RAIRO. Informatique théorique

Gianni Aguzzi

The theory of invertible algorithms

RAIRO. Informatique théorique, tome 15, nº 3 (1981), p. 253-279
<http://www.numdam.org/item?id=ITA_1981__15_3_253_0>

R.A.I.R.O. Informatique théorique/Theoretical Informatics
(vol. 15, n° 3, 1981, p. 253 à 279)

THE THEORY OF INVERTIBLE ALGORITHMS (*)

by Gianni AGUZZI (1)

Communicated by J.-F. PERROT

Abstract. — *A class of Markov algorithms, called Invertible Pointer Algorithms (IPA), is defined and its main properties are given. It is shown that every IPA implements a bijective function whose inverse function is directly defined by the algorithm itself when each of its rules is considered in the "reverse way". Moreover, the complete equivalence between the class of IPA's and that of bijective functions over recursive domains is shown, by proving that for every such bijective function it is always possible to define an equivalent IPA. Some examples are presented as well as possible applications and extensions are outlined.*

Résumé. — *On définit une classe d'algorithmes de Markov, nommée Invertible Pointer Algorithms (IPA), et on donne ses principales propriétés. Il est montré que chaque IPA représente une fonction bijective dont la fonction inverse reste directement définie par l'algorithme même en considérant chacune de ses règles « à l'envers ». De plus, on établit l'équivalence entre la classe des IPA's sur domaines récursifs et celle des fonctions bijectives sur domaines récursifs, en faisant voir que pour toute telle fonction il est toujours possible définir un IPA équivalent. Des examples sont donnés et on mentionne des applications et extensions possibles.*

INTRODUCTION

In the recent past years many studies have been devoted to the subject of Normal Markov Algorithms (NMA) [14] and related Markov Algorithm based computing systems [9, 10, 7, 6, 5, 13].

A central role in this area has been played by the studies on the improvement in the execution time: by imposing certain conditions in the rules of the algorithm, *see* for example Katznelson [11], or using the concept of "pointer" in order to speed up the search for the occurrence of a given subword into a given word and for the proper rule into the set of rules as in Cerniavsky [8] or

(*) Received February 1980, revised August 1980.
(1) Istituto di Matematica Applicata, Facoltà di Ingegneria, 50134 Firenze, Italy.

R.A.I.R.O. Informatique théorique/Theoretical Informatics, 0399-0540/1981/253/$ 5.00

Laganà [12], Leoni [13] and Aguzzi [5]; more recently, *see* Paget [15], an abstract machine (called Machine à Memoire Associative) has been proposed as an useful model for a faster execution of every Markov Algorithm based computing system, so letting the concept of NMA unchanged.

The present paper originated in our work on an automatic translator writing system for programming languages (p.l.), called APS [5, 2, 3, 4].

Our present idea about a possible automatization of the compiler writing job is very simple: suppose we have a formal system by means of which it is possible to write down the formal specification, in an operational way, of the semantics of p.l.'s. Let this system be, for example, an algorithmic system like APS, then the semantics of a p.l., S, can be stated building up an algorithm which is able to map any program $P \in S$ into a suitable object C belonging to a given set of structures, say O, for example a graph of states assumed during the virtual excution of P, or a representation of the final state reached by the machine on which P has been executed, according to the fact we are giving a sort of translator or interpreter oriented way of defining the semantics of S. Let A_S be such definitory algorithm, so we visualize the above sketched ideas in the following way:

$$\forall P \in S, \qquad A_S(P) = C, \qquad C \in O.$$

Now, suppose such an algorithm A_{S1} for a given p.l. $S1$ and an analogous algorithm A_{S2} for another language $S2$ are available. Let $S1$ be just the source language for which we are interested to get a translator of its programs into equivalent programs of $S2$, $S2$ being the target language.

If algorithm A_{S2} satisfies the following property:

$$\forall P \in S2, \quad A_{S2}(P) = C, \quad C \in O \qquad \text{and} \qquad A_{S2}^{-1}(C) = P,$$

where A_{S2}^{-1} is the algorithm representing the inverse of function given by A_{S2}, A_{S2} so being bijective over O, then:

$$\forall P \in S1, \quad A_{S1}(P) = C, \quad C \in O \qquad \text{and} \qquad A_{S2}^{-1}(C) = P', \quad P' \in S2,$$

where P' represents the desired translation of P.

At a first glance the situation seems no so good: in order to get the translation of programs of $S1$ we have to construct algorithm A_{S2}^{-1} if possible, once the semantics of $S1$ and $S2$ have been stated by means of A_{S1} and A_{S2}. This is just the start point for this work. In this paper, the existence of a formal system, to write down algorithms, is shown, such that whenever function f simulated by the algorithm is a bijective correspondence between its domain and range, then the algorithm itself interpreted in the "reverse way", constitutes, in a direct and natural way, the inverse function f^{-1}.

Thus, our first problem has a satisfactory solution: it suffices to give the semantics of $S1$ and $S2$ in the above sketched way by means of algorithms A_{S1} and A_{S2} written in the form of IPA (i.e. Invertible Pointer Algorithm, the formal system defined in the present paper), in order to directly have any translation from $S1$ to $S2$ and vice versa.

The study of actual applications of such ideas will be the subject of further work, we have always to look for algorithms representing bijective functions possibly narrowing in a suitable way their domain and/or range. For the moment let us introduce the basic notions in order to get effective "invertible" algorithms.

The main result of this paper can be summarized in the following two propositions:

"every IPA represents a bijective function, and its inverse function is represented by the IPA obtained by reversing each rule of the original one" and conversely:

"every bijective function over recursive domain is implemented by a suitable IPA".

Moreover, other interesting properties of IPA's are shown referring to the various ways IPA's can be composed among them, still obtaining an IPA, namely:

"the class of IPA's is closed under the operation of algorithm composition" and "given any pair of IPA's, say A and B, an IPA C is definable such that for any input words w and v, $C(w \star v) = A(w) \star B(v)$"; one more property is given, reflecting the way a new IPA is defined starting from a given one, namely "given an IPA A and a character 'a', a recursive IPA B is definable such that for any input word w $B(w) = if\ a \notin w\ then\ w\ else\ B(A(w))$, provided the process terminates", i.e. $B(w)$ is the first of the words $w_0 = w$, $w_1 = A(w_0)$, ..., $w_i = A(w_{i-1})$, ... such that 'a' does not occur in it.

This paper starts by giving in section 1 the definition of a class of Pointer Algorithms (PA) (*see* [5, 13]), called Right end Conditioned Pointer Algorithms (RCPA), which is shown to be equivalent to the NMA's class. This class represents a sort of normalization and extension of the NMA concept: in fact, every RCPA terminates just after the right end of the object string has been examinated and, moreover, class-names for subsets of the given alphabet can be used into the rules of the algorithm and, finally, the application of every rule can also be conditioned by the validity of a given predicate.

In section 2, the concept of "disjoint rules" is given and the notion of IPA is defined as a subclass of RCPA's. Then, the above mentioned properties are stated and some example is presented.

1. RIGHT END CONDITIONED POINTER ALGORITHMS

Let us introduce a class of NMA's which constitutes, on one hand, a sort of normalized form for NMA's, since every algorithm of this class always terminates after the examination of the rightmost characters of the object word and, on the other hand, represents an extension of the concept of NMA's conditioning the application of each rule (if it is desired) on the occurrence of a certain subword into the object word, as usual, and on the validity of a given condition over the left and right context of the matching subword in the object word and also permitting the use of names for finite classes of characters into the rules (for a larger extension of this last concept *see* also [5]).

Notation

An alphabet is any finite set A and its elements are called *characters* or *symbols*. A^* is the free monoid on A and its elements are called *words* or *strings* on A; the identity of A^* is the empty word λ.

If w, $y \in A^*$, the operation defined on A^* is called *concatenation* and is indicated by "wy". Let us call *length* the monoid homomorphism $|\ \ | : A \to N$, N being the set of natural numbers, defined by $|a|=1$, $\forall a \in A$ and $|\lambda|=0$. The free semigroup generated by A is denoted by A^+ and $A^* = A^+ \cup \{\lambda\}$.

If $w_1, w_2 \in A^*$ and there exist two words $w', w'' \in A^*$ such that $w_1 = w' w_2 w''$, then w_2 is a *subword* of w_1.

Given any word $w \in A^*$ and a relative integer n such that abs $(n) \leqq |w|$, with $n \uparrow w$ we denote the first n characters of w if $n > 0$ or the last $-n$ characters if $n < 0$ and λ if $n=0$; with $n \downarrow w$ we denote the word w leaving its first n characters if $n > 0$ or the word w leaving its last $-n$ characters if $n < 0$ or w itself if $n=0$. If abs $(n) > |w|$, then both $n \uparrow w$ and $n \downarrow w$ are λ.

In order to formalize the use of class-names for finite subsets of a given alphabet into the rules of an algorithm, let us give the following definitions.

DEFINITION 1: Let $\Gamma_\sigma = (I, O, X, \sigma, \Gamma)$ be a NMA or a Pointer Algorithm (PA), as defined in [13], where I is the input alphabet, O the output alphabet, X disjoint from $I \cup O$ is the auxiliary alphabet, σ is the start pointer and Γ a m-tuple of transformation rules, a *class* or *character language* is any subset of $T = I \cup O \cup X$. Let B be an alphabet disjoint from T and $\mu : B \to 2^T$ be a function from B into the set of classes; if $Z \subseteqq T$ is a class for which an a $a \in B$ exists such that $Z = \mu(a)$, then a is a *class-name* of Z.

We remark that a class can have several names. This concept of class-name is useful in writing down the rules of an algorithm and has been informally used since the introduction of NMA's.

