

# Grammar Based Generation: Algorithm Optimisation and Performance Improvement through Error Mining

Claire Gardent  
(Joint work with Shashi Narayan)

CNRS/LORIA, Nancy

March 26, 2014  
Melbourne University, Australia

# Grammar-Based Surface Realisation (SR)

## Efficiency and Coverage

- ▶ XTAG and Top-Down, Bottom-Up SR **algorithm** generates Penn Tree Bank sentences in 2.57 seconds  
Average (maximum) word length: 22 (134)

## Correctness

- ▶ **Error Mining** permits improving correctness and adapting the grammar to the PTB input yielding a BLEU score on covered input of 0.73

# A TD/BU algorithm for Grammar Based Surface Realisation

What is Surface Realisation?

Structured Input from the Generation Challenge Surface Realisation Task

Feature-Based Tree Adjoining Grammar

The Algorithm

Evaluation and Results



S. Narayan and C. Gardent

Structure-Driven Lexicalist Generation

Proceedings of *COLING 2012*, pp 2027 - 2041, Mumbai, India

# 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*

# The SR Shared Task Input Representations

Surface Realisation Task organised by the Generation Challenges 2011

Input derived from the PennTreebank

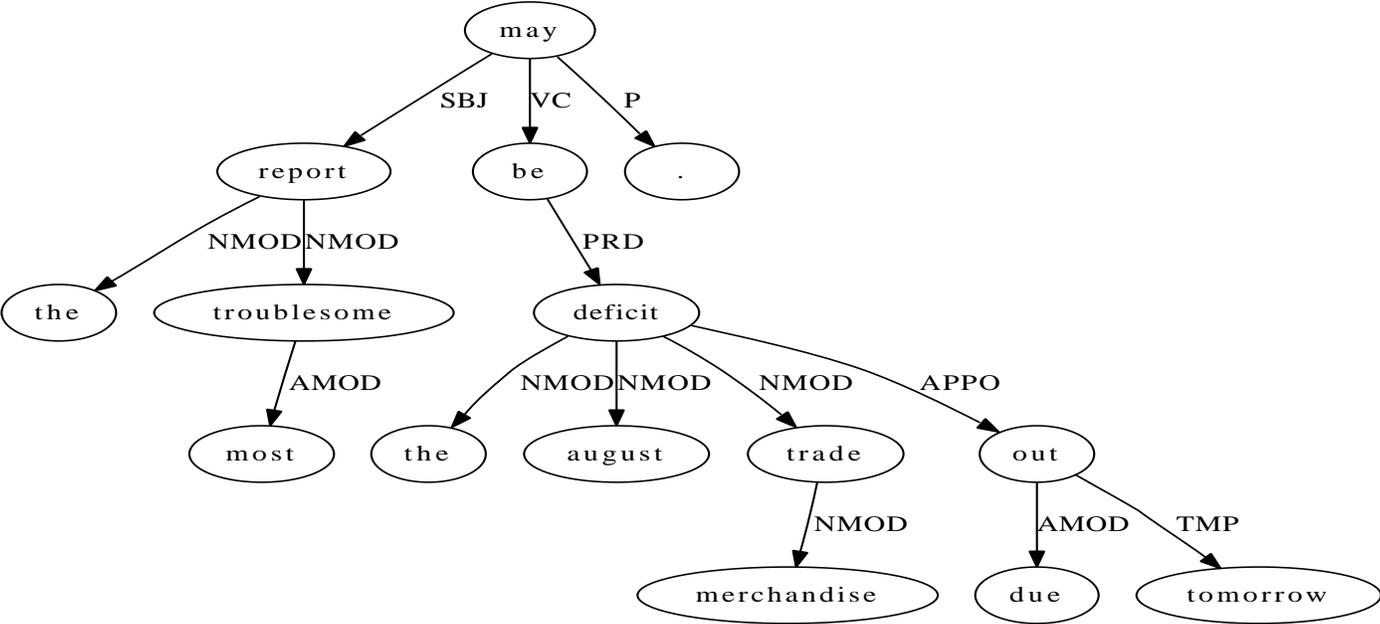
Shallow dependency structures

- ▶ 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*

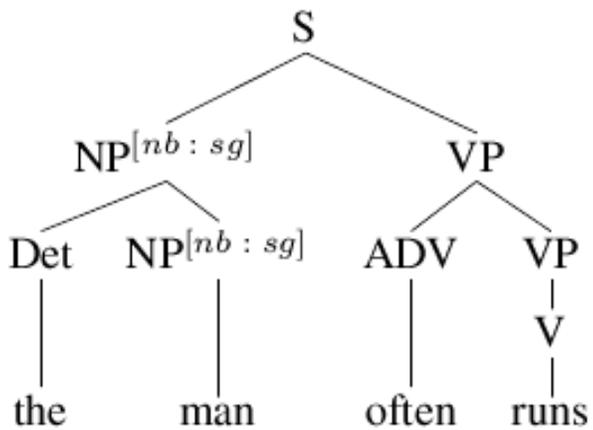
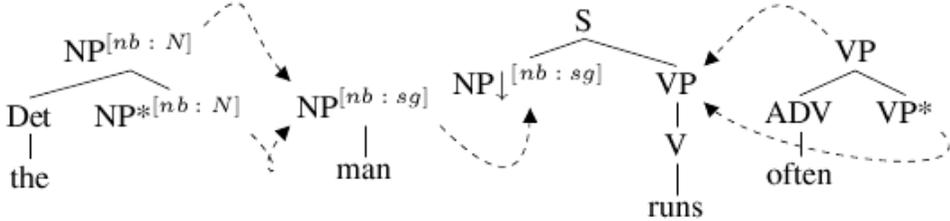


# Grammar

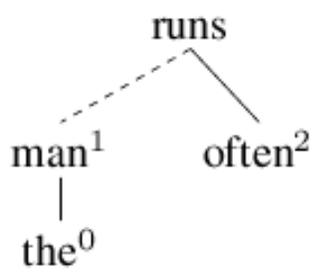
## Feature-Based Lexicalised Tree Adjoining Grammar (FB-LTAG)

- ▶ A set of trees, lexicalised with one or more words and decorated with feature structures
- ▶ 2 combining operations: substitution and adjunction
