

# Sentence Generation: Input, Algorithms and Applications

Claire Gardent

CNRS/LORIA Nancy (France)

Joint work with Paul Bedaride, Ben Gottesman, Eric Kow, Shashi Narayan, Laura Perez-Beltrachini and Sylvain Schmitz

TALN 2011, Montpellier

# Outline

- 1 Surface Realisation
- 2 Generating from semantic representations
- 3 Generating from Shallow Dependency Structures

# Surface Realization

*Surface realisation (SR) maps an abstract linguistic representation to a sentence*

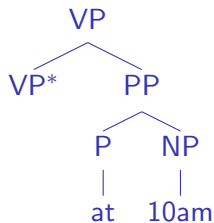
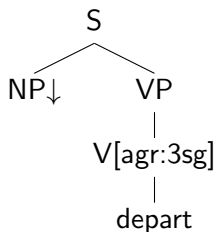
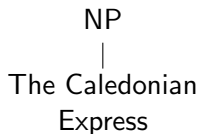
# Example

Arg1:  
Caledon-Exp

Relation:  
DEPARTURE

Arg2:  
10am

# Example

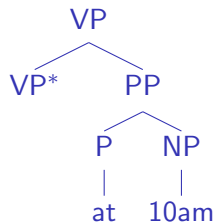
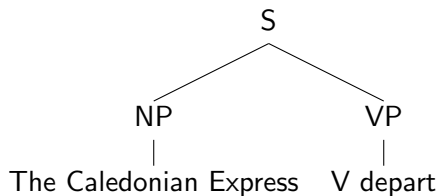


Arg1:  
Caledon-Exp

Relation:  
DEPARTURE

Arg2:  
10am

# Example

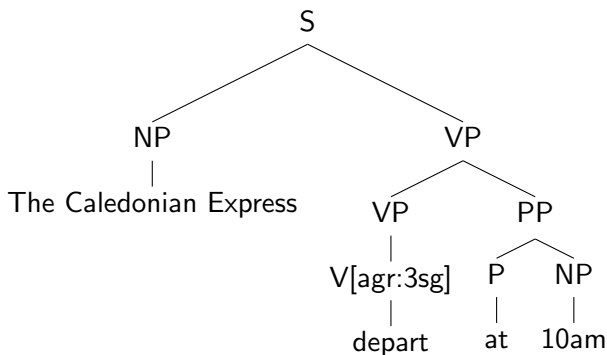


Arg1:  
Caledon-Exp

Relation:  
DEPARTURE

Arg2:  
10am

# Example



*The Caledonian Express departs at 10am.*

Arg1:  
Caledon-Exp

Relation:  
DEPARTURE

Arg2:  
10am

# What is Surface Realisation used for?

- at the end of the Natural Language Generation (NLG) pipeline, to produce the clauses making up the generated text
- in experiments on symbolic, transfer-based machine translation (<http://www.emmtee.net/>)
- to develop intelligent, user friendly interfaces to knowledge bases (cf. the Quelo querying tool or the SWAT ontology verbaliser, <http://kcap11.stanford.edu/tutorials.html>)



# What is the input to SR?

There is no well-defined input to SR. It varies depending on the application:

- LOGON Machine Translation System: flat semantics (Minimal Recursion Semantics) formulae produced by the HPSG ERG grammar
- Quelo ontology querying tool : OWL formulae
- NLG Systems: typed feature structures, dependency trees
- Generation Challenge 2011 SR Task: shallow and flat dependency structures

# What is the input to SR?

The input is more syntactic or more semantic; a tree or a graph.

## Open Questions:

- How does the choice of input representation types affect efficiency and applications ?
- Can a single grammar be used with different input representations ?
- Can a unifying framework be provided for SR algorithms working on both types of input ?