DEFINITION 2: Let T and B as above, if $y \in (T \cup B)^+$, $y = y_1 y_2 \ldots y_n$, a word $w \in T^+$, $w = w_1 w_2 \ldots w_n$, *matches* y iff:

(i) $\forall i, j = 1, 2, \ldots, n$ if $y_i = y_j$ then $w_i = w_j$;

(ii) $\forall i = 1, 2, \ldots, n$ if $y_i \in T$ then $w_i = y_i$;

(iii) $\forall i = 1, 2, \ldots, n$ if $y_i \in B$ then $w_i \in \mu(y_i)$.

The *language* of y is the subset of T^+ defined as:

$$L(y) = \{w \in T^+ \mid w \text{ matches } y\}.$$

We can now give the definition of Right end Conditioned Pointer Algorithms as:

DEFINITION 3: A Right end Conditioned Pointer Algorithm (RCPA) R_σ is a 8-tuple $R_\sigma = (I, O, X, B, C, \sigma, \omega, R)$ where:

I is the input alphabet;

O is the output alphabet;

X disjoint from $I \cup O$ is the auxiliary alphabet with $P \subset X$, set of pointers; recall that given a NMA a character $\rho \in X$ is a *pointer* for it iff it occurs at most once in all the words (*labels*) obtained during the computation for every input word;

B disjoint from $T = I \cup O \cup X$ is the set of class-names for $S = I \cup O \cup X'$, with $X' = X - P$;

C is a finite set of recursive predicates (or conditions) over the set $K = S^* \times S^*$;

σ is the *start* pointer;

ω is the *stopper* pointer;

R is an ordered m-tuple of rules which are triples (p, q, c) where:

(1) p, the left hand member (l. h. m.), and q, the right hand member (r. h. m.), belong to $(T \cup B)^+$,

(2) both p and q contain exactly one pointer in P,

(3) whenever $a \in B$ occurs in q it also occurs in p,

(4) triples containing the same pointer in their l. h. m.'s are consecutive,

(5) c is a predicate built up by means of usual logic and relational operators, elementary string functions and, possibly, predicates in C; a triple as above, is usually written as:

$$p \to q \text{ if } c,$$

where " $\to$ " is the *transformation arrow* and is called *conditioned rule;* if c is the constant predicate 1 (for true), the conditioned rule is simply written as $p \to q$, i. e. omitting the condition part,

(6) the unique *initial* rule (the first in R) has the form $(\sigma \Delta p', q, c)$, $p' \in I^*$, and

σ, the start pointer, never occurs in any other l.h.m. or r.h.m. of rules in R, $\Delta \in X'$ is the *left delimiter* for any input word,

(7) the unique *stopper* rule has the form $(p, q' \Omega\omega, c)$, $q' \in O^*$, where ω, the stopper pointer, never occurs in any other l.h.m. or r.h.m. of rules in R, and $\Omega \in X'$ is the *right delimiter* for any input word,

(8) for any input word $w \in I^*$, the result word, if any, has the form $\Delta\hat{w}\,\Omega\omega$, with $\hat{w} \in O^*$.

In order to correctly apply RCPA's, we need the following:

DEFINITION 4: Let $R_\sigma = (I, O, X, B, C, \sigma, \omega, R)$ be a RCPA as above, a conditioned rule (p, q, c) in R is *applicable* to a word $w \in T^+$, $T = I \cup O \cup X$, iff:

(i) w contains as a subword an element of the language of p, as defined in definition 2, i.e. $w = w' \bar{p} w''$, with $\bar{p} \in L(p)$ and $(w', w'') \in K$, $K = S^* \times S^*$ as defined in definition 3;

(ii) $c(w', w'') = 1$, i.e. condition c is satisfied by the left and right context of the matching subword into the word w.

Remark that, due to the main property of PA's (*see* theorem 1 in [5]), since in any label of the computation one only pointer occurs in it, if $\bar{p}$ is a subword of w, $\bar{p} \in L(p)$, no other subword of w can exist matching p.

Furthermore, w *immediately generates* $\hat{w}$ by means of the i-th rule (p_i, q_i, c_i), i.e. $w \vdash_i \hat{w}$ iff the i-th rule is the first in R applicable to w and $\hat{w} = w' \bar{q}_i w''$, where $\bar{q}_i$ is obtained from q_i substituting to every occurrence of some character $a \in B$ the character in $\bar{p}_i$ corresponding to the same a in p_i, i.e. the character associated to a during the matching phase.

In the sequel, whenever a word w has a subword matching the l.h.m. p of a given rule (p, q, c), will be freely represented as $w = w' p w''$ as well as the resulting word $\hat{w}$, after application of the rule, will be also represented as $\hat{w} = w' q w''$.

Finally, R_σ is *applicable* to a word w with $w \in I^*$ iff it exists a word $\hat{w} \in O^*$ such that $\sigma \Delta w \Omega \vdash_s^* \Delta \hat{w}\, \Omega\omega$, where $\vdash_s^*$ denotes the reflexive and stable closure of $\vdash$.

The application of a RCPA R_σ to a word $w \in I^*$ will be then indicated as: $R_\sigma(w)$ or $R_\sigma(\Delta w \Omega)$ or $R(\sigma \Delta w \Omega)$ according to the present interest in the context.

It is obvious that the presence of pointer ω in the result $\Delta\hat{w}\,\Omega\omega$ is only to signal that the output word has been generated. In any case, the role of such "artificial" pointer will be fully clarified in the next section.

For the sake of generality and completeness, let us give the following property of RCPA's.

THEOREM 1: *The class of RCPA's is equivalent to that of NMA's.*

Proof: Standing the equivalence between PA's and NMA's (*see* [5]), the proof is based on the construction, for every RCPA, of the equivalent PA on one hand, and, for every PA of the simply obtained RCPA on the other. For the complete construction *see* [1].

Sometimes a RCPA $R_\sigma=(I, O, X, B, C, \sigma, \omega, R)$ will be also simply indicated as the 7-tuple $R_\sigma=(I, O, X, C, \sigma, \omega, R)$ hiding the set of class names B, but reporting along with the rules of the algorithm the used class-names and the corresponding sets given by function μ.

We are now in a position to attack the main problem concerned with bijective functions and their inverse.

2. THE IMPLEMENTATION OF BIJECTIVE FUNCTIONS: INVERTIBLE POINTER ALGORITHMS

2.1 Disjunction of rules

Having in mind to characterize RCPA's implementing bijective correspondences between some domain $D \subseteqq I^*$ and range $R \subseteqq O^*$, let us now introduce some more definitions which will be fundamental in the sequel. The idea is to point out the concept of "disjoint patterns" as those described by (at least) a couple of pairs (q_i, c_i), (q_j, c_j) derived by the two conditioned rules (p_i, q_i, c_i) and (p_j, q_j, c_j).

DEFINITION 5: Let A be any finite alphabet, B the set of class-names of subsets of A and let the function $\mu : B \to 2^A$ as above. We say that $a, b \in A \cup B$ *agree* $(a \simeq b)$ iff:

(1) $a=b$; or

(2) $a \in B$, $b \in A$ and $b \in \mu(a)$ or $a \in A$ and $b \in B$ and $a \in \mu(b)$; or

(3) $a \in B$, $b \in B$ and $\mu(a) \cap \mu(b) \neq \emptyset$.

The negation of "$\simeq$" will be denoted by "$\not\simeq$" and it is called *disjunction relation*. It is trivial to show that $\simeq$ is an equivalence relation.

DEFINITION 6: Given an alphabet A as above, let $w=w'tw''$ and $v=v'zv''$ be two words such that $w', w'', v', v'' \in (A \cup B)^*$ and $t, z \in P$, with $P \cap (A \cup B)=\emptyset$. Then, w and v are called *simple disjoint structures* $(w \, \mathrm{d} \, v)$ iff:

(1) $t \neq z$; or

(2) $t=z$ and, if $|w'|=m$ and $|v'|=k$, at least an integer j does exist, $1 \leqq j \leqq \min(m, k)$, such that $w'_{-j} \not\simeq v'_{-j}$, where w'_{-j} and v'_{-j} represent the j-th character of w' and v' respectively, starting from their right end and going to the left; or

(3) $t=z$ and, if $|w''|=n$ and $|v''|=h$, at least an integer i does exist, $1 \leqq i \leqq \min(n, h)$, such that $w_i'' \not\updownarrow v_i''$, where w_i'' and v_i'' are the i-th character of w'' and v'' respectively, starting from the left.

Let us see some simple examples related to definition 6.

Example 1: Let $A=\{a, b, c\}$ and $x, y \in B$ such that $\mu(x)=\{a, b\}$ and $\mu(y)=\{a\}$ and let $P=\{\alpha, \beta\}$.

(1) Let $w=b\,\alpha$ and $v=ab\,\alpha\,c$, w and v are not simple disjoint structures, since it is impossible to satisfy none of conditions (1)-(3); in fact, $w'=b$, $|w'|=1$, $v'=ab$ and $|v'|=2$, so that $\min(2, 1)=1$ and $v'_{-1}=w'_{-1}$; on the other hand $w''=\lambda$, $|w''|=0$, $v''=c$ and $|v''|=1$, so $\min(0, 1)=0$, then it is impossible to find an iteger i such that $1 \leqq i \leqq 0$.

(2) Let $w=\alpha\,b$ and $v=ab\,\alpha\,c$, it is easily seen that v and w are simple disjoint structures, since it is possible to find an index i, $1 \leqq i \leqq 1$, such that $w_1''=b \neq c=v_1''$.

(3) Let $w=ab\,\beta\,x$ and $v=ax\,\beta\,y$ then w and v are not simple disjoint structures since $w'_{-1}=b$ and $v'_{-1}=x$ with $b \in \mu(x)$, furthermore $w'_{-2}=v'_{-2}=a$, on the other hand $w_1''=x$, $v_1''=y$ and $\mu(x) \cap \mu(y) \neq \emptyset$.

