Algebraic View of Grammatical Inference

Alexander S. Saidi[1,2], Souad Tayeb-bey[1]

1- Laboratoire de Reconnaissance de Formes et Vision INSA de Lyon- Bât. 403
20 Ave. Albert Einstein 69621 Villeurbanne 2- Dépt. Mathématiques, Informatique et Systèmes
Ecole Centrale de LyonB.P. 163. 69131 Ecully
saidi@cc.ec-lyon.fr, tayebbey@rfv.insa-lyon.fr

Abstract. We consider the problem of grammatical inference (GI) for classes of structured documents like summaries, dictionaries, bibliographic data basis, encyclopaedias and so on. The inference is based on examples of individual sample of these documents. In this paper, we present an algebraic framework of the GI in which rewrite rules will define the process of generalisation. The implementation algorithm discussed here is used in a document handling project in which paper documents are typographically tagged and then recognised. One of the current applications in this project is to translate paper documents into machine readable form

I- Introduction

We consider the problem of grammatical inference for classes of structured documents like summaries, dictionaries, scientific reports, bibliographic basis, encyclopaedias and so on from examples. We propose an algorithm that can identify the class of regular languages from positive examples. This is the usual case in the grammatical inference for classes of documents whose structures are learned from examples. In the proposed method, we infer a regular grammar which generalise the Prefix-tree of the given positive examples. Although negative examples can be considered, the usual situation in this kind of document recognition is to deal only with the positive representation. Theses examples are structured representation of documents which are collected by optical typographic recognition techniques ([1]). It is known that any algorithm that would construct a DFA (deterministic finite automaton) with a minimum number of states compatible with all the data already processed can identify any regular language in the limit ([2]). The search space being a lattice, we propose in this paper an algebraic framework for the grammatical inference and show a function on the lattice that characterise the construction of partitions over the Prefix-tree of the representation. In this framework, an initial algebra A_G is assigned to the Prefix-tree of the strings in the sample. Then the main result is that a quotient automaton of the Prefix-tree denotes a quotient algebra A'_G whose terms are obtained by the application of a uniquely defined homomorphism from A_G to A'_G. The corresponding function induced by this homomorphism is then defined by an algorithm. We also discuss an alternative view of GI problem base on the construction of a table of successor and predecessors of any element of the alphabet. Then the relation between this table is discussed. We show how this table lead to a automaton of the lattice. The proposed algorithms are used for GI in a project on paper document processing whose one application is the translation of the document into HTML text.

II- The GI problem

The Inductive Inference paradigm is the basis of the automatic learning problem ([1]). We are interested in the Syntactical Pattern Recognition framework where many grammatical inference algorithms exist that can be used in the learning step of pattern recognition tasks ([3],[4],[5],[6], [7]).
Most of the algorithms used to infer a regular grammar deal only with positive data. However, it is well known that the class of regular languages can not be correctly identified from only positive examples. Hence, any recursively enumerable class of language is identifiable using a complete representation with positive and negative data ([14]). However, the usual situation in (paper) document recognition is to propose a generalisation of the language presented by a set on positive examples. For example, from scientific reports, we may have :

report –>abstract, acknowledgement, outline, chapter, chapter, references.
report –>abstract, outline, chapter, subchapter, chapter, index, references.
report –>abstract, acknowledgement, outline, figure-table, chapter, chapter, index.

Although one can propose some negative examples (for instance, no report without any chapter or no report without outline), the general case is the unsupervised inference and the negative information is not available. In this paper, first we give some basic definitions for reference. Then, in section IV, an algebraic view of GI problem is given in details. In section V, some practical issues and the implementation of the proposed algorithm are reported. An alternative view of GI based on tables are presented in section VI. Then, some relationships with other works in the field is recalled in the section VII.

