Parsing with CFGs

L545
Dept. of Linguistics, Indiana University
Spring 2013
Parsing with CFGs: Overview

**Input:** a string  
**Output:** a (single) parse tree

- A useful step in the process of obtaining meaning
- We can view the problem as searching through all possible parses (tree structures) to find the right one

**Strategies:**

- top-down (goal-directed) vs. bottom-up (data-directed)
- depth-first vs. breadth-first
- left-corner parsing (adding bottom-up to top-down)
- chart parsing (saving partial results)
Parsers and criteria to evaluate them

- Function of a parser:
  - grammar + string → analysis trees

- Main criteria for evaluating parsers:
  - Correctness: for every grammar and for every string, every analysis returned by parser is an actual analysis
    - Correctness w.r.t. our target language is thus dependent upon the grammar we give the parser
  - Completeness: for every grammar and for every string, every correct analysis is found by the parser
    - For large grammars, this may not be practical, and for some situations, we may want only one analysis
  - Efficiency: storing partial parses is essential in being efficient (to be explained)
Example grammar and desired tree

**Sentence:** Book that flight.

- S → NP VP
- S → Aux NP VP
- S → VP
- NP → Det Nominal
- Nominal → Noun
- Nominal → Noun Nominal
- Nominal → Nominal PP
- NP → Proper-Noun
- VP → Verb
- VP → Verb NP
Direction of processing I

Top-down

**Goal-driven** processing is top-down:

- Start with the start symbol
- Derive sentential forms
  - If the string is among the sentences derived this way, it is part of the language

Problem: Left-recursive rules (e.g., NP → NP PP) can give rise to infinite hypotheses

- Plus, we can expand non-terminals which cannot lead to the existing input
- No tree takes the properties of the lexical items into account until the last stage
How are alternatives explored?

I. Depth-first

At every choice point: Pursue a single alternative completely before trying another alternative

▶ State of affairs at the choice points needs to be remembered. Choices can be discarded after unsuccessful exploration.

▶ Depth-first search is not necessarily complete.

Problem for top-down, left-to-right, depth-first processing:

▶ left-recursion

For example, a rule like $N' \rightarrow N' PP$ leads to non-termination.
How are alternatives explored?

II. Breadth-first

At every choice point: Pursue every alternative for one step at a time

- Requires serious bookkeeping since each alternative computation needs to be remembered at the same time.
- Search is guaranteed to be complete.
An example grammar

Lexicon:
- Vt → saw
- Det → the
- Det → a
- N → dragon
- N → boy
- Adj → young

Syntactic rules:
- S → NP VP
- VP → Vt NP
- NP → Det N
- N → Adj N
Top-down, left-right, depth-first tree traversal

S → NP VP
VP → Vt NP
NP → Det N
N → Adj N
Vt → saw

Det → the
Det → a
N → dragon
N → boy
Adj → young

S₁

NP₂

Det₃
| the₄
| Adj₆
| young₇
| boy₉

N₅
N₈

VP₁₀

Vt₁₁
| saw₁₂
| a₁₅
| dragon₁₇

NP₁₃
N₁₆
### A walk-through

<table>
<thead>
<tr>
<th>Goal</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>the young boy saw the dragon</td>
<td>expand S</td>
</tr>
<tr>
<td>NP VP</td>
<td>the young boy saw the dragon</td>
<td>expand NP</td>
</tr>
<tr>
<td>the N VP</td>
<td>the young boy saw the dragon</td>
<td>expand Det</td>
</tr>
<tr>
<td>N VP</td>
<td>the young boy saw the dragon</td>
<td>consume <em>the</em></td>
</tr>
<tr>
<td>dragon VP</td>
<td>young boy saw the dragon</td>
<td>expand N</td>
</tr>
<tr>
<td>boy VP</td>
<td>young boy saw the dragon</td>
<td>fail with <em>dragon</em></td>
</tr>
<tr>
<td>Adj N VP</td>
<td>young boy saw the dragon</td>
<td>fail with <em>boy</em>; (re)expand N</td>
</tr>
<tr>
<td>young N VP</td>
<td>young boy saw the dragon</td>
<td>consume <em>young</em></td>
</tr>
<tr>
<td>N VP</td>
<td>boy saw the dragon</td>
<td>expand N</td>
</tr>
</tbody>
</table>
A walk-through (cont.)

