Sylvain Degeilh
LIRMM/CNRS
Montpellier
France
degeilh@lirmm.fr

Anne Preller
LIRMM/CNRS
Montpellier
France
preller@lirmm.fr

Pregroups and the French noun phrase

ABSTRACT:
We study mathematical and algorithmic properties of Lambek's pregroups and illustrate them by analysing the French noun phrase. We establish robustness properties of pregroups and present a simple algorithm of complexity n^3 for deciding reduction in an arbitrary free pregroup as well as a linear algorithm covering a large class of language fragments. In the French noun phrase, the agreement of determiners, adjectives and nouns and word order are treated.

Introduction

When choosing a formal system for computational purposes, one looks for *efficiency* (taking decidability for granted), *robustness* and *provable verification*. The mathematical concept of pregroups has these properties. As we hope that our readers come from as varied backgrounds as linguistics, logic or computation, we formulate and reformulate mathematical properties in terms of their significance in linguistic analysis, like conservativity of extensions as robustness in Section 1. Section 2 covers algorithmic properties and is more technical. Both Sections are intended as an argument, why one should prefer pregroups to other systems when analysing a language fragment as we did in Section 3 for the French noun phrase.

Pregroups are an algebraic tool introduced by J. Lambek [Lamb 99] to recognise grammatically well-formed sentences in natural languages. In linguistic applications, one uses *free pregroups* where the mathematical machinery reduces to two simple schematic rewrite rules, the *contractions*. This tool provides an elegant theory which is universal, independent of the language. Up till now, fragments of Arabic, English, French, German, Italian, Japanese, Latin, Polish, Turkish among others have been analysed with the help of pregroups. Besides the simplicity and the universality, other properties can be proven, due to the mathematical character of the tool.

Indeed, the theory of pregroups is decidable. Therefore, as we shall explain below, the problem whether a given string of words is a well-formed construct of a typed language fragment is decidable. The general decision problem for free pregroups can be implemented by an algorithm of complexity n^3, guaranteeing thus an efficient algorithm for the typing checking of language fragments. Our "nearest left parentheses" algorithm in Proposition 3 adapts and simplifies an algorithm given in [Earley] for context free grammars. Earley's algorithm was written for context free grammars and uses the number of rewrite rules of the grammar as a bound. It cannot be applied as such to deciding reduction in pregroups. Indeed, the schema of the two contraction rules in pregroups yields an infinite set of rewrite rules in the corresponding context free grammar. Even if only finitely many rewrite rules are needed when type checking a given language fragment, Earley's algorithm would be quite a bit more cumbersome than ours. In certain language fragments, our algorithm simplifies further to an algorithm which is *linear* in the number of words. We also formulate a sufficient criterion for linearity (Proposition 2), which we hope will influence the way the types are chosen for a language fragment. Finally, typing with pregroups is robust. As extensions of the theory of pregroups are conservative, the typing of a language fragment can be extended to include new constructs, new words without changing the previous analysis. Thus the typing of the French noun phrase given in Section 3 extends earlier work of [Barg-Lamb] on the French sentence structure. It is also part of ongoing work of a more comprehensive fragment. Though our typing covers a liberal notion of determiners and can account for the distinction between the indefinite versus the partitive reading of the articles *du, des* etc., our typing is far from covering all aspects and aims above all at a demonstration of the potential of pregroups.

1. Mathematical properties and their linguistic meaning

To explain how pregroups work in linguistic, we recall briefly the definitions and the immediate consequences.

Definition 1: A pregroup is a partially ordered monoid in which each element a has both a *left adjoint* a^{ℓ} and a *right adjoint* a^{r} satisfying

(Contraction) $a^{\ell} \cdot a \to 1$

$a \cdot a^{r} \to 1$

(Expansion) $1 \to a^{r} \cdot a$

$1 \to a \cdot a^{\ell}$.

The arrow denotes the partial order relation, the dot denotes multiplication and is generally omitted. By definition, the multiplication is associative and the order is *compatible* with multiplication, that is

$a \to b$ and $c \to d$ implies $ac \to bd$.

A linguist will work with the *free* pregroup generated by a partially ordered set of so called *basic types*. For a given language fragment, one chooses a set of symbols, the *basic types*, and defines a partial order on this set. We use bold face symbols $\boldsymbol{a}$, $\boldsymbol{b}$ to vary over basic types.

The basic types and their iterated adjoints form a set $\boldsymbol{\Sigma}$ of *simple types* :

$$\boldsymbol{\Sigma} = \{\ldots \boldsymbol{a}^{\ell\ell}, \boldsymbol{a}^{\ell}, \boldsymbol{a}, \boldsymbol{a}^{r}, \boldsymbol{a}^{rr}, \ldots \quad \ldots., \boldsymbol{b}^{\ell\ell}, \boldsymbol{b}^{\ell}, \boldsymbol{b}, \boldsymbol{b}^{r}, \boldsymbol{b}^{rr}, \ldots\}$$

The strings of simple types will be called *types*. The arrow $a \to b$ is now read as "a *reduces to* b". In fact, the elements of the free pregroup generated by a set of basic types identify with the types constructed from the same set of basic types, see [Lambek99]. The unit 1 denotes the empty string.

The simple types inherit the order from the basic types as follows:

(I) If $\boldsymbol{a} \to \boldsymbol{b}$ then $\boldsymbol{b}^{\ell} \to \boldsymbol{a}^{\ell}, \boldsymbol{b}^{r} \to \boldsymbol{a}^{r}$, $\boldsymbol{a}^{\ell\ell} \to \boldsymbol{b}^{\ell\ell}, \boldsymbol{a}^{rr} \to \boldsymbol{b}^{rr}$, $\boldsymbol{b}^{\ell\ell\ell} \to \boldsymbol{a}^{\ell\ell\ell}, \boldsymbol{b}^{rrr} \to \boldsymbol{a}^{rrr}$, etc.

The order between basic types is "declared", i.e. can be looked up in a table. Hence the ordering of simple types follows from the same table. Indeed, to check whether $a \to b$, it suffices to check whether a and b have the same exponent, i.e. $a = \boldsymbol{a}^s$, $b = \boldsymbol{b}^s$ where $\boldsymbol{a}$ and $\boldsymbol{b}$ are basic and s consists of a finite number n of repetitions of the same symbol, either ℓ or r. If this is the case, then $a \to b$ if either n is even and $\boldsymbol{a} \to \boldsymbol{b}$ or n is odd and $\boldsymbol{b} \to \boldsymbol{a}$. Otherwise, neither a reduces to b nor b to a.

The contractions of simple types can now also be understood as rules :

(II) …, $\boldsymbol{a}^{\ell\ell\ell}\boldsymbol{a}^{\ell\ell} \to 1$, $\boldsymbol{a}^{\ell\ell}\boldsymbol{a}^{\ell} \to 1$, $\boldsymbol{a}^{\ell}\boldsymbol{a} \to 1$, $\boldsymbol{a}\boldsymbol{a}^{r} \to 1$, $\boldsymbol{a}^{r}\boldsymbol{a}^{rr} \to 1$, $\boldsymbol{a}^{rr}\boldsymbol{a}^{rrr} \to 1$,….

In linguistic applications, where one works with a free pregroup, only the contractions and ordering of simple types is needed (for the detail, see below), i.e. the rules (I) and (II) are all that is to be kept in mind when "typing" a language fragment.

As a first step, the language fragment has to be described in common grammatical terms. We select the words which are supposed to be in the mental or electronic dictionary, for example nouns, adjectives etc. and introduce the grammatical notions describing the grammatically well-formed constructs of the fragment. The next step is to define the set of basic types and its ordering. It must reflect the grammatical constructions under consideration. In general, one will aim at keeping the set of basic types as small and with few inequalities as possible. Finally, to every word of the fragment, one associates one or several types, to be written next to the word in dictionary. Meta-rules serve to organise the content of the dictionary. Instead of explicitly writing the type(s) of a word into the dictionary, it (they) may be described by a meta-rule.

The typing must be done in a way that a sequence of words is a well-formed construct (sentence, noun phrase, etc) if and only if the corresponding string of types reduces to the basic type corresponding to the construct. If a word has more than one type, then it is enough that one of the possible choices yields a string reducing to the type in question. This equivalence is the key property. Every typing that respects it is said to be *correct* (it recognises only well-formed constructs) and *complete* (it recognises all well-formed constructs) with respect to the fragment.