III- Basic definitions

We assume the reader is familiar with context-free grammars, regular grammars and regular expressions ([15]). We give some basic definitions used in the rest of the paper.
Let Σ be a finite alphabet. The set of all finite strings of symbols from Σ is denoted Σ^*. The empty string is denoted by ε. The concatenation of two strings u and v is denoted uv. If a string is u=vw then v is a *prefix* of u and w a *suffix* of u. A *language* denoted by L is any subset of Σ^*.
A *finite automaton* (FA) A is a quintuplet $(Q, \Sigma, \delta, q_0, F)$ where Q is the set of states, Σ is the set of input symbols, $\delta : Q \times \Sigma^* \rightarrow 2^Q$ is the transition function, $q_0 \in Q$ is the start state and $F \subseteq Q$ is the set of final states. For an automaton A, the (regular) language accepted by A is denoted by $L(A)$.
An automaton A is *deterministic* (DFA) if for all $q \in Q$ and for all $a \in \Sigma$, $\delta(q,a)$ has at most one (state) element. A language is regular iff it is accepted by a FA. If $A=(Q, \Sigma, \delta, q_0, F)$ is a FA and π a *partition* of Q, $B(q,\pi)$ is the only block that contains q and we denote the quotient set by Q/π as the set of all partitions $\{B(q,\pi) \mid q \in Q\}$. Given a FA A and a partition π over Q, the *quotient* (or derived) *automaton* $A/\pi = \{Q/\pi, \Sigma, \delta', B(q_0,\pi), \{B_i \in Q/\pi \mid \exists q \in B_i, q \in F\}$ where δ' is defined by $\forall B,B' \in Q/\pi$, $\forall a \in \Sigma$, $B' \in \delta'(B,a)$ iff $\exists q,q' \in Q, q \in B, q' \in B' : q' \in \delta(q,a)$.
It is easy to see that for a partition π over Q, $L(A) \subseteq L(A/\pi)$. The set of all automata derived from A is a (language inclusion) lattice Lat(A). A *Regular expression* is defined by :
$\emptyset$ is a regular expression ε is a regular expression For each $a \in \Sigma$, a is a regular expression
If r and s are regular expressions, then (r | s), (rs), (r*) are regular expressions. An optional regular expression r is denoted by [r]=(r | ε) and the regular expression (r+) = r* r.
A *context-free grammar* (CFG) is denoted by G=(N, T, P, S) where N and T are finite sets. Elements of N are called nonterminals and those of T are called terminals. P is a finite set of productions. Each production rule is of the form $A \rightarrow \omega$ with $A \in N$, $\omega \in (N \cup T)^*$, $|\omega| \geq 1$. The special nonterminal S is called the start symbol. A (right) *regular grammar* is a context-free grammar whose the productions rules are of the form $A \rightarrow \alpha$. or $A \rightarrow \alpha B$. where $\alpha \in T$, $A,B \in N$. The *language* L(G) is any string $\omega \in T^*$ such that there is a derivation from S to ω (denoted by $S \Longrightarrow^* w$). By extension, the language of any nonterminal $A \in N$ is any string $\upsilon \in T^*$ such that for $\tau,\sigma \in T^*$, $S \Longrightarrow^* \tau A \sigma \Longrightarrow^* \tau \upsilon \sigma$.

Given I_+ the positive representation from a regular language L, I_+ is said to be *structurally complete* if all transitions of (the unknown) automaton A(L) are used in the acceptance of strings in I_+. We call the A(L) the *canonical automaton* of a language L which is a DFA accepting L and has the minimal number of states. The *maximal canonical automaton* MAC with respect to I_+ (where I_+ is structurally complete) is the automaton whose language is L and has the largest number of states. One can define the *prefix tree* acceptor of I_+ denoted by $PT(I_+)$ from the MCA by merging states sharing the same prefix. $PT(I_+)$ accepts only the strings of I_+. It is well known that if I_+ is a structurally complete sample of a regular language L, then there exists a partition π over the states of $PT(I_+)$ such that $PT(I_+)/\pi$ is isomorphic to A(L). The aim of this paper is to give an algebraic specification of the state partitions of $PT(I_+)$ in order to formally characterise a function on $PT(I_+)$. This is done by the definition of a function over the terms of an algebra associated to $PT(I_+)$ that produces a quotient-algebra whose terms are (by construction) isomorphic to those of $PT(I_+)$.

The following theorems delimit the search space of L(A) ([8]): If I_+ is a positive sample, the set Γ of automata such that I_+ is structurally complete with any automaton in Γ is $\mathbf{Lat(MCA(I_+))}$. If I_+ is a positive sample, L the target language and I_+ is structurally complete with respect to A(L), then A(L) is an element of $\mathbf{Lat(PT(I_+))}$.

The automaton A(L) is learnable as a partition $PT(I_+)/\pi$ isomorphic to A(L) in the lattice of all possible partitions of $PT(I_+)/\pi$. The main aim of grammatical (regular) inference is to characterise this lattice and to guide a search in it. Note that this search space grows exponentially with the size of the state set in $PT(I_+)$ and therefore with the size of I_+. The automata we will construct in the sequel are constrained automata [9] : some kind of conditional automata where transitions are constrained to fulfil some conditions. For example, a transition δ (p, a/b) means that the transition takes place if the symbol $\alpha \in \Sigma^*$ is followed by $\beta \in \Sigma^*$. By this means, one can constrain the context (left and right) of a symbol in the automaton.

