Tomus 4. Fasciculus 4.

ACTA CYBERNETICA

FORUM CENTRALE PUBLICATIONUM
CYBERNETICARUM HUNGARICUM

Szeged, 1980
Curat: Universitas Szegediensis de Attila József nominata

4. kötet 4. füzet

ACTA CYBERNETICA

A HAZAI KIBERNETIKAI KUTATÁSOK
KÖZPONTI PUBLIKÁCIÓS FÓRUMA

Szeged, 1980. június
A Szegedi József Attila Tudományegyetem gondozásában

Tomus 4.

ACTA CYBERNETICA

FORUM CENTRALE PUBLICATIONUM
CYBERNETICARUM HUNGARICUM

Szeged, 1980
Curat: Universitas Szegediensis de Attila József nominata

INDEX

Tomus 4.

79-5678 — Szegedi Nyomda — F.v.: Dobó József igazgató

On some extensions of indian parallel context free grammars

By J. DASSOW

1. Introduction

In order to get better models for aspects of programming and natural languages some extensions of context free grammars are introduced, for instance matrix grammars, random context grammars, programmed grammars, time-variant grammars (see [1], [13], [7], [10], [11]), which are characterized by mechanisms regulating the use of the productions. The relations between the associated language families are studied by some authors (see [10], [4], [11]).

In [12], R. SIROMONEY and K. KRITHIVASAN regard a parallel version of context free grammars. In this paper we introduce some of the above mentioned extensions for these indian parallel context free grammars. An other generalization of the indian parallel grammars are the EDTOL systems (see [3]). We shall prove that all these language families coincide.

The result has also another interesting aspect. The EDTOL systems work purely parallel, i.e. all occurrences of all letters are rewritten in a single derivation step; the other extensions of the indian parallel grammar have a sequential aspect because only all occurrences of one letter are rewritten in a single step. Therefore our result can be regarded as a sequential characterization of EDTOL languages. Thus, it is of interest in connection with the sequential characterizations of ETOL languages (see [14], [6], [9], [8], [2]).

2. Definitions and notations

At first we recall the definition of the indian parallel context free grammar and its derivation process.

Indian context free grammar. An indian context free grammar is a construct $G=(V_N, V_T, P, S)$ where

i) V_N and V_T are finite nonempty sets, $V_N \cap V_T = \emptyset$,

ii) P is a finite subset of $V_N \times (V_N \cup V_T)^*$ (the elements of P are written as $A \to w$, $A \in V_N$, $w \in (V_N \cup V_T)^*$),

iii) $S \in V_N$.

Let $V=V_N\cup V_T$. Let $x\in V^+$, $y\in V^*$. We say that x directly derives y iff
i) $x=x_1Ax_2Ax_3\dots x_{n-1}Ax_n$, $A\in V_N$, $x_i\in(V\setminus\{A\})^*$,
ii) $y=x_1wx_2wx_3\dots x_{n-1}wx_n$,
iii) $A\to w\in P$.

Then we write $x\Rightarrow y$. Let $\overset{*}{\Rightarrow}$ be the reflexive and transitive closure of $\Rightarrow$.

The language $L(G)$ generated by G is defined as

$$L(G)=\{x\colon\ S\overset{*}{\Rightarrow}x,\ x\in V_T^*\}.$$

Now we define some extensions of this grammar by certain mechanisms regulating the derivation process. In all cases we use the alphabet V_N of nonterminals, the alphabet V_T of terminals, the axiom $S\in V_N$, productions $A\to w$, $A\in V_N$, $w\in V^*$, the application of a rule is in all cases defined as above, and if it is not stated otherwise then the associated language is defined in the way given above. We give the regulating mechanisms.

Indian matrix grammar. $G=(V_N, V_T, M, S)$ is an indian matrix grammar iff M is a finite set of finite sequences of productions,

$$M=\{m_1, m_2, \dots, m_r\},$$

$$m_i=[A_{i_1}\to w_{i_1}, A_{i_2}\to w_{i_2}, \dots, A_{i_s}\to w_{i_s}]\quad\text{for}\quad i=1,2,\dots,r.$$

The elements of M are called matrices. To apply such a matrix one has to apply the productions $A_{i_1}\to w_{i_1}, \dots, A_{i_s}\to w_{i_s}$ in the given order. Only those words of V_T^* are in $L(G)$ which are obtained by applications of the matrices.

Indian periodically time-variant grammars. An indian periodically time-variant grammar is a construct $G=(V_N, V_T, P, S, f)$, where f is a mapping $\mathbf{N}\to\mathfrak{P}(P)$ such that $f(i+j)=f(i)$ where m and j are fixed and $i>m$ is arbitrary. The derivation is regulated by the condition that the production used in the k-th step has to be in the set $f(k)$.

Indian random context grammar. The productions of an indian random context grammar $G=(V_N, V_T, P, S)$ are of the form

$$A\to w, R, Q$$

where R and Q are subsets of V_N. Such a production is only applicable on a word $x=x_1Ax_2Ax_3\dots x_{n-1}Ax_n$ if $x_1x_2\dots x_{n-1}x_n$ contains no letter of R and contains all letters of Q.

Indian programmed context free grammar. The productions of an indian programmed grammar $G=(V_N, V_T, P, S)$ are of the form

$$(l)\quad A\to w, F, S$$

where l is the label of the production, F and S are sets of labels. If $A\to w$ is applicable to x, then the next production has to be a rule with a label contained in the success field S. If $A\to w$ is not applicable then the next production has to have a label contained in the failure field F.

All these grammars work in a sequential-parallel way, i.e. only one letter is rewritten in a single derivation step, but all occurrences of this letter are rewritten. Starting from biological motivations EDTOL languages are defined which are also a generalization of indian parallel context free languages. The associated grammars work purely parallel as it is seen from the following definition.

EDTOL system. An EDTOL system is a construct

$$G=(V, V_T, \{P_1, P_2, \ldots, P_r\}, S)$$

where

i) V is a finite set, V_T is a nonempty subset of V,
ii) $S\in V\setminus V_T$,
iii) each P_i is a finite subset of $V\times V^*$, the projection of P_i on the first coordinate is V, and if $A\to w_1$, $A\to w_2$ are in P_i then $w_1=w_2$.

(Usually only $S\in V^+$ is required. It is easy to prove that $S\in V\setminus V_T$ does not restrict the generative power.)

Let $x\in V^+$ and $y\in V^*$. It is said that x directly derives y (also written $x\Rightarrow y$) iff

i) $x=x_1x_2\ldots x_n$, $x_i\in V$,
ii) $y=y_1y_2\ldots y_n$,
iii) there is a $j\in\{1, 2, \ldots, r\}$ such that $x_i\to y_i\in P_j$ for $i=1, 2, \ldots, n$.

The language $L(G)$ is again defined as

$$L(G) = \{x\colon S \overset{*}{\Rightarrow} x, x\in V_T^*\}.$$

We use the following notations

$\mathscr{F}$(IM) — family of indian matrix languages,
$\mathscr{F}$(IPTV) — family of indian periodically time-variant languages,
$\mathscr{F}$(IRC) — family of indian random context languages,
$\mathscr{F}$(IPCF) — family of indian programmed context free languages,
$\mathscr{F}$(EDTOL) — family of EDTOL languages.

3. Comparison of the language families

In the following proofs we will often introduce new alphabets. We make the next convention: If $U\subseteq V$ and $V_U'=\{x'\colon x\in U\}$ is a new alphabet then $w'= =x_1''x_2''\ldots x_n''$ for $w=x_1x_2\ldots x_n$ where

$$x_i''=\begin{cases} x_i' & x_i\in U \\ x_i & x_i\notin U. \end{cases}$$

Let min (w) denote the set of letters occurring in w.

Lemma 1. $\mathscr{F}$(EDTOL)$\subseteq\mathscr{F}$(IM)

Proof. Let $G=(V, V_T, \{P_1, P_2, \ldots, P_r\}, S)$ be an EDTOL system. Let $w_{x,P}$ be the right side of the production with the left side x in $P\in\{P_1, P_2, \ldots, P_r\}$. Further we put $f(U, P)=\bigcup_{x\in U} \min(w_{x,P})$ for $U\subseteq V$.

For a subset $U\subseteq V$ we introduce a new alphabet $V_U=\{x_U\colon x\in U\}$. Now we define the following matrices for $U=\{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\}\subseteq V$ and $P\in\{P_1, P_2, \ldots, P_r\}$,

$$P_U = [(x_{i_1})_U \to (w_{x_{i_1},P})_{f(U,P)}, (x_{i_2})_U \to (w_{x_{i_2},P})_{f(U,P)}, \ldots, (x_{i_k})_U \to (w_{x_{i_k},P})_{f(U,P)}],$$

$$Q_U = [(x_{i_1})_U \to x_{i_1}, (x_{i_2})_U \to x_{i_2}, \ldots, (x_{i_k})_U \to x_{i_k}],$$

and consider the indian matrix grammar

$$H = \left((V \setminus V_T) \cup \bigcup_{U \subseteq V} V_U, V_T, \bigcup_{i=1}^{r} \bigcup_{U \subseteq V} (P_i)_U \cup \bigcup_{U \subseteq V} Q_U, S_{\{S\}}\right).$$

The application of the matrix P_U models the application of P to words w with $U = \min(w)$, i.e. if $w \Rightarrow w'$ is in G then $w_U \Rightarrow (w')_{U'}$ is in H where $U = \min(w)$, $U' = \min(w')$. The application of the matrix Q_U is a translation of w_U in w. If $w \in V_U^*$ then we can apply only the matrices P_U and Q_U. Now it is easy to see that $L(G) = L(H)$. Therefore $L(G) \in \mathscr{F}(\mathrm{IM})$.

Lemma 2. $\mathscr{F}(\mathrm{IM}) \subseteq \mathscr{F}(\mathrm{IRC})$.

Proof. Let $L \in \mathscr{F}(\mathrm{IM})$ and $L = L(G)$ for the indian matrix grammar $G = (V_N, V_T, M, S)$. Let $M = \{m_1, m_2, \dots, m_r\}$, $m_i = [A_{i_1} \to w_{i_1}, A_{i_2} \to w_{i_2}, \dots, \dots, A_{i_s} \to w_{i_s}]$ for $i = 1, 2, \dots, r$. We introduce new alphabets $V^{i,j} = \{x^{i,j} : x \in V\}$, $1 \leqq i \leqq r$, $1 \leqq j \leqq s-1$. Let $V' = \bigcup_{i=1}^{r} \bigcup_{j=1}^{s-1} V^{i,j} \cup V_N$.

Now we can model the application of the matrix m_i by the following sequence of productions of an indian random context grammar

$$\begin{aligned}
&A_{i_1} \to (w_{i_1})^{i,1}, V' \setminus V_N, \emptyset \\
&\quad x \to x^{i,1}, V' \setminus (V_N \cup V^{i,1}), \min\left((w_{i_1})^{i,1}\right) \quad \text{for} \quad x \in V_N, \\
&A_{i_2}^{i,1} \to (w_{i_2})^{i,2}, V \setminus V^{i,1}, \emptyset \\
&x^{i,1} \to x^{i,2}, V \setminus (V^{i,1} \cup V^{i,2}), \min\left((w_{i_2})^{i,2}\right) \quad \text{for} \quad x^{i,1} \in V^{i,1} \\
&\quad \vdots \\
&A_{i_s}^{i,s-1} \to (w_{i_s}), V \setminus V^{i,s-1}, \emptyset \\
&x^{i,s-1} \to x, V \setminus (V^{i,s-1} \cup V_N), \min(w_{i_s}) \quad \text{for} \quad x^{i,s-1} \in V^{i,s-1}.
\end{aligned}$$

If we consider only such productions in our indian random context grammar then we can have also only such derivations which model the application of matrices. Therefore, we generate the same language. Thus $L \in \mathscr{F}(\mathrm{IRC})$.

The above construction works correctly only if we have no rule of form $A_{i_j} \to \lambda$ in the matrices (λ denotes the empty word). If we have a matrix m such that

$$m = [A_1 \to w_1, \dots, A_j \to \lambda, \dots, A_k \to w_k]$$

we use

$$m' = [A_1 \to w_1, \dots, A_j \to B_j, \dots, A_k \to w_k, B_j \to \lambda]$$

instead of m, where B_j is a new nonterminal. It is easy to see that this modification do not change the generative capacity and that our construction works also in the modified case.

Lemma 3. $\mathscr{F}(\mathrm{IRC}) \subseteq \mathscr{F}(\mathrm{IPCF})$.

Proof. Let $G = (V_N, V_T, P, S)$, $V_N = \{A_1, A_2, \dots, A_n\}$. We give a possibility to model a production of G by rules of an indian programmed context free grammar.

Let $A_j \to w$, $\{A_{i_1}, A_{i_2}, \dots, A_{i_s}\}$, $\{A_{j_1}, A_{j_2}, \dots, A_{j_t}\}$ be a rule of G. Consider a new alphabet $V' = \{x' : x \in V_N\}$ associated with the rule and the following diagram of IPCF-productions (an arc labelled by F connects a production with its failure field, and an arc labelled by S connects it with its success field).

It is obvious that we can only simulate rules of G because we have associated the primed alphabets with the productions of G. This proves $L(G) \in \mathscr{F}(\text{IPCF})$.

Lemma 4. $\mathscr{F}(\text{IPCF}) \subseteqq \mathscr{F}(\text{EDTOL})$.

Proof. Let $L \in \mathscr{F}(\text{IPCF})$, $L = L(G)$ for an indian programmed context free grammar $G = (V_N, V_T, P, S)$. Let R be the set of labels of the productions of P. With each subset I of R we associate new alphabets $V_I = \{x_I : x \in V_N\}$ and $V'_I = \{x'_I : x \in V_N\}$. We define tables $P_{l,I,i}$ for any $l \in I$, $I \subseteqq R$, $i \in \{1, 2, 3\}$ in the following way: If (l) $A \to w, F, S$ is a rule and $l \in I$ then put

$$P_{l,I,1} = \{x_I \to x_F : x \neq A\} \cup \{A_I \to f,\ f \to f,\ S \to S\} \cup \{x \to x : x \in V_T\},$$

$$P_{l,I,2} = \{x_I \to x_I : x \neq A\} \cup \{A_I \to A'_I,\ f \to f,\ S \to S\} \cup \{x \to x : x \in V_T\},$$

$$P_{l,I,3} = \{x_I \to x_S : x \neq A\} \cup \{A'_I \to w_S,\ f \to f,\ S \to S\} \cup \{x \to x : x \in V_T\}.$$

Using productions from sets of these types we can only generate words which contain only letters of V_T and V_I for a certain I with exception of one letter which can be in V'_I. If we have such a word we can apply only tables $P_{l,I,i}$. Further $P_{l,I,1}$ models the case that A_I does not occur and produces a word which consist of terminals and nonterminals of the alphabet associated with the failure field. The other two tables model the application of $A \to w$, and we get a word with nonterminals of the alphabet associated with the success field.

Let $I_1, I_2, \dots, I_r$ be the sets containing labels whose production has the left side S, and put

$$Q_i = \{S \to S_{I_i},\ f \to f\} \cup \Big\{x \to x : x \in V_T \cup \bigcup_{I \subseteqq R} (V_I \cup V'_I)\Big\}.$$

Then the EDTOL system

$$H = \Big(\{S, f\} \cup V_T \cup \bigcup_{I \subseteqq R} (V_I \cup V'_I),\ V_T,\ \{P_{l,I,i} : I \subseteqq R,\ i \in \{1, 2, 3\},\ l \in I\} \cup \cup \{Q_i : 1 \leqq i \leqq r\},\ S\Big)$$

generates L. Thus $L \in \mathscr{F}(\text{EDTOL})$.

Lemma 5. $\mathscr{F}(\text{IM}) \subseteqq \mathscr{F}(\text{IPTV})$.

Proof. The proof of [10], Theorem 11 works also in the indian parallel case.

Lemma 6. $\mathscr{F}(\text{IPTV}) \subseteqq \mathscr{F}(\text{EDTOL})$.

Proof. Let $L = L(G)$ for the indian periodically time-variant grammar $G = (V_N, V_T, P, S, f)$ where $f(i+j) = f(i)$ for $i > m$. We introduce new alphabets $V^{(k)} = \{x^{(k)} : x \in V_N\}$ for $1 \leqq i \leqq m+j-1$. For $A \to w = p \in P$, $p \in f(i)$, $1 \leqq i < m+j-1$ we define the tables

$$P_{i,p} = \{x^{(i)} \to x^{(i+1)} : x \neq A\} \cup \{A^{(i)} \to w^{(i+1)}\} \cup \{x \to x : x \in V_T\}$$

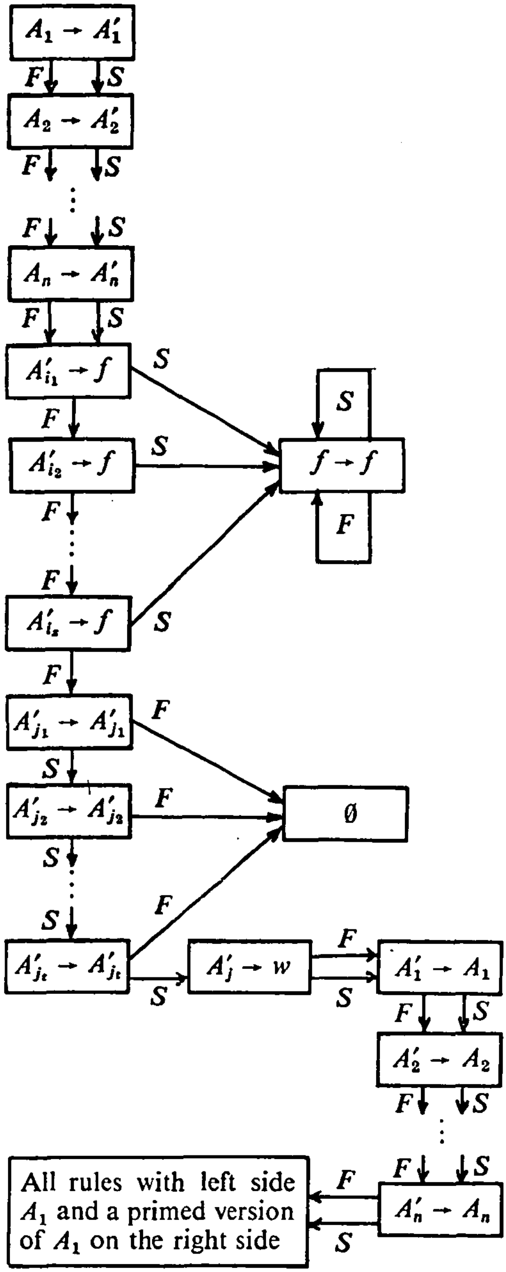

$A_1 \rightarrow A_1'$
F
S
$A_2 \rightarrow A_2'$
$A_n \rightarrow A_n'$
$A_{i_1}' \rightarrow f$
$A_{i_2}' \rightarrow f$
$A_{i_s}' \rightarrow f$
$f \rightarrow f$
$A_{j_1}' \rightarrow A_{j_1}'$
$A_{j_2}' \rightarrow A_{j_2}'$
$A_{j_t}' \rightarrow A_{j_t}'$
∅
$A_j' \rightarrow w$
$A_1' \rightarrow A_1$
$A_2' \rightarrow A_2$
$A_n' \rightarrow A_n$
All rules with left side A_1 and a primed version of A_1 on the right side

Fig. 1

and

$$P_{m+j-1,p} = \{x^{(m+j-1)} \to x^{(m)}\colon x \neq A\} \cup \{A^{(m+j-1)} \to w^{(m)}\} \cup \{x \to x\colon x \in V_T\}.$$

It is easy to see that the EDTOL system

$$H = \left(V_T \cup \bigcup_{k=1}^{m+j-1} V^{(k)}, V_T, \{P_{i,p}\colon 1 \leqq i \leqq m+j-1,\ p \in f(i)\}, S^{(1)}\right)$$

generates L.

We say that a grammar is λ-free if it contains no production with the empty word λ at the right side. The family of X-languages generated by λ-free X-grammars is denoted by $F_\lambda(X)$. As usual we identify languages which differ only in the empty word.

Theorem 1. $\mathcal{F}(\text{EDTOL}) = \mathcal{F}(\text{IM}) = \mathcal{F}(\text{IRC}) = \mathcal{F}(\text{IPCF}) = \mathcal{F}(\text{IPTV}) = \mathcal{F}_\lambda(\text{EDTOL}) = \mathcal{F}_\lambda(\text{IM}) = \mathcal{F}_\lambda(\text{IRC}) = \mathcal{F}_\lambda(\text{IPCF}) = \mathcal{F}_\lambda(\text{IPTV})$.

Proof. The first row follows directly by Lemma 1—6. Further $\mathcal{F}(\text{EDTOL}) = \mathcal{F}_\lambda(\text{EDTOL})$ is known and all our constructions in Lemma 1—4 and 6 preserve λ-freeness. If the matrix grammar in the proof of [10], Theorem 11 is λ-free then we modify the proof in the following way: The new symbols Y_j^i are not catenated with P_j^i, the last letter of P_j^i has to be in a new "primed" alphabet and the last rules have to change the letter into a "not primed" letter.

