Learning Categorial Grammars from Annotated Corpora

CauLD 2010

Noémie-Fleur Sandillon-Rezer
Supervisors : Richard Moot
Christian Retoré
Géraud Sénizergues

CAuLD, Dec. 2010

Categorial grammars

- Give types to words.
- Types help to determine if sentences are correct.

Jean	aime	Marie
np	$(np\backslash s)/np$	np

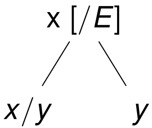
Lambek Calculus

$$\dfrac{\begin{array}{c}\text{Jean}\\ np\end{array} \quad \dfrac{\begin{array}{c}\text{aime}\\ (np\backslash s)/np\end{array} \quad \begin{array}{c}\text{Marie}\\ np\end{array}}{np\backslash s}[/E]}{s}[\backslash E]$$

J. Lambek
The mathematics of sentence structure.
The American Mathematical Monthly, 1958

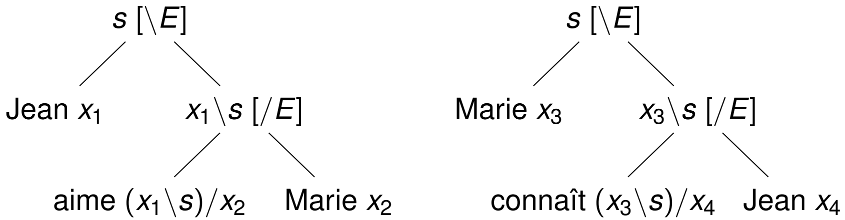
Buszkowski and Penn learning algorithm

- Learn a rigid grammar.
- Binary trees (internal nodes must be labeled $\backslash$ or $/$).
- Type of each node.
- If some words have more than one type, unification phase.



W. Buszkowski and G. Penn
Categorial grammars determined from linguistic data by unification
Studia Logica, 1990

Buszkowski and Penn learning algorithm : the perfect case



Type of the leaves are deduced from the internal nodes.

Jean	:	x_1, x_4
Marie	:	x_2, x_3
aime	:	$(x_1\backslash s)/x_2$
connaît	:	$(x_3\backslash s)/x_4$

Buszkowski and Penn learning algorithm : the perfect case

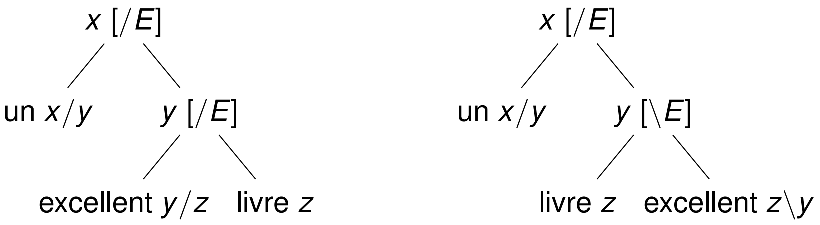
Unification

1. $x_1 = x_4$
2. $x_2 = x_3$
3. $x_1 = x_2$

Result

Jean	:	x_1
Marie	:	x_1
aime	:	$(x_1 \backslash s)/x_1$
connaît	:	$(x_1 \backslash s)/x_1$

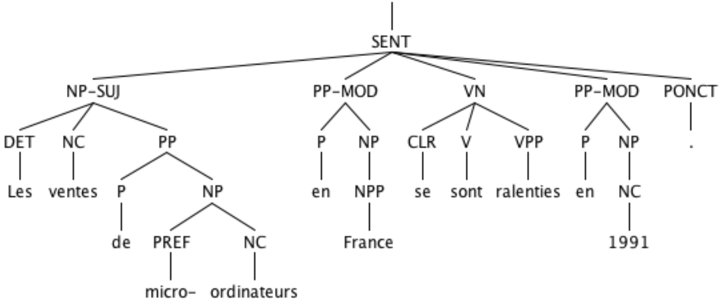
Buszkowski and Penn learning algorithm : problematic case



Type of the leaves are deduced from the internal nodes.

- Problem : the two types for "excellent" cannot be unified.
- Possible solution : *k*-valued grammars. (Problems : unclear how to decide for a global value of k, complexity of learning *k*-valued grammars).

Corpus



Sentence from the corpus of Paris VII.