# Table of Contents

- 1 Surface Realisation
- 2 Generating from semantic representations
- 3 Generating from Shallow Dependency Structures

# Generating from semantic representations

- Complexity
- Algorithms
  - ▶ Geni
  - ▶ RTGgen
- Applications
  - ▶ Generation of graduated Test Suites for Entailment Recognition
  - ▶ Error Mining in Grammars
  - ▶ Language teaching and Ontology verbalisation

# Generating from Flat Semantics

Caledon-Exp(c)

departure(e, c)

10am(e)

# Generating from Flat Semantics

NP[idx:c]  
|  
The Caledonian  
Express

S  
/ \  
NP↓ VP[idx:e]  
[idx:c] |  
V[agr:3sg]  
|  
depart

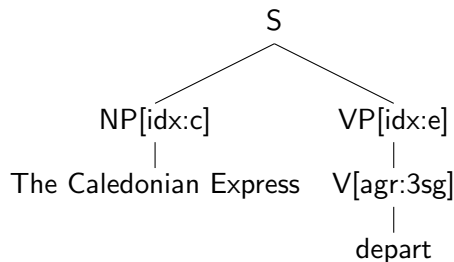
VP  
/ \  
VP\* PP  
[idx:e] / \  
P NP  
| |  
at 10am

Caledon-Exp(c)

departure(e, c)

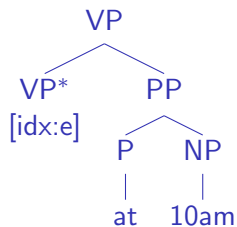
10am(e)

# Generating from Flat Semantics



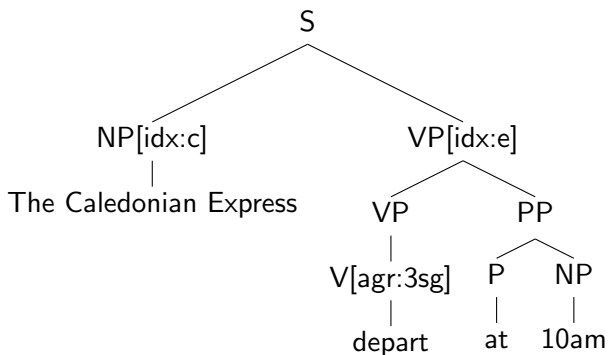
Caledon-Exp(c)

departure(e, c)



10am(e)

## Generating from Flat Semantics



*The Caledonian Express departs at 10am.*

Caledon-Exp(c)

departure(e, c)

10am(e)




# Feature-based Lexicalised Tree Adjoining Grammar

Set of trees, each anchored with a word and associated with a flat semantics.

Two combining operations: substitution and adjunction

Tree nodes decorated with Feature Structures which are unified during combination

-  [Claire Gardent and Laura Kallmayer](#)  
Semantic construction in Feature-Based TAG.  
EACL 2003, Budapest (Hungary)

# Complexity

Surface realisation is exponential in the length of the input  
(Brew92,Kay96)

- **Unordered Input:** At least  $2^n$  possible combinations with  $n$  the number of literals in the input
- **intersective modifiers:**  $2^{m+1}$  possible intermediate structures with  $m$  the number of modifiers for a given structure
- **lexical ambiguity:**  $\prod_{i=1}^{i=n} Lex_i$  possible combinations with  $Lex_i$  the number of lexical entries associated with literal  $l_i$  and  $n$  the number of literals in the input

# SR Algorithm 1: Gen1

3 steps:

- 1 **Lexical selection:** select trees whose semantics subsumes the input semantics.
- 2 **Combination:** perform substitutions or adjunctions between selected trees.
- 3 **Extraction:** Return the trees which are syntactically complete and whose semantics matches the input semantics.



Claire Gardent and Eric Kow.

A Symbolic Approach to Near-Deterministic Surface Realisation using Tree Adjoining Grammar.

ACL 2007, Prag

# Example

$jean(j) \wedge aimer(e, j, m) \wedge marie(m)$

lex selection  
(SemTAG)



$N_j$   
|  
Jean

$N_m$   
|  
Marie

S  
/ | \  
 $N_{\downarrow}^s$   $V^a$   $N_{\downarrow}^t$  ...  
|  
aime

↓ combination

S  
/ | \  
 $N^j$   $V^e$   $N^m$  ...  
| | |  
Jean aime Marie



extraction

Jean aime Marie

and also...

Marie est aimée par Jean

C'est Marie que Jean aime

C'est par Jean que Marie est aimée

...

