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.
a2. ≡ Mary lent a book to John.

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. $\not\equiv$ 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

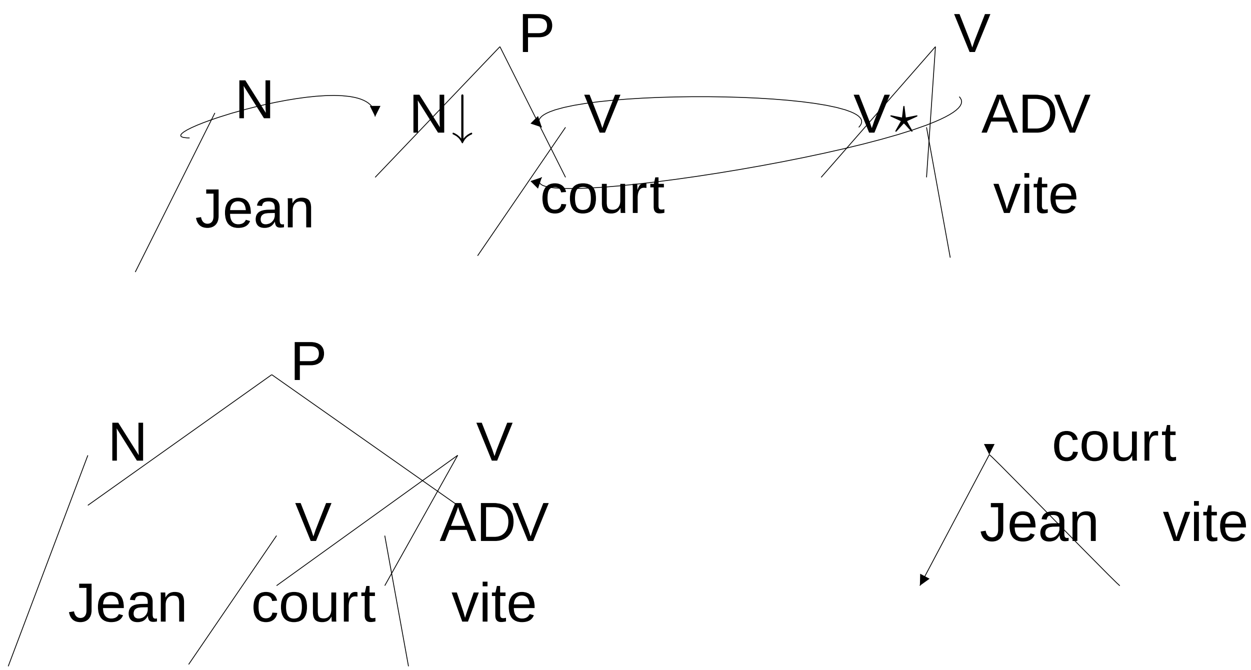
- The grammar (a Feature Based Tree Adjoining Grammar)
- The basic surface realisation algorithm
- Optimisations
- Selection
- Implementation and experimentation
- Related approaches
- Future and related work

The grammar – syntactic dimentions

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

Example



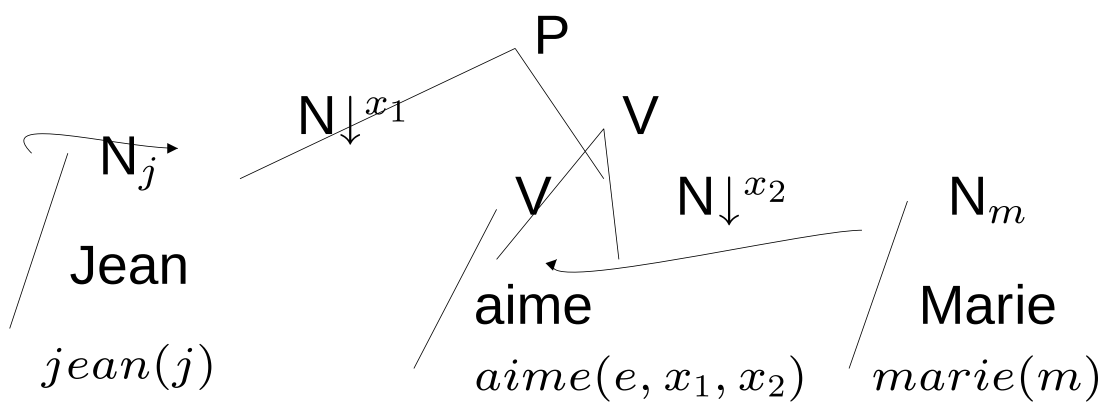
P
V
N
N↓
V
V⋆
ADV
Jean
court
vite
P
N
V
V
ADV
Jean
court
vite
court
Jean
vite

