Language Processing with Perl and Prolog
Chapter 12: Constituent Parsing

Pierre Nugues

Lund University
Pierre.Nugues@cs.lth.se
http://cs.lth.se/pierre_nugues/
Possible parsing strategies are top-down or bottom-up. Prolog uses a top-down exploration and backtracks in case of error. Ambiguity can produce two or more possible parse trees. It is necessary to use probabilistic or symbolic techniques to rank parse trees.
Bottom-up Parsing

The parser starts by attaching the lowest level constituents, such as "the" and "meal" to their corresponding nodes. It then assembles the NP (noun phrase) nodes, "the waiter" and "the meal", and further constructs the VP (verb phrase) node, "brought". Finally, it assembles the S (sentence) node, "VP NP".
Shift and Reduce

The shift and reduce algorithm implements bottom-up parsing. Two input arguments: the list of words to parse and the parsing goal. The algorithm gradually reduces words, parts of speech, and phrases until it reaches the parsing goal.

The algorithm consists of a loop of two steps:

- **Shift** a word from the phrase or sentence to parse onto a stack;
- Apply a sequence of grammar rules to **reduce** elements of the stack until there is no more word in the list and the stack is reduced to the parsing goal.
### Shift and Reduce in Action

<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>[the, waiter, brought, the, meal]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[the]</td>
<td>Shift</td>
<td>[waiter, brought, the, meal]</td>
</tr>
<tr>
<td>2</td>
<td>[det]</td>
<td>Reduce</td>
<td>[waiter, brought, the, meal]</td>
</tr>
<tr>
<td>3</td>
<td>[det, waiter]</td>
<td>Shift</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>Reduce</td>
<td>[brought, the, meal]</td>
</tr>
<tr>
<td>6</td>
<td>[np, brought]</td>
<td>Shift</td>
<td>[the, meal]</td>
</tr>
<tr>
<td>7</td>
<td>[np, v]</td>
<td>Reduce</td>
<td>[the, meal]</td>
</tr>
<tr>
<td>8</td>
<td>[np, v, the]</td>
<td>Shift</td>
<td>[meal]</td>
</tr>
<tr>
<td>9</td>
<td>[np, v, det]</td>
<td>Reduce</td>
<td>[meal]</td>
</tr>
<tr>
<td>10</td>
<td>[np, v, det, meal]</td>
<td>Shift</td>
<td>[]</td>
</tr>
<tr>
<td>11</td>
<td>[np, v, det, n]</td>
<td>Reduce</td>
<td>[]</td>
</tr>
<tr>
<td>12</td>
<td>[np, v, 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>Reduce</td>
<td>[]</td>
</tr>
</tbody>
</table>
Example:

*The meal of the day*

np --> npx. npx --> det, noun.
np --> npx, pp.

pp --> prep, np.
A chart is a data structure that avoids backtracking
It uses classical grammar rules
It is a graph (DAG) where nodes are intervals between words

Bring  the  meal

The  meal  of  the  day
Parsing with a Chart

```
0  Bring  1  the  2  meal  3
```

```
s → vp
vp → np
np → v
v → det
det → noun
noun
```
We can view rules \( vp \rightarrow v, np \) and \( vp \rightarrow v, np, pp \) in the chart.
The active chart stores constituents being parsed and marks the rules accordingly. The rule:

\[ \text{np} \rightarrow \text{det} \text{ noun} \]

is a completely parsed noun phrase: a determiner and a noun. The arc is said to be inactive. The rules below are said to be active:

\[ \text{np} \rightarrow \text{det} \bullet \text{ noun} \] A determiner has been found
\[ \text{np} \rightarrow \bullet \text{ det noun} \] We are seeking a noun phrase
The Earley Algorithm

Complexity of $O(N^3)$

Three modules: the predictor, the scanner, and the completer.

They use phrase-structure rules as:

\[
\begin{align*}
\text{start} & \rightarrow \bullet \text{ np} \\
\text{np} & \rightarrow \text{ det, noun.} \\
\text{np} & \rightarrow \text{ det, adj, noun.} \\
\text{np} & \rightarrow \text{ np, pp.} \\
\text{pp} & \rightarrow \text{ prep, np.}
\end{align*}
\]
The Predictor

The meal of the day

start --> • np
np --> • np pp
np --> • det noun
np --> • det adj noun
The Scanner

The meal of the day

0 det → the •
1 start → • np
2 np → • np pp
3 np → • det noun
4 np → • det adj noun
The Completer

The meal of the day

start --> • np
np --> • np pp
np --> • det noun
np --> • det adj noun

np --> det • adj noun
np --> det • noun
det --> the •

0 The 1 meal 2 of 3 the 4 day 5
The Next Steps (I)

The meal of the day

```
np --> det • adj noun
np --> det • noun
det --> the •
noun --> meal •
```

