Tree Adjoining Grammar

**Theory and Practice**

Claire Gardent

LORIA
Campus Scientifique,
BP 239,
F-54 506 Vandoeuvre-lès-Nancy, France

Bangkok - 2006, October 19

---

**Outline**

- Formalism and Linguistic Theory
- Grammar development
- Text ⇒ Meaning: Semantic Construction
- Meaning ⇒ Text: Surface Realisation

---

**Tree Adjoining Grammar**

A Tree Adjoining Grammar (TAG) is a quadruple $G = \langle N, T, I, A, S \rangle$ such that

- $T$ and $N$ are the terminals and nonterminals categories,
- $I$ is a finite set of initial trees, and
- $A$ is a finite set of auxiliary trees,
- $S$ is a distinguished non terminal (the axiom)

The trees in $I \cup A$ are called **elementary** trees.
The trees of $G$ are combined using **adjunction** and **substitution**.

**TAG Elementary trees**

- **Initial trees** are elementary trees whose leaves are labelled with non terminal or terminal categories. Leaf nodes labelled with non terminal are substitution nodes marked with $\downarrow$
- **Auxiliary trees** are elementary trees whose with a designated **foot node**. The root and the foot nodes are labelled with the same category.
**TAG composition operations**

- **Adjunction** inserts an auxiliary tree into a tree (Adjunction is not allowed on substitution nodes)
- **Substitution** inserts a derived or elementary tree at the substitution node of a T AG tree.

**Derived and Derivation trees**

For each derivation in a T AG there is a corresponding **derivation tree**. This tree contains:

- nodes for all elementary trees used in the derivation, and
- edges for all adjunctions and substitutions performed throughout the derivation.

Whenever an elementary tree $\gamma$ is attached to the node at address $p$ in the elementary tree $\gamma'$, there is an edge from $\gamma'$ to $\gamma$ labeled with $p$.

**Example**

A T AG is **lexicalized** iff each elementary tree has at least one leaf with a terminal label.

In a lexicalized T AG, each elementary tree has at least one leaf with a terminal label.

- computationally interesting: if the grammar is finite, the number of analyses for a sentence is finite.
- linguistically interesting: each lexical item can be associated with the set of syntactic constructions it occurs in.
In a Feature-Based TAG:

- tree nodes are decorated with two feature structures called top and bottom
- unifications on these feature structures are performed during (substitution/adjunction) and after derivation (at the end of the derivation, unification of top and bottom FS at each node)

**Unifications during substitution**

**Unifications during adjunction**

**Why TAG?**

Some motivations for proposing yet another grammar formalism:

- Formal: TAG is mildly context sensitive hence can capture the type of crossing dependencies found in Dutch or in Swiss German
- Computational: TAGs are parsable in polynomial time (complexity $O(n^6)$)
- Linguistic: TAG provides an extended domain of locality which permits enforcing such linguistic principles as:
  - Predicate-Arguments Cooccurrence Principle: each predicative unit (verb, predicative noun, adjective) has in its elementary trees at least as many substitution sites as it has arguments
  - Semantic Anchoring Principle: each elementary tree is semantically non empty
  - Compositionality Principle: an elementary tree captures exactly one semantic unit
What does an LTAG look like?

- **Elementary trees** are extended projections of lexical items. A tree is said to be anchored by a word.
- Auxiliary trees encode the recursive parts of language (Factoring of recursion, FR)
- Elementary trees can be arbitrarily large, in particular (because of FR) they can contain elements that are far apart in the final derived tree (Extended domain of locality)

### Lexical predicates (nominal arguments)

The elementary tree of a **lexical predicate** (verb, noun, adjective) contains slots for all arguments of the predicate, for nothing more.

Example:

(1) John gives a book to Mary

```
S
  NP          VP
  |         ___
  |        VNP
  |      ___
  |     PP
  |    ___
gives P NP
  | ___
to NP
```

### Lexical predicates (light verbs)

*to make a comment* forms a predicate hence *make* and *comment* are included in the same elementary tree. *make* is the anchor, *comment* is the co-anchor.

```
S
  NP          VP
  |         ___
  |        VNP
  |      ___
  |     PP
  |    ___
to make P NP
  | ___
  |   comment
```

### Another case for co-anchors

```
S
  NP          VP
  |         ___
  |        VNP
  |      ___
  |     PP
  |    ___
participated P NP
  | ___
in NP
```
Lexical predicates (sentential arguments)

Example:

(2) John expected Mary to make a comment

The sentential object is realised as a foot node in order to allow for extractions:

(3) whom does John expect to come?

