NP-Completeness of Grammars Based Upon Products of Free Pregroups

Denis Béchet

LINA - UMR 6241,
Université de Nantes & CNRS
2, rue de la Houssiniére - BP 92208
44322 Nantes Cedex 03 - France
Denis.Bechet@univ-nantes.fr

Abstract. Pregroup grammars are context-free lexicalized grammars based upon free pregroups which can describe parts of the syntax of natural languages. Some extensions are useful to model special constructions like agreements with complex features or non-projective relations or dependencies. A simple solution for these problems is given by lexicalized grammars based upon the product of free pregroups rather than on a single free pregroup. Such grammars are not necessarily context-free. However, the membership problem is NP-complete. To prove this theorem, the article defines a particular grammar built on the product of three free pregroups. This grammar is used to encode any SAT problem as a membership problem in the language corresponding to the grammar.
Keywords: Lambek Categorial Grammar, Pregroup Grammar, Free Pregroup, Product of Pregroups.

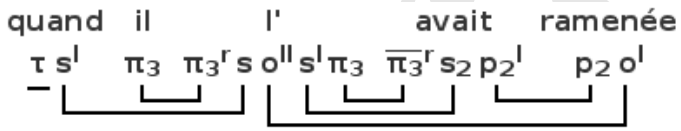
1 Introduction

Pregroup grammars [15] are a simplification of Lambek calculus [18] that can model parts of several natural languages: English [15], Italian [8], French [1], German [16, 17], Japanese [7], Persian [19], etc. As with Lambek calculus, some extensions have been proposed for various constructions. For instance, in [4, 3], simple type iterations are introduced into pregroup grammars for adjectival or adverbial phrases. [11] presents other extensions based upon modalities, product pregroup grammars and tupled pregroup grammars and applies these extensions to Polish.

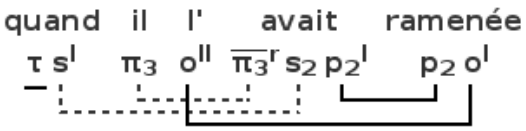
In [12, 13], the author proposes to use products of pregroups as a general construction to extend the generative power of pregroup grammars based upon free pregroups. For the author, this construction is interesting, for instance, for the Italian nominal and adjectival paradigm with binary valued features (masculine/feminine or singular/plural). With a product, every feature can be put in its own space giving a very simple solution for agreement. In [14], grammars based upon any pregroup (not only on a free pregroup) are proved to be Turing complete (the languages are all the ϵ-free recursively enumerable languages).

The construction uses a grammar based upon the product of two free pregroups and its image through a string homomorphism.

Products can also be used for long distance dependencies, in particular when the projective nature of free pregroup deductions limit axioms. For instance a pregroup analysis of “quand il l’avait ramené” (when he took her at home) needs a complex type for the clitic “ l’ ” (her).



“l’ ” is assigned $\pi_3^r s o^{ll} s^l \pi_3$ rather than o^{ll}. A better analysis would be:



For this kind of constructions, we need non-projective axioms. The product of free pregroups can be used for this purpose. The idea is used by Categorial Dependency Grammars[9] for non-projective dependencies even if in this case the polarities that define the ends of non-projective dependencies do not define complete categorial components[1].

Of course, we expect that such extensions preserve the interesting properties of pregroup grammars. One of these properties is that the membership problem is polynomial. This is no more the case with grammars based upon the products of at least 3 free pregroups: Here, we present a grammar based upon the product of 3 free pregroups that can code any SAT problem proving that the membership problem for this grammar is NP-hard.

The rest of the article begins by presenting pregroups, free pregroups and pregoup grammars (lexicalized grammars based on a free pregroup). Section 3 introduces the product of pregroups, pregroup product grammars (lexicalized grammars based on the product of several free pregroups) and gives some properties of the associated class of languages. Section 4 proves that the membership problem of the class of languages is NP-hard (in fact NP-complete). The last section concludes.

[1] The types in a CDG can be defined as the set of the product of a categorial type and a list of signed integers

2 Background

Definition 1 (Pregroup). *A* pregroup *is a structure* $(P, \leq, \circ, l, r, 1)$ *such that* $(P, \leq, \circ, 1)$ *is a partially ordered monoid* [2] *and* l, r *are two unary operations on* P *that satisfy the inequalities* $x^l x \leq 1 \leq x x^l$ *and* $x x^r \leq 1 \leq x^r x$ *for all* $x \in P$.

Definition 2 (Free Pregroup). *Let* $(P, \leq)$ *be a partially ordered set of basic types. We write* $\mathbb{Z}$ *for the set of signed integers.* $P^{(\mathbb{Z})} = \{ p^{(i)} \mid p \in P,\ i \in \mathbb{Z} \}$ *is the set of* simple types *and* $T_{(P,\leq)} = \left(P^{(\mathbb{Z})}\right)^* = \{ p_1^{(i_1)} \cdots p_n^{(i_n)} \mid 0 \leq k \leq n,\ p_k \in P \text{ and } i_k \in \mathbb{Z} \}$ *is the set of* types. *The empty sequence in* $T_{(P,\leq)}$ *is denoted by* 1. *For* X *and* $Y \in T_{(P,\leq)}$, $X \leq Y$ *iff this relation is derivable in the following system where* $p, q \in P$, $n, k \in \mathbb{Z}$ *and* $X, Y, Z \in T_{(P,\leq)}$*:*

$$X \leq X \ \ (Id) \qquad \frac{X \leq Y \quad Y \leq Z}{X \leq Z}\,(Cut)$$

$$\frac{XY \leq Z}{X p^{(n)} p^{(n+1)} Y \leq Z}\,(A_L) \qquad \frac{X \leq YZ}{X \leq Y p^{(n+1)} p^{(n)} Z}\,(A_R)$$

$$\frac{X p^{(k)} Y \leq Z}{X q^{(k)} Y \leq Z}\,(IND_L) \qquad \frac{X \leq Y q^{(k)} Z}{X \leq Y p^{(k)} Z}\,(IND_R)$$

