A Toolbox for Pregroup Grammars

Une boîte à outils pour développer et utiliser les grammaires de prégroupe

Denis Béchet (Univ. Nantes & LINA)

Annie Foret (Univ. Rennes 1 & IRISA)

Denis.Bechet@univ-nantes.fr
http://www.sciences.univ-nantes.fr/info/perso/permanents/bechet

Annie.Foret@irisa.fr, http://www.irisa.fr/prive/foret
Overview

- Grammatical formalism: pregroups
- A pregroup ToolBox: principles and illustrations

Majority (partial) composition and parsing
- Grammar construction

PPQ demo
Grammatical Formalism

- **Pregroup grammars** (PG in short) [Lambek 99]:
  - a simplification of Lambek Calculus [1958]
  - used to describe the syntax of natural languages

- **Extensions** [LATA 2008]:
  we have extended the pregroup calculus with two type constructors that PG are not able to naturally define:
  - * for iteration simple types
  - ? for optional simple types
  and preserve nice properties of PG.
A pregroup \((T, \leq, \cdot, l, r, 1)\) s. t. \((T, \leq, \cdot, 1)\) is a partially ordered monoid in which \(l, r\) are unary operations on \(T\) that satisfy:

\[
 a^l.a \leq 1 \leq a.a^l \quad \text{and} \quad a.a^r \leq 1 \leq a^r.a \quad (PREG)
\]

or equivalently:

\[
 a.b \leq c \iff a \leq c.b^l \iff b \leq a^r.c
\]
A *pregroup* \((T, \leq, \cdot, l, r, 1)\) s. t. \((T, \leq, \cdot, 1)\) is a partially ordered monoid in which \(l, r\) are unary operations on \(T\) that satisfy:

\[
a^l.a \leq 1 \leq a.a^l \quad \text{and} \quad a.a^r \leq 1 \leq a^r.a \quad (PRE)
\]

or equivalently:

\[
a.b \leq c \iff a \leq c.b^l \iff b \leq a^r.c
\]

Some equations follow from the def.

but not, in general:

\[
a^r = a = a^{lr}
\]

\[
a^{rr} \neq a \neq a^{ll}
\]

Iterated adjoints:

\[
\ldots a^{(-2)} = a^{ll}, a^{(-1)} = a^l, a^{(0)} = a, a^{(1)} = a^r, a^{(2)} = a^{rr} \ldots
\]
Pregroup : definitions

A *pregroup* \((T, \leq, \cdot, l, r, 1)\) s. t. \((T, \leq, \cdot, 1)\) is a partially ordered monoid \(^a\) in which \(l, r\) are unary operations on \(T\) that satisfy:

\[
a^l.a \leq 1 \leq a.a^l \quad \text{and} \quad a.a^r \leq 1 \leq a^r.a \quad (PRE)
\]

or equivalently:

\[
a.b \leq c \iff a \leq c.b^l \iff b \leq a^r.c
\]

Some equations follow from the def. but not, in general:

\[
a^{rl} = a = a^{lr} \quad ^b
\]

Iterated adjoints:

\[
\ldots a^{(-2)} = a^{ll}, a^{(-1)} = a^l, a^{(0)} = a, a^{(1)} = a^r, a^{(2)} = a^{rr} \ldots
\]

\(^a\)A partially ordered monoid is a monoid \((M, \cdot, 1)\) with a partial order \(\leq\) s. t.

\[
\forall a, b, c: a \leq b \Rightarrow c \cdot a \leq c \cdot b \quad \text{and} \quad a \cdot c \leq b \cdot c.
\]

\(^b\)we also have:

\[
(a.b)^r = b^r.a^r, \quad (a.b)^l = b^l.a^l, \quad 1^r = 1 = 1^l
\]
Free pregroup

Let \((P, \leq)\) be an ordered set of atomic types,

\[
\text{Types } T_{(P, \leq)} = \{p_1^{(i_1)} \cdots p_n^{(i_n)} \mid 0 \leq k \leq n, p_k \in P \text{ and } i_k \in \mathbb{Z}\}
\]

the empty sequence is denoted by 1.

For \(X\) and \(Y \in T_{(P, \leq)}\) \(X \leq Y\) iff this relation is deductible in the following system

where \(p, q \in P\), \(n, k \in \mathbb{Z}\) and \(X, Y, Z \in T_{(P, \leq)}\): 

\[
\begin{align*}
X & \leq X \quad (\text{Id}) & \frac{X \leq Y}{X \leq Z} & \frac{Y \leq Z}{X \leq Z} \quad (\text{Cut}) \\
XY & \leq Z \quad (AL) & \frac{X \leq YZ}{X \leq Y q^{(n+1)} q^{(n)} Z} (AR) \\
Xp^{(k)}Y & \leq Z \quad (IND_L) & \frac{X \leq Y q^{(k)} Z}{X \leq Y p^{(k)} Z} (IND_R) \\
\end{align*}
\]

\(q \leq p\) if \(k\) is even, and \(p \leq q\) if \(k\) is odd
Pregroup grammar

Let \((P, \leq)\) be a finite partially ordered set.

