Syntax-Semantic Interface and Tree Adjoining Grammar

Djamé Seddah

NCLT, DUBLIN

01/18/06
Objective: to build a predicate argument structure from a TAG analysis

From a Tree Adjunct Grammar (TAG) analysis, we want:

▸ to build a deep syntactic structure with all argumental relations represented

▸ to have all the analysis in the same structure

We claim it is possible if we work directly in a structure which combines **derived tree** and **derivation tree**: A shared forest
What is an interface between syntax and semantic?

A way to construct a semantic meaning of a given sentence.

So, what do you mean by semantic meaning?

- a logical formula?
- a dependency graph?
- a predicate argument structure?

Even, if they provide different level of informations, they rely on the same principle: The Freege theorem.

The meaning of an expression is a function of the meanings of its parts.
If we associate minimal sense to each part of a sentence

If we provide an interpretation function $f$

⇒ We can obtain a meaning

let’s do that for “(1) Tarzan loves Jane” and let $f$ assigns the first argument of ‘LOVE to the longest noun:

<table>
<thead>
<tr>
<th>Tarzan</th>
<th>’TARZAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane</td>
<td>’JANE</td>
</tr>
</tbody>
</table>

Semantic meaning of (1): ’TARZAN ’LOVE ’JANE

But “(2) Jane loves Tarzan” has the same meaning! We must rely on a better interpretation function and for that we may use the order induced by syntax to assign argument positions to words.
Providing this mini model:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Production</th>
<th>Annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>$S \rightarrow NP \ VP$</td>
<td>$([1]=f(NP)) \ ([PRED]=f(VP))$</td>
</tr>
<tr>
<td>r2</td>
<td>$NP \rightarrow \text{Tarzan}$</td>
<td>'TARZAN'</td>
</tr>
<tr>
<td>r3</td>
<td>$VP \rightarrow V \ NP$</td>
<td>'([PRED]=f(V)) \ ([2]=f(NP))$</td>
</tr>
<tr>
<td>r4</td>
<td>$V \rightarrow \text{love}$</td>
<td>'LOVE'</td>
</tr>
<tr>
<td>r5</td>
<td>$NP \rightarrow \text{Jane}$</td>
<td>'JANE'</td>
</tr>
</tbody>
</table>

⇒ We have to apply the rules and therefore to follow the derivations to get the proper result.
Applying this model

---

After the applications of the derivation rules, we obtain:

\[ 'TARZAN 'LOVE 'JANE \]

⇒ Once again, we have the correct interpretation

⇒ But what if we want to analyze “Jane is loved by Tarzan”? 

Applying this model

Syntactically speaking we have 2 options:

- adding the following rules:
  
  r3 $\text{VP} \rightarrow \text{V'} \text{ PP}$  ($[\text{PRED}]=f(\text{V}))$  ($[1]=f(\text{PP}))$
  
  r6 $\text{V'} \rightarrow \text{be loved}$  '$\text{LOVE}$
  
  r7 $\text{PP} \rightarrow \text{Prep NP}$  $[\text{PRED}]=f(\text{NP})$
  
  r8 .. .. ..

$\Rightarrow$ We have to modify deeply the corresponding semantic rules (r1 for the inversion of the arguments, etc.)

- trying to use the fact that we are still trying to express relation between words even if this is hidden by the mechanism behind the rules

$\Rightarrow$ So we should try to lexicalize this grammar a little bit...
if we replace the main VP rules by :

r3 \( \text{VP} \rightarrow \text{love NP} \quad \text{'Love ([1]=f(NP))} \)
r3' \( \text{VP} \rightarrow \text{be loved GP} \quad \text{'Love ([2]=f(PP))} \)

The model is a lot more readable and simplified but the problem of the inversion argument in rule r1 is still here
Let's consider these 2 trees:

Active Parse Tree

```
S
  | NP
  |   VP
  |   | NP
  |   V
Tarzan loves Jane
```

Passive Parse Tree

```
S
  | NP[be loved] Prep NP
  |   | PP
  |   | NP
  | V' by
Tarzan
```

Assume that the red part is a single unit, called $\alpha_{\text{active}}$ (resp. $\alpha_{\text{passive}}$) and represents by itself a set of derivation rules.
Lexicalization and semantic 3