(where $q \leq p$ *if* k *is even, and* $p \leq q$ *if* k *is odd)*

The construction, proposed by Buskowski [6], defines a pregroup that extends $\leq$ on basic types P to $T_{(P,\leq)}$ [3] [4].

Cut Elimination. On the one hand, the cut rule is useful for clear and compact representation of derivations. On the other hand, it creates problems for derivation search because, due to this rule, one cannot in general bound the number of hypothetical premises needed in a derivation of a given inequality. Fortunately, this rule can be eliminated in pregroups without loss of generality, i.e. every derivable inequality has a cut-free derivation (see [5]) .

Definition 3 (Pregroup Grammar). *Let* $(P, \leq)$ *be a finite partially ordered set. A* pregroup grammar *based upon* $(P, \leq)$ *is a lexicalized* [5] *grammar* $G =$

[2] We briefly recall that a *monoid* is a structure $< M, \circ, 1 >$, such that $\circ$ is associative and has a neutral element 1 ($\forall x \in M : 1 \circ x = x \circ 1 = x$). A partially ordered monoid is a monoid $< M, \circ, 1 >$ with a partial order $\leq$ that satisfies $\forall a, b, c$: $a \leq b \Rightarrow c \circ a \leq c \circ b$ and $a \circ c \leq b \circ c$.

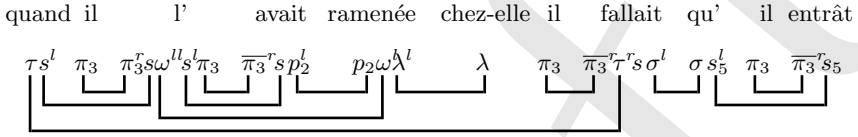
[3] Left and right adjoints are defined by $(p^{(n)})^l = p^{(n-1)}$, $(p^{(n)})^r = p^{(n+1)}$, $(XY)^l = Y^l X^l$ and $(XY)^r = Y^r X^r$. p stands for $p^{(0)}$. The left and right adjoints of $X \in T_{(P,\leq)}$ are defined recursively: $X^{(0)} = X$, $X^{(n+1)} = (X^r)^{(n)}$ and $X^{(n-1)} = (X^l)^{(n)}$.

[4] $\leq$ is only a preorder. Thus, in fact, the pregroup is the quotient of $T_{(P,\leq)}$ under the equivalence relation $X \leq Y \ \& \ Y \leq X$.

[5] A lexicalized grammar is a triple (Σ, I, s): Σ 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 strings.

(Σ, I, s) *on categories* $T_{(P,\leq)}$ *such that* $s \in P$. *G assigns a type* X *to a string* $v_1 \cdots v_n$ *of* Σ^* *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* $T_{(P,\leq)}$. The language $\mathcal{L}(G)$ *is the set of strings in* Σ^* *that are assigned* s *by* G.

Example 1. Let us look at an analysis of a complete sentence from Marcel Proust (a part of it is shown in the introduction). The basic types used in this analysis are: $\pi_3, \overline{\pi_3}$: third person (subject) with $\pi_3 \leq \overline{\pi_3}$, p_2: past participle, ω: object, s: sentence, s_5: subjunctive clause, σ: complete subjunctive clause, τ: adverbial phrase.



Using only left rules (A_L) and (IND_L) and one (Id), we can prove that the product of the assigned types is less than or equal to s. The proof is schematically presented above. In this proof, each link corresponds to one application of (A_L) (eventually with a (IND_L) when the corresponding basic types are different).

3 Product of Pregroups

A natural idea to combine pregroups is to define a structure over the product of the corresponding monoids.

Definition 4 (Product of Pregroups).
For $N \geq 1$, *let* $P_i = (M_i, \leq_i, \circ_i, l_i, r_i, 1_i)$, $1 \leq i \leq N$, *be* N *pregroups. We define* $P_1 \times \cdots \times P_N$ *as* $(M_1 \times \cdots \times M_N, \leq, \circ, l, r, (1_1, \ldots, 1_N))$ *where:*
- $(x_1, \ldots, x_N) \leq (y_1, \ldots, y_N)$ *iff* $\forall i, 1 \leq i \leq N$, $x_i \leq_i y_i$,
- $(x_1, \ldots, x_N) \circ (y_1, \ldots, y_N) = (x_1 \circ_1 y_1, \ldots, x_N \circ_N y_N)$,
- $(x_1, \ldots, x_N)^l = (x_1^{l_1}, \ldots, x_N^{l_N})$ *and* $(x_1, \ldots, x_N)^r = (x_1^{r_1}, \ldots, x_N^{r_N})$.

The product of several pregroups gives a structure that is also a pregroup[6].

3.1 Pregroup Product Grammars

Pregroup grammars are defined over a free pregroup. We relax this definition here and define grammars on any pregroup. In fact, we are interested only in the product of free pregroups.