A correct and complete typing satisfies the *principle of Substitution* : If a word is replaced by another word with the same type, then a well-formed string of words remains well-formed. Indeed, being a well-formed string of words is equivalent of having a string of types reducible to a given basic type and, by assumption, the string of types is the same before and after substitution.

Every correct and complete typing satisfies properties which permit to extend the typing to larger fragments without changing the properties of the previous typing. We call these the *robustness* properties:

a) Assigning new types to words, without changing the set of basic types :

Suppose a word *W* has type d and we add a type c such that $c \to d$. Then every string of words recognised well-formed using the type assignment d for *W* is recognised well-formed using c.

b) Extensions by new basic types.

This means that one can extend a given set $\boldsymbol{B}$ of basic types, by declaring new types and adding inequalities involving the new types, obtaining thus a larger set of basic types $\boldsymbol{B'}$. Then the free pregroup $\boldsymbol{P'}$ generated by $\boldsymbol{B'}$ includes the free pregroup $\boldsymbol{P}$ generated by $\boldsymbol{B}$. Whenever both a and b belong to the smaller pregroup $\boldsymbol{P}$ and $a \to b$ can be derived in $\boldsymbol{P}$, then this also holds in the larger $\boldsymbol{P'}$. *Conservativity* is a sort of converse of this fact, it says : if $a \to b$ can be derived in the larger pregroup $\boldsymbol{P'}$ and both a and b belong to the smaller pregroup, then the whole reduction can be done in the smaller pregroup. This is not trivial, because we might have come to the conclusion $a \to b$ by showing $a \to c$ and $c \to b$ using one of the new types c. Conservativity is a consequence of the so-called Switching Lemma in [Lambek99] (see below) provided the order of the old basic types remains unchanged.

The linguistic significance of this is that typing by pregroups is robust. Once it has been shown correct and complete for a fragment it will remain so in the extensions. It also simplifies the task of verifying that a typing is correct and complete. One can proceed step by step, extending the fragment by adding basic type after basic type, and then verify only that the typing involving the new type(s) is also correct and complete for the new constructs.

Here are the promised mathematical details on which our assertions above are based. Except Proposition 1, they are either straight forward or to be found in [Lambek99].

1. $a1 = a = 1a$ (1 is the unit of the monoid)
2. $(ab)c = a(bc)$ (multiplication is associative)
3. $a \to b$ and $c \to d$ implies $ac \to bd$ (order is compatible with multiplication).
4. $ab \to 1 \to ba$ implies $a = b^{\ell}$ and $b = a^{r}$ (adjoints are unique)
5. $(ab)^{\ell} = b^{\ell}a^{\ell}$, $(ab)^{r} = b^{r}a^{r}$ (adjunction is quasi-distributive)
6. $a \to b$ implies $b^{\ell} \to a^{\ell}$ and $b^{r} \to a^{r}$ (adjunction reverses the order).
7. $a^{\ell r} = a = a^{r\ell}$ (no mixed adjoints)

Properties 1. – 3. are in the definition of a monoid, the properties 4. - 7. can be easily derived from Definition 1. For example, to derive that $b = b^{\ell r}$, use $b^{\ell}b \to 1 \to bb^{\ell}$ and 4. with $a = b^{\ell}$. Similarly, 5. follows from 4. Indeed, $(ab)(b^{r}a^{r}) = a(bb^{r})a^{r} \to a1a^{r} = aa^{r} \to 1 \to b^{r}b = b^{r}1b \to b^{r}(a^{r}a)b = (b^{r}a^{r})(ab)$. Hence, by 4., $(ab)^{r} = b^{r}a^{r}$.

In free pregroups, important additional properties hold. First of all, they are non-commutative and the iterated adjoints of basic types are all different. The most important result is expressed in the Switching Lemma [Lambek99, Proposition 2], which we shall include here for completeness sake. Recall that the elements of the partially ordered set generating the free pregroup are called basic types, their iterated left or iterated right adjoints are called simple types. Strings of simple types are called types, the empty string being denoted 1. Lambek [loc.cit.] shows that this monoid of types is in fact the free pregroup generated by the basic types.

Hence, concatenation plays the role of multiplication and the empty string that of the unit. It is convenient to use a uniform notation for the simple types, be they iterated left adjoints or iterated right adjoints: Write

$$\ldots\boldsymbol{a}^{(-2)},\ \boldsymbol{a}^{(-1)},\ \boldsymbol{a}^{(0)},\ \boldsymbol{a}^{(1)},\ \boldsymbol{a}^{(2)},\ldots$$

for

$$\ldots\boldsymbol{a}^{\ell\ell},\ \boldsymbol{a}^{\ell},\ \boldsymbol{a}\ ,\ \boldsymbol{a}^{r},\ \boldsymbol{a}^{rr},\ldots .$$

Then, by definition, every type has the form

$$\boldsymbol{a}_1^{(n_1)}\ldots\boldsymbol{a}_k^{(n_k)}$$

where $\boldsymbol{a}_1,\ldots,\boldsymbol{a}_k$ are basic types and $n_1,\ldots,n_k$ are integers. The types form already a pregroup, if one defines adjoints by

$$(\boldsymbol{a}_1^{(n_1)}\ldots\boldsymbol{a}_k^{(n_k)})^{\ell} := \boldsymbol{a}_k^{(n_k-1)}\ldots\boldsymbol{a}_1^{(n_1-1)}$$

$$(\boldsymbol{a}_1^{(n_1)}\ldots\boldsymbol{a}_k^{(n_k)})^{r} := \boldsymbol{a}_k^{(n_k+1)}\ldots\boldsymbol{a}_1^{(n_1+1)}$$

and the order on types as the reflexive and transitive closure of the following three relations where c,d are arbitrary types and $\boldsymbol{a},\boldsymbol{b}$ are basic:

(Induced step)

$c\boldsymbol{a}^{(n)}d \to c\boldsymbol{b}^{(n)}d$, if either n is even and $\boldsymbol{a}\to\boldsymbol{b}$ or n is odd and $\boldsymbol{b}\to\boldsymbol{a}$.

(Generalised contraction)

$c\boldsymbol{a}^{(n)}\boldsymbol{b}^{(n+1)}d \to cd$, if either n is even and $\boldsymbol{a}\to\boldsymbol{b}$ or n is odd and $\boldsymbol{b}\to\boldsymbol{a}$.

(Generalised expansion)

$cd \to c\boldsymbol{a}^{(n+1)}\boldsymbol{b}^{(n)}d$, if either n is even and $\boldsymbol{a}\to\boldsymbol{b}$ or n is odd and $\boldsymbol{b}\to\boldsymbol{a}$.

Notice that for every basic type $\boldsymbol{b}$, one has the contractions

$\boldsymbol{b}^{(n)}\ \boldsymbol{b}^{(n+1)} \to 1$, whatever the integer n .

<u>Switching Lemma (Lambek)</u>: Let $a_1,\ldots,a_n$ and $b_1,\ldots,b_m$ be simple types. Then $a_1\ldots a_n \to\ b_1\ldots b_m$ if and only if there are a substring $a_{i_1}\ldots a_{i_k}$ of $a_1\ldots a_n$ and a substring $b_{i_1}\ldots b_{i_k}$ of $b_1\ldots b_m$ such that

$$a_1\ldots a_n \to\ a_{i_1}\ldots a_{i_k} \to b_{i_1}\ldots b_{i_k} \to b_1\ldots b_m\ ,\ a_{i_p} \to b_{i_p}, 1\le p\le k\ ,$$

where $a_{i_1}\ldots a_{i_k}$ is obtained from $a_1\ldots a_n$ by generalised contractions only, $b_1\ldots b_m$ is obtained from $b_{i_1}\ldots b_{i_k}$ by generalised expansions only and finally $b_{i_1}\ldots b_{i_k}$ is obtained from $a_{i_1}\ldots a_{i_k}$ by induced steps only.

Call a pair of simple types (c,d) *contractible*, if $c=\boldsymbol{a}^{(n)},\ d=\boldsymbol{b}^{(n+1)}$ and either $\boldsymbol{a}\ \to\ \boldsymbol{b}$ and n even or $\boldsymbol{b}\to\boldsymbol{a}$ and n odd.

<u>Corollary (Lambek) </u>: For simple types $a_1,...,a_n$ and b, $a_1...a_n \to b$ holds if and only if there is a simple type $b' \to b$ such that one can obtain b' from $a_1...a_n$ by repeatedly omitting pairs of contractible adjacent types.

I) The first consequence is the decidability of the *type checking problem*, namely to decide whether $a \to b$ where a is an arbitrary type and b is simple.

