Tree-Adjoining Grammar (TAG)

L614
Spring 2010

Based heavily on Abeillé and Rambow (2000) and Joshi and Schabes (1997)

TAG

• Pseudo-extension of CFGs
  – Abandon the context-free grammar formalism
  – Keep the idea of deriving complete trees in a sequence of rewriting steps—but in TAG we rewrite trees, not strings

• Highly lexicalized (LTAG):
  – Every tree is associated with exactly one lexical item
  – Every lexical item is associate with a set of trees

Phrase Structure Trees

S
  NP John
  VP really
  VP V NP
  VP likes Lyn

(1) a. S → NP VP
b. VP → really VP
c. VP → V NP
d. V → likes
e. NP → John
f. NP → Lyn

String rewriting derivation

1. S → NP VP (1a)
2. → John VP (1e)
3. → John really VP (1b)
4. → John really V NP (1c)
5. → John really likes NP (1d)
6. → John really likes Lyn (1f)

Tree Substitution Grammars

• Elementary structures are trees
• A down arrow (↓) indicates where a substitution takes place

Substitution operation

The substitution operation allows us to insert elementary trees into other elementary trees
• Where there is a (non-terminal) node marked for substitution (↓) on the frontier, an elementary tree rooted in the same category can be substituted there
Final tree

So, we end up with the following derived tree:

```
S
   NP    VP
     I     I
John  V  NP
   likes  Lyn
```

Notes:
- order of substitutions is irrelevant
- This tree is completed = there are no substitution nodes left on the frontier

Elementary trees

Let’s step back a little and look at the building blocks of TAG. Our basic elements are elementary trees, which come in two guises:

- **initial trees**, which have:
  - root node
  - interior nodes labeled by non-terminal symbols
  - frontier nodes of terminal and non-terminal symbols; substitution nodes are marked by the down arrow (↓)

⇒ Tree Substitution Grammars (TSGs) only use initial trees

Elementary trees (cont.)

- **auxiliary trees**, which have
  - root node
  - interior nodes labeled by non-terminal symbols
  - frontier nodes similar to usage in initial trees, but with a designated (*) foot node = identical label to the root node

⇒ TAGs need auxiliary trees for adjunction

⇒ In LTAG, at least one frontier node must be a terminal symbol (lexical item)

Lexicalization

Lexicalization is the process of associating at least one terminal element with every elementary tree. Adjunction is necessary if we want to lexicalize the grammars in a linguistically meaningful way, i.e., substitution isn’t enough.

```
α₁ α₂ α₃ β₁
NP   NP   VP
S
   VP
     NP
      V
    likes Lyn really VP*
```

The need for adjunction

With the elementary trees above and using only substitution, there is no way to generate *John really likes Lyn*.

We would need an elementary tree along the following, unappealing lines:

```
S
   NP    VP
     really    VP
```

Adjunction

So, we introduce the adjunction operation, which is where auxiliary trees come in.

- We can now insert one tree into another, provided that the nodes match up
- That is, an auxiliary tree can modify an XP iff its root and foot nodes are both labeled XP

Using adjunction and substitution gives us true Tree Adjoining Grammars (TAGs)
Adjunction example

\[\alpha_4 \beta_1 \Rightarrow \alpha_5\]

\[\text{S} \quad \text{VP} \quad \text{really} \quad \text{VP}^* \quad \text{S}\]

\[\text{NP} \quad \text{V} \quad \text{Lyn} \]

\[\text{NP} \quad \text{V} \quad \text{Lyn} \]

Adjunction operation

- An auxiliary tree is inserted into an initial tree (or derived tree) by cutting the initial/derived tree into two parts, above and below a node (A)
  - The node of the root of the auxiliary tree is identified with the node A
  - The node of the foot of the auxiliary tree is identified with the root of the excised tree

Selective Adjunction

One possible analysis of **put** could involve selective adjunction:

\[\alpha_6 \beta_3 \beta_4\]

\[\text{S} \quad \text{VP} \quad \text{VP}^* \quad \text{SA} (\beta_3, \beta_4, \ldots) \quad \text{PP} \quad \text{P} \quad \text{NP}^* \quad \text{VP}^* \quad \text{on} \]