From the above definitions 5 and 6 the following properties hold:

(p1) the empty word λ is never disjoint from any word $w \in A^*$; i.e. λ agrees with every character;

(p2) if w and v are simple disjoint structures, then also $w'=z'wz''$ and $v'=z'vz''$ with $z', z'' \in (A \cup B)^*$ are simple disjoint structures.

Once defined what simple disjoint structures are, the next step is to formalize what disjoint conditions are, so let us give the following:

DEFINITION 7: Given two conditioned rules of a RCPA (p_i, q_i, c_i) and (p_j, q_j, c_j), where $q_i=q_i'\,\pi\,q_i''$ and $q_j=q_j'\,\rho\,q_j''$ with $\pi, \rho \in P$ (pointer set of the RCPA) and $|q_i'|=m'$, $|q_i''|=m''$, $|q_j'|=n'$ and $|q_j''|=n''$, let $D_e \subseteqq K$ $(K=S^* \cup S^*,\ S=I \cup O \cup (X-P))$ be the extension of condition c_e with $e=i, j$, i.e. $D_e=\{(w', w'') \in K : c_e(w', w'')\}$, we say c_i *disjoint from* c_j $(c_i\,\mathrm{D}\,c_j)$ if the following predicate holds:

$$\begin{aligned}
&\textit{if } (m'=n') \textit{ and } (m''=n'') \textit{ then } ((w', w'') \in D_i)\\
&\qquad\qquad\qquad \Rightarrow \textit{not } c_j(w', w'') \textit{ and}\\
&\qquad\qquad\qquad\qquad ((w', w'') \in D_j) \Rightarrow \textit{not } c_i(w', w'') \textit{ else,} \quad (7.1)
\end{aligned}$$

$$\begin{aligned}
&\textit{if } (m'=n') \textit{ and } (m''<n'') \textit{ then } ((w', w'') \in D_i)\\
&\qquad\qquad\qquad \Rightarrow \textit{not } c_j(w', (n''-m'') \downarrow w'')\\
&\qquad \textit{and } ((w', w'') \in D_j) \Rightarrow \textit{not } c_i(w', (m''-n'') \uparrow q_j'' w'') \textit{ else,} \quad (7.2)
\end{aligned}$$

$$\begin{aligned}
&\textit{if } (m'=n') \textit{ and } (m''>n'') \textit{ then } ((w', w'')\in D_i)\\
&\qquad \Rightarrow \textit{not } c_j(w', (n''-m'')\uparrow q_i'' w'')\\
&\qquad \textit{and } ((w', w'')\in D_j)\Rightarrow \textit{not } c_i(w', (m''-n'')\downarrow w'') \textit{ else}, \quad (7.3)\\
&\textit{if } (m'<n') \textit{ and } (m''=n'') \textit{ then } ((w', w'')\in D_i)\\
&\qquad \Rightarrow \textit{not } c_j((m'-n')\downarrow w', w'')\\
&\qquad \textit{and } ((w', w'')\in D_j)\Rightarrow \textit{not } c_i(w'(n'-m')\uparrow q_j', w'') \textit{ else}, \quad (7.4)\\
&\textit{if } (m'<n') \textit{ and } (m''<n'') \textit{ then } ((w', w'')\in D_i)\\
&\qquad \Rightarrow \textit{not } c_j((m'-n')\downarrow w',\\
&\qquad (n''-m'')\downarrow w'') \textit{ and } ((w', w'')\in D_j)\\
&\qquad \Rightarrow \textit{not } c_i(w'(n'-m')\uparrow q_j', (m''-n'')\uparrow q_j'' w'') \textit{ else}, \quad (7.5)\\
&\textit{if } (m'<n') \textit{ and } (m''>n'') \textit{ then } ((w', w'')\in D_i)\\
&\qquad \Rightarrow \textit{not } c_j((m'-n')\downarrow w',\\
&\qquad (n''-m'')\uparrow q_i'' w'') \textit{ and } ((w', w'')\in D_j)\\
&\qquad \Rightarrow \textit{not } c_j(w'(n'-m')\uparrow q_j', (m''-n'')\downarrow w'') \textit{ else}, \quad (7.6)\\
&\textit{if } (m'>n') \textit{ and } (m''=n'') \textit{ then } ((w', w'')\in D_i)\\
&\qquad \Rightarrow \textit{not } c_j(w'(m'-n')\uparrow q_i', w'')\\
&\qquad \textit{and } ((w', w'')\in D_j)\\
&\qquad \Rightarrow \textit{not } c_i((n'-m')\downarrow w', w'') \textit{ else}, \quad (7.7)\\
&\textit{if } (m'>n') \textit{ and } (m''<n'') \textit{ then } ((w', w'')\in D_i)\\
&\qquad \Rightarrow \textit{not } c_j(w'(m'-n')\uparrow q_i',\\
&\qquad (n''-m'')\downarrow w'') \textit{ and } ((w', w'')\in D_j)\\
&\qquad \Rightarrow \textit{not } c_i((n'-m')\downarrow w', (m''-n'')\uparrow q_j'' w'') \textit{ else}, \quad (7.8)\\
&\textit{if } (m'>n') \textit{ and } (m''>n'') \textit{ then } ((w', w'')\in D_i)\\
&\qquad \Rightarrow \textit{not } c_j(w'(m'-n')\uparrow q_i',\\
&\qquad (n''-m'')\uparrow q_i'' w'') \textit{ and } ((w', w'')\in D_j)\\
&\qquad \Rightarrow \textit{not } c_i((n'-m')\downarrow w', (m''-n'')\downarrow w''). \quad (7.9)
\end{aligned}$$

We remark that in every argument of conditions c_i and c_j where a subword of a r.h.m. q_j and q_i does occur, also class-names may occur; in this case, as usual, such class-names stand for any possible element of the referred class.

Let us spend few words in order to illustrate, for example, case (7.1) from which the other ones can be easily understood. Roughly speaking, by case (7.1), two conditions c_i, c_j are disjoint whenever the object word $w=w' p_i w''$ and $c_i(w', w'')$ holds, and (the implication arrow "$\Rightarrow$" imposing that) $c_j(w', w'')$ does not hold, and, vice versa, if $w=w' p_j w''$ and $c_j(w', w'')$ holds then $c_i(w', w'')$ does not hold; i. e. whenever the i-th rule is applicable the left and right context of p_i in the object word must be different from the left and right context of p_j when the j-th rule is applicable.

Finally, we can characterize disjoint rules by means of the following:

DEFINITION 8: Given two conditioned rules of a RCPA (p_i, q_i, c_i) and (p_j, q_j, c_j) as above in definition 7, the pairs (q_i, c_i) and (q_j, c_j) are called *disjoint patterns* ($(q_i, c_i)\,\mathrm{dp}\,(q_j, c_j)$) iff:

$$(8.1) \qquad q_i \,\mathrm{d}\, q_j \quad \textit{or} \quad c_i \,\mathrm{D}\, c_j.$$

Furthermore, every pair of rules (p_i, q_i, c_i), (p_j, q_j, c_j) such that $(q_i, c_i)\,\mathrm{dp}\,(q_j, c_j)$ are called *disjoint rules* ($(p_i, q_i, c_i)\,\mathrm{dr}\,(p_j, q_j, c_j)$).

Example 2: Let A, B, x, y and P be as in example 1.

(1) Let:

$$q_i=ab\,\alpha\, b,\ c_i=(|w'|=|w''|)=L_1 \quad \text{and} \quad q_j=ab\,\alpha\, x,$$

$$c_j=(|w'|<|w''|)=L_2,$$

then $(q_i, c_i)\,\mathrm{dp}\,(q_j, c_j)$ since q_i and q_j are not simple disjoint structures but they are of the same length (more precisely $|q_i'|=|q_j'|$ and $|q_i''|=|q_j''|$) and, by case (7.1),

$$((w', w'')\in D_i)\Rightarrow \textit{not}\ (|w'|<|w''|)$$

and:

$$((w', w'')\in D_j)\Rightarrow \textit{not}\ (|w'|=|w''|)$$

is always true.

(2) Let $q_i=b\,\alpha\, c$, $c_i=L_1$ and $q_j=ab\,\alpha\, c$, $c_j=L_2$, then (q_i, c_i) and (q_j, c_j) are not disjoint patterns, since, on account of case (7.4),

$$((w', w'')\in D_i)\Rightarrow \textit{not}\ (|-1\downarrow w'|<|w''|)$$

and

$$((w', w'')\in D_j)\Rightarrow \textit{not}\ (|w'a|=|w''|)$$

is not true.

On the other hand, if $c_i=(|w'|<|w''|)$ and $c_j=(|w'|>|w''|)$, it is easy to see that $(q_i, c_i)\,\mathrm{dp}\,(q_j, c_j)$.

(3) Let us consider, finally, $q_i = b\,\alpha\, c$, $c_i = L_1$ and $q_j = b\,\beta\, c$, $c_j = L_1$, then $(q_i, c_i)\,\mathrm{dp}\,(q_j, c_j)$ since, in spite of $c_i = c_j = L_1$, $q_i\,\mathrm{d}\,q_j$ holds so (8.1) is satisfied.

2.2. Invertible Pointer Algorithms

We are now in a position to give some interesting property of a particular class of RCPA's, namely the class of RCPA's the set of rules of which is composed by mutually disjoint rules, so let us start with the following:

DEFINITION 9: Any RCPA $R_\sigma = (I, O, X, B, C, \sigma, \omega, R)$ such that:

(9.1) for each rule in R, if a class-name, say x, does occur in its l. h. m., then it occurs at least once in its r. h. m. too (remember that whenever class-names like x_1, x_2 such that $\mu(x_1) = \mu(x_2)$ are used in a rule, they are considered different class-names for the same class); and

(9.2) if R is a m-tuple of conditioned rules, then $\forall\, i,\, j,\, i \neq j,\, 1 \leqq i,\, j \leqq m$, $(p_i, q_i, c_i)\,\mathrm{dr}\,(p_j, q_j, c_j)$,

is called an *Invertible Pointer Algorithm* (IPA). The reason of such name will be justified by the properties of this class of RCPA's we are going to present.