<table>
<thead>
<tr>
<th>Goal</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>dragon VP</td>
<td>boy saw the dragon</td>
<td>fail with <em>dragon</em></td>
</tr>
<tr>
<td>boy VP</td>
<td>boy saw the dragon</td>
<td>consume <em>boy</em></td>
</tr>
<tr>
<td>VP</td>
<td>saw the dragon</td>
<td>expand VP</td>
</tr>
<tr>
<td>Vt NP</td>
<td>saw the dragon</td>
<td>expand Vt</td>
</tr>
<tr>
<td>saw NP</td>
<td>saw the dragon</td>
<td>consume <em>saw</em></td>
</tr>
<tr>
<td>NP</td>
<td>the dragon</td>
<td>expand NP</td>
</tr>
<tr>
<td>Det N</td>
<td>the dragon</td>
<td>expand Det</td>
</tr>
<tr>
<td>the N</td>
<td>the dragon</td>
<td>consume <em>the</em></td>
</tr>
<tr>
<td>N</td>
<td>dragon</td>
<td>expand N</td>
</tr>
<tr>
<td>dragon</td>
<td>dragon</td>
<td>consume <em>dragon</em></td>
</tr>
<tr>
<td>&lt;empty&gt;</td>
<td>&lt;empty&gt;</td>
<td>SUCCESS!</td>
</tr>
</tbody>
</table>
Remaining choices

There are still some choices that have to be made:

1. **Which leaf node should be expanded first?**
   - Left-to-right strategy moves through the leaf nodes in a left-to-right fashion

2. **Which rule should be applied first for multiple rules with same LHS?**
   - Can just use the textual order of rules from the grammar
   - There may be reasons to take rules in a particular order (e.g., probabilities)
Parsing with an agenda

Search states are kept in an agenda

- Search states consist of partial trees and a pointer to the next input word in the sentence

Based on what we’ve seen before, apply the next item on the agenda to the current tree

- Add new items to the agenda, based on the rules in the grammar which can expand at the (leftmost) node
  - We maintain the depth-first strategy by adding new hypotheses (rules) to the front of the agenda
  - If we added them to the back, we would have a breadth-first strategy
Data-driven processing is bottom-up:

- Start with the sentence.
- For each substring, find a grammar rule which covers it.
- If you finish with a sentence, it is grammatical.

Problem: Epsilon rules \(N \rightarrow \epsilon\) allow us to hypothesize empty categories anywhere in the sentence.

- Also, while any parse in progress is tied to the input, many may not lead to an S!
Bottom-up, left-right, depth-first tree traversal

S → NP VP
VP → Vt NP
NP → Det N
N → Adj N
Vt → saw

Det → the
Det → a
N → dragon
N → boy
Adj → young

S_{17} → NP_8 \rightarrow VP_{16}

NP_8 → Det_2 \rightarrow N_7
Det_2 → the
N_7 → Adj_4 \rightarrow N_6
Adj_4 → young
N_6 → N_5
N_5 → N_4
N_4 → boy
N_5 → N_4

VP_{16} → Vt_{10} \rightarrow NP_{15}
Vt_{10} → saw_9
Vt_{10} → saw_9
Det_{12} → Det_{12}
Det_{12} → Det_{12}
N_{14} → dragon_{13}
N_{14} → dragon_{13}
A walk-through