Modifiers (Adjectives)

(5) the good student

Modifiers (Adverbs and determiners)

Long distance dependencies

(4) a. who\(_i\) did John tell Sam that Bill likes \(t_i\)
b. who\(_i\) did John tell Sam that Mary said that Bill likes \(t_i\)
Implementing a TAG

- A large coverage TAG consists of several thousands of trees
- Maintaining, extending and debugging the grammar becomes extremely difficult
- Factorisation is important

XTAG (the TAG for English developed at UPenn) is structured into:

- a morphosyntactic lexicon
- a syntactic lexicon
- a set of tree schemas grouped into tree families

Morphosyntactic lexicon

Solution:

- use tree schemas (with a ♦ node where the lexical leaf will be attached as immediate daughter)
- associate verbs with tree schema names

\[
\begin{align*}
\text{aime} & \quad @\alpha n0Vn1 \
\text{écrit} & \quad @\alpha n0Vn1 \
\text{mange} & \quad @\alpha n0Vn1 \\
\alpha n0Vn1 : & \quad S\quad N\downarrow V\diamond N\downarrow \\
\end{align*}
\]

Syntactic factorisation

\[
\begin{align*}
aime & \quad @\alpha n0Vn1 \quad (\text{tense} = \text{present}, \text{pers} = 3, \text{number} = \text{sing}) \\
aiment & \quad @\alpha n0Vn1 \quad (\text{tense} = \text{present}, \text{pers} = 3, \text{number} = \text{plur}) \\
aimaient & \quad @\alpha n0Vn1 \quad (\text{tense} = \text{imparfait}, \text{pers} = 3, \text{number} = \text{plur}) \\
\end{align*}
\]

Problem: The fact that the various forms are all instances of the verb *aimer* is not captured.
### Syntaxe Lexicon

**Solution:**

- associate morphosyntactic variants with lemma
- associate lemma with verb schema

<table>
<thead>
<tr>
<th>Verb</th>
<th>Morphological Form</th>
<th>Syntactic Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>aime</td>
<td>@aimer (tense = present, pers = 3, number = sing)</td>
<td>aimer: @n0Vn1</td>
</tr>
<tr>
<td>aiment</td>
<td>@aimer (tense = present, pers = 3, number = plur)</td>
<td></td>
</tr>
<tr>
<td>aimaient</td>
<td>@aimer (tense = imparfait, pers = 3, number = plur)</td>
<td></td>
</tr>
</tbody>
</table>

### Tree Families

A tree family captures the various syntactic contexts in which a given verb type (e.g., transitive, intransitive) appears.

![Tree family example](image)

**But this is not enough ...**

- even with this type of grammar architecture, the number of tree schemas needed is in the several thousands
- this makes it very hard to debug and update the grammar
- hence, in actual TAG implementations, devices are used to factorise and share the information common to several tree schemas in the tree schema set:
  - Inheritance hierarchy of trees or tree descriptions
  - Lexical rules on trees
    - (Becker 93; Prolo 2002; Vijay-Shanker and Schabes 92; Candito 1999; Xia 2001; Gaiffe et al. 2002; Duchier et al. 2004.)
The XMG framework:
- is a higher level linguistic formalism and grammar compiler supporting semi-automatic grammar development,
- allows the linguist to capture linguistic generalisations (e.g., tree fragments) across grammatical structures (e.g., trees),

Using XMG to implement a TAG

The basic idea:
- Identify **recurrent tree fragments**
- **modularise** these tree fragments into **classes**
- **combine** classes (tree fragments) so as to generate TAG elementary trees
- Two axes for combination:
  - vertical axis: inheritance, sharing
  - horizontal axis: alternatives

Recurrent tree fragment example

Jean *mange* une pomme
La pomme que Jean *mange*

Vertical axis: structure sharing

John sleeps
- CanonicalSubject
- ActiveVerb
- IntransitiveVerb

John eats the apple
- CanonicalSubject
- ActiveVerb
- CanonicalObject
- TransitiveVerb

E.g., the tree fragments for a canonical subject occurs in all trees (active, passive, transitive, intransitive, ditransitive, etc.)
1. Structure sharing

Disjunction (choice) can be used to build tree families.