Indeed, given a string $a_1...a_n$ choose an index i such that (a_i, a_{i+1}) is contractible and omit $a_i a_{i+1}$, start again with $a_1...a_{i-1}a_{i+2}..a_n$ until an irreducible string is reached, i. e. a string without adjacent contractible types. As different choices may lead to different irreducible substrings, the answer is yes, if at least one of the choices leads to a simple type b' for which $b' \to b$.

Hence, if a correct and complete typing has been provided for a language fragment, then the fragment itself is decidable. Given a string of words in the dictionary of the fragment enumerate all possible strings of types corresponding to the words. As every word has only finitely many types, there are only finitely many such strings. Then test for each string, if it reduces to an appropriate basic type.

Another consequence is the characterisation of conservative extensions :

<u>II) Proposition 1: (Conservativity of extensions)</u> : Let $\boldsymbol{B}$ be an ordered subset of $\boldsymbol{B}'$, i.e. $\boldsymbol{B}$ is a subset of $\boldsymbol{B}'$ and $\boldsymbol{a} \to \boldsymbol{b}$ in $\boldsymbol{B}$ if and only if $\boldsymbol{a,b} \in \boldsymbol{B}$ and $\boldsymbol{a} \to \boldsymbol{b}$ in $\boldsymbol{B}'$. Then the free pregroup $\boldsymbol{P}'$ generated by $\boldsymbol{B}'$ is conservative over the free pregroup $\boldsymbol{P}$ generated by $\boldsymbol{B}$. That is to say, for all elements e,f of $\boldsymbol{P}$ such that $e \to f$ holds in $\boldsymbol{P}'$, $e \to f$ holds already in $\boldsymbol{P}$.

Proof: Let e,f be elements of $\boldsymbol{P}$ such that $e \to f$ holds in $\boldsymbol{P}'$. First consider the special case where $e \to f$ is a generalised contraction. Then there are basic types $\boldsymbol{a,b} \in \boldsymbol{B}$ and an integer n such that $e = c\boldsymbol{a}^{(n)}\boldsymbol{b}^{(n+1)}d$, $f = cd$ and either $\boldsymbol{a} \to \boldsymbol{b}$ in $\boldsymbol{B}'$ and n is even or $\boldsymbol{b} \to \boldsymbol{a}$ in $\boldsymbol{B}'$ and n is odd. By hypothesis, this implies $\boldsymbol{a} \to \boldsymbol{b}$ in $\boldsymbol{B}$ and n is even or $\boldsymbol{b} \to \boldsymbol{a}$ in $\boldsymbol{B}$ and n is odd. Therefore $e \to f$ is a generalised contraction in $\boldsymbol{P}$. By a similar argument, one shows that a generalised expansion (induced step) in $\boldsymbol{P}'$ is also a generalised induced step in $\boldsymbol{P}$.

In the general case, there are simple types $a_1,...,a_n$ and $b_1,...,b_m$ of $\boldsymbol{P}$ such that $e = a_1...a_n$, $f = b_1...b_m$ and $a_1...a_n \to b_1...b_m$ in $\boldsymbol{P}'$. Apply the Switching Lemma. There is a substring $a_{i_1},...,a_{i_k}$ of $a_1,...,a_n$ and a substring $b_{i_1},...,b_{i_k}$ of $b_1,...,b_m$ such that

$$a_1...a_n \to a_{i_1}...a_{i_k} \to b_{i_1}...b_{i_k} \to b_1...b_m\,,\ a_{i_p} \to b_{i_p} \text{ in } \boldsymbol{P}',\ 1 \le p \le k\,,$$

$a_{i_1}...a_{i_k}$ is obtained from $a_1...a_n$ by repeated generalised contractions and $b_1...b_m$ is obtained from $b_{i_1}...b_{i_k}$ by repeated generalised expansions. Now for every generalised contraction $c \to d$, from $c \in \boldsymbol{P}$ follows $d \in \boldsymbol{P}$. Indeed, d is obtained from c by omitting two simple types. Similarly, for every generalised expansion $c \to d$, from $d \in \boldsymbol{P}$ follows $c \in \boldsymbol{P}$. Hence $a_{i_1}...a_{i_k}$ and $b_{i_1}...b_{i_k}$ are in $\boldsymbol{P}$ and all the intermediary generalised contractions, expansions and induced steps take place in $\boldsymbol{P}$. By the first part of the proof, this implies that $a_1 \ldots a_n \to a_{i_1} \ldots a_{i_k} \to b_{i_1} \ldots b_{i_k} \to b_1 \ldots b_m$, and $a_{i_p} \to b_{i_p}$, $1 \le p \le k$, already hold in $\boldsymbol{P}$.

The Proposition 1 simplifies the task of verifying that the typing is correct and complete. One can proceed step by step, extend the fragment by adding new basic types and assigning new types to words. Then to show that the typing is correct and complete with respect to the larger fragment, it suffices to consider the sequences of words with new types. The only prerequisite is not to change the order between the old basic types. For example, suppose $\boldsymbol{B'} = \boldsymbol{B[c]}$ where $\boldsymbol{c}$ is a new basic type. We may declare $\boldsymbol{a} \to \boldsymbol{c}$ and/or $\boldsymbol{c} \to \boldsymbol{b}$ for some "old" basic types $\boldsymbol{a,b} \in \boldsymbol{B}$. However, we must take care not to declare both $\boldsymbol{a} \to \boldsymbol{c}$ and $\boldsymbol{c} \to \boldsymbol{b}$, unless $\boldsymbol{a} \to \boldsymbol{b}$ already holds in $\boldsymbol{B}$. Notice that this step by step approach is normally taken for granted. The conservativity property says that this approach is safe. This is not as trivial as one might think. Obviously, a sequence of words which is a grammatically well-formed construct of the smaller fragment remains so in the larger fragment. Just use the typing and the reduction in the smaller pregroup $\boldsymbol{P}$, which remains a reduction in the bigger one $\boldsymbol{P'}$. It is, however, not so obvious that a sequence of words which is not well-formed in the smaller fragment does not become well-formed in the bigger fragment, even if no new types are involved. Indeed, suppose that a sequence of words gets assigned a type c in the smaller fragment, but is not well-formed. As the typing in $\boldsymbol{P}$ is correct and complete with respect to the smaller fragment, we have $c \not\to \boldsymbol{a}$ where $\boldsymbol{a}$ is the basic type corresponding to a the grammatical notion under investigation. A priori, one cannot exclude that $c \to d$ and $d \to \boldsymbol{a}$, where d is a new type in $\boldsymbol{P'}$. But then, it would follow that $c \to \boldsymbol{a}$ in $\boldsymbol{P'}$. From this we would have to conclude that the sequence of words is now well-formed, because its type reduces to $\boldsymbol{a}$. Conservativity guarantees that this cannot happen, as from $c \to \boldsymbol{a}$ in $\boldsymbol{P'}$ it would follow that $c \to \boldsymbol{a}$ in $\boldsymbol{P}$, contradicting $c \not\to \boldsymbol{a}$.

This property is used in fact continuously and most of the time without saying so. We will do so in Section 3

2. Algorithmic Properties

The type checking problem for free pregroups is to decide by a general method whether $a_1 \ldots a_n \to b$ for arbitrary simple types $a_1, \ldots, a_n$ and any irreducible type b, i.e. a string of simple types containing no contractible pair of adjacent simple types. The Corollary to the Switching Lemma gives such a method:

For deciding whether $a_1 \ldots a_n \to b$ holds, it suffices to omit a contractible pair of adjacent types $a_i a_{i+1}$ and repeat this procedure in all possible ways until the resulting string is irreducible. If at least one of the resulting strings b' satisfies $b' \to b$, the answer is yes, otherwise it is no.

Any implementation of this decision procedure is called a *type checking* algorithm. The decision procedure for free pregroups provides a solution to the problem whether a sequence of words belongs to a language fragment. For each word in the sequence choose one of its types in the dictionary. The sequence of words belongs to the fragment if and only if one of the strings of assigned types reduces to a specific basic type, say the type of a noun phrase or the type of a sentence. An algorithm which provides associated strings of types for a string of words is called a *type assignment* algorithm. To keep this paper in reasonable limits, we restrict ourselves to the efficiency of type checking algorithms.