THEOREM 2: *Every IPA* $J_\sigma = (I, O, X, C, \sigma, \omega, J)$, *with* $J = ((p_1, q_1, c_1), (p_2, q_2, c_2), \ldots, (p_m, q_m, c_m))$, *represents a bijective function* $f : D \subseteqq I^* \to R \subseteqq O^*$.

The inverse function $f^{-1} : R \to D$, *is represented by the* inverse algorithm $J_\omega^{-1} = (O, I, X, C, \omega, \sigma, J^{-1})$, *where* ω *is the start pointer,* σ *the stopper pointer and:*

$$J^{-1} = ((q_1, p_1, c_1), (q_2, p_2, c_2), \ldots, (q_m, p_m, c_m)).$$

In other words, the following properties hold:

(2.1) *For any* w_i, w_j *with* $w_i \neq w_j$ *and* $w_i, w_j \in D$, *we have* $J_\sigma(w_i) \neq J_\sigma(w_j)$.

(2.2) *For any word* $w \in I^*$, *such that* $J(\sigma\,\Delta\, w\,\Omega) = \Delta\,\hat{w}\,\Omega\,\omega$, *we have* $J^{-1}(\Delta\,\hat{w}\,\Omega\,\omega) = \sigma\,\Delta\, w\,\Omega$.

Proof: First of all we shall prove that (2.1) holds, i. e. J_σ represents an injective function from D over R.

Suppose, by absurd, that given $w_i \neq w_j$ as in (2.1) we have $J_\sigma(w_i) = J_\sigma(w_j)$. This implies that two integers, say $k1$ and $k2$, must exist such that from $w_i^{k1} = w_i'^{k1}\, p_e\, w_i''^{k1}$ and $w_j^{k2} = w_j'^{k2}\, p_n\, w_j''^{k2}$, with $w_i^{k1} \neq w_j^{k2}$ representing the $k1$-th and $k2$-th label in the computation for w_i and w_j respectively, by means of the application of the e-th and n-th rule respectively, we would obtain:

$$w_i^{k1+1} = w_i'^{k1}\, q_e\, w_i''^{k1} = w_j'^{k2}\, q_n\, w_j''^{k2} = w_j^{k2+1}.$$

Then, the unique pointer present in the strings, on account of theorem 1 in [5] has to be in the same position in both words, and q_e and q_n have to be not simple disjoint structures. Moreover, according to the various possibilities we have for what concerns the lengths of q'_e, q''_e, q'_n, q''_n, where $q_e = q'_e \pi q''_e$ and $q_n = q'_n \pi q''_n$, we have to consider nine cases as in definition 7. Our discussion will refer only to two of these, the other ones being manageable in an analogous way. Suppose that $|q'_e| = e' = |q'_n| = n'$ and $|q''_e| = e'' = |q''_n| = n''$, then $w_i'^{k1} = w_j'^{k2}$ and $w_i''^{k1} = w_j''^{k2}$ which is impossible by the hypothesis of disjoint patterns. In fact, holding $not(q_e \mathrm{d} q_n)$, $c_e \mathrm{D} c_n$ has to be true and this implies that [for (7.1)]:

$$((w', w'') \in D_e) \Rightarrow not\ c_n(w', w'')\ and\ ((w', w'') \in D_n) \Rightarrow not\ c_e(w', w'')$$

has to be true; so the unique pair $(w_i'^{k1}, w_i''^{k1})$ cannot satisfy both c_e and c_n. Now, suppose that $e' = n'$ and $e'' < n''$, then $w_i'^{k1} = w_j'^{k2}$ and $w_i''^{k1} = (e'' - n'') \uparrow q''_n w_j''^{k2}$ which is impossible to happen, still by the hypothesis of disjoint patterns. In fact, on account of (7.2), the following has to be true:

$$((w', w'') \in D_e) \Rightarrow not\ c_n(w', (n'' - e'') \downarrow w'')$$

and:

$$((w', w'') \in D_n \Rightarrow not\ c_e(w', (e'' - n'') \uparrow q''_n w''),$$

which surely implies that $w_i'^{k1} \neq w_j'^{k2}$ or $w_i''^{k1} \neq (e'' - n'') \uparrow q''_n w_j''^{k2}$. Hence, (2.1) has been proved.

Let us now turn our attention to property (2.2). Since J_σ is an IPA and hence a RCPA too, the first and unique rule of J^{-1} applicable to any string $\Delta \hat{w} \Omega \omega$, with $\hat{w} \in D$, will be the inverse of the stopper rule in J, i. e. the last applied during the generation of $\Delta \hat{w} \Omega \omega$, so applying it we exactly get the word before the last step during the direct generation. This is possible because all the rules, on account of (9.1), are not class-names deleting. Now, on account of (9.2) all rules in J being disjoint and on account of theorem 1 [5] being unique the pointer present in the word, at every step the only applicable rule is just the reverse of that applied during the direct generation. Then, the first rule applied during the direct generation is the terminal one for J_ω^{-1}, and hence the result necessarily is $\sigma \Delta w \Omega$, so (2.2) has been proved too.

Example 3: The bijective function $J : N \to N$, defined by $J(n) = 1 + 2 + \ldots + n$, is implemented by the following IPA, assuming binary notation for integers:

$$J_\iota = (\{0, 1\}, \{0, 1\}, \{\iota, \alpha, \chi, \theta, \tau, \rho, \delta, \sigma, \varepsilon, \pi, \mu, \eta, \nabla, \Gamma, \Phi, \Delta, \Omega\}, \{c_{15}, c_{21}, c_{23}\}, \iota, \omega, J),$$

where the set of pointers:

$$P = \{\iota, \alpha, \chi, \theta, \tau, \rho, \delta, \sigma, \varepsilon, \pi, \mu, \eta\}$$

and:

$$c_{15}=(w'=\bar{w}'\,\nabla\,\bar{w}'); \qquad c_{21}=(w''=0\,\Omega \text{ or } w''=\Omega); \qquad c_{23}=(\nabla\in w'),$$

and J is:

c. initialization:

$$\iota\,\Delta\to\Delta\alpha \tag{1}$$

$$\alpha\,c\to c\,\alpha \tag{2}$$

$$\alpha\,\Omega\to\chi\,\Gamma\,0\,\Omega \tag{3}$$

c. the end of the algorithm:

$$\Delta\,0\,\chi\,\Gamma\to\Delta\,\theta \tag{4}$$

$$\chi\,\Gamma\to\Phi\,\chi\,\nabla\,\Gamma \tag{5}$$

$$c\,\Phi\,\chi\to\Phi\,c\,\tau\,c \tag{6}$$

$$\Delta\,\Phi\,\chi\to\Delta\,\delta \tag{7}$$

$$\theta\,c\to c\,\theta \tag{8}$$

$$\theta\,\Omega\to\Omega\,\omega \tag{9}$$

c. copy of n occurring in $\Delta\,n\,\Gamma$ in $\Delta\,n\,\nabla\,n\,\Gamma$:

$$c_3\,\tau\,c_1\,c_2\to c_3\,c_2\,\tau\,c_1 \tag{10}$$

$$\tau\,c\,\nabla\to\rho\,\nabla\,c \tag{11}$$

$$c\,\rho\to\rho\,c \tag{12}$$

$$\Phi\,\rho\,c\to\Phi\,\chi\,c \tag{13}$$

c. the copy is completed, we are going to successor applied to n_3, where $w=\Delta\,n_1\,\nabla\,n_1\,\Gamma\,n_3\,\Omega$:

$$\delta\,z\to z\,\delta \tag{14}$$

$$\delta\,\Gamma\to\Gamma\,\delta \qquad \text{if} \quad c_{15} \tag{15}$$

$$\delta\,\Omega\to\sigma\,\Omega \tag{16}$$

c. successor applied to n_3:

$$\Gamma\,0\,\sigma\,\Omega\to\Gamma\,\varepsilon\,1\,\Omega \tag{17}$$

$$1\,\sigma\to\sigma\,0 \tag{18}$$

$$\Gamma\,\sigma\,0\to\Gamma\,\varepsilon\,1\,0 \tag{19}$$

$$c\,0\,\sigma\,t \to c\,\varepsilon\,1\,t \tag{20}$$

$$c\,\varepsilon\,1 \to \varepsilon\,c\,1 \quad \textit{if} \quad c_{21} \tag{21}$$

$$c_1\,\varepsilon\,c_2 \to \varepsilon\,c_1\,c_2 \quad \textit{if not} \quad c_{21} \tag{22}$$

$$\Gamma\,\varepsilon \to \pi\,\Gamma \quad \textit{if} \quad c_{23} \tag{23}$$

c. predecessor applied to n_2 or to n_1 according to the present state of w:

$$d\,1\,\pi\,\Gamma \to d\,0\,\mu\,\Gamma \tag{24}$$

$$c\,1\,\pi \to c\,0\,\mu \tag{25}$$

$$0\,\pi \to \pi\,1 \tag{26}$$

$$d\,1\,\pi\,1 \to d\,\mu\,1 \tag{27}$$

$$\mu\,1 \to 1\,\mu \tag{28}$$

$$\mu\,\Gamma \to \eta\,\Gamma \quad \textit{if} \quad c_{23} \tag{29}$$

$$\mu\,\Gamma\,1 \to \chi\,\Gamma\,1 \quad \textit{if not} \quad c_{23} \tag{30}$$

c. the control is made whether $n_2=0$, in such a case predecessor is applied to n_1, otherwise to n_3:

$$\nabla\,0\,\eta\,\Gamma \to \pi\,\Gamma \quad \textit{if not} \quad c_{23} \tag{31}$$

$$\eta\,\Gamma \to \Gamma\,\delta \quad \textit{if not} \quad c_{15} \tag{32}$$

where:

$$\mu(c)=\mu(c_1)=\mu(c_2)=\mu(c_3)=\{0,\,1\}, \qquad \mu(z)=\{0,\,1,\,\nabla\},$$
$$\mu(t)=\{0,\,\Omega\}, \qquad \mu(d)=\{\Delta,\,\nabla\}.$$

In order to better understand the way of operating of such an IPA, a sort of flow-diagram is reported in the figure.

