T-(538|725)-MALV, Natural Language Processing
Parsing techniques

Hrafn Loftsson¹  Hannes Högni Vilhjalmsson¹

¹School of Computer Science, Reykjavik University

October 2009
Outline

1. Top-down parsing
2. Bottom-up parsing
3. Chart parsing
4. Probabilistic parsing
1. Top-down parsing

2. Bottom-up parsing

3. Chart parsing

4. Probabilistic parsing
Top-down parsing (í. ofansækin þáttun)

- Constructs the parse tree starting at the root down to the leaves.
- We begin at the start symbol, \( S \).
- All subtrees constructed having \( S \) to the left of the arrow: \( S \rightarrow X \)
- This is continued at the next subtree (node), \( X \).
  - Rules found having \( X \) to the left of the arrow and all subtrees constructed for the right side.
- And so on until the leaves (words/tokens) are reached.
- Trees that don’t match the input are discarded.
Top-down parsing: An example

The waiter brought the meal

```
S
/   \\  \
NP VP⇒
```

```
S
/  \  \  \
NP VP Det Noun⇒ the
```
Top-down parsing: An example

The waiter brought the meal

[Diagram showing the parse tree for "The waiter brought the meal"]

Loftsson, Vilhjálmsson  Parsing techniques
Top-down parsing: An example

The waiter brought the meal

S
  NP  VP
⇒  Det  Noun
  the

S
  NP  VP
  Det  Noun

Top-down parsing

- For example, used in DCG (Definite Clause Grammar) in Prolog.
- Depth-first strategy:
  - Does not handle left-recursion, like:
    - np -> np, pp.
    - np -> np, conj, np.
- Backtracking:
  - Reparsing of the same constituents can happen over and over again.
1 Top-down parsing

2 Bottom-up parsing

3 Chart parsing

4 Probabilistic parsing
Bottom-up parsing (í. neðansækinn þáttun)

- Constructs the parse tree starting at the leaves up to the root.
- Starts with the words/tokens.
- Checks if a word $W$ appears to the right of an arrow in some rule: $X \rightarrow W$
- The right side symbol(s) is removed and the left side, $X$, added instead.
- And so on until the root is reached.
- Trees that do not lead to the root are discarded.
The waiter brought the meal
Bottom-up parsing: An example

The waiter brought the meal

```
Det  Noun
The  waiter
```

```
NP  Verb  Det  Noun
meal  brought  the
```

```
S
NP  VP
```
Bottom-up parsing: An example

The waiter brought the meal

The ⇒ waiter ⇒ Det Noun ⇒ NP

Noun

meal ⇒ Det Noun ⇒ Verb NP ⇒ S NP VP
Bottom-up parsing: An example

The waiter brought the meal

The waiter brought the meal

Det    Noun    NP    Verb    Det
The    waiter    Det    Noun    brought    the

Noun    NP    VP    S
meal    Det    Noun    Verb    NP    NP    VP
Bottom-up parsing: An example

The waiter brought the meal

Det ⇒ Noun ⇒ NP ⇒ Verb ⇒ Det

The ⇒ waiter ⇒ Det ⇒ Noun ⇒ brought ⇒ the ⇒

Noun ⇒ NP ⇒ Verb ⇒ NP ⇒ VP ⇒ S

meal ⇒ Det ⇒ Noun ⇒ Verb ⇒ NP ⇒ NP ⇒ VP
Bottom-up parsing: An example

The waiter brought the meal

Det  Noun  NP  Verb  Det
The  waiter  Det  Noun  brought  the

Noun
meal  ⇒  Det  Noun  ⇒  Verb  NP  ⇒  NP  VP
The waiter brought the meal

Det  Noun  NP  Verb  Det
The  waiter  Det  Noun  brought  the

Noun  NP  VP  S
meal  Det  Noun  Verb  NP  NP  VP
Bottom-up parsing: An example

The waiter brought the meal

The waiter

NP

Verb

Det

brought

Det

the

meal

NP

Verb

Det

NP

S

NP

VP
Bottom-up parsing: An example

The waiter brought the meal

\[
\text{Det} \rightarrow \text{Noun} \rightarrow \text{NP} \rightarrow \text{Verb} \rightarrow \text{Det} \\
\text{The} \rightarrow \text{waiter} \rightarrow \text{Det} \rightarrow \text{Noun} \rightarrow \text{brought} \rightarrow \text{the} \\
\text{Noun} \rightarrow \text{NP} \rightarrow \text{VP} \rightarrow \text{S} \\
\text{meal} \rightarrow \text{Det} \rightarrow \text{Noun} \rightarrow \text{Verb} \rightarrow \text{NP} \rightarrow \text{NP} \rightarrow \text{VP}
\]
A bottom-up parsing method:

1. **Shifts** a word from the phrase or sentence to parse onto a stack.
2. Applies a sequence of grammar rules to **reduce** elements of the stack.