# Optimisations

- Semantic indices used to limit the impact of unordered input
- Substitutions before Adjunctions to deal with intersective modifiers
- Polarity based filtering to reduce the initial search space and limit the impact of lexical ambiguity

## Intersective modifiers

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

15 intermediate structures:

*F,L,B,FL,FB,BL,BF,LB,LF,FLB,FBL,BLF,BFL,LBF,LFB*

multiplied by the context :

x 2: **the** *F,L,B,FL,FB,BL,BF,LB,LF,FLB,FBL,BLF,BFL,LBF,LFB*

x 2: **the** *F,L,B,FL,FB,BL,BF,LB,LF,FLB,FBL,BLF,BFL,LBF,LFB* **runs**

45 structures built

## Substitutions before Adjunctions

Adjunction restricted to syntactically complete trees

The  $2^{m+1}$  intermediate structures are not multiplied out by the context :

*the cat runs*

*the fierce cat runs, the black cat runs, the little cat runs, the fierce little cat runs, the fierce black cat runs, the black fierce cat runs, ....*

16 structures built

## Polarity based filtering (Perrier 2003)

Polarity based filtering filters out all combinations of lexical items which cannot result in a grammatical structure

- The grammar trees are associated with polarities reflecting their **syntactic resources and requirements**
- A 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)



# Polarity Filtering

Many combinations are syntactically incompatible. Polarity filtering aims to detect these combinations and to filter them out.

john(j)	drink(e, j, w)	water(w)	Polarity Count
+1np	$S_{FIN}$ -2np	+1np	+0np
	$S_{INF}$ -1np		+1np

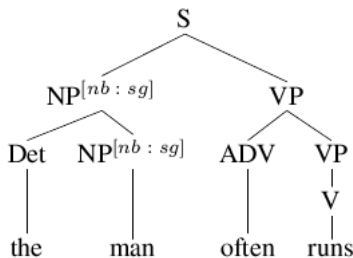
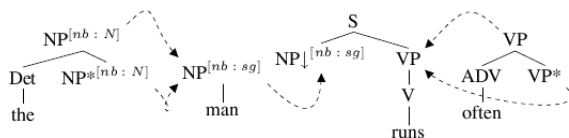
## SR Algorithm 2: RTGen

- Builds derivation rather than derived trees ...
- using a conversion from FB-LTAG to Feature Based Regular Tree Grammar (FB-RTG, Schmitz and Leroux 2009)
- Earley algorithm with packing and sharing

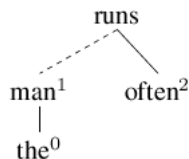


Claire Gardent, Benjamin Gottesman, and Laura Perez-Beltrachini.  
Using Regular Tree Grammars to enhance Sentence Realisation.  
*Natural Language Engineering*, 2011.

# Derived and Derivation Tree

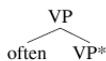
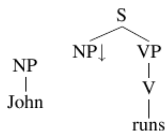


(a) Derived tree



(b) Derivation tree

# Converting a TAG to an RTG



r1.	$NP_S$	$\rightarrow$	$john(NP_A)$
r2.	$S_S$	$\rightarrow$	$runs(S_A NP_S VP_A V_A)$
r3.	$VP_A$	$\rightarrow$	$often(VP_A)$
r4.	$NP_A$	$\rightarrow$	$\epsilon$
r5.	$S_A$	$\rightarrow$	$\epsilon$
r6.	$V_A$	$\rightarrow$	$\epsilon$
r7.	$VP_A$	$\rightarrow$	$\epsilon$

# Earley Algorithm

Axiom

$$\overline{[S' \rightarrow \bullet S_S, \emptyset]}$$

Goal

$$[S' \rightarrow S_S \bullet, \phi] \text{ where } \phi \text{ is the input semantics.}$$

Prediction

$$\frac{[A \rightarrow a(\alpha \bullet B_x \beta), \varphi]}{[\sigma(B^0 \rightarrow b(\bullet B^1, \dots, B^n), \psi)]}$$

with  $\langle B \rightarrow b(B^1, \dots, B^n), \psi \rangle$  a grammar rule

$$\sigma = mgu(B, B^0), P[x] \in \psi \text{ and } \varphi \cap \psi = \emptyset$$

Completion

$$\frac{[A \rightarrow a(\alpha \bullet B \delta), \varphi][B \rightarrow b(\beta) \bullet, \psi]}{[\sigma(A \rightarrow a(\alpha(B, f) \bullet \delta), \varphi \cup \psi)]}$$

$$\text{with } \sigma = mgu(B, B^0), \varphi \cap \psi = \emptyset$$

## RTGen vs GenI

- All trees are taken into account while filtering (GenI's polarity filtering ignores auxiliary trees)
- All features can be used (GenI's polarity filtering can only use ground features i.e., categories)
- All syntactic constraints are applied (not just counting)
- Intersective Modifiers are handled using packing

## Comparison on two Automatically Generated Benchmarks

- Modifiers benchmark: modification differently distributed over the predicate argument structures + lexical ambiguity in modifiers. 1 789 input formulae.
- All benchmark: modification, varying number and type of verb arguments. 890 input formulae.

