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)

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 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' → 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:
- Syntactic rules:
  - Vt → saw
  - Det → the
  - Det → a
  - N → dragon
  - N → boy
  - Adj → young

A walk-through

Goal | Input | Action
--- | --- | ---
S | the young boy saw the dragon | expand S
NP VP | the young boy saw the dragon | expand NP
Det N VP | the young boy saw the dragon | consume the
N VP | young boy saw the dragon | expand N
dragon VP | young boy saw the dragon | fail with dragon
boy VP | young boy saw the dragon | expand N
Adj N VP | young boy saw the dragon | consume young
young N VP | young boy saw the dragon | expand Adj
N VP | boy saw the dragon | consume young

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

Direction of processing II
Bottom-up

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 → ε) 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

A walk-through

Analysis | Input | Action
---------|-------|--------
NP saw   | the dragon | reduce saw to Vt
NP Vt    | the dragon | reduce Vt to Det
NP Vt the| dragon     | shift the to Det
NP Vt Det| dragon     | shift dragon
NP Vt Det dragon | <empty> | reduce dragon to N
NP Vt Det N | <empty> | reduce Det N to NP
NP Vt NP  | <empty> | reduce Vt NP to VP
NP VP    | <empty> | reduce NP VP to S
S        | <empty> | SUCCESS!

Left-corner parsing

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.
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:
- \( VP \rightarrow V_{\text{drans}} NP PP \)
- \( VP \rightarrow V_{\text{drans}} NP \)

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.

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:
\[ \begin{array}{ccccccc}
  a & b & c & d & e & f & g \\
  w_1 & w_2 & w_3 & w_4 & w_5 & w_6 & w_7 \\
  \end{array} \]
For example:
\[ \begin{array}{ccccccc}
  a & b & c & d & e & f & g \\
  the & young & boy & saw & the & dragon & \end{array} \]

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.
  \[ \text{chart}(i,j) = \{ A | A \Rightarrow w_{i+1} \ldots w_j \} \]
- The matrix is triangular since no constituent ends before it starts.
Coverage Represented in the Chart

An input sentence with 6 words:

\[ w_0 \cdot w_1 \cdot w_2 \cdot w_3 \cdot w_4 \cdot w_5 \cdot w_6 \cdot \]

Coverage represented in the chart:

<table>
<thead>
<tr>
<th></th>
<th>0</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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>
<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>
<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>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>4–5</td>
<td>4–6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5–6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Parsing with a Passive Chart

- The CKY algorithm is used, which:
  - explores all analyses in parallel, &
  - 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

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>0</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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>
<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>
<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>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>4–5</td>
<td>4–6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></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:

\[ t_1 \text{ the } t_2 \text{ young } t_3 \text{ boy } t_4 \text{ saw } t_5 \text{ the } t_6 \text{ dragon } \]

Coverage represented in chart:

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

An Example for a Filled-in Chart

Input sentence:

\[ t_1 \text{ the } t_2 \text{ young } t_3 \text{ boy } t_4 \text{ saw } t_5 \text{ the } t_6 \text{ dragon } \]

for \( i := 1 \to \) length(string)

\[ \text{lexical_chart_fill}(j - 1, j) \]

for \( i := j - 2 \) down to 0

\[ \text{syntactic_chart_fill}(i, j) \]

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:

\[ \text{chart}(j - 1, j) := \{ X \mid X \rightarrow \text{word}_j \in P \} \]
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: \( \text{chart}(i,j) = \{ A \rightarrow BC \in P, \ i < k < j, B \in \text{chart}(i,k), C \in \text{chart}(k,j) \} \)
- Explicit loops over every possible value of k and every context free rule:
  \( \text{chart}(i,j) := \{ \} \)
  for \( k := i + 1 \) to \( j - 1 \)
  for every \( A \rightarrow BC \in P \)
    if \( B \in \text{chart}(i,k) \) and \( C \in \text{chart}(k,j) \) then
    \( \text{chart}(i,j) := \text{chart}(i,j) \cup \{ A \} \).

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>(S)</td>
<td></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>(N)</td>
<td>()</td>
<td>(VP)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>(V, N)</td>
<td>()</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>(Det)</td>
<td>(NP)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>(N)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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 \( \rightarrow V \) NP
- Regardless of further processing, that VP is never rebuilt

The Complete CYK Algorithm

Input: start category S and input string

\( n := \text{length(string)} \)

for \( j := 1 \) to \( n \)

\( \text{chart}(j-1,j) := \{ X | X \rightarrow \text{word} \in P \} \)

for \( i := j - 2 \) down to 0

\( \text{chart}(i,j) := \{ \} \)

for \( k := i + 1 \) to \( j - 1 \)

for every \( A \rightarrow BC \in P \)

if \( B \in \text{chart}(i,k) \) and \( C \in \text{chart}(k,j) \) then

\( \text{chart}(i,j) := \text{chart}(i,j) \cup \{ A \} \)