[6] The definition can also be extended to the empty product ($N = 0$). In this case, the resulting structure is the monoid with a unique element which is also the unit element.

Definition 5 (Pregroup Product Grammar).
Let $(P_1, \leq_1), \ldots, (P_N, \leq_N)$ *be* $N \geq 1$ *finite partially ordered sets. A* pregroup product grammar *based upon* $(P_1, \leq_1), \ldots, (P_N, \leq_N)$ *is a lexicalized grammar* $G = (\Sigma, I, s)$ *on categories* $T_{(P_1,\leq_1)} \times \cdots \times T_{(P_N,\leq_N)}$ *such that* $s \in P_1$. *G assigns a type* X *to a string* $v_1 \cdots v_n$ *of* Σ^* *iff for* $1 \leq i \leq n$, $\exists X_i \in I(v_i)$ *such that* $X_1 \circ \cdots \circ X_n \leq X$ *in the product of the free pregroups* $T_{(P_1,\leq_1)}, \ldots, T_{(P_N,\leq_N)}$ *with* $\circ$ *as the binary operation of the product and* $\leq$ *as its partial order.* The language $\mathcal{L}(G)$ *is the set of strings in* Σ^* *that are assigned* $(s, 1, \ldots, 1)$ *by G.*

In the definition, when a string is assigned $(s, 1, \ldots, 1)$, the first component of the product must be less than or equal to s, a special basic type of the first free pregroup. The other components must be less than or equal to the unit of the corresponding free pregroup. It is possible to have a different definition, for instance by choosing that all components must be less than or equal to the unit of this component and by adding a "wall" in Σ that is associated by the lexicon to the type $(s^r, 1, \ldots, 1)$.

3.2 Pregroup Product Grammars: Context Sensitive but NP Membership Problem

The membership problem of a string into the language associated to a pregroup grammar is polynomial in time either from the size of the string or from the size of the string plus the size of the pregroup grammar. In fact, the languages of pregroup grammars are the context-free languages [6, 2].

For pregroup product grammars, the number $N \geq 1$ of free pregroups for the product is important. For $N = 1$, the product is equivalent to a free pregroup. Thus the same result can be proved on the membership problem and the expressive power (the membership problem is polynomial in time and the languages are context-free). This is completely different if $N > 1$. With regard to the expressive power, the article proves that for $k > 1$, $L_k = \{a_1^i a_2^i \cdots a_k^i \mid i \geq 1\}$ is generated by a pregroup product grammar based upon the product of $k-1$ free pregroups.

Definition 6 (Pregroup Product Grammar for $\{a_1^i a_2^i \cdots a_k^i \mid i \geq 1\}$).
Let $P = (\{x_1, \ldots, x_k, z\}, =)$ *a partially ordered set (the partial order on basic types is equality). We consider the product of the k free pregroups based upon k copies of P. Let* $G_k = (\{a_1, \ldots, a_k\}, I_k, x_1)$ *be the pregroup product grammar based upon the product and defined by the following lexicon:*

$$
\begin{aligned}
I_k(a_1) &= \{\ (x_1 z x_1^l, 1, \ldots, 1)\ ,\ (x_1 z x_2^l, 1, \ldots, 1)\ \} \\
I_k(a_2) &= \{\ (x_2 z^l x_2^l, z, 1, \ldots, 1)\ ,\ (x_2 z^l x_3^l, z, 1, \ldots, 1)\ \} \\
I_k(a_3) &= \{\ (x_3 x_3^l, z^l, z, 1, \ldots, 1)\ ,\ (x_3 x_4^l, z^l, z, 1, \ldots, 1)\ \} \\
&\ldots \\
I_k(a_{k-1}) &= \{\ (x_{k-1} x_{k-1}^l, 1, \ldots, 1, z^l, z)\ ,\ (x_{k-1} x_k^l, 1, \ldots, 1, z^l, z)\ \} \\
I_k(a_k) &= \{\ (x_k x_k^l, 1, \ldots, 1, z)\ ,\ (x_k, \ldots, 1, z)\ \}
\end{aligned}
$$

Theorem 1. *For* $k > 1$, $L_k = \{a_1^i a_2^i \cdots a_k^i \mid i \geq 1\} = \mathcal{L}(G_k)$.

Proof. Firstly, it is easy to find a derivation in G_k corresponding to the string $A = a_1^i a_2^i \cdots a_k^i$ for $i > 0$: Using the lexicon I_k, we can associate the following expression to A:

$$\underbrace{(x_1 z^l x_1^l, 1, \ldots, 1) \circ \cdots \circ (x_1 z^l x_1^l, 1, \ldots, 1)}_{i-1} \circ (x_1 z^l x_2^l, 1, \ldots, 1) \circ$$
$$\underbrace{(x_2 z x_2^l, z^l, 1, \ldots, 1) \circ \cdots \circ (x_2 z x_2^l, z^l, 1, \ldots, 1)}_{i-1} \circ (x_2 z x_3^l, z^l, 1, \ldots, 1) \circ$$
$$\cdots$$
$$\underbrace{(x_{k-1} x_{k-1}^l, 1, \ldots, 1, z, z^l) \circ \cdots \circ (x_{k-1} x_{k-1}^l, 1, \ldots, 1, z, z^l)}_{i-1} \circ (x_{k-1} x_k^l, 1, \ldots, 1, z, z^l) \circ$$
$$\underbrace{(x_k x_k^l, 1, \ldots, 1, z) \circ \cdots \circ (x_k x_k^l, 1, \ldots, 1, z)}_{i-1} \circ (x_k, 1, \ldots, 1, z)$$