We could then express the derivation trees differently

\[ \alpha_{active} \]

\[
\begin{array}{c}
\text{r2(Tarzan)} \\
r5(Jane)
\end{array}
\]

\[ \alpha_{passive} \]

\[
\begin{array}{c}
r5(Jane) \\
r2(Tarzan)
\end{array}
\]

Observations:

- They are very similar
  - w.r.t to the order of the arcs
  - one solution: numbered the nodes according to argument positions
  - Implicit: one argument position is linked to a derivation operation on a leaf node of \( \alpha_X \)
  - Hypothesis: would it be simpler to deal with trees instead of rules -> it would simplify the semantic model
let’s call $\alpha_2$ and $\alpha_5$ the tree corresponding to r2 (NP → Tarzan) and r5 (NP → Jane):
let’s call $\alpha_1$ the tree corresponding to $\alpha_{active}$ (resp. $\alpha'_1$ and $\alpha_{passive}$):

\begin{align*}
\text{VP} & \quad \text{NP}_0 \\
\text{V} & \quad \text{NP}_1 \\
\text{loves} & \\
\quad \text{rl} & \\
\text{S} & \\
\text{NP}_0 & \quad \text{VP} \\
\text{NP}_1 & \\
\text{VP} & \quad \text{S} \\
\text{NP}_1 & \quad \text{VP} \\
\text{V'} & \quad \text{PP} \\
[\text{be loved}] & \\
\text{Prep} & \quad \text{NP}_0 \\
\text{by} &
\end{align*}

Notice the number on the leaf nodes
analysis for “Jane is loved by Tarzan”:
Result: Derived Tree (Parse Tree) and Derivation Tree (History of what have been derived).

$$\begin{align*}
S & \rightarrow NP \ VP \\
NP & \rightarrow Jane \\
VP & \rightarrow V' \ PP \\
V' & \rightarrow be \ loved \\
PP & \rightarrow Prep \ NP \\
Prep & \rightarrow by \\
NP & \rightarrow Tarzan \\
\alpha'_1(love) & \rightarrow 0-\alpha_2(Tarzan) \ 1-\alpha_5(Jane)
\end{align*}$$
Pause: Where is the semantic model?

- the derivation tree here is the semantic model
- take the head as a predicate
- take its leaves as its arguments
  ⇒ a predicate-argument structure, or a first order term
  ⇒ We do not need anymore the manually crafted semantic rules
How is it possible?

- Lexicalization:
  - Each unit of the grammar is anchored by a lexical unit
- Minimal Semantic Principle:
  - Each tree must correspond to a minimal semantic unit (msu)
  - Predicate-argument cooccurrence principle
  - Each argumental leaves nodes of a tree has to be fully realized

This the so famous **Well formedness principles**
Where is the adjunction?

So far, we described only Lexicalized Substitution Tree Grammars. In order to fully lexicalized CFG, we need an optional operation of tree insertion: The adjunction.

- only a certain type of tree can be adjoined: The auxiliary tree (always prefixed by $\beta$)
- they must have a leaf node, the foot node with the same label than the root of the tree, the path from the root to the foot is called the spine
- let’s $\beta_1$ the auxiliary tree for the raising verb “to seem”:

```
VP
  \_  \_  
V   VPinf
  \_    \_
seems to *VP
```
Example of adjunction 1/3

Analysis for “Jane seems to be loved by Tarzan”:

S
  ↘
   ↘
     ↘
       ↘
         ↘
           ↘
             ↘
               ↘
                 ↘
                   ↘
                     ↘
                       ↘
                         ↘
                           ↘
                             ↘
                               ↘
                                 ↘
                                   ↘
                                     ↘
                                       ↘
                                         ↘
                                           ↘
                                             ↘
                                               ↘
                                                 ↘
                                                   ↘
                                                     ↘
                                                       ↘
                                                         ↘
                                                           ↘
                                                              ↘
                                                                ↘
                                                                  ↘
                                                                    ↘
                                                                      ↘
                                                                        ↘
                                                                             ↘
                                                                                 ↘
                                                                                     ↘
                                                                                       ↘
                                                                                        ↘
                                                                                                    ↘
                                                                                                        ↘
                                                                                                                  ↘
                                                                                                                    ↘
                                                                                                                        ↘
                                                                                                                             ↘
                                                                                                                                ↘
                                                                                                                                  ↘
                                                                                                                                            ↘
                                                                                                                                                ↘
                                                                                                                                                    ↘
                                                                                                                                                                                                            ↘
                                                                                                                                                                                                                                                                        ↘
                                                                                                                                                                                                                                                                            ↘
Example of adjunction 2 /3