This loop repeated until there are no more words in the list and the stack is reduced to the parsing goal (the start symbol).
## Shift-Reduce parsing: An example

<table>
<thead>
<tr>
<th>It.</th>
<th>Stack</th>
<th>S/R</th>
<th>Word list</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>shift</td>
<td>the waiter brought the meal</td>
</tr>
<tr>
<td>1</td>
<td>the</td>
<td>reduce</td>
<td>waiter brought the meal</td>
</tr>
<tr>
<td>2</td>
<td>det</td>
<td>shift</td>
<td>waiter brought the meal</td>
</tr>
<tr>
<td>3</td>
<td>det waiter</td>
<td>reduce</td>
<td>brought the meal</td>
</tr>
<tr>
<td>4</td>
<td>det noun</td>
<td>reduce</td>
<td>brought the meal</td>
</tr>
<tr>
<td>5</td>
<td>np</td>
<td>shift</td>
<td>brought the meal</td>
</tr>
<tr>
<td>6</td>
<td>np brought</td>
<td>reduce</td>
<td>the meal</td>
</tr>
<tr>
<td>7</td>
<td>np verb</td>
<td>shift</td>
<td>the meal</td>
</tr>
<tr>
<td>8</td>
<td>np verb the</td>
<td>reduce</td>
<td>meal</td>
</tr>
<tr>
<td>9</td>
<td>np verb det</td>
<td>shift</td>
<td>meal</td>
</tr>
<tr>
<td>10</td>
<td>np verb det meal</td>
<td>reduce</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>np verb det noun</td>
<td>reduce</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>np verb np</td>
<td>reduce</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>np vp</td>
<td>reduce</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>s</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Top-down

- Does not spend time on parse trees which cannot end in $S$.
- Does not use the words to guide the parsing $\Rightarrow$ it leads to the expansion of trees that have no chance to yield any solution.
- Can handle null constituents (e.g. det $\rightarrow \[]$.)
Bottom-up

- Uses the words to guide the parsing.
- Does not spend time on trees which do not match the input words.
- Does not use $S$ to guide the parsing $\Rightarrow$ it constructs subtrees even if they have no chance to result in a sentence (a full tree).
- Can handle left-recursion.
Outline

1. Top-down parsing
2. Bottom-up parsing
3. Chart parsing
4. Probabilistic parsing
Reparsing of constituents

- Backtracking, a characteristic of top-down parsers, often leads to reparsing of constituents:
  - np \(\rightarrow\) npx.
  - np \(\rightarrow\) npx, pp.
  - npx \(\rightarrow\) det, noun.
  - pp \(\rightarrow\) prep, np.

- The reparsing of npx happens for a string like “the meal of the day”

- (Can be solved by the so-called left factoring (í. vinstri þáttun) but then the grammar needs to be modified; see the course Compilers)
What is it?

- A technique to avoid a parser repeating the same analysis.
- A **chart** is a data structure in which the parser stores all the possible partial results at a given position in the sentence.
- When a subsequent word is processed, the parser fetches partial parse structures obtained so far in the chart instead of reparsing them.
- At the end, the chart contains all possible parse trees and subtrees (see Fig. 11.7).
Chart parsing

\[
\begin{align*}
s & \rightarrow \text{vp.} & s & \rightarrow \text{np.} & \text{vp} & \rightarrow v, \text{np, pp.} \\
\text{vp} & \rightarrow v, \text{np.} & \text{np} & \rightarrow \text{det, noun.} & \text{np} & \rightarrow \text{det, adj, noun.} \\
\text{np} & \rightarrow \text{np, pp.} & \text{pp} & \rightarrow \text{prep, np.}
\end{align*}
\]

