ELSEVIER

Theoretical Computer Science 164 (1996) 207–221

Theoretical Computer Science

A polynomial algorithm for the membership problem with categorial grammars[1]

Alain Finkel*, Isabelle Tellier

LIFAC, ENS de Cachan, 61 av. du Président Wilson, 94235 Cachan Cédex, France

Received September 1994; revised May 1995
Communicated by M. Nivat

Abstract

This article presents an overview of a framework called categorial grammars from a theoretical and algorithmic perspective. It provides an introduction to the formalisms of these grammars and of the tools they use (the Lambek Calculus) for theoretical computer scientists who are not familiar with them. We give a clear presentation of the main classical and recent results on these grammars, especially those concerning their links with context-free grammars.

The purpose is also to use these results so as to efficiently solve the membership problem with categorial grammars. We compare the complexity of the algorithms for this problem deduced from the initial formulation of the categorial grammars and the one obtained by translating these grammars into context-free formalism and by applying the Cocke–Kasami–Younger algorithm. We show that a polynomial algorithm is always available.

1. Introduction

The theory of categorial grammars is a framework that has been introduced and developed independently of Chomsky's theory of Phrase Structure Grammars. The two main formulations of categorial grammars have been given by Bar-Hillel [3, 4] and Lambek [12]. The structure of the grammars defined particularly fits for the analysis of sentences from natural languages and has been studied by linguists [7, 11, 13], logicians [5, 6, 14, 17] and computer theorists [8]. Categorial grammars are still very often used in computational linguistics [15].

From a theoretical point of view, both formalisms (Bar-Hillel's and Lambek's) have been proved to be equivalent with context-free grammars: Gaifman had proved it in 1960 for the Bar-Hillel formalism [4] but a far as the Lambek Calculus was

* Corresponding author. E-mail: finkel@lifac.ens-cachan.fr.

[1] A partial version of this article [10] has been presented at the 5th Symposium on Logic and Language (LL5), which took place in Noszvaj, Hungary, 2–6 September 1994.

SSDI 0304-3975(95)00211-1

concerned, it was an open problem since 1958 that has recently been solved by Pentus [16]. This fundamental result, that we will clearly present here, has still not been exploited as much as it could be and it is what we intend to do in this article. The most interesting consequence to be considered is the existence and explicitation of a polynomial algorithm for the membership problem with all kinds of categorial grammars, whereas only an exponential one was available till now for some of them.

This article presents the general definitions of categorial grammars and gives different algorithms proposed for the membership problem. An explicit calculation of the complexity of each of them will lead us to the most efficient ones. Some of our examples will be borrowed from Natural Language Processing, to show the usefulness of these grammars in this context.

2. The membership problem for AB-grammars

The first and most simple formalism for categorial grammars to be studied is the one proposed by Bar-Hillel in 1953 [3], inspired by earlier works performed by Ajdukiewicz [2]. These grammars are then called categorial grammars *in the sense of Ajdukiewicz Bar-Hillel* or AB-grammars. We will study the membership problem for these grammars, and show the usefulness of the Cocke–Kasami–Younger algorithm to solve it.

2.1. Preliminaries

Let us recall that an alphabet A is a finite set and A^* is the set of finite words on A, including the empty word noted ε. The length $|w|$ of a word w in A^* is the number of occurrences of letters of A in w.

Let us note $\mathbb{N}$ the set of positive integers and $\mathbb{N}^+ = \mathbb{N} - \{0\}$.

We note card(E) the cardinal of a finite set E.

A *context-free grammar* G is a 4-tuple noted $G = \langle \Sigma, U, P, R \rangle$, where Σ is the set of *terminal symbols*, U the set of *nonterminal symbols*, P the set of *production rules*, all of the form $A \to x$ with $A \in U$ and $x \in (\Sigma \cup U)^*$, and $R \in U$ is the *initial symbol* of the grammar.

We note $-^*\!\to$ the transitive closure of the derivation rule $\to$.

We say that a context-free grammar $G = \langle \Sigma, U, P, R \rangle$ can generate the word $w \in \Sigma^*$ if and only if there exists a sequence of derivations in P such that $R -^*\!\to w$.

The language of a context-free grammar G, noted $L(G)$, is the set of all words generated by the grammar: $L(G) = \{w \in \Sigma^*;\ R -^*\!\to w\}$.

A context-free grammar $G = \langle \Sigma, U, P, R \rangle$ in *Chomsky normal form* is a context-free grammar where each production rule in P is of the form:

(i) $A \to BC$ with $B \in U$ and $C \in U$;

(ii) $A \to a$ with $a \in \Sigma$;

(iii) $A \to \varepsilon$.