- **CanonicalSubject**: John eats the apple.
- **WhSubject**: Which apple does John eat?
- ... 

```
Subject = CanonicalSubject ∨ WhSubject

Subject → (CanonicalSubject ∨ WhSubject ∨ ...) 

n0V → (Subject ∧ ActiveVerbForm)

n0Vn1 → (Subject ∧ ActiveVerbForm ∧ Object)
```

Horizontal axis: alternatives

Disjunction (choice) can be used to build tree families.
2. Alternative choice

<table>
<thead>
<tr>
<th>Canonical_Subject</th>
<th>Active</th>
<th>Extracted_Subject</th>
</tr>
</thead>
<tbody>
<tr>
<td>S V'</td>
<td>S V'</td>
<td>S</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Canonical_Act</th>
<th>Extracted_Act</th>
<th>Intransitive_Act</th>
</tr>
</thead>
<tbody>
<tr>
<td>S N V'</td>
<td>S N V'</td>
<td>S N V'</td>
</tr>
</tbody>
</table>

The XMG (eXtensible MetaGrammar) formalism

- A language to describe tree fragments:

\[
\text{Description} ::= x \rightarrow y \mid x \rightarrow^+ y \mid x \rightarrow^* y \mid x \prec y \mid x \prec^+ y \mid x \prec^* y \mid x[f:E] \mid x(p:E)
\] (1)

- A language to combine tree fragments:

\[
\text{Class} ::= \text{Name} \rightarrow \text{Content} \tag{2}
\]

\[
\text{Content} ::= \text{Description} \mid \text{Name} \mid \text{Content} \lor \text{Content} \mid \text{Content} \land \text{Content} \tag{3}
\]

Example (1 / 2)

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanSubject</td>
<td>(X [cat : s] → Y [cat : v]) \land (X → Z (mark : subst) [cat : n]) \land (Z ∼ Y)</td>
</tr>
<tr>
<td>Active</td>
<td>(X [cat : s] \land Y (mark : anchor) [cat : v] \land X → Y)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tree fragment</th>
</tr>
</thead>
<tbody>
<tr>
<td>X [cat:s]</td>
</tr>
<tr>
<td>S N V'</td>
</tr>
<tr>
<td>S V'</td>
</tr>
<tr>
<td>Z \downarrow [cat:n] Y [cat:v]</td>
</tr>
</tbody>
</table>

Example (2 / 2)

Some trees for intransitive verbs (e.g., the lexical item sleeps)

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanSubject \land Active</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tree fragment</th>
</tr>
</thead>
<tbody>
<tr>
<td>X [cat:s]</td>
</tr>
<tr>
<td>N* S</td>
</tr>
<tr>
<td>N \downarrow V'</td>
</tr>
</tbody>
</table>

(e.g. the boy sleeps)

<table>
<thead>
<tr>
<th>Tree fragment</th>
</tr>
</thead>
<tbody>
<tr>
<td>X [cat:s]</td>
</tr>
<tr>
<td>N S</td>
</tr>
<tr>
<td>N \downarrow V'</td>
</tr>
<tr>
<td>N*</td>
</tr>
</tbody>
</table>

(e.g. the boy who sleeps)
Variable scope

- The scope of the variables used within the descriptions is local by default.
- Possibility to explicitly extend variable scope by means of Import / Export declarations.
- Possibility to have global scope variables

Constraining admissible structures

Problem: it is difficult to constrain the grammar specification so as to produce only valid grammar trees.

XMG provides several automatic (optional) mechanisms for constraining the output trees produced by the compiler.

Three types of constraints:
1. Formal constraints
2. Operational constraints
3. Language-dependent constraints

Formal constraints

- Constraints assuring that the trees generated by the compiler are regular TAG trees.
- Besides being trees, the output structures must respect the following criteria:
  - each node has a unique category label,
  - each leaf node is marked either as subst, as foot or as anchor,
  - the category of a foot node is identical to that of the root node,
  - etc.

Operational constraint (1 / 3)

- Constraints controlling the combinations of tree fragments
- Constraints based on a tree logic integrating node colors
- In a tree description, each node variable is either Black, Red or White.
- A model for a colored tree description is a saturated tree.
- A saturated tree is a tree containing only red or black nodes
- The coreference of two node variables is constrained by the following rules:

\[
\begin{align*}
\circ_W + \circ_W &= \circ_W \\
\bullet_R + \circ_W &= \bullet_R \\
\bullet_R + \bullet_R &= \perp \\
\bullet_R + \{ \circ_W ; \bullet_R \} &= \perp
\end{align*}
\]
Operational constraint (2 / 3)

Colors are useful because they permit:

- Avoiding node naming issues: no names needed, the fusion of two nodes is controlled by color rather than by identical global or semi-global names.

- Simplifying the grammar specification: node equations are replaced with implicit coloured node identifications.

- Reusing the same tree fragment several times.

Operational constraint (3 / 3)

Example:

\[
\begin{array}{c}
S \circ \text{w} \\
N \bullet r \\
V \circ \text{w} \end{array} \\
\begin{array}{c}
\text{(SubjectCan)} \\
\lor \\
N \bullet r \\
S \circ \text{w} \end{array} \\
\begin{array}{c}
N \bullet r \\
S \circ \text{w} \\
V \circ \text{w} \\
\text{(Active)} \\
\land \\
N \bullet r \\
\end{array}
\]

Language-dependent constraints (1 / 2)

- For French, the ordering and uniqueness of clitics.

