Parsing with CFGs

L445 / L545
Dept. of Linguistics, Indiana University
Spring 2017

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' → 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.

Top-down, left-right, depth-first tree traversal

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

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 | expand Det
the N VP | 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 | expand Adj
young N VP | young boy saw the dragon | consume young
N VP | young boy saw the dragon | expand N

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

Bottom-up, left-right, depth-first tree traversal

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!

A walk-through

Analysis | Input | Action
----------|--------|--------
NP saw    | the dragon | reduce saw to Vt
NP Vt     | the dragon | reduce the to Det
NP Vt the | dragon     | shift dragon
NP Vt Det | dragon     | shift Det to Det
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
Partial and complete analyses:

All intermediate results need to be stored for completeness.

Dynamic programming: never go backwards.

All possible solutions are explored in parallel.

RHS with two non-terminals:

A representation of the string showing positions and word indices:

For example:

Lexicon: Syntactic rules: Left corners:

| Vt → saw | S → NP VP | S ⇒ Det |
| Det → the | VP → Vt NP | VP ⇒ Vt |
| Det → a | NP → Det N | NP ⇒ Det |
| N → dragon | N → Adj N | N ⇒ Adj |
| Adj → young |

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
- VP → V
- VP → V

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

Thus, no NP expansions are considered.

Left corner parsing example

Consider again book that flight, with these rules:

S → NP VP, Nom. → Noun, VP → Verb
S → Aux NP VP, Nom. → Noun Nom., VP → Verb NP
S → VP, Nom. → Nom. PP
NP → Det Nom., NP → Proper-Noun

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

S ⇒ Aux S ⇒ Verb VP ⇒ Verb
S ⇒ Det NP ⇒ Det
S ⇒ PropNP NP ⇒ PropNP

Moving top-down, we hypothesize S → NP VP, but the NP’s left-corner is incompatible with any category of book.

Thus, no NP expansions are considered.

Solution: Chart Parsing (Memoization)

- Store intermediate results:
  - well-formed substring table or (passive) chart
  - 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 → a
  - RHS with two non-terminals: A → BC
  - no ε rules (A → ε)

- A representation of the string showing positions and word indices:

For example:

- the, young, boy, saw, the, dragon

Well-formed substring table (passive chart)

- The well-formed substring table, henceforth (passive) chart, for a string of length n is an n x 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.
  - chart(i,j) = {A | A → w_i+1 ... w_j}
- The matrix is triangular since no constituent ends before it starts.
**Coverage Represented in the Chart**

An input sentence with 6 words:

\[ w_1 \cdots w_6 \]

Covered represented in the chart:

<table>
<thead>
<tr>
<th>FROM:</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>TO:</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>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></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>
<td></td>
</tr>
</tbody>
</table>

**Example for Coverage Represented in Chart**

Example sentence:

`the young boy saw the dragon`

Coverage represented in chart:

<table>
<thead>
<tr>
<th>FROM:</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>TO:</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>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></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>
<td></td>
</tr>
</tbody>
</table>

**Parsing with a Passive Chart**

- 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:

`the young boy saw the dragon`

Chart:

<table>
<thead>
<tr>
<th>FROM:</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>TO:</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>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></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>
<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>FROM:</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>TO:</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>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></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>
<td></td>
</tr>
</tbody>
</table>

**Lexical Lookup**

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

  \[
  chart(j - 1, j) := \{X \mid X \to \text{word}_j \in P\}
  \]
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) = \{ A \rightarrow BC \in P, i < k < j, B \in chart(i,k), C \in chart(k,j) \}
- 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) ∪ {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>[NP]</td>
<td>[VP]</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[V, N]</td>
<td></td>
<td></td>
<td>[NP]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>[Det]</td>
<td></td>
<td>[VP]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</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

Representing active chart items

- well-formed substring entry:
  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:
    \[ [A \rightarrow α \cdot β] \]
  - active chart entry:
    chart(i,j, state(\alpha, \beta)) Note: \alpha is not represented A (incompletely) goes from i to j and can be completed by finding \beta

The Complete CYK Algorithm

Input: start category S and input string

n := length(string)

for j := 1 to n
  chart(i−1,j−1) := {X | X \rightarrow word \in P}
  for i := j − 2 down to 0
    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) ∪ {A}