IV- The Algebraic View

We use here the relation between an initial many sorted algebra and grammars [goguen-77-78]. This relation over context-free grammars is easily extended to regular grammars. Such a grammar is used as a specification (rewrite) system to define an algebra. The major property is that the defined algebra is initial in a given category. To construct the algebra associated to a context-free grammar G, each nonterminal of G is assigned to a class of derivation tree. Consequently, the nonterminals of G are sorts of a many sorted algebra whose operations are defined by the production of G. The derivation tree (and hence the language) of any nonterminal X denote the carriers of the sort X.

Let G=(N,T,P,S) be a non ambiguous context-free grammar and L be its language (we do not use variables in the sequel; the algebraic representation of grammatical programming is detailed in [11], [12]). Let L_G be the terms (strings represented by their derivation tree) of G. Let the algebra A_G-algebra be associated to G whose signature is ((N$\cup$T), P') where P' is the set of names given to productions in P. The terms of this algebra are derivation trees starting from any nonterminal of G. The terms of the A_G-algebra are constructed by using the names of the productions of G. Constants (0-air operators of A_G-algebra) are elements of T in the grammar G. Two strings will be *equivalent* if their derivation tree (terms of A_G-algebra) are equivalent. By equivalence, we mean an equivalence relation over the sorts of A_G. Since the nonterminals N are sorts of A_G-algebra, any string ω such that $X \in N$, $X ==>^* \omega$ is typed by X, particularly when $\omega \in L(G)$ where for the start non terminal $S ==>^* \omega$. Note that the terms of L_G are strings whose derivation tree are the terms of A_G. An A_G-algebra is initial in a category C if for all algebra B of C, there exist a unique homomorphism $h_G : A_G \dashrightarrow B$. By the initiality of A_G, we have a homomorphism $h_G : A_G \dashrightarrow sem$ where *sem* denote any algebra of C. One can consider the language generated by a context-free (or regular) grammar as an example of sem and hence to associate a *string semantics* to the terms of A_G. One can also define another algebra assigned to another grammar (context-free or regular) G' and to define a homomorphism from G to G'. This function can be a (syntax-

directed) translation function. Here, we are interested by h : A_G --> L_G (see [11] for the details of the construction of an A_G-algebra and h). We may note that by initiality, h always exists and is defined uniquely. h is surjective and if the grammar G is not ambiguous, then h is also injective. In this case, h will be an isomorphism from A_G to L_G. For any partition of $PT(I_+)$ where some states are considered *equivalent* (merged by some function), the generated language is a super set of I_+. We will show that there exists a uniquely defined homomorphism h from the derivations in $PT(I_+)$ to the derivations in any partition of $PT(I_+)$. We also show that for some function induced by h, terms of $PT(I_+)/\pi$ are homomorphic to those of $PT(I_+)$. Thus, we will characterise different DFA of the lattice $Lat(PT(I_+)/\pi$ by the application of this function to the terms of the A_G-algebra. Let $PT(I_+)$ and its automaton (a regular grammar) G, a partition $PT(I_+)/\pi$ from $PT(I_+)$ and its automaton G'. Let A_G and A'_G be the algebra assigned to G and G'. Note that A_G and A'_G have the same signature and A'_G is a quotient algebra of A_G. Suppose further that the sorts p and p' of A_G are equivalent ($p \equiv p'$ are in the same block of states of $PT(I_+)/\pi$). Let [p] denote the equivalence class of p.

Theorem-1 : the value of a derivation tree Z in A'_G is the equivalence class of derivation tree of Z in A_G. That is : $(A'_G)(Z) = [A_G(Z)]$. proof : this is an immediate consequence of the quotient algebra A'_G of A_G. In other words, if $p \equiv p' \in [p]$, then the derivation trees of A_G whose root are p and p' are equivalents terms in A'_G. One can define i : A_G --> Σ^* a homomorphism from $PT(I_+)$ to L (or j:A'_G --> Σ^*) and show that all strings of L whose derivation tree root is [p] are equivalents. Let $V=(N \cup T)$ be the set sort of the algebra A_G and A'_G, and h : A_G --> A'_G. Let $\equiv_h = (\equiv_{hv})\ v \in V$, be a family of relations on A_G defined by : $a \equiv_h a'$ iff $h(a)=h(a')$, $a, a' \in V$. $\equiv_h$ is called the congruence relation induced by the homomorphism h. The main problem of grammatical inference is to define h : A_G --> A'_G and to assign it a function. This function is defined by f : $PT(I_+)$ --> $PT(I_+)/\pi$. Let us define the properties of h (and f).

