Pregroups, Products, and Generative Power

Gregory M. Kobele

UCLA Department of Linguistics

7. May 2005

Chieti Workshop on Pregroups

Motivation

- One natural intuition is that

  Our linguistic competence is best modeled by a finite set of generators together with operations combining them to produce more complex expressions.

- pregroup grammars (Lambek, 2004) allow us to say that there is one mode of combination, which acts uniformly on strings as concatenation, and on categories as multiplication.

But. . .

- Pregroup grammars are unable even to weakly describe certain constructions in natural language (Shieber, 1985; Buszkowski, 2001). . .
- and there are certain simple intuitions we'd like to express about others, but can't (see Kobele, 2005)

The Italian Nominal and Adjectival Paradigm

Two binary valued features

- masculine ~ feminine
- singular ~ plural

Two kinds of adjective:

	m	**f**
s	*bello*	*bella*
p	*belli*	*belle*

	m	**f**
s	*grande*	*grande*
p	*grandi*	*grandi*

Two kinds of noun:

	m	**f**
s	*gallo*	*rana*
p	*galli*	*rane*

	m	**f**
s	*cane*	*volpe*
p	*cani*	*volpi*

The Italian Nominal and Adjectival Paradigm

What we want to say:

	m	**f**
s	*-o*	*-a*
p	*-i*	*-e*

	m & f
s	*-e*
p	*-i*

1. adjectives and nouns have the same endings
2. some adjectives and nouns only inflect for number

The Italian Nominal and Adjectival Paradigm

Since pregroups operate under adjacency, there’s no way to recover the gender information from *gallo* after it goes through *triste*:

m s	s	m s
`gallo`	`triste`	`bello`

We can separate the ‘lumped together’ information into different tiers:

s	s	s
m		m
`gallo`	`triste`	`bello`

So…

- We would like a way to strengthen pregroup grammars
  - both in terms of their strong, and weak generative capacities
- while keeping as much of their simplicity as possible

Products

- For $P_1 = \langle M_1, \bullet, 1_1, \sqsubseteq, {}^{\mathbf{l}}, {}^{\mathbf{r}} \rangle$ and $P_2 = \langle M_2, \circ, 1_2, \preceq, {}^{L}, {}^{R} \rangle$ pregroups, we can form their direct product $P_1 \times P_2 = \langle M_1 \times M_2, \cdot, \langle 1_1, 1_2 \rangle, \leq, {}^{\ell}, {}^{r} \rangle$, which is also a pregroup. The operations are defined pointwise:
  1. $\langle x, y \rangle \leq \langle x', y' \rangle$ iff $x \sqsubseteq x'$ and $y \preceq y'$
  2. $\langle x, y \rangle \cdot \langle x', y' \rangle = \langle x \bullet x', y \circ y' \rangle$
  3. $\langle x, y \rangle^{\ell} = \langle x^{\mathbf{l}}, y^{L} \rangle$ and $\langle x, y \rangle^{r} = \langle x^{\mathbf{r}}, y^{R} \rangle$

Products

- We relax the definition of a pregroup grammar to allow for both
  - assignment of types to the empty string, and
  - drawing types from *any* pregroup (not just a free pregroup)
- Thus we can say that Buszkowski (2001) showed that (ϵ-free) free pregroup grammars generate exactly the (ϵ-free) context-free languages

Products

An interesting fact:

- Define an operation of 'cross-product' over grammars (i.e. lexica) (for the moment we ignore the possibility of type assignments to the empty string):

$$\mathbb{I}_1 \times \mathbb{I}_2 := \{\langle p_1, p_2, a\rangle : \langle p_1, a\rangle \in \mathbb{I}_1 \text{ and } \langle p_2, a\rangle \in \mathbb{I}_2\}$$

- We have that

$$L(\mathbb{I}_1 \times \mathbb{I}_2) = L(\mathbb{I}_1) \cap L(\mathbb{I}_2)$$

Where we are

- Because (free) pregroup grammars are incapable of describing all the constructions in human language, we want to find a way to extend them
- Looking at patterns of (systematic) syncretism in morphology, we found that we could provide a description of these patterns in the object language if we worked within a product pregroup.
- Now we examine the formal consequences of this move (an open question: how else are we to evaluate it?)
- and we look at interesting natural subclasses the structure of the pregroup formalism makes available to us.