The result of the reduction of a string of types to an irreducible one depends in general on the choice of the contracted pairs. For example, $\boldsymbol{a}^{\ell\ell}\boldsymbol{a}^{\ell}\boldsymbol{a}\boldsymbol{a}^{r}$ reduces with two contractions to the empty string, namely by contracting the left most and the right most pair. On the other hand, by contracting the central pair $\boldsymbol{a}^{\ell}\boldsymbol{a}$ one obtains the irreducible string $\boldsymbol{a}^{\ell\ell}\,\boldsymbol{a}^{r}$. The type checking problem for free pregroups can be replaced by the more specific one which checks if a string reduces to the empty string 1, see Lemma 1 below. The set of all strings $a_1 ... a_n$ such that $a_1 ... a_n \to 1$ can be characterised by a context free grammar. However, this context free grammar has an infinite set of terminal symbols (the simple types and 1) and its set of rewrite rules is infinite. Indeed, it includes all the rules of the form $E \Rightarrow E\,\boldsymbol{a}^{(n)}\,E\,\boldsymbol{a}^{(n+1)}\,E$ where $\boldsymbol{a}$ is a basic type, n an integer and E a new, non-terminal symbol standing for a string which reduces to the empty one. Hence the efficient algorithm of complexity $K\ n^3$ in [Earley] does not apply, because K is a common bound for the set of terminal symbols and the set of rewrite rules. If we are only interested in linguistic applications, we may restrict the set of strings by requiring that the intervening simple types belong to a given finite set. However, our algorithm in Propostion 3 not only is simpler than Earley's but also implements the decision procedure for type checking in an arbitrary free pregroup. Like Earley's, it reads the string from left to right and has complexity n^3, but with a much lower constant. For certain sets of strings, our algorithm can be simplified even further to one that is linear in the length n of the string, see Proposition 4. This linear algorithm is useful in linguistic applications. The strings of types associated to a word of the language fragment very often satisfy the criterion which permits the use of the linear algorithm.

We begin with a criterion for subsets of free pregroups permitting linear type checking. A triple of simple types (a,b,c) is said to be *critical,* if both (a,b) and (b,c) are contractible. This is equivalent to saying that there are basic types $\boldsymbol{a},\boldsymbol{b},\boldsymbol{c}$ and an integer n such that $a = \boldsymbol{a}^{(n-1)}$, $b = \boldsymbol{b}^{(n)}$, $c = \boldsymbol{c}^{(n+1)}$ and $\boldsymbol{b}^{(n)} \to \boldsymbol{a}^{(n)}$, $\boldsymbol{b}^{(n)} \to \boldsymbol{c}^{(n)}$. The latter is the case, exactly when either n is even and $\boldsymbol{b} \to \boldsymbol{a}, \boldsymbol{b} \to \boldsymbol{c}$ or n is odd and $\boldsymbol{a} \to \boldsymbol{b}, \boldsymbol{c} \to \boldsymbol{b}$. A string of simple types is *linear*, if it has no substring of length 3 which is a critical triple. The typing of a language fragment is said to be *linear*, if all strings of types corresponding to strings of words in the dictionary are linear. For example, $\boldsymbol{a}^{r}\boldsymbol{a}\boldsymbol{a}^{\ell}$ is linear, but $\boldsymbol{a}^{\ell}\boldsymbol{a}\boldsymbol{a}^{r}$ is not linear. Or if $\boldsymbol{a} \not\to \boldsymbol{b}$, then $\boldsymbol{c}\boldsymbol{b}\boldsymbol{a}^{\ell}\boldsymbol{a}\boldsymbol{a}\boldsymbol{b}^{r}$ is linear, but if $\boldsymbol{a} \to \boldsymbol{b}$, then $\boldsymbol{c}\boldsymbol{b}\boldsymbol{a}^{\ell}\boldsymbol{a}\boldsymbol{a}\boldsymbol{b}^{r}$ is not linear.

<u>Proposition 2:</u> Every linear type has a unique irreducible form.

Proof: Suppose $a_1 ... a_n$ is linear. Use induction on n, the length of the string. If $n = 1$, the property is obvious. For the induction step, notice that every substring of $a_1 ... a_n$ is again linear. Moreover, whenever $a_i\, a_{i+1}$ and $a_j a_{j+1}$ are both contractible, the indices $i, i+1, j, j+1$ are all different. Suppose $a_1 ... a_n$ has k pairs of contractible adjacent types. Omitting them in $a_1 ... a_n$ corresponds to k simultaneous contractions. They can be done in any order without changing the result. If $k = 0$, then $a_1 ... a_n$ is irreducible. Otherwise, the unique substring resulting from the k contractions has length less than n and we may conclude by induction hypothesis.

Lemma 1: Suppose that $a_1,...,a_n$ and b are simple types. Then $a_1...a_n \to b$ if and only if $b^\ell\ a_1...a_n \to 1$.

Proof: Assume that $b^\ell\ a_1...a_n \to 1$ and use induction on the length n. If $n=1$, $b^\ell a_1 \to 1$ must be generalised contraction. From the definition of generalised contractions, it follows at once that $a_1 \to b$. Assume $n>1$. If $b^\ell a_1$ are contractible, we conclude as above. Otherwise, the first generalised contraction transforming $b^\ell\ a_1...a_n$ to 1 must be in $a_1...a_n$. Hence the result has the form $b^\ell\ a_{i_1}...a_{i_{n-2}}$ with $b^\ell\ a_{i_1}...a_{i_{n-2}} \to 1$. By induction hypothesis, $a_{i_1}...a_{i_{n-2}} \to b$. As $a_1...a_n \to a_{i_1}...a_{i_{n-2}}$, the property follows. As $a_1...a_n \to b$ implies $b^\ell a_1...a_n \to b^\ell b \to 1$, we are done.

It is easy to device a linear-time algorithm which reads a string from left to right and produces an irreducible form. For linear strings, it will provide the only irreducible string obtainable by contractions. In fact, this left to right algorithm is a closely linked to the more general one below which solves the type checking problem without restriction on the strings.

Proposition 3 : There is an algorithm which decides in at most n^3 steps whether $a_1...a_n \to 1$.

Proof: We define a function Nlp which maps a string S of length n and an integer $i \in \{1,...,n+1\}$ to a subset of $\{1,...,i\}$ and show that

1) $S \to 1$ if and only if $0 \in Nlp(S,n+1)$
2) $Nlp(S,n+1)$ can be calculated in at most n^3 steps

We omit the parameter S to simplify notation. The intuitive idea of Nlp is the following : Process the string $S = a_1...a_n$ from left to right, looking for all possible contractions, reading the symbol a_i at stage $i+1$. This symbol could be a left "parenthesis" to a later a_p, i.e. $a_i \le a_p^{\ \ell}$, or a "right parenthesis" to some earlier a_j, i.e. $a_i \le a_j^{\ r}$. As it cannot be both in the same reduction, one has to keep track of more than one reduction, or at least of the indispensable information. It turns out that the left parentheses ready for contraction at stage i is all that is needed to continue the processing to stage $i+1$. Therefore at stage $i+1$, a_i is declared an "open" left parenthesis (to be contracted with some right parenthesis which might come up later), i.e. the index i is stored in $Nlp(i+1)$. Notice that for this choice, a_i becomes the nearest open left parenthesis for a_{i+1}. However, a_i might also be a right parenthesis to the nearest left parenthesis in a reduction made up to stage i. So, for each $j \in Nlp(i)$, we check if $a_i \le a_j^{\ r}$. If this is the case, a_j becomes "closed" and the open left parenthesis at stage j become ready again for contraction at stage $i+1$. Hence, Nlp has the following definition.

Definition 2:

$Nlp(1) = \{0\}$

$$Nlp(i+1)=\{i\}\cup\bigcup_{j\in Nlp(i),a_i\le a_j^{\ r}} Nlp(j)\ ,\ 1\le i\le n$$

The rest of the proof follows from the next 5 lemmas.

Lemma 2 : $Nlp(i+1)\subseteq\{0,...,i\}$.

This follows immediately from the definition.

Lemma 3 : If $k\in Nlp(i+1)$, then $a_{k+1}...a_i\to 1$, i.e. a_k is ready for contraction with a_{i+1}.

Indeed, use induction on $i-k$. If $i=k$, then $a_{k+1}...a_i$ is the empty string and $1\to 1$. Assume $i-k>0$ and $k\in Nlp(i)$. Then there is a $j\in Nlp(i)$ such that $k\in Nlp(j)$ and $a_i\le a_j^{\ r}$. By Lemma 2 and assumption, $k<j<i$, therefore $j-k<i-k$ and $i-j<i-k$. Thus by induction hypothesis, $a_{k+1}...a_{j-1}\to 1$ and $a_{j+1}...a_{i-1}\to 1$. Finally, $a_{k+1}...a_{j-1}a_ja_{j+1}...a_{i-1}a_i\to a_ja_i\to a_ja_j^{\ r}\to 1$.