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;empty&gt;</td>
<td>the young boy saw the dragon</td>
<td>shift <em>the</em></td>
</tr>
<tr>
<td>the</td>
<td>young boy saw the dragon</td>
<td>reduce <em>the</em> to Det</td>
</tr>
<tr>
<td>Det</td>
<td>young boy saw the dragon</td>
<td>shift <em>young</em> after failing to reduce Det</td>
</tr>
<tr>
<td>Det young</td>
<td>boy saw the dragon</td>
<td>reduce <em>young</em> to Adj after failing to reduce Det <em>young</em></td>
</tr>
<tr>
<td>Det Adj</td>
<td>boy saw the dragon</td>
<td>shift <em>boy</em></td>
</tr>
<tr>
<td>Det Adj boy</td>
<td>saw the dragon</td>
<td>reduce <em>boy</em> to N</td>
</tr>
<tr>
<td>Det Adj N</td>
<td>saw the dragon</td>
<td>reduce Adj N to N</td>
</tr>
<tr>
<td>Det N</td>
<td>saw the dragon</td>
<td>reduce Det N to NP</td>
</tr>
<tr>
<td>NP</td>
<td>saw the dragon</td>
<td>shift <em>saw</em></td>
</tr>
</tbody>
</table>
### A walk-through (cont.)

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP saw</td>
<td>the dragon</td>
<td>reduce <em>saw</em> to Vt</td>
</tr>
<tr>
<td>NP Vt</td>
<td>the dragon</td>
<td>shift <em>the</em></td>
</tr>
<tr>
<td>NP Vt the</td>
<td>dragon</td>
<td>reduce <em>the</em> to Det</td>
</tr>
<tr>
<td>NP Vt Det</td>
<td>dragon</td>
<td>shift <em>dragon</em></td>
</tr>
<tr>
<td>NP Vt Det dragon</td>
<td>&lt;empty&gt;</td>
<td>reduce <em>dragon</em> to N</td>
</tr>
<tr>
<td>NP Vt Det N</td>
<td>&lt;empty&gt;</td>
<td>reduce Det N to NP</td>
</tr>
<tr>
<td>NP Vt NP</td>
<td>&lt;empty&gt;</td>
<td>reduce Vt NP to VP</td>
</tr>
<tr>
<td>NP VP</td>
<td>&lt;empty&gt;</td>
<td>reduce NP VP to S</td>
</tr>
<tr>
<td>S</td>
<td>&lt;empty&gt;</td>
<td>SUCCESS!</td>
</tr>
</tbody>
</table>
Motivation:
- Both pure top-down & bottom-up approaches are inefficient
- The correct top-down parse has to be consistent with the left-most word of the input

**Left-corner parsing**: a way of using bottom-up constraints as part of a top-down strategy.

- Left-corner rule:
  - expand a node with a grammar rule only if the current input can serve as the left corner from this rule
- Left-corner from a rule: first word along the left edge of a derivation from the rule

Put the left-corners into a table, which then guide parsing
Grammar with left-corners

Lexicon:
Vt → saw
Det → the
Det → a
N → dragon
N → boy
Adj → young

Syntactic rules:
S → NP VP
VP → Vt NP
NP → Det N
N → Adj N

Left corners:
S ⇒ Det
VP ⇒ Vt
NP ⇒ Det
N ⇒ Adj
Left corner parsing example

Consider again *book that flight*, with these rules:

\[
\begin{align*}
S & \rightarrow \text{NP VP} & \text{Nom.} & \rightarrow \text{Noun} & \text{VP} & \rightarrow \text{Verb} \\
S & \rightarrow \text{Aux NP VP} & \text{Nom.} & \rightarrow \text{Noun Nom.} & \text{VP} & \rightarrow \text{Verb NP} \\
S & \rightarrow \text{VP} & \text{Nom.} & \rightarrow \text{Nom. PP} \\
\text{NP} & \rightarrow \text{Det Nom.} & \text{NP} & \rightarrow \text{Prop-Noun}
\end{align*}
\]

With an ambiguous word like *book*, left corners tell us the Noun reading is ruled out—it cannot start an S:

\[
\begin{align*}
S & \Rightarrow \text{Aux} & S & \Rightarrow \text{Verb} & \text{VP} & \Rightarrow \text{Verb} \\
S & \Rightarrow \text{Det} & \text{NP} & \Rightarrow \text{Det} \\
S & \Rightarrow \text{PropN} & \text{NP} & \Rightarrow \text{PropN}
\end{align*}
\]

Moving top-down, we hypothesize \( S \rightarrow \text{NP VP} \), but the NP’s left-corner is incompatible with any category of *book*:

- Thus, no NP expansions are considered
Chart parsing

Problem: Inefficiency of recomputing subresults

Two example sentences and their potential analysis:

(1) He [gave [the young cat] [to Bill]].
(2) He [gave [the young cat] [some milk]].