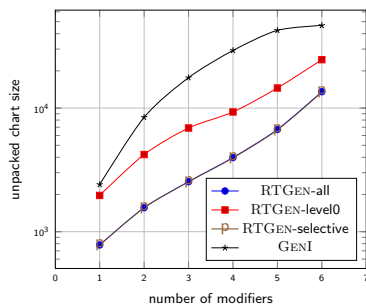


Claire Gardent, Benjamin Gottesman, and Laura Perez-Beltrachini.

Comparing the performance of two TAG-based surface realisers using controlled grammar traversal.

COLING 2010: Posters, Beijing, China.

# Modification

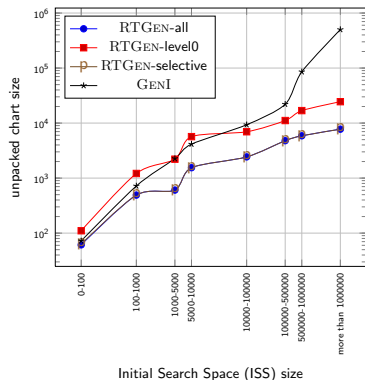


Space performance results on the MODIFIERS-benchmark.

- when using no features (RTGEN-level0) over-generation increases the number of intermediate structures.
- when using all the features (RTGEN-all) or only a selected set of them (RTGEN-selective) (almost) the same number of intermediate structures are produced.



# Overall efficiency



- For more complex cases, RTGen's sharing and packing mechanisms perform better.

performance results on the  
COMPLEXITY-benchmark

Space

# Using GenI to generate Entailment Benchmarks

## RTE Challenge (Recognising Textual Entailment)

- Would a human say that Text1 can be inferred from Text2 ?
- Basic semantic task
- Useful for semantic based applications such as Question Answering, Summarising, Information Extraction, etc.
- Based on real world data. AI complete.



Paul Bedaride and Claire Gardent.

Benchmarking for syntax-based sentential inference.

COLING 2010, Beijing, China.

## Using GenI to generate Entailment Benchmarks

GenI can be used to generate linguistically focused entailment benchmarks annotated with information that supports detailed error mining.

T:            The man gives the woman the flowers that smell nice  
*smell*:{*n0Va1,active,relSubj,canAdj*}

*give*:{*n0Vn2n1,active,canSubj,canObj,canIObj*}

H:            The flowers are given to the woman  
*give*:{*n0Vn1Pn2,shortPassive,canSubj,canIObj*}

Entailment:  TRUE

## Using Genl to generate Entailment Benchmarks

- 1 Use Genl to generate a Generation Bank i.e., a set of tuples  $\langle$  semantics, sentence, syntactic properties  $\rangle$
- 2 Pair generation bank items and use theorem provers to determine whether or not there is entailment between the two sentences

T:  $S_1, \tau_1$

H:  $S_2, \tau_2$

Entailment: TRUE

If  $\langle \phi_1, S_1, \tau_1 \rangle, \langle \phi_2, S_2, \tau_2 \rangle, \phi_1 \models \phi_2$

# Paraphrases and Tree Properties

- One semantics generates several syntactic paraphrases  
⇒ Syntax based textual entailments
- Each elementary tree in the grammar is associated with the set of classes used to build this tree  
⇒ These classes (called Tree properties) can be used to support error mining i.e., to identify the phenomena most often associated with entailment detection errors