- ▶ XMG Reimplementation of XTAG (large coverage of English)

# Example FB-LTAG



(a) Derived tree



(b) Derivation tree

# Grammar-Based Surface Realisation Algorithms

Two main approaches

## Head-Driven algorithm

Used for recursively structured input data e.g., logical formulae  
Use input structure to guide the search

**Cons:** Logical Form Equivalence Problem

## Lexicalist

Used for unstructured input data (e.g., MRS formula)  
Selects lexical entries bottom-up from the input semantic literals

**Cons:** Computationally expensive (Unordered input, Lexical ambiguity, Intersective modifiers)

# Structure-Driven Lexicalist Surface Realisation

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

- ▶ Select grammar rules bottom up for each input tree node (Lexicalist)
- ▶ Uses the structure of the input to guide the search and prune the search space (Structure Driven)

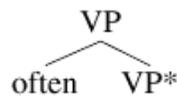
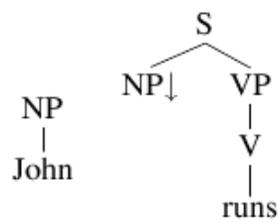
Integrates various optimisations previously proposed for parsing/generation

Parallelised

# Structure-Driven Lexicalist Generation

- ▶ **FB-LTAG converted to FB-RTG** to construct derivation rather than derived trees  
(Koller and Striegnitz, 2002; Gardent and Perez-Beltrachini 2010)
- ▶ Top-down filter using the structure of the input (**Head-Driven** algorithm, Shieber et al. 1990)
- ▶ Bottom-up **polarity filter** on local input trees.  
(Bonfante 2004; Gardent and Kow 2007).
- ▶ **Language model** used to prune competing intermediate structures  
(Bangalore and Rambow 2000; White 2004)
- ▶ **Parallelism** used to explore the possible completions of the top-down predictions simultaneously rather than sequentially.