1 Product = 2 Stacks

- We can view a 2-stack automaton as an 8-tuple

$$M := \langle Q, \Sigma, \Gamma, \delta, \#, q_0, Q_f \rangle$$

  where

  – Q, Σ, Γ are finite, pairwise disjoint sets (of states, input symbols, and stack symbols, respectively)

  – $Q_f \subseteq Q$ is the set of final states

  – $q_0 \in Q$ is the initial state

  – $\# \notin \Gamma$ is the empty stack symbol

  – $\delta : Q \times \Sigma_\epsilon \times (\Gamma \cup \{\#\}) \times (\Gamma \cup \{\#\}) \rightarrow 2^{Q \times \Gamma^* \times \Gamma^*}$ is the transition function.

1 Product = 2 Stacks

- An instantaneous description $id \in \Gamma^*\{\#\}\Gamma^* Q\Sigma^*$.
  - We define a relation $\Rightarrow$ over the set of instantaneous descriptions as follows, for $\gamma, \gamma', \eta, \eta' \in \Gamma^*$, $\sigma \in \Sigma^*$, $g, g' \in \Gamma$, $a \in \Sigma_\epsilon$, $q, q' \in Q$:
    1. $\gamma g \# \gamma' g' q a \sigma \Rightarrow \gamma \eta \# \gamma' \eta' q' \sigma$
       iff $\langle q', \eta, \eta' \rangle \in \delta(\langle q, a, g, g' \rangle)$
    2. $\# \gamma' g' q a \sigma \Rightarrow \eta \# \gamma' \eta' \# q' \sigma$
       iff $\langle q', \eta, \eta' \rangle \in \delta(\langle q, a, \#, g' \rangle)$
    3. $\gamma g \# q a \sigma \Rightarrow \gamma \eta \# \eta' \# q' \sigma$
       iff $\langle q', \eta, \eta' \rangle \in \delta(\langle q, a, g, \# \rangle)$
    4. $\# q a \sigma \Rightarrow \eta \# \eta' q' \sigma$
       iff $\langle q', \eta, \eta' \rangle \in \delta(\langle q, a, \#, \# \rangle)$

- the language of a 2-stack automaton is here defined in terms of empty stacks and final state:

$$L(M) := \{\sigma : \exists q_f \in Q_f .\ \# q_0 \sigma \Rightarrow^* \# q_f\}$$

1 Product = 2 Stacks

- Given a 2-stack automaton $M = \langle Q, \Sigma, \Gamma, \delta, \#, q_0, Q_f \rangle$, we construct an equivalent pregroup grammar as follows:
  1. Let P be the free pregroup over $Q \cup \Gamma \cup \{\#\} \cup \{s\}$, where s is a new symbol not in $Q \cup \Gamma \cup \{\#\}$. We draw types from $P \times P$.

1 Product = 2 Stacks

- Instead of $\langle b_1, b_2, a\rangle$ we write

$$\begin{pmatrix} b_1 \\ b_2 \\ \mathtt{a} \end{pmatrix}$$

- The intuition behind the translation:

  An expression has the form

$$\begin{pmatrix} \#\gamma^{\ell} q \\ \#\gamma'^{\ell} q \\ \mathtt{w} \end{pmatrix}$$

  and intuitively represents an instantaneous description

$$rev(\gamma)\#rev(\gamma')q\sigma$$

  Or rather, a machine in state q with $rev(\gamma)$ in the first stack, and $rev(\gamma')$ in the second.

1 Product = 2 Stacks

$\mathbb{I}$ is the smallest set containing

1. for q_0 the start state,

$$\begin{pmatrix} \#q_0 \\ \#q_0 \\ \epsilon \end{pmatrix}$$

2. for $q_f \in Q_f$ a final state,

$$\begin{pmatrix} q_f^r \#^r s \\ q_f^r \#^r s \\ \epsilon \end{pmatrix}$$

3. for $\langle q', rev(\eta), rev(\eta') \rangle \in \delta(\langle q, a, g, g' \rangle)$, where $g, g' \in \Gamma \cup \{\#\}$,

$$\begin{pmatrix} q^r g \eta^\ell q' \\ q^r g' \eta'^\ell q' \\ \mathsf{a} \end{pmatrix}$$

Interim Summary

- We can thus “get everything” without losing any of the nice properties of the pregroup formalism.
- However, now our syntax doesn’t restrict the class of languages weakly generated!