We recall that if $G = \langle \Sigma, U, P, R \rangle$ is a context free grammar in Chomsky normal form without the rule $R \to \varepsilon$, and w is a word in Σ^* of length n, then there exists an algorithm in $O(n^3)$, called the Cocke–Kasami–Younger algorithm, that decides if w belongs to the language $L(G)$.

We will first define an AB-grammar and the language associated with it. The key idea of AB-grammars is to associate categories to the elements of the alphabet. The categories are built with a fractional notation (with two noncommutative operators noted / and \) and relations are defined to combine categories with one another.

Definition 1 (*from* [3, 4]). An *AB-grammar* G is a 4-tuple $G = \langle V, C, f, S \rangle$ with:

- V is the *alphabet* of G;
- C is a finite set of *basic categories* of G.

From C, we define the set of *all possible categories* of G as the closure of C for / and \ noted C'. For this, we need to define a sequence of sets:

- $C_0 = C$;
- $C_1 = C_0 \cup \{x/y;\ x, y \in C_0\} \cup \{x\backslash y;\ x, y \in C_0\}$;
- for all n in $\mathbb{N}$: $C_{n+1} = C_n \cup \{x/y;\ x, y \in C_n\} \cup \{x\backslash y;\ x, y \in C_n\}$.

The closure of C, also written C', is then $C' = \bigcup_{n \in \mathbb{N}} C_n$.

- f is the *function* : $V \to \mathscr{P}_f(C')$ where $\mathscr{P}_f(C')$ is the set of finite subsets of C' which associates each element v in V with the finite set $f(v) \subseteq C'$ of its possible categories;
- $S \in C$ is the *axiomatic category* of G.

Despite the fact that C' is an infinite set, we note C'^* the set of *finite* concatenations of categories in C'.

Definition 2. For every category a and b in C', we will say that both couples $(a/b, b)$ and $(b, a\backslash b)$ can *reduce* to the category a and we note: $(a/b).b \to a$ and $b.(a\backslash b) \to a$.

Let X and Y be two elements of C'^*, X is said to be *directly rewritten* into Y (what will be noted $X \to Y$) if and only if one of these conditions holds:

(i) there exist $U \in C'^*$, $V \in C'^*$, $a \in C'$ and $b \in C'$ such that $X = U.(a/b).b.V$ and $Y = U.a.V$ (rule $R1$ with $(a/b, b)$ as reduced couple);

(ii) there exist $U \in C'^*$, $V \in C'^*$, $a \in C'$ and $b \in C'$ such that $X = U.b.(a\backslash b).V$ and $Y = U.a.V$ (rule $R'1$ with $(b, a\backslash b)$ as reduced couple).

The relation $\to^*$ is the transitive closure of $\to$.

Notation. We call Reduce(X, u, v) with $X \in C'^*$ and $u, v \in C'$ the function defined by:

- if there exists $Y \in C'^*$ such that if $X \to Y$ and the reduced couple (by the rule $R1$ or $R'1$) is (u, v), then Reduce $(X, u, v) = Y$;
- Reduce (X, u, v) is undefined elsewhere.

The fractional notation for the categories a/b (resp. $a\backslash b$) of the grammar is justified by the fact that followed (resp. preceded) by a category b, it can be rewritten into a. Operators / and \ can be considered as oriented divisions over categories.

The language of an AB-grammar can now be defined as the set of all words for which there exists at least one affectation of categories that can be reduced, thanks to the previous rules, to the axiomatic category.

Definition 3. Let $G = \langle V, C, f, S\rangle$ be an AB-grammar. Let $f^*: V^* \to \mathscr{P}_f(C'^*)$ be the extension of the function f to V^*, defined as follows for every w in V^*:

$$f^*(w) = \{c_1 \dots c_n;\ \forall i \in \{1, \dots, n\}\ c_i \in f(w_i) \text{ and } w = w_1 \dots w_n\}.$$

Definition 4. Let $G = \langle V, C, f, S\rangle$ be an AB-grammar. The *language* of G noted $L(G)$ is

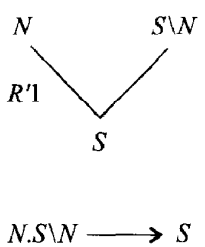
$$L(G) = \{w \in V^*;\ \exists c \in f^*(w),\ c \to^* S\}.$$

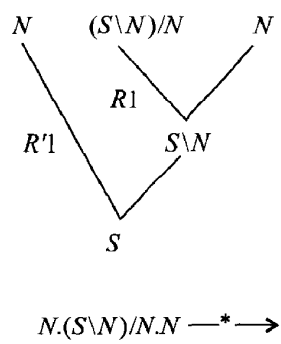
Example (*from Natural Language Processing*). Let us consider an AB-grammar $G = \langle V, C, f, S\rangle$ defined by:

- the alphabet V is equal to {John, Mary, runs, loves};
- the set of basic categories C is equal to $\{N, S\}$ (N stands for *nouns* and S for *sentence*);
- $f(\text{John}) = \{N\}; f(\text{Mary}) = \{N\}, f(\text{runs}) = \{S\backslash N\}, f(\text{loves}) = \{(S\backslash N)/N\}$;
- $S \in C$ is the axiomatic category.