By Theorem 1, we get some information on properties of the extensions of indian parallel context free grammars, because we have knowledge on $\mathcal{F}(\text{EDTOL})$.
— In [3], closure properties under AFL-operations are given.
— There are context free languages which are not in $\mathcal{F}(\text{EDTOL})$.
— It is known that the families of matrix languages, programmed context free languages, random context languages and periodically time-variant languages properly contain $\mathcal{F}(\text{EDTOL})$. Thus the indian parallel restriction reduces the generative capacity of the considered extensions.
— The proof of v. Solms [14] works also in the indian parallel and deterministic case. This proves that indian random context grammars of special type generate already all indian random context languages.

A further language family which is equal to the above families is given in [2], Theorem 2.

Finally we want to mention without proof that all our language families also coincide with the family of indian unordered scattered context languages, which are the indian parallel version of the unordered scattered context grammars of [5].

TECHNOLOGICAL UNIVERSITY OTTO VON GUERICKE
DEPARTMENT OF MATHEMATICS AND PHYSICS
MAGDEBURG, GRD

References

[1] Abraham, S., Some questions of phrase structure grammars, *Comput. Linguistics*, v. 4, 1965, pp. 61—70.

[2] Dassow, J., ETOL systems and compound grammars, Rostock, Math. Colloq., v. 11, 1979, pp. 41—46.

[3] Ehrenfeucht, A. & G. Rozenberg, On images of inverse homomorphisms, *Automata, Languages, Development*, Ed. A. Lindenmayer, G. Rozenberg, North-Holland, 1976.

[4] Mayer, O., Some restrictive devices for context-free grammars, *Inform. and Control*, v. 20, 1972, pp. 69—92.

[5] Milgram, D. & A. Rozenfeld, A note on scattered context grammars, *Inform. Process. Lett.*, v. 1, 1971, pp. 47—50.

[6] Penttonen, M., ETOL-grammars and N-grammars, *Inform. Process. Lett.*, v. 4, 1975, pp. 11—13.

[7] Rosenkrantz, D. J., Programmed grammars and classes of formal languages, *J. Assoc. Comput. Mach.*, v. 10, 1969, pp. 107—131.

[8] Rozenberg, G., More on ETOL systems versus random context grammars, *Inform. Process. Lett.*, v. 5, 1976, pp. 102—106.

[9] Rozenberg, G. & D. Vermeir, Context-free programmed grammars and ETOL systems, *Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science, v. 45, 1976, pp. 482—487.

[10] Salomaa, A., On grammars with restricted use of productions, *Ann. Acad. Sci. Fenn. Ser. A I Math.*, No. 454, 1969.

[11] Salomaa, A., *Formal languages*, Academic Press, 1973.

[12] Siromoney, R. & K. Krithivasan, Parallel context-free languages, *Inform. and Control*, v. 24, 1974, pp. 155—162.

[13] van der Walt, A. P. J., Random context languages, *Information Processing* 1971, North-Holland, 1972, pp. 68—66.

[14] von Solms, S. H., On TOL languages over terminals, *Inform. Process. Lett.*, v. 3, 1975, pp. 69—70.

(Received June 6, 1979)

О вентильных схемах

О. Б. Лупанов

Вентильные схемы являются одним из основных модельных объектов, на которых изучаются закономерности сложности управляющих систем. Целесообразность их изучения объясняется, с одной стороны, их простотой, и, с другой стороны, возможностью использовать методы и конструкции, разработанные для них, в случае «более сильных» классов управляющих систем.

Вентильную схему можно определить как ориентированный граф, в котором выделено некоторое подмножество вершин — множество полюсов — и эти вершины занумерованы. С каждой вентильной схемой S связывается матрица из нулей и единиц $A=\|a_{ij}\|$ — матрица проводимостей ($a_{ij}=1$ тогда и только тогда, когда в S имеется ориентированный путь из полюса i в полюс j). Очевидно, что матрица проводимостей любой вентильной схемы является транзитивной. Важным классом вентильных схем являются такие, в которых полюса разбиты на два подмножества: «входные» — с номерами $P=\{1, \ldots, p\}$ и «выходные» — с номерами $Q=\{p+1, \ldots, p+q\}$ и на матрицу проводимостей наложено дополнительное ограничение: $a_{ij}=0$, если $i\neq j$ и либо $i\in P, j\in P$, либо $i\in Q, j\in Q$, либо $i\in Q, j\in P$. В этом случае система проводимостей полностью определяется подматрицей данной матрицы, имеющей p строк и q столбцов; ее обычно и называют матрицей проводимостей.

Одной из основных задач в теории вентильных схем является задача синтеза — построение по данной матрице A вентильной схемы, имеющей в качестве матрицы проводимостей матрицу A. Для решения этой задачи существует тривиальный способ, сводящийся к непосредственному соединению полюсов вентилями. Однако этот способ является неэкономным. Поэтому задача синтеза уточняется — требуется указать метод построения схем, который, с одной стороны, является не очень трудоемким и, с другой стороны, позволяет строить достаточно простые схемы. Для характеристики сложности схем вводится функция $B(p, q)$ — минимальное число вентилей, достаточное для реализации любой матрицы с p строками и q столбцами (функция Шеннона). Пусть $B_r(p, q)$ — аналогичная функция для схем глубины r (глубина схемы — максимальная длина цепи от входа к выходу).

Первые результаты об оценках функций $B(p, q)$ и $B_r(p, q)$ были получены в работе автора [1].

Теорема 1 [1]. *Пусть выполнены условия**

а) $p \to \infty$;

б) $p \leqq q$;

в) $\dfrac{\log q}{p} \to 0.$

Тогда

$$B_2(p, q) \sim \frac{pq}{\log q}.$$

Следствие 1. При условиях а), б), в)

$$\frac{pq}{\log (pq)} \lesssim B(p, q) \lesssim \frac{pq}{\log q}.$$

Следствие 2. При условиях а), б), в) и дополнительном условии

г$_0$) $\dfrac{\log p}{\log q} \to 0$

выполняются соотношения

$$B(p, q) \sim B_2(p, q) \sim \frac{pq}{\log q}.$$

Конструкция, использованная в методе синтеза вентильных схем глубины 2, легла в основу методов синтеза «более сильных» классов управляющих систем (контактных схем, схем из функциональных элементов, автоматов и т. д.) — см., например, [2].

Уточнение оценок для $B(p, q)$ было получено Э. И. Нечипоруком.

Теорема 2 [3, 4]. *Пусть выполнены условия* а) *и* б) *теоремы* 1, *а также условие*

г$_c$) $\lim \dfrac{\log p}{\log q} = \dfrac{\mu}{\mu(\varrho - 1) + \varrho}$, *где* μ, ϱ *— целые числа, большие нуля.*

Тогда

$$B(p, q) \sim B_3(p, q) \sim \frac{pq}{\log (pq)}.$$

Замечание. Условие г$_c$) включает важный в приложениях случай, когда $p \asymp q$.

Случай, когда не выполняется условие в), был исследован В. А. Орловым. Оказалось, что в этом случае функция $B_2(p, q)$ ведет себя «ступенчатообразно». Именно справедливы утверждения.

* Здесь и ниже имеется в виду, что p и q являются функциями некоторого параметра n, и имеются в виду асимптотические соотношения при $n \to \infty$; log означает логарифм по основанию 2.

Теорема 3 [5]. *Пусть k произвольное фиксированное целое положительное число. Тогда*

$$B_2([k\log q], q) \sim (k+1)q.$$

Теорема 4 [5]. *Пусть выполнены условия*

а′) $q \to \infty$,

в$_\alpha$) $\lim \frac{p}{\log q} = \alpha$, *причем* $\alpha > 1$, α *не целое.*

Тогда

$$B_2(p, q) \sim [\alpha+1]q.$$

Теорема 5 [5]. *Пусть* $q \geqq p2^{p-1}-p$. *Тогда*

$$B_2(p, q) = p2^{p-1}-p+q.$$

Теорема 6 [5]. *Пусть выполнено условие* а), *а также условия*

в$_1$) $\frac{p}{\log q} \to 1$,

д) $q \geqq p2^{p-1}-p$.

Тогда

$$B_2(p, q) \sim 2q.$$

Теорема 7 [5]. *Пусть выполнены условия* а), в$_1$), *а также*

д$^-$) $q \leqq 2(2^p-p-1)$.

Тогда

$$B(p, q) \sim B_2(p, q) \sim 2q.$$

Теорема 8 [5]. *Пусть выполнены условия* а) *и*

д$^+$) $q \geqq 2(2^p-p-1)$.

Тогда

$$B(p, q) \sim B_2(p, q) \sim 2\cdot 2^p+q.$$

Наряду с задачей о реализации произвольных матриц (заданных размеров) рассматривался вопрос о реализации матриц из специальных классов и о реализации конкретных матриц.

Пусть $B_r(p, q, \alpha)$ функция Шеннона для матриц с p строками q столбцами, имеющих αpq единиц. Пусть

$$\alpha^* = \min(\alpha, 1-\alpha), \quad H(\alpha) = \alpha\log\frac{1}{\alpha}+(1-\alpha)\log\frac{1}{1-\alpha}.$$

Э. И. Нечипорук доказал следующие утверждения.

Теорема 9 [3, 4]. *Пусть выполнены условия* б), *а также*

а$_\alpha$) $\alpha p \to \infty$,

е$_{\alpha,\varrho}$) $\dfrac{\log q}{\log \dfrac{1}{\alpha}} \to \varrho$, $\varrho > 0$, *целое*

ж$_{\alpha,\varrho}$) $q\alpha^\varrho \to \infty$.

Тогда

$$B_2(p, q, \alpha) \sim \frac{\alpha p q}{\varrho}.$$

Теорема 10 [12, 4]. *Пусть выполнены условия* а), б), *а также*

в$_\alpha$) $H(\alpha)\dfrac{p}{\log q} \to \infty$,

е$_\infty$) $\dfrac{\log q}{\log \dfrac{1}{\alpha^*}} \to \infty$.

Тогда

$$B_2(p, q, \alpha) \sim H(\alpha)\frac{pq}{\log q}.$$

Теорема 11 [4]. *Пусть выполнены условия* а), в$_\alpha$), е$_\alpha$), *а также*

г$_1$) $\log p \sim \log q$.

Тогда

$$B_3(p, q, \alpha) \sim H(\alpha)\frac{pq}{2\log q}.$$

Пусть $B_r^\gamma(p, q)$ — функция Шеннона для не всюду определенных матриц. у которых число определенных элементов — нулей и единиц — равно γpq (остальные $(1-\gamma)pq$ элементов не определены и при реализации матрицы заменяются нулями и единицами так, чтобы реализация была простейшей). Э. И. Нечипорук установил следующий факт.

Теорема 12 [13, 4]. *Пусть выполнены условия* а), б), *а также*

в$^\gamma$) $\dfrac{\gamma p}{\log q} \to \infty$,

е$^\gamma$) $\dfrac{\log q}{\log \dfrac{1}{\gamma}} \to \infty$.

Тогда

$$B_2^\gamma(p, q) \sim \gamma\frac{pq}{\log q}.$$

Конструкция, использованная при доказательстве верхней оценки в этой теоремы, применялась впоследствии многими авторами при реализации не всюду определенных булевых функций в различных классах схем (например, [6—9]).

При изучении сложности реализации конкретных матриц наиболее высокие оценки (порядка $p^{3/2}$ в случае квадратных матриц порядка p) были получены Э. И. Нечипоруком («матрицы без прямоугольников» — см. [10]), а также Т. Г. Тарьяном (матрицы Адамара — см. [11]).

Отметим в заключение некоторые задачи, решение которых, по-видимому, будет связано с созданием новых методов.

1. Получить асимптотическое выражение для $B(p, q)$ в случае $\log p \asymp \log q$ (например, при $\frac{\log p}{\log q} \to \delta$, δ — иррациональное число; т. е. снят ограничение г$_c$) — см. стр. 312).

2. Получить асимптотическую формулу для функции Шеннона $B(p)$ в случае схем, реализующих произвольные транзитивные матрицы с p строками и p столбцами (в которых не выделены специально входы и выходы); оценки

$$c_1 \frac{p^2}{\log p} \leqq B(p) \leqq c_2 \frac{p^2}{\log p}$$

получаются легко на основе, например, теоремы 1.

3. Построить «эффективно» последовательность матриц порядка p, которые реализуются лишь со сложностью, существенно большей, чем $p^{3/2}$.

Литература

[1] Лупанов, О. Б., О вентильных и контактно-вентильных схемах, *ДАН СССР*, т. 111, № 6, 1956, стр. 1171—1174.

[2] Лупанов, О. Б. О синтезе некоторых классов управляющих систем, *Проблемы кибернетики*, вып. 10, 1963, стр. 63—97.

[3] Нечипорук, Э. И., О вентильных схемах, *ДАН СССР*, т. 148, № 1, 1963, срт. 50—53.

[4] Нечипорук, Э. И., О топологических принципах самокорректирования, *Проблемы кибернетики*, вып. 21, 1969, стр. 5—102.

[5] Орлов, В. А., Реализация «узких» матриц вентильными схемами, *Проблемы кибернетики* вып. 22, 1970, стр. 45—52.

[6] Шоломов, Л. А., О реализации недоопределенных булевых функций схемами из функциональных элементов, *Проблемы кибернетики*, вып. 21, 1969, стр. 215—226.

[7] Мадатян, Х. А., О реализации не всюду определенных матриц заданной «густоты» вентильными схемами глубины два, *Кибернетика*, Киев, № 6, 1973, стр. 12—15.

[8] Липатов, Е. П., Об одном случае неравномерного локального кодирования, *Проблемы кибернетики*, вып. 26, 1973, стр. 95—107.

[9] Pippenger, N., Information theory and the complexity of Boolean functions, *Math. Systems Theory*, v. 10, 1976/77, No. 2, pp. 129—167.

[10] Нечипорук, Э. И., Об одной булевской матрице, *Проблемы кибернетики*, вып. 21, 1969, стр. 237—240.

[11] Tarján, T. G., Complexity of lattice-configurations, *Studia Sci. Math. Hungar.*, v. 10, 1975, pp. 203—211.

[12] Нечипорук, Э. И., О синтезе вентильных схем, *Проблемы кибернетики*, вып. 9, 1963, стр. 37—44.

[13] Нечипорук, Э. И., О сложности вентильных схем, реализующих булевские матрицы с неопределенными элементами, *ДАН СССР*, т. 163, № 1, 1965, стр. 40—42.

(Поступило 2-ого августа 1979 г.)

Об асимптотических оценках сложности управляющих систем

О. Б. Лупанов

Этот доклад содержит обзор некоторых результатов в области асимптотической теории сложности управляющих систем. Здесь, с одной стороны, будет кратко охарактеризовано общее состояние теории и, с другой стороны, несколько более подробно будут изложены некоторые результаты последних лет.

Асимптотическая теория синтеза управляющих систем фактически началась с работ Шеннона [1, 2]. С. В. Яблонским было введено весьма общее понятие управляющей системы (УС) и были сформулированы основные задачи теории управляющих систем [3]. Полностью привести здесь определение УС не представляется возможным. Укажем лишь некоторые примеры УС. Управляющими системами являются релейно-контактные схемы, цифровые вычислительные машины, программы (для ЭВМ), арифметические формулы и т. д. Все эти объекты характеризуются тем, что они обладают некоторой структурой, схемой S, и реализуют определенную функцию f. С каждым множеством УС естественным образом связывается множество $\mathfrak{S}$ их схем и множество $\mathfrak{F}$ их функций.

Одной из основных задач теории УС является задача синтеза: по функции f из $\mathfrak{F}$ требуется найти УС, схема которой реализует f. В настоящее время для исследования разных вопросов, связанных с задачей синтеза УС (как и для других задач теории УС) характерно рассмотрение конкретных классов УС (модельных объектов). К таким классам можно отнести следующие.

1. Вентильные схемы.
2. Дизъюнктивные нормальные формы.
3. Формулы, являющиеся суперпозициями базисных формул.

В частности, формулы в базисе $\&$, $\vee$, $\neg$.

4. Схемы из функциональных элементов.
5. Контактные схемы.
6. Автоматы (логические сети).
7. Алгоритмы.

Всюду в дальнейшем речь будет идти о сложности задания, а не о сложности вычисления.

Задача синтеза решается, как правило, неоднозначно. Для уточнения постановки задачи вводится мера сложности схем $L(S)$ функционал, удовлетворяющий некоторым естественным условиям. Например, $L(S)$ в случае формул можно определить как число вхождений символов переменных, а в случае схем из функциональных элементов как число элементов схемы; или несколько более общим образом: функциональным элементам (базисным формулам) приписываются веса, $L(S)$ определяется как сумма весов всех элементов схемы (всех вхождений символов базисных формул соответственно). После этого задача о синтезе уточняется так: для любой функции f из $\mathfrak{F}$ требуется найти схему S из $\mathfrak{S}$, реализующую f и такую, что $L(S)$ минимально («минимальную схему»). Это минимальное значение будем обозначать через $L(f)$.

В большинстве случаев существует алгоритм для построения минимальных схем, основанный на переборе всех схем определенной сложности. Однако трудоемкость этого алгоритма весьма велика, и он практически нереализуем при использовании ЭВМ даже в случае функций от сравнительно небольшого числа переменных (5—10). Все попытки найти более эффективные алгоритмы пока не привели к цели. Более того, С. В. Яблонским была высказана гипотеза (и получены первые результаты в направлении обоснования этой гипотезы), что «полный перебор» в этих задачах необходим [4]. Поэтому задачу синтеза приходится уточнять дальше. Один из подходов к этому принадлежит К. Шеннону [2]. При этом подходе отказываются от нахождения минимальной схемы для каждой функции. Вместо этого рассматривают задачу синтеза для целого класса функций (например, для класса $\mathfrak{F}^{(n)}$ булевых функций от n переменных); кроме того, требование минимальности заменяется требованием «почти минимальности». Более точная постановка задачи состоит в следующем. Пусть $L(n)=\max L(f)$, где максимум берется по всем функциям $f(x_1, \ldots$ $\ldots, x_n)$ из $\mathfrak{F}^{(n)}$. Функция $L(n)$ получила название функции Шеннона. Требуется найти алгоритм, который для каждой функции $f(x_1, \ldots, x_n)$ строит схему S такую, что $L(S) \lesssim L(n)$; трудоемкость этого алгоритма должна быть существенно меньше трудоемкости полного перебора.

Первые результаты в этом направлении были получены К. Шенноном [2]. Им был предложен оптимальный по порядку алгоритм синтеза контактных схем и получены оценки

$$\frac{2^n}{n} \lesssim L(n) \lesssim \frac{2^{n+2}}{n}.$$

Асимптотически наилучший алгоритм был построен автором, и тем самым установлена асимптотика функции Шеннона: $L(n) \sim \frac{2^n}{n}$ [5, 6]. Автором были построены также асимптотически наилучшие алгоритмы синтеза формул и схем из функциональных элементов в произвольном конечном базисе [7, 8]. Асимптотики функций Шеннона для этих случаев имеют соответственно вид

$$L(n) \sim \varrho \frac{2^n}{\log_2 n}, \quad L(n) \sim \varrho \frac{2^n}{n},$$

где ϱ — константа, простым способом определяемая по базису.

На основе этих методов позже появились аналогичные асимптотические результаты для других классов управляющих систем. Среди них отметим работу В. А. Кузьмина, установившего асимптотику функции Шеннона для сложности реализации булевых функций некоторыми типами алгоритмов (нормальные алгорифмы, машины Тьюринга) и выяснившего зависимость этой асимптотики от числа букв используемого алфавита [9].

Исследовался также вопрос о влиянии на сложность реализации различных дополнительных требований, предъявляемых к схемам. Здесь в первую очередь следует отметить результаты Э. И. Нечипорука о реализации функций в базисах, некоторые элементы которых имеют нулевые веса [10], и о синтезе самокорректирующихся контактных схем [11]. Сюда же можно отнести результат автора об асимптотике функции Шеннона для схем из пороговых элементов:

$$L(n) \sim 2\left(\frac{2^n}{n}\right)^{1/2};$$

в случае реализации систем m функций асимптотика имеет вид (при условии $\left.\frac{m}{2^n}\to 0\right)$

$$L(n, m) \sim 2\left(\frac{m\,2^n}{n-\log_2 m}\right)^{1/2} \quad [12].$$

Здесь $L(S)$ — число элементов в схеме S; для случая, когда $L(S)$ равно сумме модулей весов входов элементов, асимптотика была получена ранее Е. Ю. Захаровой [13].