  - (Perlmutter, 70):

    \begin{itemize}
    \item first they appear in front of the verb in a fixed order according to their rank (a-b) and second two different clitics in front of the verb cannot have the same rank (c).
    \end{itemize}

- For instance the clitics *le, la* have the rank 3 and *lui* the rank 4 (rank is a node property).

  \begin{itemize}
  \item (a) Jean le$_3$ lui$_4$ donne
        John gives it to him
  \item (b) *Jean lui$_4$ le$_3$ donne
        *John gives to him it
  \item (c) *Jean le$_3$ la$_3$ donne
        *John gives it it
  \end{itemize}
Language-dependent constraints (2 / 2)

\[
\begin{array}{c}
S & \xleftarrow{\text{Cl}_{1}^{3}} & V' \\
N_{\downarrow} & \wedge & \text{(Jean)} \\
V' & \xleftarrow{\text{Cl}_{1}^{4}} & V \\
\end{array}
\quad
\begin{array}{c}
V' & \xleftarrow{\text{Cl}_{1}^{4}} & V \\
\text{(le)} \\
\end{array}
\quad
\begin{array}{c}
\text{(lui)} \\
V' & \xleftarrow{\text{Cl}_{1}^{4}} & V \\
\end{array}
\]

⇒

\[
\begin{array}{c}
S & \xleftarrow{\text{Cl}_{1}^{3}} & \text{Cl}_{1}^{4} & V' \\
N_{\downarrow} & \wedge & \text{(Jean le lui donne)} \\
\text{Cl}_{1}^{4} & \wedge & \text{V'} \\
V & \xleftarrow{\text{Cl}_{1}^{4}} & V' \\
\end{array}
\quad
\begin{array}{c}
S & \xleftarrow{\text{Cl}_{1}^{3}} & \text{Cl}_{1}^{4} & V \\
N_{\downarrow} & \wedge & \text{(Jean lui le donne)} \\
\text{Cl}_{1}^{4} & \wedge & \text{Cl}_{1}^{3} \\
V & \xleftarrow{\text{Cl}_{1}^{4}} & V \\
\end{array}
\]

XMG in practice

- The XMG system has been used successfully to compute a core TAG for French (6,000+ trees computed from a description containing 293 classes) FraG.
- The compilation of a TAG with more than 6,000 trees takes about 15 min with a P4 processor 2.6 GHz and 1 GB RAM.
- XMG is released under the terms of the GPL-like CeCILL license and can be freely downloaded at http://sourcesup.cru.fr/xmg

XMG Features

- The XMG framework supports a fully declarative language for specifying tree based grammar.
- XMG is being used to develop both TAG (Nancy, France; Tuebingen, Germany; Arabic, USA) and Interaction grammars (G. Perrier)
- XMG also allows for semantic information to be integrated in the grammar (see below)

Semantic construction

- Parsing associates with a sentence a syntactic structure
- It can also associate with a sentence a semantic representation (cf. HPSG ERG, LFG F-structure and glue semantics)
- Semantic Construction systematically assigns a meaning representation to a constituent
  
  Example 1:  \[\text{John loves Mary} \Rightarrow \text{love(j,m)}\]
  
  Example 2:  \[\text{loves Mary} \Rightarrow \lambda x.\text{love}(x,m)\]
Montague style semantic construction

Montague provided us with a systematic way of associating syntactic constituents with semantic representations:

- The **semantic representation** of a word is a lambda term.
- Syntactic rules are paired with semantic rules.
- A semantic rule specifies how the semantics of the sub-constituents described by the paired syntactic rule combine to yield the semantics of the mother constituent.
- Semantic composition is done by **beta reduction**.

### Semantic construction in large scale unification grammars

```
cat S 
  [pred run j] 

[cat NP 
  [pred j] 
  [arg Jon]] 

S \rightarrow NP VP 
S.sem = VP.sem 
VP.sem.arg = NP.sem
```

### Semantic construction in TAG

- Earlier proposals based semantic construction on the derivation tree.
- Idea: the derived tree records constituent structure; the derivation tree represents predicate-argument dependencies.
- Nodes in the tree represent elementary trees and are associated with the semantic representation of the corresponding lexical item.
- Tree edges reflect the substitutions and adjunctions used to combine elementary tree. An upward arrow indicates an adjunction, downward arrows indicate substitution.

```
S_{\lambda x. run(x)} \equiv_{\beta} run(j) 
NP_j VP_{\lambda x. run(x)} 
Jon runs 

S \rightarrow NP VP 
Sem_S = Sem_{VP}(Sem_{NP})
```
Semantic construction in TAG (example)

(6) Bill often sees Mary.

```
    see
   / \                   /
  bill  often  mary

often(see(bill)(mary))
```

But ...

the approach fails to extend to:
- Quantifiers
- Questions
- Raising

Raising

(7) John claims Bill is likely to win

```
  to_win
  /   \
  1   2
Bill  claims  is_likely
       /   \
      1   2
John
```

Problem: there is no connection between Bill and is likely

Quantifiers

```
barks
dog
   /
  every
```

(a)

Problem: there is no connection between the quantifier (every) and its scopal argument (barks).
Questions

(8) Who does Paul think John said Bill like?