## Example variants

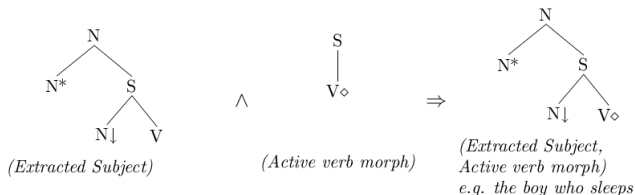
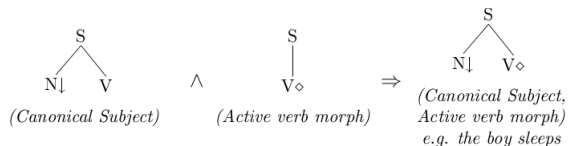
- (1) a. The flower which smells nice is given to the woman by the man
- b. The flower which smells nice is given the woman by the man
- c. The flower which is given the woman by the man smells nice
- d. The flower which is given to the woman by the man smells nice
- e. The flower that smells nice is given to the woman by the man

# Associating sentences with tree properties

- SemTag and SemXTAG are compiled from a factorised grammar description (metagrammar).
- Each tree is associated by the compiler with the set of MetaGrammar classes used to build that tree

# Grammar and Metagrammar

Each TAG elementary tree is associated with a set of “tree properties”.





## Example variants with Tree Properties

- (2) a. The flower which smells nice is given to the woman by the man  
*give:n0Vn1Pn2-Passive-CanSubj-ToObj-ByAgt,*  
*smell:n0V-active-OvertSubjectRelative*
- b. The flower which smells nice is given to the woman by the man  
*give:n0Vn2n1-Passive,*  
*smell:n0V-active-OvertSubjectRelative*
- c. The flower which is given to the woman by the man smells nice  
*give:n0Vn2n1-Passive-CovertSubjectRelative,*  
*smell:n0V-active*
- d. The flower which is given to the woman by the man smells nice  
*give:n0Vn1Pn2-Passive-OvertSubjectRelative,*  
*smell:n0V-active*
- e. The flower that smells nice is given to the woman by the man  
*give:n0Vn1Pn2-Passive,*  
*smell:n0V-CovertSubjectRelative*

# Generating Annotated Entailment Pairs

Generation Bank: generate sentences from semantics

- 81 input formula distributed over 4 verb types
- Generates 226 syntactic variants

Entailment Benchmark: pair generated sentences and annotate the pair as true or false entailment by comparing their semantics using FOL prover

- 6 396 entailment-pairs (42.6% true and 57.4% false entailments)

# Comparing Systems on Syntactic Entailment

system	ERROR	TN	FN	TP	FP	TN/N	TP/P	Prec
afazio	0	360	147	353	140	0.7200	0.7060	71.3%
nutcracker	155	22	62	312	449	0.0467	0.8342	60%
srl	0	487	437	63	13	0.9740	0.1260	55.0%

**Table:** Results of three systems on the entailment benchmark ( TN = true negatives, FN = false negatives, TP = true positives, FP = false positives, Prec = Precision, ERROR: no parse tree found)

# Mining Errors

- Tree Features can help identify the most likely sources of failures.
- We use (Sagot and la Clergerie 2010)'s suspicion rate to compute the probability that a given pair of sets of syntactic tags is responsible for an RTE detection failure.
- The tag set pairs with highest suspicion rate indicate which syntactic phenomena often cooccurs with failure.

## Example Suspect

n0Vs1:act:CanSubj nil 0.85

- T contains a verb with a sentential argument not present in H
- *T: Bill sees the woman give the flower to the man*  
*H: The man gives the flower to the woman.*
- In such cases, we found that the sentential argument in T is usually incorrectly analysed, the analyses produced are fragmented and entailment goes through (False Positive).

# Spotting Overgeneration

Output sentences mostly overgeneration (rough estimate: 70%).

- The grammar was initially designed for parsing, where we are mostly concerned with *undergeneration*.
- It's hard to detect overgeneration with a parser, but easier with a surface realiser.
- Metagrammars are highly factorised, thus fast to develop. But mistakes have wide repercussions.



Gardent, C. and Kow, E. ENLG 2007, Dagstuhl (Germany). "Spotting overgeneration suspects"

# Spotting Overgeneration

Output sentences mostly overgeneration (rough estimate: 70%).

- The grammar was initially designed for parsing, where we are mostly concerned with *undergeneration*.
- It's hard to detect overgeneration with a parser, but easier with a surface realiser.
- Metagrammars are highly factorised, thus fast to develop. But mistakes have wide repercussions... as do their corrections.

