MATHEMATICA — REVUE D'ANALYSE NUMÉRIQUE
ET DE THÉORIE DE L'APPROXIMATION

L'ANALYSE NUMÉRIQUE ET LA THÉORIE DE L'APPROXIMATION
Tome 4, N° 2, 1975, pp. 179 — 200

SOME OBSERVATIONS CONCERNING THE ALGEBRAIC TREATING OF THE FORMAL LANGUAGES

by
TEODOR RUS
(Cluj — Napoca)

The necessity of a formal device suitable for solving the problems of relations between programming languages and between them and computing systems, becomes very urgent. The paper is an attempt on this way. Having in view on one side the form of word free structures of the programming languages, generated by their data base structures, primitive operations and control sequences, using some syntactical rules, given by operator schemes, similar to te operations in a word free algebra, and on the other side the observations, the operations schemes in this case are concerning the heterogeneous operations, in the paper we have the following purposes:

1. Organize a formal language in the form of heterogeneous word free algebra, on levels.
2. A formal definition of the semantic domain associated to a formal language.
3. To give a way of formal representation of such a structure in an other one, which generalises the notion of homomorphism, for a formal definition of semantic preserving translation algorithms.

The solutions of the above problems are presented in sections 2,3. Section 1 is devoted to the definition of the formal language by means of X-categories, used in sequel.

1. A CATEGORIAL CHARACTERIZATION OF THE FORMAL LANGUAGES

1.1. General considerations on the rewriting system

The pair (Σ, P) is a rewriting system (semi-Thue system) when Σ is an alphabet and P is a set of productions. (Σ, P) is an indexed rewriting system when P is an injection from an initial segment of natural

numbers to $\Sigma^* \times \Sigma^*$. In this way every production $\alpha \to \beta \in P$ is distinguishable by its index, so that $P(n) = \alpha \to \beta$ can be written as $\alpha \overset{n}{\to} \beta$. The relation $\to$ can be extended to $\Rightarrow$ by concatenation in the following way:

If $\theta, \psi \in \Sigma^*$, $\theta = \nu\alpha\mu$, $\psi = \nu\beta\mu$ and $\alpha \to \beta \in P$ then $\theta \Rightarrow \psi$ Let us consider the transitive reflexive closure of $\Rightarrow$ and denote it by $\overset{*}{\Rightarrow}$.

For $V_N \subseteq \Sigma^*$, $V_T \subset \Sigma$, the quadruple (Σ^*, V_N, V_T, P) will be called grammar and the language generated by $G = (\Sigma^*, V_N, V_T, P)$ is defined as follows:

$$L(G) = \{w \in V_T^* \mid \exists\, \sigma \in V_N,\ \sigma \overset{*}{\Rightarrow} w\}$$

The extensions of the relations defined by P from $\to$ *to* $\Rightarrow$ and *to* $\overset{*}{\Rightarrow}$ define a category. Let us denote this category by F.

The objects O of F are the elements of Σ^*. The set M of morphisms of F is defined by:

(i). The length zero derivations, $\alpha \overset{*}{\Rightarrow} \alpha$, $\alpha \in \Sigma^*$ are identities in M, denoted by 1_α.

(ii). If $p \in P$ then p is considered as a morphism of length 1 and belongs to M. So, $P \subseteq M$. When P is an indexed set, then $\langle n, \alpha \overset{n}{\to} \beta\rangle \in P$ and $\alpha \overset{n}{\to} \beta \in M$. The domain of $\alpha \overset{n}{\to} \beta$ will be α and the codomain of $\alpha \overset{n}{\to} \beta$ will be β.

(iii) For every pair of identities in M, 1_μ, 1_ν and every morphism $\alpha \overset{n}{\to} \beta \in M$ the morphism $1_\mu * (\alpha \overset{n}{\to} \beta) * 1_\nu \in M$ and is determined uniquely by μ, ν and $\alpha \overset{n}{\to} \beta$, where $*$ is the concatenation of morphisms.

(iv). For each $x, y \in M$ such that codomain of x belongs to the domain of y is defined the composition $y \circ x$, and $y \circ x \in M$.

From this definition it follows that each morphism in M is equivalent to a derivation in (Σ, P) from its domain to its codomain. Then, for two objects $\alpha, \beta \in \Sigma^*$, Hom (α, β) is the set of all derivations from α to β, or the set of morphisms from α to β.

Following HOTZ [1], BENSON [3], GRIFFITHS [2], on the set of morphisms from α to β can be defined a congruence relation $\sim$ and for studying the syntactical structure defined by (Σ, P) in Σ^* the category $F/_\sim = (\Sigma^*, M/_\sim)$ can be considered. The category $F/_\sim$ is an X-category as it was defined by HOTZ [1]. From this reason and because X-categories will be used in the sequel we shall redefine it.

1.2. The notion of X-category

The system $X = (O, M, Q, Z, \circ, *)$ is a X-category if the following properties hold:

(i). O and M are sets, Q and Z are functions, $Q, Z: M \to O$; Q will be called domain function and Z will be called codomain function.

(ii). $\circ$ is a partial operation on M, $\circ: M^2 \to M$, called composition, and is defined in the following way: if $(x, y) \in M^2$ then $y \circ x$ is defined iff $Q(y) = Z(x)$ and the following equalities take place:

$$Q(y \circ x) = Q(x),\ Z(y \circ x) = Z(y)$$

(iii). $\circ$ is associative where it is defined.

(iv). For each object $\alpha \in O$ there exists a uniquely determined $1_\alpha \in M$ called identity on α, so that for every $y, x \in M$ for which $x \circ 1_\alpha$ and $1_\alpha \circ y$ are defined and $Z(1_\alpha) = Q(x)$, $Q(1_\alpha) = Z(y)$ then $x \circ 1_\alpha = x$ and $1_\alpha \circ y = y$.

(v). The algebraic systems $(O, *, \Lambda)$, $(M, *, 1_\Lambda)$ are monoids, $\Lambda \in \Sigma^*$ is empty string in Σ^* and $Q, Z: (M, *, 1_\Lambda) \to (O, *, \Lambda)$ are monoid homomorphisms.

(vi). For every $x_1, x_2, y_1, y_2 \in M$ so that $Q(y_i) = Z(x_i)$, $i = 1,2$ the following identity takes place:

$$(y_1 * y_2) \circ (x_1 * x_2) = (y_1 \circ x_1) * (y_2 \circ x_2)$$

(vii). For each $1_\mu, 1_\nu \in M$, $1_\mu * 1_\nu = 1_{\mu * \nu}$

Now, given a rewriting saystem (Σ, P) the corresponding X-category is denoted by $D = (\Sigma^*, M, Q, Z, \circ, *)$ where $(\Sigma^*, *, \Lambda)$ is the free monoid generated by Σ and $(M, *, 1_\Lambda)$ is the monoid of derivations. Using the notation $Hom_D\ (\alpha, \beta)$ for the set of all morphisms from α to β in the category D we have:

1. $\alpha \overset{*}{\Rightarrow} \beta$ is equivalent to $Hom_D(\alpha, \beta) \neq \emptyset$
2. $V_N \subseteq \Sigma^*$, $V_T \subset \Sigma$, $G = (\Sigma^*, V_N, V_T, P)$ then the language generated by G is $L(G) = \{w \in V_T^* \mid \exists\, x \in M,\ Q(x) \in V_N,\ Z(x) = w\}$
3. If $\beta \in \Sigma^*$ then the syntactical structures of β are $\bigcup_{\sigma \in V_N} Hom_D(\sigma, \beta)$
4. The set of all syntactical structures of the language is

$$\mathcal{S}(\Sigma^*, V_N, V_T, P) = \bigcup_{w \in V_T^*} \bigcup_{\sigma \in V_N} Hom_D(\sigma, w)$$

For the study of some relations between languages the notion of X-functor is wanted. In order to define that, let $X_j = (O_j, M_j, Z_j, \circ, *)$ $j = 1,2$ be two X-categories. A pair of functions $H = (h_0, h_M)$ will be called X-functor if the following properties take place:

(i). $h_O: (O_1, *, \wedge) \to (O_2, *, \wedge)$ and

(ii). $h_M: (M_1, *, 1_\Delta) \to (M_2, *, 1_\Delta)$

are monoid homomorphisms.