For the first component:

$$\underbrace{(x_1 z^l x_1^l) \cdots (x_1 z^l x_1^l)}_{i-1} x_1 z^l x_2^l \underbrace{(x_2 z x_2^l) \cdots (x_2 z x_2^l)}_{i-1} x_2 z x_3^l$$
$$\underbrace{(x_3 x_3^l) \cdots (x_3 x_3^l)}_{i-1} x_3 x_4^l \cdots \underbrace{(x_k x_k^l) \cdots (x_k x_k^l)}_{i-1} x_k \leq x_1$$

For the other components:

$$1 \cdots 1 \underbrace{z^l \cdots z^l}_{i} \underbrace{z \cdots z}_{i} 1 \cdots 1 \leq 1$$

Therefore $L_k \subseteq \mathcal{L}(G_k)$.

For the other direction, we prove that if $A \in \mathcal{L}(G_k)$ then for $1 \leq i \leq k-1$, every occurrence of a_i in A must be before any occurrence of a_{i+1} and the number of a_i is the same as the number of a_{i+1}. The first property is given by the basic types $x_1, \ldots x_k$ of the first component of the derivation of A in G_k. Couples of x_i^l and x_i form a list from left to right leaving only one basic type x_1. For the second property, the number of a_i is the same as the number of a_{i+1}, because in the i-th component, basic type z is given by a_i as z^l and by a_{i+1} as z (each z on the right corresponds exactly to one z^l on the left).

In the Chomsky hierarchy, a pregroup product grammar can be simulated by a context-sensitive grammar using contextual rules. Intuitively, in the context-sensitive grammar, some contextual rules play the role of the free pregroup left rules (A_L) and (IND_L) of Definition 2. A second set of contextual rules performs local "permutations" of simple types that are not in the same component: A simple type in the i-th component permutes with a simple type in the j-th component if the first one is before the second one and if $i > j$.

In fact, the membership problem is clearly a NP problem because if we want to check that a string is in the language associated to a pregroup product grammar where the product has N components, we only have to produce an assignment for each symbol and prove that the N concatenations of each component of the types are less than or equal to s or 1 which are N polynomial problems.

The conclusion of this remark is that the languages of pregroup product grammars are contextual but most probably several context-sensitive language are not generated by a pregroup product grammar (the membership problem of the context-sensitive languages is PSPACE-complete). The next section proves that the membership problem is also NP-hard. Thus pregroup product grammars are not mildly context-sensitive [10].

4 Pregroup Product Grammars: NP-hard

The section presents the main result of the paper: The membership problem for a particular pregroup product grammar is NP-hard. The proof is based upon an encoding of any SAT problem. The grammar is based upon the product of 3 free pregroups. As a consequence, the membership problem of pregroup product grammars is NP-complete at least for pregroup product grammars built with at least 3 free pregroups.

The proof uses the product of three copies of the free pregroup on $P_{SAT} = \{t, f\}$ with equality as the partial order on basic types. The set of elements of the pregroup is $T_{SAT} = T_{(P_{SAT},=)} \times T_{(P_{SAT},=)} \times T_{(P_{SAT},=)}$. The first component corresponds to the encoding of the formula that we want to satisfy. The two other components are used to propagate the boolean values of variables.

The formula is transformed into a string and it can be satisfied iff the string is included in the language generated by a fixed pregroup product grammar $\mathcal{G}_{SAT}$ based upon T_{SAT}.

Definition 7 (Formula Transformation $\mathcal{T}_n(F)$).
A boolean formula F that contains (at most) n variables $v_1, \ldots, v_n$, operators $\wedge$ (binary conjunction), $\vee$ (binary disjunction) and $\neg$ (negation) is transformed into a string $\mathcal{T}_n(F) \in \{a, b, c, d, e, \wedge, \vee, \neg\}^*$. $\mathcal{T}_n(F)$ and $[F]_n$ are defined as follows:

- $\mathcal{T}_n(F) = \underbrace{a \cdots a}_{n}[F]_n \underbrace{e \cdots e}_{n}$
- $[v_i]_n = \underbrace{b \cdots b}_{i-1} c \underbrace{b \cdots b}_{n-i} \underbrace{d \cdots d}_{n}$
- $[F_1 \vee F_2]_n = \vee [F_1]_n [F_2]_n$
- $[F_1 \wedge F_2]_n = \wedge [F_1]_n [F_2]_n$
- $[\neg F_1]_n = \neg [F_1]_n$

Example 2. A boolean formula is transformed into a string using the prefix notation for operators. The transformations of $v_1 \wedge v_1$ and $v_1 \vee (v_1 \wedge v_2)$ are:

$$\mathcal{T}_1(v_1 \wedge v_1) = a \wedge \underbrace{cd}_{for\ v_1}\ \underbrace{cd}_{for\ v_1}\ e$$

$$\mathcal{T}_2(v_1 \vee (v_1 \wedge v_2)) = aa \vee \underbrace{cbdd}_{for\ v_1} \wedge \underbrace{cbdd}_{for\ v_1}\ \underbrace{bcdd}_{for\ v_2}\ ee$$