Let u and v be the two following words: $u = \text{John.runs}$ and $v = \text{Mary.runs}$. Then u and v belong to $L(G)$ because: $N.S\backslash N$ belongs to both $f^*(u)$ and $f^*(v)$ and, by applying the rule R1, we can reduce it to $S: N.S\backslash N \to S$. Similarly the words $u = \text{John.loves.Mary}$ and $v = \text{Mary.loves.John}$ belong to $L(G)$ because $N.(S\backslash N)/N.N \to N.(S\backslash N) \to S$ (rule $R1$ and then rule $R'1$).

These results can be presented into analysis trees:





2.2. Complexity of the naive algorithm for the membership problem

From the previous definitions of Section 2.1, it is possible to define a naive algorithm to decide if a word belongs to the language of an AB-grammar. Let $G = \langle V, C, f, S\rangle$ be an AB-grammar and $w \in V^*$ be a word of length n.

```
Algorithm 1
(* definition of the recursive procedure that tries to reduce all possible couples of
a sequence of categories containing nbcat categories* )
procedure reduce-couples (var: sequence, nbcat):
  begin
  case:
    (nbcat = 1) and (sequence = S) then exit (*success* );
    (nbcat = 1) and (sequence ≠ S) then exit (* failure* );
    else for each (nbcat − 1) possible couples (x,y) of consecutive categories in
      sequence do if possible (* if the result of Reduce (sequence, x, y) is defined * )
        sequence := Reduce (sequence, x, y);
        nbcat := nbcat − 1;
        apply (*recursively* ) procedure reduce-couples (sequence, nbcat);
      else exit (*failure* );
  end reduce-couples;

(*main program * )
  begin
  associate each element of the alphabet with its possible categories;
  for each possible sequence of n catgories:
    apply reduce-couples (sequence, n);
  end.
```

Fig. 1.

Algorithm 1. Algorithm 1 (Fig. 1) decides if w belongs to $L(G)$.

Notation. Let k_G be the maximal number of different categories associated with the same element of the alphabet: $k_G = \max_{v \in V}(\mathrm{card}(f(v)))$.

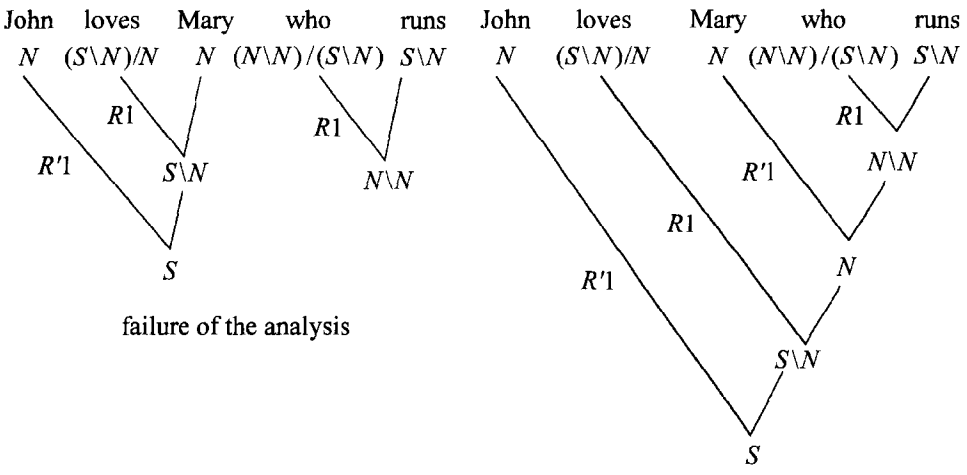
Proposition 1. *For an AB-grammar G and a word w of length n, the time complexity of Algorithm 1 is, in the worst case,* $(k_G)^n * (n-1)!$