(iii). $H:(O_1, M_1, Q, Z, \circ) \to (O_2, M_2, Q_2, Z_2, \circ)$ is a functor, that is

(iii$_1$). If $\alpha \in O_1$ then $h_M(1_\alpha) = 1_{h_O(\alpha)}$

(iii$_2$). For each $y, x \in M$ so that $y \circ x$ is defined $h_M(y \circ x) = h_M(y) \circ h_M(x)$

(iii$_3$). The following diagrams commute

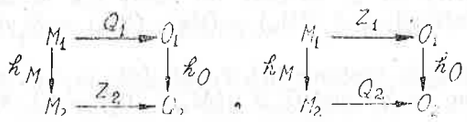

Fig. *A*

The pair $H = (h_M, h_O)$ is a cofunctor if instead of (iii$_2$) and (iii$_3$) we consider

(iii$'_2$). $h_M(y \circ x) = h_M(x) \circ h_M(y)$, for every $x, y \in M$ for which $y \circ x$ is defined.

(iii$'_3$). The following diagrams comute

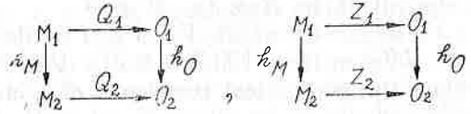

Fig. *B*

1.3. The notion of interpretation of a language

For semantic definition of a language we shall consider the notion of interpretation [3] of the language in some sets and functions between these sets. For this purpose the strings which belong to a language will be interpreted as cartesian products of some sets, which will be associated with each symbol of the alphabet Σ. Derivations will be interpreted as functions defined on the cartesian product of the corresponding sets. Let $\mathcal{S}et = (U, F, Q, Z, \circ, \times)$ be a X-category, where U is an univers (COHN [4]), F is a collection of functions defined on the membres of U, $\circ$ is composition operation of functions and $\times$ is the associative cartesian product of the sets. Let now $D = (\Sigma^*, M, Q, Z, \circ, *)$ be the X-

category associated to a rewriting system (Σ, P). Then an interpretation of D in $\mathcal{S}et$ is a X-cofunctor $I: D \to \mathcal{S}et$ so that for every $a \in \Sigma$, $I(a) \neq \emptyset$ and $I(a) \neq \Lambda$ where Λ is the identity set according to the operation $\times$.

Because D is freely generated by (Σ, P) a particular interpretation I is determined by the following conditions:

(i). For each $a \in \Sigma$, $I(a) \neq \emptyset$, $I(a) \in \mathrm{U}$.

(ii). For each $\alpha \to \beta \in P$, $I(\alpha \to \beta): I(\beta) \xrightarrow{f} I(\alpha)$, $f \in F$

The above considerations were made by BENSON [3] using HOTZ 's paper [1]. Now we can observe that for an effective definition of semantics on this way, (i) and (ii) above are not enough. This follows from the fact that not every $a \in \Sigma$ belongs also to the language, and there is no method to treat these cases when something like that appears in I $(\alpha \to \beta)$. That is because the category $D = (\Sigma^*, M, Q, Z, \circ, *)$ charcterizes the syntactical structures in Σ^*, but the language $L(G)$ generated by $G = (\Sigma^*, V_N, V_T, P)$ is generally not a subcategory of D. We can use D to define the language but not for an internal characterization of the language.

Let now $w_1, w_2 \in L(G)$ be given. What kind of relations can we define between w_1 and w_2? Of course, if $\sigma_1 \overset{*}{\Rightarrow} w_1$ and, $\sigma_1 \overset{*}{\Rightarrow} w_2$, then a relation is that, that both belong to the same class of language. But this class can only be defined using the derivations from M. On the other side, the morphisms from M do not belong to the language. In this way we have here the question of the language and metalanguage. The language is our subject to study and especially its internal structure. We are able to do that only by means of metalanguage, or by means of morphisms from M.

The internal structure of the language, supplied by the grammar $G = (\Sigma^*, V_N, V_T, P)$ can be studied only by a detailed study of the role of symbols from V_T during the generation of the language. This will lead us to an overlapping of the language on more levels, so that a level will be the set of generators of the next one. This overlapping must follow the natural way of language definition, which is pointed out by every natural or artificial language. The principles in this overlapping are the following:

(i). There is a first level of the language called dictionary. In the case of natural languages the dictionary is just the dictionary of the language and in the case of programming languages it is its basic data structure, primitive operations and control sequences.

(ii). Let the levels $0,1,\ldots, i-1$ be built. The level i (if it exists) will be built from the elements of levels $0,1,\ldots, i-1$ in a recursively way, by the rules specified by the grammar.

From these observations follows that in the set M of syntactical structures of the elements of the language must be defined an ordering relation which will decompose the set M in classes following the principles (i) and (ii).

The overlapping of the language will show us that we are able to associate to a language an internal algebrical structure richer than that of X-category. This structure will be defined using the notion of free Σ-algebra defined by HIGGINS [5].

2. THE OVERLAPPING OF A FORMAL LANGUAGE

2.1. The general considerations on the decomposition of P

For the definition of the overlapping discussed above we shall start with general definition of the language by means of a rewriting system (Σ, P). So, for $V_N \subseteq \Sigma^*$, $V_T \subset \Sigma$ we consider the grammar $G = (\Sigma^*, V_N, V_T, P)$. Our hypothesis will be that $V_N \subseteq \Sigma$ and the grammar G is a context-free grammar. From § 1.2 we have:

$$\mathcal{S}(\Sigma^*, V_N, V_T, P) = \bigcup_{w \in V_T^*} \bigcup_{\sigma \in V_N} Hom_D(\sigma, w)$$

$$L(G) = \{w \in V_T^* \mid \exists x \in M,\ Q(x) \in V_N,\ Z(x) = \omega\}$$

The classification of the production set P which will supply an overlapping of the language will be relating only to the set $\mathcal{S}(\Sigma^*, V_N, V_T, P)$ that is, only to that part of M which can be considered as syntactical structures in $L(G)$. If this set of morphisms is denoted by $M(L)$ then $M(L) \subseteq M$. The idea is to define an ordering relation on $M(L)$ so that $M(L)$ can be considered decomposed in a finite set of classes $M_0, M_1, \ldots, M_n$ and if $x, y \in M(L)$ then if $x < y$ in the early ordering, that means $x \in M_i(L)$, $y \in M_j(L)$, $i < j$.

For that we consider the grammar $G = (\Sigma^*, V_N, V_T, P)$ not having just one axiom. Each symbol from V_N can be considered an axiom like in RUS [6]. That does not change the fact that G is context-free in a classical way, because there exists a context-free grammar G' with just one axiom so that $L(G) = L(G')$. This idea is very natural having in view the programming language. So, not every expression in a programming language is a program, but belongs to the language. For instance, the identifiers are no programs but belong to the language in which they are defined. In this way the idea of overlapping of the language can be realized in the following way:

(i). The first level of the language is $L_0(G)$ and is defined by the strings $w \in L(G)$ for which we have: if $w \in L_0(G)$ that means that there exists $A \to w \in P$ and $A \in V_N$, $w \in V_T^*$.

Let M_0 be the set of morphisms from $M(L)$ which are syntactical structure for $L_0(G)$.

(ii). Let the levels 0, 1, ..., $i-1$ be built, denoted by $L_0(G)$, $L_1(G)$, ..., $L_{i-1}(G)$ respectively. The level i denoted by $L_i(G)$ will be built by that strings $w \in L(G)$ for which we have: if $w \in L_i(G)$ that means that there exists $A \in V_N$ so that is $x \in Hom_D(A, w)$ then x can be represented like a composition $\circ$ and concatenation $*$ of morphisms from $M_0, M_1, \ldots, M_{i-1}, M_i$.

The effective realization of (i) and (ii) above will be made by an ordering relation defined on the powerset of P. This lead to a decomposition of P in classes of subsets. Such classes were considered by SALOMAA [7] regarding the problem of a restriction on the set of derivations with a context-free grammar. If in Salomaa's decompositions we shall use a criterium like that used in this paper then our grammar can be considered a matrix grammar or an ordered grammar or a time-varying grammar according to the ordering definition. But we shall not restrict the derivations.

2.2. The algorithm for decomposition of P

For describing the algorithm of decomposition of P in classes we shall use the following notations: if $p \in P$ then $V = V_N \cup V_T$, $p = (\alpha, \beta)$ $\alpha \in V_N$, $\beta \in V^*$, and we suppose that $\beta \neq \Lambda$, where Λ is the empty string. By $pr_1(p)$ we denote α and by $pr_2(p)$ we denote β.

If $pr_2(p) = \beta$ then $\beta = t_1 A_1 t_2 A_2 \ldots t_n A_n t_n + 1$ where $t_i \in V_T^*$ or $t_i = \Lambda$, $i = 1, 2, \ldots, n+1$ and $A_i \in V_N$, $i = 1, 2, \ldots, n$. By $Cat(p)$ we denote the set $\{A_1, A_2, \ldots, A_n\}$. If $P_1 \subseteq P$ is a subset of P then $CT(P_1) = \bigcup_{p_i \in P_1} Cat(p_i)$.