A. Abeillé and L. Clément
Building a treebank for french
Treebanks, Kluwer, Dordrecht, 2003

Corpus

Limitations

Planar trees $\Rightarrow$ cannot apply usual learning algorithms

Solution

Tree transducer $\Rightarrow$ binarise trees. Assign "$[\backslash E]$" or "$[/E]$" and categories to the nodes depending on the labels given by the annotation.

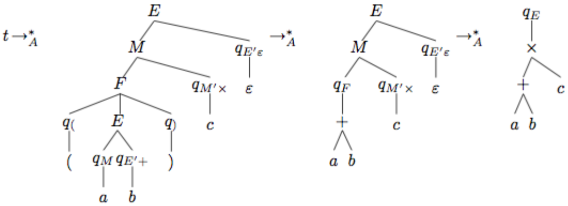
Selection of a reduced corpus

About $4,5\%$ of the corpus : 545 simple verbal sentences.
Extension to the totality of the corpus in a second step.

What is a tree transducer ?

Global principle

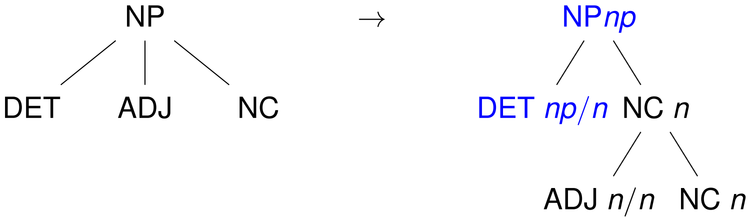
Ask for a tree in input and write a new tree on the output.



Example of bottom-up tree transducer from Tata.

Hubert Comon, Max Dauchet, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi.
Tree automata techniques and applications, 1997.

Small Example

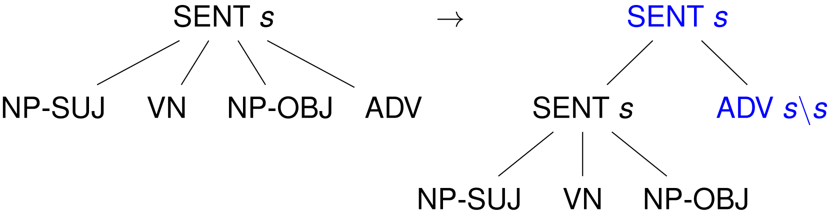


The same rule will be applied for each NP, if the first daughter is a DET, regardless the number or the syntactic category of other daughters.

Why a G-transducer ?

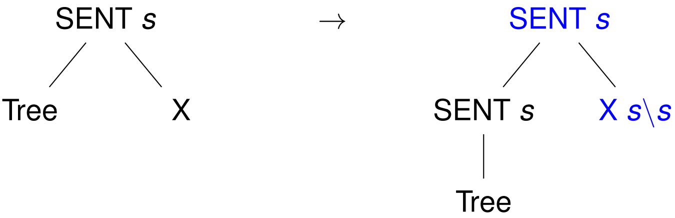
- G stands for Generalized,
- Way to express our rules easily,
- Equivalent to top-down transducers.

G-transducer : recursive



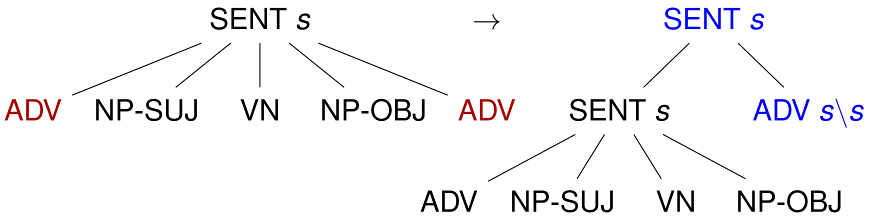
Apply the same set of rules for the first and the second node SENT.
This rule treats the final adverb.

G-transducer : parameterized



The same rule is applied if X = ADV or PP-MOD or AdP-MOD, etc.

G-transducer : priority rules



When more of one rule can be applied, the order of application is always the same.

Specifications

ε-free	:	by the way we defined it, there is no $\varepsilon-$rule in a G-transducer.
Non-erasing	:	each node which occurs in the initial tree occurs in the result too.
Deterministic	:	there is only one possible result tree.
Linear	:	variables of initial tree occur only one time in the transformation.
Ordered	:	order and multiplicity of leaves are kept.