<table>
<thead>
<tr>
<th>liked</th>
<th>who</th>
<th>said</th>
<th>Bill</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>think</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Paul</td>
<td>does</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b)

Problem: there is no connection between the scoping operator who and its scopal argument think.

(9) Mary, Paul claims, John seems to love

<table>
<thead>
<tr>
<th>to love</th>
<th>Mary claims seems John</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paul</td>
<td></td>
</tr>
</tbody>
</table>

Problem: there is no connection between claim and seem.

Multiple adjunctions

Other TAG approaches to semantic construction

- Use multi-component (Kallmeyer and Joshi 2002) or synchronous TAG (Shieber 2006)
- Use glue logic (Crouch and Frank)
- Semantic construction based on the derived tree (Gardent and Kallmeyer 2003)

Semantic representation language

- **flat semantics** formulas with labels and unification variables
  
  ```
  John \ i_j : john(j)  
  Mary \ i_m : mary(m)  
  loves \ i_l : loves(X, Y)  
  ```

- composition operation: **union modulo unification**
  
  ```
  John loves Mary \ i_l : love(j, m), i_j : john(j), i_m : mary(m)  
  ```
Unification based Semantic construction in FTAG

- Each elementary tree is associated with a formula $\phi$ representing its meaning (flat underspecified semantics)
- Elementary tree nodes are decorated with unification variables occurring in $\phi$
- The meaning of a derived tree is the union of the meanings associated with the elementary trees under the unifications made during processing

Exemple: “John loves Mary”

$S \rightarrow NP_X \rightarrow VP \rightarrow NP_Y \rightarrow NP_m$

$\text{name}(j, john) \ l_0: \text{love}(X, Y) \ \text{name}(m, mary)$

$\Rightarrow l_0: \text{love}(j, m), \text{name}(j, john), \text{name}(m, mary)$

Quantification

In XMG

$S \rightarrow \text{Det} \rightarrow N_{x, S_1} \rightarrow N_{x_1, l_2} \rightarrow V \rightarrow V \rightarrow \text{barks}$

$\text{every} \ l_0: \forall(x, S_1, h_2), \ h_2 \geq S_2 \ \text{dog} \ l_2: \text{bark}(X_2)$

$l_1: \text{dog}(X_1)$

$\Rightarrow l_0: \forall(x, h_1, h_2), \ h_2 \geq l_2, \ l_1: \text{dog}(x), \ l_2: \text{bark}(x)$

- XMG supports multi-level descriptions
- In particular, a class can specify both syntax (a tree description) and semantics (a flat semantics formula)
- XMG was used to develop a core grammar for French containing semantic information, SemFraG.
A verbal tree with semantic information

Transitive:  Subject:  Active:  UnaryRel:

Distinct realisations of the same argument

Disjunction enables the description of alternatives (distinct argument realisations, alternations)

The passive verbal tree

The passive verbal tree

Outline  Formalism  Grammar  Semantic Construction  Surface Realisation  Summary  Outline  Formalism  Grammar  Semantic Construction  Surface Realisation  Summary
Factorisation in XMG

- through the many uses of the same syntactic or semantic fragment (e.g., Subject fragment used in 35 verb classes)
- through the compact description of alternations (in the sense of B. Levin) and of their associated semantics: one semantics, \( n \) alternations.

Semantic construction

Two ways of performing semantic construction based on the derived tree:

- During parsing
- After parsing

Semantic construction during parsing

- semantic formulae represented by complex FS sharing feature values with tree nodes semantic indices:
  
  \[
  \begin{bmatrix}
  \text{semf} \\
  \text{label} \quad l_r \\
  \text{pred} \quad \text{runs} \\
  \text{arg1} \quad X
  \end{bmatrix}
  \]

- this semantic formula labels the anchor node and thus will never be modified except through unification (no merging of this node during parsing)
- semantic indices are unified when adjunction and substitution take place
- the semantics resulting from parsing is the union of the \( \text{semf} \) features at the anchor node of the elementary trees involved
But ...

there are a number of disadvantages:

- Theoretical: the approach is outside TAG since formally a
  TAG may not contain recursive feature structures
- Computational: integrating semantic information into the
  trees might decrease the amount of sharing possible during
  tabulation based parsing
- Practical: TAG parsers often cannot deal with recursive
  feature structures