Theorem-2 : Let A_G and A'_G be the algebra assigned to $PT(I_+)$ and $PT(I_+)/\pi$. Furthermore, let h : A_G --> A'_G be a homomorphism and $\equiv_h$ be the congruence relation induced by h. If the homomorphism h is surjective, then $A_{G/\equiv h} \approx A'_G$. Proof : according to the theorem of homomorphism [13]. Let f : $A_{G/\equiv h}$ --> A'_G be the family of functions defined by f([a])=h(a), a is of a sort of A_G. We have [a]=[a'] implies that h(a)=h(a'). To show that f is an isomorphism : - f is bijective : as h is surjective by assumption, then f is surjective. Further, if f([a])=f([a']), then h(a)=h(a') and consequently [a]=[a']. Hence f is injective.

- f is a homomorphism : let ($r : p <-- \alpha\ q$) be an operation of A_G. The homomorphism condition may be satisfied by : $f((A_{G/\equiv h})\ r([\alpha], [q])) = f([(A_G)(\alpha, q)]) = h((A_G)(\alpha, q)) = (A'_G)(h(\alpha), h(q))$ and by homomorphism condition $= (A'_G)(f([\alpha]), f([q]))$

Informally, this means that given h and f the function induced by h, if the states $p \equiv p'$ under some conditions, then for every derivation in $PT(I_+)/\pi$, there is a derivation in $PT(I_+)$. Note that we have seen that the inverse is true in the lattice L(PT(I+)).

V- Practical issues : definition of the homomorphism h

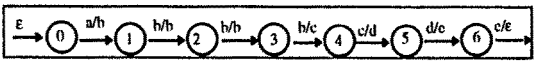
Below, we will give an algorithm (infer) that corresponds to the function f. The function *infer* induced by the homomorphism h is parametered by an integer k>0. A derivation tree in A_G is of the form p0(a1/a2, p1(a2/a3, p3(a3/a4,p4(.....))...))) where ai/aj means that ai is followed by aj. Informally speaking, if one can find two derivations. p0(a1/a2/.../ak, p1(a2/a3/...ak+1, p3(a3/a4/.../ak+2,p4(.....))...))) and q0(b1/b2/../bk, q1(b2/b3/../bk+1, q3(b3/b4/../bk+2,q4(.....))...))) where for i=1..k-1, j=i+1, ai/aj=bi/bj, then for l=0..k-1, pl≡ql.

Note that k>0 defines the extent of the function infer below where for k=0, the language of the generated partition is Σ^*. The more k is great, the "lower" is $PT(I+)/\pi$ in the lattice whose bottom element is PT(I+) and whose top element is Σ^*. It is straightforward to see that for $0<i^2j$, $L(PT(I+)/\pi, i) \subseteq L(PT(I+)/\pi, j)$. Hence, the value of the parameter k defines the depth of the search in the lattice.

The function infer induced by the homomorphism h is defined as follows :

Given an integer value k≥1, consider every string I_+. The inference process starts with *infer(<q0, ω>, {}, k)*, I_+, q_0 the start symbol (initial state). The function infer will generate a set of conditional rules for I_+ if k>1. Let the call infer(<q,ω>, G, k) be the general case. A state variable is a state like q_i where i is an unknown natural number to be defined at the end of each step.

Function infer(<q_i,ω>, G_i, k) returns G_{i+1} =
if <q,ω> = <_,ε> return G_i; if |ω|<k then let k= |ω| -- for this last turn !
if there exists a rule <p,υ> <-- <p', γ> in G_i where v=α_1,...,α_k ρ and ω=α_1,...,α_k φ and unify(q,p,G_i) (1)
then return infer(<p', α_2,...,α_k φ>, G_i) else create a rule r : <p, ω > <-- <p', α_2,...,α_k φ> -- p' of the form q , i will
return infer(<p', α_2,...,α_k φ>, G_i ∪ {r});-- be defined later. end if; end infer;

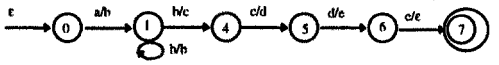


When returned from infer, different variables states of the grammar G_n of each step are assigned natural numbers beginning from the least natural not still used. Note that merges are done in infer function at (1).
The unify function sets the relations between states.