Proof. The maximal number of possible sequences of n categories in $f^*(w)$ is bounded by (and possibly equal to) $(k_G)^n$. For each one of these sequences, one has to test every possible combination between two consecutive categories, so that a couple of categories can be reduced to one. In the worst case, from a sequence of i categories, there are $(i-1)$ possibilities to reduce it to a new sequence of $(i-1)$ categories. If we suppose necessary to develop all possible trees (because only the last one will finally lead to the desired category S), then the maximal number of series of combinations is worth: $(n-1)*(n-2)*\cdots*2 = (n-1)!$

The final maximal number of reductions is then $(k_G)^n * (n-1)!$ □

Remarks. In most cases, the time complexity of the application of Algorithm 1 is far less important. Nevertheless, in the case of Natural Language Processing (for which the membership problem is equivalent to the syntactic analysis of a sentence) it is necessary to try all possible combinations even if a successful parsing has already been performed because another one can reveal a different semantic interpretation (for ambiguous sentences).

There are also a lot of cases where a partial analysis does not lead to a complete parsing of the sentence, as in the example:



2.3. *AB-grammars and context-free grammars*

Let us now draw the links existing between AB-grammars and context-free grammars.

Theorem 1 (Gaifman [4]). *For every AB-grammar G, there exists a context-free grammar G' so that $L(G) = L(G')$.*

Proof. Let $G = \langle V, C, f, S \rangle$ be an AB-grammar. The proof is a constructive one; it consists in building an explicit context-free grammar G' that satisfies the condition. We will present here this construction; the equivalence of the languages will be considered as obvious.

First, if F is a given finite set of categories (F is a finite subset of C'), we need to define the set of the *constituents* of F, which will be noted Const(F): it is the (finite) set

containing all possible categories which appear as factors of categories in F. From F, we can define a sequence of sets:

- $F_0 = F$;
- $F_1 = \{x \in C';\ \exists y,\ x/y \in F_0$ or $x\backslash y \in F_0$ or $y\backslash x \in F_0$ or $y\backslash x \in F_0\}$;
- for all n in $\mathbb{N}$: $F_{n+1} = \{x \in C';\ \exists y,\ x/y \in F_n$ or $x\backslash y \in F_n$ or $y/x \in F_n$ or $y\backslash x \in F_n\}$,

then $\mathrm{Const}(F) = \bigcup_{n\in\mathbb{N}} F_n$.

It can be noticed that, as F is finite, there exists $k \in \mathbb{N}$ such that $\mathrm{Const}(F) = \bigcup_{n\leqslant k} F_n$.

We now define the following context-free grammar $G = \langle \Sigma, U, P, R\rangle$:

- the set of terminal symbols of G' is $\Sigma = V$;
- the set of nonterminal symbols is $U = \mathrm{Const}(f(V))$;
- the set P of production rules of G' is the union of three sets, $P = P1 \cup P2 \cup P3$, with:

  $P1 = \{A \to A/B.B;\ A/B \in \mathrm{Const}(f(V))\}$;

  $P2 = \{A \to B.A\backslash B;\ A\backslash B \in \mathrm{Const}(f(V))\}$;

  $P3 = \{A \to a;\ A \in f(a)$ and $a \in V\}$;
- the initial symbol R of G' is equal to the axiomatic category S.

The production rules $P1$ and $P2$ express all possible combinations of categories with the rules $R1$ and $R'1$, and the production rules of $P3$ express the affectation of categories to the elements of the alphabet of G. It is then obvious that an equivalence between analysis trees for G and derivation trees for G' can be defined. □

Theorem 2 (Gaifman [4]). *For every context-free grammar G', there exists an AB-grammar G so that $L(G') = L(G)$.*

The proof of this result is omitted, because it will be of no use for the remaining of this paper. Nevertheless, to exemplify that the categorial framework also fits for the definition of context-free languages, we give an AB-grammar G so that $L(G) = \{a^n b^n;\ n \in \mathbb{N}^+\}$.

Example. Let $G = \langle V, C, f, S\rangle$ be an AB-grammar defined as:

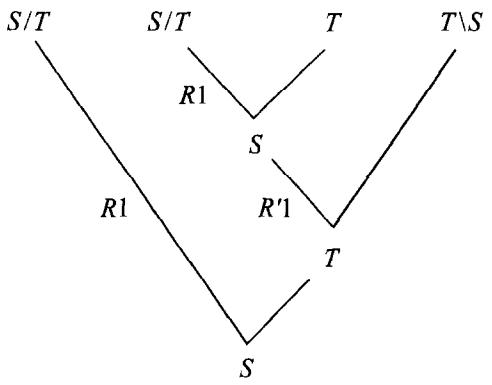
- the alphabet of G is $V = \{a, b\}$;
- the set of basic categories is $C = \{S, T\}$;
- $f(a) = \{S/T\}$ and $f(b) = \{T, T\backslash S\}$;
- $S \in C$.