Creation of transduction rules

- In accordance with usual linguistic rules :
  - NP has always *np* type.
  - Adverbs and other modifier must not change the type of their sister node.
  - Adjectives are always linked with the nearest name.
- Systematic study of each sentence of reduced corpus.

Creation of transduction language

```
(rule [options]
    (root pattern)
    (replacement))
```

Keyword "rule"	:	Marks the begining of a rule.
Options	:	Allows to control the rule.
Root	:	Represents the node we want to process.
Pattern	:	Daughters of root. When the keyword *tree* is used it represents an arbitrary sequence of nodes.
Replacement	:	Tree which replace the root and pattern.

Transduction Algorithm

- Search for the matching rule.
- When we find it, the tree is transformed.
- Apply the same process to each daughter we need to transform.
- Forward the type if the node has only one daughter.

Transducer

- Transduce a set of tree.
- Needs a file containing the rules and a file containing trees.

Result

A set of binarised trees where internal nodes and leaves have types.

`Transducer` : options

log	:	Represents the sub-trees that we hadn't treated yet.
ruleUsage	:	Lists the used rules and the time they have been applied.
out	:	Result. In addition to succeeded sentences, it contains : Number of sentences processed. Number of sentences succeeded. Number of failures. Number of rules applied. Number of rules not found.
rule	:	Specifies the rules file.
verbose	:	Add more information on failures.

Lexicalizer et Tregex_converter

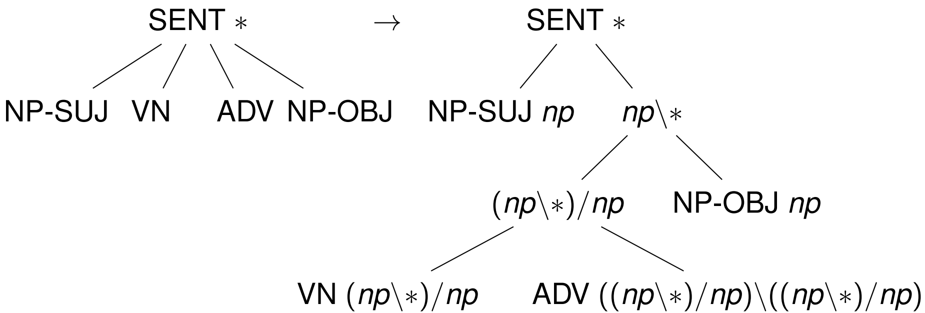
Lexicalizer

Create a lexicon, from a result file given in input.

Tregex_converter

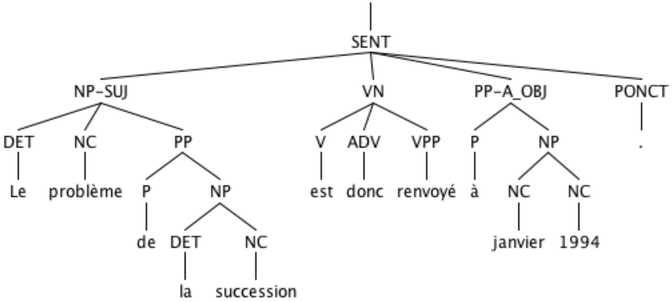
Convert the result file in a Standford Tregex format.

Example of rule



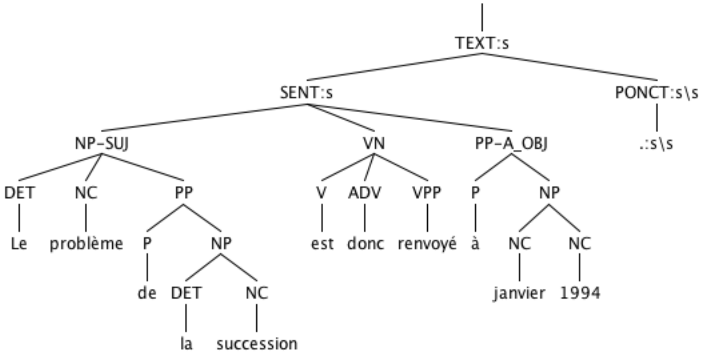

```
(rule
   (SENT:* NP-SUJ NV ADV NP-OBJ)
   ("SENT:*" "NP-SUJ:np"
      (":np\*" (":(np\*)/np" "VN:(np\*)/np"
               "ADV:((np\*)/np)\((np\*)/np)")
            "NP-OBJ:np"))))
```