В случае схем в автоматных базисах положение оказалось более сложным. Для некоторых частных случаев удалось получить асимптотические формулы для функции Шеннона (Б. А. Трахтенброт [14], То Суан Зунг [15]). В общем же случае задача об асимптотическом поведении функции Шеннона для произвольного автоматного базиса оказалась алгоритмически неразрешимой (В. А. Орлов [16]). Для частного случая (реализация булевых функций схемами в автоматных базисах) результат В. А. Орлова может быть сформулирован следующим образом. Существует бесконечное множество автоматных базисов $\{\mathfrak{B}\}$, такое что для базиса $\mathfrak{B}$ функция Шеннона имеет асимптотику вида

$$L(n) \sim C_{\mathfrak{B}}\frac{2^n}{n}$$

($C_{\mathfrak{B}}$ константа, зависящая от базиса $\mathfrak{B}$), но не существует алгоритма, определяющего по любому базису из $\{\mathfrak{B}\}$ значение константы $C_{\mathfrak{B}}$.

Результаты асимптотической теории показывают, что поведение функции Шеннона слабо зависит от класса УС. Кроме того, оказывается, что почти все функции из $\mathfrak{F}^{(n)}$ имеют почти одинаковую сложность, асимптотически равную сложности самой сложной функции.

Из приведенных выше оценок видно, что почти все функции допускают лишь очень сложную реализацию и поэтому практически недоступны. Поэтому возникает вопрос о выделении классов функций, реализуемых более просто. Примеры таких классов известны давно, со времени первых работ по синтезу.

Затем С. В. Яблонским [4] было построено и изучено континуальное семейство функций, допускающих более простую схемную реализацию, чем большинство функций. Это классы функций, замкнутые относительно операции подстановки констант вместо (некоторых) переменных. Каждый класс характеризуется некоторым числовым параметром σ, отражающим «мощность» класса ($0 \leqq \sigma \leqq 1$). Для этих классов при $\sigma \neq 0$ С. В. Яблонским были построены асимптотически наилучшие методы синтеза и получены асимптотики функции Шеннона.

Впоследствии автором был предложен один общий подход к синтезу схем — принцип локального кодирования [17]. Этот подход позволяет по описанию класса функций (с соблюдением некоторых специальных требований) строить асимптотически наилучший метод синтеза для функций из этого класса. С помощью этого принципа оказалось возможным единым способом получить методы синтеза для известных классов функций, а также для многих новых классов. Асимптотика сложности схем в конечных базисах для функций из этих классов определяется числом M_n функций $f(x_1, \ldots, x_n)$ в этом классе и имеет вид $\varrho \frac{\log M_n}{\log_2 \log_2 M_n}$. Эта функция может принимать значения от величин, близких к n, до $\frac{2^n}{n}$. Принцип локального кодирования особенно удобен при применении к достаточно богатым классам УС (схемы из функциональных элементов, автоматы, алгоритмы). Интересный вариант принципа локального кодирования описан в работе Е. П. Липатова [18].

Как уже отмечалось, большинство булевых функций допускает лишь очень сложную схемную реализацию. Однако доказательство этого факта является неэффективным. Первая «эффективная» нелинейная нижняя оценка была получена Б. А. Субботовской для сложности реализации линейной функции от n переменных формулами в базисе &, V, ⌉; эта оценка имеет вид $Cn^{3/2}$ [19]. В. М. Храпченко усилил эту оценку до n^2; тем самым установлен порядок сложности [20]. Им же предложен общий метод установления квадратичных нижних оценок для сложности формул в базисе &, V, ⌉ [21]. Близкие к квадратичным нижние оценки в «более сильных» классах УС (формулы в произвольном базисе; контактные схемы) были получены Э. И. Нечипоруком [22]. Общий метод получения нелинейных нижних оценок для формул в произвольном базисе предложен Л. Ходесом и Е. Шпекером [23]. В «более слабых» классах УС (формулы в функционально неполных базисах) удается получать нижние оценки порядка n^c, где C — произвольная константа (Э. И. Нечипорук [24], М. М. Рохлина [25]). До сих пор не удалось получить ни одной нелинейной нижней оценки для схем из функциональных элементов (в полном базисе булевых функций). Для неполных базисов в k-значной логике удается получать экспоненциальные оценки в случае схем из функциональных элементов (Г. А. Ткачев [26]).

В связи с установлением некоторых общих закономерностей сложности особый интерес сейчас, по-видимому, представляет изучение различных мер сложности и связи между ними, выявление различных нетривиальных ситуаций, а также установление более общих закономерностей.

Важной мерой сложности, связанной с числом элементов схемы, является ее глубина, т. е. максимальная длина цепочки элементов, соединяющей вход схемы с ее выходом. Соответствующую функцию Шеннона будем обозначать через $T(n)$. Из анализа известных методов синтеза и из нижней оценки для $L(n)$ легко получается асимптотическая формула для $T(n)$. Например, для базиса &, $\vee$, $\neg$ справедливо соотношение $T(n) \sim n$. Более точная формула была получена лишь совсем недавно С. Б. Гашковым [27]

$$T(n) = n - \log_2 \log_2 n + O(1)$$

(нижняя оценка непосредственно следует из нижней оценки для $L(n)$ в случае формул; ранее известные верхние оценки для $T(n)$ таковы:

$$T(n) \leqq n + \log^* n \quad [28], \qquad T(n) \leqq n + O(1) \quad [29]).$$

Глубину схемы можно трактовать как ее задержку; однако, как показал В. М. Храпченко, эти характеристики не всегда совпадают [30]. Полный анализ возможных функций $T(n)$ в случае, когда некоторые базисные элементы имеют нулевую задержку, провел С. А. Ложкин. Оказалось, что возможны лишь три типа поведения функции $T(n)$:

$$T(n) \sim \tau n; \quad T(n) \sim \alpha \log n; \quad T(n) = c \quad (n \geqq n_0) \quad [31].$$

Кроме того, оказалось, что во многих случаях возможно одновременное (т. е. в одной схеме) достижение асимптотики сложности и задержки [32, 33].

Из работ последних лет по исследованию других мер сложности следует отметить результаты О. М. Касим-Заде [34]. Он продолжил начатые М. Н. Вайнцвайгом [38] исследования так называемой мощности схем из функциональных элементов (мощность схемы — это максимальное число ее элементов, имеющих на выходе единицу). Пусть $E_{\mathfrak{A}}(n)$ соответствующая функция Шеннона для базиса $\mathfrak{A}$. Основные результаты О. М. Касим-Заде состоят в следующем

1. Для почти всех (в некотором естественном смысле) базисов $\mathfrak{A}$

$$E_{\mathfrak{A}}(n) \asymp n.$$

2. Существует бесконечно много различных по порядку функций $E_{\mathfrak{A}}(n)$. Например, для любого целого положительного m существует базис $\mathfrak{A}_m$, для которого

$$E_{\mathfrak{A}_m}(n) \asymp \left(\frac{2^n}{n}\right)^{1/m}.$$

3. Для любого конечного базиса $\mathfrak{A}$ имеет место лишь одна из возможностей: либо $\log E_{\mathfrak{A}}(n) \asymp n$, либо $\log E_{\mathfrak{A}}(n) \asymp \log n$; при этом по базису эффективно устанавливается, какая из возможностей имеет место.

4. Для базиса &, $\vee$, $\neg$ возможно одновременное достижение (в одной схеме) асимптотики сложности и порядка мощности.

В заключение отметим, что обзоры такого рода (а также по более широкому кругу вопросов) делались раньше [35—37].

Литература

[1] Shannon, C. E., A symbolic analysis of relay and switching circuits, *Trans. Amer. Electr. Eng.*, v. 57, 1938, pp. 713—722.

[2] Shannon, C. E., The synthesis of two-terminal switching circuits, *Bell System Tech. J.*, v. 28, 1949, No. 1, pp. 59—98.

[3] Яблонский, С. В., Основные понятия кибернетики, *Проблемы кибернетики*, вып. 2, 1959, стр. 7—38.

[4] Яблонский, С. В., Об алгоритмических трудностях синтеза минимальных контактных схем, *Проблемы кибернетики*, вып. 2, 1959, стр. 75—121.

[5] Лупанов, О. Б., О синтезе контактных схем, *ДАН СССР*, т. 119, № 1, 1958, стр. 23—26.

[6] Лупанов, О. Б., О синтезе некоторых классов управляющих систем, *Проблемы кибернетики*, вып. 10, 1963, стр. 63—97.

[7] Лупанов, О. Б., О сложности реализации функций алгебры логики формулами, *Проблемы кибернетики*, вып. 3, 1960, стр. 61—80.

[8] Лупанов, О. Б., Об одном методе синтеза схем, Известия вузов, *Радиофизика*, т. 1, № 1, 1958, стр. 120—140.

[9] Кузьмин, В. А., Реализация функций алгебры логики автоматами, нормальными алгорифмами и машинами Тьюринга, *Проблемы кибернетики*, вып. 13, 1965, стр. 75—96.

[10] Нечипорук, Э. И., О сложности схем в некоторых базисах, содержащих нетривиальные элементы с нулевыми весами, *Проблемы кибернетики*, вып. 8, 1962, стр. 123—160.

[11] Нечипорук, Э. И., О топологических принципах самокорректирования, *Проблемы кибернетики*, вып. 21, 1969, стр. 5—102.

[12] Лупанов, О. Б., О синтезе схем из пороговых элементов, *Проблемы кибернетики*, вып. 26, 1973, стр. 109—140.

[13] Захарова, Е. Ю., О синтезе схем из пороговых элементов, *Проблемы кибернетики*, вып. 9, 1963, стр. 317—319.

[14] Трахтенброт, Б. А., Асимптотическая оценка сложности логических сетей с памятью, *ДАН СССР*, т. 127, № 2, 1959, стр. 281—284.

[15] То Суан Зунг, Об асимптотических закономерностях сложности автоматов из некоторых классов, *Проблемы кибернетики*, вып. 22, 1970, стр. 5—43.

[16] Орлов, В. А., О сложности реализации ограниченно-детерминированных операторов схемами в автоматных базисах, *Проблемы кибернетики*, вып. 26, 1973, стр. 141—182.

[17] Лупанов, О. Б., Об одном подходе к синтезу управляющих систем—принципе локального кодирования, *Проблемы кибернетики*, вып. 14, 1965, стр. 31—110.

[18] Липатов, Е. П., Об одном случае неравномерного локального кодирования, *Проблемы кибернетики*, вып. 26, 1973, стр. 95—107.

[19] Субботовская, Б. А., О реализации линейных функций формулами в базисе $\vee$, &, $-$, *ДАН СССР*, т. 136, № 3, 1961, стр. 553—555.

[20] Храпченко, В. М., О сложности реализации линейной функции в классе п-схем, *Мат. заметки*, т. 9, № 1, 1971, стр. 35—40.

[21] Храпченко, В. М., Об одном методе получения нижних оценок сложности п-схем, *Мат. заметки*, т. 10, № 1, 1971, стр. 83—92.

[22] Нечипорук, Э. И., Об одной булевской функции, *ДАН СССР*, т. 169, № 4, 1966, стр. 765—767.

[23] Hodes, L., E. Specker, Lengths of formulas and elimination of quantifiers I. Proc. Logic Colloquium, Hannover 1966, *Contributions to mathematical logic*, North-Holland, Amsterdam, 1968, pp. 175—188.

[24] Нечипорук, Э. И., О реализации дизъюнкции и конъюнкции в некоторых монотонных базисах, *Проблемы кибернетики*, вып. 23, 1970, стр. 291—293.

[25] Рохлина, М. М., О схемах, повышающих надежность, *Проблемы кибернетики*, вып. 23, 1970, стр. 295—301.

[26] Ткачев, Г. А., О сложности реализации одной последовательности функций k-значной логики, Вестник Московского университета, Вычислительная математика и кибернетика, № 1, 1977, стр. 45—57.

[27] Гашков, С. Б., О глубине булевых функций, *Проблемы кибернетики*, вып. 34, 1978, стр. 265—268.

[28] Spiza, P. M., On the time necessary to compute switching functions, *IEEE Trans. Comput.*, C—20, No. 1, 1977, pp. 104—105.

[29] McColl, W. F., M. S. Paterson, The depth of all Boolean functions, *SIAM J. Comput.*, v. 6, No. 2, 1977, pp. 373—380.

[30] Храпченко, В. М., Различие и сходство между задержкой и глубиной, *Проблемы кибернетики*, вып. 35, 1979, стр. 141—168.

[31] Ложкин, С. А., Асимптотическое поведение функций Шеннона для задержек схем из функциональных элементов, *Мат. заметки*, т. 19, № 6, 1976, стр. 939—951.

[32] Лупанов, О. Б., О схемах из функциональных элементов с задержками, *Проблемы кибернетики*, вып. 23, 1970, стр. 43—81.

[33] Loshkan, S. A., Über die Synthese von Schemata aus Funktional-elementen mit Verzögerung, *Wiss. Z. Humboldt-Univ. Berlin Math.-Natur. Reihe*, v. 24, 1975, No. 6, pp. 730—732.

[34] Касим-Заде, О. М., Об одной мере сложности схем из функциональных элементов, *ДАН СССР*, т. 250, № 4, 1980, стр. 797—801.

[35] Yablonski, S. V., A survey of some results in the field of discrete mathematics, *Proc. IFIP Congress 1968, Edinburgh*, North-Holland, Amsterdam, 1969, pp. 266—270.

[36] Лупанов, О. Б., Об асимптотических оценках сложности управляющих систем, Международный конгресс математиков в Ницце 1970, Доклады советских математиков, М., 1972, стр. 162—167.

[37] Лупанов, О. Б. О методах получения оценок сложности и вычисления индивидуальных функции, *Дискретный анализ*, Новосибирск, вып. 25, 1974, стр. 3—18.

[38] Вайнцвайг, М. Н., О мощности схем из функциональных элементов, *ДАН СССР*, т. 139, № 2, 1961, стр. 320—323.

(Поступило 2-ого августа 1979 г.)

Structure of program runs of non-standard time

By L. CSIRMAZ

1. Introduction

In this section the set of natural numbers is denoted by $\mathcal{N}$, and the set of Peano axioms (with $+$ and $\cdot$ only) by PA. In our point of view, a program is a finite sequence of labelled statements. The labels are (distinct) natural numbers. Each statement is either an assignment of the form "$v \leftarrow \tau$" where v is a variable symbol and τ is a term of Peano arithmetic containing operator symbols $+$ and $\cdot$ only, or is an if-statement of the form "IF χ THEN l" where χ is a quantifier free formula of PA and $l \in \mathcal{N}$ is a label. Denote by V_p the (finite) set of variable symbols occuring in the program p and let L_p be the set of the labels of the statements and $h \in \mathcal{N} \setminus L_p$ (the "halt" label). A *run* of the program p is a sequence $\langle l_i, f_i \rangle_{i \in \mathcal{N}}$, where

(i) $l_i \in L_p \cup \{h\}$ and $f_i \colon V_p \to \mathcal{N}$ is a valuation of the variables for every $i \in \mathcal{N}$;

(ii) if $l_i = h$ then $l_{i+1} = l_i$, $f_{i+1} = f_i$;

(iii) if the statement labelled by l_i is "$v \leftarrow \tau$" then

$$l_{i+1} = \begin{cases} l_i + 1 & \text{if } l_i + 1 \in L_p, \\ h & \text{otherwise,} \end{cases}$$

$$f_{i+1}(w) = \begin{cases} f_i(w) & \text{if } w \in V_p,\ w \neq v, \\ \tau[f_i] & \text{if } w = v; \end{cases}$$

(iv) if the statement labelled by l_i is "IF χ THEN l" then

$$l_{i+1} = \begin{cases} l & \text{if } l \in L_p \text{ and } \chi[f_i] \text{ is true,} \\ l_i + 1 & \text{if } l_i + 1 \in L_p \text{ and } \chi[f_i] \text{ is false,} \\ h & \text{otherwise,} \end{cases}$$

$$f_{i+1} = f_i.$$

The run of the program halts, if $l_i = h$ for some i [cf. 4].

It is well-known that every program can be written in this form because we have made no restriction on the content of variables [5]. Moreover, there is a straighforward way to prove partial correctness of programs of this type: assign formulas to every element of $L_p \cup \{h\}$ and prove (say, from the Peano axioms) that if the formula assigned to l_i is satisfied then after executing the statement belonging to l_i, the formula assigned to l_{i+1} will be satisfied, too. Then, if the run halts, the formula assigned to h is satisfied, i.e. the program is partially correct. This method is the so called Floyd—Hoare derivation [6].

It is easy to give a rigorous proof that if a program has a Floyd—Hoare derivation, i.e. if we may assign formulas and prove what we have to prove then the program is partially correct. But, alas, there are programs which always halt, always give the same result but have no Floyd—Hoare derivation. For example let φ be a formula of Peano arithmetic such that neither φ nor its negation are provable from PA. Our program checks whether its only input is the Gödel number of a proof of φ from PA. If it is, it prints 1, if not, i.e. if the input is either not a Gödel number of a proof or does not prove φ, it prints 0. Our program always halts because it is a decidable property to be a proof of a concrete formula, and always prints 0 because there is no proof of φ. Moreover, we can not prove this (from PA) because then we would be able to prove in PA that there is a non-provable formula, i.e. that PA is consistent, which is impossible.

This difficulty vanishes if we allow the program to operate not only on $\mathcal{N}$ but on any model of PA and to run not only through finite time but through non-standard time. This idea is behind the concept of continuous trace, it simulates the non-standard runs of programs, see [1], section 3 of [7], and [8].

2. Notation, definitions

Denote by L the set of classical first order formulas of type t, where t is the similarity type of arithmetic, i.e. it consists of "$+, \cdot, 0, 1$" with arities "2, 2, 0, 0", respectively. PA denotes the following (infinite) set of axioms:

P1 $x+1 \neq 0$

P2 $x+1 = y+1 \leftrightarrow x = y$

P3 $x+0 = x$

P4 $x+(y+1) = (x+y)+1$

P5 $x \cdot 0 = 0$

P6 $x \cdot (y+1) = (x \cdot y)+x$

P7 for all formulas φ with x as free variable

$$[\varphi(0) \wedge \forall x(\varphi(x) \to \varphi(x+1))] \to \forall x \varphi(x).$$

We will use other relation and function symbols, as e.g. $x<y$ or rem (x, y) which are definable in PA. We introduce the bounded quantifiers $(\forall x<y)\varphi(y) \leftrightarrow \leftrightarrow \forall x(x<y \to \varphi(x))$, etc., too. The following reformulation of the axiom of induction

P8 $[(\forall y < x)\varphi(y) \to \varphi(x)] \to \forall x\varphi(x)$

may be obtained from P7 substituting $\varphi(x)$ by $(\forall y<x)\varphi(y)$ [cf. 2].

The inclusion $A\subset B$ allows the sets A and B be equal.

To save space we use vector notation in place of sequence of symbols of same type. E.g. we write $\varphi(\bar{x})$ instead of $\varphi(x_1, x_2, \dots, x_n)$, etc. The dimension of vectors is always clear from the context.

Definition. Let A be any model of PA with universe A. Let $\varphi(x_1, \dots, x_n, y_1, \dots, y_n)$ be a formula of PA so that

$$\mathrm{PA} \vdash \forall x_1 \dots \forall x_n \exists ! y_1 \dots \exists ! y_n \varphi(x_1, \dots, y_n). \tag{2.1}$$

Let $\vec{q}_a=\langle q_a^1, \dots, q_a^n\rangle$ be a sequence of length n of elements of A for every $a\in A$. We say that the sequence $\langle\vec{q}_a\rangle_{a\in A}$ is a *continuous trace* (ct in short) of φ if

$$\mathbf{A} \models \varphi(\vec{q}_a, \vec{q}_{a+1}) \quad \text{for every} \quad a\in A; \qquad (2.2$$

for every formula ψ of PA and every sequence $\vec{p}$ of elements of A

$$\mathbf{A} \models [\psi(\vec{q}_0, \vec{p}) \land \bigwedge_{a\in A} (\psi(\vec{q}_a, \vec{p}) \to \psi(\vec{q}_{a+1}, \vec{p}))] \to \bigwedge_{a\in A} \psi(\vec{q}_a, \vec{p}). \tag{2.3}$$

In the remaining part of this paper we fix the model A of PA, the formula φ, the sequence $\vec{q}_0$ and its length n. Whenever we speak about continuous traces we mean ct of φ with first element $\vec{q}_0$ in A.