⇒ We might want a way to say that locative VP modifiers can adjoin here → we’ll come back later to using features to redefine adjunction constraints

Null Adjunction

For when you absolutely cannot have an adjunct modifying a phrase

\[\text{S} \quad \text{VP} \quad \text{S}^* \quad \text{VP}^* \quad \text{VP}\]

\[\text{NP} \quad \text{V} \quad \text{John} \]

\[\text{NP} \quad \text{V} \quad \text{Lyn} \]

Obligatory Adjunction

For when you absolutely must have adjunction at a node:

\[\alpha \beta_1 \beta_2\]

\[\text{S} \quad \text{VP} \quad \text{Aux} \quad \text{NP} \quad \text{VP}^* \quad \text{Aux} \quad \text{VP}^* \quad \text{VP}\]

\[\text{NP} \quad \text{V} \quad \text{seen} \quad \text{Aux} \quad \text{is}\]

This is often used to handle complement structures where the complement and the mother are the same category

- i.e., the adjunction operation is not the same as identifying linguistic adjuncts
Derived Trees and Derivation Trees

TAG distinguishes between derived trees and derivation trees.

- Derived trees are akin to context-free/phrase structure trees
- Derivation trees are akin to dependency trees

TAG provides a way of having both kinds of representations.

Example Lexicon

Recall the following lexical entries:

\[
\begin{align*}
\alpha_1 & \rightarrow \text{NP} \\
\alpha_2 & \rightarrow \text{VP} \\
\alpha_3 & \rightarrow \text{S} \\
\beta_1 & \rightarrow \text{V} \\
\beta_2 & \rightarrow \text{NP} \\
\beta_3 & \rightarrow \text{VP} \\
\end{align*}
\]

Derived Trees

The derived tree is obtained by gluing all the tree pieces together until there's a normal-looking PS tree:

```
S
   NP       VP
   John     V
         NP   NP
                likes Lyn
```

But this tells us nothing about how the tree was derived.

How to come up with a derivation tree

Each node in the derivation tree records the address of the node in the parent tree to which the adjunction/substitution was performed:

- 0 is the root node address
- \( k \) is the address of the \( k^{th} \) child of the root node
- \( p.q \) is the address of the \( q^{th} \) child of the node at address \( p \) (sort of like the \( q^{th} \) child of the \( p^{th} \) child)

Derivation Trees

The derivation tree records a history of the derivation and in the process captures the dependency relations among words in the sentence.

```
\begin{align*}
\alpha_1 & \rightarrow \text{S} \\
\alpha_2 & \rightarrow \text{NP} \\
\alpha_3 & \rightarrow \text{VP} \\
\beta_1 & \rightarrow \text{V} \\
\beta_2 & \rightarrow \text{NP} \\
\beta_3 & \rightarrow \text{VP} \\
\end{align*}
```

Derivation tree address

Lyn gets the annotation 2.2 because VP is the second daughter of S, and NP is the second daughter of VP.

```
**Locality**

TAG has a different notion of *locality* than in other formalisms

- On the one hand, an initial tree (e.g., lexical entry) can be of arbitrary size, so the domain of locality is increased.
  ⇒ Extended domain of locality (EDL)

- On the other hand, small initial trees can have multiple adjunctions inserted within them, so what are normally considered non-local phenomena are treated locally
  ⇒ Factoring recursion from the domain of dependencies (FRD)

**Domain of locality: agreement**

The lexical entry for a verb like *loves* will contain a tree like the following:

```
  S
   /\ NP_{3.sg} VP
    /\   V NP
     /\ loves
```

With this extended domain of locality, we can easily state agreement between the subject and the verb in a lexical entry

**CFG notion of agreement**

Compare the corresponding CFG rules; agreement has to be transferred between at least three different rules:

- $S \rightarrow NP_{3.sg} VP_{3.sg}$
- $VP_{3.sg} \rightarrow V_{3.sg} NP$
- $V_{3.sg} \rightarrow loves$

**Factoring recursion from domain: Extraction**

Another advantage of TAG’s domain of locality is how extraction phenomena can be captured in a lexical entry