# 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. | $VP_A$ | $\rightarrow$ | $\epsilon$                |

# The Algorithm

Starts from the root node of the input tree

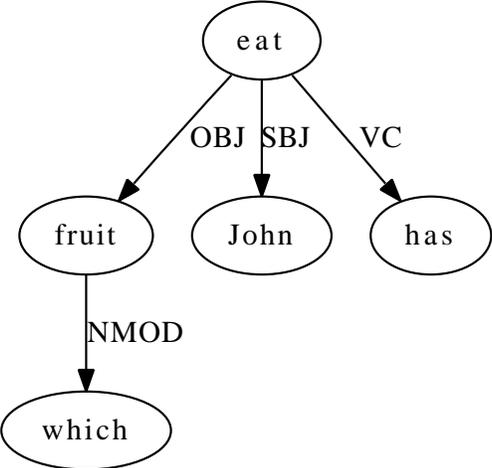
Processes all children nodes *in parallel* spreading lexical selection constraints *top-down* and combining FB-RTG rules *bottom-up*

4 main steps

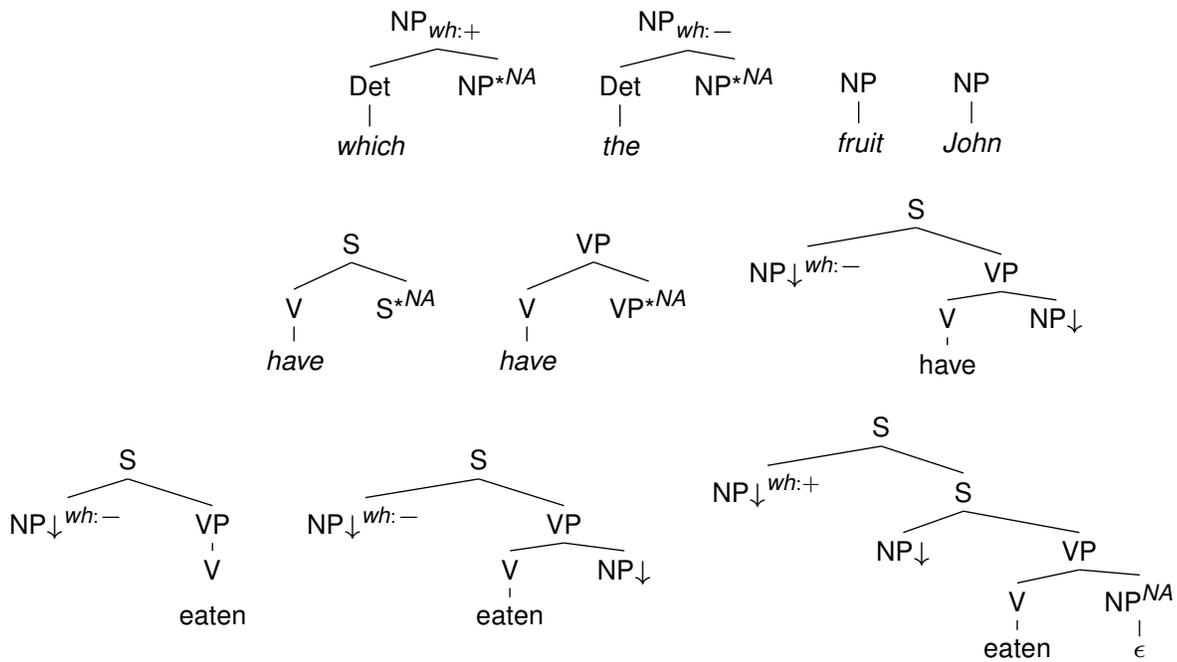
- ▶ Bottom-Up Lexical Selection and Top-Down Filtering
- ▶ Bottom-Up Local Polarity Filtering
- ▶ Bottom-Up Generation
- ▶ N-Gram Filtering

# Example

## Input Dependency Tree



# An FB-LTAG



... and the corresponding FB-RTG

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

# Bottom-Up Lexical Selection and Top-Down 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 Top-Down Filtering

Rule selection for *eat*:

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

Rule selection and filtering for *has*:

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

# Bottom-Up Local Polarity Filtering

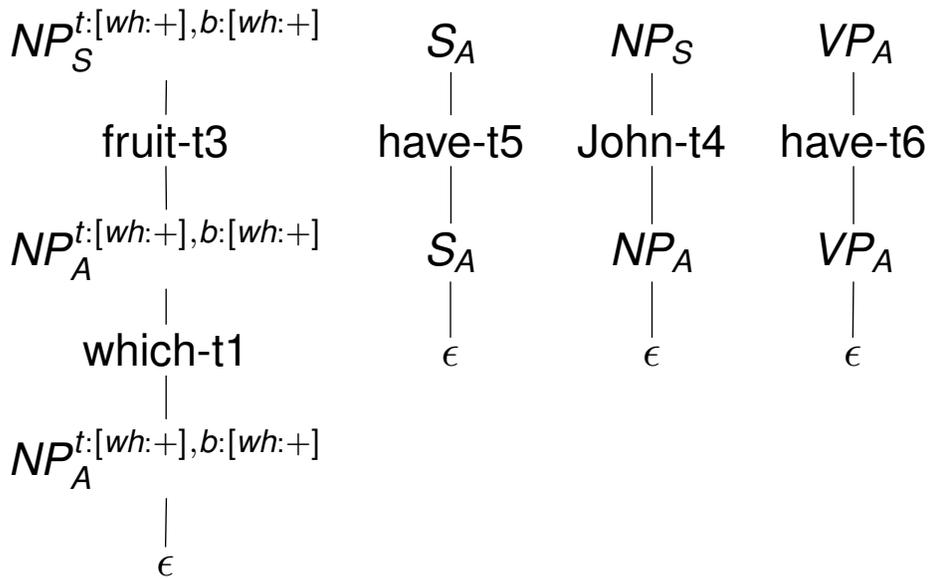
Global Polarity Filtering (Gardent and Kow 2005) filters out

- ▶ 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

- ×  $S_S^{[t:T, b:B]} \rightarrow eat(S_A^{[t:T, b:B]} NP_S^{[t:[wh:-]]} VP_A)$
- ✓  $S_S^{[t:T, b:B]} \rightarrow eat(S_A^{[t:T, b:B]} NP_S^{[t:[wh:-]]} VP_A NP_S)$
- ✓  $S_S^{[t:T, b:B]} \rightarrow eat(S_A^{[t:T, b:B]} NP_S^{[t:[wh:+]]} S_A NP_S VP_A)$



## 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) Total	S(6 – 10) Total	S(11 – 20) Total	S(All) Total
	1084	2232	5705	13661
BL	<b>0.85</b>	10.90	110.07	—
SEQ	1.49	2.84	4.36	4.52
PAR	<b>1.53</b>	2.56	2.66	<b>2.57</b>

- ▶ 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)			
	<i>S</i> (1)	<i>S</i> (3)	<i>S</i> (5)	<i>S</i> (6)
	Total	Total	Total	Total
	190	3619	2910	1093
SEQ	<b>0.89</b>	3.65	5.24	8.20
PAR	<b>0.97</b>	2.63	2.86	3.09

The impact of parallelism increases with the branching factor.

# Coverage and BLEU score

Coverage: 81.74%

- ▶ No robustness mechanism added.

BLEU score: 0.73 (for covered data)

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

# Error Mining as a way to Improve Grammar Correctness

March 26, 2014



S. Narayan and C. Gardent

Error Mining with Suspicion Trees

Proceedings of *COLING 2012*, pp 2011 - 2025,, Mumbai, India



C. Gardent and S. Narayan

Error Mining on Dependency Trees

Proceedings of *ACL 2012*, pp 592 - 600, Jeju Island, Korea

## Error Mining (van Noord 2004)

Goal: Identify errors in grammar or lexicon which leads to parsing failure

Method:

- ▶ Parse  $n$  sentences ( $S$ )
- ▶ Divide the input set of sentences  $S$  into the set of sentences for which parsing succeeds (PASS) and the set of sentences for which parsing fails (FAIL)
- ▶ Identify n-grams or words that frequently occur in FAIL (high *Suspicion Score*)

$$S = \frac{ct(w_i | FAIL)}{ct(w_i)}$$

# Error Mining applied to Generation

Input to generation = Unordered Dependency Trees

Search for subtrees (*Suspicious Forms*) in the input which frequently lead to generation failure and rarely lead to generation success (high *Suspicion Score*).