The corresponding grammar rules:

▶ VP → V_{ditrans} NP PP_{to}
▶ VP → V_{ditrans} NP NP

Regardless of final sentence analysis, the object NP (the young cat) will have the same analysis

⇒ No need to analyze it twice
Solution: Chart Parsing (Memoization)

- Store intermediate results:
  - a) completely analyzed constituents:
    well-formed substring table or (passive) chart
  - b) partial and complete analyses:
    (active) chart

- In other words, instead of recalculating that *the young cat* is an NP, we’ll store that information
  - Dynamic programming: never go backwards

- All intermediate results need to be stored for completeness.
- All possible solutions are explored in parallel.
Cocke Younger Kasami (CYK) Algorithm

- Grammar has to be in Chomsky Normal Form (CNF):
  - RHS with a single terminal: \( A \rightarrow a \)
  - RHS with two non-terminals: \( A \rightarrow BC \)
  - no \( \epsilon \) rules (\( A \rightarrow \epsilon \))

- A representation of the string showing positions and word indices:
  \[
  \cdot_0 \ w_1 \cdot_1 \ w_2 \cdot_2 \ w_3 \cdot_3 \ w_4 \cdot_4 \ w_5 \cdot_5 \ w_6 \cdot_6
  \]

For example:

\[
\cdot_0 \ \text{the} \cdot_1 \ \text{young} \cdot_2 \ \text{boy} \cdot_3 \ \text{saw} \cdot_4 \ \text{the} \cdot_5 \ \text{dragon} \cdot_6
\]
Well-formed substring table (passive chart)

- The well-formed substring table, henceforth (passive) chart, for a string of length \( n \) is an \( n \times n \) matrix.
- The field \((i,j)\) of the chart encodes the set of all categories of constituents that start at position \( i \) and end at position \( j \), i.e.
  - \( \text{chart}(i,j) = \{A \mid A \Rightarrow^* w_{i+1} \ldots w_j\} \)
- The matrix is triangular since no constituent ends before it starts.
Coverage Represented in the Chart

An input sentence with 6 words:

```
·0  w1 ·1  w2 ·2  w3 ·3  w4 ·4  w5 ·5  w6 ·6
```

Coverage represented in the chart:

<table>
<thead>
<tr>
<th>FROM:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0–1</td>
<td>0–2</td>
<td>0–3</td>
<td>0–4</td>
<td>0–5</td>
<td>0–6</td>
</tr>
<tr>
<td>1</td>
<td>1–2</td>
<td>1–3</td>
<td>1–4</td>
<td>1–5</td>
<td>1–6</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2–3</td>
<td>2–4</td>
<td>2–5</td>
<td>2–6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>3–4</td>
<td>3–5</td>
<td>3–6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>4–5</td>
<td>4–6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td>5–6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example for Coverage Represented in Chart

Example sentence:

\[0 \text{ the} \quad 1 \text{ young} \quad 2 \text{ boy} \quad 3 \text{ saw} \quad 4 \text{ the} \quad 5 \text{ dragon} \quad 6\]

Coverage represented in chart:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>the</td>
<td>the young</td>
<td>the young boy</td>
<td>the young boy saw</td>
<td>the young boy saw the</td>
<td>the young boy saw the dragon</td>
</tr>
<tr>
<td>1</td>
<td>young</td>
<td>young boy</td>
<td>young boy saw</td>
<td>young boy saw the</td>
<td>young boy saw the</td>
<td>young boy saw the dragon</td>
</tr>
<tr>
<td>2</td>
<td>boy</td>
<td>boy saw</td>
<td>boy saw the</td>
<td>boy saw the</td>
<td>boy saw the</td>
<td>boy saw the dragon</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>saw</td>
<td>saw the</td>
<td>the</td>
<td>the</td>
<td>the dragon</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The CKY algorithm is used, which:
- explores all analyses in parallel,
- in a bottom-up fashion, &
- stores complete subresults