Output: if \( S \in \text{chart}(0,n) \) then accept else reject

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:
  \( \text{chart}(i,j,A): \text{from } i \text{ to } j \text{ there is a constituent of category } A \)
- More elaborate data structure needed to store partial results:
  - rule considered + how far processing has succeeded
  - dotted rule: \( [A \rightarrow \alpha, \bullet, \beta] \)
  - 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:

\[
\begin{array}{ll}
\text{vp} & \rightarrow \bullet \ v\text{-ditr} \ np \ pp-to \\
\text{vp} & \rightarrow \text{v\text{-ditr}} \ np \ pp-to \\
\text{vp} & \rightarrow \text{v\text{-ditr}} \ np \ pp-to \\
\text{vp} & \rightarrow \text{v\text{-ditr}} \ np \ pp-to \\
\end{array}
\]

- 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 \([A \to \alpha \gamma B\beta]\) 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**: \([\alpha[\text{start} \to w \cdot n]]\)

A closer look at the three actions

**Prediction**

- For each \([A \to \alpha \gamma B\beta]\) in chart
- For each \(B \to \gamma\) in rules
- Add \([B \to \gamma \cdot]\) 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

**Completion**

Completion combines two rules in order to move the dot, i.e.,

- 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**

- Let \(w_1 \ldots w_j \ldots w_n\) be the input string
- For each \([A \to \alpha \gamma_{j-1} w_j \beta]\) in chart
- Add \([A \to \alpha w_j \gamma \beta]\) 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

Earley’s algorithm without scanning

**General setup**

- **Start**: Add \(\alpha[\text{start} \to w \cdot s]\) to chart
- For each \(w_j\) in \(w_1 \ldots w_n\)
  - Add \([w_j \to \cdot]\) to chart

**Success state**: \([\alpha[\text{start} \to s \cdot n]]\)
A tiny example grammar

Lexicon:
- vp $\rightarrow$ left
- det $\rightarrow$ the
- n $\rightarrow$ boy
- n $\rightarrow$ girl

Syntactic rules:
- s $\rightarrow$ np vp
- np $\rightarrow$ det n

An example run

<table>
<thead>
<tr>
<th>Start</th>
<th>Predict from</th>
<th>Predict from</th>
<th>Predict from</th>
<th>Predict from</th>
<th>Predict from</th>
<th>Predict from</th>
<th>Predict from</th>
<th>Predict from</th>
<th>Complete with</th>
<th>Complete with</th>
<th>Complete with</th>
<th>Complete with</th>
<th>Complete with</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>1</td>
<td>0</td>
<td>[start $\rightarrow \bullet_0 s$]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>predict from 1</td>
<td>2</td>
<td>0</td>
<td>[s $\rightarrow \bullet_n np vp]$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>predict from 2</td>
<td>3</td>
<td>0</td>
<td>[np $\rightarrow \bullet_n det n]$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>predict from 3</td>
<td>4</td>
<td>0</td>
<td>[det $\rightarrow \bullet_n$ the]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>scan “the”</td>
<td>5</td>
<td>0</td>
<td>[the $\rightarrow \bullet_n$]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>complete 4 with 5</td>
<td>6</td>
<td>0</td>
<td>[det $\rightarrow \bullet_n$ the]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>complete 3 with 6</td>
<td>7</td>
<td>0</td>
<td>[np $\rightarrow \bullet_n$ det $\bullet_n$ n]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>predict from 7</td>
<td>8</td>
<td>0</td>
<td>[n $\rightarrow \bullet_n$ boy]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>predict from 7</td>
<td>9</td>
<td>0</td>
<td>[n $\rightarrow \bullet_n$ girl]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>scan “boy”</td>
<td>10</td>
<td>0</td>
<td>[boy $\rightarrow \bullet_n$]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>complete 8 with 10</td>
<td>11</td>
<td>0</td>
<td>[n $\rightarrow \bullet_n$ boy]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>complete 7 with 11</td>
<td>12</td>
<td>0</td>
<td>[np $\rightarrow \bullet_n$ det n $\bullet_n$]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>complete 2 with 12</td>
<td>13</td>
<td>0</td>
<td>[s $\rightarrow \bullet_n np$ $\bullet_n$ vp]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>predict from 13</td>
<td>14</td>
<td>0</td>
<td>[vp $\rightarrow \bullet_n$ left]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>scan “left”</td>
<td>15</td>
<td>0</td>
<td>[left $\rightarrow \bullet_n$]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>complete 14 with 15</td>
<td>16</td>
<td>0</td>
<td>[vp $\rightarrow \bullet_n$ left]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>complete 13 with 16</td>
<td>17</td>
<td>0</td>
<td>[s $\rightarrow \bullet_n np$ vp $\bullet_n$]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>complete 1 with 17</td>
<td>18</td>
<td>0</td>
<td>[start $\rightarrow \bullet_n$ s]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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