Definition 8 (Pregroup Product Grammar $\mathcal{G}_{SAT}$).
The pregroup product grammar $\mathcal{G}_{SAT} = (\{a, b, c, d, e, \wedge, \vee, \neg\}, I_{SAT}, t)$, based upon the product of three copies of the free pregroup on $(P_{SAT}, =)$ where $P_{SAT} = \{t, f\}$, is defined by the following lexicon:

$$
\begin{aligned}
I_{SAT}(a) &= \{\ (1, t^l, 1)\ ,\ (1, f^l, 1)\ \} \\
I_{SAT}(b) &= \{\ (1, t, t^l)\ ,\ (1, f, f^l)\ \} \\
I_{SAT}(c) &= \{\ (t, t, t^l)\ ,\ (f, f, f^l)\ \} \\
I_{SAT}(d) &= \{\ (1, t^l, t)\ ,\ (1, f^l, f)\ \} \\
I_{SAT}(e) &= \{\ (1, t, 1)\ ,\ (1, f, 1)\ \} \\
I_{SAT}(\wedge) &= \{\ (tt^l t^l, 1, 1)\ ,\ (ff^l t^l, 1, 1)\ ,\ (ft^l f^l, 1, 1)\ ,\ (ff^l f^l, 1, 1)\ \} \\
I_{SAT}(\vee) &= \{\ (tt^l t^l, 1, 1)\ ,\ (tf^l t^l, 1, 1)\ ,\ (tt^l f^l, 1, 1)\ ,\ (ff^l f^l, 1, 1)\ \} \\
I_{SAT}(\neg) &= \{\ (tf^l, 1, 1)\ ,\ (ft^l, 1, 1)\ \}
\end{aligned}
$$

We write $\leq_{T_{(P_{SAT},=)}}$ for the partial order of the free pregroup on $(P_{SAT}, =)$ and $\leq_{SAT}$ for the partial order of the product of the three free pregroups based on $(P_{SAT}, =)$. The types assigned to the strings of $\mathcal{L}(\mathcal{G}_{SAT})$ are $\leq_{SAT} (t, 1, 1)$

Example 3. The formula $v_1 \wedge v_1$ can be satisfied for $v_1 = true$. There exists a type assignment of the symbols of $\mathcal{T}_1(v_1 \wedge v_1) = a \wedge cd\ cd\ e$ by $\mathcal{G}_{SAT}$ that is $\leq_{SAT} (t, 1, 1)$:

$$
\underbrace{(1, t^l, 1)}_{for\ a} \circ \underbrace{(tt^l t^l, 1, 1)}_{for\ \wedge} \circ \underbrace{(t, t, t^l)}_{for\ c} \circ \underbrace{(1, t^l, t)}_{for\ d} \circ \underbrace{(t, t, t^l)}_{for\ c} \circ \underbrace{(1, t^l, t)}_{for\ d} \circ \underbrace{(1, t, 1)}_{for\ e}
$$

$$
\leq_{SAT} (t, 1, 1)
$$

The formula $v_1 \wedge \neg v_2$ can be satisfied for $v_1 = true$ and $v_2 = false$. There exists a type assignment of the symbols of $\mathcal{T}_2(v_1 \wedge \neg v_2) = aa \wedge cbdd\ \neg\ bcdd\ ee$ by $\mathcal{G}_{SAT}$ that is $\leq_{SAT} (t, 1, 1)$:

$$
\underbrace{(1, f^l, 1)}_{for\ a} \circ \underbrace{(1, t^l, 1)}_{for\ a} \circ \underbrace{(tt^l t^l, 1, 1)}_{for\ \wedge} \circ \underbrace{(t, t, t^l)}_{for\ c} \circ \underbrace{(1, f, f^l)}_{for\ b} \circ \underbrace{(1, f^l, f)}_{for\ d} \circ \underbrace{(1, t^l, t)}_{for\ d} \circ
$$

$$
\underbrace{(tf^l, 1, 1)}_{for\ \neg} \circ \underbrace{(1, t, t^l)}_{for\ b} \circ \underbrace{(f, f, f^l)}_{for\ c} \circ \underbrace{(1, f^l, f)}_{for\ d} \circ \underbrace{(1, t^l, t)}_{for\ d} \circ \underbrace{(1, t, 1)}_{for\ e} \circ \underbrace{(1, f, 1)}_{for\ e}
$$

$$
\leq_{SAT} (t, 1, 1)
$$

Theorem 2. *A boolean formula F that contains (at most) n variables $v_1, \ldots, v_n$, operators $\wedge$ (binary conjunction), $\vee$ (binary disjunction) and $\neg$ (negation) can be satisfied iff $\mathcal{T}_n(F) \in \mathcal{L}(\mathcal{G}_{SAT})$*

Example 4. Example 3 shows two formulas that can be satisfied. Their transformations using $\mathcal{T}_n$ are in $\mathcal{L}(\mathcal{G}_{SAT})$. The formula $v_1 \wedge \neg v_1$ cannot be satisfied. A type assignment of $\mathcal{T}_1(v_1 \wedge \neg v_1) = a \wedge cd\ \neg cd\ e$ by $\mathcal{G}_{SAT}$ would produce the following type where for $1 \leq i \leq 11$, $x_i \in \{t, f\}$, $x_2 = x_3 \wedge x_4$ and $x_7 = \neg x_8$