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

From CYK to Earley

- CKY algorithm:
  - explores all analyses in parallel
  - is 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

Dotted rule examples

- A dotted rule represents a state in processing a rule.
- Each dotted rule is a hypothesis:
  \[ \begin{array}{c}
  \text{We found a vp if we still find} \\
  \text{vp \rightarrow \bullet v-ditr np pp-to} \\
  \text{vp \rightarrow v-ditr np pp-to} \\
  \text{vp \rightarrow v-ditr np pp-to} \\
  \text{vp \rightarrow v-ditr np pp-to} \\
  \end{array} \]
- The first three are active items (or active edges)
- The last one is a passive item/edge
The three actions in Earley’s algorithm

- **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: $\delta[start \to s \cdot n]$

A closer look at the three actions

**Scanning**

For each $B \rightarrow \gamma$ in chart, add $[B \rightarrow \gamma \cdot \gamma]$ 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 $[A \rightarrow \alpha B \gamma]$ in chart, add $[A \rightarrow \alpha B \cdot \gamma]$ 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 scanning 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.

A closer look at the three actions

**Prediction**

For each $[A \rightarrow \alpha B \beta]$ in chart:
- Add a rule to the chart saying that we have not seen anything in $\beta$, 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.

Eliminating scanning

- **Scanning**:
  - For each $[A \rightarrow \alpha \cdot \gamma \cdot w_j \beta]$ in chart, add $[A \rightarrow \alpha w_j \cdot \gamma \beta]$ to chart.

- **Completion**:
  - For each $[A \rightarrow \alpha \cdot B \beta]$ and $[B \rightarrow \gamma \cdot \gamma]$ in chart, add $[A \rightarrow \alpha B \cdot \gamma \beta]$ to chart.

Observation = 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 $[w_j \rightarrow ]$ to chart.

Earley’s algorithm without scanning

**General setup**:

- Apply prediction and completion to every item added to chart.

**Start**:

- Add $[start \to \epsilon, 0, s]$ to chart.
- For each $w_j$ in $w_1 \ldots w_n$:
  - Add $[w_j \rightarrow ]$ to chart.

Success state: $\delta[start \to s \cdot n]$
**Parsing with CFGs**

**Direction of processing**
- Top-down
- Bottom-up
- Left-corner parsing
- Chart parsing
- CYK
- Earley

### Parsing with CFGs

#### A tiny example grammar

**Lexicon:**
- vp → left
- det → the
- n → boy
- n → girl

**Syntactic rules:**
- s → np vp
- np → det n

#### An example run

1. **start** → • s
2. **predict from 1**
   - 0 [s → • np vp]
3. **predict from 2**
   - 0 [np → • det n]
4. **predict from 3**
   - 0 [det → • the]
5. **complete 4 with 5**
6. **complete 3 with 6**
7. **complete 7 with 8**
8. **predict from 7**
9. **predict from 7**
10. **scan "the"**
11. **complete 8 with 10**
12. **complete 7 with 11**
13. **complete 2 with 12**
14. **predict from 13**
15. **scan "left"**
16. **complete 14 with 15**
17. **complete 13 with 16**
18. **complete 1 with 17**
19. **complete 2 with 18**
20. **complete 1 with 19**
21. **complete 2 with 20**
22. **complete 2 with 21**
23. **complete 2 with 22**
24. **complete 2 with 23**
25. **complete 2 with 24**
26. **complete 2 with 25**
27. **complete 2 with 26**
28. **complete 2 with 27**
29. **complete 2 with 28**
30. **complete 2 with 29**
31. **complete 2 with 30**
32. **complete 2 with 31**
33. **complete 2 with 32**
34. **complete 2 with 33**
35. **complete 2 with 34**
36. **complete 2 with 35**
37. **complete 2 with 36**
38. **complete 2 with 37**
39. **complete 2 with 38**
40. **complete 2 with 39**
41. **complete 2 with 40**
42. **complete 2 with 41**
43. **complete 2 with 42**
44. **complete 2 with 43**
45. **complete 2 with 44**
46. **complete 2 with 45**

### 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 → 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