We shall need the notion of coding function of sequences. Let rem (x, y) be the remainder when x is divided by y and define the ordered pair $\langle x, y\rangle$ as $(x+y)\cdot$ $\cdot(x+y)+x$. Let moreover the triplet $\langle x, y, z\rangle=\langle x, \langle y, z\rangle\rangle$ and define the formulas PAIR (z), SEQ (u) and the functions LENGTH (u), ELEM (u, i) as follows.

$$\mathrm{PAIR}(z) = \forall u(u\cdot u \leqq z\land(u+1)\cdot(u+1) > z \to z \leqq u\cdot u+u);$$

$$\mathrm{SEQ}(u) = \mathrm{PAIR}(u)\land \forall x \forall y(u = \langle x, y\rangle \to \mathrm{PAIR}(y));$$

$$\mathrm{LENGTH}(u) = \begin{cases} n & \text{if } \mathrm{SEQ}(u) \text{ and } u = \langle x, y, n\rangle, \\ 0 & \text{otherwise,} \end{cases}$$

$$\mathrm{ELEM}(u, i) = \begin{cases} \mathrm{rem}(m, 1+(i+1)\cdot b) & \text{if } \mathrm{SEQ}(u) \text{ and } u = \langle m, b, n\rangle \text{ and } i < n, \\ 0 & \text{otherwise.} \end{cases}$$

A straighforward proof shows that

$$\mathrm{PA} \vdash \mathrm{PAIR}(z) \to \exists ! x \exists ! y(z = \langle x, y\rangle)$$

$$\mathrm{PA} \vdash \forall u \exists ! n(\mathrm{LENGTH}(u) = n)$$

$$\mathrm{PA} \vdash \forall u \forall i \exists ! x(\mathrm{ELEM}(u, i) = x).$$

We say that $u\in A$ is a *sequence* if $\mathbf{A}\models \mathrm{SEQ}(u)$, its length is n if $\mathbf{A}\models \mathrm{LENGTH}(u)=n$ and its i-th element is a if $\mathbf{A}\models \mathrm{ELEM}(u, i)=a$. Note that 0 is a sequence of length 0.

The following theorem says that every sequence can be lengthened by 1 [3].

Theorem 1. $\mathrm{PA} \vdash \forall u\, \forall z\, \exists v(\mathrm{SEQ}(u) \to$
$\{\mathrm{SEQ}(v) \wedge \mathrm{LENGTH}(v) = \mathrm{LENGTH}(u) + 1 \wedge$
$(\forall i < \mathrm{LENGTH}(u))(\mathrm{ELEM}(u, i) = \mathrm{ELEM}(v, i)) \wedge$
$\mathrm{ELEM}(v, \mathrm{LENGTH}(u)) = z\})$. □

3. The result

First we prove some lemmas. We remind that $\mathbf{A}$, φ, $\vec{q}_0$ and n are fixed.

Lemma 1. There are a formula Φ of PA and a unique sequence $\langle \vec{q}_a \rangle_{a \in A}$ of sequences of length n of elements of A with the given $\vec{q}_0$ such that

$$\mathrm{PA} \vdash \forall m\, \forall x_1 \ldots \forall x_n\, \exists! y_1 \ldots \exists! y_n \Phi(m, x_1, \ldots, x_n, y_1, \ldots, y_n) \tag{3.1}$$

$$\mathrm{PA} \vdash \forall m\, \forall \vec{x}\, \forall \vec{y}\, \forall \vec{z} (\Phi(m, \vec{x}, \vec{y}) \wedge \Phi(m+1, \vec{x}, \vec{z}) \to \varphi(\vec{y}, \vec{z})) \tag{3.2}$$

$$\mathbf{A} \models \Phi(a, \vec{q}_0, \vec{q}_a) \quad \text{for every} \quad a \in A \tag{3.3}$$

$$\mathbf{A} \models \varphi(\vec{q}_a, \vec{q}_{a+1}) \quad \text{for every} \quad a \in A. \tag{3.4}$$

Proof. Let $\Phi_1(m)$ be

$$\forall x_1 \ldots \forall x_n\, \exists u \chi(x_1, \ldots, x_n, u, m).$$

where χ is "u is a sequence of length $m+1$ such that every element of u is a sequence of length n, the elements of the 0-th element of u are $x_1, \ldots, x_n$ in this order and for every $i < m$ the i-th element $\vec{y}$ and the $(i+1)$-st element $\vec{z}$ of u satisfy $\varphi(\vec{y}, \vec{z})$".

It is clear that $\mathrm{PA} \vdash \Phi_1(0)$, one only have to use Theorem 1 n times. In view of (2.1) and Theorem 1, $\mathrm{PA} \vdash \Phi_1(m) \to \Phi_1(m+1)$ holds. Therefore, by the induction axiom, $\mathrm{PA} \vdash \forall m \Phi_1(m)$. A very similar argument shows that the following formula, denoted by Φ_2, is also PA provable (by induction on i):

$\forall x_1 \ldots \forall x_n\, \forall u\, \forall v\, \forall i$ ("if u and v are sequences as above and $i \leq \min(\mathrm{LENGTH}(u), \mathrm{LENGTH}(v))$ then the elements of the i-th element of u and v coincide").

Now let Φ be $\exists u(\chi(x_1, \ldots, x_n, u, m) \wedge$ "the elements of the m-th element of u are $y_1, \ldots, y_n$ in this order"). The existence in (3.1) is ensured by Φ_1, the uniqueness is by Φ_2. (3.2) is trivial. Consider now the valuation of Φ in $\mathbf{A}$ where the values of $x_1, \ldots, x_n$ are $q_0^1, \ldots, q_0^n$, respectively while m has the value $a \in A$. Denote the values of $y_1, \ldots, y_n$ for which $\Phi(m, \vec{x}, \vec{y})$ holds in $\mathbf{A}$ by $q_a^1, \ldots, q_a^n$, respectively. The $\vec{q}_a$'s are determined uniquely by (3.1) and (3.3) is satisfied by definition. (3.4) follows immediately from (3.2). □

Definition. The sequence $\langle \vec{q}_a \rangle_{a \in A}$ defined previously is called the *standard continuous trace* (sct in short). (We will see later that this sequence forms a continuous trace indeed.)

Lemma 2. Let ψ be any formula of PA, $\vec{p}$ a fixed sequence of elements of A and

$$\mathbf{A} \not\models \psi(a, \vec{q}_a, \vec{p})$$

for some $a \in A$. Then there is a least suffix with this property, i.e. $a \in A$ such that

$$\mathbf{A} \not\models \psi(a, \vec{q}_a, \vec{p}) \quad \text{but} \quad \mathbf{A} \models \psi(b, \vec{q}_b, \vec{p}) \quad \text{if} \quad b < a.$$

Proof. Suppose the contrary, i.e. whenever $\mathbf{A} \models \psi(b, \vec{q}_b, \vec{p})$ for every $b < a$ then $\mathbf{A} \models \psi(a, \vec{q}_a, \vec{p})$. Denote by $\Psi(m, \vec{x}, \vec{z})$ the formula $\exists! \vec{y}(\Phi(m, \vec{x}, \vec{y}) \wedge \wedge \psi(m, \vec{y}, \vec{z}))$. Then, by the reformulation of the induction axiom,

$$\mathrm{PA} \vdash [(\forall n < m) \Psi(n, \vec{x}, \vec{z}) \rightarrow \Psi(m, \vec{x}, \vec{z})] \rightarrow \forall m \, \Psi(m, \vec{x}, \vec{z}).$$

Now valuate this in $\mathbf{A}$ putting $\vec{q}_0$ instead of $\vec{x}$ an $\vec{p}$ instead of $\vec{z}$. Notice, that the implication in the square bracket holds, therefore the second half of the implication holds also, i.e. $\mathbf{A} \models \psi(a, \vec{q}_a, \vec{p})$ for every $a \in A$, which is a contradiction. □

Corollary. In virtue of this lemma, we may use induction type proofs for the sequence $\langle \vec{q}_a \rangle_{a \in A}$. □

Lemma 3. Let $u, v \in A$, $0 < v$. If $\vec{q}_u = \vec{q}_{u+v}$ then

$$\vec{q}_{u+x} = q_{u+x+v \cdot y}$$

for every $x, y \in A$.

Proof. Let $y = 1$ and use induction by the Corollary on x. After this fix x and use induction on y. □

Lemma 4. There exists an $E \subset A$ such that

$$\text{if } b_1, b_2 \in E, b_1 \neq b_2 \quad \text{then} \quad \vec{q}_{b_1} \neq \vec{q}_{b_2}; \tag{3.5}$$

$$\text{for every } a \in A \text{ there exists } b \in E \text{ such that } \vec{q}_a = \vec{q}_b; \tag{3.6}$$

$$\text{either } E = A \text{ or for some } e \in E, E = \{a \in A \colon a \leqq e\}. \tag{3.7}$$

Proof. If all of the elements of the sequence $\langle \vec{q}_a \rangle_{a \in A}$ are different, the set $E = A$ satisfy (3.5)—(3.7). If not, there is an $a \in A$ so that $\vec{q}_a$ occurs at least twice in the sequence. This property is expressible by an L-formula, so, by Lemma 2, we can assume that this a is the minimal one, i.e. $\vec{q}_b$ is unique if $b < a$. There are other occurrences of $\vec{q}_a$, hence, also by Lemma 2, there is a second one, i.e. there is an $e \geqq a$ such that $\vec{q}_a = \vec{q}_{e+1}$ but $\vec{q}_a \neq \vec{q}_b$ if $a < b \leqq e$. We claim that the set $E = \{a \in A \colon a \leqq e\}$ satisfies (3.5) and (3.6). It is sufficient to see (3.5) in case $a < b_1 < < b_2 \leqq e$ only. Suppose $\vec{q}_{b_1} = \vec{q}_{b_2}$. Lemma 3 with the cast $u = b_1, v = b_2 - b_1$, $x = \mathrm{rem}\,(e + 1 - b_1, b_2 - b_1)$ gives $\vec{q}_{e+1} = \vec{q}_{b_1 + x} \neq \vec{q}_a$ which is a contradiction. (3.6) is an easy consequence of Lemma 3. □

Now we have all of the tools for the proof of the main result of this paper. First we need some more preliminaries.

Definition. The subset $S \subset A$ is a *slice* if $a \in S$ implies $a + 1 \in S$ and $b \in S$ for all $b < a$.

The subset $T \subset A$ is a *thread* if $a \in T$ implies, $a + 1 \in T$, $a - 1 \in T$ (for $a \neq 0$) and $a, b \in T$, $a < b$ imply that for some natural number n

$$b = a + 1 + \ldots + 1 \quad (n \text{ times}).$$

The function $f: A \to A$ is a *projector* if $f(0)=0$ and $f(a+1)=f(a)+1$ for every $a \in A$.

The sequence $\langle \vec{p}_a \rangle_{a\in A}$ is a *projection* of the sequence $\langle \vec{q}_a \rangle_{a\in A}$ if there exists a projector f and a slice S such that

$$\vec{p}_a = \vec{q}_{f(a)} \text{ for every } a \in A \tag{3.8}$$

$$\mathrm{Rng}\,(f) \subset S \tag{3.9}$$

$$\text{for every } b \in S \text{ there is an } a \in A \text{ such that } \vec{q}_b = \vec{q}_{f(a)}. \tag{3.10}$$

Theorem 2. The standard continuous trace $\langle \vec{q}_a \rangle_{a\in A}$ forms a continuous trace.

Proof. (2.2) of the definition is satisfied by (3.4) of Lemma 1. (2.3) is immediate from the Corollary of Lemma 2. □

Theorem 3. The projections of the standard continuous trace are continuous traces. Moreover every continuous trace is a projection of the standard one.

REMARKS. An easy consequence of Theorem 3 is that if A is a non-standard model of PA then for all φ and $\vec{q}_0$, the cardinality of ct's of φ with first element $\vec{q}_0$ is $2^{|A|}$.

Now consider a ct $\langle \vec{p}_a \rangle_{a\in A}$ the defining formula of which is

$$\varphi(x_1, \ldots, x_n, y_1, \ldots, y_n) = \text{“}y_1 = x_1 + 1 \wedge \ldots\text{”}$$

and let $p_0^1 = 0$ for the initial position $\vec{p}_0$. Then $p_{a+1}^1 = p_a^1 + 1$ for every $a \in A$ but one can not hope for $p_a^1 = a$ in general. Actually, by Theorem 3, the standard ct is the only ct which has this property. We may interpret this phenomenon as follows. We add a “clock” to a continuous trace and suppose that the clock works well (i.e. it jumps by 1 at every step). If we require the clock to show the correct time then the ct is unique. Compare with Theorem 3.3 of [7].

Proof of the theorem. First let $f: A \to A$ be the projector and S the slice of a projection. We prove that $\langle \vec{p}_a \rangle_{a\in A}$ is a ct. (2.2) of the definition follows from $f(a+1)=f(a)+1$. To prove (2.3) let $\psi \in \mathrm{L}$ be arbitrary and assume

$$\mathbf{A} \models \psi(\vec{p}_0, \vec{p}) \wedge \bigwedge_{a\in A} (\psi(p_a, \vec{p}) \to \psi(\vec{p}_{a+1}, \vec{p})).$$

By the hypotheses, $\vec{p}_0 = \vec{q}_0$, for all $b \in S$ there is an $a \in A$ such that $\vec{q}_b = \vec{p}_a$ and if $\vec{q}_b = \vec{p}_a$ then $\vec{q}_{b+1} = \vec{p}_{a+1}$. So we know that

$$\mathbf{A} \models \psi(\vec{q}_0, \vec{p}) \quad \text{and} \quad \mathbf{A} \models \psi(\vec{q}_a, \vec{p}) \to \psi(\vec{q}_{a+1}, \vec{p}) \quad \text{for all} \quad a \in S \tag{3.11}$$

and it is enough to show that this implies

$$\mathbf{A} \models \psi(\vec{q}_a, \vec{p}) \quad \text{for all} \quad a \in S.$$

Suppose the contrary. Then, by Lemma 2, there is a least counter-example, say b. But this b belongs to S because S is a slice, which contradicts (3.11).

Now we turn to the second and longer part of the proof. Let $\langle \vec{p}_a \rangle_{a\in A}$ be any ct, $\vec{p}_0 = \vec{q}_0$. We claim that for any $a \in A$ there exists a least $b \in A$ such that $\vec{p}_a = \vec{q}_b$. Indeed, let $\psi(\vec{y}, \vec{x})$ be “$\exists m \Phi(m, \vec{x}, \vec{y})$”. It is clear that $\mathbf{A} \models \psi(\vec{p}_0, \vec{q}_0)$ with $m=0$ and if $A\models\psi(\vec{p}_a,\vec{q}_0)$ with some m, then $A\models\psi(\vec{p}_{a+1},\vec{q}_0)$ with $(m+1)$ because the successors are unique. Then by (2.3), $A\models\psi(\vec{p}_a,\vec{q}_0)$ for all $a\in A$ which states the existence of b. Finally, Lemma 2 ensures a least one. Denote this b by $f(a)$ and denote S the range of the function f. Let moreover E and e be as defined in Lemma 4.

It is clear that $S\subset E$, $f(0)=0$ and if $f(a)\neq e$ then $f(a+1)=f(a)+1$. Now let $b\in A$, $b\notin S$ and let $\psi(\vec{x},\vec{y},m)$ be the formula "$(\exists m'<m)\Phi(m',x,y)$". Since $0<b$ and if $f(a)<b$ then $f(a+1)\leqq f(a)+1<b$, we know

$$A\models\psi(\vec{p}_0,\vec{q}_0,b)\quad\text{and}\quad A\models\psi(\vec{p}_a,\vec{q}_0,b)\rightarrow\psi(\vec{p}_{a+1},\vec{q}_0,b).$$

By (2.3) of the definition of ct, $A\models\psi(\vec{p}_a,\vec{q}_0,b)$, i.e. $f(a)<b$ for all $a\in A$. This means that if $b\in S$ and $c<b$ then $c\in S$.

We distinguish two cases.

1. $E=A$ or $E\neq A$ but $e\notin S$. In this case f is a projector and S is a slice, therefore $\langle\vec{p}_a\rangle_{a\in A}$ is a projection.

2. $e\in S$, i.e. $S=E$. Let $b\in E$ so that $\vec{q}_b=\vec{q}_{e+1}$. By Lemma 3 $\vec{q}_u=\vec{q}_{e+1}=\vec{q}_b$ if $u=a+y\cdot(e+1-a)$, so if we choose y large enough then the thread T_u of u does not contain b. For each thread T define the function $g_T\colon T\rightarrow A$ as follows. If T is the thread of 0 then let $g_T(a)=a$. Otherwise if $f(v)=b$ for some $v\in T$ then let $g_T(a)=b+a-v$. Otherwise if $f(v)=e$ for some $v\in T$ then let $g_T(a)=u-1+a-v$, otherwise let $g_T(a)=f(a)$. Finally, let $g(a)=g_T(a)$ if a is in the thread T.

It is clear from the definition of g_T that g is a projector and $\vec{p}_a=\vec{q}_{f(a)}=\vec{q}_{g(a)}$. By Lemma 4, every $\vec{q}_b$ equals to some $\vec{p}_a$, i.e. in this case the projector g and the whole A as a slice shows that $\langle\vec{p}_a\rangle_{a\in A}$ is a projection. □

Abstract

Continuous traces are introduced to simulate program runs when time is measured by the elements of a non-standard model of Peano axioms. This concept is a very useful one in considerations of program verification. We give here a full description of continuous traces in every model of PA. It turns out that there is exactly one continuous trace definable by a formula of PA and every other one can be got from this by a simple transformation.

MATHEMATICAL INSTITUTE OF THE
HUNGARIAN ACADEMY OF SCIENCES
REÁLTANODA U. 13—15.
BUDAPEST, HUNGARY
H—1053

References

[1] ANDRÉKA H., I. NÉMETI, Completeness of Floyd logic, *Bull. of Section of Logic* v. 7, 1978, pp. 115—121.

[2] CHANG, C. C., H. J. KEISLER, *Model theory*, North Holland, 1973.

[3] CSIRMAZ, L., On definability in Peano arithmetic, *Bull. of Section of Logic*, v. 8, 1979, pp. 148—153.

[4] MANNA, Z., *Mathematical theory of computation*, McGraw-Hill, 1974.

[5] PÉTER, R., *Rekursive Funktionen in der Komputer-Theorie*, Akadémiai Kiadó, 1976.

[6] PRATT, V. R., Semantic considerations on Floyd-Hoare logic, 17-th *IEEE Symp. Found. Comp. Sci.*, 1976.

[7] SAIN, I., There are general rules for specifying semantics, *Observations on Abstract Model Theory*, CL & CL, 1979, No. 2, to appear.

[8] SÁNTÁNÉ—TÓTH, E., M. SZŐTS, Colloquium on Logic in Programming, Salgótarján, *Rundbrief der Fachgruppe der Gesellschaft für Informatik*, 1979, No. 17.

(Received April 5, 1979)

Nondeterministic programming within the frame of first order classical logic, Part 1

By T. Gergely and L. Úry

1. Introduction

1.1 Nondeterminism in computer science

In computer practice a lot of phenomena have arisen that deviate from the deterministic attitude forming the base of traditional programming. These nondeterministic phenomena may be due to varying reasons.

Considering the reasons three main types of nondeterminism can be distinguished. The first type of nondeterminism is quite independent from the will of programmers and its causes are hidden in the construction and functioning of computers. Due to this nondeterminism almost every program has some uncertainty while execution. These could be caused by power cut, current trouble, machine break-down or by any other unforseeable reason. If the computer works in time-sharing mode the uncertainty further increases and the behaviour of a program will depend on the other programs executed alternately with it. Moreover it depends on the memory requirements of the programs, on the number of peripheries at disposal, etc. In computers allowing parallel computations further causes of uncertainty interfer, namely the speed difference of certain processes and communication, the noise level of the communication channel, the concurrency for resources etc. In interactive mode another type of uncertainty is caused by the randomness of interactions affecting the program under execution.

We may call *probability programming* the methods that consider the above uncertainties and its theory should be based on the usage of the tools of mathematical statistics and those of theory of probability. Random events occuring in program execution are handled by these tools. In this type of programming the commands do not have a uniquely defined result, only its distribution is known. Thus the running of a program can be described by using stochastic process e.g. by using either Markov or semi-Markov chains. One of the main aims of the theory of such type of programming is to minimalize the expectable number of failures.

The second type of nondeterminism is already connected with the programmer's will. The programmer's attitude is still deterministic, but he uses probabilistic methods containing well defined randomness in the solution of some tasks. A widespread method of this type is connected with the use of random number.generator. The programming style using this method is deterministic and it also supposes the determinism of the computer, however, for the solution of certain tasks it uses one of the Monte-Carlo methods.

The third type of nondeterminism is connected with the *essential* and *logical* uncertainty enclosed in the solutions of tasks. It embodies two kinds of uncertainties. The first one occurs in such a situation of problem solving when there are more alternat ives selecting from which any, the result will be produced without difficulty. The second kind of uncertainty is connected with such a situation where only certain alternatives lead to correct results but, in advance, we do not know which one. A programming style which considers the above mentioned two kinds of uncertainties suggests a nondeterministic attitude in contrast with the deterministic one of the traditional style of programming. The main difference between these two kinds of attitudes is that the nondeterministic one considering different kinds of choices does not specify how to make them though the deterministic attitude does not leave the question how to make choices (if there are any) unspecified. Programming theory connected with the third type of nondeterminism is in the focus of our further investigations.

1.2 Some reasons of interest in nondeterministic programming

Recently nondeterministic programming has attracted more and more attention. It provides the programmers to concentrate on some important questions about deterministic programs without specifying details irrelevant to the questions to be analysed. Thus this programming attitude can be used to reason about deterministic programs.