Function unify(q, p, G) returns Boolean =
if q = p then return true; else if one of (or both) p and q is a state variable (say p)
then set (p=q) in Gi; return true; else return false; end if;end if;
end unify;

V-1- A simple example

We consider the following simple positive set for I_+ : I^+ = {abbbcde, abccdee, abcde}
Let K=2. Then consider the first example " abbbcde ". The A_G term corresponding to this example is (x/y means "x" followed by "y") : p0(a/b, **p1**(b/b, **p2**(b/b, **p3**(b/c, p4(c/d, p5(d/e, p6(e/ε)))))))) depicted by the following path in PT(I_+) : Following *infer* function, the states with the same prefix of length KE are merged from 1..k-



1. Here the states 1,2 and 3 are merged and give the following automata: In other words, we have δ(p1, b/b) = p2 and δ(p2, b/b) = p3. Then p1, p2 and p3 are merged. Nota bene : merging prefixes is done in other situations in an automaton : when designing an automaton, every derivation begins from q_0 where we want that every example of I_+ begins with q_0.
The following is the trace of the construction of the grammar by the call
infer(<p0, "abbbcde">, {},2). For the sake of simplicity, we note "a"/"b".ω by "ab".ω (that is, ω∈Σ* is concatenated to the string "ab") :

<p0, "abbbcde"> Current goal There is no candidate rule Conclusion : create the rule <p0 , "ab".ω> --> <p_1, "b". ω>
<p1, "bbbcde"> Current goal create the rule <p_1, "bb".ω> --> <p_1, "b". ω>
<p_1, "bbcde"> Candidate rule< p_1, "bb". ω> --> <p_1, "b". ω> rule found **i=j. The states i and j will be merged**
<p_1, "bcde"> create the rule <p_1, "bc".ω> --> <p_4, "c". ω>
<p_4, "cde"> create the rule <p_4, "cd".ω> --> <p_5, "d". ω>
<p_5, "de"> create the rule <p_5, "de".ω> --> <p_6, "e". ω> create the rule<p_6, "e"> --> <>
<>

Here **j=i** means that these states are merged. When the derivation is completed, integer values are assigned to the variables beginning by 1. That would set i=1, k=2, l=3, m=4. In order to have a set of rules corresponding to the above DFA (see the figure above) and for the sake of clarity, let's say that k=4, l=5, m=6.
Note that during the construction of this automaton, the rule <p_j , "bb".ω> --> <p_j , "b". ω> is reused and handles the recursive part of the derivation. The set of rules of **G1** obtained from this example analysis is :

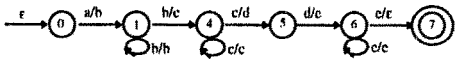
<p0 , "ab".ω> --> <p_1, "b". ω> <p_1, "bb".ω> --> <p_1, "b". ω> <p_1, "bc".ω> --> <p_4, "c". ω> <p_4, "cd".ω> --> <p_5, "d". ω> <p_5, "de".ω> --> <p_6, "e". ω> <p_6, "e"> --> <>

The regular expression : The extraction of the regular expression associated to G1 is straightforward. We have :**L(G1) = a b* c d e**

Note that we apply a usual generalisation rule applied in the Grammatical Inference where every v+, $v \in \Sigma^*$ is generalised to v*. The call infer(<p0, "abcodee">, G1, 2) for the next example adds two more rules to G1 : <p_4, "cc".ω> --> <p_4, "c". ω> <p_6, "ee".ω> --> <p_6, "e". ω>. Then **G2** is :

<p0 , "ab".ω>--><p_1, "b". ω> <p_1, "bb".ω>--><p_1, "b". ω> <p_1, "bc".ω>--><p_4, "c". ω>
<p_4, "cd".ω>--><p_5, "d". ω> <p_4, "cc".ω>--><p_4, "c". ω> <p_4, "de".ω>--><p_6, "e". ω>
<p_6, "ee".ω>--><p_6, "e". ω> <p_6, "e"> --><>

The third example "abcde" is accepted by G2 and does not change the grammar. The final DFA is :



The regular expression associated to G2 is $L(G2) = ab^*c^*de^*$

V-2- The Constraint Logic Program associated to G1

The grammar G2 can be immediately prototyped by a (constraint) logic program. We implemented the grammatical inference by a logic program which is parametered by the value of K.

The (constraint) logic program Prg^{at} associated to G2 where K=2 is given by the predicate aut^{at} :

aut^{at}(['a', 'b' | L], p0)<-- aut^{at}(['b' | L], p1).
aut^{at}(['b', 'c' | L], p1)<--aut^{at}(['c' | L], p4).
aut^{at}(['c', 'c' | L], p4)<--aut^{at}(['c' | L], p4).
aut^{at}(['e', 'e' | L], p6)<--aut^{at}(['e' | L], p6).
aut^{at}([ε], p7).
aut^{at}(['b', 'b' | L], p1)<--aut^{at}(['b' | L], p1).
aut^{at}(['c', 'd' | L], p4)<--aut^{at}(['d' | L], p5).
aut^{at}(['d', 'e' | L], p5)<--aut^{at}(['e' | L], p6).
aut^{at}(['e',ε | L], p6)<--aut^{at}([ε | L], p7).