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



$$\Rightarrow jean(j), marie(m), aime(e, j, m)$$

Linguistic coverage

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

Surface realisation algorithm

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

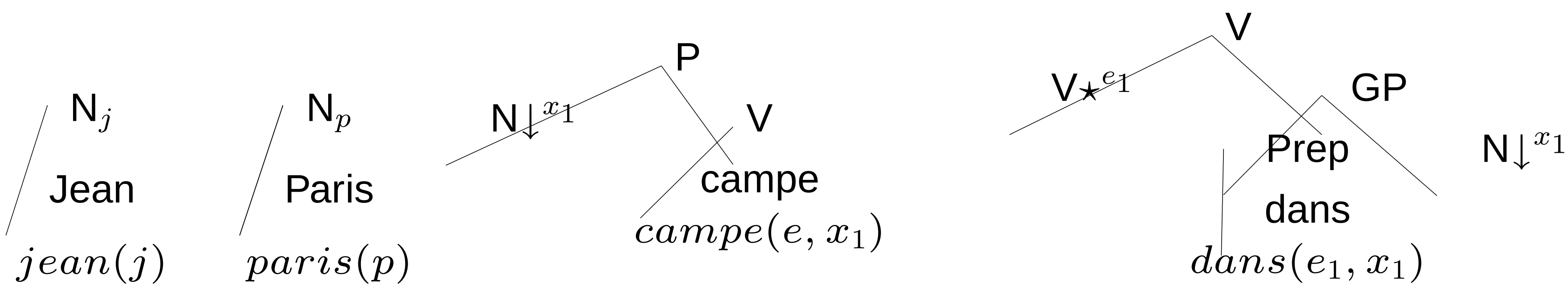
Basic algorithm (simplified)

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

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

Lexical lookup phase



Substitutions

Agenda	Charte	Combination	AgendaA
Jean,campe,dans,Paris			
campe,dans,Paris	Jean		
dans,Paris,JeanCampe	Jean, campe	$\downarrow$(campe,Jean)	
Paris,JeanCampe	Jean,campe,dans		
JeanCampe, dansParis	Jean,campe,dans,Paris	$\downarrow$(dans,Paris)	
dansParis	Jean,campe,dans,Paris,JeanCampe		
	Jean,campe,dans,Paris,JeanCampe		dansParis

Example (Ct'ed)

Adjunctions

Agenda	Charte	Combination	AgendaA
Jean,Paris,JeanCampe	dansParis		
Paris,JeanCampe	dansParis,Jean		
JeanCampe	dansParis,Jean,Paris		
	dansParis,Jean,Paris,JeanCampe	⋆(JeanCampe, ,dansParis)	
JeanCampeDansParis	dansParis,Jean,Paris,JeanCampe		
	dansParis,Jean,Paris,JeanCampe, JeanCampeDansParis		

Optimisations

- Substitutions < Adjunctions
- Elimination of redundant structures
- Polarity based filtering

Multiple modifiers

`fierce(x),little(x),cat(x),black(x)`

For n modifiers, $n!$ intermediate structures :

fierce cat, fierce black cat, little cat,little black cat, fierce little cat, black cat

multiplied by the context :

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
  $\Rightarrow$ 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

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

Lexical look up :

`tableau(t)`	`cout(t,g)`	`grand(g)`
$\tau_{tableau}$	τ_{cout}	$\tau_{est\ eleve}$
$\tau_{peinture}$	τ_{coute}	τ_{cher}

Le tableau coûte cher
Le coût du tableau est élevé
* $\tau_{peinture}$, τ_{coute}, $\tau_{est\ eleve}$

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)

Example (Ct'ed)

tableau(t)	cout(t,g)	grand(g)
$\tau_{tableau}$ +1np $\tau_{peinture}$ +1np	τ_{cout} τ_{coute} -1np	$\tau_{est\ eleve}$ -1np τ_{cher}

tableau(t)		cout(t,g)		grand(g)	
$\tau_{tableau}$	+1np	τ_{cout}	+1np	τ_{cher}	+1np
$\tau_{peinture}$				$\tau_{est\ eleve}$	0np
		τ_{coute}	0np	τ_{cher}	0np
				$\tau_{est\ eleve}$	-1np

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)

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 seminaire d'equipe hebdomadaire special.

Polarity filtering and Paraphrase selection results

Grammar of 2751 trees.

	Example 6	Example 6
Possible combinations	1 377	1 003 833
Combinations explored	9	9
Sentences (w/o selection)	6	36
Sentences (with selection)	2	12

Polarity filtering and Paraphrase selection results

Chart size is reduced by 77% and 87%

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

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

Optimisations	Chart size	CPU time
pol	1258	0.61 s
pol + factor	219	0.47 s

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

Koller and Striegnitz 2004

- 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 grammars 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. explicitely 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)