Parsing with CFGs

Linguistics 545
Spring 2010

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
- adding bottom-up to top-down: left-corner parsing
- making it more efficient: chart parsing (i.e., 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
    - May not always be practical, and 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.

- \[ S \rightarrow NP \ VP \]
- \[ S \rightarrow Aux \ NP \ VP \]
- \[ S \rightarrow VP \]
- \[ NP \rightarrow Det \ Nominal \]
- \[ Nominal \rightarrow \ Noun \]
- \[ Nominal \rightarrow \ Noun \ Nominal \]
- \[ Nominal \rightarrow \ Nominal \ PP \]
- \[ NP \rightarrow \ Proper-Noun \]
- \[ VP \rightarrow \ Verb \]
- \[ VP \rightarrow \ Verb \ NP \]

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., \( NP \rightarrow NP \ PP \)) 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
Vt → saw
Det → the
Det → a
N → dragon
N → boy
Adj → young

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

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
VP | the young boy saw the dragon | expand VP
NP | Det N VP | expand Det
N | N VP | consume the
Vt | saw | expand N
Det | the N VP | fail with dragon
Det | a | fail with boy; (re)expand N
N | young boy saw the dragon | expand Adj
N | boy saw the dragon | consume young
Adj | young N VP | expand N
N | Vt NP | consume dragon
<empty> | <empty> | SUCCESS!

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
- Add new items to (the front of) 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 (\(N \rightarrow \epsilon\)) 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**

A walk-through

<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 saw to Vt shift the</td>
</tr>
<tr>
<td>NP Vt</td>
<td>the dragon</td>
<td>reduce the to Det shift dragon</td>
</tr>
<tr>
<td>NP Vt the</td>
<td>dragon</td>
<td>reduce dragon to N</td>
</tr>
<tr>
<td>NP Vt Det</td>
<td>dragon</td>
<td>reduce Det N to NP</td>
</tr>
<tr>
<td>NP Vt Det N</td>
<td></td>
<td>reduce Det N to NP</td>
</tr>
<tr>
<td>NP Vt NP</td>
<td></td>
<td>reduce Det N to NP</td>
</tr>
<tr>
<td>NP VP</td>
<td></td>
<td>reduce Det N to NP</td>
</tr>
<tr>
<td>S</td>
<td></td>
<td>SUCCESS!</td>
</tr>
</tbody>
</table>

**Left-corner parsing**

Motivation:
- Both pure top-down and 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 can then guide parsing
Grammar with left-corners

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

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

Left corners:
S ⇒ Det
VP ⇒ Vt
NP ⇒ Det
N ⇒ Adj

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_{\text{ditrans}}, NP PP_{to}
• VP → V_{\text{dtrans}}, NP NP

Regardless of the final sentence analysis, the ditransitive verb (gave) and its first 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.

CFG Parsing: The Cocke Younger Kasami Algorithm

• Grammar has to be in Chomsky Normal Form (CNF), only
  – RHS with a single terminal: A → a
  – RHS with two non-terminals: A → BC
  – no ∈ rules (A ∈)
• A representation of the string showing positions and word indices:

\[
| w_1 | w_2 | w_3 | w_4 | w_5 | w_6 | w_7 \\
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>w_1</td>
<td>1</td>
<td>w_2</td>
<td>2</td>
<td>w_3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>w_4</td>
<td>5</td>
<td>w_5</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>w_6</td>
<td>8</td>
<td>w_7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\]

For example: the young boy saw the dragon

The 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. 
**Filling in the Chart**

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

  for \( j := 1 \) to length(string)
  
  \[ \text{lexical_chart_fill}(j - 1, j) \]
  
  for \( i := j - 2 \) down to 0
  
  \[ \text{syntactic_chart_fill}(i, j) \]

**Coverage Represented in the Chart**

An input sentence with 6 words:

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

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></td>
<td>0</td>
<td>0–1</td>
<td>0–2</td>
<td>0–3</td>
<td>0–4</td>
<td>0–5</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>3–4</td>
<td>3–5</td>
<td>3–6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4–5</td>
<td>4–6</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>