(both equalities come from entries of $\wedge$ and $\neg$ of the lexicon I_{SAT} – we identify here true with t and false with f):

$$\underbrace{(1, x_1^l, 1)}_{a} \circ \underbrace{(x_2 x_3^l x_4^l, 1, 1)}_{\wedge} \circ \underbrace{(x_5, x_5, x_5^l)}_{c} \circ \underbrace{(1, x_6^l, x_6)}_{d} \circ \underbrace{(x_7 x_8^l, 1, 1)}_{\neg} \circ \underbrace{(x_9, x_9, x_9^l)}_{c} \circ \underbrace{(1, x_{10}^l, x_{10})}_{d} \circ \underbrace{(1, x_{11}, 1)}_{e}$$

The type must be $\leq_{SAT} (t, 1, 1)$. Therefore, $x_2 x_3^l x_4^l x_5 x_7 x_8^l x_9 \leq_{T_{(P_{SAT},=)}} t$, $x_1^l x_5 x_6^l x_9 x_{10}^l x_{11} \leq_{T_{(P_{SAT},=)}} 1$ and $x_5^l x_6 x_9^l x_{10} \leq_{T_{(P_{SAT},=)}} 1$. As a consequence, $x_2 = t$, $x_3 = x_7$, $x_4 = x_5$, $x_8 = x_9$, $x_1 = x_5$, $x_6 = x_9$, $x_{10} = x_{11}$, $x_5 = x_6$ and $x_9 = x_{10}$. There is no solution to all these equations: The transformation of the formula $v_1 \wedge \neg v_1$ through $\mathcal{T}_1$ is not in $\mathcal{L}(\mathcal{G}_{SAT})$.

Proof. Firstly, we prove that if a formula F on variables $v_1, \ldots, v_n$ can be satisfied, then $\mathcal{T}_n(F)$ is in $\mathcal{L}(\mathcal{G}_{SAT})$. Let $(x_1, \ldots, x_n) \in \{true, false\}^n$ be an assignment of variables $v_1, \ldots, v_n$ that satisfies F. Using the assignment, the occurrences of the variables and the occurrences of the operators of F can be annotated by boolean values that correspond to the value of the variable or the output value of the operator plus the input value for $\neg$ or both input values for $\vee$ and $\wedge$. Of course, the boolean values associated to an operator follow the truth table of the corresponding boolean operator. Now, we can assign a type in I_{SAT} to each symbol of $\mathcal{T}_n(F)$:

- The assignment of the i-th a in $\mathcal{T}_n(F) = \underbrace{a \cdots a}_{n}[F]\underbrace{e \cdots e}_{n}$ corresponds to the value x_{n+1-i} of the $(n+1-i)$-th boolean variable v_{n+1-i}. If x_{n+1-i} is true, the occurrence is assigned to $(1, t^l, 1)$, otherwise, it is assigned to $(1, f^l, 1)$.
- The assignment of the i-th e in $\mathcal{T}_n(F) = \underbrace{a \cdots a}_{n}[F]\underbrace{e \cdots e}_{n}$ corresponds to the value x_i of the i-th variable. If x_i is true, the occurrence is assigned to $(1, t, 1)$ otherwise to $(1, f, 1)$
- The i-th b or c in $[v_j]_n = \underbrace{b \cdots b}_{j-1} c \underbrace{b \cdots b}_{n-j} \underbrace{d \cdots d}_{n}$ corresponds to the value x_i of the i-th variable. If $i = j$, we have c. Then, if x_i is true, the occurrence is assigned to (t, t, t^l) otherwise to (f, f, f^l). If $i \neq j$, we have b. If x_i is true, the occurrence is assigned to $(1, t, t^l)$ otherwise to $(1, f, f^l)$.
- The i-th d in $[v_j]_n = \underbrace{b \cdots b}_{j-1} c \underbrace{b \cdots b}_{n-j} \underbrace{d \cdots d}_{n}$ corresponds to the value x_{n+1-i} of the $(n+1-i)$-th boolean variable v_{n+1-i}. If x_{n+1-i} is true, the occurrence is assigned to $(1, t^l, t)$, otherwise, it is assigned to $(1, f^l, f)$.
- For $\neg$ in $[\neg F_1]_n = \neg[F_1]_n$, the assignment of variables $v_1, \ldots, v_n$ that satisfies F induces a boolean value to the sub-formula F_1 that is either true or false. The output value of $\neg F_1$ is the opposite value (false for true and true for false). Thus, $\neg$ is assigned to $(tf^l, 1, 1)$ if the input is $false$ and the output is $true$ or to $(ft^l, 1, 1)$ if the input is true and the output is false.
- For $\wedge$ in $[F_1 \wedge F_2]_n = \wedge[F_1]_n[F_2]_n$, the assignment of variables $v_1, \ldots, v_n$ that satisfies F, induces a boolean value to each sub-formula F_1 and F_2. The

output follows the truth table of the logical "and" operator. Following the input values, the assignment of $\wedge$ is given by the following table (the values of the inputs are reverse in the type because they appeared as left adjoints t^l or f^l):

$F_1(x_1,\ldots,x_n)$	$F_2(x_1,\ldots,x_n)$	$\wedge$
$true$	$true$	$(tt^lt^l, 1, 1)$
$true$	$false$	$(ff^lt^l, 1, 1)$
$false$	$true$	$(ft^lf^l, 1, 1)$
$false$	$false$	$(ff^lf^l, 1, 1)$

– $\vee$ is very similar to $\wedge$ except that we follow the truth table of the logical "or" operator:

$F_1(x_1,\ldots,x_n)$	$F_2(x_1,\ldots,x_n)$	$\vee$
$true$	$true$	$(tt^lt^l, 1, 1)$
$true$	$false$	$(tf^lt^l, 1, 1)$
$false$	$true$	$(tt^lf^l, 1, 1)$
$false$	$false$	$(ff^lf^l, 1, 1)$

Now, we create three derivations (one for each component) that prove that the type assignment of $\mathcal{T}_n(F)$ (with the values $x_1,\ldots,x_n$ for the boolean variables $v_1,\ldots,v_n$) is $\leq_{SAT}$ $(t,1,1)$. The first component starts with $s \leq_{T_{(P_{SAT},=)}} t$, the other components with $1 \leq_{T_{(P_{SAT},=)}} 1$. The applications of (A_L) on the first component follow the syntactic tree of F written with the prefix notation for binary operators $\wedge$ and $\vee$. For this component, only the assignments of c, $\neg$, $\wedge$ and $\vee$ are important (the other symbols are assigned to 1 in the first component). The application of rule (A_L) between an occurrence of f^l (on the left) and an occurrence of f (on the right) corresponds to the link between the output of a variable or an operator that is false and one of the inputs of an operator. Similarly the application of rule (A_L) between an occurrence of t^l (on the left) and an occurrence of t (on the right) corresponds to the propagation of the true value. The basic type t that remains at the end is the value of the main operator or variable. It is t because F is true in this case. The two other components are used to synchronize the value given to each occurrence of the variables $v_1,\ldots,v_n$ (each c in $\mathcal{T}_n(F)$). For each occurrence of v_i, this is done on the complete vector of variables $v_1,\ldots,v_n$ but only one of the values (the value that corresponds to v_i) is copied into the first component. If we write $\overline{true} = t$ and $\overline{false} = f$ and if we only look at the second and third components, we have, for $[v_i]_n$, the type $\overline{x_1}\cdots\overline{x_n}\,\overline{x_n}^l\cdots\overline{x_1}^l$ for the second component and the type $\overline{x_1}^l\cdots\overline{x_n}^l\overline{x_n}\cdots\overline{x_1}$ for the third component. The n occurrences of a in $\mathcal{T}_n(F)$ give the type $\overline{x_n}^l\cdots\overline{x_1}^l$ for the second component and 1 for the third. The n occurrences of e give the type $\overline{x_1}\cdots\overline{x_n}$ for the second component and 1 for the third. Obviously, if we write $X = \overline{x_1}\cdots\overline{x_n}$, the global type of the second component is $\underbrace{X}_{for\ a}\underbrace{X^lX}_{for\ v_{i_1}}\cdots\underbrace{X^lX}_{for\ v_{i_m}}\underbrace{X^l}_{for\ e}$ which is $\leq_{T_{(P_{SAT},=)}} 1$. For the

third component, each variable corresponds to $X^l X$, which is $\leq_{T_{(P_{SAT},=)}} 1$. Thus, the type assigned to $\mathcal{T}_n(F)$ using $x_1, \ldots, x_n$ for $v_1, \ldots, v_n$ is $\leq_{SAT} (t, 1, 1)$ and $\mathcal{T}_n(F) \in \mathcal{L}(\mathcal{G}_{SAT})$.

The reverse inclusion proves that if F is a boolean function with n variables $v_1, \ldots, v_n$ and if $\mathcal{T}_n(F) \in \mathcal{L}(\mathcal{G}_{SAT})$ then F can be satisfied. The derivations of the three components that prove that a type assignment of $\mathcal{T}_n(F)$ by I_{SAT} is $\leq_{SAT} (t, 1, 1)$ only use rule (A_L). The other rules (except for the applications of (Id) giving $t \leq_{T_{(P_{SAT},=)}} t$ for the first component and giving $1 \leq_{T_{(P_{SAT},=)}} 1$ for the second and third components and the cut rule) are never used in the system because the right part of the inequalities is either a basic type t for the first component or the unit for the other components and because the partial order on the simple types of the free pregroup is equality. Moreover, $\mathcal{G}_{SAT}$ only uses the four simple types t, t^l, f, and f^l and an assignment of each symbol of $\mathcal{T}_n(F)$ gives always the same formula when basic types t and f are identified. Thus, there exists at most one class of equivalent derivations of any type assignment of $\mathcal{T}_n(F)$ if we look at the set of applications of rule (A_L) in the three components of the type assignment. In a derivation, each application of (A_L) corresponds to an "axiom" between one t^l on the left and one t on the right or between one f^l on the left and one f on the right (as it is shown in Example 1) and all the "axioms" form a projective structure (like the couples of corresponding parentheses in a string of the Dyck language). The class of equivalent derivations (some applications of (A_L) can commute) must correspond to the construction shown above: The first component corresponds to the applications of rule (A_L) that propagate the output of variables and operators to the inputs of the corresponding operator in F. The remaining basic type of the first component (f or t) is the output of F. The second and the third components synchronize the variables in such a way that all the occurrences of the same variable have the same value. Now, if $\mathcal{T}_n(F) \in \mathcal{L}(\mathcal{G}_{SAT})$, the type assignment of the symbols of $\mathcal{T}_n(F)$ is such that the variable v_i has the value corresponding to the second component of the type assignment of the i-th e of $\mathcal{T}_n(F)$: if it is $(1, t, 1)$, v_i is set to true, if it is $(1, f, 1)$ v_i is set to false. For this set of values, the first component of the assignment of $\mathcal{T}_n(F)$ is $\leq_{T_{(P_{SAT},=)}} t$. This means that the value of F is true when the variables $v_1, \ldots, v_n$ are set to the values above. Thus F can be satisfied.

