Structure-Driven Lexicalist Generation

Shashi Narayan[1] and Claire Gardent[1]

(1) Université de Lorraine/LORIA, Nancy (2) CNRS/LORIA, Nancy

December 14, 2012
COLING 2012
Mumbai, India

This talk is about ...

Optimising Surface Realisation

using a Grammar-Based Surface Realiser

and evaluating it on a large benchmark derived from the PennTreebank and recently made available by the Generation Challenge Surface Realisation (SR) Task

What is Surface Realisation?

SR maps INPUT DATA to SENTENCES

The input data can be more syntactic or more semantic; a tree or a graph:

- Dependency trees (SR Task)
- OWL triples
- First Order Logic (FOL) Formulae
- Flat semantics (MRSs)
- ...

$\exists x.(Man(x) \wedge \exists y.(Apple(y) \wedge eat(e,x,y) \wedge now(e)))$
$\Rightarrow$ *A man eats an apple*

Grammar-Based Surface Realisation Algorithms

Two main approaches

Head-Driven algorithm (Shieber et al. 1990)

Used for recursively structured input data e.g., logical formulae
Use this structure to guide the search
Hybrid Top-Down and Bottom-Up search

Lexicalist

Used for unstructured (Flat) input data (MRS formula) e.g.,
Bottom-Up Search

Head-Driven Surface Realisation

Use semantic structure to identify the syntactic heads and generate using their top-down predictions

Top-Down guidance restrict the search space

Logical Form Equivalence (LFE) problem

The input structures must match the structures generated by the grammar
E.g., $small(x) \wedge cat(x) \neq cat(x) \wedge small(x)$

Lexicalist Approach

Selects lexical entries bottom-up from the input semantic literals

Eschews the Logical Form Equivalence problem (because no structure)

Computationally expensive

- Unordered input: 2^n possible combinations to explore assuming each literal selects exactly one lexical entry with n the number of literals in the input
- Lexical ambiguity: $\prod_{1 \leq i \leq n} e_i$ with e_i the number of lexical entries selected by the i-th literal
- Intersective modifiers: 2^m possible combinations with m the number of modifiers modifying the same entity

Structure-Driven Lexicalist Surface Realisation

Combines techniques and ideas from the head-driven and the lexicalist approach.

Lexicalist

- Select grammar rules bottom up for each input unit (i.e., lemma)

Structure Driven

- Uses the structure of the input to guide the search

Outline

Related work (Optimisation of Surface Realisation algorithms)

Structured Input from the Generation Challenge Surface Realisation Task

Tree Adjoining Grammar and Regular Tree Grammar

The Algorithm

Evaluation and Results

Surface Realisation Algorithms and Optimisations

- Grammar Based with Symbolic Optimisations (polarity filtering, building derivation rather than derived trees, tabular, modifiers last): (Gardent and Kow 2007; Koller and Striegnitz, 2002; Gardent and Perez-Beltrachini 2010; Carrol et al. 1999).
- Grammar Based with Statistical Optimisations (supertagging, language and tree models): HPSG (Carroll and Oepen 2005); TAG (Bangalore and Rambow 2000); CCG: (White 2004, Espinosa et al, 2008)
- Statistical realisers (Guo et al 2011, Bohnet et al 2011, Stent 2011): Cascaded classifiers and n-gram models to map input to sentences

Structure-Driven Lexicalist Generation

- FB-LTAG converted to FB-RTG
  (Koller and Striegnitz, 2002; Gardent and Perez-Beltrachini 2010)
- Parallelism used to explore the possible completions of the top-down predictions simultaneously rather than sequentially.
- Top-down filter using the structure of the input
  Head-Driven algorithm
- Bottom-up polarity filter on local input trees.
  (Bonfante 2004; Gardent and Kow 2007).
- Language model used to prune competing intermediate substructures
  (Bangalore and Rambow 2000; White 2004)

Input Representations

From the Generation Challenge Surface Realisation Task (Shallow Track)