The query <-*aut*^{at}(ω, *p0)* submitted to $Prg1^{at}$ will succeed if ω∈L(G2). In order to apply an ascending evaluation procedure, the initial state p_0 should appear in a basic clause (not in the query). To do this, the above program is transformed to a dual one (using e.g. magic set techniques [16]) in which any rule of the form aut^{at}([α,β|L], pi)<--aut^{at}([β|L], pj).is transformed to aut^{dat}([α|L], β, pj)<--aut^{dat}(L, α, pi).From Prg^{at}, we obtain Prg^{dat}:

aut^{dat}([], ε , p0).
aut^{dat}(['a' | L], 'b', p1<-aut^{dat} (L , 'a', p0).
aut^{dat}(['b' | L], 'c' , p4<-aut^{dat} (L , 'b' , p1).
aut^{dat}(['c' | L], 'd' , p5<-aut^{dat} (L , 'c' , p4).
aut^{dat}(['e' | L], 'e' , p6<-aut^{dat} (L , 'e' , p6).
aut^{dat}(['b' | L], 'b' , p1<-aut^{dat} (L , 'b' , p1).
aut^{dat}(['c' | L], 'c' , p4<-aut^{dat} (L , 'c' , p4).
aut^{dat}(['d' | L], 'e' , p6<-aut^{dat} (L , 'd' , p5).
aut^{dat}(['e' | L] ε , p7)<--aut^{dat} (L , 'e' , p6).

where a rule aut^{dat}([α | *L*], β, *q*) ← aut^{dat} (*L*, α, *p*) states that in the transition δ(p,α)=q, β has been the last symbol (the left context of α) and α will then be the left context for the next transition. To verify if a string ω∈L(G2), one should reverse ω (denoted by ϖ) and then submit the query <-aut^{dat}(ϖ, ε, *X*). This query will succeed (and will assign a value to X) in Prg^{dat} if the query <-aut^{at}(ω, p_0) succeeds in Prg^{at}(See e.g. [9] for details).

In order to prove the properties of such a program, we would simplify Prg^{dat} by omitting the transition symbols and all other variables. We obtain a logic program on which we would apply the immediate consequence operator ([18]). The least fix-point of this operator is defined by $T\uparrow^{\omega} = \cup_{i} T\uparrow^{i}$ denoted by $T\uparrow^{\omega}$ where $T\uparrow^{0} = \{aut^{dat}(p0)\}$ and $T\uparrow^{i+1} = T\uparrow^{i}$. $T\uparrow^{\omega}$ characterise the set of available states of the automaton. Furthermore, for an automaton A=(Q, Σ, δ, p_0, F), aut(f)∈$T\uparrow^{\omega}$, f∈F, if there exists a valid string (starting from q0) which ends to the final state f.

V-3- A realistic example

Let us consider the following example of final research reports. We apply the function *infer* for K=1 and K=2.

Content -> Title, Intr, Abs, Ack, Oln, Chap, Chap, Ref, Ind. Content -> Title, Abs, Oln, Chap, Chap, Chap, Ref, Anx, Anx.
Content -> Title, Intr, Abs, Oln, Chap, SChap, Chap, SChap, Ref, Ind, Anx.
Content -> Title, Intr, Oln, Chap, SChap, Chap, SChap, Chap, SChap, Ref. Content -> Title, Oln, Chap, SChap, Chap, SChap, Chap, SChap, Ref.

For the sake of simplicity, we abbreviated symbols as follows : Intr :Introduction Abs :Abstract SChap : subchapter Ack : Acknowledgement Chap : Chapter Ref : Reference Ann : Annexes Oln :Outline Ind : Index. The set of rules generated for K=1:

<q0,"Title",ω>->q7,ω> <q1,"Anx",ω>->q2,ω> <q1,ε>->ο <q2,"Anx",ω>->q2,ω> <q2,"Ind",ω>->q1,ω> <q2,ε>->ο
<q3,"SChap",ω>->q3,ω> <q3,"Chap",ω>->q3,ω> <q3,"Ref",ω>->q2,ω> <q4,"Oln",ω>->q3,ω> <q5,"Oln",ω>->q3,ω>
<q6,"Ack",ω>->q4,ω> <q6,"Oln",ω>->q3,ω> <q6,"Abs",ω>->q5,ω> <q7,"Oln",ω>->q3,ω> <q7,"Abs",ω>->q4,ω> <q7,"Intr",ω>->q6,ω>

