Generation and selection of grammatical paraphrases

Claire Gardent (CNRS/LORIA, Nancy)
and
Eric Kow (INRIA/LORIA, Nancy)

25 April 2005
Paraphrases

*Paraphrases* are sentences with the same core meaning e.g.,

(1) a1. John borrowed a book from Mary.

*Grammatical paraphrases* vary wrt alternation (e.g., passive/active) and argument realisation (cliticisation, extraction, etc.)

(2) b1. John loves Mary.
    b2. ≡ Mary is loved by John.

(3) b1. John looks at Mary.
    b2. ≡ It is Mary that John looks at.
Paraphrases and Natural Language Processing

Analysis: String $\rightarrow$ Meaning

- Need to recognise different formulations of the same core content
  - Information extraction
  - Question answering
  - Summarisation

Generation: Meaning $\rightarrow$ String

- Need to generate all paraphrases
- and to select one
Generating paraphrases

- Combinatorial issue: NL allows many paraphrases
- Linguistic issue: not all paraphrases are appropriate for all contexts:

(4) c1. Peter persuaded John to borrow a book from Mary.
    c2. $\neq$ Peter persuaded Mary to lend a book to John.
(5) d1. It is not Sarah, it is MARY, John looks at.
    d2. ?? It is not Sarah, John looks at Mary.
Generating paraphrases

A surface realiser must

- be able to produce all paraphrases (to preserve completeness)
- be highly optimised (to deal with the combinatorics)
- be able to chose the best paraphrase (to provide a single output)
Proposal

- a **generative** approach: the grammar covers (generates) all paraphrases
- **optimised** to reduce the combinatorics early on in the surface realisation process (by drastically reducing the search space before search starts)
- and **selective**: supports the selection of those paraphrases that conform to the given contextual restrictions
Overview

- The grammar (a Feature Based Tree Adjoining Grammar)
- The basic surface realisation algorithm
- Optimisations
- Selection
- Implementation and experimentation
- Related approaches
- Future and related work
The grammar – syntactic dimensions

Lexicalised Feature Based Tree Adjoining Grammar (FTAG)

- Set of trees (initial and auxiliary)
- Each tree is anchored with a word
- The nodes of the trees are labelled with two feature structures (TOP and BOTTOM)
- Two combining operations: substitution and adjonction
Jean court vite

Varsaw, 25 April 2005
The grammar – semantic dimension

Unification based semantic construction

- The trees are associated with semantic representations in which the semantic parameters are unification variables
- The (appropriate) tree nodes are labelled with semantic parameters
- The semantic of a derived tree is the union of the semantic representations of the trees entering in its derivation modulo unification
Exemple

\[ \Rightarrow \text{jean}(j), \text{marie}(m), \text{aime}(e, j, m) \]
Linguistic coverage

- Basic subcategorisation frames
- Redistributions: active, passive, middle, reflexive, impersonal, impersonal passive
- Argument realisation: cliticisation, extraction, omissions, word order variation
Surface realisation algorithm

- Tabular and bottom-up
- + Optimisations
- + Parameterisation for paraphrase selection
Basic algorithm (simplified)

1. Input: the grammar ($G$), a semantic representation ($Sem$)
2. Declaration: Chart, Agenda, AgendaA $\leftarrow 0$
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 $\leftarrow$ Chart, Chart $\leftarrow$ 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

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

Lexical lookup phase

[Diagram showing the lexical lookup phase with nodes and edges labeled accordingly.]
Substitutions

<table>
<thead>
<tr>
<th>Agenda</th>
<th>Charte</th>
<th>Combination</th>
<th>AgendaA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jean, campe, dans, Paris</td>
<td>Jean</td>
<td>↓(campe, Jean)</td>
<td></td>
</tr>
<tr>
<td>campe, dans, Paris</td>
<td>Jean, campe</td>
<td>↓(dans, Paris)</td>
<td></td>
</tr>
<tr>
<td>dans, Paris, JeanCampe</td>
<td>Jean, campe, dans</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Paris, JeanCampe</td>
<td>Jean, campe, dans, Paris</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JeanCampe, dansParis</td>
<td>Jean, campe, dans, Paris</td>
<td></td>
<td>dansParis</td>
</tr>
</tbody>
</table>
Example (Ct’ed)

Adjunctions

<table>
<thead>
<tr>
<th>Agenda</th>
<th>Charte</th>
<th>Combination</th>
<th>AgendaA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jean, Paris, JeanCampe</td>
<td>dansParis, Jean</td>
<td>*(JeanCampe, dansParis)</td>
<td></td>
</tr>
<tr>
<td>Paris, JeanCampe</td>
<td>dansParis, Jean</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JeanCampe</td>
<td>dansParis, Jean, Paris, JeanCampe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JeanCampeDansParis</td>
<td>dansParis, Jean, Paris, JeanCampe, JeanCampeDansParis</td>
<td></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:

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

**the fierce cat, the fierce black cat, the little cat, the little black cat, the fierce little cat, the black cat**

**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 < Adjunctions

Adjunction restricted to syntactically complete trees

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

*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
  ⇒ Only one operation order allowed (left to right)
- Distinct relative ordering of multiple adjunctions to a given node
  ⇒ 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

Semantic Representation: \text{tableau}(t), \text{cout}(t,g), \text{grand}(g)

<table>
<thead>
<tr>
<th>tableau(t)</th>
<th>cout(t,g)</th>
<th>grand(g)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_{\text{tableau}}$</td>
<td>$\tau_{\text{cout}}$</td>
<td>$\tau_{\text{est eleve}}$</td>
</tr>
<tr>
<td>$\tau_{\text{peinture}}$</td>
<td>$\tau_{\text{coute}}$</td>
<td>$\tau_{\text{cher}}$</td>
</tr>
</tbody>
</table>

Le tableau coûte cher
Le coût du tableau est élevé

* $\tau_{\text{peinture}}, \tau_{\text{coute}}, \tau_{\text{est eleve}}$
Example (Ct’ed)

- The grammar trees are associated with polarities reflecting their syntactic resources and requirements.