0 1 meal 2 of 3 the 4 day 5

start --> • np
np --> • np pp
np --> • det noun
np --> • det adj noun
The Next Steps (II)

The meal of the day

Pierre Nugues
Language Processing with Perl and Prolog
### The Prolog Database

<table>
<thead>
<tr>
<th>Module</th>
<th>New chart entries</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Position 0</td>
</tr>
<tr>
<td>start</td>
<td>arc(start, ['.'], np], 0, 0)</td>
</tr>
<tr>
<td>predictor</td>
<td>arc(np, [., d, n], 0, 0), arc(np, [., d, a, n], 0, 0), arc(np, [., np, pp], 0, 0)</td>
</tr>
<tr>
<td></td>
<td>Position 1</td>
</tr>
<tr>
<td>scanner</td>
<td>arc(d, [the, .], 0, 1)</td>
</tr>
<tr>
<td>completer</td>
<td>arc(np, [d, ., a, n], 0, 1), arc(np, [d, ., n], 0, 1)</td>
</tr>
<tr>
<td>predictor</td>
<td>[]</td>
</tr>
<tr>
<td></td>
<td>Position 2</td>
</tr>
<tr>
<td>scanner</td>
<td>arc(n, [meal, .], 1, 2)</td>
</tr>
<tr>
<td>completer</td>
<td>arc(np, [d, n, .], 0, 2)</td>
</tr>
<tr>
<td>completer</td>
<td>arc(np, [np, ., pp], 0, 2), arc(start, [np, .], 0, 2)</td>
</tr>
<tr>
<td>predictor</td>
<td>arc(pp, [., prep, np], 2, 2)</td>
</tr>
<tr>
<td>Position 3</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td><strong>scanner</strong></td>
<td>arc(prep, [of, .], 2, 3)</td>
</tr>
<tr>
<td><strong>completer</strong></td>
<td>arc(pp, [prep, ., np], 2, 3)</td>
</tr>
<tr>
<td><strong>predictor</strong></td>
<td>arc(np, [., d, n], 3, 3), arc(np, [., d, a, n], 3, 3), arc(np, [., np, pp], 3, 3)</td>
</tr>
</tbody>
</table>

| Position 4 | 
| --- | --- |
| **scanner** | arc(d, [the, .], 3, 4) |
| **completer** | arc(np, [d, ., a, n], 3, 4), arc(np, [d, ., n], 3, 4) |
| **predictor** | [] |

| Position 5 | 
| --- | --- |
| **scanner** | arc(n, [day, .], 4, 5) |
| **completer** | arc(np, [d, n, .], 3, 5) |
| **completer** | arc(np, [np, ., pp], 3, 5), arc(pp, [prep, np, .], 2, 5) |
| **completer** | arc(np, [np, pp, .], 0, 5) |
| **completer** | arc(np, [np, ., pp], 0, 5), arc(start, [np, .], 0, 5) |
Probabilistic Context-Free Grammars

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

where

\[ P(\text{lhs} \rightarrow \text{rhs}_i | \text{lhs}) = \frac{\text{Count}(\text{lhs} \rightarrow \text{rhs}_i)}{\sum_j \text{Count}(\text{lhs} \rightarrow \text{rhs}_j)}. \]
## An Example of PCFG

<table>
<thead>
<tr>
<th>Rules</th>
<th>$P$</th>
<th>Rules</th>
<th>$P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s \rightarrow np \ vp$</td>
<td>0.8</td>
<td>$det \rightarrow \ the$</td>
<td>1.0</td>
</tr>
<tr>
<td>$s \rightarrow vp$</td>
<td>0.2</td>
<td>$noun \rightarrow \ waiter$</td>
<td>0.4</td>
</tr>
<tr>
<td>$np \rightarrow det\ noun$</td>
<td>0.3</td>
<td>$noun \rightarrow \ meal$</td>
<td>0.3</td>
</tr>
<tr>
<td>$np \rightarrow det\ adj\ noun$</td>
<td>0.2</td>
<td>$noun \rightarrow \ day$</td>
<td>0.3</td>
</tr>
<tr>
<td>$np \rightarrow pronoun$</td>
<td>0.3</td>
<td>$verb \rightarrow \ bring$</td>
<td>0.4</td>
</tr>
<tr>
<td>$np \rightarrow np\ pp$</td>
<td>0.2</td>
<td>$verb \rightarrow \ slept$</td>
<td>0.2</td>
</tr>
<tr>
<td>$vp \rightarrow v\ np$</td>
<td>0.6</td>
<td>$verb \rightarrow \ brought$</td>
<td>0.4</td>
</tr>
<tr>
<td>$vp \rightarrow v\ np\ pp$</td>
<td>0.1</td>
<td>$pronoun \rightarrow \ he$</td>
<td>1.0</td>
</tr>
<tr>
<td>$vp \rightarrow v\ pp$</td>
<td>0.2</td>
<td>$prep \rightarrow \ of$</td>
<td>0.6</td>
</tr>
<tr>
<td>$vp \rightarrow v$</td>
<td>0.1</td>
<td>$prep \rightarrow \ to$</td>
<td>0.4</td>
</tr>
<tr>
<td>$pp \rightarrow prep\ np$</td>
<td>1.0</td>
<td>$adj \rightarrow \ big$</td>
<td>1.0</td>
</tr>
</tbody>
</table>
Parse Trees of *Bring the meal of the day*