- A \textit{pregroup grammar} based on \((P, \leq)\) is a lexicalized grammar \(G = (\Sigma, I, s)\) such that
  - \(s \in T_{(P, \leq)}\);
  - \(G\) assigns a type \(X\) to a string \(v_1, \ldots, v_n\) of \(\Sigma^*\) iff for \(1 \leq i \leq n, \exists X_i \in I(v_i)\) such that \(X_1 \cdots X_n \leq X\) in the free pregroup \(T_{(P, \leq)}\).

- \textit{The language} \(\mathcal{L}(G)\) is the set of strings in \(\Sigma^*\) that are assigned \(s\) by \(G\).

\(^a\)a lexicalized grammar is a triple \((\Sigma, I, s)\): \(\Sigma\) is a finite alphabet, \(I\) assigns a finite set of categories (or types) to each \(c \in \Sigma\), \(s\) is a category (or type) associated to correct sentences.
Our example is taken from Lambek, with the atomic types:

\[ \pi_2 = \text{second person}, \]
\[ p_2 = \text{past participle}, \]
\[ o = \text{object}, \]
\[ q = \text{yes-or-no question}, \]
\[ q' = \text{question} \]

This sentence gets type \[ q' (q' \leq s) \):

whom have you seen

\[ q' o \ll q^l \q p_2 \pi_2^l \pi^2_2 \ p_2 \ o^l \]

\[ q' \leq s \]
Our example is taken from Lambek, with the atomic types:

\[ \pi_2 = \text{second person}, \]
\[ p_2 = \text{past participle}, \]
\[ o = \text{object}, \]
\[ q = \text{yes-or-no question}, \]
\[ q' = \text{question} \]

This sentence gets type \( q' (q' \leq s) \):

\[ \text{whom have you seen} \]

\[ q' o^{ll} q \quad q p_2^{l} \pi_2^{l} \quad \pi_2 \quad p_2 \quad o^{l} \]
Our example is taken from Lambek, with the atomic types:

\[ \pi_2 = \text{second person}, \]
\[ p_2 = \text{past participle}, \]
\[ o = \text{object}, \]
\[ q = \text{yes-or-no question}, \]
\[ q' = \text{question} \]

This sentence gets type \( q' (q' \leq s) \):

whom have you seen

\[ q' o^l q^l q p_2^l \pi_2^l \pi_2 p_2^l o^l \]
Our example is taken from Lambek, with the atomic types:

\[
\begin{align*}
\pi_2 &= \text{second person}, \\
p_2 &= \text{past participle}, \\
o &= \text{object}, \\
q &= \text{yes-or-no question}, \\
q' &= \text{question}
\end{align*}
\]

This sentence gets type \( q' (q' \leq s) \):

whom have you seen
Partial Composition

\[ [C] \text{ (partial composition) : for } k \in \mathbb{N}, \]

\[ \Gamma, X p_{1}^{(n_1)} \cdots p_{k}^{(n_k)}, q_{k}^{(n_k+1)} \cdots q_{1}^{(n_1+1)} Y, \Delta \xrightarrow{C} \Gamma, XY, \Delta \]

if \( p_i \leq q_i \) and \( n_i \) is even
or if \( q_i \leq p_i \) and \( n_i \) is odd ,
for \( 1 \leq i \leq k \).

Example :

\[ \Gamma, \begin{array}{c}
\square \quad q' o ll q^l, qp_2^l \pi_2^l \end{array} [1], \Delta \xrightarrow{C} \Gamma, q' o ll p_2^l \pi_2^l, \Delta \]
Majority (Partial) Composition

A partial composition \( \xrightarrow{C} \) is a *majority partial composition* \( \xrightarrow{\@} \) if the width of the result is not greater than the maximum widths of the arguments.

A partial composition that is not a majority composition:

\[
\Gamma, \begin{array}{l}
q' o^l l q^l, q p_2^l \pi_2^l
\end{array}, \Delta \xrightarrow{C} \Gamma, q' o^l l p_2^l \pi_2^l, \Delta
\]

A majority composition:

\[
\Gamma, \begin{array}{l}
q' o^l l q^l, q o^l \pi_2^l
\end{array}, \Delta \xrightarrow{\@} \Gamma, q' \pi_2^l, \Delta
\]
Parsing using Majority Composition

Parsing of “whom have you seen?”