Lemma 4 : If $a_{k+1}...a_i\to 1$, then $k\in Nlp(i+1)$.

Again proceed by induction on $i-k$. If $i=k$, then $k\in Nlp(i+1)$ by definition. Let $i-k>0$ and assume that $a_{k+1}...a_i\to 1$. Then there is an index j such that $k+1\le j\le i-1$ such that $a_{j+1}...a_{i-1}\to 1$, $a_i\le a_j^{\ r}$ and $a_{k+1}...a_{j-1}\to 1$. By induction hypothesis, this implies that $j\in Nlp(i)$ and $k\in Nlp(j)$. So, $k\in Nlp(i+1)$ by definition.

Lemma 5 : $0\in Nlp(n+1)$ if and only if $a_1...a_n\to 1$.

Apply Lemma 3 and Lemma 4 to $k=0$, $i=n$.

Lemma 6 : $Nlp(i+1)$ can be calculate from $Nlp(i)$ in at most i^2+1 steps. The complexity of $Nlp(n+1)$ is n^3.

Indeed, to calculate $Nlp(i+1)$ we first copy i into $Nlp(i+1)$ and then must compare a_i with $a_j^{\ r}$ for every $j\in Nlp(i)$. By Lemma 2, we make at most i comparisons. Each time the comparison succeeds, the elements of $Nlp(j)$ are copied into $Nlp(i+1)$. But $Nlp(j)$ has at most j elements and j is bounded by i. On the whole, we have executed at most i^2+1 steps, counting a comparison and recopying an element as one-step operation. Finally, to calculate $Nlp(n+1)$, we must calculate $Nlp(1),...,Nlp(n)$ one after the other, so we do at most $n(n^2+1)$ steps. Notice that the constant of n^3 is 1 in this estimate.

Corollary 1: If there is a bound K such that $Nlp(i)<K$ for all i, then the complexity of $Nlp(n+1)$ is linear n.

Indeed, the number of steps from $Nlp(i)$ to $Nlp(i+1)$ is bounded by K^2+1, hence $Nlp(n+1)$ can be calculated in at most $(K^2+1)n$ steps.

Notice that Nlp makes enough calculations to scan all possible reductions of the string. In some situations, it might be enough to find just one reduction yielding an irreducible string. The hope is that if there are fewer reductions to keep track of, it is more likely to find a bound for the set of "nearest left open parentheses". A slight modification of Nlp produces such a reduction, which can be calculated in at most $4n$ steps. The idea is to open a new left parenthesis, only if necessary. Call the corresponding function Llp, the "lazy left parenthesis" function. It is defined on $\{1,...,n\}$, the string S of length n being given, and takes subsets of $\{1,...,n\}$ as values.

<u>Definition 3</u>:

$$Llp(1)=\{0\}$$

$$Llp(i+1)=\begin{cases}\bigcup_{j\in Link(i)} Llp(j), \text{ if } Link(i)\neq\varnothing \\ \{i\} \text{ else}\end{cases}$$

where $Link(i)=\{j\in Llp(i): 1\le j,\ a_i\le a_j{}^r\}$, $1\le i\le n$.

One shows easily that $Llp(i+1)$ is included in $Nlp(i+1)$ and has exactly one element. Moreover, $j\in Link(i)$ means that a_j is contracted with a_i in the reduction defined by Llp. More precisely, say that i and j are *linked*, if $j\in Link(i)$ or $i\in Link(j)$. Then :

<u>Lemma 7</u> : 1) If $k\in Llp(i+1)$, then $a_{k+1}...a_i\to 1$ and whenever $k+1\le m\le i$ there exists a p such that m and p are linked and $k+1\le p\le i$.

2) If $j\in Link(i)$, then $a_j...a_i\to 1$.

3) $m\in Link(p)$ implies for all $i>p$ and all $j\in Llp(i)$ that $j>p$ or $j<m$.

4) Every index p is linked to at most one index m.

Proof: 1) Proceed as in Lemma 3. Assume $k\in Llp(i+1)$ and $k+1\le m\le i$. If $k=i$, there is nothing to show. If $k<i$, then there is $j\in Link(i)$ such that $k\in Llp(j)$. As $Link(i)\subseteq Llp(i)$, the induction hypothesis applies to $i-j$ and to $j-k$. Remark that if $m=j$ or $m=i$, then m is indeed linked to some p with $k+1\le p\le i$. 2) is an immediate consequence of 1). 3) Suppose $m\in Link(p)$, $i>p$ and $j\in Llp(i)$. Use induction on $i-p$. In the case of $i=p+1$, we have $j\in Llp(m)$, as $Link(p)\neq\varnothing$. Hence $j<m$. If $i>p+1$, then either $j=i-1>p$ or $j\in Llp(i-1)$ and therefore by induction hypothesis $j>p$ or $j<m$. To see 4), remark first that an index p cannot be linked to two different smaller indices, as $Link(p)$ has at most one element. By 3)

an index m cannot be linked to two larger ones, say p and i with $p<i$. And it also cannot be linked to a smaller and a larger one, as $m \in Link(p)$, $p \in Link(i)$ would also contradict 3).

Finally, the next and last Lemma confirms that the unlinked elements of the string in increasing order form an irreducible substring.

Lemma 8 : Let *Unlinked* $= \{i_1,\ldots,i_q\}$ be the set of unlinked indices in increasing order. Then the following holds

I) every index less than i_1 (respectively between i_ℓ and $i_{\ell+1}$, respectively larger than i_q) is linked to some index below i_1 (respectively between i_ℓ and $i_{\ell+1}$, respectively larger than i_q). II) a_{i_ℓ} and $a_{i_{\ell+1}}$ are not contractible.

Proof: The assertion I) follows from Lemma 7, 1) and 2). To show II), assume $i = i_{\ell+1}$ and let $j \in Llp(i)$. By choice of i, $a_i \not\leq a_j^{\,r}$, i.e. a_j and a_i are not contractible. Hence, it suffices to show that $j = i_\ell$. In view of Lemma 7, 1), it suffices to show that j is not linked to any index. If j was linked to a smaller index, this would contradict Lemma 7, 3). If j was linked to an index greater than i, this would imply that i is linked by Lemma 7, 1). Finally, j cannot be linked to a larger index which would be less than i, because of Lemma 7, 1) and 4).

Proposition 4 : The type checking problem of a linear fragment can be decided by a linear algorithm.

Proof: By Lemma 1, it suffices to find an algorithm which is linear in the length of the string and produces an irreducible form of the string. For this it suffices to add a new step at stage $i+1$: erase j and i from $\{1,\ldots,n\}$, whenever the test $j \in Link(i)$ succeeds. Together with the two steps to calculate $Llp(i+1)$, at most four operations are performed on the whole at stage $i+1$.

3. French Noun phrases

By Proposition 1 on conservativity of extensions, the typing proposed below may be viewed as an extension of the typing in [Barg-Lamb]. To make things work, the new basic types have to be related to the old ones. For example, we introduce the basic type $\boldsymbol{n}_{gn}$ to denote a complete noun phrase, depending on its *gender* g and *number n*. We postulate $\boldsymbol{n}_{gn} \to \boldsymbol{n}$, where $\boldsymbol{n}$ is a type used in [loc.cit.] in the situations where the gender and number are of no importance.

The noun phrases analysed here are either names or a determiner followed by a noun. After the noun or between it and the determiner may be adjectives. A noun alone is in general an incomplete noun phrase, though French knows some noun phrases formed from a noun without a determiner. As these are rather archaic and exceptional, we will not consider them in what follows. The so-called prenominal adjectives precede the noun, the postnominal adjectives follow it. A noun with correctly declined and correctly placed adjectives forms an incomplete noun phrase. A determiner transforms an incomplete noun phrase into a complete noun phrase, which may be subject or object in a sentence. The notion of determiner follows [Le bon usage], it includes the indefinite article *un, une*, the definitive article, *le, la, l', les* and its contracted forms with *de* namely *du, des*, the possessive and demonstrative pronouns such as well as the preposition *de* followed by an adverb of degree like *beaucoup, peu*

3.1 Nouns and Adjectives

Nouns are count-nouns like *chat, pomme* and mass-nouns like *eau, pain, vent* etc, but also *courage, beauté* and so on. Noun phrases inherit this distinction. Nouns as well as adjectives vary in gender and number. Therefore we distinguish the types of noun phrases by indices g and n, where $g = 1$ stands for masculine $g = 2$ for feminine, whereas $n = 1$ means singular and $n = 2$ means plural. Thus we use the symbol $\boldsymbol{c}_{gn}$ for a count-noun phrase and similarly $\boldsymbol{m}_{gn}$ for a mass-noun phrase. The gender of a noun is given in the dictionary as well as its plural form. Number and gender of noun and adjectives must agree in an noun phrase and constitute the number and gender of the noun phrase. Many mass-nouns have no plural form, for example *riz* . If it has one, we treat the plural form as count-noun.