This algorithm is used to:
- add top-down guidance (only use rules derivable from start-symbol), but avoid left-recursion problem
- store partial analyses
### An Example for a Filled-in Chart

**Input sentence:**

\[ \text{the young boy saw the dragon} \]

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{Det}</td>
<td>{}</td>
<td>{NP}</td>
<td>{}</td>
<td>{}</td>
<td>{S}</td>
</tr>
<tr>
<td>1</td>
<td>{Adj}</td>
<td>{N}</td>
<td>{}</td>
<td>{}</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>2</td>
<td>{}</td>
<td>{N}</td>
<td>{}</td>
<td>{}</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>3</td>
<td>{}</td>
<td>{}</td>
<td>{V, N}</td>
<td>{}</td>
<td>{VP}</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>{}</td>
<td>{}</td>
<td>{Det}</td>
<td>{NP}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>{}</td>
<td>{}</td>
<td>{}</td>
<td>{}</td>
<td>{}</td>
<td>{}</td>
</tr>
</tbody>
</table>
Filling in the Chart

- We build all constituents that end at a certain point before we build constituents that end at a later point.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td>10</td>
<td>15</td>
<td>21</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
<td>9</td>
<td>14</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
<td>13</td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>12</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16</td>
</tr>
</tbody>
</table>

for \( j \leftarrow 1 \) to length(\textit{string})

\[ \text{lexical} \_\text{chart} \_\text{fill}(j - 1, j) \]

for \( i \leftarrow j - 2 \) down to 0

\[ \text{syntactic} \_\text{chart} \_\text{fill}(i, j) \]
**lexical_chart_fill(j-1,j)**

- **Idea:** Lexical lookup. Fill the field \((j - 1, j)\) in the chart with the preterminal category dominating word \(j\).
- **Realized as:**

\[
chart(j - 1, j) := \{X | X \rightarrow \text{word}_j \in P\}
\]
syntactic_chart_fill(i,j)

- Idea: Perform all reduction steps using syntactic rules s.t. the reduced symbol covers the string from \( i \) to \( j \).

- Realized as: 
  
  $$
  chart(i, j) = \begin{cases} 
  A & A \rightarrow BC \in P, \\
  & i < k < j, \\
  & B \in chart(i, k), \\
  & C \in chart(k, j) 
  \end{cases}
  $$

- Explicit loops over every possible value of \( k \) and every context free rule:
  
  $chart(i, j) := \{}.$
  
  for \( k := i + 1 \) to \( j - 1 \)
    
    for every \( A \rightarrow BC \in P \)
      
      if \( B \in chart(i, k) \) and \( C \in chart(k, j) \) then
        
        $chart(i, j) := chart(i, j) \cup \{A\}.$
  
The Complete CYK Algorithm

Input: start category $S$ and input string $n$ := length(string)

for $j$ := 1 to $n$
    $chart(j - 1, j) := \{X \mid X \to \text{word}_j \in P\}$
    for $i$ := $j - 2$ down to 0
        $chart(i, j) := \{}$
        for $k$ := $i + 1$ to $j - 1$
            for every $A \to BC \in P$
                if $B \in chart(i, k)$ and $C \in chart(k, j)$ then
                    $chart(i, j) := chart(i, j) \cup \{A\}$

Output: if $S \in chart(0, n)$ then accept else reject
How memoization helps

If we look back to the chart for the sentence *the young boy saw the dragon*:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{Det}</td>
<td>{}</td>
<td>{NP}</td>
<td>{}</td>
<td>{}</td>
<td>{S}</td>
</tr>
<tr>
<td>1</td>
<td>{Adj}</td>
<td>{N}</td>
<td>{}</td>
<td>{}</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>{N}</td>
<td>{}</td>
<td>{}</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>{V, N}</td>
<td>{}</td>
<td>{}</td>
<td>{VP}</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>{Det}</td>
<td>{NP}</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>{N}</td>
</tr>
</tbody>
</table>