\( q' \leq s \)

whom have you seen

\( q'q^l \quad qp^l_2 \pi^l_2 \quad \pi_2 \quad p_2o^l \)
Parsing algorithm in $n^3$

1. Types for words
   - whom $\mapsto \{q'ollql\}$
   - have $\mapsto \{qp_2\pi_2\}$
   - you $\mapsto \{\pi_2\}$
   - seen $\mapsto \{p_2o_l\}$

2. Add types for words with $GCONC^+$

3. Rec. Calculus of types per segment, with $\@$

4. Test whether the sentence has atomic type $s$ or $x \leq s$
Parsing algorithm in $n^3$

1. Types for words
   - whom $\mapsto \{q' o^{ll} q^{l}\}$
   - have $\mapsto \{qp_2^{l} \pi_2^{l}\}$
   - you $\mapsto \{\pi_2\}$
   - seen $\mapsto \{p_2 o^{l}\}$

2. Add types for words with $GCONC^+$
   - $\mapsto \text{nothing}$

3. Rec. Calculus of types per segment, with $\mapsto$

4. Test whether the sentence has atomic type $s$ or $x \leq s$
Parsing algorithm in $n^3$

1. Types for words
2. $+ \text{ types to words with } GCONC^+$

3. Rec. Calculus of types per segment, with $\Rightarrow$

<table>
<thead>
<tr>
<th>Length = 1:</th>
<th>whom</th>
<th>have</th>
<th>you</th>
<th>seen</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>${q' o^{ll} q^l}$</td>
<td>${qp^l_2 \pi^l_2}$</td>
<td>${\pi_2}$</td>
<td>${p_2 o^l}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Length = 2:</th>
<th>whom have</th>
<th>have you</th>
<th>you seen</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\emptyset$</td>
<td>${qp^l_2}$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Length = 3:</th>
<th>whom have you</th>
<th>have you seen</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>${q' o^{ll} p_2^l}$</td>
<td>${qo^l}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Length = 4:</th>
<th>whom have you seen</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>${q' \text{ and } q' o^{ll} o^l}$</td>
</tr>
</tbody>
</table>

4. Test whether the sentence has atomic type $s$ or $x \leq s$
Parsing algorithm in $n^3$

1. Types for words
2. $+$ types to words with $GCONC^+$
3. Rec. Calculus of types per segment, with $\rightarrow$

- **Length = 1:**
  - whom
  - have
  - you
  - seen
  - $\{q'o^llq^l\}$
  - $\{qp_2^l\pi_2^l\}$
  - $\{\pi_2\}$
  - $\{p_2o^l\}$

- **Length = 2:**
  - whom have
  - have you
  - you seen
  - $\emptyset$
  - $\{qp_2^l\}$
  - $\emptyset$

- **Length = 3:**
  - whom have you
  - have you seen
  - $\{q'o^llp_2^l\}$
  - $\{qo^l\}$

- **Length = 4:**
  - whom have you seen
  - $\{q' \text{ and } q'o^llo^l\}$

4. Test whether the sentence has atomic type $s$ or $x \leq s$:
   - $q' \in$ and $q' \leq s$
PG with ? and * : proposal

**Weakening**

\[
\frac{XY \leq Z}{XP^{(2k+1)} \ Y \leq Z} \quad (\ast - \ W_L)
\]

\[
\frac{X \leq YZ}{X \leq YP^{(2k)} \ Z} \quad (\ast - \ W_R)
\]

**Contraction**

\[
\frac{XP^{(2k+1)} \ Y \leq Z}{XP^{(2k+1)} \ Y \leq Z} \quad (\ast - \ C_L)
\]

\[
\frac{X \leq YP^{(2k)} \ P^{(2k)} \ Z}{X \leq YP^{(2k)} \ Z} \quad (\ast - \ C_R)
\]

\[
\frac{XP^{(2k+1)} \ Y \leq Z}{XP^{(2k+1)} \ Y \leq Z} \quad (\ast - \ C'_L)
\]

\[
\frac{X \leq YP^{(2k)} \ P^{(2k)} \ Z}{X \leq YP^{(2k)} \ Z} \quad (\ast - \ C'_R)
\]
**PG with ? and * : properties**

- **Property.** [Optional and Iterated Basic Types]
  For $a$, a basic type:
  
  $a \leq a^?$  \hspace{1cm}  a^* a \leq a^*$
  
  $a \leq a^?$  \hspace{1cm}  a a^* \leq a^*$
  
  $1 \leq a^?$  \hspace{1cm}  1 \leq a^*$

- **Theorem.** The extended calculus defines a pregroup that extends the free pregroup based on $(P, \leq)$.