Example



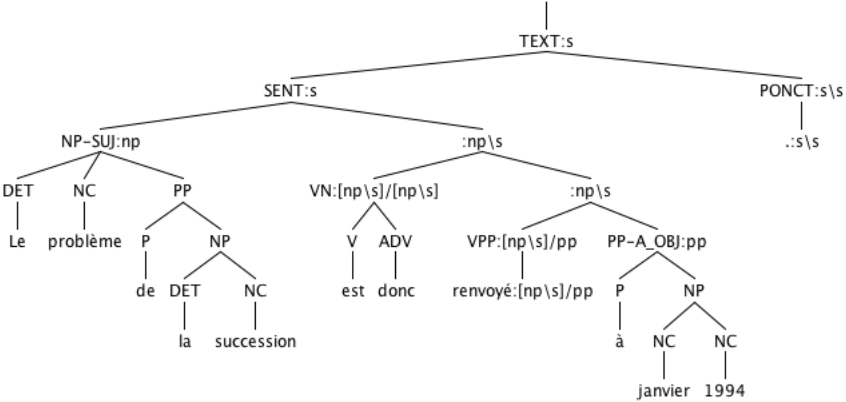
Simple sentence of the reduced corpus.

Example



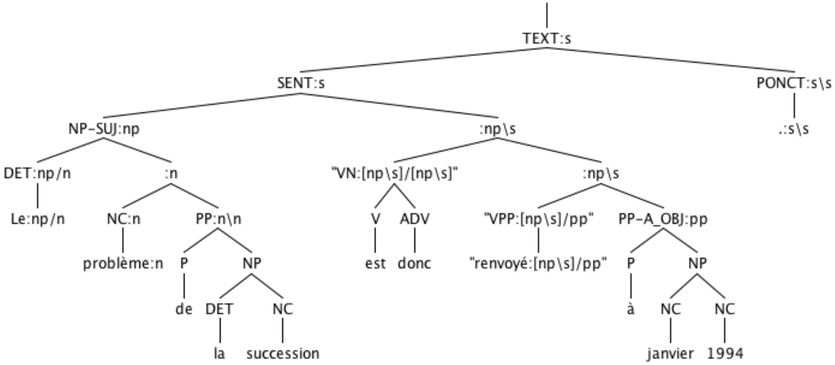
Above all, we apply the punctuation rule.

Example



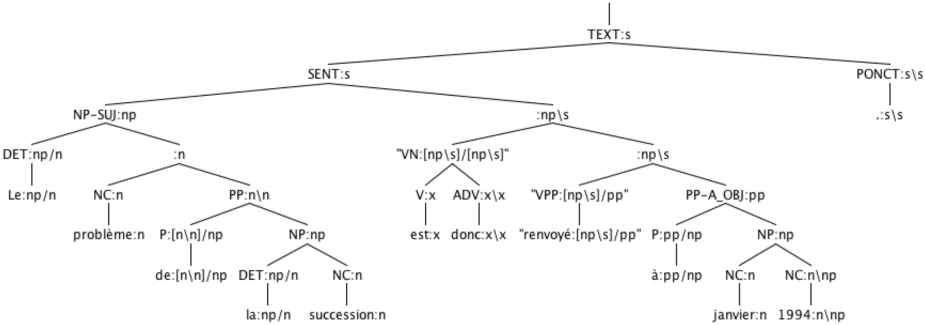
Binarize the sub-tree of *SENT* .

Example



Binarize the remaining sub-tree (*NP-SUJ*).

Example



Computation of types of last binary trees.

Experimental Results

Positive Ones

- 99% of the sentences of the reduced corpus have been treated.
- 2052 words in the lexicon (2056 words in the reduced corpus).
- 29% of the whole corpus is treated yet (about 3700 sentences).
- Lexicon of 20614 words extracted from these 29%.

Words untreated in the reduced corpus.

- Annotation errors.
- Complex cases.

```
171:la - 1:((np\s)\(np\s))/n, 1:(n\n)/n, 1:(s/s)/n,
         1:(s\s)/n, 167:np/n
```

New tools

- Compact description of a transducer :
  - parametrized
  - recursive
  - including a priority system
- Creation of a description langage of transduction and its interpreter.

http://www.labri.fr/perso/nfsr

Extention of existing rules

- Generalization of some rules (*NP*).
- Management of coordination-sentences.