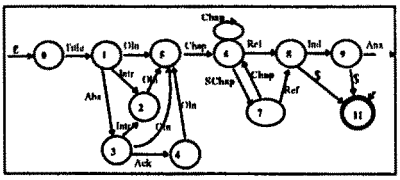
The language of this automaton is : L = Title [Intr [Abs [Ack]] | Abs] (SChap | Chap)* Ref ([Ind] Anx)* [Ind]. It is easy to note that for K=1, the language accepted by the automaton is larger set over Σ* than the one for K=2 (see also the next section). The set of rules generated for K=2 is given in the next page. Note that the recursive part of the language for K=2 is on the Chapter × Sub_Chapter part. The associated language to this automaton is : L = Title (Abs | Intr [Abs [Ack]]) L1 where L1 = Oln (Chap SChap)* (Chap SChap | Chap*) L2 L2 = Ref [Anx Anx | Ind [Anx]]. The set of rules generated for K=2 :

<q0,"Title"."Oln",ω>→<q5,"Oln",ω> <q0,"Title"."Abs",ω>→<q11,"Abs",ω> <q0,"Title"."Intr",ω>→<q8,"Intr",ω>
<q1,"Ind"."Anx",ω>→<q9,"Anx",ω> <q1,"Ind">→<> <q2,"Ref"."Anx",ω>→<q10,"Anx",ω>
<q2,"Ref"."Ind",ω>→<q1,"Ind",ω> <q2,"Ref">→<> <q3,"Chap"."Chap",ω>→<q3,"Chap",ω> <q3,"Chap"."Ref",ω>→<q2,"Ref",ω>
<q4,"Chap"."SChap",ω>→<q12,"SChap",ω> <q4,"Chap"."Chap",ω>→<q3,"Chap",ω> <q5,"Oln"."Chap",ω>→<q4,"Chap",ω>
<q6,"Ack"."Oln",ω>→<q5,"Oln",ω> <q7,"Abs"."Oln",ω>→<q5,"Oln",ω> <q7,"Abs"."Ack",ω>→<q6,"Ack",ω>
<q8,"Intr"."Oln",ω>→<q5,"Oln",ω> <q8,"Intr"."Abs",ω>→<q7,"Abs",ω> <q9,"Anx">→<> <q10,"Anx"."Anx",ω>→<q9,"Anx",ω>
<q11,"Abs"."Oln",ω>→<q5,"Oln",ω> <q12,"SChap"."Chap",ω>→<q4,"Chap",ω> <q12,"SChap"."Ref",ω>→<q2,"Ref",ω>

VI- An alternative view : The table of contexts

We give here an alternative construction of the result of *infer* and give the related automaton. We believe that this representation is a suitable alternative for the GI problem with a set of positive examples.This is done by the construction of a table where every symbol of Σ is given with its successor and predecessor. For the above example, we have :

Symbol α∈Σ	Predecessor of α	Successor of α
Title	ε	Intr, Abs, Oln
Intr	Title	Abs, Oln
Abs	Intr, Title	Ack, Oln
Ack	Abs	Oln
Oln	Ack, Abs, Intr, Title	Chap
Chap	Oln, Chap, SChap	Chap, Ref, SChap
Ref	Chap, SChap	Ind, Anx, $
Anx	Ref, Ind, Anx	Anx, $
SChap	Chap	Chap, Ref
Ind	Ref	Anx, $



The special symbol '$' denotes the end of a string (a final state of the automaton; a '$' is added at the end of each string before analysing) while ε denote the empty string at the beginning of a string. In fact, the table shows that some symbol of Σ can be followed (resp. preceded) by a set of other symbols in Σ. The canonical automaton associated to this table is (once again, for the sake of clarity, we distinguish between ε - the empty string at start - and '$' which is the very last symbol of each string) : Note that there are three recursive parts on the nodes 6, 6-7-6, and 10. In order to generalise the examples of I_+, the context in which the symbol α∈Σ is present can be "forgotten" and let the symbol α be followed by its successor anywhere in a string whatever be the predecessor of α. Let a symbol α∈Σ, P the set of its predecessors and S the set of its successors. Consider also the (language inclusion) lattice(PT(I_+)) whose top element is Σ* and whose bottom is PT(I_+).An interesting use of this table is based on the well known but yet simple generalisation rule in Inductive Learning ([19]) which "forgets" the context of a sequence. Consequently, in the case of the above table, we can :

1- Allow α∈Σ to be followed by an element of S and preceded by an element of P according to I_+. This corresponds to K=3 in function *infer*. For example, if α="Abs", the only possible triplets of the form β."Abs".δ are <Intr, Abs, Ack>, <Title, Abs, Oln> and <Intr, Abs, Oln>. 2- Allow α∈Σ to be followed by any element of S and preceded by any element of P. This gives a good degree of generalisation. This is a special case of infer with K=2. For α="Abs", the possible cases are : {"Intr", "Title"} x "Abs" x {"Ack", "Oln"} which makes 4 possibilities (generalisation). 3- Allow a∈Σ to be preceded and followed by any element of Σ*. This is a special case of infer with K=0. This gives the Top element of the lattice. For α="Abs", the possible cases are β x "Abs" x δ with δ,β∈Σ. Other configurations are possible which correspond to K=1 : allowing α to be only followed (resp. preceded) by any symbol of S (resp. P) and

free for the rest. However, during our experimentation on paper documents, the case (2) above showed to be a reasonable configuration. While an automaton specifies rules to construct sequences of, say characters, the choices above (variations of K) let generate strings with various degree of constraints. Note that if negative examples are taken into account, more constraints must be verified and the precedence table alone will not suffice.