```
  S
   /\ NP_{i} VP
    /\ V NP_{i}
     /\ loves
```

This will license a clause like *Which book Max read*

**Example trees for extraction**

The derived and derivation trees for *Which book Max read*:

```
  which book
  NP_{i}

  Max
  NP

  read
  V
```

**Extraction: strengths**

One of the strengths of this method is that we can adjoin a phrase like *do you think*, and we still maintain the appropriate dependency relations:
The derivation tree

The derivation tree maintains the same relations, simply adding another branch.

That is, even though the derived tree is much higher, the dependency relations are the same.

Extraction: weaknesses

Some extraction phenomena are not as easy to handle in TAG, such as the following:

(2) This building, John bought a picture of.

What’s wrong with this?

• The normal TAG view of extraction depends on adjunction, which is defined as involving a tree with identical root and foot nodes

• But picture is an NP, and we need to add a sentence in-between

Problems with picture phrases

• Adjunction of this entry for bought into the picture tree is needed to get This building, John bought a picture of, but it is impossible

• TAG has to be extended to multi-component TAG (MCTAG), which we won’t cover in detail, except to show a possible new entry for picture:

Using features in TAG

We have alluded to using features before, but we have not properly introduced them

• Features can be added to nodes in a tree

• In order for a tree to be substituted or adjoined, it must match the features of the node it is attaching to.

• In this way, we can reconstruct the ideas of obligatory, null, and selective adjunction

MCTAG practice

How would you propose to write analyze extraposed relative clauses, as in the following?

(3) Somebody, lives nearby [who, has a CD-burner].

(Example taken from slides by Laura Kallmeyer, Timm Lichte, and Wolfgang Maier, Grammar Formalisms: Extensions of TAG)
A simple way of using features is simply as we’ve seen before, e.g., to enforce agreement:

\[
\begin{align*}
S & \quad \text{NP} \quad \text{VP} \\
\text{NP} & \quad \text{VP} \\
\text{VP} & \quad \text{NP}
\end{align*}
\]

- Above seen’s VP node, the tree is tensed; below, it is not.
- These features do not unify, so the tree is not legal without adjunction

To reconstruct the three kinds of adjunction, we need to define top and bottom feature structures

- top = tree above this node has these features, i.e., behaves like this
- bottom = tree below this node has these features

Mostly, we have just been looking at the formal description of TAGs; we need to further restrict these trees to make them match language phenomena. Some possible constraints:

- An elementary tree is the maximal syntactic projection of a lexical item
- Auxiliary trees are only used for modifiers, functional categories, predicates with verbal complements, and raising predicates
- An elementary tree is associated with a semantic meaning

We can also group elementary trees into tree families in order to be able to capture linguistic generalizations (right now, each lexical tree has to be individually stipulated)

You can view a lexical entry’s initial tree as a supertag, i.e., a part-of-speech tag with more syntactic information than usual

<table>
<thead>
<tr>
<th>Traditional tags</th>
<th>Supertags</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adj</td>
<td>Adj</td>
</tr>
<tr>
<td>asleep</td>
<td>asleep</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Adj</td>
<td>Adj</td>
</tr>
<tr>
<td>other</td>
<td>other</td>
</tr>
<tr>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td>VP</td>
<td>VP</td>
</tr>
<tr>
<td>VP</td>
<td>VP</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
</tr>
</tbody>
</table>

We can now capture distinctions between adjectives without having to specify new categories
The TAG formalism presents some problems for parsing (more details in the Joshi and Schabes (1997) paper):

- Adjunction is a complicated operation because it can wrap strings around other strings
  - John loves Mary can become John probably loves Mary completely
- Thus, more memory is required to parse a string and more operations are needed for chart parsing

References