In such diagram the application of successor and predecessor function to an integer (in binary notation) n is denoted by $S(n)$ and $P(n)$ respectively.

Moreover, the arrow "$\to\to$" means that the word on its left side is transformed into the word present on its right side.

Let us see some step of the computation for $J_\iota(10)$:

$$\iota\,\Delta\,1\,0\,\Omega \underset{1}{\vdash} \Delta\,\alpha\,1\,0\,\Omega \underset{2}{\vdash} \ldots \underset{3}{\vdash} \Delta\,1\,0\,\chi\,\Gamma\,0\,\Omega \underset{5}{\vdash} \Delta\,1\,0\,\Phi\,\chi\,\nabla\,\Gamma\,0\,\Omega$$

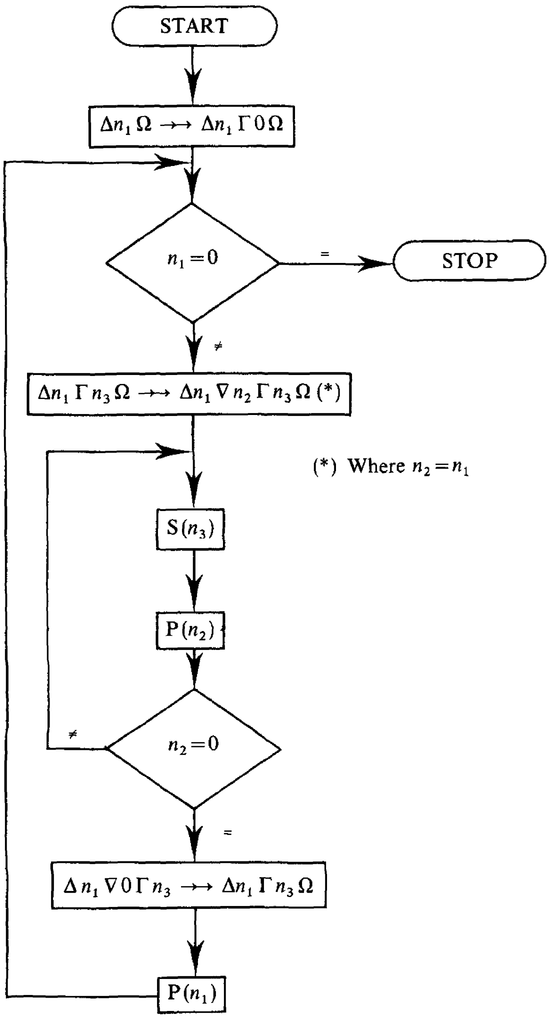
$$\underset{6}{\vdash} \Delta\,1\,\Phi\,0\,\tau\,0\,\nabla\,\Gamma\,0\,\Omega \underset{11}{\vdash} \Delta\,1\,\Phi\,0\,\rho\,\nabla\,0\,\Gamma\,0\,\Omega \underset{12}{\vdash} \ldots \underset{11}{\vdash} \Delta\,\Phi\,1\,0\,\rho\,\nabla\,1\,0\,\Gamma\,0\,\Omega$$

$$\underset{12}{\vdash} \ldots \underset{17}{\vdash} \Delta\,1\,0\,\nabla\,1\,0\,\Gamma\,\varepsilon\,1\,\Omega \underset{23}{\vdash} \ldots \underset{27}{\vdash} \Delta\,1\,0\,\nabla\,\mu\,1\,\Gamma\,1\,\Omega$$

$$\underset{28}{\vdash}\ldots\underset{29}{\vdash}\Delta 1 0 \nabla 0 \eta \Gamma 1 0 \Omega \underset{31}{\vdash}\ldots\underset{15}{\vdash}\Delta 1 \nabla 1 \Gamma \delta 1 0 \Omega$$

$$\underset{14}{\vdash}\ldots\underset{21}{\vdash}\Delta 1 \nabla 1 \Gamma \varepsilon 1 1 \Omega \underset{23}{\vdash}\ldots\underset{29}{\vdash}\Delta 1 \nabla 0 \eta \Gamma 1 1 \Omega$$

$$\underset{31}{\vdash}\ldots\underset{30}{\vdash}\Delta 0 \chi \Gamma 1 1 \Omega \underset{4}{\vdash}\Delta \theta 1 1 \Omega \underset{8}{\vdash}\ldots\underset{8}{\vdash}\Delta 1 1 \theta \Omega \underset{9}{\vdash}\Delta 1 1 \Omega \omega.$$



Flow-diagram of application of f

Vice versa, let us apply J_ω^{-1} to 11:

$$\Delta 1 1 \Omega \omega \underset{9}{\vdash} \Delta 1 1 \theta \Omega \underset{8}{\vdash} \ldots \underset{4}{\vdash} \Delta 0 \chi \Gamma 1 1 \Omega$$

$$\underset{30}{\vdash} \ldots \underset{31}{\vdash} \Delta 1 \nabla 0 \eta \Gamma 1 1 \Omega \underset{29}{\vdash} \ldots \vdash \ldots$$

$$\underset{14}{\vdash} \Delta \delta 1 \nabla 1 \Gamma 1 0 \Omega \vdash \ldots \underset{7}{\vdash} \iota \Delta 1 0 \Omega. \quad (\text{last step: } 1)$$

Thus, we exactly obtain the mirror image of the direct computation.

Let us now give some other useful properties of IPA's; namely, those referring to the various ways IPA algorithms can be composed, still obtaining an IPA algorithm.

THEOREM 3: *Given two* IPA's $I_\sigma(O_1, O_2, X_1, B_1, C_1, \sigma, \omega, I)$ *and* $J_\xi(I_1, O_1, X_2, B_2, C_2, \xi, \theta, J)$ *with* $P_1 \subset X_1$ *and* $P_2 \subset X_2$ *such that* $P_1 \cap P_2 = \emptyset$, $B_1 \cap B_2 = \emptyset$, *it is possible to define an IPA* K_ξ, *corresponding to their composition, i.e. such that:*

$$(3.1) \qquad \forall w \in I_1^*, \quad K_\xi(w) = I_\sigma(J_\xi(w)).$$

Proof: We define algorithm K_ξ corresponding to the usual meaning of function composition. So, let $K_\xi = (I_1, O_2, X_3, B_3, C_3, \xi, \omega, K)$ where $B_3 = B_1 \cup B_2 \cup \{z\}$, $X_3 = X_1 \cup X_2$, $C_3 = C_1 \cup C_2$ and K is the following $(m_1 + m_2 + 3)$-tuple of rules (I and J being a m_1-tuple and m_2-tuple of rules respectively):

$$K = (J, (q\Omega\theta, q\theta\Omega, c), (z\theta, \theta z, 1), (\Delta\theta, \sigma\Delta, 1), I),$$

where the terminal rule in J is $(p, q\Omega\theta, c)$ and z is the class-name such that $\mu(z) = O_1$.

K is an IPA since I and J both satisfy (9.1) and (9.2) and added rules also satisfy (9.1) and (9.2). Moreover, being P_1 disjoint from P_2, I and J are composed by all mutually disjoint rules and the added ones are also disjoint from any other since pointer θ only occurs in the r.h.m. of terminal rule in J but, in this case it is preceded by delimiter Ω, while in the r.h.m. of the added rules it is preceded by a character different from Ω. Pointer σ, being the start pointer for I_σ never occurs in any r.h.m. Furthermore, applying K_ξ to any word $w \in I_1^*$ such that $J_\xi(w)$ is defined, after the application of part J, we just get $J_\xi(w)$. At this point the only applicable rule is the (m_2+1)-th and then the (m_2+2)-th; after a suitable number of its applications, we surely find the (m_2+3)-th to be applicable; its application leads to the start of application of part I, thus exactly obtaining $I_\sigma(J_\xi(w))$, hence (3.1) holds.

Theorem 3 can be trivially extended to any finite deepness of IPA algorithm composition, so we can state:

COROLLARY 1: *The class of IPA algorithms is closed under the operation of algorithm composition.*

Proof: Trivial extension of theorem 3.

Another kind of IPA's composition is shown in the following:

THEOREM 4: *Given two IPA's:*

$$A_\alpha=(I_1, O_1, X_1, B_1, C_1, \alpha, \omega_1, A)$$

and:

$$B_\beta=(I_2, O_2, X_2, B_2, C_2, \beta, \omega_2, B),$$

with $P_1 \subset X_1$ *and* $P_2 \subset X_2$ *such that* $P_1 \cap P_2 = \emptyset$ *and* $B_1 \cap B_2 = \emptyset$ *with* Δ_1, Ω_1 *and* Δ_2, Ω_2 *left and right delimiters for input words* $w \in I_1^*$ *and* $v \in I_2^*$ *respectively, the IPA* C_α *can be defined such that, if* $\star \notin I_1 \cup I_2 \cup O_1 \cup O_2 \cup X_1 \cup X_2 \cup B_1 \cup B_2$*:*

(4.1) $\quad \forall w \in I_1^*, \quad v \in I_2^*, \qquad C_\alpha(w \star v)=A_\alpha(w) \star B_\beta(v).$

Proof: Let:

$$C=(I_1 \cup I_2 \cup \{\star, \Omega_1, \Delta_2\}, O_1 \cup O_2 \cup \{\star, \Omega_1, \Delta_2\}, (X_1-\{\omega_1\}) \cup X_2, B_1 \cup B_2, C_1 \cup C_2, \alpha, \omega_2, C)$$

with Δ_1 and Ω_2 as left and right delimiters and C is the following (A and B being a m_1-tuple and m_2-tuple of rules respectively) (m_1+m_2)-tuple of rules $C=(A', B)$.