We will show that $L(G) = \{a^n b^n;\ n \in \mathbb{N}^+\}$.

First, let us show by recurrence on n that for every $n \in \mathbb{N}^+$, $a^n b^n \in L(G)$:

- $n = 1$: the word $u_1 = a.b \in L(G)$ because $S/T.T \in f(u_1)$ and, by applying rule $R1$, we have $S/T.T \to S$.
- We develop the case $n = 2$ because the analysis tree is an interesting one: the word $u_2 = a.a.b.b \in L(G)$ because $S/T.S/T.T.T\backslash S \in f(u_2)$ and we can write

the analysis tree:



- Let us suppose that there exists $n \geqslant 2$ such that $u_n = a^n b^n \in L(G)$. It means that there exists a sequence L in $f^*(u_n)$ containing $2 * n$ categories so that $L \to^* S$. Then $u_{n+1} = a^{n+1}b^{n+1} = a.a^n b^n.b \in L(G)$ because $S/T.L.T\backslash S$ is a sequence of categories associated to $a^{n+1}b^{n+1}$ (it belongs to $f^*(u_{n+1})$) and $S/T.L.T\backslash S \to^* S/T.S.T\backslash S \to S/T.T \to S$, so $S/T.L.T\backslash S \to^* S$ (this sequence of reductions is the same as the one of the case $n = 2$).

So $\{a^n b^n;\ n \in \mathbb{N}\} \subseteq L(G)$.

To prove that $L(G) \subseteq \{a^n b^n;\ n \in \mathbb{N}^+\}$, we can apply Theorem 1 to the grammar G. We obtain the context-free grammar $G' = \langle \Sigma, U, P, R \rangle$ with:

- $\Sigma = \{a, b\}$;
- $U = \{S/T,\ T,\ T\backslash S,\ S\}$;
- P is constituted of the rules:
  $S \to S/T.T$;
  $T \to S.T\backslash S$;
  $S/T \to a$;
  $T \to b$;
  $T\backslash S \to b$;
- $R = S$.

It is obvious to see that G' is a classical context-free grammar in Chomsky normal form verifying $L(G') = \{a^n b^n;\ n \in \mathbb{N}^+\}$. So finally, $L(G) = \{a^n b^n;\ n \in \mathbb{N}^+\}$. □

This last argument is enough to prove both inclusions, but the first part shows how categorical grammars apply.

2.4. *A polynomial algorithm for the membership problem*

From Gaifman's constructive proof, we can deduce a new algorithm for the membership problem that will be more efficient than Algorithm 1. So let $G = \langle V, C, f, S \rangle$ be an AB-grammar, as usual, and let us consider a word w of length n.

Algorithm 2
begin
from G, compute the rules of the context-free grammar G'
apply the Cocke–Kasami–Younger algorithm to w and G'
end.

Fig. 2.

Algorithm 2. Algorithm 2 (Fig. 2) decides if w belongs to $L(G)$.

Proposition 2. *For an AB-grammar G and a word w of length n, the time complexity of Algorithm 2 is in* $O(n^3)$.

Proof. The pre-treatment to translate the grammar G into G' consists in:
- enumerating all the elements of $\text{Const}(f(V))$;
- building the sets $P1$ to $P3$.

There are at most $\text{card}(\text{Const}(f(V))) + \text{card}(f(V))$ production rules of G' to be built. This translation is a fixed sequence of operations which only depends on G and is independent of n. The grammar G' is in Chomsky normal form (without the rule $R \to \varepsilon$). The membership problem for the word w with this kind of grammar can be performed in time in $O(n^3)$ by the Cocke–Kasami–Younger algorithm. □

3. L-grammars

The formalism of AB-grammars is very restrictive as far as rewriting rules are concerned (only $R1$ and $R'1$ are allowed). In 1958, Lambek [12] has proposed a calculus whose purpose is to extend the manipulation of the categories. This calculus leads us to a new kind of formalism: the categorial grammars *in the sense of Lambek*, or L-grammars.

L-grammars are generalizations of AB-grammar. We show that the same results as the ones exposed in the previous section hold for L-grammars. As a matter of fact, the naive algorithm for the membership problem with these grammars is also exponential, but a translation into context-free formalism and the use of the Cocke–Kasami–Younger algorithm leads us to a polynomial time complexity.

3.1. Lambek calculus and L-grammar

A (product-free) *Lambek Calculus* [12] is a logical formalism among sequents: a *sequent* is a logical formula noted $A \to C$ (A for *antecedent* and C for *consequence*), where A and C are sequences of categories. The purpose of this calculus is to define a class of valid sequents, which will be considered as legal combinations of categories.