Result: Derived Tree

```
S
  \__ NP
    \  Jane
  \  V
  \   \seems
  \   \  to
  \  VP
  \   \ VPinf
  \   \  VP
  \   \   \ V'
  \   \   \  be loved
  \   \   \   \ Prep
  \   \   \    by
  \   \    \ NP
  \    \    \ Tarzan
```
Results (suite) : Derivation tree for “Tarzan seems to love Jane” :

\[ \alpha_1'(love) \]

\[ 0-\alpha_2(Tarzan) \quad 1-\alpha_5(Jane) \quad \beta_2(seem) \]

- the link between \( \beta_2(seem) \) and \( \alpha_1'(love) \) do not reflect a dependency relation but a modifier one (it’s of course disputable)

- depending of the type of anchors (predicative or modifier), the adjunction link can be in the other direction.... Here are come the problems we will discuss the next time.
A lexicalized Formalism

- Grammar contains elementary trees (initial and auxiliary trees)
- Each tree is anchored by a lexical unit
- Two operations: substitution and adjonction
- As opposed to CFG, derivation tree and derived tree are not isomorphic anymore
- As opposed to LFG and HPSG, parsable in polynomial time
Operations on trees

- **The substitution** is a context free derivation of an initial tree to a leaf node of any elementary tree.

- **The adjunction** is a contextual insertion operation of an auxiliary tree within an elementary tree.
Derivation tree: describes the derived tree construction (i.e., the strict record of the operations used to parse a sentence)

Derived tree: syntactic structure of a sentence

Ex: Given the trees $\gamma$ and $\beta$, with $\beta$ adjoined on the node 1 of $\gamma$

![Diagram of derivation and derived trees]
Differences with other formalism:

- Features are atomic values only and then non reentrant

  ⇒ Features are used only to control the subcategorization frame and to restrict the number of possible derivations according to a feature value

  ⇒ because of the adjunction, features are splitted into 2 fields by node: the top field and the bottom field

  ⇒ No Slash feature in TAG
Adjunction : Update of the features
Substitution : Update of the features
Illustration on a feature $V_{\text{inf}=+}$:
If we want to be sure that $\beta_2$ adjoin on tree with an infinitive, we have to add some informations to this tree:

Tree $\beta_2$

```
  VP
 /\   \
|  \  |
V    VP
  \   |
     seems
       to $*VP(\text{top}:V_{\text{inf}=+}|\text{bot}:\emptyset)$
```
Illustration on a feature $\text{Vinf}=+$ : (suite)

Tree $\alpha'_1$

```
S
  /\    \\
 NP\  \ VP(top:Vinf=--|bot:Vinf=+)
    /   \       /
  V'    PP  NP\  \\
 |     |    |   |
 be loved Prep by
```

By having two different top and bottom values on the node VP, we force the adjunction of an auxiliary tree of root VP and whose foot node has the value top:Vinf=+, therefore no more unification clash.
Who do you think that Mary claim that Sarah liked?

Derivation Process

Long Distance Dependencies
Derivation Tree

\( \alpha_1 \) liked

- (1) \( \alpha_4 \) who
- (2) \( \beta_2 \) claim
  - (1) \( \alpha_3 \) Mary
  - (0) \( \beta_1 \) think
    - (1) \( \alpha_3 \) you
- (1) \( \alpha_5 \) sarah
Outline

- Is the Derivation Tree a good structure for Semantic?
- Is it Possible to Use both Derived Tree and Derivation Tree for that?
- What are Shared Forests, Derivation Forests or Dependancy Forests?
- What more can we do than Regular LTAG? (control, ellipsis..)
- What are Multi-Component TAG, Synchronous TAG and Metagrammars?