On not getting everything

- Can we find any "natural" subclasses of pregroup grammars (in our new sense) that get something like the "right" family of languages?
- A natural option is to place restrictions on allowable types – either in the lexicon, or in general:
  - Lambek (2004) gives a "performance restriction", which restricts types to those of length less than n
  - another option is to place a condition on the lexicon
    - ∗ in the 2-stack translation, we had lexical types which had multiple atoms in them, and so this might seem a natural restriction,
    - ∗ however, we can simulate a queue automaton just using lexical types of the form $a\alpha^{\ell}$, and $\alpha^{r}a$, which seem pretty simple

Global Index Grammars

- Castaño (2004) introduces Global Index Grammars (GIGs) as a variant of (linear) indexed grammars – instead of associating a stack with a non-terminal, there is a single, global, stack accessible to everything.

- The Global Index Languages (GILs) are semi-linear and bounded polynomially parsable. They contain non- Multiple Context-Free Languages (MCFLs), like the multiple copy language $\{ww^+ : w \in \Sigma^*\}$, and it is an open question whether the MCFLs are properly included in the GILs, or not.

- We can also look at GIGs as context-free grammars with productions labeled by subwords of a Dijk language: $x, \overline{x}, \overline{x}x, \epsilon$, thus connecting with the tradition of grammars with controlled derivations (Dassow and Păun, 1989).

Global Index Grammars

- Castaño places two restrictions on GIGs (above and beyond them being CFGs labeled in the above way):
  1. only rules in Greibach Normal Form ($A \rightarrow aB_1 \ldots B_n$) can be labeled with an opening parenthesis (x)
  2. rules labeled with either an opening (x) or a closing ($\overline{x}$) parenthesis can only be used in a derivation if they are rewriting the left-most non-terminal

Global Index Grammars

- Given a GIG $G = \langle N, T, I, S, \#, P\rangle$, where all productions in P are in GNF, we construct a pregroup grammar as follows
  1. Let P_1 be the free pregroup over N, and P_2 the free pregroup over I. We draw types from $P_1 \times P_2$.
- The intuition behind the translation:

  An expression has the form

$$\begin{pmatrix} AB_n^\ell \dots B_1^\ell \\ \delta \\ \mathtt{w} \end{pmatrix}$$

  where δ is a substring of a Dijk word, and $AB_n^\ell \dots B_1^\ell$ is a context-free production in GNF

Global Index Grammars

$\mathbb{I}$ is the smallest set containing, for each $A \rightarrow_{\delta} aB_1 \ldots B_n \in P$, the expression

$$\begin{pmatrix} AB_n^{\ell} \ldots B_1^{\ell} \\ \delta' \\ \mathsf{a} \end{pmatrix}$$

where,

if δ is	then δ' is
ϵ	ϵ
x	x
$\overline{x}$	x^r
$\overline{x}x$	$x^r x$

Global Index Grammars

What about the restriction to left-most derivation?!

- pregroup grammars always yield a ‘left-corner’ derivation
- but when a CFG is in GNF, ‘left-corner’ coincides with left-most

Thus we don’t have to make the additional stipulation Castaño makes in his system – we get it ‘for free’.

Summary

- Drawing types from products of free pregroups increases the generative power of pregroup grammars, and allowing the empty string to be assigned a type in this setting makes them r.e.
- Intersection of languages can be modeled by taking the cross-product of the respective lexica (allowing the empty string gives us in essence closure under erasing homomorphisms).
- By implementing a simple lexical restriction on type assignments, we can define a class of pregroup grammars that are semi-linear.
- Pregroups have a ‘built-in’ leftmost-derivation-like property, which allows us to give a simpler statement of Castaño’s restrictions.

References

Buszkowski, W. (2001). Lambek grammars based on pregroups. In P. de Groote, G. Morrill, and C. Retoré (Eds.), *Logical Aspects of Computational Linguistics*, Volume 2099 of *Lecture Notes in Artificial Intelligence*. New York: Springer.

Castaño, J. M. (2004). *Global Index Languages*. Ph. D. thesis, Brandeis University.

Dassow, J. and G. Păun (1989). *Regulated Rewriting in Formal Language Theory*. Berlin: Springer-Verlag.

Kobele, G. M. (2005). Agreement bottlenecks in Italian. ms. UCLA.

Lambek, J. (2004, August). A computational algebraic approach to English grammar. *Syntax 7*(2), 128–147.

Shieber, S. M. (1985). Evidence against the context-freeness of natural language. *Linguistics and Philosophy 8*, 333–343.