Study of special cases

Non-verbal sentences

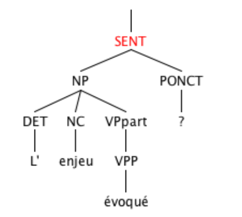


Figure: We can apply existing rules for the sub-tree, but we must create a rule for the SENT node.

Correction of corpus

Ending punctuation.

Analysis of sentences

The resulting lexicon can be used for parsing as follows :

- If a word occurs more than *k* times in the lexicon, use one of its type.
- Else, use one of the type of its category (Part-of-Speech, eg. V, ADV, DET, NC).

Probabilistic Automata

- Generate a tree automaton using the Marion/Besombes algorithm.
- Add weights (probabilities) using frequencies of transitions in the corpus.
- Generate the most probable tree for a given string (using intersection of tree automata and the trivial finite state automaton generating only this string).

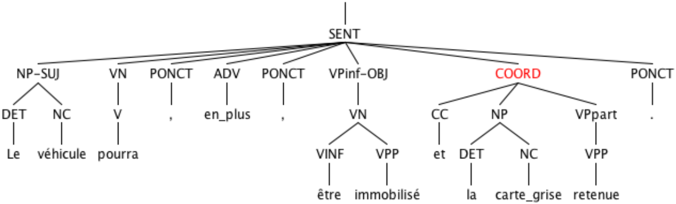
J. Besombes and J.Y. Marion
Learning tree languages from positive examples and membership queries
Algorithmic Learning Theory, 2004

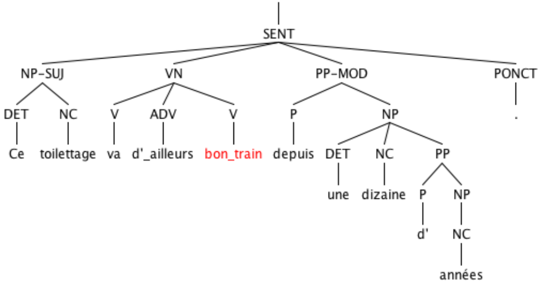
Perspectives

- Refine types of words.
- Extend the transducer to the whole corpus.
- Propose a new learning algorithm.

Thanks for your attention !

Complexes cases





Extracts of lexicon

a 61 :

$((np\backslash s)/(np\backslash s))/(np\backslash s)$ 16
$((np\backslash s)/np)/(np\backslash s)$ 10
$((np\backslash s)/pp)/(np\backslash s)$ 4
$((s/(np\backslash s))/(np\backslash s))/np$ 1
$(np\backslash((np\backslash s)/pp))/(np\backslash s)$ 1
$(np\backslash s)/(np\backslash s)$ 17
$(np\backslash s)/np$ 8
$(s/s)/np$ 4

et 32 :

$((n\backslash n)\backslash(n\backslash n))/(n\backslash n)$ 7
$((n\backslash np)\backslash(n\backslash np))/(n\backslash np)$ 1
$((np/n)\backslash(np/n))/(np/n)$ 1
$((s\backslash s)\backslash(s\backslash s))/(s\backslash s)$ 1
$(n\backslash n)/np$ 7
$(n\backslash np)/np$ 3
$(np\backslash np)/n$ 1
$(np\backslash np)/np$ 2
$(s/s)/(np\backslash s)$ 1
$(s\backslash s)/s$ 8

était 11 :

$((np\backslash s)/np)/(np\backslash s)$ 1
$((np\backslash s)/pp)/(np\backslash s)$ 1
$((np\backslash s)\backslash s)/np$ 1
$(np\backslash s)/(n\backslash n)$ 2
$(np\backslash s)/(np\backslash s)$ 1
$(np\backslash s)/np$ 3
$np\backslash((np\backslash s)/pp)$ 1
$np\backslash s$ 1

le 144 :

$(((np\backslash s)/np)\backslash((np\backslash s)/np))/n$ 1
$(((np\backslash s)/pp)\backslash((np\backslash s)/pp))/n$ 1
$((np\backslash s)\backslash(np\backslash s))/n$ 1
$(np\backslash s)/((np\backslash s)/np)$ 4
$(pp\backslash(np\backslash s))/((pp\backslash(np\backslash s))/np)$ 1
$(s/s)/n$ 2
$(s\backslash s)/((s\backslash s)/np)$ 1
$(s\backslash s)/n$ 6
np/n 126
$s/(s/np)$ 1

Figure: Extracts of lexicon