Lemma 1. *If $G = (\Sigma^*, V_N, V_T, P)$ and $L(G) \neq \emptyset$ then there exists at least one production $p \in P$ so that $pr_1(p) \in V_N$, $pr_2(p) \in V_T^*$.*

Proof. From $L(G) \neq \emptyset$ it follows that there exists at least one $w \in L(G)$, $w \neq \Lambda$. Then there exists a syntactical structure or a derivation of w so that $A \in V_N$ and $A \overset{*}{\Rightarrow} w$. That means that there is $p \in P$, $pr_1(p) = A$, $pr_2(p) = \beta$. If $w = \beta$ then p is the production from lemma 1. If $w \neq \beta$ then there exists p_1 so that $p_1 \in P$, $A \Rightarrow \beta \overset{*}{\Rightarrow} w$ where $\beta = \beta_1 B \beta_2$ and $pr_1(p_1) = B$, $pr_2(p_1) = \beta_3$. If $\beta_1, \beta_2, \beta_3 \in V_T^*$ then $p_1 = (B, \beta_3)$ is the production. From the hypothesis of $w \in L(G)$ follows that w has at least one finite derivation. That means that in a finite number of steps as above we shall find the production p.

Theorem 1. *If $L(G) \neq \emptyset$ then there exists an unique decomposition of the set P in the subsets $P_0, P_1, \ldots, P_n$ so that the following conditions take place:*

(i). *If $p \in P$ and $p \in P_0$ then $Cat(p) = \emptyset$.*

(ii). *If* $p \in P_i$ *for* $i = 1, 2, \ldots, n$ *then*

$$Cat(p) \subseteq \left(\bigcup_{q \in \bigcup_{j=0}^{i} P_j} pr_1(q) \right) \cup pr_1(p)$$

(iii). $\bigcup_{i=0}^{n} P_i \subseteq P$, $P_i \cap P_j = \emptyset$, $i \neq j$.

(iv.). *The decomposition* $P_0, P_1, \ldots, P_n$ *is the finest one with the bigest cardinality of the classes* $P_0, P_1, \ldots, P_n$ *which has the properties* (i)—(iii).

(v). *If we consider the grammar* $G' = (\Sigma^*, V_N, V_T, P')$ *where* $P' = \bigcup_{i=0}^{n} P_i$ *then it follows* $L(G) = L(G')$.

Proof. The demonstration of this theorem will be made by describing the algorithm which will decompose P so that (i) — (v) hold. For that let DEC be the name of the algorithm. The steps of this algorithm will be denoted by DEC followed by digits.

DEC1. Set index I to zero, $I := 0$. It will count the number of classes.

DEC2. Decompose the set P in two subsets P^1 and P^2 so that $P = P^1 \cup P^2$, $P^1 \cap P^2 = \emptyset$ according to the following definition:

$$P^1 = \{p \in P \mid pr_1(p) \notin Cat(p)\}$$

$$P^2 = \{p \in P \mid pr_1(p) \in Cat(p)\}$$

DEC3. The first class of P is P_0, $P_0 \subseteq P^1$, and is built by the relation $P_0 = \{p \in P^1 | pr_2(p) \in V_T^*\}$.

Observation: From lemma 1 it follows that $P_0 \neq \emptyset$.

DEC4. Let $P_0, P_1, \ldots, P_i$ be already built. Consider the set C defined by $C := P^1 \setminus \bigcup_{j=0}^{i} P_j$ where by $\setminus$ we have denoted the operation of subtraction on sets.

DEC5. Verify the condition $C = \emptyset$? If this condition is true, the algorithm continue with DEC7; if the condition is false, the next step is DEC6.

DEC6. Choose an arbitrary element $p \in C$ and verify the condition $Cat(p) \subseteq \bigcup_{q \in \bigcup_{j=0}^{i} P_j} pr_1(q)$. If this condition is true then we have obtained

a new element of P_{i+1} (at the begining $P_{i+1} = \emptyset$). That means to perform the operation $P_{i+1} := P_{i+1} \cup p$ and the element p will be cleaned from C. The next step of the algorithm in this case is DEC5. If the condition above is false, the next step of the algorithm is DEC7.

DEC7. Verify if all the elements of P^1 are classified. For that set I to $I + 1$, $I := I + 1$, and verify if P_I is empty. If so, then all productions from P^1 which can be classified are already did. The other productions can be eliminated from P because they don't generate elements of the language. In this case set I to $I - 1$, $I := I - 1$ and the next step of the algorithm is DEC9. If $P_I \neq \emptyset$ that means that a new subset has been built. Then perform the operation $C := P^1 \setminus \bigcup_{j=0}^{I} P_j$ and algorithm continue with the step DEC8.

DEC8. Verify the condition $C = \emptyset$? If it is true the next step is DEC9 because all elements of P^1 are classified. If the condition is false the next step of the algorithm is DEC6.

DEC9. Set a new variabile N to I, $N := I$ and begin the classification of P^2.

DEC10. Verify the condition $P^2 = \emptyset$? If it is true the algorithm is completed and subsets are $P_0, P_1, \ldots, P_n$ where n is the value of N If $P^2 = \emptyset$ is false it follows the step DEC11.

DEC11. Choose an arbitrary element $p \in P^2$ and build the corresponding element p^* by erasing from $pr_2(p)$ all symbols equal with $pr_1(p)$.

DEC12. Set I to zero, $I := 0$

DEC13. Verify if $Cat(p^*) = \emptyset$ or if

$$Cat(p^*) \subseteq \bigcup_{q \in \bigcup_{j=0}^{I} P_j} pr_1(q)$$

If at least one of these two condition hold, then $P_{i+1} := P_{i+1} \cup p$ where i is the value of I, and p will be erased from P^2. Next step of algorithm is DEC10. If no one of the above condition hold, the algorithm will look up for a new subset for classifiying p seting I to $I + 1$, $I := I + 1$, and choosing the next step to be DEC13. If there are no sets for classifying p, then is fulfilled $I = N$ and p can be erased from P^2 because there are no elements $w \in L(G)$ which can be obtained by using p. In this case the algorithm continue with DEC10.

The operations used to DEC description can be easily programmed in any assembly language or LISP-like language. The flow diagram of DEC is presented in figure 1.