- At cell (3,6), a VP is built by combining the V at (3,4) with the NP at (4,6), based on the rule VP → V NP
- Regardless of further processing, that VP is never rebuilt
From CYK to Earley

- CKY algorithm:
  - explores all analyses in parallel
  - bottom-up
  - stores complete subresults

- desiderata:
  - add top-down guidance (to only use rules derivable from start-symbol), but avoid left-recursion problem of top-down parsing
  - store partial analyses (useful for rules right-hand sides longer than 2)

- Idea: also store partial results, so that the chart contains
  - passive items: complete results
  - active items: partial results
Representing active chart items

- well-formed substring entry:
  \( \text{chart}(i, j, A) \): from \( i \) to \( j \) there is a constituent of category \( A \)

- More elaborate data structure needed to store partial results:
  - rule considered + how far processing has succeeded
  - dotted rule:
    \[ i[A \rightarrow \alpha \bullet j \beta] \]

- active chart entry:
  \( \text{chart}(i, j, \text{state}(A, \beta)) \)  Note: \( \alpha \) is not represented
  \( A \) (incompletely) goes from \( i \) to \( j \) and can be completed by finding \( \beta \)
Dotted rule examples

- A dotted rule represents a state in processing a rule.
- Each dotted rule is a hypothesis:

<table>
<thead>
<tr>
<th>Rule</th>
<th>We found a vp if we still find</th>
</tr>
</thead>
<tbody>
<tr>
<td>$vp \rightarrow \bullet v\text{-ditr} np pp\text{-to}$</td>
<td>a $v\text{-ditr}$, a $np$, and a $pp\text{-to}$</td>
</tr>
<tr>
<td>$vp \rightarrow v\text{-ditr} \bullet np pp\text{-to}$</td>
<td>a $np$ and a $pp\text{-to}$</td>
</tr>
<tr>
<td>$vp \rightarrow v\text{-ditr} np \bullet pp\text{-to}$</td>
<td>a $pp\text{-to}$</td>
</tr>
<tr>
<td>$vp \rightarrow v\text{-ditr} np pp\text{-to} \bullet$</td>
<td>nothing</td>
</tr>
</tbody>
</table>

- The first three are **active items** (or **active edges**)
- The last one is a **passive item/edge**
The three actions in Earley’s algorithm

In $i[A \rightarrow \alpha \bullet_j B\beta]$ we call $B$ the *active constituent*.

- **Prediction**: Search all rules realizing the active constituent.
- **Scanning**: Scan over each word in the input string.
- **Completion**: Combine an active edge with each passive edge covering its active constituent.

**Success state**: $0[start \rightarrow s \bullet n]$
A closer look at the three actions

Prediction

Prediction: for each \( i[A \rightarrow \alpha \bullet_j B \beta] \) in chart
for each \( B \rightarrow \gamma \) in rules
add \( j[B \rightarrow \bullet_j \gamma] \) to chart

Prediction is the task of saying what kinds of input we expect to see

- Add a rule to the chart saying that we have not seen \( \gamma \), but when we do, it will form a \( B \)
- The rule covers no input, so it goes from \( j \) to \( j \)

Such rules provide the top-down aspect of the algorithm
A closer look at the three actions

Scanning

**Scanning:** let $w_1 \ldots w_j \ldots w_n$ be the input string

for each $i[A \rightarrow \alpha \bullet_{j-1} w_j \beta]$ in chart

add $i[A \rightarrow \alpha \ w_j \bullet_j \beta]$ to chart

Scanning reads in lexical items

- We add a dotted rule indicating that a word has been seen between $j - 1$ and $j$
- Such a completed dotted rule can be used to complete other dotted rules