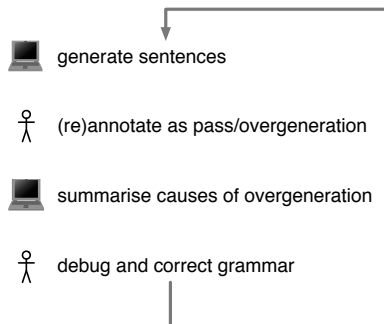


Gardent, C. and Kow, E. ENLG 2007, Dagstuhl (Germany). "Spotting overgeneration suspects"

# Semi-automated test harness

We use an *incremental* regression testing approach, interleaving manual annotations with an automated

- 1 Derivations log
- 2 Suspects report
- 3 Progress report





# Derivations log

For each sentence, its derivation tree, lexical selection and the tree properties for each lexical item.

```
Output: jean se demande si c'est paul qui vient
```

```
demander:n8 <-(s)- venir
```

```
demander:n1 <-(s)- jean
```

```
venir:n4 <-(s)- paul
```

```
demander Tn0ClVs1int-630
```

```
  CanonicalSubject NonInvertedNominalSubject
```

```
  SententialInterrogative
```

```
venir Tn0V-615
```

```
  CleftSubject NonInvertedNominalSubject
```

```
paul TproperName-45
```

```
jean TproperName-45
```

## Suspects report

For each lemma: tree families, trees and tree properties which are consistently associated with overgeneration.

```
input t90
Lemma: dire
Tn0Vn1 (all) - InfinitiveSubject Passive
  [699] CanonicalCAgent Passive
  [746] CanonicalGenitive dePassive
  [702] CleftCAgentOne Passive
  [752] CleftDont dePassive
```

Also, what *combinations* of lexical items are consistently associated with overgeneration.

```
Input t70
consistently overgenerating derivation items
le:Tdet-17:n0 <-(a)- riche:Tn0vA-90
```

# Progress report

Notifies the linguist of possible regressions (passing sentences that no are longer produced), and also of any improvements.

New output?

```
jean dit c'est l'homme volontaire qui part
```

Oops! We lost these passes:

```
jean dit l'homme volontaire part
```

Hooray! no longer overgenerates:

```
dit part l'homme volontaire jean  
dit jean part l'homme volontaire
```

## One fourth the outputs

We got a 70% reduction in strings produced per input after 13 modifications to the metagrammar (31 lines, 12 linguist-hours).

	total	max	mean	median
before	28000	4900	200	25
after	8400	710	60	12

# Generating Teaching Material for Learners of French

- Given an input semantics, GenI generates several syntactic variants realising that input
- Each variant is characterised by tree features
- **Selection constraints** can be used to impose formal constraints on the output

## Application to Computer Aided Language Learning

From the same knowledge base, adaptive and varied teaching material (exercises, examples, exercise solutions) can be generated to suit (i) a specific teaching goal and (ii) the learner level

## Tree properties as selectors

Tree properties can be used as a basis for paraphrase selection if we enrich the input semantics accordingly:

```
give(e,j,b,m), john(j), mary(m)
```

John gives the book to Mary.

Mary is given the book by John.

The book is given to Mary by John.

Each tree property acts as a filter; the surface realiser retains only lexically selected items that possess the requested properties.

## Tree properties as selectors

Tree properties can be used as a basis for paraphrase selection if we enrich the input semantics accordingly:

```
give(e,j,b,m) [PassiveForm], john(j), mary(m)
```

~~John gives the book to Mary.~~

Mary is given the book by John.

The book is given to Mary by John.

Each tree property acts as a filter; the surface realiser retains only lexically selected items that possess the requested properties.

## Tree properties as selectors

Tree properties can be used as a basis for paraphrase selection if we enrich the input semantics accordingly:

```
give(e,j,b,m) [PassiveForm, CanonicalToObject], john(j), mary(m)
```

~~John gives the book to Mary.~~

~~Mary is given the book by John.~~

The book is given to Mary by John.

Each tree property acts as a filter; the surface realiser retains only lexically selected items that possess the requested properties.