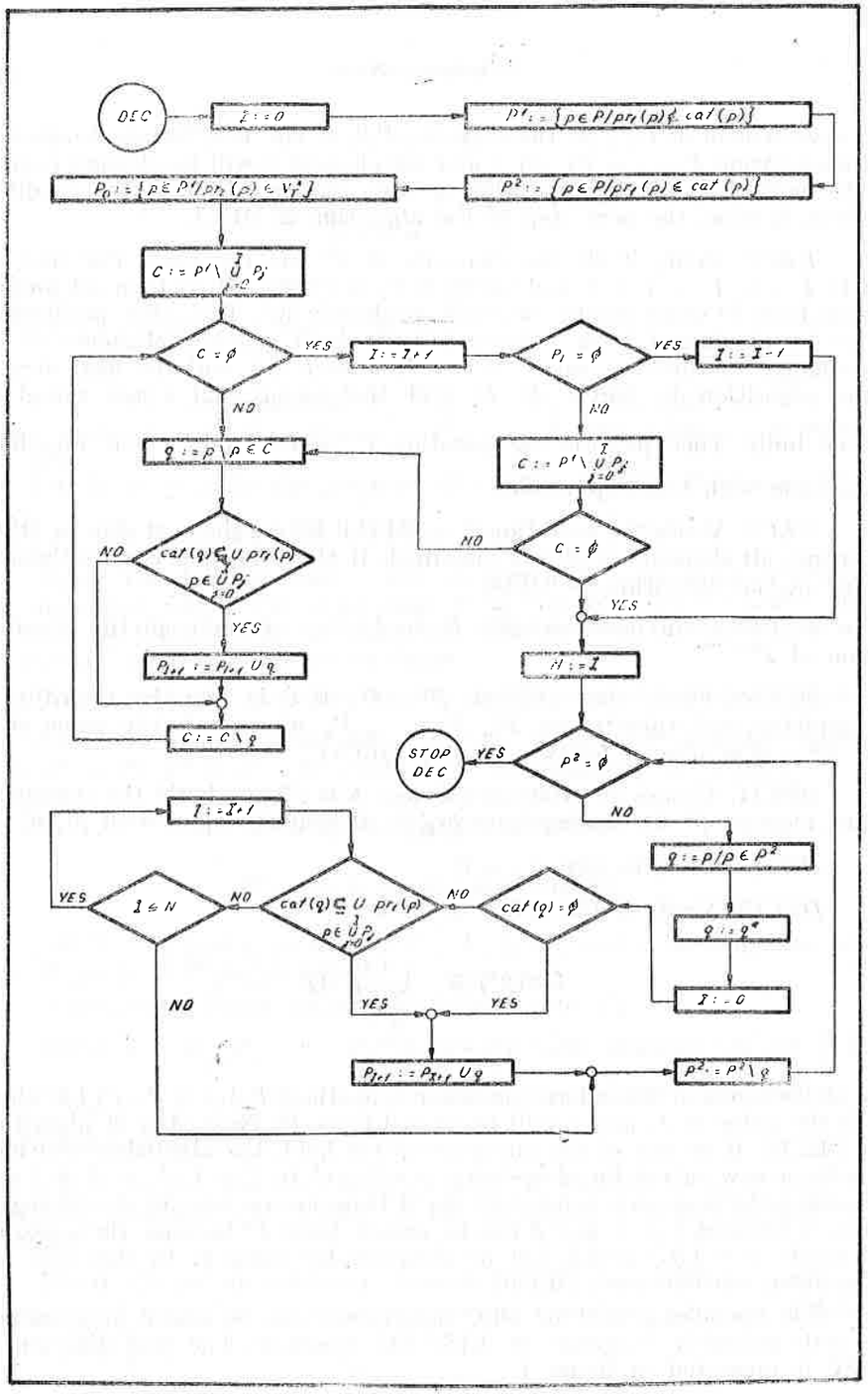

Fig. 1. *The flow diagram of DEC*

Let now $P' = \bigcup_{i=0}^{n} P_i$ where n is the value of N, and $G' = (\Sigma^*, V_N, V_T, P')$. From the algorithm described above it results that $L(G) = L(G')$. The other properties follows from the algorithm description too.

Example: $\Sigma = \{X_0, X_1, a, b, c\}$, $V_N = \{X_0, X_1\}$, $V_T = \{a, b, c\}$,

$$P = \{X_0 \to X_0X_1,\ X_0 \to aX_0b,\ X_1 \to cX_1,\ X_0 \to ab,\ X_1 \to c\}$$

$$P^1 = \{X_0 \to ab,\ X_1 \to c\}$$

$$P^2 = \{X_0 \to X_0X_1,\ X_0 \to aX_0b,\ X_1 \to cX_1\}$$

In this case we have $P_0 = P^1$ and $P_1 = P^2$, $n = 1$. If $G_0 = (\Sigma^*, V_N, V_T, P_0)$ and $G_1 = (\Sigma^*, V_N, V_T, P_0 \cup P_1)$ then $L(G_0) = \{ab, c\}$, $L(G_1) = \{a^nb^nc^m | n \geqslant 0, m \geqslant 0\}$ and $L(G) = L(G_1)$. For $m = 0$, $L(G_1) = \{a^nb^n\}$ and for $n = 0$ $L(G_1) = \{c^m\}$. That means that $L(G_0) \subset L(G_1)$.

From the above algorithm it follows immediately.

Lemma 2. *If* $w \in L(G)$ *and* $x \in Hom_D(A, w)$ *is a synthactical description of* w, *so that for a given* i, $A \in \{pr_1(p) | p \in P_i\}$ *then for each substring* w' *of* w, $w = w_1w'w_2$ *so that* $w' \in L(G)$ *there is a syntactical description* $y \in Hom_D(B, w')$ *where*

$$B \in \left\{ pr_1(p) \,\middle|\, p \in \bigcup_{j=0}^{i} P_i \right\}.$$

This result makes it posible to define the overlaping of the language $L(G)$ as follows:

1. The first level of the language is $L_0(G)$ and it is generated by $G_0 = (\Sigma^*, V_N, V_T, P_0)$.

2. The second level of the language is $L_1(G)$ and it is generated by $G_1 = (\Sigma^*, V_N, V_T, P_0 \cup P_1)$

. . . . . .

i. The i-th level of the language is $L_i(G)$ and it is generated by $G_i = (\Sigma^*, V_N, V_T, P_0 \cup P_1 \cup \ldots \cup P_i)$

. . . . . .

n. The n-th (the last) level of the language is $L(G_n)$ and it is generated by $G_n = (\Sigma^*, V_N, V_T, P_0 \cup P_1 \cup \ldots \cup P_n)$

The sets $P_0, P_1, \ldots, P_i, \ldots, P_n$ are the subsets of the set P defined by the algorithm DEC. Now it is clear that we have

(i) $L_i(G) \subseteq L_{i+1}(G)$, $i = 0, 1, \ldots, n-1$

(ii) $L_n(G) = L(G)$

According to the above relations between the levels of the language $L(G)$ we can define the relations between the syntactical descriptions of the strings of the language. Let us denote by $M(L)$ all morphisms of

X-category D which are syntactical descriptions of the language. Then we have:

1'. The first class of $M(L)$, denoted by $M_0(L)$ is the same with P_0

2'. If the classes $M_0(L)$, $M_1(L)$, ..., $M_i(L)$ have been built, then $M_{i+1}(L)$ will be built by all morphisms from $\mathcal{S}(\Sigma^*, V_N, V_T, P)$ with the following properties:

(i). If $x \in M_{i+1}(L)$ then there exists a decomposition of x according to $\circ$ and $*$ so that $x = y\{\overset{\circ}{*}\}z$, $y \in P_{i+1}$, $z \in M_0 \cup M_1 \ldots \cup M_{i+1}$ or $z \in P_{i+1}$, $y \in M_0 \cup M_1 \cup \ldots \cup M_{i+1}$ where $y\{\overset{\circ}{*}\}z$ means $y \circ z$ or $y * z$.

(ii). If $l(x)$ is the length of x, that is, the number of rules from P_0, P_1, ..., P_{i+1} appllied to build x, then $l(x) \geq i + 1$.

The study of the morphism classes M_0, M_1, ..., M_n will lead to some methods of syntactical analysis using normal algorithms, RUS [8] But the classes M_0, M_1, ..., M_n show us a kind of structure which can be defined on the strings of the language. This structure which is described by M_0, M_1, ..., M_n in an indirect way, can be directly considered. For that we have to define an algebraical structure on $L(G)$, defined by a set of operations supplied in $L(G)$ by the given grammar G. That is, if o is such an operation, and if $w_1, w_2, \ldots, w_n \in L(G)$, then we can build a new string of $L(G)$ by the operation o, $w = o(w_1, w_2, \ldots, w_n) \in L(G)$ The string $o(w_1, w_2, \ldots, w_n)$ can be built by using a free algebra scheme in a word algebra.

Such a structure will be defined in the next section using Σ— algebras defined by HIGGINS [5], for characterizing heterogeneous algebrical structure. For this reason we shall begin the next section with the definition of ΣS-algebras which are suitable structures for characterizing the internal structure of a language.

3. ΣS-ALGEBRA OF A FORMAL LANGUAGE

3.1. ΣS-algebras

Let I be a given non empty set and a family $\mathcal{R}(I)$ of relations on I. An $(n+1)$-ary $(n \geq 0)$ relation on I, $r \in \mathcal{R}(I)$ with the properties that from $(i_1, i_2, \ldots, i_n, i) \in r$ and $(i_1, i_2, \ldots, i_n, j) \in r$ follows $i = j$ will be called an n-ary operation. Such an operation can be considered as a function $I^n \to I$. Let $\mathcal{R}(I)$ be the set of all relations on I. Then $\mathcal{O}(I) \subseteq \mathcal{R}(I)$. The sets $\mathcal{O}(I)$ and $\mathcal{R}(I)$ can be indexed by the n-arity of their components, $\mathcal{O}(I) = \bigcup_n \mathcal{O}_n(I)$, $\mathcal{R}(I) = \bigcup_n \mathcal{R}_n(I)$ where $\mathcal{O}(I)$ is the set of all operations on I.

A pair $(I, \mathfrak{A})$, $\mathfrak{A} \subseteq \mathcal{R}(I)$ will be called a relational algebra on I and a pair $(I, \mathfrak{A})$, $\mathfrak{A} \subseteq \mathcal{O}(I)$ will be called a partial algebra on I.

