11-722 Grammar Formalisms

Parsing Tree-Adjoining Grammars

Alon Lavie

February 12, 2004

References:
Aravind Joshi & Yves Schabes, "Tree-Adjoining Grammars"
Anthony Kroch & Aravind Joshi, "The Linguistic Relevance of TAG"
XTAG Online Tutorial: http://www.cis.upenn.edu/xtag/tutorial.html
Goal: define practical parsers for TAGs that can be easily modified to handle extensions to basic TAGs.

There is a CYK-like algorithm for parsing TAGs. Its average-case time complexity is:

\( O(n^6) \)

Best-case time complexity:

\( O(n^6) \)

Worst-case time complexity:

\( O(n^6) \)

For CFGs, parsing algorithms that use top-down information are faster and more efficient in practice (but same worst-case complexity).

Approach: Adapt Earley’s parsing algorithm for CFGs to TAGs:

- Requires binary elementary trees (0 or 2 branches at each node)
- Purely bottom-up (no predictive information)
- Best-case time complexity:
  \( O(n^6) \)
- Worst-case time complexity:
  \( O(n^6) \)

The solution is not obvious since adjunction allows the splicing of one tree into the middle of another.

Goal: design an efficient bottom-up TAG parser that scans the input left to right and uses top-down information.

The solution is not obvious since adjunction allows the splicing of one tree into the middle of another.

Goal: design an efficient bottom-up TAG parser that scans the input left to right and uses top-down information.
Two dot positions are equivalent if no node is crossed between them.

ends when the dot reaches the root.

A tree traversal starts with the dot at the root.

right and above (r.a) ·

left and above (l.a), left and below (l.b), right and below (r.b),

tree · The dot can be in one of four positions with respect to a node in a

tree · The dot moves the dot through a tree.

Parser moves the dot through a tree ·

Allows us to explicitly define parser 'confurations' - how the

been matched with the input, and what remains to be matched.

The "dot" represents "where we are" ·

At any time there is only one dot in a tree ·

Use a dotted tree data structure similar to dotted CFG rules.

Dot1 Trees
10.1. Example of a tree traversal.

A Dotted Tree
The chart collects items in a chart rather than constructing sets of items.

\[ \text{\textbf{The Chart}} \]
There are five main chart operations:

1. **SCAN**
2. **PREDICT**
3. **COMPLETE**
4. **ADJOIN**
5. **SUBSTITUTE**

These are the five main chart operations:
The Parsing Operations

<table>
<thead>
<tr>
<th>Parse</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td><img src="1" alt="Input Diagram" /></td>
<td><img src="2" alt="Input Diagram" /></td>
<td><img src="3" alt="Input Diagram" /></td>
</tr>
<tr>
<td>Parse</td>
<td><img src="1" alt="Parse Diagram" /></td>
<td><img src="2" alt="Parse Diagram" /></td>
<td><img src="3" alt="Parse Diagram" /></td>
</tr>
<tr>
<td>Scan</td>
<td><img src="1" alt="Scan Diagram" /></td>
<td><img src="2" alt="Scan Diagram" /></td>
<td><img src="3" alt="Scan Diagram" /></td>
</tr>
<tr>
<td>Output</td>
<td>+<img src="1" alt="Output Diagram" /></td>
<td>+<img src="2" alt="Output Diagram" /></td>
<td>+<img src="3" alt="Output Diagram" /></td>
</tr>
</tbody>
</table>

Figure 10.17. The core operations of the parser.
SCAN is a bottom-up operation that scans the input string. It has two cases:

1. The dot is left and above a non-empty terminal that matches the next item in the input.

   \[ [\alpha, \text{dot}, \ell, i', j', k', l', \text{nil}] \leftarrow [\alpha, \text{dot}, \text{ra}, i', j', k', l', l + 1, \text{nil}] \]

   Move the dot without consuming any input item.

2. The dot is left and above an empty symbol.

   \[ [\alpha, \text{dot}, \ell, i', j', k', l', \text{nil}] \leftarrow [\alpha, \text{dot}, \ell, i', j', k', l', \text{nil}] \]

   Move the dot with the input item consumed.
The SCAN Operation
Parsing Operations: PREDICT

- **PREDICT** is a top-down operation that predicts new items based on what has already been seen.

- The **PREDICT** operation has three cases for adjunction and one for substitution:
  1. The dot is left and above a non-terminal that allows adjoining
     → Predict all auxiliary trees adjoinable at the dotted node
  2. The dot is left and above a non-terminal that allows adjoining
     → Predict that no adjoining takes place at the dotted node
  3. The dot is left and below the foot node of an auxiliary tree
     → Try to recognize the subtree below any node to which the auxiliary tree could have been adjoined
  4. The dot is left and above a non-terminal marked for substitution
     → Predict all initial trees that could be substituted at the dotted node
The PREDICT Operation

(1) \[ \begin{array}{c}
\text{A} \\
[i,j,k,l,nil] \\
\end{array} \rightarrow \begin{array}{c}
\text{A} \\
[l,-,l,nil] \\
\end{array} \]

(2) \[ \begin{array}{c}
\text{A} \\
[i,j,k,l,nil] \\
\end{array} \rightarrow \begin{array}{c}
\text{A} \\
[l,-,l,nil] \\
\end{array} \]

(3) \[ \begin{array}{c}
\text{A} \\
[l,-,l,nil] \\
\end{array} \rightarrow \begin{array}{c}
\text{A} \\
[l,-,l,nil] \\
\end{array} \]
**Parsing Operations:**

- **COMPLETE**
  - is a bottom-up operation that combines two items from the tree.
  - **COMPLETE** has two cases:
    - 1. The dot is right and below a non-terminal node.
    - 2. The dot is right and below an auxiliary tree adjoined on the dotted node.