## Syntactic Variants

(3) a.  $I_j: \text{jean}(j)$   $I_a: \text{see}(e, j, m)$   $I_m: \text{marie}(m)$

b. L'homme voit Marie

c. Marie est vue par l'homme

d. C'est l'homme qui voit Marie

e. C'est l'homme par qui Marie est vue

f. C'est l'homme par qui est vue Marie

g. C'est par l'homme que Marie est vue

h. C'est par l'homme qu'est vue Marie

i. C'est Marie qui est vue par l'homme

j. C'est Marie que voit l'homme

k. C'est Marie que l'homme voit

## Varying the teaching material

Teaching goal: Passive, Advanced User

(4) a.  $I_j:jean(j) I_a:see(e,j,m)[Passive] I_m:marie(m)$

b. ~~L'homme voit Marie~~

c. Marie est vue par l'homme

d. ~~C'est l'homme qui voit Marie~~

e. C'est l'homme par qui Marie est vue

f. C'est l'homme par qui est vue Marie

g. C'est par l'homme que Marie est vue

h. C'est par l'homme qu'est vue Marie

i. C'est Marie qui est vue par l'homme

j. ~~C'est Marie que voit l'homme~~

k. ~~C'est Marie que l'homme voit~~

## Varying the teaching material

Teaching goal: Passive, Beginner (no inverted subject)

(5) a.  $I_j: \text{jean}(j) I_a: \text{see}(e, j, m) [\text{Passive}, \text{NonInvSubj}] I_m: \text{marie}(m)$

b. ~~L'homme voit Marie~~

c. Marie est vue par l'homme

d. ~~C'est l'homme qui voit Marie~~

e. C'est l'homme par qui Marie est vue

f. ~~C'est l'homme par qui est vue Marie~~

g. C'est par l'homme que Marie est vue

h. ~~C'est par l'homme qu'est vue Marie~~

i. C'est Marie qui est vue par l'homme

j. ~~C'est Marie que voit l'homme~~

k. ~~C'est Marie que l'homme voit~~

## Varying the teaching material

Selection constraints can be used

- To produce the solution from the exercise (Active/Passive transformation)
- To vary the exercising materials: from the same KB, produce all possible realisation of the input semantics that respects the teaching goal and the learner model
- To vary the teaching material (example sentences)

### Application

Selection constraints used in the Allegro I-FLEG (Interactive Game for Learning French) serious game for learning French

# Table of Contents

- 1 Surface Realisation
- 2 Generating from semantic representations
- 3 Generating from Shallow Dependency Structures**

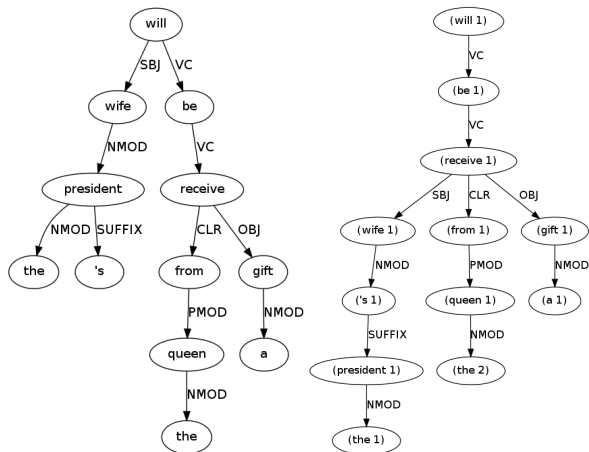
# Generating from Shallow Dependency Structures

- NLG systems often produce much more specific abstract linguistic structures as input for Surface Realisation
- Such structures can be derived from parse trees

## Generation Challenge Surface Realisation Task

Shallow dependency trees are derived from the Penn Treebank and used as input to test and compare surface realisers.