Let us consider now a non emty set Ω, $\Omega \cap I = \emptyset$ so that $\Omega = \bigcup_n \Omega_n$. *An* Ω-structure on I is defined by a function $\alpha : \Omega \to \mathcal{R}(I)$ which applies

Ω_n in $\mathcal{R}_{n+1}(I)$, $n = 0, 1, \ldots$. In this way a symbol acts under the mapping α as an $(n+1)$-ary relation on I. If each symbol $\omega \in \Omega_n$, $n = 0, 1, \ldots$ acts under α as an n-ary operation, then (I, α) is called an Ω-partial algebra. In this case $\omega \in \Omega_n$, $\omega\alpha \in \mathcal{O}(I)$ $(i_1, i_2, \ldots, i_n, i) \in \omega(\alpha)$ and we denote this operation by $i_1 i_2 \ldots i_n\omega$ or $i = i_1 i_2 \ldots i_n\omega$.

For a given non emty I and a non emty *set* Ω, and an application $\alpha : \Omega \to \mathcal{R}(I)$ the pair $\Sigma = (I, \alpha)$ is called an Ω-scheme or operator scheme [5].

Let now A be a family of sets, $A = \{A_i\}_{i \in I}$ indexed by I. We suppose that for each $\omega \in \Omega_n$, $n = 0, 1, \ldots$ and each $(i_1, i_2, \ldots, i_n, i) \in \omega\alpha$ is given a function $\omega_{i_1 i_2 \ldots i_n} : A_{i_1} \times A_{i_2} \times \ldots \times A_{i_n} \to A_i$.

The family $A = \{A_i\}_{i \in I}$ together with all functions given by a Σ-scheme will be called an Ω-algebra with the operator scheme Σ or shortly, Σ-algebra

In this way the notion of algebra is defined by using heterogeneous operations defined by a given operator scheme Σ. For that two things must be precised:

(i). A function $I^n \to I$ with the role of domain selector of operations defined by the scheme.

(ii). A composition law for the elements of the selected sets by (i).

For algebrical porposes (i) and (ii) are quite enough to charracterize the structure defined by operator scheme $\Sigma = (I, \alpha)$, on a family of sets $A = \{A_i\}_{i \in I}$. That because the problem is not to see how the operations defined by operator scheme are computed, but it is enough to say that for each $\omega \in \Omega$, $\omega\alpha$ will select the domain and codomain of operation, so that

(1). $(i_1, i_2, \ldots, i_n, i) \in \omega\alpha$

(2). $(a_1, a_2, \ldots, a_n)\omega_{i_1 i_2 \ldots i_n i} = a_1 a_2 \ldots a_n \omega_{i_1 i_2 i \ldots i_n i}$ for each $a_k \in A_{i_k}$, $k = 1, 2, \ldots, n$ and $a_1 a_2 \ldots a\, \omega_{i_1 i_2 \ldots i_n i} \in A_i$

For the algorithmic point of view this definition must has a completion in the sense of indicating how the operations $\omega_{i_1 i_2 \ldots i_n i}$ are computed. That means, that from the algorithmic point of view it is not enough to say that n-tuple $(a_1, a_2, \ldots, a_n)$ belongs to the domain of $\omega_{i_1 i_2 \ldots i_n i}$. It is necessary also to indicate the way of depending of the building process of $a_1 a_2, \ldots a_n \omega_{i_1 i_2 \ldots i_n i}$ by the values of $a_1, a_2, \ldots, a_n$. That can be better seen considering this process as an algorithm described by a sequences of steps. The order of the execution steps is given by the values of $a_1, a_2, \ldots, a_n$ which are data of the algorithm.

In the usual cases of operation in the numerical structures, this aspect is not very clear beacuse the numbers have two different determinations. On one side as symbolic names and on the other side as values of these names. The value of a number is predicated by its symbolic form, or in other words, the value of a number is implicitely given by its symbolic representation. If we consider that speaking about sets we use the name of their elements, considered as variable on these sets, then this double aspect of the name and of the value arise immediately.

The above observations lead us to a generalization of the notion of operator scheme. That arises by determining an operation under the following conditions:

(i). A function $I^n \to I$ which will select the domain and codomain of operations.

(ii). A composition law for the elements of the selected sets by (i).

(iii). A state word of the operation which will indicate the way in which the process of building the result is depending by the value of the operands.

For clarifying this condition (iii) let us take an example. Suppose that A is a set of predicates, B is a set of operation names and C is a set of operation names too. In the programming languages often we have constructions of the form *if A then B else C*. Let us denote the type of A by 1, types of B and C by 2 and the type of the above construction by 3. Then one element of an operator scheme as (1, 2, 2, 3) is not enough for showing what we expect from our operation. This element of operator scheme should be written (if 1, then 2, else 2, 3) or in other form ((1, 2, 2), (if, then, else), (3)). This is not only an example in programming languages. But we try to indicate a general scheme for this kind of operations, using on one side the context-free grammars and on the other side Σ-algebras defined by HIGGINS [5].

By an operator scheme ΣS we mean a pair (I, α) where α is a function $\alpha : \Omega \to \mathcal{O}(I) \times ST$, where ST is a given set of words, charracterizing the selected operation by α, and called semantics selectors or state words.
In this way to each $\omega \in \Omega_n$ will correspond by α a triple $(i_1, i_2, \ldots, i_n)$ in the domain of $\omega\alpha$, (i) in the codomain of $\omega\alpha$ and $s \in ST$ of $\omega\alpha$.

Let now $A = \{A_i\}_{i \in I}$ be a family of sets indexed by I and a ΣS scheme as considered above. Then for $\omega \in \Omega_n$ we have the followings

If $(i_1, i_2, \ldots, i_n) \in D(\omega\alpha)$, $(i) \in R(\omega\alpha)$, $(s_1, s_2, \ldots, s_{n+1}) \in ST(\omega\alpha)$, where by D, R, ST we have denoted respectively domain, codomain and state words of the operation $\omega\alpha$, then for each $a_k \in A_{i_k}$ $(s_1a_1, s_2a_2, \ldots, s_na_n, s_{n+1})\,\omega\alpha = s_1a_1s_2a_2 \ldots s_na_ns_{n+1}$ belongs to A_i.

In this definition ω is not explicitely represented in the result because sometimes it is represented by one of the semantic selectors or state words. But sometimes it is necessary to explicit them in the form of the „name of operation" or „name of result". In this case ω will be a label of the result and we shall be able to define operations on these names too, being able in this maner to have an algebraical description of the algorithmic processes. The form of representing of these labels, when used, will be $\omega : s_1a_1s_2a_2 \ldots s_na_ns_{n+1}$ or avoiding any ambiguity in the form $\omega i_1i_2 \ldots i_ni : s_1a_1s_2a_2 \ldots s_na_ns_{n+1}$. In this way the labels becom themselves operands of the type i.

A family of sets $A = \{A_i\}_{i \in I}$ indexed by I together with all operations defined by a ΣS-operator scheme, $\Sigma S = (I, \alpha)$, where $\alpha : \Omega \to \mathcal{O}(I) \times ST$ will be called a ΣS-algebra.

When $ST = \emptyset$ then ΣS-algebra is just a Σ-algebra.

3.2. ΣS-Algebra associated to a grammar

Let $G = (\Sigma^*, V_N, V_T, P)$ be a given context-free grammar, supplied by a semi-Thue system (Σ, P) and $D = (\Sigma^*, M, Q, Z, \circ, *)$ be X-category generated by (Σ, P). The language generated by G is defined by $L(G) = \{w \in V_T^* \mid \exists x \in M, Q(x) \in V_N, Z(x) = w\}$. As we have shown there is an unique decomposition of P in the subsets $P_0, P_1, \ldots, P_n$ so that theorem 1 hold. Using this decomposition we shall define now in $L(G)$ a structure of ΣS-algebra. For that, let us use the following notions:

By the index of G we shall understand the set V_N indexed by its elements. For avoiding any misunderstanding we denote the index of G by I_G, supposing that I_G is a set of natural numbers so that the following hold:

(i). If $i \in I_G$ then there exists $A_i \in V_N$;

(ii). If $i, j \in I_G$, $i \neq j$, then there exists $A_i, A_j \in V_N$, $A_i \neq A_j$.
Given a production $p \in P$, $p = (\alpha, \beta)$ then $pr_1(p) = \alpha$, $pr_2(p) = \beta$, $\beta = s_1A_1s_2A_2 \ldots s_nA_ns_{n+1}$, $s_i \in V_T^*$ or $s_i = \Lambda$, $A_i \in V_N$, $i = 1, 2, \ldots, n$, $n+1$
By $pr_{V_T}(p)$ we shall denote the sequence $(s_1, s_2, \ldots, s_n, s_{n+1})$ and by $pr_{V_N}(p)$ we shall denote the sequence $(A_1, A_2, \ldots, A_n)$. By $I(pr_{V_N}(p))$ we denote the sequence $(i_1, i_2, \ldots, i_n)$ where i_k is the index of A_k in I_G, for $k = 1, 2, \ldots, n$.