The formation of the noun phrase also has to take into account the dislike of French speakers of the hiatus where two consecutive vowels would clash. For example, *l'arbre, cet arbre, bel arbre, mon eau*, but **le arbre, ce arbre, *beau arbre, *ma eau.*[1] Every type $\boldsymbol{x}$ will have a copy $\boldsymbol{x'}$ used for words or sequences of words of which the first letter is a vowel.[2] Thus we have basic types $\boldsymbol{x}_{gn}$, where $\boldsymbol{x}$ stands for $\boldsymbol{c}, \boldsymbol{c'}, \boldsymbol{m}$ or $\boldsymbol{m'}$.

Adjectives are not only divided into *prenominal* adjectives and *postnominal* adjectives[3], but they also must respect a certain order, if more than one precedes or follows the noun. For example *bon vin, vin blanc, beau petit chat, autre beau petit chat*, *vin blanc pétillant, bon petit vin blanc pétillant* etc. but **vin bon, *blanc vin, *petit beau chat, *vin pétillant blanc* etc. If two adjectives should occupy the same position, they must be linked by a copula, for example *film noir et blanc* [4]. We assume that the classification of the adjectives into prenominal and postnominal hierarchy classes is known and can be looked up in the dictionary. We use Arabic digits for the prenominal classes $C_1, C_2, \ldots$, Roman ones for the postnominal classes $C_I, C_{II}, \ldots$, i.e. we have classes C_h where $h \in \{1,2,\ldots\} \cup \{I, II, \ldots\}$. The lower the number of its class, the closer the adjective will be to the noun. Thus *blanc* belongs to C_I, *petit* to C_1, *pétillant* to C_{II}. This distribution into hierarchy classes necessitates the introduction of corresponding basic types for the incomplete noun phrases $\boldsymbol{x}_{hgn}$ where $h \in \{1,2,\ldots\} \cup \{I, II, \ldots\}$ or $h = 0$ for bare nouns:

We postulate

$$\boldsymbol{x}_{hgn} \to \boldsymbol{x}_{gn}, \text{ for all } h,\ \boldsymbol{x} = \boldsymbol{c}, \boldsymbol{c'}, \boldsymbol{m} \text{ or } \boldsymbol{m'}.$$

---

[1] This phenomenon is even more pervasive in the spoken language where the otherwise silent terminal consonant of a word is pronounced, if the following word starts with a vowel.

[2] There are words whose first letter is a silent *h* which are assimilated to words starting with a vowel.

[3] Some adjectives may belong to both classes, especially if classic French or regional variations are also to be covered

[4] This can be done with the usual polymorphic typing of the copula. To keep the paper in reasonable limits, we ignore this case.

The type x_{0gn} corresponds to a noun, it is given in the dictionary, for example

amande : $\boldsymbol{c}'_{021}$

pain : $\boldsymbol{m}_{011}$

eau : $\boldsymbol{m}'_{021}$

The types $\boldsymbol{x}_{ign}$, for $i \in \{1,2,...\}$ correspond to incomplete noun phrases starting with an adjective in hierarchy class C_i. The types $\boldsymbol{x}_{kgn}$ for $k \in \{I,II,...\}$ correspond to incomplete noun phrases starting with a noun and ending with an adjective in class C_k. This will be accomplished by assigning the types to adjectives as follows:

Meta-Rule 1 (Special adjectives): The masculine form singular of a few special adjectives like *beau, noubeau, vieux* has a variant *bel, nouvel, viel* to be used if the next word starts with a vowel.[5] The type of *beau*, *bel* etc. is therefore

beau,... : $\boldsymbol{x}_{211}\boldsymbol{x}_{h11}{}^{\ell}$, for $h = 1,0,I,II,...$

vieux, … : $\boldsymbol{x}_{111}\boldsymbol{x}_{h11}{}^{\ell}$, for $h = 0,I,II,...$

bel, ... : $\boldsymbol{x}_{211}\boldsymbol{x}'_{h11}{}^{\ell}$, $h = 1,0,I,II,...$

viel, : $\boldsymbol{x}_{111}\boldsymbol{x}'_{h11}{}^{\ell}$, for $h = 0,I,II,...$

where $\boldsymbol{x} = \boldsymbol{c}$, $\boldsymbol{x} = \boldsymbol{m}$.

The types of the non special adjectives and the feminine singular and the plural form of the special adjectives are described by the meta-rule below:

[5] In the spoken language, every adjective ending in a silent consonant will have a variant form where the last letter is audible if the following word starts with a vowel.

Meta-Rule 2 (Adjectives):

Let A be an adjective and A_{gn} its declined form of gender g and number n.

1) If A belongs to the prenominal hierarchy class $\boldsymbol{C}_i$, $i = 1, 2, \ldots$, then

$$A_{gn} \quad : \boldsymbol{x}_{i\,gn}\boldsymbol{y}_{hgn}{}^{\ell},\ h = i-1,\ldots,0,I,II,\ldots,$$

where

if A starts with a consonant, either $\boldsymbol{x} = \boldsymbol{c}$ and $\boldsymbol{y} = \boldsymbol{c}, \boldsymbol{c}'$ or $\boldsymbol{x} = \boldsymbol{m}$ and $\boldsymbol{y} = \boldsymbol{m}, \boldsymbol{m}'$,

if A starts with a vowel, either $\boldsymbol{x} = \boldsymbol{c}'$, $\boldsymbol{y} = \boldsymbol{c}, \boldsymbol{c}'$ or $\boldsymbol{x} = \boldsymbol{m}'$, $\boldsymbol{y} = \boldsymbol{m}, \boldsymbol{m}'$.

2) If A belongs to the postnominal hierarchy class C_k, $k = I, II, \ldots$, then

A_{gn} : $\boldsymbol{x}_{h\,gn}{}^{r}\,\boldsymbol{x}_{i\,gn}$, where $\boldsymbol{x} = \boldsymbol{c}, \boldsymbol{c}', \boldsymbol{m}, \boldsymbol{m}'$, $0 \le h < k$.

Examples:

vin : $\boldsymbol{m}_{011}$

blanc : $\boldsymbol{c}_{011}{}^{r}\boldsymbol{c}_{I11}$, $\boldsymbol{m}_{011}{}^{r}\boldsymbol{m}_{I11}$

pétillant : $\boldsymbol{c}_{011}{}^{r}\boldsymbol{c}_{II11}$, $\boldsymbol{c}_{I11}{}^{r}\boldsymbol{c}_{II11}$, $\boldsymbol{m}_{011}{}^{r}\boldsymbol{m}_{II11}$, $\boldsymbol{m}_{I11}{}^{r}\boldsymbol{m}_{II11}$,

bon : $\boldsymbol{c}_{211}\boldsymbol{c}_{011}{}^{\ell}$, $\boldsymbol{c}_{211}\boldsymbol{c}_{I11}{}^{\ell}$, $\boldsymbol{c}_{211}\boldsymbol{c}_{II11}{}^{\ell}$, …, $\boldsymbol{m}_{211}\boldsymbol{m}_{011}{}^{\ell}$, $\boldsymbol{m}_{211}\boldsymbol{m}_{I11}{}^{\ell}$, $\boldsymbol{m}_{211}\boldsymbol{m}_{II11}{}^{\ell}$, …

vin blanc

$\boldsymbol{m}_{011}\ \boldsymbol{m}_{011}{}^{r}\boldsymbol{m}_{I11} \quad \to \quad \boldsymbol{m}_{I11}$

vin pétillant

$\boldsymbol{m}_{011}\ \boldsymbol{m}_{011}{}^{r}\boldsymbol{m}_{II11} \quad \to \quad \boldsymbol{m}_{II11}$

vin blanc pétillant

$\boldsymbol{m}_{011}\ \boldsymbol{m}_{011}{}^{r}\boldsymbol{m}_{I11}\ \boldsymbol{m}_{I11}{}^{r}\boldsymbol{m}_{II11} \quad \to \quad \boldsymbol{m}_{II11}$

bon vin blanc pétillant

$\boldsymbol{m}_{211}\boldsymbol{m}_{II11}{}^{\ell}\ \boldsymbol{m}_{011}\ \boldsymbol{m}_{011}{}^{r}\boldsymbol{m}_{I11}\ \boldsymbol{m}_{I11}{}^{r}\boldsymbol{m}_{II11} \to \boldsymbol{m}_{211}\boldsymbol{m}_{II11}{}^{\ell}\ \boldsymbol{m}_{II11} \to \boldsymbol{m}_{211}$.