- All combinations 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).
Example (Ct’ed)

<table>
<thead>
<tr>
<th>tableau(t)</th>
<th>cout(t,g)</th>
<th>grand(g)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_{tableau}$ +1np</td>
<td>$\tau_{cout}$</td>
<td>$\tau_{est~eleve}$ -1np</td>
</tr>
<tr>
<td>$\tau_{peinture}$ +1np</td>
<td>$\tau_{coute}$ -1np</td>
<td>$\tau_{cher}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>tableau(t)</th>
<th>cout(t,g)</th>
<th>grand(g)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_{tableau}$ +1np</td>
<td>$\tau_{cout}$</td>
<td>+1np</td>
</tr>
<tr>
<td>$\tau_{peinture}$</td>
<td>$\tau_{coute}$ 0np</td>
<td>$\tau_{cher}$ 0np</td>
</tr>
<tr>
<td>$\tau_{test~eleve}$</td>
<td>$\tau_{coute}$ 0np</td>
<td>-1np</td>
</tr>
</tbody>
</table>
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), jeann(j), marie(m)

cleft:j
declarative:e
interrogative:e

C’est Jean qui regarde Marie
Jean regarde Marie
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)
Implementation and Experimentation

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

(6) The manager in that office interviews a new consultant from Germany.  
   *Le directeur de ce bureau auditionne un nouveau consultant d’Allemagne.*

(7) 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 6</th>
<th>Example 6</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>

Varsaw, 25 April 2005
Polarity filtering and Paraphrase selection results

Chart size is reduced by 77% and 87%

<table>
<thead>
<tr>
<th>Optimisations</th>
<th>Example 6 Chart sz</th>
<th>Example 7 Chart sz</th>
<th>Time</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>522</td>
<td>362</td>
<td>0.9 s</td>
<td>2.1 s</td>
</tr>
<tr>
<td>pol</td>
<td>125</td>
<td>46</td>
<td>0.2 s</td>
<td>0.7 s</td>
</tr>
<tr>
<td>pol + factor</td>
<td>77</td>
<td>30</td>
<td>0.3 s</td>
<td>1.1 s</td>
</tr>
<tr>
<td>pol + select</td>
<td>24</td>
<td>10</td>
<td>0.1 s</td>
<td>0.3 s</td>
</tr>
<tr>
<td>Carroll</td>
<td>n/a</td>
<td>n/a</td>
<td>1.8 s</td>
<td>4.3 s</td>
</tr>
<tr>
<td>Koller</td>
<td>n/a</td>
<td>n/a</td>
<td>1.4 s</td>
<td>0.8 s</td>
</tr>
</tbody>
</table>

Varsaw, 25 April 2005
Polarity filtering and Tabulation results

Decreases the chart size by 83%.

(8) 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>

Varsaw, 25 April 2005
Related approaches

Improving the efficiency of surface realisation:

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

- build a complete syntactic skeleton before modifiers are handled
- The “adjunction after substitution” idea is inspired from this proposal
- Extracted modifiers as in *Which office did work in?* need specific treatment
- in TAG, all modifiers are treated using adjunction so that no specific treatment is required for extracted ones.
- no *global* optimisation
White 2004

- complete NPs are first built before they are combined with the verb
- also feasible in TAG (adjunction would then apply on specific sets of lexical entries and the results combined with the verb)
- Compare the relative efficiency of both approaches within the TAG framework?
- uses n-gram scores versus polarities to reduce search space
- Best = most frequent vs. Best = most appropriate
The subset of the TAG grammar which is used for a given realisation task is translated into a set of lexical entries in a dependency grammar defining well formed TAG derivation trees. This set of entries is then parsed by an efficient constraint-based dependency parser thus producing the derivation trees associated by the grammar with the set of input lexical entries. A post processing phase produces the derived trees on the basis of the derivation trees output by the first step.
Koller and Striegnitz 2004

- *Global* optimisation: well formed derivation trees vs syntactically invalid combinations
- Constraint propagation versus on finite state techniques.
- Koller et al. explicitly ignores feature information
- Compare run times once feature structures are taken into account?
- The postprocessing step producing derived trees from derivation trees is undefined
- Combine the Koller et al approach with the tabular surface realisation algorithm?
Future work

- Scaling up (extending the grammar to other types of paraphrases)
- Evaluating (testsuite)
- Testing (application)
Related work

Deep processing of paraphrases:

- Paraphrastic FTAG + parser (with Yannick Parmentier)
- Robust recognition and “interpretation” of alternations and synonymic paraphrases using XIP (a cascaded finite state automaton parser), VerbNet, and WordNet (with Marilisa Amoia)