For the decomposition of P in the subsets $P_0, P_1, \ldots, P_n$ let us consider the following sets:

$$S_i = \bigcup_{p \in P_i} pr_{V_T}(p),\ O_i = \bigcup_{p \in P_i} I(pr_{V_N}(p)),\ T_i = \bigcup_{p \in P_i} I(pr_1(p)).$$

It is clear that to each $p \in P_i$ we can associate a triple (o_i, s_i, t_i) where $o_i \in O_i$, $s_i \in S_i$, $t_i \in T_i$. The triple $(o_i(p), s_i(p), t_i(p))$ will be called operation scheme supplied by p · $o_i(p)$ is called domain of the operation, $t_i(p)$ is called codomain of the operation and $s_i(p)$ is called state word or semantics selector of the operation. Let now

$$\Sigma_i = \{(o_i(p), s_j(p), t_i(p)) \mid o_i(p) \in O_i,\ s_i(p) \in S_i,\ t_i(p) \in T_i\}$$

be for a given subset P_i, and we call it operator scheme of the subset P_i. For the grammar G we can consider

$$O_G = \bigcup_{i=0}^{n} O_i,\ S_G = \bigcup_{i=0}^{n} S_i,\ T_G = \bigcup_{i=0}^{n} T_i,\ \Sigma_G = \bigcup_{i=0}^{n} \Sigma_i$$

called respectively domains of operations supplied by G, state words or semantics selectors of operations supplied by G, codomain of operations supplied by G and operator scheme of operations supplied by G.

Example: Let G be the grammar from the example in 2.2, where we consider for X_0 the index 1 and for X_1 the index 2. Then we have:

$$I_G = \{1, 2\},\ S_0 = \{ab,\ c\},\ S_1 = \{(\Lambda,\ \Lambda,\ \Lambda,),\ (a,\ b),\ (c,\ \Lambda)\}$$

$$O_0 = \{(\ ),\ (\ )\},\ O_1 = \{(1,\ 2),\ (1),\ (2)\},\ T_0 = \{(1),\ (2)\}$$

$$T_1 = \{(1),\ (1),\ (2)\},\ \Sigma_0 = \{(\ (\ ),\ (ab),\ (1)),\ ((\ ),\ (c),\ (2))\}$$

$$\Sigma_1 = \{((1,\ 2),\ (\Lambda,\Lambda,\Lambda,),\ (1)),\ ((1),\ (a,\ b),\ (1)),\ ((2),\ (c,\Lambda),\ (2))\}$$

Using the notion of ΣS-scheme defined by a grammar G, we shall define the operations in the language $L(G)$. In this way the structure of the language will be described by a structure of ΣS-algebra. ΣS-algebra associated to the language will be in the this way a characterization of both, syntactic and semantics structure of the language. This will be a result of the procedure by which we build ΣS-algebra of the language $L(G)$ in the form of free ΣS- algebra of words, by considering some interpretations (or realisations) of this structure in some ΣS-algebras defined on the sets.

An operation described by an element $\sigma \in \Sigma_G$ will be defined as follows: let $X = \{X_i\}_{i \in I_G}$ be a family of sets, indexed by the index I_G of the grammar. A pair $(X_i,\ i),\ i \in I_G$ will be called a set of the type i. If for $\sigma \in \Sigma_G,\ \sigma = (o,\ s,\ t)$, then $o = (i_1,\ i_2,\ \ldots,\ i_n),\ s = (s_1,\ s_2,\ \ldots,\ s_n, s_{n+1},\ t = (i),\ i_k \in I_G,\ k = 1,\ 2,\ \ldots,\ n,\ i \in I_G,\ s_k \in S_G\ k = 1,\ 2,\ \ldots,\ n,\ n+1$, then the operation defined by σ in $X = \{X_i\}_{i \in I_G}$ is defined as a composition law $\omega_{i_1 i_2 \ldots i_n i} : s_1 * X_{i_1} \times s_2 * X_{i_2} \times \ldots \times s_n * X_{i_n} \to s_{n+1} * X_i$ where s_k are semantics selectors of the algorithm steps in the process of evaluating of $\omega_{i_1 i_2 \ldots i_n i}$. If all $s_i = \Lambda,\ i = 1,\ 2,\ \ldots,\ n,\ n+1$ the operation is defined as an usual heterogeneous operation, that is, the corresponding algorithm steps of its evaluation are selected in a standard way. A standard selection of the algorithm steps can be defined on some components of the operation and a non standard selection (which depend of the selected semantics for operands) can be defined on onther components. The first will correspond to $s_i = \Lambda$ and the second will correspond to $s_i \neq \Lambda$. The symbol * was used to ditinguish by the symbol $\times$ of the cartesian product.

Given a grammar G and Σ_G the ΣS-scheme associated to G, a family of sets indexed by I_G, $X = \{X_i\}_{i \in I_G}$ together with all operations supplied by Σ_G in X will be called ΣS-algebra associated to G and will be denoted by $\Sigma S(G,\ X) = (X = \{X_i\}_{i \in I_G},\ \Sigma_G)$.

Let now define the ΣS-algebra defined by G in the language $L(G)$ For that we shal build $\Sigma S(G,\ L(G))$ as a free ΣS- algebra of words as follows: Let $L(G)$ be considered as the set of all strings of the language. Then we have:

(i). $S_0 \subseteq L(G)$ being considered as individual constants of $\Sigma S(G,\ L(G))$.

(ii). Let now be $w_1,\ w_2,\ \ldots,\ w_n \in L(G)$ of the types $i_1,\ i_2,\ \ldots,\ i_n$ and $\sigma \in \Sigma_i,\ i \geqslant 1$. If $\sigma = (o,\ s,\ t)$ and if $o = (i_1,\ i_2,\ \ldots,\ i_n)$ then we can write $\sigma(w_1,\ w_2,\ \ldots,\ w_n) = s_1 w_1 s_2 w_2 \ldots s_n w_n s_{n+1}$ or using the operation name $\omega_{i_1 i_2 \ldots i_n i} : s_1 w_1 s_2 w_2 \ldots s_n w_n s_{n+1}$. Of course, $\sigma(w_1,\ w_2,\ \ldots,\ w_n)$ belongs to the language and has the type i, where $t = (i)$. Now, because $L_0(G) = S_0$ and because $L_i(G) \subseteq L_{i+1}(G),\ i = 0,\ 1,\ \ldots,\ n,\ L_n(G) = L(G)$, we can formulate the following theorem:

THEOREM 2. *For each i, $0 \leqslant i \leqslant n$ the language $L_i(G)$ is a free ΣS-algebra of words, generated by $L_0(G) \cup L_1(G) \cup \ldots \cup L_{i-1}(G)$, with the operator scheme $\Sigma_{i_G} = \bigcup_{j=0}^{i} \Sigma_j$ and will be denoted by*

$$L_i(G) = \Sigma S(L_i(G),\ \Sigma_{i_G}).$$

In this way the free Σ -algebra of words supplied by G in $L(G)$ can be written as the pair

$$\Sigma S(G,\ L(G)) = \Sigma S(L(G),\ \Sigma_G)$$

This result was obtained by considering only operations of the n-arity 0, 1, and 2, that is by reducing the grammar to a Ghomsky normal form in the paper RUS [6].

From the above theorem it followst hat we have on one side the overlapping of the language and on the other side the ΣS-structure defined on each level of the language. Also, theorem 2 show us the relation between levels from the generation and structure point of view.

We can define now the semantic domain of a given language in the following way: Let $X = \{X_i\}_{i \in I_G}$ be a given family of sets indexed by the index I_G of the grammar. ΣS-algebra (when is defined) $\Sigma S(X,\ G) = (X = \{X_i\}_{i \in I_G},\ \Sigma_G)$ is called an element of the semantic domain of $L(G)$. Because the ΣS-algebra of the language $L(G)$ is $\Sigma S(L(G),\ G) = (L(G),\ \Sigma_G)$, it is similar to $\Sigma S(X,\ G)$, we can define an homomorphism $H : L(G) \to \Sigma S(X,\ G)$, called semantic homomorphism in the following way:

(i). First we define H on the structure associated to $L_0(G)$, which is generally a finite set (or denumerable set). For that let $w \in L_0(G)$ be an element of the type $i \in I_G$. Then $H(w)$ will be the injection of w in the set X_i which has the same type as w.