Assume that the next input token comes from the part right of the root node of an auxiliary tree adjoined on the dotted node. Try to further recognize the same tree by combining two items.

Form a new item that spans a larger portion of the input.
The Complete Operation
ADJOIN is a bottom-up operation that combines two items by adjoinction to form a new item that spans a larger portion of the input.

The ADJOIN operation has a single case.

The dot is right above a non-terminal that allows adjoining an auxiliary tree at the dotted node.

Parsing Operations: ADJOIN
The ADJOIN Operation
The SUBSTITUTE operation has a single case.

1. The dot is left and above a non-terminal marked for substitution.

Substitute a completed initial tree at the dotted node.

1511-722 Grammar Formalisms

PARSING OPERATIONS: SUBSTITUTE
The algorithm imposes no conditions on the grammar

- Elementary trees are not required to be binary
- The empty string may appear on the frontier of elementary trees
- Elementary trees are not required to be binary
- The algorithm imposes no conditions on the grammar

The algorithm is an agenda driven parser

- The algorithm is off-line (it needs to know the length of the input before starting) but can be modified to run on-line
- An input string is accepted if a tree that is derived from an initial tree and spans the input is found (i.e. if an item of the form

\[
\lambda, 0, \lambda, 0, -1, n, \text{null} \quad \text{is in the chart}
\]

and spans the input is found (i.e. if an item of the form

\[
\lambda, 0, \lambda, 0, -1, n, \text{null} \quad \text{is in the chart}
\]

and spans the input is found (i.e. if an item of the form

\[
\lambda, 0, \lambda, 0, -1, n, \text{null} \quad \text{is in the chart}
\]

and spans the input is found (i.e. if an item of the form

\[
\lambda, 0, \lambda, 0, -1, n, \text{null} \quad \text{is in the chart}
\]

and spans the input is found (i.e. if an item of the form

\[
\lambda, 0, \lambda, 0, -1, n, \text{null} \quad \text{is in the chart}
\]

and spans the input is found (i.e. if an item of the form

\[
\lambda, 0, \lambda, 0, -1, n, \text{null} \quad \text{is in the chart}
\]

and spans the input is found (i.e. if an item of the form

\[
\lambda, 0, \lambda, 0, -1, n, \text{null} \quad \text{is in the chart}
\]

and spans the input is found (i.e. if an item of the form

\[
\lambda, 0, \lambda, 0, -1, n, \text{null} \quad \text{is in the chart}
\]

and spans the input is found (i.e. if an item of the form

\[
\lambda, 0, \lambda, 0, -1, n, \text{null} \quad \text{is in the chart}
\]

and spans the input is found (i.e. if an item of the form

\[
\lambda, 0, \lambda, 0, -1, n, \text{null} \quad \text{is in the chart}
\]

and spans the input is found (i.e. if an item of the form

\[
\lambda, 0, \lambda, 0, -1, n, \text{null} \quad \text{is in the chart}
\]

and spans the input is found (i.e. if an item of the form

\[
\lambda, 0, \lambda, 0, -1, n, \text{null} \quad \text{is in the chart}
\]

and spans the input is found (i.e. if an item of the form

\[
\lambda, 0, \lambda, 0, -1, n, \text{null} \quad \text{is in the chart}
\]

and spans the input is found (i.e. if an item of the form

\[
\lambda, 0, \lambda, 0, -1, n, \text{null} \quad \text{is in the chart}
\]

and spans the input is found (i.e. if an item of the form

\[
\lambda, 0, \lambda, 0, -1, n, \text{null} \quad \text{is in the chart}
\]

and spans the input is found (i.e. if an item of the form

\[
\lambda, 0, \lambda, 0, -1, n, \text{null} \quad \text{is in the chart}
\]
The Recognizer Algorithm
Example: The Grammar
Example: Recognition

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(F+G)</td>
<td>Summary of</td>
</tr>
</tbody>
</table>
A large class of unambiguous TAGS can be parsed in linear time

Unambiguous TAGS can be parsed in \( O(u N|I \cap \mathcal{A}||A|) \) time

There are at most \( \frac{u N|I \cap \mathcal{A}||A|}{6} \) instances of the indices \( (b,d,e,k,l,q) \)

There are at most \( u N|I \cap \mathcal{A}||A| \) pairs of dotted trees to combine

The addition operation may be called at most \( u N|I \cap \mathcal{A}||A| \) times

\[ u N|I \cap \mathcal{A}||A| \]

\[ u N|I \cap \mathcal{A}||A| \]

\[ u N|I \cap \mathcal{A}||A| \]

\[ u N|I \cap \mathcal{A}||A| \]

\[ u N|I \cap \mathcal{A}||A| \]

\[ u N|I \cap \mathcal{A}||A| \]

The worst-case time complexity of the general algorithm is

\[ O(u N|I \cap \mathcal{A}||A|) \]