- **Theorem.** [The Cut Elimination] The cut rule can be eliminated in the extended calculus: every derivable inequality has a cut-free derivation.

- **Property.** [Decidability]
The provability of $X \leq Y$ in this system is decidable.
PPQ - overview

- Input form
- Lexicon loading
- Type assignment
- Internal reductions
- Result reporting
- Net simplification
- Net calculus
- Majority composition
<?xml version="1.0" encoding="UTF-8"?>
<grammar>
  <pregroup>
    <order inf="n" sup="n-bar"/>
      ...
  </pregroup>
  <sentence type="s"/>
  <lexicon>
    <w><word>whom</word>
      <type><simple atom="q'"/>
        <simple atom="o" exponent="-2"/>
        <simple atom="q" exponent="-1"/>
      </type>
    </w>
      ...
  </lexicon>
</grammar>

Lefff 2.5.5: 534753 entries ➞ SQLite database lexicon.
### PPQ - majority composition

The Matrix Content

<table>
<thead>
<tr>
<th>cell 1-4</th>
<th>q'</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell 1-3</td>
<td>q' o²l p²</td>
</tr>
<tr>
<td>cell 1-2</td>
<td></td>
</tr>
<tr>
<td>cell 1-1</td>
<td>q' o²l q</td>
</tr>
<tr>
<td>cell 2-4</td>
<td>q o¹</td>
</tr>
<tr>
<td>cell 2-3</td>
<td>q p²</td>
</tr>
<tr>
<td>cell 2-2</td>
<td>q p² π²</td>
</tr>
<tr>
<td>cell 3-4</td>
<td></td>
</tr>
<tr>
<td>cell 3-3</td>
<td>π²</td>
</tr>
<tr>
<td>cell 4-4</td>
<td>p² o¹</td>
</tr>
</tbody>
</table>

The table represents the matrix content of the pregroup grammar, with cells containing various elements such as `q`, `q'`, `p`, and `π`. The table structure and content indicate a complex system of relationships amongst these elements, typical in pregroup grammatical analysis.
[FR: Now, when he took back her, he ought to enter]

PPQ can output an XML representation of nets if the result must be used by another program
Now, when he took back her, he ought to enter

[FR: Now, when he took back her, he ought to enter]
Grammar construction

Diagram: construction of PG grammars (with or without "macro-types") and use of CAMELIS

- Allows to define, visualize, navigate, control, query, update

CAMELIS

Prototypes for languages
- English
- French
- Breton

Prototypes for languages

Lexicon with "macros" (.xml)

Complementary lexicon

Order (.xml)

PG model for macro-types

PG model for macro-types

Translator xsl

Traduire-def.xsl

Traduire-def2ctx.xsl

PG types in xml: "define"

XML2CTX/

LIS2XML/

A Toolbox for Pregroup Grammars – p. 19/
Diagram: Around Lefff towards PG grammars (via "macro-types")

1. **Lefff (Source)**
   - LEFFF2CTX/
   - lefff2xml.sed

2. **Lexicon with "macros" (.xml)**
   - ¬<>m™orph=🔗”"!<legen
   - ¬<>m™orph="ms">abstrait</legen

3. **Complementary lexicon**
   - ⇓ ⇓ ⇓

4. **PG model for macro-types**
   - = "define" as string (.xml)
   - <define name="detM"...>

5. **Translator xsl**
   - Traduire-defM.xsl
   - Traduire-def.xsl

6. **PG types in xml: "define"**
   - XML2CTX/
   - Traduire-def2ctx.xsl

7. **LIS context**
   - LIS2XML/
   - XML2CTX/
   - CAMELIS

8. **LIS context (file.ctx)**
   - mk "!" cat is "poncts", ...
   - mk "abstrait" cat is "adj", detail is "ms"

9. **PG grammar (.xml)**
   - <grammar>
   - <pregroup>
   - <phrase type="s"/>

10. **Parser PPQ**
    - ← Raw sentence
    - ← or XML sentence *(corpus or txt2xml.sed, ...)*
    - parse in XML,...

A Toolbox for Pregroup Grammars – p. 20/??
Conclusion

Parser using majority composition – a tool for experiments:

- **Learning Categorial Grammars** (Learnability of Pregroup Grammars [Studia Logica 87(2/3) 2007])
- Allow parsing that follows a partial tree (XML input) and can label subparts of sentences (like named entities)
- To test different ideas, including:
  - extensions of pregroups (*, ?) [LATA 2008]
  - “long distant dependencies” [To appear 2009]
  - different type assignment styles and languages
  - pregroup net sorting and filtering
  - ...

Small demo...