A' is the m_1-tuple A where the stopper rule $(p\,\Omega_1, q\,\Omega_1\,\omega_1, c)$ is substituted by $(p\,\Omega_1 \star, q\,\Omega_1 \star \beta, c)$.

On account of disjoint pointer sets and A_α, B_β being IPA's, C_α is an IPA too. Moreover, property (4.1) is proved considering that whenever the ex-stopper rule in A' is applied, the word $A_\alpha(w) \star \beta\,\Delta_2\,v\,\Omega_2$ is obtained. At this point, the unique rule to be applied is just the first in B, so starting the computation for the word $\Delta_2\,v\,\Omega_2$. It is clear now that during this computation the only part of the string which can be transformed is $\Delta_2\,v\,\Omega_2$, so finally getting the word $A_\alpha(w) \star B_\beta(v)$.

One more property is given, reflecting the way a new IPA may be defined starting from a given one.

THEOREM 5: *Given an IPA,* $A_\alpha=(I, I, X, N, C, \alpha, \omega_1, A)$ *and a character* $a \in I$*, an IPA* B_β *can be defined such that:*

(5.1) $\quad \forall w \in I^*, \qquad B_\beta(w)=$ *if* $a \notin w$ *then* w *else* $B_\beta(A_\alpha(w))$, *provided the process terminates.*

Thus, B_β represents a recursive function and $B_\beta(w)$ is the first of the words:

$$w_0 = w, \qquad w_1 = A_\alpha(w_0), \qquad \ldots, \qquad w_i = A_\alpha(w_{i-1}), \; \ldots,$$

sych that "a" does not occur in it.

Proof: Let define:

$$B_\beta = (I,\, I,\, X \cup \{\beta, \sigma, \rho, \omega_2\},\, N \cup \{c, x\},\, C \cup \{c_1\},\, \beta,\, \omega_2,\, B),$$

where $c_1 = (w'' = 2 \downarrow w_0)$, B is the following (A being a m-tuple of rules) $(m+9)$-tuple:

$$\begin{aligned} B = ((\beta\,\Delta,\ \Delta\sigma,\ c_1),\ (\sigma\, c,\ c\,\sigma,\ 1),\ (\sigma\,\Omega,\ \Omega\,\omega_2,\ 1),& \\ (\sigma\, a,\ \rho\, a,\ 1),\ (c\,\rho,\ \rho\, c,\ 1),\ (\Delta\rho,\ \alpha\,\Delta,\ 1),& \\ A,\ (q\,\Omega\,\omega_1,\ q\,\omega_1\,\Omega,\ 1),\ (x\,\omega_1,\ \omega_1\, x,\ 1),\ (\Delta\omega_1,\ \Delta\sigma,\ not\, c_1)),& \end{aligned}$$

where c and x are class-names such that $\mu(c) = I - \{a\}$, $\mu(x) = I$; furthermore pointers β, σ, ρ, $\omega_2 \notin X$, and $(p\,\Omega,\ q\,\Omega\,\omega_1,\ t)$ is the stopper rule in A.

Algorithm B_β is an IPA since A_α is an IPA and, the new pointers not belonging to X, new rules are disjoint from those in A, satisfy (9.1) and are mutually disjoint each other (remark the disjunction between the first and last rule resulting by the presence of c_1 in the first and its negation in the last one). Equality (5.1) is proved considering that by means of the first six rules in B, the control is made onto the object word whether character a does or does not occur in it. If it does not occur, the computation stops by means of the third rule. Otherwise, by means of rules 4,5 and 6 we get the start of the application of A_α to the present word. After its execution, by means of the last three rules a jump to the second one is performed, so having a new test for the occurrence of character a into the result word.

In order to show the main property of IPA's, namely that for any bijective function, with recursive domain, it is possible to build up an equivalent IPA, let us give the following definition.

DEFINITION 10: Given any RCPA $J_\sigma = (I, O, X, B, C, \sigma, \omega, J)$, with $P \subset X$, $X' = X - P$, J a m-tuple of rules, let $S = I \cup O \cup X'$ and $K = S^* \times S^*$, for $1 \leqq i \leqq m$, we call $LR_i \subseteq K$, *Left and Right context of rule* i the set of all the string pairs, the first and second element of which are any possible left and right context of p_i (and hence of q_i in the i-th rule) in any label in the computation of $J_\sigma(w)$ for any word $w \in I^*$.

For what concerns sets LR_k the following nice property can be given.

LEMMA 1: *Given any RCPA J_σ as above, whose domain $D \subseteqq I^*$ is a recursive set, with Δ, $\Omega \in X$ left and right delimiters for every input word, for $1 \leqq k \leqq m$, the set LR_k is recursive and is defined by means of suitable context-free grammars G'_k, G''_k derived from J_σ.*

Proof: The proof is constructive. Let us associate to any k, $1 \leqq k \leqq m$, the set I_k composed by the order numbers, d, of any rule whose r. h. m., q_d, is not disjoint from the l. h. m. of rule k, p_k, i. e. such that *not* $(p_k \, \mathrm{d} q_d)$. This set constitutes the set of order numbers of admissible rules which could have been applied just before rule k is applied. For $k=1$, $I_1=\emptyset$. Let :

$$p_k = p'_k \pi p''_k \qquad \text{with} \quad |p'_k| = k', \quad |p''_k| = k''$$

and:

$$q_d = q'_d \pi q''_d \qquad \text{with} \quad |q'_d| = d', \quad |q''_d| = d'',$$

where π is the common pointer. Define, for each d in I_k, $1 \leqq k \leqq m$, the *derived left context* $w'_k(d)$ of p_k as:

$$\text{(L.1)} \quad w'_k(d) = \textit{if } 1 \uparrow p'_k p''_k = \Delta \textit{ then } \lambda \textit{ else if } k' = d' \textit{ then } w'_d \textit{ else if } k' > d' \textit{ then } (d'-k') \downarrow w'_d \textit{ else } w'_d (d'-k') \uparrow q_d,$$

where w'_d is the class-name (i. e. non terminal symbol) for the set of all possible left contexts of p_d, and operator " $\downarrow$ " is trivially extended to each elements of the set referred by w'_d, so $n \downarrow w'_d$ still represents the appropriate derived set. Analogously, define, for each d in I_k the *derived right context* $w''_k(d)$ of p_k as:

$$\text{(R.1)} \quad w''_k(d) = \textit{if } -1 \uparrow p'_k p''_k = \Omega \textit{ then } \lambda \textit{ else if } k'' = d'' \textit{ then } w''_d \textit{ else if } k'' > d'' \textit{ then } (k''-d'') \downarrow w''_d \textit{ else } (k''-d'') \uparrow q_d w''_d,$$

where w''_d is the class-name for the set of all possible right contexts of p_d. Strings $w'_k(d)$ and $w''_k(d)$ are defined having in mind the semantics of PA's. In fact, rule k is applicable if the pointer present in its l. h. m. and its left and right contexts are those occurring in the first label of the computation or have been generated by the application of a certain rule, whose r. h. m. must be not disjoint from p_k, otherwise rule k would not be applicable; so we got set I_k. Moreover, the left and right contexts of p_d, when rule d has been applied, did not change when rule k is going to be applied. Then, the four cases expressed into (L.1) and (R.1) reflect the actual state of both $w'_k(d)$ and $w''_k(d)$: in fact, the first case being trivial, if $k'=d'$, then $w'_k(d)$ exactly is the left context of p_d, i.e. w'_d; if $k'>d'$, we are probably describing in p_k the rightmost part of the left context when rule d was applied, so $w'_k(d)$ is w'_d leaving its last $(k'-d')$ elements; the fourth case, $k'<d'$, means that in p_k we are not describing those rightmost characters which do exist in its left context, so they are to be appended to w'_d in order to get the exact $w'_k(d)$. Similar considerations can be carried on for what concerns (R.1).

Now, the probable left and right context for p_k, for any k, can be defined as:

$$\text{(L.2)}\quad \begin{cases} w'_1=\lambda \qquad \text{and} \qquad I_1=\emptyset, \\ \text{by definition and} \\ w'_k ::= w'_k(d1)|\ldots|w'_k(dn), \\ \text{where } I_k=\{d1, d2, \ldots, dn\},\ 1<k\leqq m, \end{cases}$$

and :

$$\text{(R.2)}\quad \begin{cases} w''_1=|p_1|\downarrow w, \\ w \text{ being the class-name for } D, \text{ recursive domain of } J_\sigma, \text{ by definition and} \\ w''_k ::= w''_k(d1)|\ldots|w''_k(dn), \qquad 1<k\leqq m. \end{cases}$$

Now, we have to check for each k, $1<k\leqq m$, whether set I_k has been well established, by means of controlling for each d in I_k whether p_k is or is not disjoint from $w'_d q_d w''_d$, where all alternatives for w'_d and w''_d have to be taken as ordered pairs $(w'_d(i), w''_d(i))$. Remember, in fact, that p'_k or p''_k could be longer than the corresponding q'_d and q''_d; hence, even if $not\,(p_k \mathrm{d} q_d)$ is true, the necessity to check whether rule k is actually applicable after rule d has been applied. Then, for each d in I_k, such that $(p_k \mathrm{d} w'_d q_d w''_d)$ is true, discard d from I_k and hence, delete $w'_k(d)$ and $w''_k(d)$ from (L.2) and (R.2) respectively.