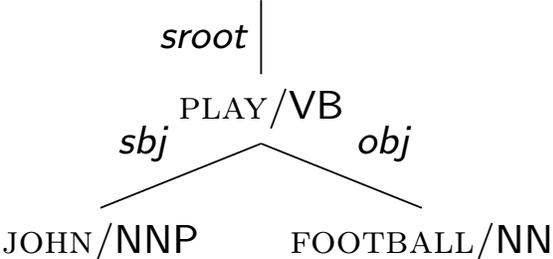
# Error Mining applied to Generation

Three main modifications w.r.t. Van Noord's approach

1. The input is a tree. Suspicious forms are subtrees (not strings)  
We adapt the HybridTreeMiner algorithm to efficiently enumerate subtrees of the input.
2. The suspicion score takes into account both Pass and Fail (not just Pass).
3. The output of error mining is a tree (not a list).  
This output tree highlights the relations between suspicious forms and facilitates grammar correction.

# Example Suspicious Form

Suspicious forms are Suspicious subtrees of the SR dependency trees labelled with lemma, parts-of-speech and/or dependency information



(*sroot*-PLAY/VB, (*sbj*-JOHN/NNP), (*obj*-FOOTBALL/NN))

WORD	(PLAY, (JOHN), (FOOTBALL))
POS	(VB, (NNP), (NN))
<i>dep</i>	( <i>sroot</i> , ( <i>sbj</i> ), ( <i>obj</i> ))
WORD/POS	(PLAY/VB, (JOHN/NNP), (FOOTBALL/NN))
<i>dep</i> -POS	( <i>sroot</i> -VB, ( <i>sbj</i> -NNP), ( <i>obj</i> -NN))

## Enumerating Subtrees

HybridTreeMiner algorithm (Chi et al. 2004): Build *an enumeration tree* whose nodes are all possible subtrees of  $T$  and such that, at depth  $d$  of this enumeration tree, all possible frequent subtrees consisting of  $d$  nodes are listed.

1. Convert the unordered labelled trees to a canonical form called BFCF (Breadth-First Canonical Form)
2. Enumerate the subtrees of the BFCF trees in increasing order of size using two tree operations called join and extension

# Adpating the HybridTreeMiner algorithm for Error Mining

Use **support** (nb of FAIL and PASS sentences for a given form  $f$ ) and suspicion score to prune the search space.

for a larger tree  $t$  to be added to the enumeration tree, the suspicion score of all subtrees contained in a new tree  $t$  must be smaller or equal to  $S(t)$ .

$$S(f_n) \geq S(t_{n-1}), \forall t_{n-1} \in t_n$$

Construct the tree breadth first rather than depth first

The enumeration process take 10-15 minutes for a dataset of 123,523 trees.

## Suspicion Score, $S_{score}(f)$

Captures the degree to which a form  $f$  is associated with failure. It is high when  $f$  is *often present in data associated with failure* and/or *often absent in data associated with success*.

$$S_{score}(f) = \frac{1}{2}(\text{Fail}(f) * \ln \text{count}(f) + \text{Pass}(\neg f) * \ln \text{count}(\neg f))$$

$$\text{Fail}(f) = \frac{\text{count}(f|\text{FAIL})}{\text{count}(f)}$$

$$\text{Pass}(\neg f) = \frac{\text{count}(\neg f|\text{PASS})}{\text{count}(\neg f)}$$

## Constructing the output Suspicion Tree

We adapt the ID3 decision tree algorithm to yield a tree whose nodes are suspicious forms and whose structure highlights the relations between suspicious forms .

The Error Mining algorithm recursively partitions the data by

1. selecting the suspicious form with highest suspicion score (*attribute selection*).
2. using this attribute to **split the data** into two subsets, a subset containing that attribute and a subset excluding that attribute (*dataset division*).

## Pruning the Suspicion Tree

1. Only consider those forms whose frequency is above a given threshold.
2. Only consider a larger suspicious form if its suspicion rate is larger than the suspicion rate of all smaller forms it contains.
3. Limit depth of tree (max 10).

Building a suspicion tree for a dataset of 123,523 trees takes about one minute.

## Experiments and Results

We applied error mining to the results of our SR algorithm on the SR data.