The possibility to consider actions not describing their details makes the nondeterministic programming very useful in the field of Artificial Intelligence, e.g. in natural language understanding, in problem solving, in robot planning, etc. E.g. it suggests a fractional method of problem solving or robot-planning as follows. First a global algorithm — a "global plan" can be designed as a nondeterministic program, then, by analyzing this program and completing it with appropriate parts we get a concrete deterministic program, i.e. a complete detailed algorithm to solve the task.

One of the most significant reasons why nondeterministic programming becomes more and more important is that it plays a significant role in the elaboration of the theory of interactive and parallel programming. Most.of its applications are connected with this area. See e.g. Hoare (1978), Milner (1973), Owicki and Gries (1975), Plotkin (1976), etc.

In view of aboves it is quite natural that nondeterministic programming plays an increasing role in both the theory and practice of programming.

The aim of our investigation in the present work is the elaboration of a mathematical theory of nondeterministic programming which can handle both syntax and semantics by using mathematical tools and provides tools to speak about program properties and to prove them. The elaboration of this theory will be done by using the approach developed in Gergely and Úry (1978).

1.3 Some words on our approach

The essence of the programming situation to be considered is that beside programming language such a new language arises that is suitable to describe the properties and the meaning of programs and our expectations towards programs in an unambiguous way. Thus this new language is a descriptive one in contrast with the programming language which serves to give instruction, i.e. commands. To assure unambiguity the descriptive language should have well defined and exact semantics beyond the syntax. The syntax should be suitable to describe program properties and with the power of proof to analyse whether certain features of programs correspond to the expectations given by the specification, i.e. to analyse the correctness of programs. The semantics of the descriptive language should provide unambiguous understanding of the meaning of programs, hence it has to be compatible with the semantics of programming language. There are two main possibilities to give exact semantics. The first is to characterize programs according to the question "*what* the program does?" the second is to do it according to the question "*how* the programs do it?"

In programming theories we find the following three approaches for the exact handling of semantics: *operational, functional* and *resultative*. The first aims to give a direct answer to both questions to *what* and *how*. The resultative or, in other words, axiomatic semantics neglects the question *how* and characterizes only the main properties of the change of data environment of the program produced while execution. Functional or, in other words, denotational semantics also gives the meaning of programs answering both questions, though it does it in an indirect way.

We wish to elaborate such a theory of nondeterministic programming which would be also a useful base for developing the mathematical theory of interactive and parallel programming. In order to understand the main features of interaction and parallelism in detail the execution processes themselves are to be considered. This permits to follow up the specific features connected with the mutual effects of the processes (e.g. communication, interaction). Thus operational semantics seems to be adequate to our aim. To have this type of semantics first the question *what* has to be answered by the characterization of changes in data caused by the execution of program and, secondly, the flow of programming processes in time has to be described to have an answer to the question *how*.

Thus the theory of programming to be developed has to have such a descriptive language that is capable of describing and characterizing both the data environment of programs and the time related to program execution. The first requirement is quite familiar with nearly every theory of programming, but not so is the time consideration, for programming theories do not consider time explicitly except for some of the most recent works.

Of course in the case of sequential programming the time aspects can be characterized through the change of data without considering time explicitly. However this approach is not applicable in those cases where time plays a primary and independent role as e.g. in interactive, real time and parallel programming. Therefore the programming theory to be developed here will contain tools that also provide explicit time consideration.

1.4 The role of classical first order logic in a theory of programming

To develop a mathematical theory of programming that corresponds to our aim the first problem is to introduce such a descriptive language that satisfies the above mentioned expectation concerning the characterization of time and data and it has to have exact and well defined semantics and appropriate tools to prove different statements about the program properties. In the case of sequential deterministic programming the first order classical language was quite satisfactory to be a descriptive language for the corresponding theory of programming as it was shown in GERGELY and ÚRY (1978) where the frame of classical first order mathematical logic was used to develop the theory of programming.

In the present work we show that the above mentioned logical frame is sufficient to develop the corresponding theory and the first order language can be used in the role of the descriptive one for the case of sequential nondeterministic programming. Why do we prefer the classical first order language? Because

— it has a well defined exact and transparent syntax and semantics;

— it has a special branch, the model theory with very strong mathematical methods to investigate semantics;

— it has a well developed proof theory that offers effective notion of proof and effective tools and methods for proving;

— it is currently used in the research practice so its use is fairly familiar;

— it is the simplest one of the languages of mathematical logic by which a programming language can be investigated since the propositional language is not suitable for this.

So it is justified to try to elaborate the mathematical basis of programming theory within the frame of classical mathematical logic that is the most highly developed branch of mathematical logic. This is encouraged by the fact that data environments of programs can be given without major restriction of generality by first order language.

In this work the mathematical foundations of programming theory, and the elaboration of the theory itself is done by strictly keeping to the frame of first order logic.

In the theory both date and time will be explicitly discussed by using first order language.

1.5 A short survey

Nondeterministic programming is mainly used in the area of Artificial Intelligence and in the investigation of parallel computation as it has already been mentioned. In connection with the first area MANNA (1970) introduces a nondeterministic programming language which is very similar to the language to be introduced here. It contains both kinds of choices, but it does not allow the description of time conditions.

Several works are devoted to the nondeterminism in connection with parallel computation. In the early work of ASHCROFT and MANNA (1970) parallelism has already been explained in terms of nondeterminism. Milner handles nondeterminism by using oracles. In the case of two computing processes executed parallelly an oracle is such an infinite sequence of 0 and 1 that be however far contains both elements 0 and 1. By this oracle the execution of two parallelly computed processes can be described so that 0 and 1 denote which process is at work. The theory using oracles is described in MILNER (1973) and (1978).

A very elegant mathematical theory of the same handling of nondeterminism is developed in PLOTKIN (1975). Developing the theory of programming both MILNER and PLOTKIN use denotational description of semantics of nondeterministic programs. EGLI (1975) also uses this type of the description of semantics.

An axiomatic definition of semantics is given in OWICKI and GRIES (1976).

The parallel programming containing nondeterminism suggests to introduce effective nondeterministic elements into the language. The very simple command *choice* S_1, S_2 has been replaced by the guarded commands introduced in DIJKSTRA (1975). In HOARE (1978) and FRANCEZ et al. (1978) input-output commands are added to the guarded ones. Analogical commands are also introduced in MILNER (1978). Not depending on the aboves we mention the work of HAREL and PRATT (1978) where the execution of programs is supposed to be ambiguous and in the descriptive language modalities are introduced in order to handle the ambiguity. Thus by this descriptive language such statements can be expressed that "there exists such a run...", "every run is such...". In this work the semantics is operational. An analoguous descriptive language is developed in MIRKOWSKA (1978) within the frame of algorithmic logic. Summarizing aboves we would like to emphasize that so far no work has been engaged in using tools to describe time conditions explicitly. The problem of completeness is discussed only in HAREL and PRATT (1978) and MIRKOWSKA (1978). The previous shows that the introduced descriptive language is complete with respect to arithmetics, in the latter it is proved that the nondeterministic algorithmic logic is ω-complete. We note that these two results are really equivalent.

1.6 What is new and the contents of the work

The theory of nondeterministic programming is developed strictly within the frame of first order classical logic. The semantics of nondeterministic programs is described in an operational way by using a special type of games. A descriptive language to describe program properties is introduced by using the classical first order language. This descriptive language allows to describe and to speak explicitly about both time and data.

Moreover the question of completeness is discussed and a complete calculus in the spirit of Floyd and Hoare is introduced.

The first part including the first three sections contains the conceptual and mathematical base providing exact tools to handle nondeterminism. Thus the next section is devoted to the main tool of our theory to a special type of games. Section 3 contains the main notions of classical first order mathematical logic and arithmetic and the representation of data and time in the frame of first order logic. Here first of all the description of time properties is discussed in details. In Section 4 the basic notions and properties of games are introduced within the frame of first order logic. A very simple but powerful enough nondeterministic programming language is introduced in Section 5. Its semantics is given by using associated games. With respect to nondeterministic programs many different questions can arise. Some of them are given in Section 6. Here we immediately show that an adequate descriptive language is needed to answer the questions for each one. Selecting two questions in connection with partial and total correctness we give the appropriate descriptive language and show that it is complete. In Section 7 we introduce a calculus which is analoguous to that of introduced by Floyd and Hcare for the sequential deterministic programming (see details in GERGELY and ÚRY (1978)). In Section 8 we illustrate the use of the calculus by some examples. Present paper consists of two parts. The first one contains the first three sections.

1.7 Basic conventions

We use basic notations and concepts of the naive set theory in the usual fashion. The notation $\{x|\varphi(x)\}$ denotes the set of all x such that $\varphi(x)$. Both inclusion and proper inclusion are denoted by the same symbol $\subset$. The empty set is denoted by $\emptyset$. In the case of natural numbers for ordered finite set we use intervals defined by $[i, j]\overset{d}{=}\{k|i\leqq k\leqq j\}$. The *domain* and *range* of a function f are denoted by $\text{do}\,f$ and $\text{rg}\,f$ respectively. $f\colon A\to B$ denotes that f is a function for which $\text{do}\,f=A$ and $\text{rg}\,f=B$. A function $f\colon A\to B$ is *injective* if for any $a, b\in A$ if $f(a)=f(b)$ then $a=b$. It is called *bijective* if it is injective and $f(A)=B$.

The symbol $\langle s_i\rangle_{i\in I}$ denotes a function f with domain I such that $f(i)=s_i$ for all $i\in I$. Such a function f is called *sequence*.

For a non-empty set A let A^+ denote the set of all finite non-empty sequences formed from the elements of A. ${}^AB$ denotes the set of all functions from A to B. ω is the least infinite ordinal. $|A|$ denotes the cardinality of the set A. Moreover for informal logic we use "iff" for "if and only if" and w.r.t. for "with respect to".

The end of significant units like proofs, definitions etc. is marked by the symbol □.

2. Games

2.1 More about nondeterminism

As we have seen the uncertainty in nondeterministic programming is caused by two kinds of choices. The first one: from the alternatives one chooses such a possible step of a task solution that leads to the result. The other one is when each one of the alternative steps may be chosen and the result thus can be reached.

Having a nondeterministic program its execution can be so imagined that there is someone who represents the interests of the program, say Mr. A. He is the one who makes the first kind of choices. In opposition there is someone else, say Mr. B, representing the circumstances influencing the program execution. He is the one who makes the other kind of choices without being influenced by the interests of the program.

Thus we have a situation analoguous to a game situation where two players A and B are playing. Player A has to choose so that whatever B chooses the course of the game should favour A. This analogy suggests the games to be a useful and easily handable tool for our investigation. The games are very close to our intuition because they are widespread. At the same time they clearly represent essential nondeterminisms e.g. that of due to the uncertainty of players in the moves of one another. This uncertainty is quite analoguous to that of the nondeterministic programming. Thus we use an appropriate type of games as the main tool in our theory of nondeterministic programming.

Now let us consider what type of game is adequate to represent the nondeterminism of nondeterministic programming.

According to the aboves we say that the rules define the circumstances within the frame of which a game can be played, i.e. they define the *game-frame.* A game-frame still does not possess goals for the players though in the type of games used here the goals are well-defined for both players. E.g. such a goal can be either to win, or not to lose. Having goals players aspire to win or not to lose within a given game-frame. Games considered here are antagonistic in the sense that players try to achieve in fact two opposite goals. Beside the goals such rules are to be introduced that specify the *conditions* by which one of the players wins, or loses or the play is draw. These conditions can be defined by the appropriate set of those situations (or states) that provide the winning (or not losing) of one of the players. In this case to achieve his goal the player is to reach the winning situations, i.e. he has to make moves providing the appropriate states. The improvement of the positions of one of the players at the same time is spoiling the positions of the other one. Another possibility is to give a rule specifying a payment function which renders a payment to each possible state. This payment is to be received by one of the players and is payed by the other one and its sum depends on the situation. Moreover the gain of one of the players and the loss of the other one is equal but opposite in sign. Now the winning of each player means to receive the greatest payment. Thus the goal of a player is to maximize the payment he receives.

If we add rules describing the winning conditions to the game-frame we get the corresponding *game.* It is obvious that with respect to a given game-frame a lot of games can be defined, which only differ in the winning conditions.

2.2 Basic notions of games

Let us consider such a type of game that presumes two players and possesses well-defined rules for each of them. A game presupposes a sequence of moves, each of which is an occassion for a choice between certain alternatives.

The rules of the game specify for each move which player does it and what his alternatives are. These rules are finitely describable, and are to be known by each player. At each move, the player precisely knows what his alternatives are and his choice will become immediately known to the other player. Moreover each player precisely knows what moves, i.e. what choices have been made previously. Thus the players have full information about what has happened in the game so far and what else can ever happen during the course of it. For the latter the rules are to specify that no choice can be made by chance (e.g. by a dice). This means that each move is deterministic in such a sense that the situation formed after having the moves is foreseen in a unique way. A *course* of game contains a complete sequence of choices (moves) made by the players.

Thus the type of games used here consists of the rules that define the circumstances of playing and of the goals and rules defining the winning conditions.

To illustrate the abovesaids let us consider the following version of the well-known game NIM. First of all let us see its frame. There is a single pile of chips containing e.g. 21 chips and there are two players A and B. The two players take turns one after the other picking up chips from the pile. At each move, a player must take at least one chip and at most three ones. This is the game-frame. If we fix the winning circumstances then we get the game of the game-frame. Let us suppose that the player who picks up the last chip loses and thus we have got a game. Note that this kind of NIM game is often called Last One Loses. Of course we can define an opposite game with respect to the same frame, namely we add the following rule: the player picking up the last chip wins. By adding another winning condition we get a new game. There are a lot of other possibilities.

A game-frame graphically can be represented by a tree. The nodes of the tree correspond to the situations involved in the game. The arcs emanating from a given node are the alternatives associated with the corresponding move. A tree representing all the possible moves of both players and all the possible corresponding situations is called a *game-tree* or an *and/or-tree.* A *path* of the game tree represents a play of the corresponding game-frame. The winning condition can be represented by a set of paths per players leading to winning. Thus a tree represents a game-frame. Marking out the winning paths of both players we get the tree representation of a game of the given game-frame.

Note that the representability of a game by a tree means that the game is of full information, i.e. the players have full information about the course of the game because each node of the tree includes the history of its acces since each node, except the root, has exactly one predecessive node.

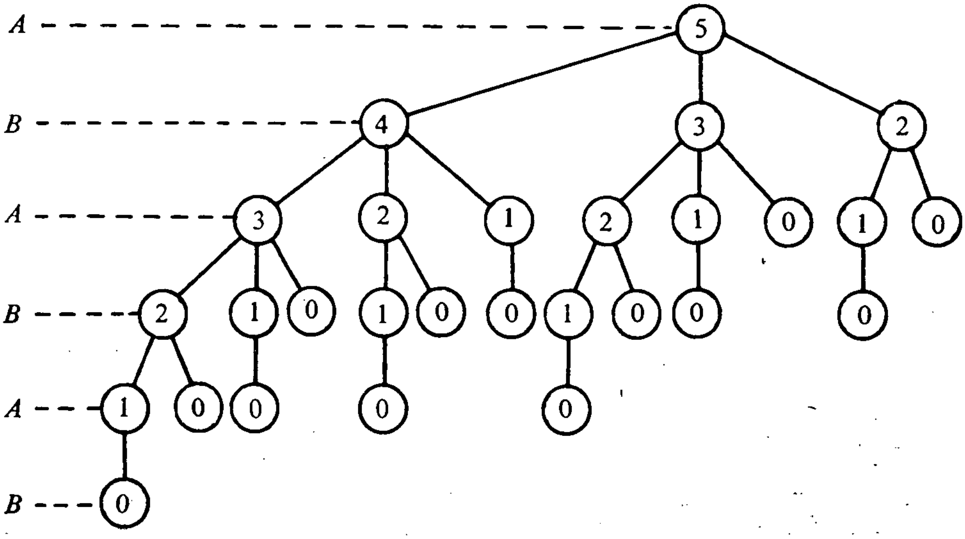


Fig. 1

To illustrate the abovesaids let us see the game-tree (Fig. 1) of the game Last One Loses for the case when the players begin with five chips in the pile. The nodes are labelled by the number of the remained chips in the pile. At alternate levels of depth in the tree, alternate players choose which move to make. To be definite we suppose that player A moves first. Each depth level is labelled by the name of the player who has the next choice at that level.

Since the so far mentioned type of game is a basic means of the theory of programming to be developed and since we wish to execute the investigation within a mathematical frame it is necessary to provide the mathematical definition of the basic notions of the games. To introduce games as mathematical objects we use their tree representation.

Let N be the set of natural numbers and let N^* denote the set of all finite sequences consisting of the elements of N. Λ denotes the empty sequence.

Let us take the following functions:

$$pair\colon\ N^* \times N \to N^*$$

$$left\colon\ N^* \setminus \{\Lambda\} \to N^*$$

$$right\colon\ N^* \setminus \{\Lambda\} \to N$$

$$length(v) = \begin{cases} 0 & \text{if } v = \Lambda \\ length(left(v)) + 1 & \text{otherwise} \end{cases}$$

for any $v \in N^*$.

Intuitively speaking by the use of the function *pair* we can construct a new sequence if we add a natural number to a given sequence from the right side. The functions *left* and *right* provide the decomposition of sequences.

Definition 2.1. A set $V \subset N^*$ is said to be a *tree* iff the following properties hold:

(i) $\Lambda \in V$,

(ii) if $v \in V$ and $v \neq \Lambda$ then $left(v) \in V$. □

Example 2.2. Let us consider the following tree in graphical representation shown in Fig. 2.

According to our definition this can be represented as the following tree: {0, 01, 014, 0148, 015, 02, 026, 0269, 03, 037}, where 0 stands for Λ.

The graphical representation of this tree is shown in Fig. 3. □

Let $v, w \in V$. If $left\ (w) = v$ then w is a *successor* of v and v is a *predecessor* of w in the tree V.

The set of all successors of a node v in V is

$$S_V(v) \overset{d}{=} \{w \in V \mid left(w) = v\}.$$

Let $W \subset V$ be such that it satisfies the conditions (i) and (ii) of 2.1. Then W is a *subtree* of V.

Definition 2.3. A subtree $W \subset V$ is said to be a *path* of V iff the function *left:* $W \setminus \{\Lambda\} \rightarrow W$ is an injection.

Definition 2.4. Let V be a tree and $C \subset V$. The pair (V, C) is said to be a *game-frame.* □

The above defined game-frame provides frame for games of two players with full information. Let Mr. A and Mr. B be the players. The set C indicates those nodes of the tree V in which player A makes moves. Thus a (V, C) game-frame provides the following course. The starting point $v_0 = \Lambda$, for an arbitrary node v_n if $v_n \in C$ then it is A's turn and he can choose one of the alternatives and the course is driven into the corresponding node of $S_V(v_n)$. If $v_n \notin C$ then it is B's turn and the move is analogous to the above situation.

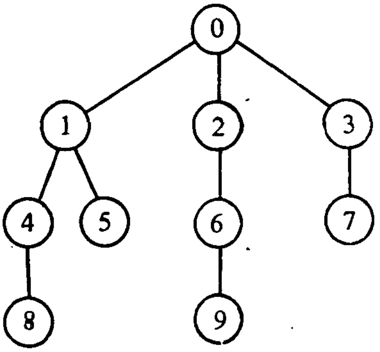


Fig. 2

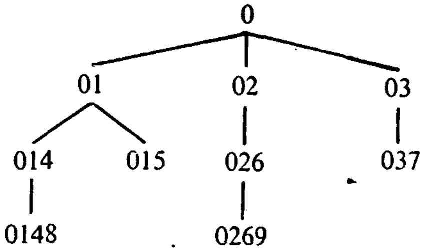


Fig. 3

A course in the game-frame (V, C) is a path. Along a course there are the following possibilities:

(i) the course is finite, i.e. neither A nor B can move further because the corresponding set of successive nodes is empty;

(ii) the course is infinite, i.e. A and B can move further in every node.

Thus a game course of the game-frame (V, C) is a finite or infinite path in the tree V.

To have a game in the frame of a given (V, C) a winning condition is needed. According to the aboves the winning condition can be given as a set of those paths in V along which the player, say A, wins. Thus let Γ_A and Γ_B be sets such that $\Gamma_A \cap \Gamma_B = \emptyset$. We say that $\Gamma_A(\Gamma_B)$ is the set of winning paths in V of the player $A(B)$ if it contains those paths along which player $A(B)$ wins. If the course of the game provides such a path that belongs neither to Γ_A nor to Γ_B we have a play which is draw. We note that there is a lot of different possibilities to give the sets Γ_A and Γ_B. For example it may be the case when player A aspires not to win as well as not to lose. This means that A wishes to prevent the winning of player B. In such cases it is quite enough to give the set Γ_B. The set Γ_A is unnecessary because any path that does not belong to Γ_B is satisfactory to player A.

Definition 2.5. Let (V, C) be a game-frame. Moreover, let Γ_A and Γ_B be the set of all winning paths of the players A and B respectively. The quadruple $\mathfrak{A} = (V, C, \Gamma_A, \Gamma_B)$ is said to be a *game* of the game-frame (V, C). □

A player can move in such a way that he decides in advance which alternative he chooses in each possible situation. This means that the player uses a special set of rules that tells him what choices he should make for all situations that might arise during the course of a game. This set of rules is called a strategy which is definable by mathematical tools.