Definition 5 (*Lambek* [12]). A *Lambek Calculus* LC is a triple LC $= (K, A, R)$ where:

- K is a set of *categories* (i.e. basic categories and fractional categories);
- A is a set of *axioms*: for every category $x \in K$, the sequent: $x \to x$ is an axiom (and then is a valid sequent);
- R is a set of *inference rules* among sequents (if the sequent above the line is valid, then the one under is also valid):

$$\frac{T, x \to y}{T \to y/x} \qquad \frac{x, T \to y}{T \to y \backslash x;}$$

$$\frac{T \to x \qquad U, y, V \to z}{U, y/x, T, V \to z} \qquad \frac{T \to x \qquad U, y, V \to z}{U, T, y \backslash x, V \to z;}$$

where $x \in K$, $y \in K$ and T, U and V are sequences of categories (T must be nonempty).

Notation. Every valid sequent $X \to Y$ in a Lambek Calculus LC is noted: $\mathrm{LC} \vdash X \to Y$.

Remarks. It can be noticed that rules $R1$ and $R'1$ used in AB-grammar (Definition 2) are valid sequents of all Lambek Calculus: for every category a and b in C', we obviously have $\mathrm{LC} \vdash a/b,\ b \to a$ and $\mathrm{LC} \vdash b,\ a \backslash b \to a$. Nevertheless, Zielonka has proved [17] that no Lambek calculus LC $= (K, A, R)$ where K is infinite could be simulated by a *finite* number of such theorems considered as axioms.

A fundamental property of each inference rule in a Lambek Calculus is that there is one operator (/ or \ ) more under the line than there are above. This property assures the decidability of the calculus. As a matter of fact, for a given sequent to be proved valid, one just has to try to remove each one of its operators by applying the inference rules of R in backward chaining, until, if possible, only axioms are left.

Definition 6. A *L-grammar* G is a 5-tuple $G = \langle V, C, f, S, \mathrm{LC} \rangle$ verifying:

- $\langle V, C, f, S \rangle$ is an AB-grammar;
- LC $= (C', A, R)$ is a Lambek Calculus.

The language of a L-grammar can now be defined as the set of all words for which there exists at least one affection of categories which is the antecedent of a valid sequent in the Lambek Calculus of the grammar, whose consequence is the axiomatic category of this grammar.

Definition 7. For a L-grammar G, the language of G written $L(G)$ is

$$L(G) = \{w \in V^*;\ \exists c \in f^*(w),\ \mathrm{LC} \vdash c \to S\}.$$

3.2. Complexity of the naive algorithm for the membership problem

From the previous Definitions 5–7, it is possible to define a naive algorithm to decide if a word belongs to the language of a L-grammar. Let $G = \langle V, C, f, S, \mathrm{LC} \rangle$ be a L-grammar and $w \in V^*$ be a word of length n.

We first need to define the length $\|c\|$ of a category $c \in C'$.

Definition 8. Let $x \in C'$. The length $\|x\|$ of x is:
- $\|x\| = 1$ if $x \in C$;
- $\|x/y\| = \|x\backslash y\| = \|x\| + \|y\|$.

$\|x\|$ is the number of occurrences of basic categories in x.

Lemma. *For any category $c \in C'$, $\|c\| - 1$ is the total number of operators appearing in c.*

Proof of the lemma. Let us reason by recurrence on the length of c:
- let $c \in C$, then $\|c\| = 1$ and $\|c\| - 1 = 0$ is the number of operators in c;
- let $c = x/y$ or $c = x\backslash y$ and let us suppose that for every category $d \in C'$ verifying $\|d\| < \|c\|$, we have that $\|d\| - 1$ is the number of operators in d.
  $\|c\| = \|x\| + \|y\|$ with $\|x\|, \|y\| \in \mathbb{N}^+$ so we have $\|x\| < \|c\|$ and $\|y\| < \|c\|$ so, with the hypothesis of recurrence, $\|x\| - 1$ and $\|y\| - 1$ are the number of operators respectively in x and y, then the number of operators in c is $(\|x\| - 1) + (\|y\| - 1) + 1 = \|x\| + \|y\| - 1 = \|c\| - 1$. □

Algorithm 3. Algorithm 3 (Fig. 3) decides if w belongs to $L(G)$.

```
Algorithm 3
(* procedure that decides if A → C is a valid sequent *)
  procedure valid? (A, C):
  begin
  if A = C then exit (*success*);
  else if possible do:
(* if there are operators left and if the Lambek rules are applicable *)
     choose an operator inside the sequent A → C;
     remove it by applying one of the rules of
       inference in backward chaining;
     apply (*recursively*) valid? to the
       remaining sequents;
     else exit (*failure*);
  end valid?;
(*main program*)
  begin
  associate each word with its possible categories;
  for each possible sequence of categories:
     valid? (sequence, s)
  end.
```

Fig. 3.