We remark that if some set I_k, $k>1$, after this deletion, results empty, rule k results never applicable and hence, can be deleted from the algorithm. After this control, and possible deletions, have been performed for every k, the set LR_k can be defined as the set of pairs:

$$\text{(LR.1)}\quad \begin{cases} LR_1=(\lambda, w''_1) \quad \text{and for } 1<k\leqq m, \\ LR_k=\{(w'_k(d1), w''_k(d1)), (w'_k(d2), w''_k(d2)), \ldots, (w'_k(dn), w''_k(dn))\}. \end{cases}$$

Thus, for $1<k\leqq m$, $G'_k=(A, N', P', w'_k)$, where $A=I\cup O\cup X'\cup B$, N' is the set of class-names for left contexts of any rule, P' is the set of involved productions starting from (L.2) and w'_k is the distinct symbol, is the suitable context-free grammar defining any possible left context of p_k.

Analogously, $G''_k=(A, N'', P'', w''_k)$ is the context-free grammar defining any possible string belonging to w''_k, where w''_k is the distinct symbol, A is as before, N'' is the set of class-names for right contexts of any rule, P'' is the set of involved productions starting from (R.2). We remark also, that any element in B occurring in any production is still interpreted as the occurrence of any one element of the class of characters it refers to.

Note that if rule n is a terminal one, then the set $\{w' q_n w''\}$ with $(w', w'') \in LR_n$, is just the set of resulting words when the terminal rule n has been applied.

Thus, $U_t \{w' q_t w''\}$, where t ranges over the order number of every terminal rule of a given RCPA is just the range of the algorithm.

For an example referring to the above lemma 1, *see* next example 4.

We can now state the announced equivalence between the class of bijective functions over recursive domains and the class of IPA's over recursive domains, which follows from theorem 5 and the following:

THEOREM 6: *For any given bijective function $f : D \to R$, with recursive domain $D \subseteqq I^*$ and hence range $D \subseteqq O^*$, there always exists an equivalent IPA F_σ implementing it.*

Proof: We shall prove this theorem by defining the suitable F_σ.

Let $F'_\sigma = (I, O, X', B', \{1\}, \sigma, \omega, F')$ be the RCPA, where each condition appended to each rule in F' is the constant predicate 1, implementing the given function f. Note that F' is simply derived, by means of theorem 1, from the existing, on account of the main thesis of computability theory, NMA implementing function f.

Two cases are now to be considered:

(*a*) F'_σ is an IPA, i.e. F' satisfy both conditions (9.1) and (9.2), then F'_σ is the requested algorithm and the proof is trivially complete, $F_\sigma = F'_\sigma$;

(*b*) F' does not satisfy (9.1) or (9.2) or both. Let us construct the desired IPA in this case too.

First of all, rewrite any rule in F', for which (9.1) does not hold, into its corresponding elementary rules; i.e. any rule of type $(p' c p'', q)$, with class-name $c \notin q$ and $\mu(c) = \{c_1, c_2, \ldots, c_n\}$, is substituted by the n rules $(p' c_1 p'', q)$, $(p' c_2 p'', q)$, $\ldots$, $(p' c_n p'', q)$; thus, repeating the above procedure as many times as necessary, we get a set of rules, equivalent to the original one, satisfying condition (9.1). Let us still call F'_σ this RCPA with such possible expanded set of rules and accordingly decreased set B'. Now, since $\forall w_i, w_j \in D$, $w_i \neq w_j$, $F'(w_i) \neq F'(w_j)$, f being bijective, the computations for w_i and w_j never have common labels; namely let:

$w_i^k = w_i'^k p_e w_i''^k$ the k-th label in the computation for w_i, and, analogously;

$w_j^m = w_j'^m p_n w_j''^m$ the m-th label in the computation for w_j, with k, $m \geqq 1$ and $1 \leqq e, n \leqq t$ (F' being a t-tuple of rules). Then, $w_i^k \neq w_j^m$ and still $w_i^{k+1} = w_i'^k q_e w_i''^k \neq w_j'^m q_n w_j''^m = w_j^{m+1}$. This is true for all k-th and m-th label as well as for all e-th and n-th rule.

Let us consider the following cases:

(i) $e = n$, then the following predicate holds:

$$\text{(i.1)} \qquad (w_i'^k \neq w_j'^m \text{ or } w_i''^k \neq w_j''^m);$$

(ii) $e \neq n$ and $(q_e \, \mathrm{d} q_n)$ holds, then inequality of the two labels at least follows from disjunction of q_e and q_n;

(iii) $e \neq n$ and *not* $(q_e \, \mathrm{d} q_n)$ holds, then, being $q_e = q'_e \pi q''_e$ and $q_n = q'_n \pi q''_n$, we have nine different subcases (recall definition 7) to consider, according to the various combinations over the lengths of q'_e, q''_e, q'_n and q''_n. Let us consider only two of such cases, advising that the other seven can be treated in an analogous way. Let $e' = |q'_e|$, $e'' = |q''_e|$, $n' = |q'_n|$ and $n'' = |q''_n|$, then it may be:

(*a*) $e' = n'$ and $e'' = n''$, it follows that (i.1) still holds, or

(*b*) $e' = n'$ and $e'' < n''$, it follows that:

$$\text{(iii.1)} \qquad (w_i'^k \neq w_j'^m \text{ or } (e'' - n'') \uparrow q''_n w_j''^m \neq w_i''^k) \text{ holds.}$$

And so on.

The only point we are interested with is point (iii). In this case, in fact, we do not have the desired disjunction, but it can be still obtained by means of the following arguments. If two rules fall in case (*a*) of (iii) then (i.1) holds, that means that if $(w', w'') \in LR_e$ (*see* lemma 1) never can happen that $(w', w'') \in LR_n$ and vice versa, i.e. $LR_e \cap LR_n = \emptyset$. Then:

$$((w', w'') \in LR_e) \Rightarrow (w', w'') \notin LR_n \qquad \textit{and} \qquad ((w', w'') \in LR_n) \Rightarrow (w', w'') \notin LR_e$$

holds, which is just the disjunction condition for predicates $c_e = (w', w'') \in LR_e$ and $c_n = (w', w'') \in LR_n$, when $e' = n'$ and $e'' = n''$, as given in (7.1).

Furthermore, when two rules fall in case (*b*) of point (iii), (iii.1) holds. This means that whenever a pair $(w', w'') \in LR_e$ then the pair $(w', (n'' - e'') \downarrow w'') \notin LR_n$, and, vice versa, if $(w', w'') \in LR_n$ then $(w', (e'' - n'') \uparrow q''_n w'') \notin LR_e$. Hence, if c_e and c_n are as above, we have:

$$((w', w'') \in LR_e) \Rightarrow \textit{not } c_n(w', (n'' - e'') \uparrow w'')$$

and

$$((w', w'') \in LR_n \Rightarrow \textit{not } c_e(w', (e'' - n'') \downarrow q''_n w''),$$

which is just the disjunction condition for predicates c_e, c_n when $e' = n'$ and $e'' < n''$ as given in (7.2).

So, in both cases (*a*) and (*b*), disjoint rules are obtained by appending c_e and c_n as condition part to rules e and n respectively.

Thus, carrying on such procedure for every k_i-tuple of non disjoint rules in F' a set F of disjoint rules (i. e. with disjoint patterns) is obtained. Finally, the desired IPA $F_\sigma=(I, O, X, B, C, \sigma, \omega, F)$ is obtained where: $B=B'$, $X=X'$, $C=\{c_{i1}, c_{i2}, \ldots, c_{ir}\}$, with each c_{ij} found as above described, $r=k_1 \times k_2 \times \ldots \times k_s$, s being the number of k_i-tuple of non disjoint rules in F', and F is derived from F' by inserting the appropriate conditions into the interested rules, so satisfying both (9.1) and (9.2).

An example of application of both lemma 1 and theorem 6 is in order.

Example 4: Consider the successor function for binary numbers equal or greater than zero. Its domain is clearly recursive and is defined by the following context-free grammar $W=(\{0, 1\}, \{c, n, w\}, Q, w)$ with production set Q composed by:

(Q.1) $c ::= 0 \mid 1$

(Q.2) $n ::= 1 \mid nc$

(Q.3) $w ::= 0 \mid n$

A possible *PA* implementing successor function is the following: $S=(\{0, 1\}, \{0, 1\}, \{\alpha, \delta, \varepsilon, \sigma, \Delta, \Omega\}, \sigma, S)$ where $P=\{\alpha, \delta, \varepsilon, \sigma\}$ and S is:

$$\sigma \Delta c \to \Delta \alpha c \tag{1}$$

$$\alpha c \to c \alpha \tag{2}$$

$$\alpha \Omega \to \delta \Omega \tag{3}$$

$$\Delta 0 \delta \Omega \to \Delta 1 \varepsilon \Omega \tag{4}$$

$$c 0 \delta \to c 1 \varepsilon \tag{5}$$

$$1 \delta \to \delta 0 \tag{6}$$

$$\Delta \delta \to \Delta 1 \varepsilon \tag{7}$$

$$\varepsilon 0 \to 0 \varepsilon \tag{8}$$

$$\varepsilon \Omega \to \Omega \qquad \text{with} \quad \mu(c)=\{0, 1\}. \tag{9}$$

We point out that at a first glance rules (4) and (5) seem to be replacable by the unique rule $0\,\delta \to 1\,\varepsilon$; if one try to do such substitution, the resulting algorithm is no more bijective. It is, in fact, easily seen that from initial strings $v=\sigma\,\Delta\,1\,\Omega$ and $w=\sigma\,\Delta\,0\,1\,\Omega$, with $v \neq w$, we would get the same result $\Delta\,1\,0\,\Omega$ for both.