Definition 2.6. Let (V, C) be a game-frame. A function *str* defined on C is a strategy of player A for the game-frame (V, C) iff $str\ (v) \in S_V(v)$ for every $v \in C$. □

The other player's strategy can be similarly defined, but we do not need it.

The function *str* gives the successor for each $v \in C$ and it seems that this depends only on v. However, remember that each node v includes its prehistory.

The strategy of player A defines what move he has to make when he achieves a situation v where his turn is the next. From the above definition follows that a strategy provides the moves in each possible statement many of which do not appear during a game because the player never reaches them if he plays according to the given strategy.

So it is quite natural to define the strategy in a less redundant way, namely considering only the subtree that can be potentially arisen by using the strategy.

Definition 2.7. Let *str* be a strategy of player A for the game-frame (V, C). A subtree $R_{str} \subset V$ is generated by the strategy *str* iff it has the following properties:

(i) if $v \in C \cap R$ then $S_{R_{str}}(v) = \{str(v)\}$ i.e. v has exactly one successor in R_{str} that is picked up by *str*,

(ii) if $v \in R \setminus C$ then $S_{R_{str}}(v) = S_V(v)$.

This subtree R_{str} is unique. □

Thus if player A makes moves in accordance with his strategy *str*, then during a course of the game-frame (V, C) any of the paths of R_{str} can be realized. However since the moves of A are determined by *str*, player B can choose any of his alternatives. Thus B can realize any of the paths of R_{str}. According to the aboves it is quite natural to define a strategy of the player A for a game-frame (V, C) by means of an appropriate subtree of V.

Definition 2.8. Let (V, C) be a game-frame. A subtree $R \subset V$ is said to be a *run* of the game-frame iff the following properties hold:

(i) if $v \in C \cap R$ then there is a unique successor w of v in R. (I.e. there is a unique $w \in R$ such that *left* $(w) = v$.)

(ii) if $v \in R \setminus C$ then $S_R(v) = S_V(v)$. □

It is obvious that for any run R there exists a strategy *str* of player A such that

$$R_{str} = R.$$

Note that for a given run R the appropriate function *str* is not unique because while defining it we consider only its subdomaine $R \cap C$ and its values on $C \setminus R$ can be arbitrary. So far the strategy has been introduced for a game-frame (V, C). Considering winning conditions, i.e. a game $(V, C, \Gamma_A, \Gamma_B)$, we can speak about winning strategy or not losing strategy. A strategy of player A is winning (not losing) iff moving accordingly the course of game realizes only paths belonging to Γ_A (not belonging to Γ_B). I.e. if $\pi \subset R$ then $\pi \in \Gamma_A$ ($\pi \notin \Gamma_B$).

For the illustration of the so far introduced notions let us see the following

Example 2.9. Let us consider the game Last One Loses with game-tree given in Fig. 1. In this case there is one pile of five chips and players have the alternatives to pick up from one to three chips at a move. The frame of this game is the pair (V, C), where

$$V = \{5, 54, 543, 5432, 54321, 543210, 54320, 5431, 54310, 5430, 542, 5421, 54210,$$
$$5420, 541, 5410, 53, 532, 5321, 53210, 5320, 531, 5310, 530, 52, 521, 5210, 520\},$$
$$C = \{5, 543, 542, 541, 532; 531, 530, 521, 520, 54321, 54320, 54310, 54210, 53210\}.$$

The winning condition of the game Last one Loses is as follows:

$$\begin{aligned}\Gamma_A = \{&(5, 54, 543, 5432, 54321, 543210),\\ &(5, 54, 543, 5430),\\ &(5, 54, 542; 5420),\\ &(5, 54, 541, 5410),\\ &(5, 53, 532, 5320),\\ &(5, 53, 531, 5310),\\ &(5, 52, 521, 5210)\}.\end{aligned}$$

Γ_B consists of all paths not belonging to Γ_A.

Let us consider the following run R:

$$R = \{5, 54, 543, 5430, 542, 5420, 541, 5410\}.$$

A corresponding winning strategy *str* of player *A* is the following:

v	5	543	542	541	54321	532	531	521
str (v)	54	5430	5420	5410	543210	5320	5310	5210

3. Logic and arithmetic

3.1 Logic

We intend to develop the theory of nondeterministic programming within the frame of classical first order mathematical logic. To be able to do so we recall the basic notions and definitions that we need to reach our aim.

Definition 3.1. A similarity type ϑ is a pair of functions $(\vartheta_R, \vartheta_F)$ such that $\text{rg}\ \vartheta_F \subset \omega$, $\text{rg}\ \vartheta_R \subset \omega \setminus \{0\}$, $\text{do}\ \vartheta_F \cap \text{do}\ \vartheta_R = \emptyset$ and $|\text{do}\ \vartheta_R| \cdot |\text{do}\ \vartheta_F| \leqq \omega$. The elements of do ϑ_R and do ϑ_F are called relation and function symbols respectively. ϑ_R and

ϑ_F give the arity of symbols. The 0-ary function symbols are called constant ones. □

For the following we fix a similarity type ϑ for which $=\in \mathrm{do}\,\vartheta_R$ and $\vartheta_R(=)=2$.

Definition 3.2. A ϑ-type model $\mathfrak{A}$ is a function on $\mathrm{do}\,\vartheta_R\cup \mathrm{do}\,\vartheta_F\cup\{\emptyset\}$ such that

(i) $\mathfrak{A}(\emptyset)=A$ is a nonempty set, which is called the universe of the model,

(ii) $\mathfrak{A}(\varrho)\subset {}^{\vartheta_R(\varrho)}A$ for any $\varrho\in \mathrm{do}\,\vartheta_R$,

(iii) $\mathfrak{A}(=)$ is the diagonal relation on ${}^2A$,

(iv) $\mathfrak{A}(f)\colon {}^{\vartheta_F(f)}A\to A$ for any $f\in \mathrm{do}\,\vartheta_F$.

In a special case ${}^0A=\{\emptyset\}$ i.e. if $\vartheta_F(f)=0$ then $\mathfrak{A}(f)$ can be identified with an element of A. □

In general instead of $\mathfrak{A}(s)$ we write $s_{\mathfrak{A}}$ where $s\in \mathrm{do}\,\vartheta_R\cup\,\mathrm{do}\,\vartheta_F$. A ϑ-type model will always be denoted by a German capital and its universe by the corresponding Roman capital. M_ϑ denotes the class of all ϑ-type models.

Now we turn to the definition of the syntax.

Definition 3.3. Let V be any denumerable set. Let T_ϑ^V be the minimal set satisfying the following properties:

(i) $V\subset T_\vartheta^V$,

(ii) for any n and $f\in\vartheta_F^{-1}(n)$ if $\tau_1, \dots, \tau_n\in T_\vartheta^V$ then $f(\tau_1, \dots, \tau_n)\in T_\vartheta^V$.

The elements of T_ϑ^V are called terms.

Take $A_\vartheta^V=\{\varrho(\tau_1, \dots, \tau_n)\mid \varrho\in\vartheta^{-1}(n), n\in\omega, \tau_1, \dots, \tau_n\in T_\vartheta^V\}$. The elements of A_ϑ^V are called atomic formulas.

The set F_ϑ^V of ϑ-type formulas with variable symbols belonging to V is the minimal set satisfying the following properties:

(i) $A_\vartheta^V\subset F_\vartheta^V$,

(ii) if $\varphi, \psi\in F_\vartheta^V$ then $\varphi\wedge\psi\in F_\vartheta^V$,

(iii) if $\varphi\in F_\vartheta^V$ then $\neg\varphi\in F_\vartheta^V$,

(iv) if $\varphi\in F_\vartheta^V$ and $v\in V$ then $\exists v\varphi\in F_\vartheta^V$.

Let Q_ϑ^V be the minimal set satisfying the above conditions (i)—(iii). The elements of Q_ϑ^V are called quantifier free formulas. □

We use the following abbreviations

a) $\varphi\vee\psi$ for $\neg(\neg\varphi\wedge\neg\psi)$,

b) $\varphi\to\psi$ for $\neg(\neg\psi\wedge\varphi)$,

c) $\varphi\leftrightarrow\psi$ for $\neg(\neg\psi\wedge\varphi)\wedge\neg(\neg\varphi\wedge\psi)$,

d) $\forall v\varphi$ for $\neg\exists v\neg\varphi$,

where $v\in V$ and $\varphi, \psi\in F_\vartheta^V$.

For any $s\in T_\vartheta^V\cup F_\vartheta^V$ let $\mathrm{Var}\, s$ denote the set of free variable symbols occuring in s.

For any $v\in V$, $\tau\in T_\vartheta^V$ and $\varphi\in F_\vartheta^V$ let $\varphi[\tau/v]$ be the formula obtained from φ by replacing every free occurrence of v in φ by τ so that there would not be a collision between the variable symbols of τ and the variable symbols of φ occuring with quantifiers.

Now we define the semantics of the first order language by defining a relation $\models_\vartheta\subset M_\vartheta\times F_\vartheta^V$.

Definition 3.4. Let $\mathfrak{A}\in M_\vartheta$. A valuation of V in $\mathfrak{A}$ is a function $q\colon V\to A$, i.e. a valuation is an element of ${}^VA$. Now we extend the valuation q to a function $\bar{q}\colon T_\vartheta^V\to A$ taking:

(i) $\bar{q}(v)=q(v)$ for every $v\in V$;

(ii) $\bar{q}(f(\tau_1,\dots,\tau_n))=f_{\mathfrak{A}}(\bar{q}(\tau_1),\dots,\bar{q}(\tau_n))$ for every $n\in\omega$, $f\in\vartheta_F^{-1}(n)$ and $\tau_1,\dots,\tau_n\in T_\vartheta^V$. □

Instead of $\bar{q}(\tau)$ we write $\tau[q]$. It is clear that $\tau[q]$ depends only on the values of Var τ. So sometimes we use the following notations:

(i) a variable symbol is often written underlined by a waved line to denote its value by a given valuation. E.g. if q is a given valuation then we write $\underset{\sim}{x}$ instead of $q(x)$;

(ii) let $\underset{\sim}{\bar{a}}\in A$ denote an arbitrary finite sequence of elements from A. For any $\tau\in T_\vartheta^V$ supposing that $\underset{\sim}{\bar{a}}$ contains at least as many elements as Var τ we write $\tau[\underset{\sim}{\bar{a}}]$ instead of $\tau[q]$.

The validity relation is defined by the following well known

Definition 3.5. Let $\mathfrak{A}\in M_\vartheta$ be arbitrary. Moreover, let $\mathfrak{A}\models_\vartheta\subset F_\vartheta^V\times{}^VA$ be the following relation:

(i) $\mathfrak{A}\models_\vartheta\varrho(\tau_1,\dots,\tau_n)[q]$ iff $(\tau_1[q],\dots,\tau_n[q])\in\varrho_{\mathfrak{A}}$ for any atomic formula;

(ii) $\mathfrak{A}\models_\vartheta(\varphi\vee\psi)[q]$ iff $\mathfrak{A}\models_\vartheta\varphi[q]$ and $\mathfrak{A}\models_\vartheta\psi[q]$;

(iii) $\mathfrak{A}\models_\vartheta(\neg\varphi)[q]$ iff $\mathfrak{A}\not\models\varphi[q]$;

(iv) $\mathfrak{A}\models_\vartheta\exists v\varphi[q]$ iff there is a valuation $q^*\colon V\to A$ such that $q^*\restriction_{V\setminus\{v\}}=q\restriction_{V\setminus\{v\}}$ and $\mathfrak{A}\models_\vartheta\varphi[q^*]$.

$\mathfrak{A}\models_\vartheta\varphi[q]$ means that the formula φ is valid in the model $\mathfrak{A}$ by the valuation q.

In the end $\mathfrak{A}\models_\vartheta\varphi$ iff for every valuation $q\in{}^VA$, $\mathfrak{A}\models_\vartheta\varphi[q]$. □

So the ϑ-type first order language $L_\vartheta=(F_\vartheta^V, M_\vartheta, \models_\vartheta)$ has been defined. If it does not cause ambiguity we write $\models$ instead of $\models_\vartheta$.

Now let $Ax\subset F_\vartheta^V$ be an arbitrary consistent set of formulas. Restricting M_ϑ to $Md(Ax)\overset{d}{=}\{\mathfrak{A}\in M_\vartheta|\mathfrak{A}\models Ax\}$ from the language L_ϑ we can define a new first order language $L_\vartheta^{Ax}=(F_\vartheta^V, Md(Ax), \models)$, which consists of the class of the models of Ax only. Further on in this study while an Ax is considered it is always supposed to be consistent without claiming this explicitly.

The notion of definiability plays a main role among the tools of our investigation. We recall that this notion is used in mathematical logic in two different senses. In the first one it is considered when and how new symbols with given properties can be added to a fixed language. This is the topic of the Definition Theory. For us, however, the other sense which is interested in knowing whether a function or a relation given in an arbitrary model can be expressed in a fixed language is more useful. We introduced the main definitions corresponding to this second approach.

Let us fix a language L_ϑ and let $\mathfrak{A}\in M_\vartheta$ be arbitrary.

Definition 3.6. A partial function $g\colon{}^nA\to A$ is said to be parametrically definable in $\mathfrak{A}$ iff there is a formula $\varphi\in F_\vartheta^V$ such that

(i) Var $\varphi=\{x_1,\dots,x_n,y,a_1,\dots,a_m\}$;

(ii) There are $\underset{\sim}{a}_1, \ldots, \underset{\sim}{a}_m \in A$ such that for any

$$\underset{\sim}{\vec{x}} \in A \quad \text{and} \quad \underset{\sim}{y} \in A, \ \mathfrak{A} \models \varphi[\underset{\sim}{\vec{x}}, \underset{\sim}{y}, \underset{\sim}{\vec{a}}] \quad \text{iff} \quad g(\underset{\sim}{\vec{x}}) = \underset{\sim}{y}.$$

Similarly, a relation $\varrho \subset {}^nA$ is said to be parametrically definable in $\mathfrak{A}$ iff there is a formula $\varphi \in F_\vartheta^V$ such that

(i) Var $\varphi = \{x_1, \ldots, x_n, a_1, \ldots, a_m\}$;

(ii) There are $\underset{\sim}{a}_1, \ldots, \underset{\sim}{a}_m \in A$ such that for any $\underset{\sim}{\vec{x}} \in A$, $\mathfrak{A} \models \varphi[\underset{\sim}{x}, \underset{\sim}{\vec{a}}]$ iff $\underset{\sim}{\vec{x}} \in \varrho$.

A partial function g or a relation ϱ is definable iff the appropriate φ does not contain a_i's. □

We say that the above φ parametrically defines the partial function g or the relation ϱ in $\mathfrak{A}$.

Now we also fix an $Ax \subset F_\vartheta^V$. Let us suppose that for any $\mathfrak{A} \in Md(Ax)$ a function $g_{\mathfrak{A}}\colon {}^nA \to A$ (a relation $\varrho_{\mathfrak{A}} \subset {}^nA$) is given. Take $G = \{g_{\mathfrak{A}} | \mathfrak{A} \in Md(Ax)\}$ $(R = \{\varrho_{\mathfrak{A}} | \mathfrak{A} \in Md(Ax)\}$.

Definition 3.7. G (or R) is *parametrically definable* in Ax iff there is a formula φ which parametrically defines $g_{\mathfrak{A}}$ (or $\varrho_{\mathfrak{A}}$) in $\mathfrak{A}$ for every $\mathfrak{A} \in Md(Ax)$.

If the set $\{g_{\mathfrak{A}} | \mathfrak{A} \in Md(Ax)\}(\{\varrho_{\mathfrak{A}} | \mathfrak{A} \in Md(Ax)\})$ is parametrically definable and the definition is given by the formula φ then the function symbol g (the relation symbol ϱ) is said to be *universally definable in* $Md(Ax)$.

Example 3.8. Let $\varphi \in F_\vartheta^V$ be such that Var $\varphi = \{x_1, \ldots, x_k, y\}$, and suppose that

$$Ax \models \forall x_1 \ldots x_k \forall y \forall z (\varphi \wedge \varphi[z/y] \to y = z) \tag{1}$$

If so then in every model $\mathfrak{A}$ of Ax φ defines a partial function in the following way:

(i) $\underset{\sim}{\vec{x}} \in \text{do} f$ iff $Ax \models \exists y \varphi[\underset{\sim}{\vec{x}}]$,

(ii) $f(\underset{\sim}{\vec{x}}) = \underset{\sim}{y}$ iff $Ax \models \varphi[\underset{\sim}{\vec{x}}, \underset{\sim}{y}]$.

By (1) this definition is good and thus we use the following abbreviation:

$$\text{Parc}\, \varphi \overset{\text{d}}{=} \forall \vec{x} \forall y \forall z (\varphi \wedge \varphi[z/y] \to y = z). \quad \square$$

We say that the above φ parametrically defines G or R. If φ contains no a_i's we omit the adjective "parametrically".

Remark 3.9. If the above G is definiable in Ax and every $g_{\mathfrak{A}}$ is total then a new function symbol g "can be added" to ϑ_F of arity n with the following new axiom

$$Ax_g\colon \forall \vec{x} \forall y (y = g(\vec{x}) \leftrightarrow \varphi(\vec{x}, y))$$

where φ defines G. So we get a new language $L_\vartheta^{Ax'}$, where $Ax' = Ax \cup \{Ax_g\}$. The phrase "can be added" means that for any $\varphi \in F_\vartheta^V$, $Ax' \models \varphi$ iff $Ax \models \varphi$.

The details see in Section 2.9 of Mendelson (1964).

A similar fact holds for the above R.

3.2. Arithmetic

As known arithmetic plays an important role in computer science. It provides an unambiguous characterization of any formal language syntax. This permits the widespread use of computers since their functioning is based on natural number representation. While the numeric use of computers arithmetic plays an important role since the data form a structure satisfying the basic features of arithmetic. Arithmetic is also important to formalize our intuitive concept about discrete time connected with computer functioning. Thus in our investigation of programming theory arithmetic plays an important role. Namely, it provides formal tools to characterize sequences which prove to be useful in the study of program properties.

Let η be the type of arithmetic, i.e. do $\eta_R=\{=\}$, do $\eta_F=\{0, 1, +, \cdot\}$ and $\eta_F(0)=\eta_F(1)=0, \eta_F(+)=\eta_F(\cdot)=2$.

For the axiomatization of the arithmetic we choose the well-known Peano axioms:

$$A_1 \overset{\mathrm{d}}{=} \neg(v+1=0)$$

$$A_2 \overset{\mathrm{d}}{=} v+1=w+1 \rightarrow v=w$$

$$A_3 \overset{\mathrm{d}}{=} v+0=v$$

$$A_4 \overset{\mathrm{d}}{=} v+(w+1)=(v+w)+1$$

$$A_5 \overset{\mathrm{d}}{=} v\cdot 0=0$$

$$A_6 \overset{\mathrm{d}}{=} v(w+1)=(v\cdot w)+v$$

$$A_{7\varphi} \overset{\mathrm{d}}{=} \varphi[0/v] \wedge \forall v(\varphi \rightarrow \varphi[v+1/v]) \rightarrow \forall v\varphi$$

Take $I=\{A_{7\varphi} | \varphi \in F_\vartheta^V$ and $v \in \mathrm{Var}\,\varphi\}$. The set of Peano-axioms is

$$PA \overset{\mathrm{d}}{=} \{A_i | 0 \leqq i \leqq 6\} \cup I.$$

For detailed analysis of PA see e.g. Mendelson (1964).

As usually we use the following abbrevations

$$x \leqq y \text{ instead of } \exists z(z+x=y),$$

$$x<y \text{ instead of } x \leqq y \wedge \neg x=y.$$

We recall that for every infinite cardinal there are at least continuum number of non-isomorphic models of that cardinality of PA. For every $\mathfrak{A} \in Md(PA)$ its smallest submodel A_c satisfies PA and these submodels are isomorphic to each other and they are called standard models of PA. We would like to consider only standard models but unfortunately that is impossible at a first order language because there is no first order formula describing exactly the standard part of the models of PA. Thus if we are interested whether a first order formula is valid then we must consider not only standard models but nonstandard ones as well.

As usually $\mathfrak{N}$ denotes the fixed standard model of PA and $\mathbf{N}$ stands for its universe.

For handling of non-standard models see e.g. Robinson (1966).

3.3 The role of time in the theory of programming

As mentioned already in Introduction often not only the output result of a computing process is significant, but its temporal course too. Thus we would like to develop such a theory of nondeterministic programming that handles both data and time explicitly by the help of first order tools.

The representation of data within the frame of first order logic is straightforward; it can be done by the universe of the classical models. However relations and functions of the models correspond to data properties and to their possible changes respectively. Thus from the point of view of data computers are represented by the models of first order languages. Thus the previously mentioned representation neglects the explicit time representation. How to represent time is a question that should be looked at in details. But the functioning of computers is controlled by an "inner clock" so the change in data happens in time.