Of course because the membership problem in $\mathcal{L}(\mathcal{G}_{SAT})$ is a NP problem, this problem is NP-complete. As a consequence, the membership problem of $\mathcal{L}(G)$ when G is a pregroup product grammar is also NP-complete. The problem is still open for pregroup product grammar based of two free pregroups but this problem is most probably NP-complete.

5 Conclusion

The article introduces pregroup product grammars, grammars based of the product of free pregroups. It is shown that the class of languages is very expressive.

For instance, $\{x_1^i \cdots x_N^i \mid i \geq 1\}$ for any $N \geq 1$ can be generated. However, the membership problem is NP-complete. Thus even if they are much more expressive, pregroup product grammars are less interesting than pregroup grammars with respect to the complexity of the membership problem.

References

1. Bargelli, D., Lambek, J.: An algebraic approach to french sentence structure. In: de Groote, P., Morill, G., Retoré, C. (eds.) Logical aspects of computational linguistics: 4th International Conference, LACL 2001, Le Croisic, France, June 2001. vol. 2099. Springer-Verlag (2001)
2. Béchet, D.: Parsing pregroup grammars and Lambek calculus using partial composition. Studia logica 87(2/3) (2007)
3. Béchet, D., Dikovsky, A., Foret, A., Garel, E.: Introduction of option and iteration into pregroup grammars. In: Casadio, C., Lambek, J. (eds.) Computational Algebraic Approaches to Natural Language, pp. 85–107. Polimetrica, Monza (Milan), Italy (2008), http://www.polimetrica.com
4. Béchet, D., Dikovsky, A., Foret, A., Garel, E.: Optional and iterated types for pregroup grammars. In: Martn-Vide, C., Otto, F., Fernau, H. (eds.) Language and Automata Theory and Applications, Second International Conference, LATA 2008, Tarragona, Spain, March 13-19, 2008, Revised Papers. Lecture Notes in Computer Science (LNCS), vol. 5196, pp. 88–100. Springer (2008), http://grammars.grlmc.com/LATA2008
5. Buszkowski, W.: Cut elimination for the lambek calculus of adjoints. In: Abrusci, V., Casadio, C. (eds.) New Perspectives in Logic and Formal Linguisitics, Proceedings Vth ROMA Workshop. Bulzoni Editore (2001)
6. Buszkowski, W.: Lambek grammars based on pregroups. In: de Groote, P., Morill, G., Retoré, C. (eds.) 4th Intern. Conf. "Logical Aspects of Computational Linguists". pp. 95–109. No. 2099 in LNAI (2001)
7. Cardinal, K.: An algebraic study of Japanese grammar. Master's thesis, McGill University, Montreal (2002)
8. Casadio, C., Lambek, J.: An algebraic analysis of clitic pronouns in italian. In: de Groote, P., Morill, G., Retoré, C. (eds.) Logical aspects of computational linguistics: 4th International Conference, LACL 2001, Le Croisic, France, June 2001. vol. 2099. Springer-Verlag (2001)
9. Dekhtyar, M., Dikovsky, A.: Categorial dependency grammars. In: Moortgat, M., Prince, V. (eds.) Proc. of Intern. Conf. on Categorial Grammars. pp. 76–91. Montpellier (2004)
10. Joshi, A., Vijay-Shanker, K., Weir, D.: The convergence of mildly context-sensitive grammar formalisms. In: Sells, P., Schieber, S., Wasow, T. (eds.) Fundational issues in natural language processing. MIT Press (1991)
11. Kiślak-Malinowska, A.: Extended pregroup grammars applied to natural languages. Logic and Logical Philosophy 21(3), 229–252 (2012)
12. Kobele, G.M.: Pregroups, products, and generative power. In: Proceedings of the Workshop on Pregroups and Linear Logic 2005, Chieti, Italy, May 2005 (2005)
13. Kobele, G.M.: Agreement bottlenecks in Italian. In: Casadio, C., Lambek, J. (eds.) Computational Algebraic Approaches to Natural Language, pp. 191–212. Polimetrica, Monza (Milan), Italy (2008), http://www.polimetrica.com

14. Kobele, G.M., Kracht, M.: On pregroups, freedom, and (virtual) conceptual necessity. In: Eilam, A., Scheffler, T., Tauberer, J. (eds.) Proceedings of the 29th Pennsylvania Linguistics Colloquium. University of Pennsylvania Working Papers in Linguistics, vol. 12.1, pp. 189–198 (2006)
15. Lambek, J.: Type grammars revisited. In: Lecomte, A., Lamarche, F., Perrier, G. (eds.) Logical aspects of computational linguistics: Second International Conference, LACL '97, Nancy, France, September 22–24, 1997; selected papers. vol. 1582. Springer-Verlag (1999)
16. Lambek, J.: Type grammar meets german word order. Theoretical Linguistics 26, 19–30 (2000)
17. Lambek, J., Preller, A.: An algebraic approach to the german noun phrase. Linguistic Analysis 31, 3–4 (2003)
18. Lambek, J.: The mathematics of sentence structure. American Mathematical Monthly 65, 154–170 (1958)
19. Sadrzadeh, M.: Pregroup analysis of persian sentences (2007), http://eprints.ecs.soton.ac.uk/13970/