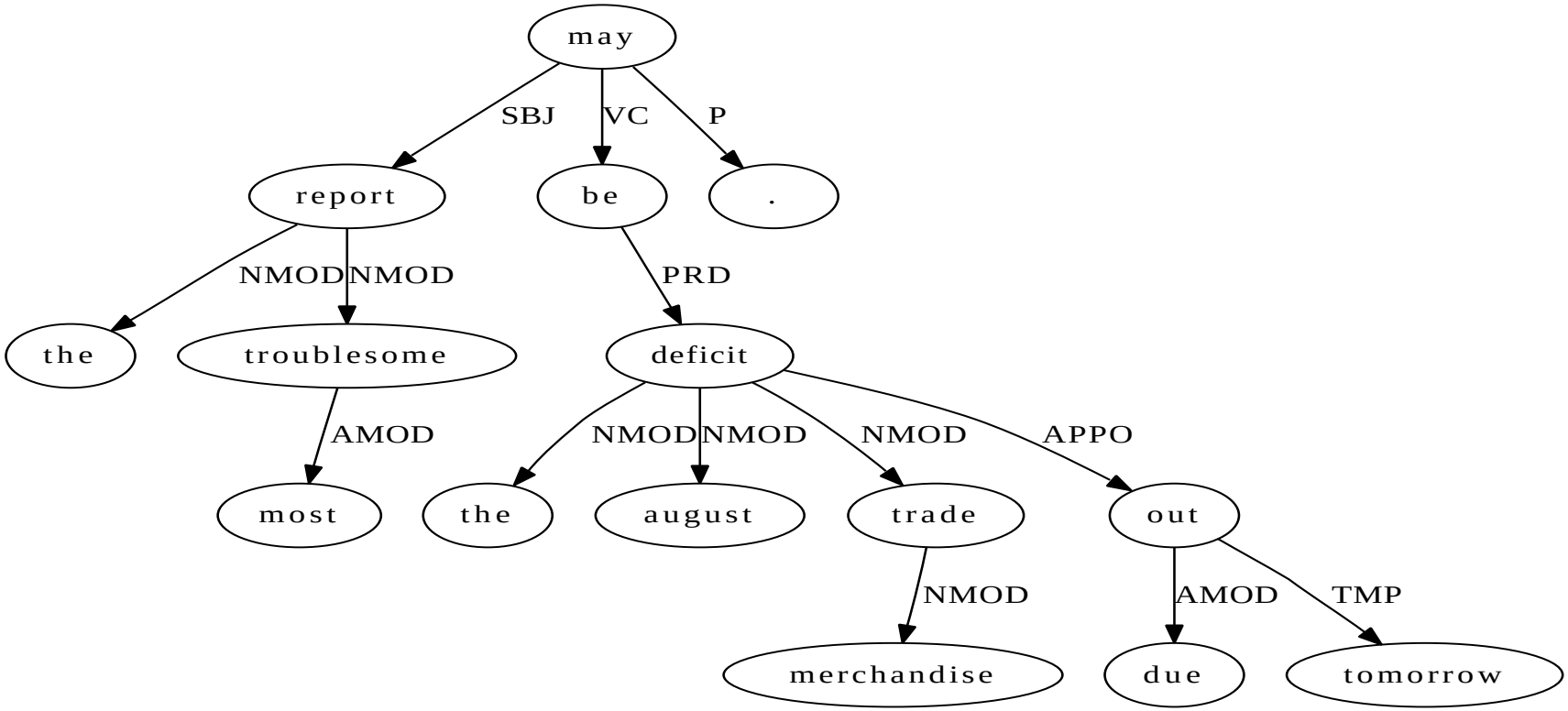
Shallow dependency structures

- Unordered trees
- Edges are labelled with syntactic functions
- Nodes labelled with lemmas, part of speech tags and partial morphosyntactic information

All words of the original sentence are represented by a node in the tree

Example Input

The most troublesome report may be the August merchandise trade deficit due out tomorrow