Similarly,

petit chat, petit chat noir: $\boldsymbol{c}_{111}$

beau (petit) chat (noir): $\boldsymbol{c}_{211}$

A comment on our use of indices is appropriate: Notice that the types of **petite chat* or **chat petit* will not reduce to a simple type. Types which differ "only" by the value of an index, say $\boldsymbol{c}_{011}$ and $\boldsymbol{c}_{111}$, are just as different types as those which look "completely different", say $\boldsymbol{m}_{\boldsymbol{II}11}$ and $\boldsymbol{c}_{012}$. The use of indices is convenient, when defining the dictionary, i.e. when assigning types to the words of the language fragment. Each index, better each position of the index in the subscript of a type symbol represents a "feature" of the concept, like gender, number, position of the adjective. The fact that a (sequence of) word(s) starts with a vowel or consonant could also reasonably be called a feature. For reasons of readability, this feature is expressed by the presence or absence of the symbol $'$ in the superscript, (prime). The theory of pregroups does not include unification of features, but indices are a good device to find efficient type assignment algorithms .

When checking whether *bon vin* is an incomplete noun phrase, we must try all possible type assignments of this sequence of two words until we hit one which reduces to the appropriate basic type. Only one of the possible types for *bon*, namely $\boldsymbol{m}_{211}\boldsymbol{m}_{011}{}^{\ell}$, will result in a string which reduces the type of *bon vin* to $\boldsymbol{m}_{211}$. Notice that another type for *bon* must be used in *bon vin blanc pétillant.* Type assignment can be made more efficient by keeping the variables as long as possible. The Meta-rule describes types for *bon* as $\boldsymbol{x}_{211}\boldsymbol{x}_{h11}{}^{\ell}$, for $h = 1,0,I,II,\ldots$, $\boldsymbol{x} = \boldsymbol{c}, \boldsymbol{m}$. This corresponds to eight or more types, depending on the number of hierarchy classes. As a first step, the improved type assignment algorithm would assign the string $\boldsymbol{x}_{211}\boldsymbol{x}_{h11}{}^{\ell}\ \boldsymbol{m}_{011}$ to *bon vin* and at a second step make $\boldsymbol{x} = \boldsymbol{m}$ and $h = 0$. Hence, a simple calculus of equality of "features" is part of an efficient type assignment algorithm.

3.2. Determiners

A determiner transforms an incomplete noun phrase into a complete noun phrase, which may be subject or object in a sentence. The notion of determiner follows [Le bon usage], it includes the indefinite article *un, une*, the definitive article, *le, la, l', les* and its contracted forms with *de* namely *du, des*, the possessive and demonstrative pronouns such as well as the preposition *de* followed by an adverb of degree like *beaucoup, peu*

The complete noun phrase formed with the possessive or demonstrative pronoun, definite or indefinite article can be subject or attribute or direct object. It has type $\boldsymbol{n}_{gn}$ or $\boldsymbol{n'}_{gn}$ where the indices g, n represent gender and number. If the latter do not matter, we use $\boldsymbol{n}$ with $\boldsymbol{n}_{gn} \to \boldsymbol{n}$. In view of Proposition 1 our work can be considered as an extension of the analysis given in [Barg-Lamb]. In some constructions with the preposition *de*, one needs to know if the complete noun phrase is formed with a mass-noun or a count-noun. Therefore it is convenient to introduce a type $\vec{\boldsymbol{y}}_{gn} \to \boldsymbol{n}_{gn}$, for $\boldsymbol{y} = \boldsymbol{c}, \boldsymbol{m}$ and $\vec{\boldsymbol{y}}_{gn} \to \boldsymbol{n'}_{gn}$, for $\boldsymbol{y} = \boldsymbol{c'}, \boldsymbol{m'}$.

General complete noun phrase

Roughly speaking, names are complete noun phrases and so are nouns, with or without adjectives, when preceded by an article or a demonstrative or possessive pronoun.

Albert : $\vec{\boldsymbol{c}}'_{11}$

Marie : $\vec{\boldsymbol{c}}_{21}$

le : $\boldsymbol{n}_{11}\boldsymbol{x}_{11}{}^{\ell}$, $\boldsymbol{x}=\boldsymbol{c},\boldsymbol{m}$

les : $\boldsymbol{n}_{g2}\boldsymbol{x}_{g2}$, $\boldsymbol{x}=\boldsymbol{c}$, $g=1,2$

ces, mes,… : $\vec{\boldsymbol{x}}_{g2}\boldsymbol{x}_{g2}$, $\boldsymbol{x}=\boldsymbol{c}$, $g=1,2$

ce, mon, ton, son, notre, votre, leur : $\vec{\boldsymbol{x}}_{11}\boldsymbol{x}_{11}{}^{\ell}$, $\boldsymbol{x}=\boldsymbol{c},\boldsymbol{m}$

la, cette, ma, ta, sa, notre, votre, leur : $\vec{\boldsymbol{x}}_{21}\boldsymbol{x}_{21}{}^{\ell}$, $\boldsymbol{x}=\boldsymbol{c},\boldsymbol{m}$

cette, mon, ton, son, notre, votre, leur : $\vec{\boldsymbol{x}}_{21}\boldsymbol{x'}_{21}{}^{\ell}$, $\boldsymbol{x}=\boldsymbol{c},\boldsymbol{m}$

l’ : $\vec{\boldsymbol{x}}_{g1}\boldsymbol{x'}_{g1}{}^{\ell}$, $g=1,2$, $\boldsymbol{x}=\boldsymbol{c},\boldsymbol{m}$

cet : $\vec{\boldsymbol{x}}_{11}\boldsymbol{x'}_{11}{}^{\ell}$, $\boldsymbol{x}=\boldsymbol{c},\boldsymbol{m}$

ces, mes,… : $\vec{\boldsymbol{x}}_{g2}\boldsymbol{x}_{g2}$, $\boldsymbol{x}=\boldsymbol{c}$, $g=1,2$

un : $\vec{\boldsymbol{x}}'_{11}\,\boldsymbol{x}_{11}{}^{\ell}$, $\boldsymbol{x}=\boldsymbol{c},\boldsymbol{m}$

une : $\vec{\boldsymbol{x}}'_{21}\boldsymbol{x}_{21}{}^{\ell}$, $\boldsymbol{x}=\boldsymbol{c},\boldsymbol{m}$

The difference between the types of *le, les* and the other determiners lies in the fact that prepositions like *de, à* contract with *le, les* to yield a new word: *du,* (**de le*), *des* (**de les*) etc.

Example :

un bon vin blanc

$$\vec{\boldsymbol{m}}'_{11}\boldsymbol{m}_{11}{}^{\ell}\ \boldsymbol{m}_{211}\boldsymbol{m}_{II1}{}^{\ell}\ \boldsymbol{m}_{011}\ \boldsymbol{m}_{011}{}^{r}\boldsymbol{m}_{I11} \to \vec{\boldsymbol{m}}'_{11}\boldsymbol{m}_{11}{}^{\ell}\ \boldsymbol{m}_{211} \to \vec{\boldsymbol{m}}'_{11}\boldsymbol{m}_{11}{}^{\ell}\ \boldsymbol{m}_{11} \to \vec{\boldsymbol{m}}'_{11} \to \boldsymbol{n}'_{11}$$

Notice that these determiners yield complete noun phrases which can be subject, object and attribute: *un bon vin blanc me plait, j’aime un bon vin blanc, c’est un bon vin blanc.*

Partitive complete noun phrase

French has complete noun phrases formed with the partitive article, *du, de la, de l’, des.* Functioning as partitive [6], *de* transforms an incomplete noun phrase into a complete one. This partitive noun phrase can be direct object of a verb (*Il mange du pain*), attribute *(C’est du sable)* or even subject (*des enfants jouent dans la rue*), i.e. the partitive article is understood as an indefinite article. In everyday French however, one rarely uses a noun phrase with the partitive article in the singular as the subject of a sentence :**Du pain est sur la table, ?De l’eau s’est infiltrée dans les fondements, ?Du sable gêne l’engrenage* are replaced by *il y a du pain sur la table, il y a de l’eau qui s’est infiltrée…, il y a du sable qui gêne l’engrenage.*