Semantic construction after parsing
(Kallmeyer and Romero, 2004)

A 3 step process:

- extraction of the semantic information included in the
  grammar → semantic lexicon
- pure syntactic parsing on the basis of the grammar without
  any semantic feature → derivation forest
- construction of the semantic representation based on:
  - the semantic lexicon and
  - the derivation tree or forest

Step 1: Extraction of the semantic lexicon

Idea: extract from a semantic TAG all semantic information and
store it into a semantic lexicon which associates a lemma and a
syntactic tree name with a ⟨ Semantic Tree, Semantic formula ⟩ pair.

The extraction process:

- For each tree in the grammar $G$, number the tree nodes with
  their gorn address → new grammar $G_n$
- For each tree in the grammar $G_n$:
  (a) create a purely syntactic FB-TAG tree
  (b) create an entry in the semantic lexicon composed of the lemma, the tree name, the semantic
  representation, and the semantic features associated with numbered tree nodes

Example

$$S \rightarrow NP \quad VP$$

$$NP \quad VP$$

$$[\text{top}]$$

$$\text{gen} \quad \text{num} \quad \text{idx} \quad \text{bot}$$

$$[\text{top}]$$

$$\text{gen} \quad \text{num} \quad \text{idx} \quad \text{bot} \quad \text{X}$$

$$\text{gen} \quad \text{num} \quad \text{idx} \quad \text{bot} \quad \text{X}$$

$$\text{generates}$$

$$\text{runs}$$
Step 2: Syntactic Parsing

Parsing uses the automatically extracted syntactic grammar:

- after extraction of the semantic lexicon, the resulting grammar consists of a purely syntactic FB-TAG
- we use it to parse sentences with a FB-TAG parser (namely the DyALog system)
- as a result, we obtain a derivation forest, i.e. a shared representation of all possible derivation trees of the given sentence

Step 3: Computing Semantic Representations

To compute the semantic representation(s) of a sentence, we traverse the derivation forest and recursively combine the associated semantic trees thus performing unifications on these trees and indirectly on the associated semantic formula.
More precisely

- **top-down traversal** of the derivation forest

- for each node of the derivation forest (corresponding to an elementary tree), retrieve its **semantic entry** i.e., its entry in the semantic lexicon

- for each derivation edge, apply the unifications corresponding to the operation (substitution or adjunction) associated with that edge

- collect the semantic representations of each entry of the semantic lexicon (these representations contain unified variables)

Example derivation

- Sentence to parse: *John runs*

- Derivation forest

  ![Derivation forest diagram](image)

- Content of the semantic lexicon:

  ![Semantic lexicon diagram](image)
Example derivation

\[ l_j: \text{john}(\mathbb{I}_j) \]
\[ \mathbb{I}_r: \text{runs}(\mathbb{I} X) \]

Implementation

- A core TAG for French includes semantic information: roughly 3000 non-anchored FB-TAG trees
- Semantic construction is done after parsing
- Parsing is done with a tabular parser yielding a derivation forest
- Semantic construction works on the derivation forest
Surface Realisation

A grammar can associate with a sentence a syntactic structure and a semantic representation.

Semantic Construction: Text ⇒ Meaning

Example 1: John loves Mary ⇒ love(j,m)
Example 2: loves Mary ⇒ λx.love(x,m)

Surface Realisation: Meaning ⇒ Text

Example 1: John loves Mary ⇐ love(j,m)
Example 2: loves Mary ⇐ λx.love(x,m)

GenI: a TAG based surface realiser

- Tabular and bottom-up
- + Optimisations
- + Parameterisation for paraphrase selection

Algorithm outline

Three main steps:

Lexical selection step: Select trees whose semantics subsumes the input semantics.

Realisation step: Perform substitutions or adjunctions between selected. Substitutions are applied first, then adjunctions.

Success lookup step: Return the trees which are syntactically complete and whose semantics matches the input semantics.