We assume that a change in data corresponds to a command which is executed for a timecycle of the machine. Let us denote the set of these disjoint time intervals by T. From theoretical point of view the time intervals of T can be considered as time moments supposing that the change takes place infinitely fast. We also assume that a machine works as long as it is needed i.e. as long as it is required by the program. This means that a machine itself can work infinitely long never stopping due to a break-down. However it stops only if it is required by the program and by this the program execution terminates. Let us consider the simplest case when there are only assignment statements.

The execution of a program on a machine is but the execution of assignment statements step by step i.e. iteratively. The transition of states of the machine representing the change in data is defined by the transition function. This function can be defined by induction on T as follows. In case we already know the state S_t of the machine at moment t then the state at the next moment $t+1$ can be defined by the state S_t using the concrete command that is to be executed in the moment t. To describe this by mathematical tools the closeness of the transition function under iteration (recursion) must be ensured.

Thus to represent time an arbitrary structure can be used which provides the starting moment, the generation of the next moment and the induction by succession. For example if we take a $\{(0, 0), ({}', 1)\}$-type structure $\mathfrak{T}=(T, 0, {}')$ on which the induction works well then this can be used to represent time. Here T represents the set of time moments. Note that further on it will be also supposed that on the set of time moments T the usual addition and multiplication are also considered and the time moments are in order.

Thus to represent discrete time the use of the structure $\mathfrak{N}=(\mathbf{N}, 0, 1, +, \cdot)$ of natural numbers is obvious. However our main standpoint is to use classical first order language to describe models. Thus we cannot restrict ourselves to the standard model $\mathfrak{N}$ but any model $\mathfrak{T}$ of an appropriate first order axiomatization of $\mathfrak{N}$ is allowed. Consequently beside $\mathbf{N}$, which is very close to our intuition, very strange sets of time moments are allowed as well. Especially such sets T in which "infinitely large" (or non-standard) time moments also occur.

As usual the theories of programming developed so far use either implicitely or explicitely the set of natural numbers $\mathbf{N}$ to represent time.

So our assumption that the structure representing time has to satisfy only one condition, namely the axiom scheme of induction, seems not to be very close to our intuition. Thus let us go into a bit more details.

Our first notion is purely theoretical. If the iteration is the essence of programming then to represent time any such model can be chosen that provides to follow the changes done by iteration. So on this structure the induction must be allowed. Hence developing a theory of programming we have no reason to introduce further restriction for time (e.g. to suppose that time moments belong to N).

If we have a practical look at it then the situation seems to be totally different. Namely, in practice there exists no procedure containing non-standard number of steps. So the "infinitely large" time moment seems to be a fiction. However, if we consider the history of mathematics this opinion can be dissipated. That is, infinitesimal values play an important role in the history of mathematical analysis but their reason for the existence was only recently observed by A. Robinson and his followers.

Non-standard analysis is applied in computer science as well. It provides some very effective methods to solve differential and integral equations.

In order to develop a theory of programming being able to analyse the real situations of programming practice there is no reason to restrict it to the considered notions of "standard and real" machines and time. It is not our aim, of course, to investigate machines with non-standard time. Nevertheless if we have a theory of programming which can handle non-standard time as well then the execution of a program being correct within the frame of this theory will be correct in any machine with any type of time, especially in the machines with standard time.

Indeed well written and well used programs, in our opinion, can be executed in machines with arbitrary type of time, though programmers having developed the programs know absolutely nothing about this. This is so because programmers write down programs thinking in first order language though always imagining the standard time (i.e. the set N) to it. These impressions, fortunately are not embedded in the programs!

It may seem that the first order language is not sufficient to think about programs for it might provide far too many restrictions. However we have proved in Gergely and Úry (1978) that within the frame of classical first order logic for the sequential and deterministic programming a theory of programming of unified attitude can be developed and this frame fully satisfies the solution of the tasks of a programming theory. Present work shows that this frame is completely satisfactory for developing a theory of non-deterministic programming as well.

Now we give the mathematical description of the programming situation. Let ϑ be an arbitrary similarity type containing the type η of arithmetic $(\eta\subset\vartheta)$. Intuitively ϑ provides the *name* of those relations and functions which have to be understood by the computer. The properties of relations and functions are described by the set of formulas $Ax\subset F_\vartheta^V$. The set of axioms Ax expresses the expectations with respect to data and "hardware". Intuitively we always suppose that $PA\subset Ax$ i.e. the computer "understands" the arithmetic in the form of Peano axiomatization.

According to the above saids it is clear that at least two sets are needed to characterize a computer: — the set A of possible data and the set T of possible time moments. We intend to speak of both time and data in a first order language. Since time and data are of different entities it is advisable to distinguish their languages while describing a computer. This could be done by the use of a two sorted first order language, where the first sort corresponds to time and the second one to data. If different data types were allowed then we need a many sorted first order language. The mixed sorted functions and relations describe the connection between time and data.

For the sake of simplicity let us stick to the frame of the classical (one sorted) first order language and to describe time we use the same language as for data representation with the only difference that a new unary relation symbol ζ is introduced. By this we supply our models with an inner time supposing that time can be modelled by data. Of course not each data type can be satisfactory for this aim, e.g. the Boolean data type is not.

Already in this approach we seem to meet the advantages provided by the explicit handling of time. Therefore, it was not necessary to introduce and use the many-sorted first order language.

To describe time we introduce a new unary relation symbol $\zeta \notin$ do ϑ and let us add ζ to the type ϑ. So, we have $\vartheta^* = \vartheta \cup \{(\zeta, 1)\}$. Expectations with respect to time beyond data would be given by a set of axioms $Ax^*(Ax \subset Ax^* \subset F^V_{\vartheta *})$. Of course the set Ax^* is larger than the set of axioms Ax expressing the expectations with respect to data. To formalize the minimal properties expected from time we introduce the following notations $\zeta^*(x) \overset{d}{=} \exists t\ (x \leqq t \wedge \zeta(t))$ (where the relation symbol $\leqq$ is the ordering used in PA),

$$B_0 \overset{d}{=} \zeta^*(0),$$

$$B_1 \overset{d}{=} \zeta^*(x) \to \zeta^*(x+1).$$

The fulfilment of the formulas B_0 and B_1 provides that the set of time moments is not empty. The induction under ζ^* can be formalized as follows:

$$B_{2\varphi} \overset{d}{=} [\varphi(0) \wedge \forall x(\zeta^*(x) \wedge \varphi(x) \to \varphi(x+1))] \to \forall x(\zeta^*(x) \to \varphi(x)).$$

B_1 and $B'_{2\varphi}$s provide the closing under addition and multiplication.

According to the abovesaids with respect to time we always suppose that ζ^* satisfies PA^*, where

$$PA^* = \{B_0, B_1\} \cup \{B_{2\varphi} | \varphi \in F^V_{\vartheta *}, x \in \operatorname{Var} \varphi\}.$$

Definition 3.10. A set of formulas $Ax^* \subset F^V_{\vartheta *}$ is said to be a ϑ-*type system* if $PA^* \subset Ax^*$. $\mathfrak{A}$ is *a model with inner time* $T_{\mathfrak{A}}$ of the system Ax^* if $\mathfrak{A} \models Ax^*$ and $T_{\mathfrak{A}} = \{a \in A | \mathfrak{A} \models \zeta^*[a]\}$. □

Examples 3.11. (i) Let $Ax = PA$ and $Ax^* = PA \cup \{\forall x\, \zeta(x)\} \cup PA^*$. In this case any model $\mathfrak{A} \in Md(PA)$ will be the model of Ax^* if ζ is interpreted by the universe A itself. The model $\mathfrak{A}'$ provided by such a way will be evidently a model of Ax^* with inner time A.

(ii) Let $\tau_1 = \tau_2$ be a Diophantine equation which has no solution in $\mathbf{N}$, but $PA \not\models \neg \tau_1 = \tau_2$. Let $Ax^* = PA \cup PA^* \cup \{\forall \vec{x} \zeta^*(\vec{x}) \to \neg \tau_1(\vec{x}) = \tau_2(\vec{x})\}$. Using the method of (i), from a model $\mathfrak{A} \in Md(PA)$ a model $\mathfrak{A}'$ can be arisen, which would be a model inner time of Ax^* if the equation $\tau_1 = \tau_2$ has no solution in A.

(iii) Let $Ax^* = PA \cup PA^*$, and let $\mathfrak{A} \in Md(PA)$ be arbitrary.

It is obvious that if ζ would be interpreted by the standard part of the model $\mathfrak{A}$ (i.e. by **N**) then the model $\mathfrak{A}'$ arisen by using the method of (i) would be a model of Ax^* with inner time **N**. □

Remarks 3.12. (i) In Definition 3.10 it would be satisfactory to claim that $Ax^* \vdash PA^*$. Thus in Definition 3.11 (i) $Ax^* = PA \cup \{\forall x \zeta(x)\}$ would be enough.

(ii) Intuitively speaking a ϑ-type system Ax^* provides the description of the hardware of a computer. It fixes those features that characterize the static (Ax) and the dynamics ($Ax^* \setminus Ax$) of the computer. Ax^* may have a lot of models which are usually different but it does not completely define a machine. However only those features of machines are interesting in our investigation that are true in every model of Ax^*. □

3.4 Recursive definiability

Let Ax^* be a ϑ-type system. A fairly often used method of implicit definition is the recursive one. In order to understand this situation in the case when Ax^* refers to time we need the following. Let ϱ be a new k-ary relation symbol not occuring in ϑ^* ($\varrho \notin \text{do}\, \vartheta^*$). Let $\varphi(\varrho)$ denote the inclusion $\varphi \in F^V_{\vartheta^* \cup \{(\varrho,k)\}}$ i.e. $\varphi(\varrho)$ is a formula of the syntax the type of which is the extention of ϑ^* with the k-ary relation symbol ϱ. Moreover we need a tool by which we can reduce a formula of $F^V_{\vartheta^* \cup \{(\varrho,k)\}}$ to a formula of $F^V_{\vartheta^*}$. For this the following type of substitution can be used.

Definition 3.13. Let $\varphi \in F^V_{\vartheta^* \cup \{(\varrho,k)\}}$ and let $\chi \in F^V_{\vartheta^*}$ with $\operatorname{Var} \chi = \{x_1, \dots, x_k\}$. Let $\varphi[\chi/\varrho]$ be defined by the following way:

(i) if ϱ does not occur in φ then $\varphi[\chi/\varrho] = \varphi$,

(ii) if $\varphi = \varrho\,(\tau_1, \dots, \tau_k)$ then $\varphi[\chi/\varrho] = \chi[\langle \tau_i/x_i \rangle_{i \in [1,k]}]$,

(iii) if $\varphi = \varphi_1 \langle\rangle \varphi_2$ where $\langle\rangle$ is either $\wedge$ or $\vee$ then $\varphi[\chi/\varrho] = \varphi_1[\chi/\varrho] \langle\rangle \varphi_2[\chi/\varrho]$,

(iv) if $\varphi = \neg\psi$ then $\varphi[\chi/\varrho] = \neg\psi[\chi/\varrho]$,

(v) if $\varphi = Qv\psi$ then $\varphi[\chi/\varrho] = Qv\psi[\chi/\varrho]$ where Q is either $\forall$ or $\exists$. □

We are interested whether the equation $\varrho \leftrightarrow \varphi(\varrho)$ has a solution in Ax^* i.e. whether a formula $\chi \in F^V_{\vartheta^*}$ exists such that

$$Ax^* \models \chi \leftrightarrow \varphi[\chi/\varrho].$$

In this case we say that χ is a solution of the recursive equation $\varrho \leftrightarrow \varphi(\varrho)$ in Ax^*. Moreover for some types of formulas in $F^V_{\vartheta^* \cup \{(\varrho,k)\}}$ there exists a minimal solution of the above recursive equation.

Definition 3.14. A formula $\varphi \in F^V_{\vartheta^* \cup \{(\varrho,k)\}}$ with $\operatorname{Var} \varphi = \{x_1, \dots, x_k\}$ is a pedigree formula iff there is a formula $\psi \in F^V_{\vartheta^* \cup \{(\varrho,k)\}}$ such that

(i) $Ax^* \models_{\vartheta^*} \varphi \leftrightarrow \psi$;

(ii) ψ has the form $\psi = \psi_0 \vee \psi_1$, where ϱ does not occur in ψ_0 and the symbols $\neg$ and $\forall$ do not act on ϱ in ψ_1;

(iii) all occurences of ϱ in ψ contain only variable symbols;

(iv) bounded variable symbols of ψ are distinct from each other. □

The forthcoming theorem shows the recursive definition to be allowed, provided that we make only "positive" statements. This latter is contained in condition (ii) of the above definition of pedigree formula. It is needed in recursive definition to consider only already existing objects and not to speak about such that have not occured so far but may do so sometimes in the future. So for example we cannot say which objects should not belong to a recursively defined set. So the condition (ii) provides the constructive feature of the recursive definiability. The conditions (iii) and (iv) are merely technical. If a formula $\varphi \in F^V_{\vartheta* \cup \{(\varrho,k)\}}$ satisfies conditions (i) and (ii) then it is already a pedigree formula, of course with another ψ as if φ satisfied conditions (iii) and (iv) as well.

Theorem 3.15. For any pedigree formula $\varphi \in F^V_{\vartheta* \cup \{(\varrho,k)\}}$ there exists a formula $\chi_\varphi \in F^V_{\vartheta*}$ such that

(i) Var $\chi_\varphi = \text{Var}\ \varphi$;

(ii) $Ax^* \models \chi_\varphi \leftrightarrow \varphi[\chi_\varphi/\varrho]$ i.e. χ_φ is a solution of the recursive equation $\varrho \leftrightarrow \varphi(\varrho)$;

(iii) if χ is any other solution of the recursive equation, i.e. it is a formula of $F^V_{\vartheta*}$ such that $Ax^* \models \chi \leftrightarrow \varphi[\chi/\varrho]$ then $Ax^* \models \chi_\varphi \to \chi$, i.e. χ_φ is the minimal solution.

Sketch of the proof. It is similar to that of Theorem 3.4 in Gergely and Úry (1978). It uses the fact, that there is a formula $\psi = \psi_0 \vee \psi_1$ such that the properties (i)—(iv) of Definition 3.14 hold. By using the property (i) and the following fact: if $Ax^* \models \varphi_1 \leftrightarrow \varphi_2$ for any $\varphi_1, \varphi_2 \in F^V_{\vartheta* \cup \{(\varrho,k)\}}$ then for any $\chi \in F^V_{\vartheta*}$ with exactly k variable symbols:

$$Ax^* \models_{\vartheta*} \varphi_1[\chi/\varrho] \leftrightarrow \varphi_2[\chi/\varrho]$$

to prove the theorem it is enough to construct such a formula χ_φ that $Ax^* \models \chi_\varphi \leftrightarrow \leftrightarrow \psi[\chi_\varphi/\varrho]$. The idea of the construction is that χ_φ is either ψ_0 or it builds up from ψ_0 applying ψ_1 ζ-many times. The building up of χ_φ can be done similarly to that of Gergely and Úry (1978). □

Abstract (to Part 1)

Nondeterministic programming play an increasing role in the theory of programming. This role is discussed in Section 1 together with the role of classical first order logic in developing a theory of programming. Two kinds of nondeterminism are considered: *any* and *every*. The uncertainty in programs that use both *any* and *every* is quite analogous to that of game situations. So in our theory games are the centre of interest. The basic constructions of games are introduced in Section 2. The theory will be built within the frame of classical first order logic. The basic notions and constructions needed to develop this theory are given in Section 3.

RESEARCH INSTITUTE FOR
APPLIED COMPUTER SCIENCE
CSALOGÁNY U. 30—32.
BUDAPEST, HUNGARY
H—1536

References

[1] Ashcroft, E. and Z. Manna, Formalization of properties of parallel programs, *Artificial Intelligence Memo* AIM—110, Stanford University, Stanford, 1970.

[2] Dijkstra, E. W., Guarded commands; nondeterminacy and formal derivation of programs, *Comm. ACM*, v. 18, 1975, № 8, pp. 453—457.

[3] Egli, H., A mathematical model for nondeterministic computations, Technological University, Zürich, 1975.

[4] Francez, N., C. A. R. Hoare and W. P. de Roever, Semantics of nondeterminism, concurrency and communications, *Mathematical Foundations of Computer Science*, Ed. J. Winkowski, Springer Verlag, Berlin, 1978.

[5] Gergely, T. and L. Úry, Mathematical theories of programming (manuscript), Budapest, 1978.

[6] Harel, D. and V. R. Pratt, Nondetermism in logics of programs, Report MIT/LCS/TM—98, 1978.

[7] Hoare, C. A. R., Communicating sequential processes, *Comm. ACM*, v. 21, 1978, № 8, pp. 666—677.

[8] Manna, Z., The correctness of nondeterministic programs, *Artificial Intelligence*, v. 1. 1970, № 1—2, pp. 1—26.

[9] Mendelson, E., *Introduction to mathematical logic*, Van Nostrand, N. Y., 1964.

[10] Milner, R., An approach to the semantics of parallel programs, *Proceedings of the Convegno di Informatica Teorica*, Instituto di Elaborazione delle Informazione, Pisa, 1973.

[11] Milner, R., Synthesis of communicating behaviour, *Mathematical Foundations of Computer Science*, Ed. J. Winkowski, Springer Verlag, Berlin, 1978.

[12] Mirkowska, G., Algorithmic logic with nondeterministic programs, *Proceedings of Colloquium Mathematical Logic in Programming*, North Holland, 1980 (under publication).

[13] Owicki, S. and D. Gries, An axiomatic proof technique for parallel programs I., *Acta Inform.*, v. 6, 1976, pp. 319—340.

[14] Plotkin, G. D., A powerdomain construction, *SIAM J. Comput.*, v. 5, 1976, № 3, pp. 452—487.

[15] Robinson, A., *Nonstandard analysis*, North Holland, Amsterdam, 1966.

(Received Sept. 7, 1979)

Nondeterministic programming within the frame of first order classical logic, Part 2

By T. GERGELY and L. ÚRY

In this part we develop a mathematical theory of nondeterministic sequential programming by using the mathematical tools introduced in the first part (see GERGELY and ÚRY (1980)). Our investigation is concentrated around the problem of completeness. While this we introduce an appropriate complete descriptive language and a complete calculus in the spirit of Floyd and Hoare. The usage of the calculus is illustrated by three examples.

4. Definable games

We aim at developing a theory of non-deterministic programming within the frame of first order language and therefore we have to formalize the games providing consideration of nondeterminism. Unfortunately the basic notions belonging to games introduced in Section 2 are not that of first order language. To keep ourselves within the frame of first order logic we have to consider such version of these notions that are parametrically definable. Thus we permit only parametrically definable games, strategies etc. However it might happen that the property "to be a definable game" is not definable while each game is definable. By using the arithmetization of formal languages we show below that this is not the case.

Let Ax^* be a ϑ-type system and let $\mathfrak{A} \in Md(Ax^*)$ be arbitrary. There are several definable bijections between $A \times A$ and A. Let us fix one of them and denote it by *pair*. Let *left* and *right* be the two components of the inverse of the function *pair*, i.e. for any $x, y \in A$

$$\mathit{left}\,(\mathit{pair}\,(x, y)) = x \text{ and } \mathit{right}\ (\mathit{pair}\,(x, y)) = y.$$

Remember that in any model $\mathfrak{A} \in Md(Ax^*)$ induction can be done by inner time $T_{\mathfrak{A}}$ i.e. by the set $\{t \in A \mid \mathfrak{A} \models \zeta^*[t]\}$ (see Definition 3.10). Thus the usual notion of sequence (see Section 1.7) has to be modified in the following way.

Definition 4.1. Let $\mathfrak{A} \in Md(Ax^*)$ and let D be an initial segment of inner time $T_{\mathfrak{A}}$. A function $s: D \to A$ is a *finite* (in $\mathfrak{A}$) sequence. If $D = [0, n-1]$ then s is called an *n-long sequence*. A function $s: T_{\mathfrak{A}} \to A$ is said to be a *sequence* or a *ζ-long sequence*. □

Let $K \subset A \setminus \{0\}$ be a definable set and let $^*K$ denote the set of all parametrically definable finite (in $\mathfrak{A}$) sequences containing only elements of K. Let Λ denote the empty sequence.

Lemma 4.2. "To be an element of $^*K$" is definable in $\mathfrak{A}$.

Proof. Let φ be the formula which defines K in $\mathfrak{A}$, i.e. $\mathrm{Var}\,\varphi = \{x\}$ and $\mathfrak{A} \models \varphi[a] \Leftrightarrow a \in K$.