Grammars

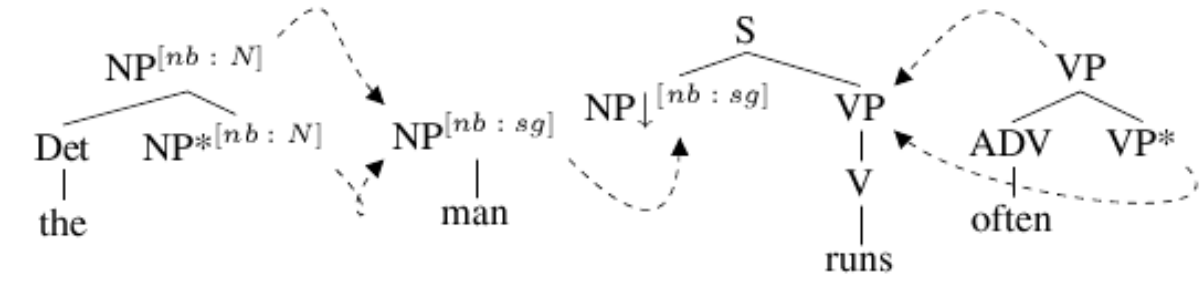
Feature-Based Lexicalised Tree Adjoining Grammar (FB-LTAG)

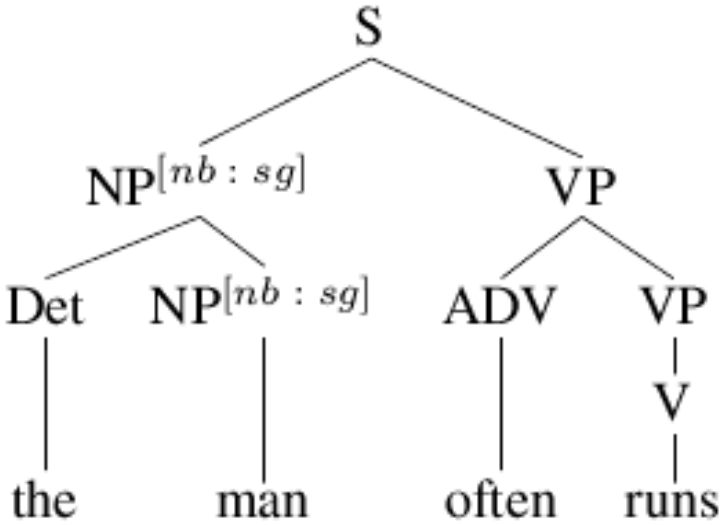
- A set of trees, lexicalised with one or more words and decorated with feature structures
- 2 combining operations: substitution and adjunction

Converted to a Feature-Based Regular Tree Grammar (FB-RTG)

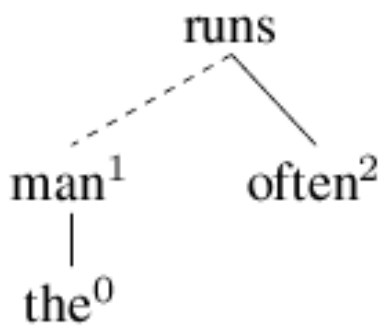
- Generates derivation trees first
- More efficient: fewer nodes, optimised context free algorithms
  (Koller and Striegnitz, 2002; Gardent and Perez-Beltrachini 2010).