We introduce a new type $\widehat{\boldsymbol{n}}_{gn}$, $g=1,2;\ n=1,2$, together with a super-type $\widehat{\boldsymbol{n}}$ such that $\widehat{\boldsymbol{n}}_{gn}\to\widehat{\boldsymbol{n}}$. It is the type of a complete noun phrase which in general will not be used as subject. The plural partitive article *des*

[6] i.e. determining part of a mass or a group

transforms a plural count noun into a complete noun phrase. The same holds for the singular partitives *du, de la, de l'*, when preceding a mass-noun phrase. [7]

Hence the types

des : $\widehat{\boldsymbol{n}}_{g2}\boldsymbol{x}_{g2}^{\ell}$, $\boldsymbol{x}=\boldsymbol{c},\boldsymbol{c}'$ [8]

du : $\widehat{\boldsymbol{n}}_{11}\boldsymbol{m}_{11}^{\ell}$

de : $\widehat{\boldsymbol{n}}_{g1}\vec{\boldsymbol{m}}_{g1}^{\ell}$

Examples

(Je mange) des pommes

$$\widehat{\boldsymbol{n}}_{22}\boldsymbol{c}_{22}^{\ell}\ \boldsymbol{c}_{22} \rightarrow \widehat{\boldsymbol{n}}_{22}$$

(Je mange) du pain

$$\widehat{\boldsymbol{n}}_{11}\ \boldsymbol{m}_{11}^{\ell}\ \boldsymbol{m}_{11} \rightarrow \widehat{\boldsymbol{n}}_{11}$$

(Il y a) de l'air

$$\widehat{\boldsymbol{n}}_{11}\vec{\boldsymbol{m}}_{1}^{\ell}\ \vec{\boldsymbol{m}}_{11} \rightarrow \widehat{\boldsymbol{n}}_{11}$$

Two comments: The first concerns the use of a partitive complete noun phrase as the subject of a sentence. Compare

(1) *Des gens vous demandent.*

(2) **Des nombres pairs sont divisibles par deux.*

The first sentence is generally accepted, see [Le bon usage], [Carlier], [Kleiber], whereas the second is rejected. The obvious reason lies in the meaning of the two sentences. The partitive article has the meaning of a existential quantifier, whereas the rejected sentence (2) would require a universal quantifier. The latter can be rendered by the definite article *les*:

(3) *Les nombres pairs sont divisibles par deux.*

Our analysis assigns different types to the noun phrases *des nombres pairs* and *les nombres pairs*, namely $\widehat{\boldsymbol{n}}_{12}$ and $\boldsymbol{n}_{12}$. By an appropriate typing of the French sentence structure, it will therefore be possible to accept (1) and (3) and to reject (2).

Our second comment concerns the use of *de* instead of *des* if the noun is preceded by an adjective. We have

(i) *des fleurs*

(ii) *des jolies fleurs*

---

[7] The partitive article is also used to construct a postnominal complement to a noun phrase like *le temps des cerises, le goût du pain, la forme de la pomme.* One would have to declare still more types for *des, du, de* which is beyond the scope of this paper.

[8] This implies that *des jolies fleurs* is accepted as a complete noun phrase.

(iii) *de jolies fleurs*

Our typing up to this point does not recognize (iii). This can easily be repaired by assigning to *de* new types, namely

de : $\hat{\boldsymbol{n}}_{g2}\boldsymbol{c}_{hg2}{}^{\ell}$, $h = 1,2,...$

d' : $\hat{\boldsymbol{n}}_{g2}\boldsymbol{c}'_{hg2}{}^{\ell}$, $h = 1,2,...$

Notice that this typing accepts *de jolies fleurs* and *d'autres fleurs* as complete noun phrases, but not *de fleurs.* However, the latter will be analysed as a "quasi" complete noun phrase by assigning still more types to *de* as we shall explain next..

Adverbial determiners

The preposition *de* preceded by adverbs of degree like *assez*, *beaucoup, combien, peu* also functions as a determiner, for example *peu d'enfants, beaucoup de pain, combien de sable.* Hence, *d'enfants, de pain, de sable* may be considered as "almost" complete noun phrases.

Yet another type $\bar{\boldsymbol{x}}_{gn}$ will have to account for the "almost" complete noun phrases formed with *de* alone, without a subsequent definite article. Then we declare following new types for *de*

de : $\bar{\boldsymbol{c}}_{gn}\boldsymbol{c}_{gn}{}^{\ell}$, $\bar{\boldsymbol{m}}_{g1}\boldsymbol{m}_{g1}{}^{\ell}$,

d' : $\bar{\boldsymbol{c}}_{gn}\boldsymbol{c}'_{gn}{}^{\ell}$, $\bar{\boldsymbol{m}}_{g1}\boldsymbol{m}'_{g1}{}^{\ell}$,

assez, beaucoup : $\boldsymbol{n}_{gn}\bar{\boldsymbol{x}}_{gn}{}^{\ell}$, either $\boldsymbol{x} = \boldsymbol{c}$ and $n = 2$ or $\boldsymbol{x} = \boldsymbol{m}$ and $n = 1$.

Examples :

(Je mange) beaucoup *de* *pain*

$\boldsymbol{n}_{11}\bar{\boldsymbol{m}}_{11}{}^{\ell}$ $\quad\bar{\boldsymbol{m}}_{11}\boldsymbol{m}_{11}{}^{\ell}$ $\quad\boldsymbol{m}_{11}$ $\quad\rightarrow \boldsymbol{n}_{11}$

Beaucoup de *chats (arrivent)*

$\boldsymbol{n}_{12}\bar{\boldsymbol{c}}_{12}{}^{\ell}$ $\quad\bar{\boldsymbol{c}}_{12}\boldsymbol{c}_{12}{}^{\ell}$ $\quad\boldsymbol{c}_{12}$ $\quad\rightarrow \boldsymbol{n}_{12}$

Anticipating on the analysis of the sentence structure, we have introduced two different types for the complete noun phrase, $\boldsymbol{n}_{gn}$ and $\hat{\boldsymbol{n}}_{gn}$. The subtypes $\vec{\boldsymbol{x}}_{gn}$ of $\boldsymbol{n}_{gn}$ were needed to recognise the transformation of a complete noun phrase formed with a definite article (*l'eau*) into a complete "indefinite" noun phrase (*de l'eau).*

4. Conclusion

In Section 1, we have established the conditions for extending a language fragment without changing the completeness and correctness of the typing of the original fragment. The results of Section 2 show how and when the efficient general type checking algorithm can be improved to a linear one; an important property in implementations. Sections 1 and 2 combine to an argument that complexity, besides the actual grammar, could and should influence the typing of language fragments. The criterion for linearity has already influenced our typing in Section 3. There we have also explained how each position of an index in the subscript corresponds to a "feature", used to organise the dictionary in a succinct way. Thus, meta-rules are not but part of the grammar, but help to organise the dictionary.

Future work should take advantage of this succinct organisation of the dictionary to devise efficient type assignment algorithms. Certainly, more criterions for language fragments which can be decided in linear time would be welcome. We are convinced that pregroups are the tool for real time applications verifying grammars. Larger and larger fragments of natural languages need to be typed.

References

[Barg Lamb] Danièle Bargelli, Joachim Lambek, An algebraic approach to the French sentence structure, Logical Aspects of Computational Linguistics, 4th International Conference, LACL 2001, Le Croisic, France, June 27-29, 2001, Proceedings 2099: pp. 62-78, 2001.

[Carlier] Anne Carlier, La Résistance des articles *du* et *des* à l'interprétation générique, in D. Amiot et al., editors, Le syntagme nominal syntaxe et sémantique, Artois Presses Université, 2001

[Earley] Jay Earley, An efficient context-free parsing algorithm, Communications of the AMC, Volume 13, Number 2, pp 94-102, 1970

[Kleiber] Georges Kleiber, Indéfinis: lecture existentielle et lecture partitive, in G. Kleiber et al., editors, Typologie des groupes nominaux, Presses Universités Rennes, 2001

[Lambek99] Joachim Lambek, Type Grammar revisited, in A. Lecomte et al., editors, Logical Aspects of Computational Linguistics, Springer LNAI 1582, pp.1 –27, 1999

[Le bon usage] Maurice Grevisse, André Goosse, Le bon usage, grammaire française, Duculot, 2001