- ▶ Corrections:
  - ▶ 11 rewrite rules (Gen-1, Dt-4, Adv-1, Inf-3, Aux-1 and Final-1), to adapt the SR data to the input expect by our generator
  - ▶ 2 grammar corrections and
  - ▶ a few lexicon updates

Test Data	# Failures Before EM	# Failures after EM	Error Reduction
26725	19280 (72.1%)	5157 (19.3%)	-52.8

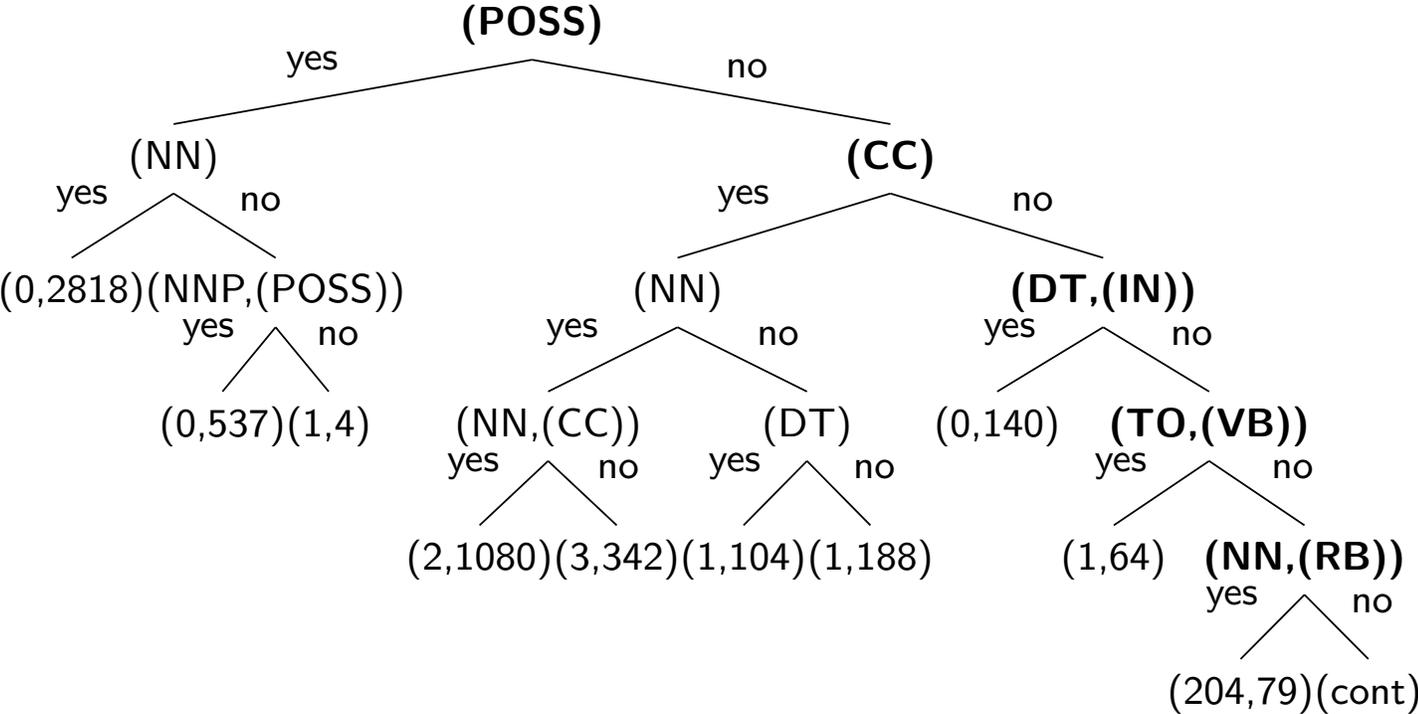
- ▶ sentences from minimum size 1 to maximum size 134 with the average size 22 with
- ▶ the coverage of 81.74% and the BLEU score 0.73 (for the covered data)