Example FB-LTAG



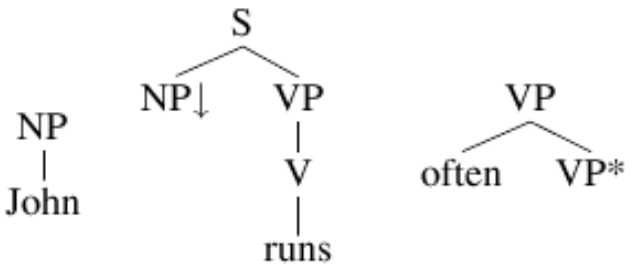


(a) Derived tree



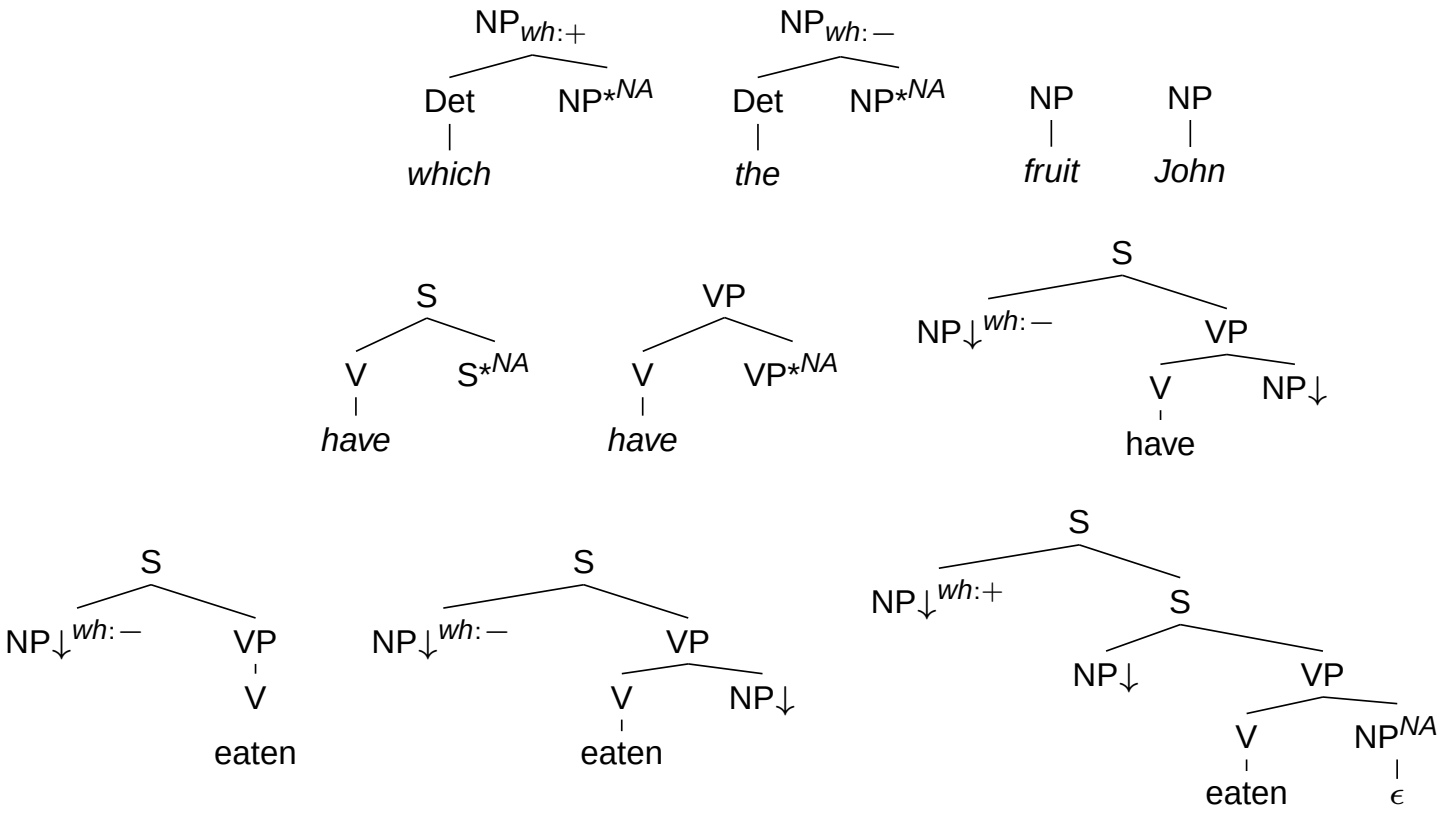
(b) Derivation tree

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	$\rightarrow$	ϵ
r5.	S_A	$\rightarrow$	ϵ
r6.	V_A	$\rightarrow$	ϵ
r7.	VP_A	$\rightarrow$	ϵ