A “dotted” rule

- Represents what has been parsed so far.
- \text{np} \rightarrow \text{det noun} \bullet \text{ (inactive arc)}
- \text{np} \rightarrow \text{det} \bullet \text{ noun} \text{ (active arc)}
- \text{np} \rightarrow \bullet \text{ det noun} \text{ (active arc)}
Dotted rules: An example
Dotted rules: An example

<table>
<thead>
<tr>
<th>Rules</th>
<th>Arcs</th>
<th>Constituents</th>
</tr>
</thead>
<tbody>
<tr>
<td>s → vp</td>
<td>[0,0]</td>
<td>• Bring the meal</td>
</tr>
<tr>
<td>vp → v np</td>
<td>[0,1]</td>
<td>• Bring • the meal</td>
</tr>
<tr>
<td>np → det noun</td>
<td>[1,1]</td>
<td>• the meal</td>
</tr>
<tr>
<td>np → np pp</td>
<td>[1,1]</td>
<td>• the meal</td>
</tr>
<tr>
<td>np → det noun</td>
<td>[1,2]</td>
<td>the • meal</td>
</tr>
<tr>
<td>np → det noun</td>
<td>[1,3]</td>
<td>the meal •</td>
</tr>
<tr>
<td>np → np pp</td>
<td>[1,3]</td>
<td>the meal •</td>
</tr>
<tr>
<td>vp → v np</td>
<td>[0,3]</td>
<td>Bring the meal •</td>
</tr>
<tr>
<td>s → vp</td>
<td>[0,3]</td>
<td>Bring the meal •</td>
</tr>
</tbody>
</table>

The table does not show all possible rules.
Chart parsing

The Earley algorithm

- An efficient context-free parsing algorithm (1970)
  - http://portal.acm.org/citation.cfm?id=362035
- A top-down method.
- Can handle left-recursion and empty constituents.
- Uses three operations:
  - Predictor
  - Scanner
  - Completer
The Earley algorithm

Predictor

- Selects all possible further parses by selecting all the rules that can process active arcs.
- For a rule: \( \text{lhs} \rightarrow c_1 c_2 \ldots \bullet c \ldots c_n \)
- The predictor introduces new parsing goals: \( c \rightarrow \bullet x_1 x_2 \ldots x_k \)

Scanner

- Accepts a new word from the input.
- The PoS to the right of a dot are matched against the word.
- Puts the rule: \( \text{pos} \rightarrow \text{word} \bullet \) into the chart.
The Earley algorithm

Completer

- Uses the new constituents generated by the Scanner to advance the dot of active arcs expecting them, and possibly complete the corresponding constituents.
- It first searches for rules having the dot and the end of a rule: $c \rightarrow x_1 x_2 \ldots x_n \bullet$
- Then it searches a rule like: $\text{lhs} \rightarrow c_1 c_2 \ldots \bullet c \ldots c_n$ and
- moves the dot over: $\text{lhs} \rightarrow c_1 c_2 \ldots c \bullet \ldots c_n$, and
- inserts the new arc into the chart.
## The Earley algorithm: An example

<table>
<thead>
<tr>
<th>Chart#</th>
<th>Rules</th>
<th>Arcs</th>
<th>Module</th>
<th>Constituents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>s → np</td>
<td>[0,0]</td>
<td>Start state</td>
<td>the meal of the day</td>
</tr>
<tr>
<td>0</td>
<td>np → det noun</td>
<td>[0,0]</td>
<td>Predictor</td>
<td>the meal of the day</td>
</tr>
<tr>
<td>0</td>
<td>np → det adj noun</td>
<td>[0,0]</td>
<td>Predictor</td>
<td>the meal of the day</td>
</tr>
<tr>
<td>0</td>
<td>np → np pp</td>
<td>[0,0]</td>
<td>Predictor</td>
<td>the meal of the day</td>
</tr>
<tr>
<td>1</td>
<td>det → the</td>
<td>[0,1]</td>
<td>Scanner</td>
<td>meal of the day</td>
</tr>
<tr>
<td>1</td>
<td>np → det noun</td>
<td>[0,1]</td>
<td>Completer</td>
<td>meal of the day</td>
</tr>
<tr>
<td>1</td>
<td>np → det adj noun</td>
<td>[0,1]</td>
<td>Completer</td>
<td>meal of the day</td>
</tr>
<tr>
<td>2</td>
<td>noun → meal</td>
<td>[1,2]</td>
<td>Scanner</td>
<td>of the day</td>
</tr>
<tr>
<td>2</td>
<td>np → det noun</td>
<td>[0,2]</td>
<td>Completer</td>
<td>of the day</td>
</tr>
<tr>
<td>2</td>
<td>np → np pp</td>
<td>[0,2]</td>
<td>Completer</td>
<td>of the day</td>
</tr>
<tr>
<td>2</td>
<td>s → np</td>
<td>[0,2]</td>
<td>Completer</td>
<td>of the day</td>
</tr>
<tr>
<td>2</td>
<td>pp → prep np</td>
<td>[2,2]</td>
<td>Predictor</td>
<td>of the day</td>
</tr>
</tbody>
</table>
### The Earley algorithm: An example (cont.)