VII- Conclusion

A new algorithm for GI has been presented which is immediately prototyped by a constraint logic program. The algebraic specification allows to show that the homomorphism *h* exists and we gave an implementation of it by the *infer* function. An alternative view of GI based on the construction of the table of successors and predecessors of any symbol in Σ^* has been outlined. The relationship between this alternative and infer function and the formalisation of this approach are in hands. Most of the related works on Grammatical Inference deal with positives and negative examples. When only positive examples are available (which describe the characteristic cases), researchs concern rather the Structured Documents field and have led to several document standards like ODA and SGML. Among other works in the field, [20] and [21] proposed similar methods for document analysis. But the algebraic framework of the grammatical Inference, the logical aspects and the table manipulation for the direct grammar extraction have, as well as known, not yet been investigated. This work is developed inside a paper document processing project where GI results are used to classify and then translate documents into machine readable form. The generated logic program is augmented to handle some attributes of the logical structure of paper documents such as typographic attributes.

VIII- References

[1] S. Tayeb-Bey, A. S. Saidi "Grammatical Formalism for Document Understanding System : From Document towards HTML Text". BSDIA'97, November 1997, Brasilia.

[2] E.M. Gold. "Language identification in the limit". Information and Control, 10(5)- 1967.

[3] H.S. Fu and T. Booth : "Grammatical Inference: Introduction and Survey". parts 1 & 2. IEEE Trans. Sys. man and Cyber. SMC-5: 95-11.

[4] R. C. Gonzalez and M. G. Thomason. "Syntactic Pattern Recognition, an Introduction". Addison Wesley. Reading Mass. 1978.

[5] H.S. Fu. "Syntactic Pattern Recognition and Applications". Prentice hall, N.Y. 1982.

[6] L. Miclet. "Grammatical Inference". Syntactic and Structural Pattern Recognition. H. Bunk and San Feliu eds. World Scientific.

[7] J. Onica, P. Garcia. "Inferring regular Languages in Polynomial Update time". Pattern Recognition and Image Analysis. 1992.

[8] P. Dupont, L. Miclet & E. Vidal. "What is the search space of Regular Inference?". ICGI'94, Grammatical Inference and Applications. Springer-Verlag-94.

[9] L. Fribourg, M. V. Peixoto. "Automates concurrents à Contraintes". TSI. 13 (6). 1994.

[10] J. A. Goguen, J.W. Tatcher, E.G. Wagner, J.B. Wright. "Initial Algebra Semantics and Continuous Algebra". JACM 24(1). 1977.

[11] A. S. Saidi : "Extensions Grammaticales de la Programmation Logique". PhD. 1992.

[12] A. S. Saidi. "On the unification of phrases". IFIP-94 .

[13] H. Ehrig, B. Mahr. " Fundamentals of Algebraic Specification". Vol-1 & 2. Springer-Verlag1985.

[14] E.M. Gold. "Complexity of automaton identification from given data"". Information and Control, 37- 1978.

[15] J. E. Hopcroft, J.D. Ullmann. "Formal Languages and their Relation to Automata". Addison-Wesley 1969.

[16] F. Bancilhon & all. "Magic Sets and Other Strange Ways to Implement Logic Programs". Proc. ACM Symp. on principles of Databases Systems. Boston 1986.

[17] F. Coste, J. Nicols : "Regular Inference as a graph coloring Problem". ICML'97. 1997.

[18] K.R. Apt, M.H. Van Emden : "Contribution to the Theory of Logic Programming". JACM. 29(3°. 1982.

[19] R.S. Michalski & all. "Machine Learning : An Artificial Intelligence Approach", vol. 1 & 2. Springer-Verlag 1984 and Morgan Kaufmann 1986.

[20] H. Ahohen, H. Mannila. "Forming Grammars for structured documents". Research report. University of Helsinki. 1994.

[20] P. Frankhauser, Y. Xu. "MarkitUp! an incremental approach to document structure recognition". Electronic Publishing-Organisation, Dissemination and Design, 6(4). 1994.