(ii). Let us suppose that H was defined on the levels $L_0(G), L_1(G), \ldots, L_i(G)$. For extenison of H to $L_{i+1}(G)$ we proceed as follows: If $w \in L_{i+1}(G)$, because $L_0(G) \cup L_1(G) \cup \ldots \cup L_i(G)$ is the set of generators for $L_{i+1}(G)$ there exist the strings $w_1,\ w_2,\ \ldots,\ w_n$ form $\bigcup_{j=0}^{i} L_j(G)$ and $\sigma \in \Sigma_{(i+1)G}$

so that $\sigma(w_1, w_2, \ldots, w_n) = w$, or $w = \omega_{i_1 i_2 \ldots i_n t}: s_1 w_1 s_2 w_2 \ldots s_n w_n s_{n+1}$. Then we have

$$H(w) = \omega_{i_1 i_2 \ldots i_n t}: s_1 H(w_1) s_2 H(w_2) \ldots w_n H(w_n) s_{n+1}$$

In this way the semantic homomorphism defined on the first level of the language will be homomorphic extend to all the language. The homomorphism H is in fact an interpretation of the language in the ΣS-algebra defined by G in the family of sets $X = \{X_i\}_{i \in I_G}$. This interpretation is given by a constructive method and describealso the role which state words have.

The set of all ΣS-algebras defined by a given grammar G will be called semantic domain of the language generated by G. It is quite clear that a language has a family of semantics, but each element of this family has a structure similar to that of language, because it is defined by the same grammar. It should be interesting to study the classes of languages defined by some identity relations defined in ΣS-algebras. We should obtain in this way the varieties of the languages.

3.3. Relations between languages

In the last part of the paper we shall define the relation of translation between two languages. For that let two grammars G and G' be given and the languages $L(G)$ and $L(G')$ generated respectively by G and G'. We suppose that G and G' are context-free and that operator schemes defined by them are respectively Σ_G and $\Sigma_{G'}$. First of all we shall define the notion of representation of Σ_G-scheme in $\Sigma_{G'}$-scheme. For that we define a structure on $\Sigma_{G'}$ using some composition law in $\Sigma_{G'}$ as in COHN [4]. Let for that $I_{G'}$ be the index of G'. If $\sigma' \in \Sigma_{G'}$ then $\sigma' = (o', s', t')$, $o' \in I^n_{G'}$, $s' \in S_{G'}$, $t' \in I_G$. Let $\mathbf{N}$ be the set of all n-aryties of operations from $O(I_{G'})$. As we have observed $O(I_{G'})$ can be indexed by the set $\mathbf{N}$, $O(I_{G'}) = \{O_n(I_{G'})\}_{n \in \mathbf{N}}$, and if $I_{G'}$ is not empty then $\mathbf{N}$ is not empty too.

A function $\beta: \mathbf{N}^k \to \mathbf{N}$ will be called an operation scheme on $O(I_{G'})$ if the following conditions hold:

(i). For each $(n_1, n_2, \ldots, n_k) \in \mathbf{N}^k$ which belongs to the domain of β there exists in $O(I_{G'})$ the operations $o_1, o_2, \ldots, o_k$ of the n-aryties $n_1, n_2, \ldots, n_k$.

(ii). There exists a function $\theta_d: I^{n_1}_{G'} \times I^{n_2}_{G'} \times \ldots \times I^{n_k}_{G'} \to I^n_{G'}$ where $n = \beta(n_1, n_2, \ldots, n_k)$.

(iii). There exists an operation $\theta_r \in O_k(I_{G'})$ so that the composition $\theta_r(o_1, o_2, \ldots, o_k)$ is defined in $O(I_{G'})$.

(iv). There exists an operation $o \in O_n(I_{G'})$, $o: I^n_{G'} \to I_{G'}$ denoted by $o(\theta_d, \theta_r)$, where $n = \beta(n_1, n_2, \ldots, n_k)$ and $I^n_{G'}$ belongs to the codomain of θ_d.

(v). The following diagram of operation composition in $O(I_{G'})$ commutes

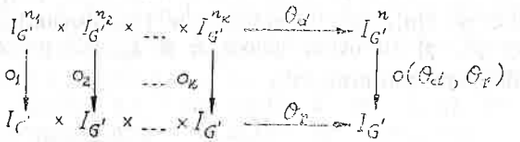

Fig. C

Let now Θ be a set so that $\Theta \cap \mathbf{N} = \emptyset$ and a function $\mathfrak{d}: \Theta \to \mathcal{O}(\mathbf{N})$ where $\mathcal{O}(\mathbf{N})$ denotes the operation scheme as above. In this way we have defined on the set $O(I_{G'})$ of operations on $I_{G'}$ a Σ-algebra Because $\Sigma_{G'}$ is a subset of $O(I_{G'})$ the restriction of this structure to the set of operations defined by $\Sigma_{G'}$ will be called a Σ-structure on $\Sigma_{G'}$ and will be denoted by $\sum_{G'} = (\mathfrak{d}, \mathbf{N}_{G'})$. The effect of some operations $\theta \in \sum_{G'}$ is described in the following way: let $A = \{A_i\}_{i \in I_{G'}}$ be a family of sets indexed by $I_{G'}$ and $\theta \in \Theta$. $\mathfrak{d}\theta$ is an operation in $\mathcal{O}(\mathbf{N})$. Let this operation be $\mathfrak{d}\theta: \mathbf{N}^k \to \mathbf{N}$. Then the operation supplied by θ in $A = \{A_i\}_{i \in I_{G'}}$ is described by the diagram

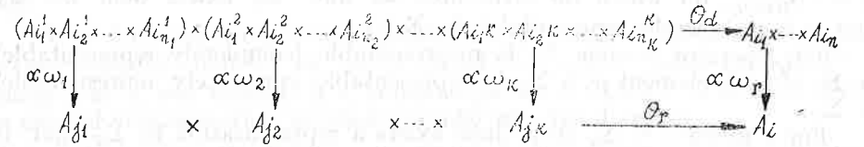

Fig. D

The process which is described by this diagram is as follows:

(i). The operations $o_1, o_2, \ldots, o_k$ defined by $\Sigma_{G'}$ on the family $A = \{A_i\}_{i \in I_{G'}}$ are determinad by $\alpha\omega_1, \alpha\omega_2, \ldots, \alpha\omega_k$. To these operations are corresponding in $\sum_{G'}$ the operation $\alpha\omega = \mathcal{O}(\theta_d, \theta_r)$ where θ_d and θ_r are described by the diagram of operation composition.

(ii). The domain of $\alpha\omega$ $(\alpha\omega_1, \alpha\omega_2, \ldots, \alpha\omega_k)$ where $\alpha\omega_r$ is given by the operation θ_r, is given by the codomain of θ_d, and the codomain of $\alpha\omega_r$ $(\alpha\omega_1, \alpha\omega_2, \ldots, \alpha\omega_k)$ is given by the codomain of θ_r.

(iii). The effective construction of the result is made by a special composition of $\alpha\omega_1, \alpha\omega_2, \ldots, \alpha\omega_k$ indicated by $\alpha\omega_r$. In this composition sometimes we must use a state word which is determined by the state words of given operations from the composition.

Let now $\Sigma S(A, G)$ and $\Sigma S(B, G')$ be two ΣS-algebras defined by Σ_G and $\Sigma_{G'}$ in the families $A = \{A_i\}_{i \in I_G}$ and $B = \{B_i\}_{i \in I_{G'}}$ and a pair of functions (F, φ), $F : A \to B$, $\varphi : I_G \to I_{G'}$, $F = \{F_i\}_{i \in I_G}$ $F_i : A_i \to B_{\varphi(i)}$. An operation σ defined by Σ_G is called representable in $\Sigma(\Sigma_{G'}, \Sigma_{G'})$ if there exists in $\Sigma_{G'}$ the operations $\sigma'_1, \sigma'_2, \ldots, \sigma'_k$ and $\theta \in \Sigma_{G'}$ so that the transformation of the effect of σ on the family $A = \{A_i\}_{i \in I_G}$ by (F, φ) is equal to the effect of $\theta(\sigma'_1, \sigma'_2, \ldots, \sigma'_k)$ on the transformation of the family $A = \{A_i\}_{j \in I_G}$ by (F, φ) In other words, $\sigma \in \Sigma_G$ is representable in $\Sigma_{G'}$ if the following diagram commutes:

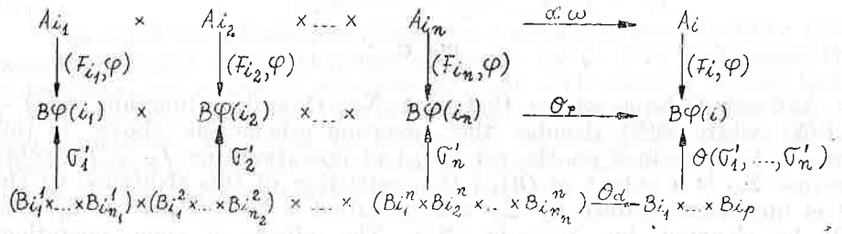

Fig. E

Of course, here is implicitely postulated that the result types of $\sigma'_1, \sigma'_2, \ldots, \sigma'_k$ is $\varphi(i_1), \varphi(i_2), \ldots, \varphi(i_n)$.