# Suspicion tree and Grammar Correction

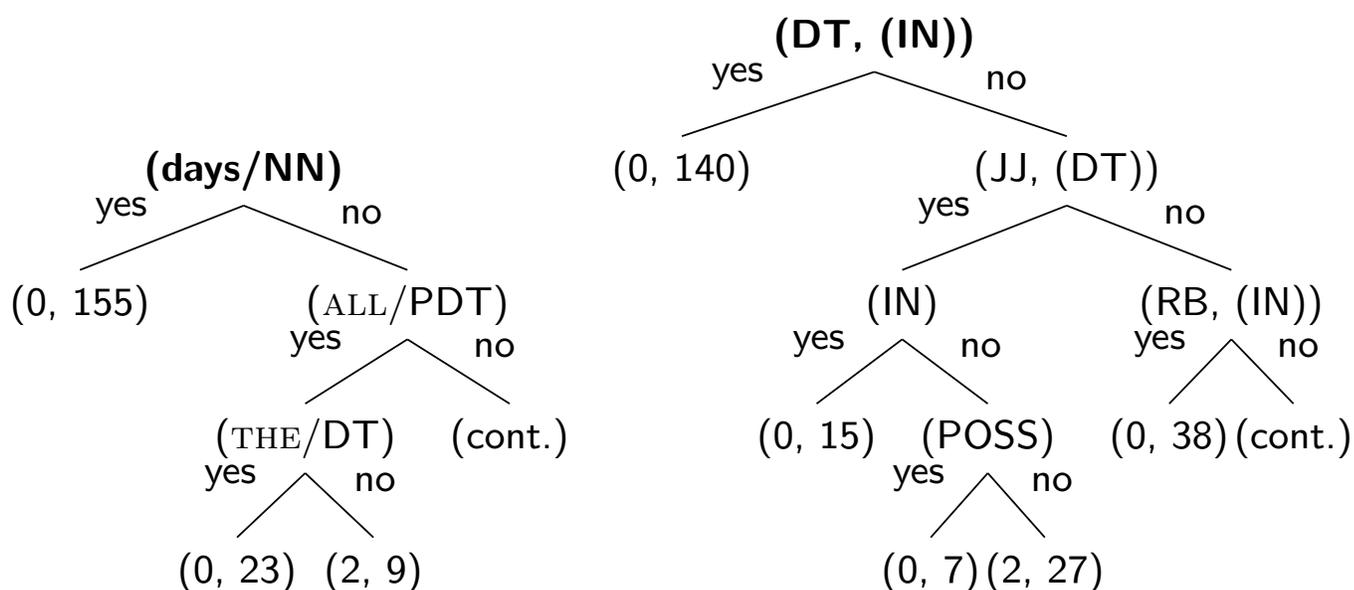
The structure of the suspicion tree helps the linguist correct the grammar by

- ▶ ordering suspicious forms from the most to the least suspicious
- ▶ showing forms that are suspicious independently of context and require a single correction
- ▶ showing forms that are suspicious independently of context and require several corrections (several subcases)
- ▶ showing forms that are suspicious in some but not all contexts

# From the most to the least Suspicious Form



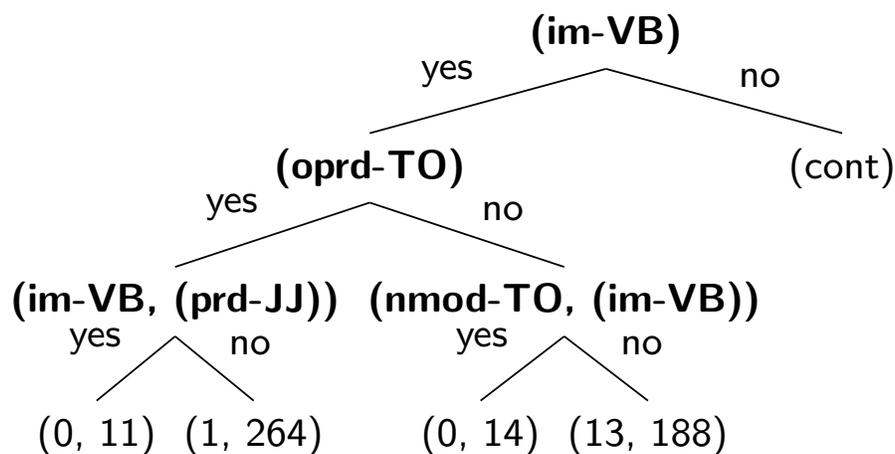
Forms that are suspicious independently of context and require a single correction



days/NN. Incorrect TAG family. Lexicon update

(DT,(IN)) e.g., *some of us*. DT in Input, PRP in TAG. Rewrite Input

## Forms that require several corrections



opr-d-TO: Infinitival verbs which complement a control/raising verb.