Algorithm (simplified)

1. Input: the grammar (G), a semantic representation (Sem)
2. Declaration: Chart, Agenda, AgendaA ← 0
3. Initialisation of the agenda: all trees in G whose semantic subsumes part of Sem are added to the agenda
4. Processing the agenda (substitutions): for each tree I in Agenda which can combine by substitution with a tree J in Chart, add IJ to Agenda; the trees with no empty substitution node but with a foot node are moved to AgendaA
5. Reinitialisation: Agenda ← Chart, Chart ← AgendaA
6. Processing of agenda (Adjunctions)
7. Output: all the strings which are the yield of a syntactically complete tree whose semantic is Sem
Example

$\text{Sem} = \{\text{campe}(s, j), \text{jean}(j), \text{dans}(s, l), \text{paris}(l)\}$

Lexical lookup phase

\[
\begin{array}{c}
N_j \\
\text{Jean} \\
\text{jean}(j) \\
N_p \\
\text{Paris} \\
\text{paris}(p) \\
V \\
\text{campe}(e, x_1) \\
\text{Prep} \\
\text{dans}(e_1, x_1) \\
V^*_{x_1} \\
\text{GP} \\
\text{dans}(e_1, x_1) \\
\end{array}
\]

Substitutions

<table>
<thead>
<tr>
<th>Agenda</th>
<th>Chart</th>
<th>Combination</th>
<th>AgendaA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jean, campe, dans, Paris, JeanCampe</td>
<td>Jean</td>
<td>$(\text{campe}, \text{Jean})$</td>
<td></td>
</tr>
<tr>
<td>dans, Paris, JeanCampe</td>
<td>Jean, campe</td>
<td>$(\text{dans}, \text{Paris})$</td>
<td>dansParis</td>
</tr>
<tr>
<td>JeanCampe, dansParis</td>
<td>Paris, JeanCampe</td>
<td>这几个内容没有显示出来</td>
<td></td>
</tr>
</tbody>
</table>

Example (Ct'ed)

Adjunctions

<table>
<thead>
<tr>
<th>Agenda</th>
<th>Chart</th>
<th>Combination</th>
<th>AgendaA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jean, Paris, JeanCampe, Paris</td>
<td>dansParis</td>
<td>这几个内容没有显示出来</td>
<td></td>
</tr>
<tr>
<td>JeanCampe</td>
<td>Paris, JeanCampe</td>
<td>这几个内容没有显示出来</td>
<td></td>
</tr>
<tr>
<td>JeanCampe, dansParis</td>
<td>JeanCampe, Paris, JeanCampe</td>
<td>这几个内容没有显示出来</td>
<td></td>
</tr>
<tr>
<td>JeanCampe, Paris, JeanCampe</td>
<td>JeanCampe, Paris, JeanCampe</td>
<td>这几个内容没有显示出来</td>
<td></td>
</tr>
</tbody>
</table>

Optimisations

- Substitutions (Adjunctions)
- Elimination of redundant structures
- Polarity based filtering
Multiple modifiers

\[ \text{fierce}(x), \text{little}(x), \text{cat}(x), \text{black}(x) \]

For \( n \) modifiers, \( n! \) intermediate structures:

\[ \text{fierce cat, fierce black cat, little cat, little black cat, fierce little cat, black cat} \]

multiplied by the context:

\[ \text{fierce cat, fierce black cat, little cat, little black cat, fierce little cat, black cat} \]
\text{the fierce cat, the fierce black cat, the little cat, the little black cat, the fierce little cat, the black cat} \]
\text{the fierce cat runs, the fierce black cat runs, the little cat runs, the little black cat runs, the fierce little cat runs, the black cat runs} \]

Substitutions \( \langle \) Adjunctions

Adjunction restricted to syntactically complete trees

The \( n! \) intermediate structures are not multiplied out by the context:

\[ \text{the cat runs, the fierce cat runs, the fierce black cat runs, the little cat runs, the little black cat runs, the fierce little cat runs, the black cat runs} \]

Elimination of redundant structures

The same syntactic structure can be constructed in different ways:

- Distinct relative ordering of substitutions within a tree
- Distinct relative ordering of adjunctions within a tree
  \( \Rightarrow \) Only one operation order allowed (left to right)
- Distinct relative ordering of multiple adjunctions to a given node
  \( \Rightarrow \) No adjunction on foot node

Polarity based filtering

- The search space created by the lexical lookup phase is exponential in the number of literals present in the input semantics
- Nb of possible combinations: \( \prod_{1 \leq i \leq n} a_i \) avec:
  \( a_i \), the degree of lexical ambiguity of the \( i \)-th literal and \( n \), the number of literals in the input semantics.
- Polarity based filtering filters out all combinations of lexical items which cannot result in a grammatical structure
Example (Ct’ed)

- The grammar trees are associated with polarities reflecting their syntactic resources and requirements.
- All combination of trees covering the input semantics but whose polarity is not zero is necessarily syntactically invalid and is therefore filtered out.
- A finite state automata is built which represent the possible choices (transitions) and the cumulative polarity (states).
- The paths leading to a state with polarity other than zero are deleted (automata minimisation).

Ambiguity

Many combinations are syntactically incompatible. The goal is to detect these combinations and filter them out.