A Slightly Bigger FB-LTAG

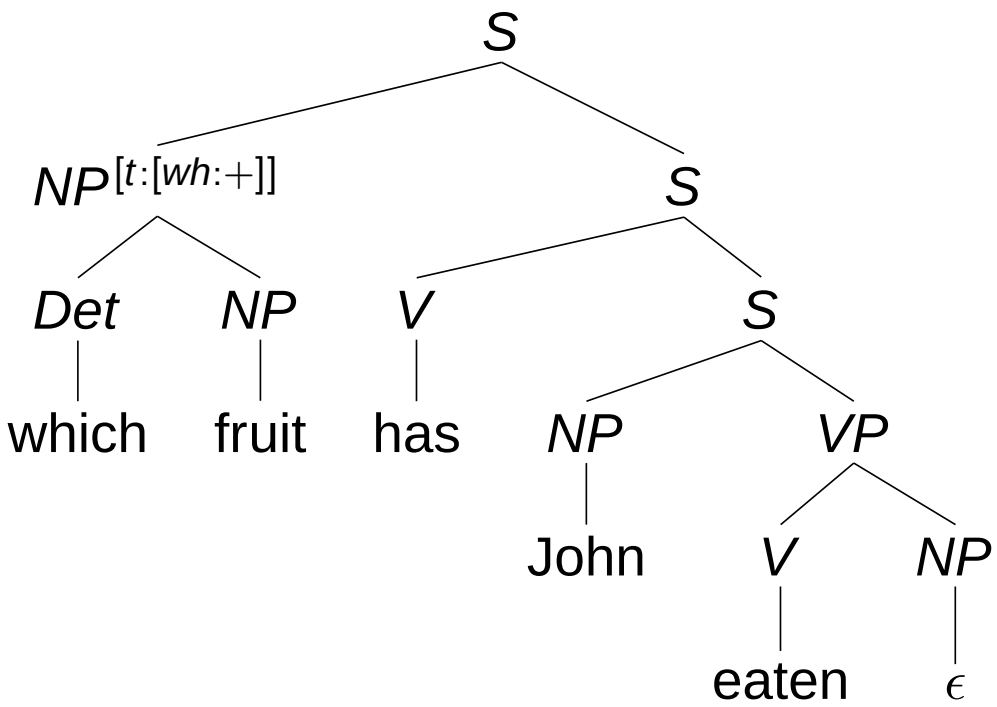


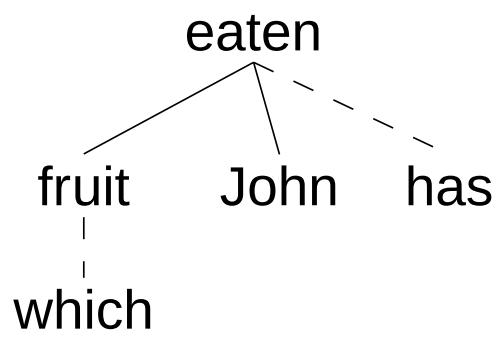
... and the corresponding FB-RTG

$$
\begin{array}{lcl}
NP_A^{[t:T]} & \rightarrow & which(NP_A^{[t:T,b:[wh:+]]}) \\
NP_A^{[t:T]} & \rightarrow & the(NP_A^{[t:T,b:[wh:-]]}) \\
NP_S^{[t:T]} & \rightarrow & fruit(NP_A^{[t:T]}) \\
NP_S^{[t:T]} & \rightarrow & John(NP_A^{[t:T]}) \\
S_A^{[t:T]} & \rightarrow & have(S_A^{[t:T]}) \\
VP_A^{[t:T]} & \rightarrow & have(VP_A^{[t:T]}) \\
S_S^{[t:T,b:B]} & \rightarrow & have(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:-]]}\ VP_A\ NP_S) \\
S_S^{[t:T,b:B]} & \rightarrow & eat(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:-]]}\ VP_A) \\
S_S^{[t:T,b:B]} & \rightarrow & eat(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:-]]}\ VP_A\ NP_S) \\
S_S^{[t:T,b:B]} & \rightarrow & eat(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:+]]}\ S_A\ NP_S\ VP_A) \\
X_A^{[t:T,b:T]} & \rightarrow & \epsilon
\end{array}
$$

A Derivation

Which fruit has John eaten?





The Algorithm

Starts from the root node of the input tree

Processes all children nodes *in parallel* spreading lexical selection constraints *top-down* and combining FB-RTG rules *bottom-up*

4 main steps

- Top-Down Lexical Selection and Filtering
- Bottom-Up Local Polarity Filtering
- Bottom-Up Generation
- N-Gram Filtering