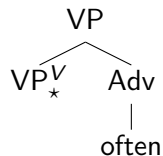
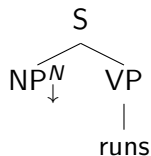
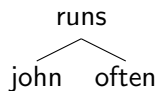
# Derivation and Dependency Trees



## From RTGen to DRTGen

- The elementary trees of FB-TAG are associated with dependency triples (rather than with semantic literals)
- The dependency relations between words  $\langle W, Pos, Rel, Head \rangle$  guide the search
- Mismatches between dependency structures and derivation trees are rewritten using graph rewriting tools (GrGen.Net)
- Error mining techniques are used to identify these mismatches





`subj(john, runs)`  
`vmod(often, runs)`

`subj(N, runs)`

`vmod(often, V)`

$NP_S^i \rightarrow John$

$S_S \rightarrow runs(NP_S^N VP_A)$   
 $subj(N, runs)$

$VP_A^V \rightarrow often(VP_A)$   
 $vmod(often, V)$

# Earley Algorithm with Dependency Tree Input

$$\text{Prediction (Subst)} \quad \frac{[A \rightarrow a(\alpha \bullet B_S \beta), \varphi, \tau]}{[\sigma(B_S^0 \rightarrow b(\bullet B^1, \dots, B^n), \psi, \emptyset)]}$$

with  $\langle B_S^0 \rightarrow b(B^1, \dots, B^n), \psi \rangle$  a grammar rule  
 $\sigma = mgu(B, B^0)$ ,  $r(b, a) \in \varphi$  and  $\varphi \cap \psi = \emptyset$

$$\text{Prediction (Adj)} \quad \frac{[A \rightarrow a(\alpha \bullet B_A \beta), \varphi, \tau]}{[\sigma(B_A^0 \rightarrow b(\bullet B^1, \dots, B^n), \psi, \emptyset)]}$$

with  $\langle B_A^0 \rightarrow b(B^1, \dots, B^n), \psi \rangle$  a grammar rule  
 $\sigma = mgu(B, B^0)$ ,  $r(b, a) \in \psi$  and  $\varphi \cap \psi = \emptyset$

# Completion

Completion (Subs) 
$$\frac{[A \rightarrow a(\alpha \bullet B \delta), \varphi_a, \tau_a][B \rightarrow b(\beta) \bullet, \varphi_b, \tau_b]}{[\sigma(A \rightarrow a(\alpha(B, f) \bullet \delta), \varphi_a, \tau_a \cup \tau_b \cup \{r(b, a)\})]}$$

with  $\sigma = mgu(B, B^0), r(b, a) \in \varphi_a, \varphi_a \cap \varphi_b = \emptyset$

Completion (Adj) 
$$\frac{[A \rightarrow a(\alpha \bullet B \delta), \varphi_a, \tau_a][B \rightarrow b(\beta) \bullet, \varphi_b, \tau_b]}{[\sigma(A \rightarrow a(\alpha(B, f) \bullet \delta), \varphi_a, \tau_a \cup \tau_b \cup \{r(b, a)\})]}$$

with  $\sigma = mgu(B, B^0), r(b, a) \in \varphi_b, \varphi_a \cap \varphi_b = \emptyset$

# Example Run

	<i>Rule</i>	<i>Infce</i>	<i>Cdn</i>	<i>Covered</i>
1	$S' \rightarrow \bullet S$	<i>Axiom</i>		$\emptyset$
2	$S \rightarrow runs(\bullet NP_S VP_A)$	$P_S(1)$		$\emptyset$
3	$NP_S \rightarrow john\bullet$	$P_S(2)$	$subj(j, r) \in \delta(2)$	$\emptyset$
4	$S \rightarrow runs(NP_S \bullet VP_A)$	$C(2, 3)$		$\{subj(j, r)\}$
5	$VP_A \rightarrow often(\bullet VP_A)$	$P_A(4)$	$vmod(o, r) \in \delta(5)$	$\emptyset$
6	$VP_A \rightarrow \epsilon\bullet$	$P_A(5)$		$\emptyset$
7	$VP_A \rightarrow often(VP_A\bullet)$	$C(5, 6)$		$\emptyset$
8	$S \rightarrow runs(NP_S VP_A\bullet)$	$C(4, 7)$		$\{subj(j, r), vmod(o, r)\}$

# Conclusion

The development, evaluation and comparison of surface realisers has long been restricted by:

- the lack of well defined and abundant input
- the lack of applications (mainly as a backend to NLG systems)

# Things are changing

The Generation Challenge SR Task provides large scale deep and shallow input

Exciting research questions are arising e.g.,

- SR for developing intelligent Natural Language Interfaces to knowledge bases (Quelo, SWAT, SRI, etc.)
- SR for Quality focused transfer based machine translation (LOGON)
- SR to detect overgeneration and improve deep symbolic grammars

Thanks!