If $\sigma \in \Sigma_G$ is representable in $\Sigma_{G'}$ for every family $A = \{A_i\}_{i \in I_G}$ and $B = \{B_j\}_{j \in I_G}$, for which the structures Σ_G and $\Sigma_{G'}$ exist, then we say that σ is completely representable in $\Sigma_{G'}$.

The operator scheme Σ_G is respresentable (completely representable) in $\Sigma_{G'}$ if each element $\sigma \in \Sigma_G$ is representable completely representable) in $\Sigma_{G'}$.

For a given $\sigma \in \Sigma_G$ even there exists a representation in $\Sigma_{G'}$ that is not unique. If we shall denote by $\mathbf{R}(\sigma)$ the set of all representations of σ in $\Sigma_{G'}$ sometimes we can define an ordering relation on $\mathbf{R}(\sigma)$, having in view either the criterium of minimal number of operations $\sigma' \in \Sigma_{G'}$ which contribute to the representation or the minimal n-aryties of these operations. In the programming languages the first criterium leads to so called translators which generate programs with high speed and the second criterium leads to translators which generate programs with an economical use of the memory space.

Let now G and G' be given. We shall define a translator $\mathbf{T} : L(G) \to L(G')$ for a given representation of Σ_G in $\Sigma_{G'}$. For that we shall use the structure of free ΣS- algebras of words of the languages $L(G)$ and $L(G')$ respectively $\Sigma S(L(G), \Sigma_G)$ and $\Sigma S(L(G'), \Sigma_{G'})$ Because these structures can be considered on the levels we have:

$$L_0(G) = \Sigma S(L_0(G), \Sigma_{0_G}), \quad L'_0(G') = \Sigma S(L_0(G'), \Sigma_{0_{G'}})$$

For $\sigma_0 \in \Sigma_0$, let $r(\sigma_0) = \theta(\sigma'_1, \sigma'_2, \ldots, \sigma'_n)$, $\sigma'_i \in \Sigma_{G'}$, $i = 1, 2, \ldots, n$ be a given representation. Because $\sigma_0 = ((), (s), (t))$ using the representation functions we have $F(s) \in S_{G'}$, $\varphi(t) \in I_{G'}$ and $\sigma'_0 = ((\ ), (F(s)), (\varphi(t)))$. In this way we have for every $\sigma_0 \in \Sigma_{0_G}$ a representation by only on operation σ'_0 in $\Sigma_{0_{G'}}$. Let us suppose that $\mathbf{T}$ is defined for all $L_0(G)$ as above. Because $L_0(G)$ is a finite set, $\mathbf{T}$ can be easy build for $L_0(G)$. For the programming languages $\mathbf{T} : L_0(G) \to L_0(G')$ can be represented by the algorithm of scann.

Let now $L_i(G) = \Sigma S(L_i(G), \Sigma_{i_G})$ be given. We know that $L_i(G)$ is freely generated by $L_0(G) \cup L_1(G) \cup \ldots L_{i-1}(G)$. Supposing that $\mathbf{T}$ has been defined on these levels we can build $\mathbf{T}$ on $L_i(G)$ as follows: let $w_1, w_2, \ldots, w_n$ generators and $\sigma \in \Sigma_{i_G}$ so that $\sigma(w_1, w_2, \ldots, w_n)$ belongs to $L_i(G)$and $r(\sigma) = \theta(\sigma'_1, \sigma'_2, \ldots, \sigma'_k)$ the given representation of σ. Because $w_1, w_2, \ldots, w_n$ are generators they can be supposed that have already translations $\mathbf{T}(w_1), \mathbf{T}(w_2), \ldots, \mathbf{T}(w_n)$. Then the translation of $\sigma(w_1, w_2, \ldots, w_n$ will be the expression

$$\theta(\sigma'_1, \sigma'_2, \ldots, \sigma'_k)(\mathbf{T}(w_1), \mathbf{T}(w_2), \ldots, \mathbf{T}(w_n)) \in L(G')$$

Because $L(G) = L_n(G)$ the translation $\mathbf{T}$ defined above is well defined on the language $L(G)$.

THEOREM 3. *Let G and G' be two context-free grammars and $L(G)$ and $L(G')$ the languages generated by G and G'. For every given representation of Σ_G in $\Sigma_{G'}$ translation $\mathbf{T} : L(G) \to L(G')$ defined above is semantic preserving.*

Proof. In order to proove this theorem we shall consider an element of the semantic domain of $L(G)$, $\Sigma S(A, G) = (A = \{A_i\}_{i \in I_G}, \Sigma_G)$ The semantic as it was defined is a homomorphism $H : L(G) \to \Sigma S(A, G)$ For the sake of demonstration we shall consider now the diagram

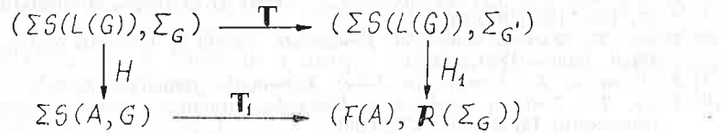

Fig. F

If this diagram commutes then the theorem is proved. $\mathbf{T}_1$ and $\mathbf{H}_1$ are induced functions by $\mathbf{T}$ and H. Having in view that $\mathbf{R}(\Sigma_G) \subseteq \Sigma_{G'}$ it follows that $\Sigma S(F(A), R(\Sigma_G))$ is an algebra similar to a subalgebra of $\Sigma S(L(G'), \Sigma_{G'})$. In this way H_1 is a homomorphism. In the same way $\mathbf{T}_1$ is a homomorphism. If $\mathbf{T}_1 H = H_1 \mathbf{T}$ on the set of generators then by extension the equality holds for all algebras. Choosing now F the identity on

the family $A = \{A_i\}_{i \in I_G}$ the relation $\mathbf{T}_1 H = H_1 \mathbf{T}$ on the set of generators is obviously from the definition of the semantic and of the representation.

Observations 1. For the programming languages, when $L(G)$ is the source language and $L(G')$ is the object language the translation can be built as follows:

(i). Define the structures on $L(G)$ and $L(G')$.

(ii). Choose the representation of Σ_G in $\Sigma_{G'}$.

(iii). Define the translation on data base structures, pimitive operations and control sequences, that is on $L_0(G)$.

(iv). Translation of a program is built having in view its representation in $L(G)$ by means of the generators on one side, and on the other side the corresponding representation in $L(G')$ by means of representation of Σ_G in $\Sigma_{G'}$, and using the function defined by (iii).

In this case the syntactic analysis problem and semantics anaylsis problem are solved together.

2. Theory of representation of an operation set Σ_G in an other operation set $\Sigma_{G'}$ has as origin so called clones of operations defined by COHN [4].

3. The translation $\mathbf{T}$ of two languages $L(G)$ and $L(G')$ is a relation because the representation of Σ_G in $\Sigma_{G'}$ in not unique. But, for a given fixed representation, $\mathbf{T}$ becomes a function described by means of an algorithm, whose program is called compiler.

REFERENCES

[1] Hotz, G., *Eindeutigkeit und Mehrdeutigkeit formaler Sprachen.* EIK **2**: *4*, 235–246 (1966).

[2] Griffiths, T.V., *Some remarks on derivations in general rewriting systems.* Information and Control, **12**, 27–54 (1968).

[3] Benson, D. B, *Syntax and Semantics: A categorial view.* Information and Control, **17**, 145–160 (1970).

[4] Cohn, P.M., *Universal Algebra.* Harper and Row, New York, 1965.

[5] Higgins, Ph. J., *Algebras with a Scheme of operators.* Mathematische Nachrichten, **27**, 115–132 (1963/64).

[6] Rus, T., *Algebra limbajelor formalizate.* Studii şi Cercetări Matematice (Bucureşti), **19**, 9, 1309 – 1324, (1967).

[7] Salomaa, A., *Lectures in Arhus.* Danemark, January 1972.

[8] Rus, T., *Tratarea algebrică a limbajelor formalizate.* Studii şi Cercetări Matematice (Bucureşti), **19**, 2, 259–272, (1967).

Received 1. XI. 1973.

Institutul de Tehnică de Calcul
Filiala Cluj-Napoca