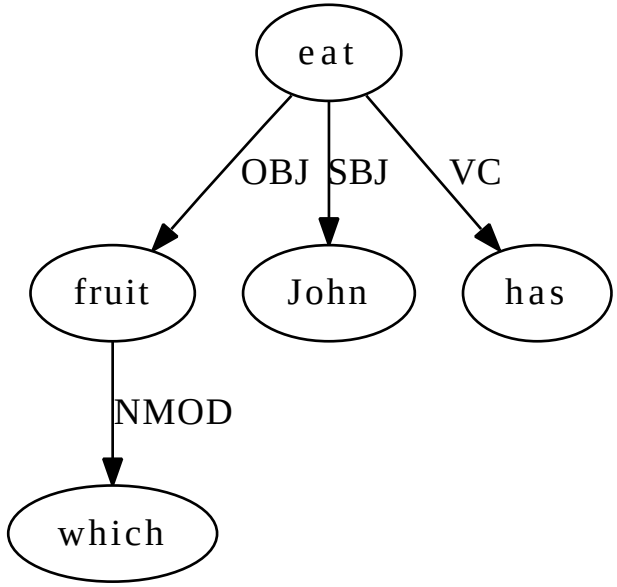
Top-Down Lexical Selection and Filtering

Lexical Selection: for each input node *n* with lemma *w*, selects all FB-RTG rules which can be lexicalised by *w*

Top-Down Filtering: Only keep those rules whose left-hand side category occurs at least once in the right-hand side of the rules selected by the parent node.

Example

Input Dependency Tree



Example Top-Down Filtering

Rule selection for *eat*:

$$
\begin{array}{lll}
S_S^{[t:T,b:B]} & \rightarrow & eat(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:-]]}\ VP_A) \\
S_S^{[t:T,b:B]} & \rightarrow & eat(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:-]]}\ VP_A\ NP_S) \\
S_S^{[t:T,b:B]} & \rightarrow & eat(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:+]]}\ S_A\ NP_S\ VP_A)
\end{array}
$$

Rule selection and filtering for *has*:

$$
\begin{array}{lll}
\surd & S_A^{[t:T]} & \rightarrow\ have(S_A^{[t:T]}) \\
\surd & VP_A^{[t:T]} & \rightarrow\ have(VP_A^{[t:T]}) \\
\times & S_S^{[t:T,b:B]} & \rightarrow\ have(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:-]]}\ VP_A\ NP_S)
\end{array}
$$

Bottom-Up Local Polarity Filtering

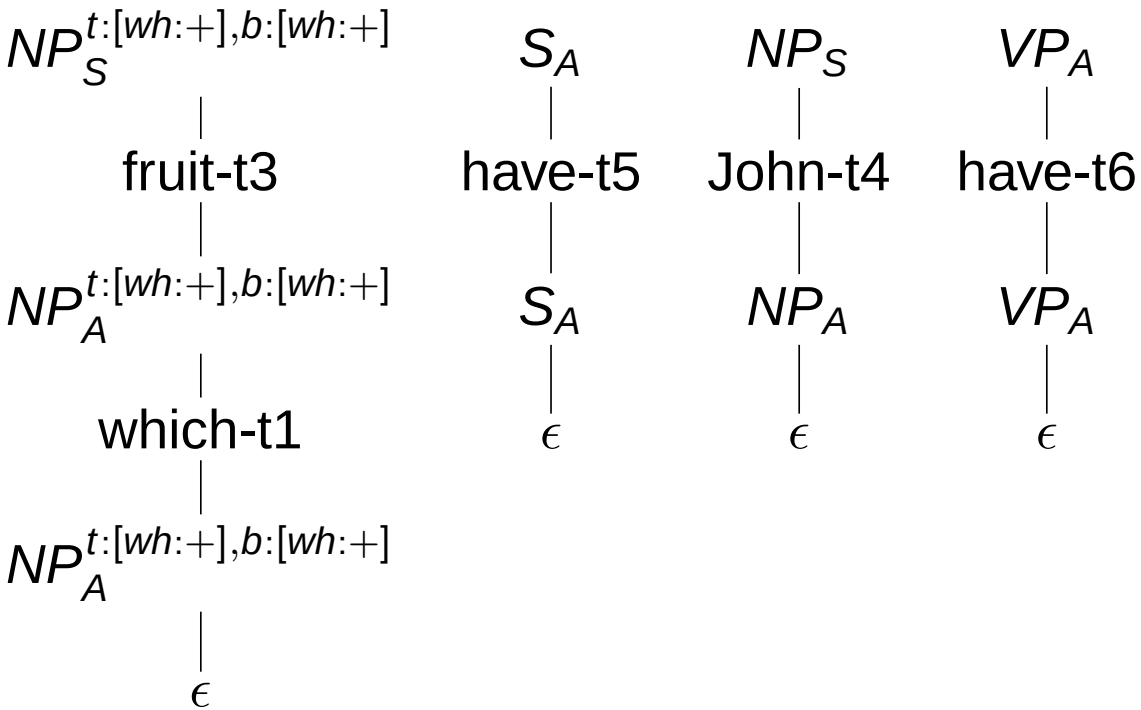
Global Polarity Filtering (Gardent and Kow 2005) filters out