This algorithm is trivially put in the form of a RCPA, by substituting rule (9) by (9′) $\varepsilon\,\Omega \to \Omega\,\omega$, and setting $C=\{1\}$, and $X=X \cup \{\omega\}$, $B=\{c\}$. Such

RCPA yet satisfies (9.1), but the set S does not satisfies (9.2) because of rules (4) and (7), the r.h.m.'s of which are not simple disjoint structures.

In order to find the appropriate conditions c_4 and c_7, to get disjoint patterns and hence disjoint set of rules, let us follow theorem 6.

Such a theorem tells us that appropriate conditions are $c_4=(w', w'')\in LR_4$ and $c_7=(w', w'')\in LR_7$. Then, by means of lemma 1, let us define LR_4 and LR_7, where LR_4 is defined by the context-free grammars with axioms w'_4 and w''_4, and LR_7 is defined by the context-free grammars with axioms w'_7 and w''_7 which are defined below.

For $k=1$, $I_1=\emptyset$ and $w'_1 ::= \lambda$, $w''_1 ::= 3\downarrow w_0$ where the input word $w_0=\sigma\,\Delta\,w\,\Omega$, w being defined by grammar W.

For $k=2$, $I_2=\{1, 2\}$ so that:

$$w'_2(1)=w'_1\,\Delta=\Delta \quad \text{and} \quad w''_2(1)=w''_1=3\downarrow w_0,$$
$$w'_2(2)=w'_2\,c \quad \text{and} \quad w''_2(2)=1\downarrow w''_2,$$

then:

$$w'_2 ::= \Delta\,|\,w'_2\,c \quad \text{and} \quad w''_2 ::= w''_1\,|\,1\downarrow w''_2.$$

For $k=3$, $I_3=\{2\}$ so that:

$$w'_3(2)=w'_2\,c \quad \text{and} \quad w''_3(2)=\lambda,$$

then:

$$w'_3 ::= w'_2\,c \quad \text{and} \quad w''_3 ::= \lambda.$$

For $k=4$, $I_4=\{3\}$ so that:

$$w'_4(3)=\lambda \quad \text{and} \quad w''_4(3)=\lambda,$$

then:

$$w'_4 ::= \lambda \quad \text{and} \quad w''_4 ::= \lambda.$$

For $k=5$, $I_5=\{3, 6\}$ so that:

$$w'_5(3)=-2\downarrow w'_3 \quad \text{and} \quad w''_5(3)=\Omega,$$
$$w'_5(6)=-2\downarrow w'_6 \quad \text{and} \quad w''_5(6)=0\,w''_6,$$

then:

$$w'_5 ::= -2\downarrow w'_3\,|\,-2\downarrow w'_6; \qquad w''_5 ::= \Omega\,|\,0\,w''_6.$$

For $k=6$, $I_6=\{3, 6\}$ so that:

$$w'_6(3)=-1\downarrow w'_3; \qquad w''_6(3)=\Omega,$$
$$w'_6(6)=-1\downarrow w'_6; \qquad w''_6(6)=0\,w''_6,$$

then:

$$w'_6 ::= -1 \downarrow w'_3 \mid -1 \downarrow w'_6; \qquad w''_6 ::= \Omega \mid 0\, w''_6.$$

For $k=7$, $I_7 = \{3, 6\}$ so that:

$$w'_7(3)=\lambda \quad \text{and} \quad w''_7(3)=\Omega,$$
$$w'_7(6)=\lambda \quad \text{and} \quad w''_7(6)=0\, w''_6,$$

then:

$$w'_7 ::= \lambda \quad \text{and} \quad w''_7 ::= \Omega \mid 0\, w''_6.$$

It suffices now to check if the found definitions for the left and right contexts are correct; we see that $3 \in I_7$ and the control whether *not* $(w'_3\, q_3\, w''_3\, \mathrm{d} p_7)$ is true or not, results in controlling whether *not* $(w'_2\, c\, \delta\Omega\, \mathrm{d}\, \Delta\delta)$ is true or not, and this is clearly false. Then, 3 must be discarded from I_7 and, correspondently, $w'_7(3)$ and $w''_7(3)$ have to be deleted from definitions of w'_7 and w''_7, maintaining $w'_7(6)$ and $w''_7(6)$,

$$I_7 = \{6\} \quad \text{and} \quad w'_7 ::= \lambda; \quad w''_7 ::= 0\, w''_6.$$

Thus, we have $LR_4 = \{(\lambda, \lambda)\}$ and $LR_7 = \{(\lambda, 0\, w''_6)\}$; so $c_4 = (w', w'') \in LR_4$ and $c_7 = (w', w'') \in LR_7$. Equivalent conditions could be $c_4 = (w'' = \lambda)$ and $c_7 = (w'' \neq \lambda)$.

We remark that, in this case, disjunction can be also obtained by modifying rule (7) as (7') $\Delta\delta 0 \to \Delta 1 \varepsilon 0$, without using explicit condition, it is easily seen that $(q_4\, \mathrm{d} q'_7)$. In any case, by appending c_4 and c_7 or transforming rule (7), the new set of rules S satisfies (9.1) and (9.2) so the whole algorithm is an IPA.

Thus, as we have before seen, condition holding for w' and w'' when rule (7) is going to be applied can be merged into the l. h. m., and hence into the r. h. m. too, since this condition can be expressed by means of the string structure of a finite subword, namely the l. h. m. of the rule. If you consider, instead, algorithm J_ι of example 3, it could be seen that condition given for rule (15) or (29) cannot be expressed by means of the occurrence of a given subword of finite and fixed length into the object string, i. e. it cannot be expressed only by the l. h. m. of the rule. It so requires to be explicitely stated and, eventually, implemented by means of a suitable PA as it has been shown in the complete proof of theorem 1 in [1].

4. CONCLUDING REMARK

The IPA class defined in this paper along with its outlined properties, seems to be promising both in mathematics and in computer science. Note that for PA's a method of compilation has been studied and implemented [13] so getting an IPA actually executable by machine: it suffices, in fact, by means of theorem 1, to construct its equivalent PA and it is obviously possible to operate in the same way to execute its, directly defined, inverse algorithm.

Moreover, the concept of IPA's extended to the APS system [5], is under our investigation. If such result will be fully reached, a very powerful both theoretic and practical device will be available, especially in the area of applications sketched in the introduction.

ACKNOWLEDGEMENTS

My thanks go to the referee who read the first copy of this paper with great care and suggested many improvements in the presentation of the final version.

REFERENCES

1. G. AGUZZI, *The Theory of Invertible Algorithms*, Sem. Ist. Mat. Appl. Fac. Ing., Firenze, 1980.
2. G. AGUZZI, F. CESARINI, R. PINZANI, G. SODA and R. SPRUGNOLI, *Towards an Automatic Generation of Interpreters*, in Lecture Notes in Computer Science, vol. 1, 1973, pp. 94-103, Springer-Verlag, Berlin.
3. G. AGUZZI, F. CESARINI, R. PINZANI, G. SODA and R. SPRUGNOLI, *An APL Implementation of an Interpreter Writing System*, in APL Congress 73, 1973, pp. 9-15, North-Holland Pub. Co., Amsterdam.
4. G. AGUZZI, F. CESARINI, R. PINZANI, G. SODA and R. SPRUGNOLI, *Tree Structures Handling by APS*, in Lecture Notes in Computer Science, Vol. 19, 1974, pp. 120-129, Springer-Verlag, Berlin.
5. G. AGUZZI, R. PINZANI and R. SPRUGNOLI, *An Algorithmic Approach to the Semantics of Programming Languages*, in Automata, Languages and Programming, M. NIVAT, Ed., 1973, pp. 147-166, North-Holland Pub. Co., Amsterdam.
6. A. CARACCIOLO DI FORINO, *Generalized Markov Algorithms and Automata*, in Automata Theory, CAIANIELLO, Ed., Academic Press, New York.
7. A. CARACCIOLO DI FORINO, L. SPANEDDA and N. WOLKENSTEIN, *Panon 1B: a Programming Language for Symbol Manipulation*, Calcolo, 1966, pp. 245-255.
8. V. S. CERNIAVSKII, *On a Class of Normal Markov Algorithms*, A.M.S. Translations, 2, Vol. 48, 1965, pp. 1-35.
9. D. J. FARBER, R. E. GRISWOLD and I. P. POLONSKY, *Snobol, a String Manipulation Language*, J. Assoc. Comp. Mach., vol. 11, 1964, pp. 21-30.
10. B. A. GALLER and A. J. PERLIS, *A view of Programming Languages*, Addison Wesley, 1970.
11. J. Katznelson, *The Markov Algorithm as Language Parser: Linear Bounds*, J. Comp. System Sc., Vol. 6, 1972, pp. 465-478.
12. M. R. LAGANÀ, G. LEONI, R. PINZANI and R. SPRUGNOLI, *Improvements in the Execution of Markov Algorithms*, Boll. Un. Mat. Italiana, Vol. 11, (4), 1975, pp. 473-489.

13. G. LEONI and R. SPRUGNOLI, *The Compilation of Pointer Markov Algorithms*, in International Computing Symposium 1975, E. GELENBE and D. POTIER, Eds., 1975, pp. 129-135, North-Holland Pub. Co., Amsterdam.

14. A. A. MARKOV, *The Theory of Algorithms, Israel Program for Scientific Translations*, Jerusalem, 1962.

15. M. PAGET, *Propriétés de Complexité pour une Famille d'Algorithmes de Markov*, R.A.I.R.O. informatique théorique, Vol. 12, (1), 1978, pp. 15-32.