<table>
<thead>
<tr>
<th>Chart#</th>
<th>Rules</th>
<th>Arcs</th>
<th>Module</th>
<th>Constituents</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>prep -&gt; of ●</td>
<td>[2,3]</td>
<td>Scanner</td>
<td>● the day</td>
</tr>
<tr>
<td>3</td>
<td>pp -&gt; prep ● np</td>
<td>[2,3]</td>
<td>Completer</td>
<td>● the day</td>
</tr>
<tr>
<td>3</td>
<td>np -&gt; ● det noun</td>
<td>[3,3]</td>
<td>Predictor</td>
<td>● the day</td>
</tr>
<tr>
<td>3</td>
<td>np -&gt; ● det adj noun</td>
<td>[3,3]</td>
<td>Predictor</td>
<td>● the day</td>
</tr>
<tr>
<td>3</td>
<td>np -&gt; ● np pp</td>
<td>[3,3]</td>
<td>Predictor</td>
<td>● the day</td>
</tr>
<tr>
<td>4</td>
<td>det -&gt; the ●</td>
<td>[3,4]</td>
<td>Scanner</td>
<td>● day</td>
</tr>
<tr>
<td>4</td>
<td>np -&gt; det ● noun</td>
<td>[3,4]</td>
<td>Completer</td>
<td>● day</td>
</tr>
<tr>
<td>4</td>
<td>np -&gt; det ● adj noun</td>
<td>[3,4]</td>
<td>Completer</td>
<td>● day</td>
</tr>
<tr>
<td>5</td>
<td>noun -&gt; day ●</td>
<td>[4,5]</td>
<td>Scanner</td>
<td>● day</td>
</tr>
<tr>
<td>5</td>
<td>np -&gt; det noun ●</td>
<td>[3,5]</td>
<td>Completer</td>
<td>● day</td>
</tr>
<tr>
<td>5</td>
<td>pp -&gt; prep np ●</td>
<td>[2,5]</td>
<td>Completer</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>np -&gt; np pp ●</td>
<td>[0,5]</td>
<td>Completer</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>s -&gt; np ●</td>
<td>[0,5]</td>
<td>Completer</td>
<td></td>
</tr>
</tbody>
</table>
Outline

1. Top-down parsing
2. Bottom-up parsing
3. Chart parsing
4. Probabilistic parsing
Sometimes, we may want to find the most likely parse tree . . .

. . . instead of generating all possible trees.

This is possible if a treebank exists for the language.

A treebank is a syntactically annotated corpus.

**PCFG**

- **PCFG** – **P**robabilistic **C**ontext **F**ree **G**rammar (Collins 1996; Charniak 1997)
- A CFG where each rule is augmented with its probability $P(lhs \rightarrow rhs|lhs)$
- The probabilities are derived from a treebank.
Probabilistic parsing

The probabilities are estimated with maximum likelihoods:

\[ P(lhs \rightarrow rhs_i | lhs) = \frac{\text{Count}(lhs \rightarrow rhs_i)}{\sum_j \text{Count}(lhs \rightarrow rhs_j))} \]

The probability for a sentence \( S \) to have the parse tree \( T \) is defined as the product of the probabilities attached to rules used to produce the tree:

\[ P(T, S) = \prod_{\text{rule}(i) \text{Producing } T} P(\text{rule}(i)) \]

See the calculation of probabilities for two parse trees on page 296.