return all dotted rules.

**Earley's Parsing Algorithm**
Goals: $[u \cdot s \leftarrow 0, s', 0]$

Axioms: $[0, s', 0 \leftarrow 0, s', 0]$

and also that $\alpha \leftarrow \forall \; m \cdots m_{i+1} \leftarrow \forall \; m \cdots m_{i+1}$

Item Form: $[\forall \; \alpha \leftarrow \forall \; \alpha]$

Parsing: Earley deduction
[\[ \cdot \text{B} \text{A} \Rightarrow \text{A}, \text{?}\]]

[\[ \text{?} \text{B} \Rightarrow \text{B}, \text{?}\]]

\[ \text{?} \text{B} \Rightarrow \text{B}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} + \text{?} \Rightarrow \text{?}, \text{?}\]

\[ \text{?} \Rightarrow \text{?}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]

\[ \text{?} \text{A} \Rightarrow \text{A}, \text{?}\]
if $S \in \mathcal{C}[0, n]$ then accept else reject

```
if $S' \not\in \mathcal{C}[0, n]$ then accept else reject
```

Early\’s parsing algorithm
for k such that edge `∈ C[k,'] do
for edge ` ∈ teststates(edge) do
/* complete */
enteredge([k`,edge`,`])
/* predict */
{C[j,'] `∈ C[j,'] `∈ [C[j,'], edge'] = C[j,']}
/* occurs check */
enteredge([j`,edge`,])
/* Earley's parsing algorithm */