These rules provide the bottom-up component to the algorithm
Completion (fundamental rule of chart parsing):

for each $i[A \rightarrow \alpha \bullet_k B \beta]$ and $k[B \rightarrow \gamma \bullet_j]$ in chart

add $i[A \rightarrow \alpha B \bullet_j \beta]$ to chart

Completion combines two rules in order to move the dot, i.e., indicate that something has been seen

- A rule covering B has been seen, so any rule A which refers to B in its RHS moves the dot
- Instead of spanning from $i$ to $k$, A now spans from $i$ to $j$, which is where B ended

Once the dot is moved, the rule will not be created again
Eliminating scanning

**Scanning:** for each \( i[A \rightarrow \alpha \bullet_{j-1} w_j \beta] \) in chart
add \( i[A \rightarrow \alpha w_j \bullet_j \beta] \) to chart

**Completion:**
for each \( i[A \rightarrow \alpha \bullet_k B \beta] \) and \( k[B \rightarrow \gamma \bullet_j] \) in chart
add \( i[A \rightarrow \alpha B \bullet_j \beta] \) to chart

**Observation:** Scanning = completion + words as passive edges. One can thus simplify scanning to adding a passive edge for each word:

for each \( w_j \) in \( w_1 \ldots w_n \)
add \( j-1[w_j \rightarrow \bullet_j] \) to chart
Earley’s algorithm without scanning

**General setup:**
apply prediction and completion to every item added to chart

**Start:**
add \(0[start \rightarrow \bullet_0 s]\) to chart

for each \(w_j\) in \(w_1 \ldots w_n\)
add \(j-1[w_j \rightarrow \bullet_j]\) to chart

**Success state:** \(0[start \rightarrow s \bullet_n]\)
A tiny example grammar

Lexicon:

vp → left

det → the

n → boy

n → girl

Syntactic rules:

s → np vp

np → det n
An example run

start
predict from 1
predict from 2
predict from 3
scan "the"
complete 4 with 5
complete 3 with 6
predict from 7
predict from 7
scan "boy"
complete 8 with 10
complete 7 with 11
complete 2 with 12
predict from 13
scan "left"
complete 14 with 15
complete 13 with 16
complete 1 with 17

1. $0[start \rightarrow \bullet_0 s]$
2. $0[s \rightarrow \bullet_0 np \ vp]$
3. $0[np \rightarrow \bullet_0 det \ n]$
4. $0[det \rightarrow \bullet_0 \ the]$
5. $0[the \rightarrow \bullet_1]$
6. $0[det \rightarrow the \bullet_1]$
7. $0[np \rightarrow det \bullet_1 n]\$
8. $1[n \rightarrow \bullet_1 boy]\$
9. $1[n \rightarrow \bullet_1 girl]\$
10. $1[boy \rightarrow \bullet_2]\$
11. $1[n \rightarrow boy \bullet_2]\$
12. $0[np \rightarrow det n \bullet_2]\$
13. $0[s \rightarrow np \bullet_2 vp]\$
14. $2[vp \rightarrow \bullet_2 left]\$
15. $2[left \rightarrow \bullet_3]\$
16. $2[vp \rightarrow left \bullet_3]\$
17. $0[s \rightarrow np vp \bullet_3]\$
18. $0[start \rightarrow s\bullet_3]$
Improving the efficiency of lexical access

- In the setup just described:
  - Words are stored as passive items so that prediction is used for preterminal categories
  - Set of predicted words for a preterminal can be huge
- If each word in the grammar is introduced by a preterminal rule $cat \rightarrow word$, one can add a **passive item for each preterminal category** which can dominate the word instead of for the word itself
- What needs to be done:
  - syntactically distinguish syntactic rules from rules with preterminals on the left-hand side, i.e. lexical entries.
  - modify scanning to take lexical entries into account
Earley parsing

The Earley algorithm is efficient, running in polynomial time.

- Technically, however, it is a recognizer, not a parser

To make it a parser, each state needs to be augmented with a pointer to the states that its rule covers

- For example, VP points to state where V was completed and state where NP was completed
- Also true of the CKY algorithm: pointers need to be added to make it a parser