<table>
<thead>
<tr>
<th>TO:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0–1</td>
<td>0–2</td>
<td>0–3</td>
<td>0–4</td>
<td>0–5</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>3–4</td>
<td>3–5</td>
<td>3–6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4–5</td>
<td>4–6</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>

**Parsing with a Passive Chart**

- The CKY algorithm is used, which:
  - explores all analyses in parallel,
  - in a bottom-up fashion, &
  - stores complete subresults

- The reason this algorithm is used is to:
  - add top-down guidance (to only use rules derivable from start-symbol), but avoid left-recursion problem of top-down parsing
  - store partial analyses

**Example for Coverage Represented in Chart**

Example sentence:

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

Coverage represented in 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></td>
<td>0</td>
<td>0–1</td>
<td>0–2</td>
<td>0–3</td>
<td>0–4</td>
<td>0–5</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>3–4</td>
<td>3–5</td>
<td>3–6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4–5</td>
<td>4–6</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>

**An Example for a Filled-in Chart**

Input sentence:

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

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></td>
<td>0</td>
<td>0–1</td>
<td>0–2</td>
<td>0–3</td>
<td>0–4</td>
<td>0–5</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>3–4</td>
<td>3–5</td>
<td>3–6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4–5</td>
<td>4–6</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>

<table>
<thead>
<tr>
<th>TO:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0–1</td>
<td>0–2</td>
<td>0–3</td>
<td>0–4</td>
<td>0–5</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>3–4</td>
<td>3–5</td>
<td>3–6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4–5</td>
<td>4–6</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>

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

**Realized as:**

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

- Idea: Perform all reduction steps using syntactic rules such that the reduced symbol covers the string from i to j.

- Realized as: chart(i,j) = \{ A | A → BC ∈ P, i < k < j, B ∈ chart(i,k), C ∈ 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 → BC ∈ P
  if B ∈ chart(i,k) and C ∈ 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 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

Dotted rule examples

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

\[
\begin{align*}
vp & → • \text{v-ditr } np \text{ pp-to} & \text{We found a vp if we still find} \\
v → \text{v-ditr} & \text{ • np pp-to} & \text{a np and a pp-to}
\end{align*}
\]

The first three are examples of active items (or active edges)

The last one is a passive item/edge.
The three actions in Earley's algorithm

In $i[A \rightarrow \alpha \bullet B \beta]$ 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: $0[\text{start} \rightarrow \text{s} \bullet \text{n}]$

A closer look at the three actions

Scanning

Scanning: let $w_1 \ldots w_j \ldots w_n$ be the input string
for each $i[A \rightarrow \alpha \bullet w_{j-1} w_j \beta]$ in chart
add $i[A \rightarrow \alpha w_{j} \bullet w_{j} \beta]$ to chart

Scanning reads 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

Completion (fundamental rule of chart parsing):
for each $i[A \rightarrow \alpha \bullet B \beta]$ and $i[B \rightarrow \gamma \bullet \beta]$ in chart
add $i[A \rightarrow \alpha B \bullet \gamma]$ 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 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 $i[A \rightarrow \alpha \bullet w_{j-1} w_j \beta]$ in chart
add $i[A \rightarrow \alpha w_{j} \bullet \beta]$ to chart

Completion: for each $i[A \rightarrow \alpha \bullet B \beta]$ and $i[B \rightarrow \gamma \bullet \beta]$ in chart
add $i[A \rightarrow \alpha B \bullet \beta]$ 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 $i[j-1[w_j \rightarrow \bullet \beta]$ to chart

Earley's algorithm without scanning

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

Start:
add $0[\text{start} \rightarrow \text{s} \bullet \text{n}]$ to chart
for each $w_j$ in $w_1 \ldots w_n$
add $i[j-1[w_j \rightarrow \bullet \beta]$ to chart

Success state: $0[\text{start} \rightarrow \text{s} \bullet \text{n}]$
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

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. The 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, a VP would point to the state where its V was completed and the
  state where its NP was completed

• This is also true of the CKY algorithm we saw earlier: pointers need to be added to
  make it a parser