Notation. Let k_G be the maximal number of different categories associated with the same element of the alphabet: $k_G = \max_{v \in V}(\mathrm{card}(f(v)))$.

Let m_G be the maximal number of operators inside the categories associated with the grammar: $m_G = \max_{c \in f(V)}(\| c \| - 1)$.

Proposition 3. *For a L-grammar G and a word w of length n, the time complexity of Algorithm 3 is, in the worst case,* $(k_G)^n * (n * m_G)!$.

Proof. As for the proof of Proposition 1, there are at most $(k_G)^n$ possible sequences of n categories in $f^*(w)$. At the beginning of the algorithm, one has to choose one among at most $n * m_G$ possible operators and remove it. For every possible choice, at each state of the algorithm, one has then to choose a new operator among the remaining ones. $(n * m_G)!$ possible trees are to be built. In the worst case, the number of operations of removing is finally $(k_G)^n * (n * m_G)!$. □

3.3. L-grammars and context-free grammars

Since L-grammars are generalizations of AB-grammars, it has been known since a long time that L-grammars had at least the power of context-free grammars. But it is only recently (1992) that Pentus has proved that L-grammars are weakly equivalent with context-free grammars [16]. We present here clearly the key ideas of this proof.

Theorem 3 (Pentus [16]). *For every L-grammar G, there exists a context-free grammar G′ so that* $L(G) = L(G')$.

Proof (*Sketch*). The center of Pentus' proof is that all sequents containing categories of a particular L-grammar and provable with the Lambek Calculus can also be proved with a restricted system where all sequents contain at most three categories.

Let $G = \langle V, C, f, S, \mathrm{LC} \rangle$ be a L-grammar and let us explicitly build a context-free grammar $G' = \langle \Sigma, U, P, R \rangle$ weakly equivalent with G.

Because the alphabet V is finite and because each member of V can be associated (with f) with a finite number of categories, it is possible to consider the maximal length of all categories in $f(V)$. Let M be the least integer such that for every $v \in V$ and for every category $c \in f(v)$ we have $\| c \| \leqslant M$. We have $M = \mathrm{Max}_{v \in V}(\mathrm{Max}_{c \in f(v)}(\| c \|))$.

We now call Z the finite set of all possible categories whose length is smaller than M: $Z = \{x \in C' \text{ such that } \| x \| \leqslant M\}$. Then, $G' = \langle \Sigma, U, P, R \rangle$ is defined by:

- the set of terminal symbols is $\Sigma = V$;
- the set of nonterminal symbols is $U = Z$;

- the set P of production rules of G' is the union of three sets, $P = P1 \cup P2 \cup P3$, with
  $P1 = \{A \to a;\ A \in f(a) \text{ and } a \in V\}$;
  $P2 = \{A \to BC;\ (A,B,C) \in Z^3 \text{ and } L \vdash BC \to A\}$;
  $P3 = \{A \to B;\ (A,B) \in Z^2 \text{ and } L \vdash B \to A\}$;
- the initial symbol R of G' is equal to the axiomatic category S.

3.4. Polynomial algorithm for the membership problem

It is surprising that, as far as we know, all consequences of Pentus' theorem have not been made explicit yet. As a matter of fact, from Pentus' constructive proof, we can deduce a new algorithm for the membership problem that will be more efficient than Algorithm 3. So let $G = \langle V, C, f, S, \mathrm{LC}\rangle$ be a L-grammar, as usual, and let us consider a word w of length n.

Algorithm 4. Algorithm 4 (Fig. 4) decides if w belongs to $L(G)$.

Proposition 2. *For a L-grammar G and word w of length n, the time complexity of Algorithm 4 is in $O(n^3)$.*

Proof. The pre-treatment to translate the categorial grammar G into G' needs a fixed number of operations independent of n. It is a long process that can be decomposed into:

- enumerating all the elements of Z;
- building the sets $P1$ to $P3$: for $P2$ and $P3$, Algorithm 3 has to be used to check all the sequents of the form $BC \to A$ and $B \to A$, for every A, B and C in Z.

Although long and complicated this treatment is performed once for all and depends only on G (and not on w).