*He will **try to assuage** the fears about finances.*

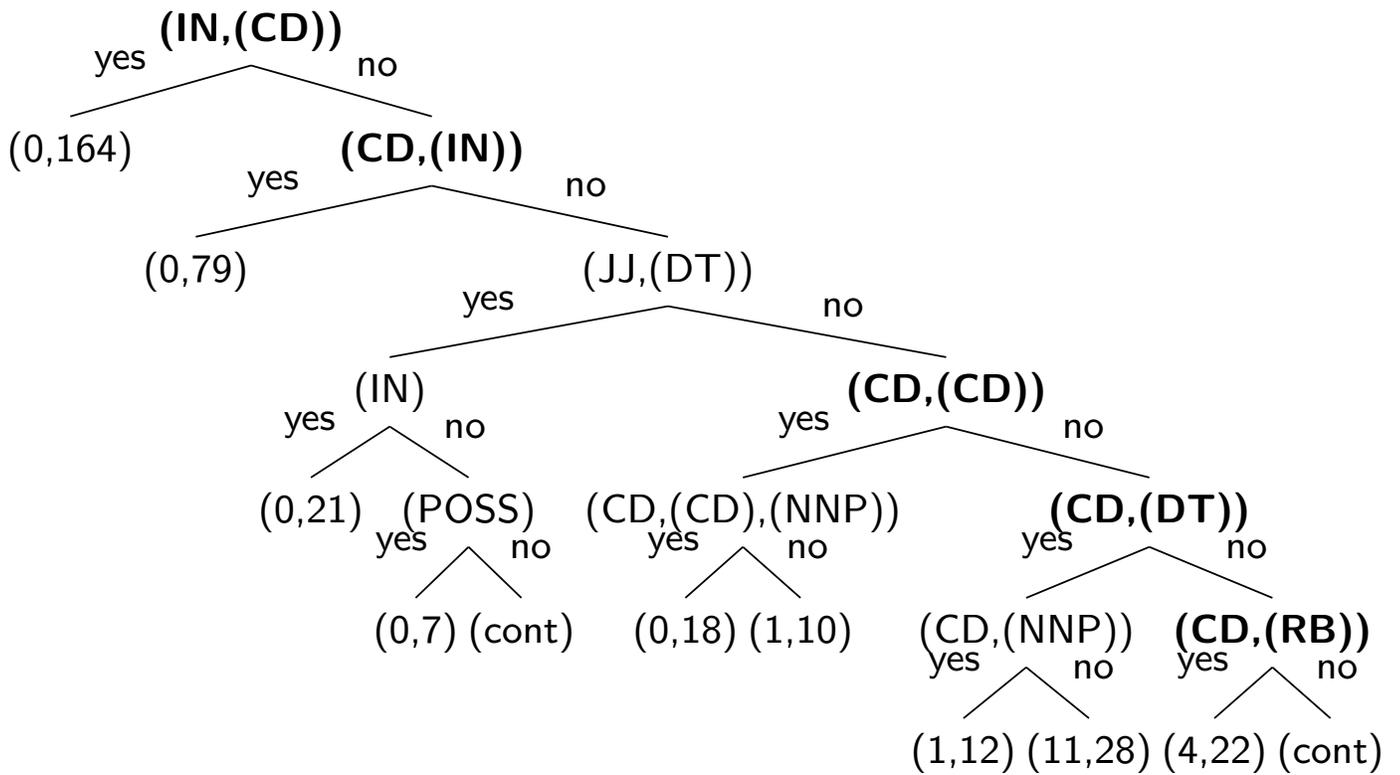
im-VB,(prd-JJ): Inf. verbs subcategorising for an adjectival phrase.

*Amex expects **to be fully operational** by tomorrow.*

im-VB: Infinitival verbs modifying a noun.

*The **ability to trade** without too much difficulty has steadily deteriorated.*

# Forms that are suspicious in some but not all contexts



CD: noun or determiner. Determiner but not noun in TAG lexicon.

N.B., CD does not appear in suspicion tree as it generates fine when used

# Conclusion

- ▶ Proposed a novel approach to error mining which supports a linguistically meaningful error analysis.
- ▶ Permits quickly identifying the main sources of errors while providing a detailed description of the various subcases of these sources if any.
- ▶ We applied it to the analysis of undergeneration in a grammar based surface realisation algorithm.
- ▶ The approach is generic in that permits mining trees and strings for suspicious forms of arbitrary size and arbitrary conjunctions of labelling.