- Sets of rules which cover the input
- but cannot possibly lead to a valid derivation
- either because a substitution node cannot be filled
- or because a root node fails to have a matching substitution site

Local (Structure-Driven) Polarity Filtering: on each local tree

Example of Local Polarity Filtering

$$
\begin{array}{lll}
\times & S_S^{[t:T,b:B]} & \rightarrow \quad eat(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:-]]}\ VP_A) \\
\surd & S_S^{[t:T,b:B]} & \rightarrow \quad eat(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:-]]}\ VP_A\ NP_S) \\
\surd & S_S^{[t:T,b:B]} & \rightarrow \quad eat(S_A^{[t:T,b:B]}\ NP_S^{[t:[wh:+]]}\ S_A\ NP_S\ VP_A)
\end{array}
$$



Bottom-Up Generation and N-Gram filtering

For each local tree in the input, the rule sets passing the local polarity filter are tried out for combination.

Only *the n best scoring n-grams* let through after each bottom-up generation step are kept.
The language model helps finding the most likely ordering of modifiers.

Evaluation and Results

Test data: The SR Data

- Dependency trees derived from the Penn Treebank
- 26 725 inputs
- Average (maximum) word length: 22 (134)
- Average (maximum) branching degree: 4 (18)

Algorithms compared:

- Baseline: A strictly top-down algorithm
  (No time information available for systems participating in SR Task, only coverage and BLEU)
- SEQ: The SDL algorithm without parallelism
- PAR: The SDL algorithm with parallelism

Evaluation Focus: Efficiency (Time)

Evaluation and Results

	Sentences (Length L)							
	$S(0-5)$		$S(6-10)$		$S(11-20)$		$S(All)$	
	Total	Succ	Total	Succ	Total	Succ	Total	Succ
	1084	985	2232	1477	5705	520	13661	2744
BL	**0.85**	0.87	10.90	10.76	110.07	97.52	–	–
SEQ	1.49	1.63	2.84	3.64	4.36	6.03	4.52	3.18
PAR	**1.53**	1.66	2.56	3.28	2.66	4.14	**2.57**	2.78

- Maximum arity = 3. Else BL times out.
- Many time out for BL on input longer than 10
- For short sentences (0-5), BL outperforms SDL
- For sentences with more than 5 words, SDL increasingly outperforms BL

Branching factor and Parallelism

	Sentences (Arity)							
	$S(1)$		$S(3)$		$S(5)$		$S(6)$	
	Total	Succ	Total	Succ	Total	Succ	Total	Succ
	190	178	3619	1039	2910	137	1093	18
SEQ	**0.89**	0.94	3.65	3.39	5.24	4.62	8.20	**7.29**
PAR	**0.97**	1.03	2.63	3.10	2.86	3.88	3.09	**4.76**

The impact of parallelism increases with the branching factor.

Coverage and BLEU score

Coverage: 81.74%

- No robustness mechanism added.

BLEU score: 0.73

- No ranking module
- Best statistical system in SR Task: 0.88
- Best symbolic system in SR Task: 0.37

Conclusion

Results

- an optimised grammar-based SR algorithm
- which permits testing the grammar for under- and over-generation
  $\Rightarrow$ “lean, linguistic” grammar e.g., for Language Learning (Gardent and Perez-Beltrachini 2011)

Future Work

- Compare local polarity filtering and supertagging
- Add a ranking module (to improve BLEU)
- Generating coordination and ellipses
- Adapt algorithm to graph shaped input and reversible grammar

Thanks!