Removing the rules of $P3$ is again a fixed treatment done in a time independent of n. It consists in:

- removing the cycles of the form $A \to^* B \to^* A$;
- from place to place, starting with the rules whose left member is R and then going on for every rule $A \to B$ in $P3$; by adding the possible right members of the rules

---

Algorithm 4
begin
from G, compute the rules of G';
remove the rules of P3;
apply the Cocke–Kasami–Younger algorithm to w and G'
end.

Fig. 4.

whose left member is B to the rules whole left member is A, and then by removing the rule $A \to B$.

This can be performed in at most $O(N^3)$ where N is the number of rules in the translated context-free grammar G': $N = \text{card}(P1) + \text{card}(P2) + \text{card}(P3)$.

Then G' is in Chomsky normal form (without rule $R \to \varepsilon$). This treatment done, the membership problem is solved in $O(n^3)$ by the Cocke–Kasami–Younger algorithm. □

4. Conclusion

The formalisms of categorial grammars are usually not very well known by theoretical computer scientists. However, a lot of recent work has shown the linguistic interests of such grammars [13, 15]. It is quite easy to define a particular AB- or L-grammar given the language it is supposed to recognized: everything depends only on the affectation of the categories to the elements of the alphabet. Till now, the main disadvantage of this approach was the poor efficiency of the algorithms available for the membership problem (which is equivalent to the syntactic analysis) [1, 11].

Our study suggests a two-phases treatment for the use of a categorial grammar. As a matter of fact, as soon as automatic manipulations are to be done with the grammar, it is more efficient to translate it into context-free formalism. After a fixed pre-treatment, a *polynomial algorithm* is available for the membership problem. This study shows that, from an algorithmic viewpoint, a direct implementation of the definitions of the grammar is far less interesting than an indirect path through context-free grammars that have already been intensively studied.

AB- and L-grammars are now included in the formal theory of grammars and in Chomsky's hierarchy. Extensions have already been proposed to the Lambek Calculus that are not context-free [9]. Pentus' theorem surely offers some other and still unexplored perspectives. It should give a new start to the study of categorial grammars.

References

[1] E. Arts, Parsing second order Lambek grammar in polynomial time, in: *Proc. 9th Amsterdam Coll.*, 1994.

[2] K. Ajdukiewicz, Die Syntaktische Konnexität, *Studia Philosophica* **1** (1935) 1–27; English translation: Syntactic connexion, in: McCall, ed., *Polish Logic 1920–1939* (Clarendon, Oxford, 1967) 207–231.

[3] Y. Bar-Hillel, A quasi-arithmetical notation for syntactic description, *Languae* 29 (1953), 47–58.

[4] Y. Bar-Hillel, C. Gaifman and E. Shamir, On categorial and phrase structure grammars, *Bull. Res. Council Israel* **9F** (1960) 1–16.

[5] J. Benthem (Van), The Lambek calculus, in [15], pp. 35–68.

[6] W. Buszkowski, Generative power of categorial grammar, in [15], pp. 69–94.

[7] N. Chomsky, Formal properties of grammars, in: R. Duncan-Luce et al., eds., *Handbook of Mathematical Psychology, Vol. 2* (New York, 1963) 323–418.

[8] J.M. Cohen, The equivalence of two concepts of categorial grammar, *Inform. and Control* **10** (1967) 475–484.
[9] M. Emms, Extraction covering extensions of Lambek calculus ae not CF, in: *Proc. 9th Amsterdam Coll.* 1994.
[10] A. Fenkel and I. Tellier, A categorial overview on categorial grammars, in: *Proc 5th Symp. on Language and Logic (LL5)*, Noszvaj, Hungary, 2–6 September 1994.
[11] H. Karlgren, Categorial grammar – a basis for natural language calculus?, *Studia Logica* **37** (1978) 65–78.
[12] J. Lambek, The mathematics of sentence structure, *Amer. Math. Monthly* **65** (1958) 154–170.
[13] M.McGee Wood, *Categorial Grammars* (Routledge, London, 1993).
[14] M. Moortgat, *Categorial Investigations: Logical and Linguistic Aspects of the Lambek Calculus* (Foris Publications, Dordrecht, 1988).
[15] R.T. Oehrle, E. Bach and D. Wheeler, eds., *Categorial Grammars and Natural Language Structure* (Reidel, Dordrecht, 1988).
[16] M. Pentus, Lambek grammars are context free, in: *8th Annual IEEE Symposium on Logic in Computer Science*, Montreal, Canada (IIE Computer Society Press, 1993) 429–433.
[17] W. Zielonka, Axiomatizability of Ajdukiewicz–Lambek calculus by means of cancellation schemes. *Z. Math. Logik Grundlag. Math.* **27** (1981) 215–224.