Semantic Representation: tableau(t), cout(t,g), grand(g)

<table>
<thead>
<tr>
<th></th>
<th>tableau(t)</th>
<th>cout(t,g)</th>
<th>grand(g)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tableau</td>
<td>Ttableau</td>
<td>Tcout</td>
<td>Test elev</td>
</tr>
<tr>
<td>Peinture</td>
<td>Tpeinture</td>
<td>Tcoute</td>
<td></td>
</tr>
<tr>
<td>Cher</td>
<td>Tcher</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Lexical look up:

Tableau coûte cher
Le coût du tableau est élevé
* peinture, cout, test elev

Polarity filtering (Perrier 2003)

We associate each tree in the grammar with a set of polarities representing syntactic resources and requirements.
We construct an automaton to represent the possible combinations of lexical items and their polarities.

We then perform automaton minimisation to remove the incompatible combinations.

Paraphrase selection

- The generator can be parameterised by one (or more) restrictor(s)
- Restrictor ::= -Synt:SemIdex
- The grammar trees are (automatically) associated with polarities of the form +Synt:SemIdex
- Polarity based filtering eliminate all tree combinations which fail to satisfy the property expressed by the restrictor.
Exemple

regarde(e,j,m), jean(j), marie(m)
-cleft:j  
-C'est Jean qui regarde Marie
-declarative:e  
-Jean regarde Marie
-interrogative:e  
-Jean regarde-t'il Marie?

Implementation and Experimentation

- Implemented in Haskell (Carlos Areces, Eric Kow)
- Graphic interface
- Debugging and testing facilities (batch processing, step-wise visualisation of the different data structures)

Test cases of Carroll et al. 1999 and Koller and Striegnitz 2002.

(10) The manager in that office interviews a new consultant from Germany.

Le directeur de ce bureau auditionne un nouveau consultant d’Allemagne.

(11) The manager organizes an unusual additional weekly departmental conference.

Le directeur organise un nouveau séminaire d’équipe hebdomadaire spécial.

Polarity filtering and Paraphrase selection results

Grammar of 2751 trees.

<table>
<thead>
<tr>
<th></th>
<th>Example 10</th>
<th>Example 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Possible combinations</td>
<td>1 377</td>
<td>1 003 833</td>
</tr>
<tr>
<td>Combinations explored</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>Sentences (w/o selection)</td>
<td>6</td>
<td>36</td>
</tr>
<tr>
<td>Sentences (with selection)</td>
<td>2</td>
<td>12</td>
</tr>
</tbody>
</table>
Polarity filtering and Paraphrase selection results

Chart size is reduced by 77% and 87%

<table>
<thead>
<tr>
<th>Optimisations</th>
<th>Example 10</th>
<th></th>
<th>Example 11</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Chart sz</td>
<td>Time</td>
<td>Chart sz</td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>none</td>
<td>522</td>
<td>0.9 s</td>
<td>362</td>
<td>2.1 s</td>
</tr>
<tr>
<td>pol</td>
<td>125</td>
<td>0.2 s</td>
<td>46</td>
<td>0.7 s</td>
</tr>
<tr>
<td>pol + factor</td>
<td>77</td>
<td>0.3 s</td>
<td>30</td>
<td>1.1 s</td>
</tr>
<tr>
<td>pol + select</td>
<td>24</td>
<td>0.1 s</td>
<td>10</td>
<td>0.3 s</td>
</tr>
<tr>
<td>Carroll</td>
<td>n/a</td>
<td>1.8 s</td>
<td>n/a</td>
<td>4.3 s</td>
</tr>
<tr>
<td>Koller</td>
<td>n/a</td>
<td>1.4 s</td>
<td>n/a</td>
<td>0.8 s</td>
</tr>
</tbody>
</table>

Polarity filtering and Tabulation results

Decreases the chart size by 83%.

(12) The fact that the manager organizes a conference annoys the consultant.

Que le directeur organise un seminaire ennuie le consultant

<table>
<thead>
<tr>
<th>Optimisations</th>
<th>Chart size</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>pol</td>
<td>1258</td>
<td>0.61 s</td>
</tr>
<tr>
<td>pol + factor</td>
<td>219</td>
<td>0.47 s</td>
</tr>
</tbody>
</table>

Related approaches

Improving the efficiency of surface realisation:

- HPSG based approach (Carroll et al. 99; Carroll and Oepen 06)
- greedy strategy (White 04)
- Constraint based approach of (Koller and Striegnitz 2002)

Conclusion

- TAG departs from other linguistic framework by its linguistic and computational properties: extended domain of locality, factoring out of recursion, mildly context-sensitive
- Semantic construction in TAG is still an open research area
- Surface realisation benefits from the substitution/adjunction distinction
Acknowledgments

Many thanks to Benoit Crabbé, Eric Kow, Laura Kallmeyer and Yannick Parmentier for allowing me to recycle some of their slides.