**Parse trees**

**T1:** \[\text{vp(verb(bring),}
\text{ np(np(det(the), noun(meal)),}
\text{ pp(prep(of), np(det(the), noun(day)))))}\]

**T2:** \[\text{vp(verb(bring),}
\text{ np(np(det(the), noun(meal))),}
\text{ pp(prep(of), np(det(the), noun(day))))}\]
Computing the Probabilities

\[
P(T_1, \text{Bring the meal of the day}) = \\
P(vp \rightarrow v, np) \times P(v \rightarrow \text{Bring}) \times P(np \rightarrow np, pp) \times \\
P(np \rightarrow det, noun) \times P(det \rightarrow the) \times P(noun \rightarrow meal) \times \\
P(pp \rightarrow prep, np) \times P(prep \rightarrow of) \times P(np \rightarrow det, noun) \times \\
P(det \rightarrow the) \times P(noun \rightarrow day) = \\
0.6 \times 0.4 \times 0.2 \times 0.3 \times 1.0 \times 0.3 \times 1.0 \times 0.6 \times 0.3 \times 1.0 \times 0.3 = 0.00023328,
\]

\[
P(T_2, \text{Bring the meal of the day}) = \\
P(vp \rightarrow v, np, pp) \times P(v \rightarrow \text{Bring}) \times P(np \rightarrow det, noun) \times \\
P(det \rightarrow the) \times P(noun \rightarrow meal) \times P(pp \rightarrow prep, np) \times P(prep \rightarrow of) \times \\
P(np \rightarrow det, noun) \times P(det \rightarrow the) \times P(noun \rightarrow day) = \\
0.1 \times 0.4 \times 0.3 \times 1.0 \times 0.3 \times 1.0 \times 0.6 \times 0.3 \times 1.0 \times 0.3 = 0.0001944.
\]
Computing the Probabilities

```
VP 0.00023328
  Verb 0.4
  NP 0.000972
    NP 0.09
      Det 1.0
      Noun 0.3
      Bring
      the
    Prep 0.6
      NP 0.09
      Det 1.0
      Noun 0.3
      of
      the
      day
  PP 0.054
    NP 0.09
      Det 1.0
      Noun 0.3
      meal
    Det 1.0
      Noun 0.3
      the
```

```
VP 0.0001944
  Verb 0.4
  NP 0.09
    Det 1.0
    Noun 0.3
    Bring
    the
  Prep 0.6
    NP 0.09
    Det 1.0
    Noun 0.3
    of
    the
    day
  PP 0.054
    NP 0.09
      Det 1.0
      Noun 0.3
      meal
```
PCF grammars do not take into account the lexicon and the attachment preferences of *of* and *to*.

\[
P(T_1|\text{Bring the meal of the day}) \quad = \quad P(T_1|\text{Bring the meal to the table})' \\
\frac{P(T_2|\text{Bring the meal of the day})}{P(T_2|\text{Bring the meal to the table})} = \frac{P(vp \rightarrow v, np) \times P(np \rightarrow np, pp)}{P(vp \rightarrow v, np, pp)}
\]
Constituent parsing

Recall = \frac{\text{Number of correct constituents generated by the parser}}{\text{Number of constituents in the manually bracketed corpus}}.

Precision = \frac{\text{Number of correct constituents generated by the parser}}{\text{Total number of constituents generated by the parser}}.

<table>
<thead>
<tr>
<th>Bracketing</th>
<th>Crossing brackets</th>
</tr>
</thead>
<tbody>
<tr>
<td>(( ((bring) (the meal)) (of the day))</td>
<td>( ) ( )</td>
</tr>
<tr>
<td>((bring) ((the meal) (of the day)))</td>
<td>( ) ( )</td>
</tr>
</tbody>
</table>