Let us consider the following formula:

$$\varphi^*(s) \overset{\mathrm{d}}{=} \exists t(\mathit{left}(s) \leqq t \wedge \zeta(t) \wedge \forall i(i \leqq \mathit{left}(s) \rightarrow \varphi[\Gamma(\mathit{right}(s), \mathit{left}(s), i)/x])),$$

where Γ is the well-know Gödel-function (see e.g. in Mendelson (1964)). Since the functions Γ, *left* and *right* are definable in Ax^* the formula φ^* is equivalent to a formula of $F^V_{\mathfrak{d}^*}$. For the sake of convenience we suppose that a formula φ^* (e.g. $i \leqq \mathit{left}(a)$) at the same time denotes the corresponding formula of $F^V_{\mathfrak{d}^*}$, i.e. $\varphi^* \in F^V_{\mathfrak{d}^*}$.

Now let $s \in A$ be an element such that

$$\mathfrak{A} \models \varphi^*[s].$$

Let us take the following sequence:

$$f_s(i) \overset{\mathrm{d}}{=} \Gamma(\mathit{right}(s), \mathit{left}(s), i) \quad \text{for any} \quad i \leqq \mathit{left}(s).$$

Now we prove that $s \mapsto f_s$ is a surjective map, i.e. the elements of $^*K$ are coded by s but these codes are not unique.

Let $f\colon [0, n] \rightarrow K$ be a parametrically definable function in $\mathfrak{A}$. By using the generalized Sequence Number Theorem (see Gergely and Úry (1978), Theorem 2.8) there is a $b \in A$ such that for any $i \in [0, n]$, $f(i) = \Gamma(b, n, i)$.

Let $s \overset{\mathrm{d}}{=} \mathit{pair}(c, b)$. If f is finite in $\mathfrak{A}$ then by using the fact that Ax^* is a system and the definition of Γ we have $\mathfrak{A} \models \zeta^*[s]$ i.e. there is a $t \in A$ such that $s \leqq t$ and $\mathfrak{A} \models \zeta[s]$. Thus $\mathfrak{A} \models \varphi^*[s]$ and s codes the given function f. □

Lemma 4.3. The following functions are parametrically definable in $\mathfrak{A}$ (and also in Ax^*)

$$\mathit{pair}\colon {}^*K \times K \rightarrow {}^*K,$$

$$\mathit{left}\colon {}^*K \rightarrow {}^*K,$$

$$\mathit{right}\colon {}^*K \rightarrow A. \quad \square$$

Note that here the functions *pair, left* and *right* are defined on the sequences. However we use the same notation as on page 355 because the present case can be obtained by iterative usage of the functions introduced there. Thus this notion does not lead to ambiguity. For the sake of convenience we suppose that the empty sequence Λ is coded by 0.

Further on we do not distinguish the elements of $^*K$ from the corresponding elements of A coding them.

Definition 4.4.

$$\mathit{length}(a) = \begin{cases} 0 & \text{if} \quad a = \Lambda \\ \mathit{length}(\mathit{left}(a)) + 1 & \text{otherwise} \end{cases}$$

Definition 4.5. A set $V \subset {}^*(A \setminus \{0\})$ is called a tree iff

(i) $\Lambda \in V$,

(ii) $v \in V \Rightarrow left\,(v) \in V$.

For any V and $v \in V$ take $S_V(v) \overset{d}{=} \{w \in V \mid left(w) = v\}$.

A tree V is called a *path* iff *left*: $V \setminus \{\Lambda\} \to V$ is an injection, i.e. any $v \in V$ has at most one successor.

A tree V is *definable in* $\mathfrak{A}$ iff it is parametrically definable in $\mathfrak{A}$ as a unary relation on A. □

Note that because 0 codes Λ it is sufficient to use $A \setminus \{0\}$ instead of A in the above definition. Thus 0 can be maintained to denote the end of a sequence.

Definition 4.6. A *trace in* $\mathfrak{A}$ is a function of the form f: $V \to {}^nA$ (for some natural number n) where V is a tree in $\mathfrak{A}$. If f is parametrically definable then it said to be a *definable trace in* $\mathfrak{A}$. □

Definition 4.7. Let V be a trace in $\mathfrak{A}$. A *game-frame* in $\mathfrak{A}$ is a pair $GF = (V, C)$ where $C \subset V$ is an arbitrary set. C is called the nodes of choice of V. A run in a *game-frame* $GF = (V, C)$ is a subtree $R \subset V$ such that

(i) if $c \in C \cap R$ then c has a unique successor in R,

(ii) if $r \in R \setminus C$ then $S_V(r) = S_R(r)$.

A *strategy in a game-frame* G is a function *str*: $C \to \operatorname{rg} str$ in such a way that for any $c \in C$, $str\,(c) \in S_V(c)$.

A game-frame is *definable* in $\mathfrak{A}$ iff V and C are parametrically definable. A run R of GF is *definable* in $\mathfrak{A}$ if both GF and R are parametrically definable in $\mathfrak{A}$.

A strategy *str* in GF is *definable* in $\mathfrak{A}$ iff GF itself and the function *str* are parametrically definable. □

Definition 4.8. Let $GF = (V, C)$ be an arbitrary game-frame and let *str* be a strategy in GF. A run R is said to be generated by the strategy *str* if $\{str(c)\} = S_R(c)$ for any $c \in C \cap R$. □

We note that if *str* is definable then there exists a minimal definable run generated by *str* and this is denoted by R_{str}.

Definition 4.9. A tree V in $\mathfrak{A}$ is *finitary* iff for any $v \in V$ there is a $d \in A$ such that

1) $\mathfrak{A} \models \zeta[d]$,

2) if *pair* $(v, e) \in S_V(v)$ then $e \leqq d$, i.e. any node of V has only finitely many successors (in $\mathfrak{A}$). □

Definition 4.10. A *game* in $\mathfrak{A}$ is a quadruple $G = (V, C, \Gamma_A, \Gamma_B)$ where (V, C) is a game-frame and Γ_A, Γ_B are disjoint sets of paths in V.

A game $(V, C, \Gamma_A, \Gamma_B)$ is definable in $\mathfrak{A}$ iff

(i) (V, C) is a definable game-frame,

(ii) Γ_A and Γ_B are definable sets,

(iii) each path of Γ_A and Γ_B is definable. □

Definition 4.11. A strategy *str* in the game-frame $GF = (V, C)$ is a winning (non-losing) strategy of player A in the game $(V, C, \Gamma_A, \Gamma_B)$ iff each definable path of R_{str} belongs to Γ_A (not to Γ_B). □

According to this definition a run R of the game is non-losing for player A if none of its paths belong to any of Γ_B and R.

Thus a stategy of the player A is a winning (non-losing) one in the game $(V, C, \Gamma_A, \Gamma_B)$ iff moving accordingly the course of game realizes only paths belonging to Γ_A (not belonging to Γ_B). If we are interested only in non-losing strategies then it is enough to consider games of the form $(V, C, \emptyset, \Gamma_B)$.

In order to show that among others the property "to be a definable tree" is expressible by a first order formula we need the following well known theorem about the existence of a universal formula, though the precise form of this formula is not necessary to our investigation.

Theorem 4.12. (On universal formula.) Let us fix an arbitrary ϑ-type system Ax^*. There is a recursive map $F^V_{\vartheta*} \to \mathbf{N}$ ($\varphi \mapsto \ulcorner\varphi\urcorner$ where $\ulcorner\varphi\urcorner$ denotes the Gödel number of the formula φ) and a formula *Valid* $(g, \vec{a}, x) \in F^V_{\vartheta*}$ such that for any $\mathfrak{A} \in Md(Ax^*)$ and $\varphi \in F^V_{\vartheta*}$, $\mathfrak{A} \models \varphi[\vec{a}, x]$ iff $\mathfrak{A} \models \mathit{Valid}[\ulcorner\varphi\urcorner, \vec{a}, x]$.

Proof. See in Mendelson (1964).

Theorem 4.13. (Expressibility.) For any Ax^* the following properties are expressible in Ax^* by using the formulas of $F^V_{\vartheta*}$:

a) "to be a definable tree",

b) "to be a definable path in a definable tree",

c) "to be a definable game-frame",

d) "to be a definable run in a definable game-frame",

e) "to be a definable strategy in a definable game-frame",

f) "to be a non-losing definable run of a definable game",

g) "to be a non-losing definable strategy of a definable game",

h) "there is a non-losing definable strategy in a definable game".

Proof. Each statement can be proved by the same method. Thus we detail only the proof of property a) by showing the existence of the formula corresponding to this case.

a) Let Ω be the variable symbol the values of which correspond to the Gödel number of the formula φ that parametrically defines a tree in $\mathfrak{A}$ and let $\vec{a}$ be the vector of variable symbols the values of which correspond to the parameters.

The property "to be an element of tree V" can be defined by using the formula *Valid* (see Theorem 4.13). Namely *Valid* $(\Omega, \vec{a}, x)$ means that the element x is an element of the tree Ω with parameter $\vec{a}$ (here of course we identify the definable objects with the Gödel number of the appropriate formulas defining them). Having this defining formula we can construct the formula defining the property a). Namely

$$\mathit{Tree}(\Omega, \vec{a}) \overset{\mathrm{d}}{=} \mathit{Valid}(\Omega, \vec{a}, \Lambda) \wedge \forall x \big(\mathit{Valid}(\Omega, \vec{a}, x) \to \mathit{Valid}(\Omega, \vec{a}, \mathit{left}(x)\big).$$

Let $\mathfrak{A} \in Md(Ax^*)$. Now for fixed $\underset{\sim}{\Omega}, \underset{\sim}{\vec{a}} \in A$ take

$$V_{\underset{\sim}{\Omega}\underset{\sim}{\vec{a}}} \overset{\mathrm{d}}{=} \{v \in A \mid \mathfrak{A} \models \mathit{Valid}[\underset{\sim}{\Omega}, \underset{\sim}{\vec{a}}, v]\}.$$

It is clear that $V_{\underset{\sim}{\Omega}, \underset{\sim}{\vec{a}}}$ is a parametrically definable tree in $\mathfrak{A}$. Moreover if φ defines a tree V by parameters $\vec{a}$ then, using 4.12, we have $V = V_{\ulcorner\varphi\urcorner, \vec{a}}$.

b) Let $Path(\Omega, g, \vec{a}, \vec{b}) \overset{d}{=} \forall x (Valid(g, \vec{b}, x) \to Valid(\Omega, \vec{a}, x)) \wedge$

$$\forall x \, \forall y \, \{Valid(g, \vec{b}, x) \wedge Valid(g, \vec{b}, y) \wedge left(x) = left(y) \to x = y\}.$$

It is clear that if $\mathfrak{A} \in Md(Ax^*)$ and $\underset{\sim}{\Omega}, \underset{\sim}{g}, \underset{\sim}{a}, \underset{\sim}{\vec{b}} \in A$ then $V_{\underset{\sim}{g}, \underset{\sim}{\vec{b}}}$ is a definable path in $V_{\underset{\sim}{\Omega}, \underset{\sim}{\vec{a}}}$. We omit the proof of properties c), e) and f).

d) Let $GF(\Omega_V, \Omega_C, \vec{a})$ be the formula corresponding to property c)

$$R(\Omega_V, \Omega_C, \vec{a}, r, \vec{b}) \overset{d}{=} GF(\Omega_V, \Omega_C, \vec{a}) \wedge Tree(r, \vec{b}) \wedge \forall x [Valid(r, \vec{b}, x) \to$$
$$Valid(\Omega_V, \vec{a}, x)] \wedge \forall x \forall y [Valid(r, \vec{b}, x) \wedge Valid(\Omega_V, \vec{b}, y) \wedge left(y) = x \wedge$$
$$\neg Valid(\Omega_c, \vec{a}, x) \to Valid(r, \vec{b}, y)] \wedge$$
$$\forall x \, \forall y \, \forall z \, [Valid(r, \vec{b}, x) \wedge Valid(r, \vec{b}, y) \wedge Valid(r, \vec{b}, z) \wedge$$
$$left(y) = x \wedge left(z) = y \wedge Valid(\Omega_C, \vec{a}, x) \to y = z].$$

It is evident that this formula is good.

g) Let

$$NL(\Omega_V, \Omega_C, \Omega_{\Gamma_B}, \vec{a}) \overset{d}{=} \exists r \, \exists \vec{b} \, (R(\Omega_V, \Omega_C, \vec{a}, r, \vec{b}) \wedge$$
$$\forall g \, \forall \vec{c} \, (Path(r, g, \vec{b}, \vec{c}) \to \neg Valid(\Omega_{\Gamma_B}, pair(\vec{a}, \vec{c}), g))).$$

This formula means that there is a run r with parameters $\vec{b}$ such that r is a run of the game defined by Ω_V and Ω_C and no path g with parameter $\vec{c}$ of r belongs to Ω_{Γ_B}. □

Now we reformalize the well-know König's lemma for the case of definable finitary trees.

Lemma 4.14. (König.) Let V be a parametrically definable and finitary tree in $\mathfrak{A}$ such that for any $d \in A$ satisfying $\mathfrak{A} \models \zeta^*[d]$ there is a d-long definable path. Then there is a ζ-long definable path as well.

Proof. Any proof of the original form of this lemma can be repeated because the definability of the tree and paths provides the expressibility of each step of the proof in the first order language. Details are omitted. □

5. Nondeterministic programming language

We introduce a nondeterministic programming language of the autocode level. This level provides a relatively simple definition of semantics considering time conditions as well. This description is done by using games introduced in Section 4. Having exact semantics we turn to the investigation of the question of descriptive languages, which is one of the foundamental component of a mathematically based programming theory.

Now we introduce a ϑ-type nondeterministic programming language NP_ϑ. The programs of the language NP_ϑ might be "executed" in the models of on arbitrary ϑ-type system. Altogether the "meaning" of the programs varies by systems and by their models.

Let us fix a denumerable infinite set Y of variable symbols.

Definition 5.1. The set U_ϑ of ϑ-type nondeterministic commands consists of the following elements:

(i) j: $y \leftarrow \tau$,

(ii) j: $\Box y \leqq \tau$,

(iii) j: if χ then $\Box$ $k_1, \ldots, k_r$

where $\Box \in \{any, every\}$, $j, k_1, \ldots, k_r$ are natural numbers, $y \in Y$, $\tau \in T_\vartheta^Y$ and $\chi \in Q_\vartheta^Y$. □

Further on sign □ in commands stands either for *any* or *every*.

Thus the commands of U_ϑ have the form j: u, where the natural number j is said to be the *label* of the command.

Definition 5.2. A ϑ-type nondeterministic program p is a nonempty sequence of ϑ-type commands $p = \langle i_0: u_0, \ldots, i_n: u_n \rangle \in U_\vartheta^+$ in which no two commands have the same labels, i.e. for any $j, k \in [0, n]$ if $j \neq k$ then $i_j \neq i_k$.

Let NP_ϑ denote the set of all ϑ-type nondeterministic programs:

$NP_\vartheta \overset{d}{=} \{p \mid p$ is a ϑ-type nondeterministic program$\}$. □

Note that we often write a program p in the form of column

$$\begin{matrix} i_0 : u_0 \\ \cdots \\ i_n : u_n \end{matrix}$$

instead of the form of row. In the column form we always assume that $i_0 < \ldots < i_n$.

For any nondeterministic program $p = \langle i_0: u_0, \ldots, i_n: u_n \rangle \in NP_\vartheta$ we use the following notations:

(i) $\operatorname{Var} p \overset{d}{=} \bigcup_{j \in [0,n]} \operatorname{Var}(i_j: u_j)$ denotes the set of variable symbols occuring in the program p. Here $\operatorname{Var}(i_j: u_j)$ is the set of variable symbols occuring in the command i_j: u_j defined as follows

$$\operatorname{Var}(j: y \leftarrow \tau) \overset{d}{=} \operatorname{Var} \tau \cup \{y\},$$

$$\operatorname{Var}(j: \Box y \leqq \tau) \overset{d}{=} \operatorname{Var} \tau \cup \{y\},$$

$$\operatorname{Var}(j: \textit{if } \chi \textit{ then } \Box\, k_1, \ldots, k_r) \overset{d}{=} \operatorname{Var} \chi.$$

For the sake of convenience we often use Y_p instead of $\operatorname{Var} p$.

(ii) $i_{n+1} \overset{d}{=} \min\{k \mid k \notin \{i_0, \ldots, i_n\}\}$.

The programming language NP_ϑ seems to be far too weak though it is powerful enough as shown in its deterministic counter part in Gergely and Úry (1978). Now we note only the command j: *if* $x = x$ *then* □ $k_1, \ldots, k_n$ is the same as j: *goto* □ $k_1, \ldots, k_n$. We use the latter in the language NP_ϑ as an abbreviation.

Since the "meaning" of a program is its "execution" let us consider the case when the execution takes place in an arbitrary model of a given ϑ-type system Ax^*.

Definition 5.3. Let us fix a system Ax^*. Let us consider an arbitrary model $\mathfrak{A}\in Md(Ax^*)$ and an arbitrary program $p=\langle i_0\colon u_0, \dots, i_n\colon u_n\rangle\in NP_\vartheta$.

Let $q\colon \mathrm{Var}\, p\to A$ be an arbitrary evaluation of the variable symbols of p.

Let $G_q=(V_q, C_q)$ be a game-frame and $f_q=(l, s)\colon V_q\to \mathbf{N}\times{}^{\mathrm{Var}\, p}A$ be a trace with the following properties:

1. G_q and f_q are parametrically definable in $\mathfrak{A}$;
2. (i) $\mathfrak{A}\models\zeta^*\,[\mathit{length}\,(v)]$ for any $v\in V_q$,
(ii) $l(0)=i_0$ and $s(0)=q$,
(iii) if $l(v)\notin\{i_m \mid m\leqq n\}$ then there exists no successor of v, i.e. $S_{V_q}(v)=\emptyset$;
3. Let us suppose that $l(v)=i_m$

(i) if $u_m = y\leftarrow\tau$ then $S_{V_q}(v) = \{\mathit{pair}\,(v, 0)\overset{\mathrm{d}}{=} w\}$, $l(w) = i_{m+1}$,

$$s(w)(x)=\begin{cases} s(v)(x) & \text{if } \; x\in \mathrm{Var}\, p\setminus\{y\},\\ \tau_{\mathfrak{A}}[s(v)] & \text{if } \; x=y,\end{cases}$$

(ii) if $u_m=\square y\leqq\tau$ then $S_{V_q}(v)=\{\mathit{pair}\,(v, e)\mid 0\leqq e\leqq\tau_{\mathfrak{A}}[s(v)]\}$ and for any $w\in S_{V_q}$, $l(w) = i_{m+1}$,

$$s(w)(x)=\begin{cases} s(v)(x) & \text{if } \; x\in \mathrm{Var}\, p\setminus\{y\},\\ \mathit{right}\,(w) & \text{if } \; x=y,\end{cases}$$

(iii) if $u_m=$ *if* χ *then* $\square$ $k_1, \dots, k_r$ then
a) if $\mathfrak{A}\models\neg\chi[s(v)]$ then $S_{V_q}(v)=\{\mathit{pair}\,(v, 0)\}$ and $l(w)=i_{m+1}$ for $w\in S_{V_q}(0)$,
b) if $\mathfrak{A}\models\chi[s(v)]$ then $S_{V_q}(v)=\{\mathit{pair}\,(v, k_j)\mid j\in[1, r]\}$ and $l(w)=\mathit{right}\,(w)$ for any $w\in S_{V_q}(v)$.

In both cases a) and b) for any $w\in S_{V_q}(v)$, $s(w)=s(v)$.

The set of nodes of choice in the cases (ii) and (iii) is defined as follows

$$C_q\overset{\mathrm{d}}{=}\{v\in V_q \mid l(v)=i_m \text{ and } u_m \text{ is either } \mathit{any}\;\; y\leqq\tau \text{ or } \mathit{if}\;\chi\;\mathit{then\ any}\;\; k_1, \dots, k_r\}.$$

The game-frame of the above properties is said to be the *q-game-frame associated* to the program p; the trace f_q of above properties is said to be a *trace* of the program p in the model $\mathfrak{A}$ starting with input data q. □

Now we take the points of the definition one by one and show what conditions of programming are indicated by them. The q-game frame (V_q, C_q) describes the nondeterminism of the trace of a program; the function $f_q=(l, s)$ shows the actual value of the variable symbols of the program in each moment of execution.

If the program gets in a state represented by the node $v\in V_q$ then in the next step the command labelled by $l(v)$ will be executed with the $s(v)$ evaluation of the variable symbols.

Condition 1 is in accordance with the assumption that the first order language is used to describe the ϑ-type models.

Condition 2 (ii) provides that the game goes on for time $T=\zeta^*$ and it only stops when the situation 2 (iii) arises, i.e. when the control $l(v)$ gets such a label that does not occur among $i_0, \dots, i_n$. So the termination of the trace is equivalent to a jump onto such label. This is the reason why a special command *stop* is not included in U_ϑ.

According to 2 (ii) a trace starts always with the first command of the program, i.e. with the label i_0 with an evaluation q given in advance. This is the reason why a *start* command is not used.

Conditions of point 3 describe the one step transition of the program execution in the usual way. If a no control command with label i_m is executed then after the execution the control gets on label i_{m+1}. This is why the virtual label i_{n+1} was introduced to indicate the termination of the execution.

While executing commands with nondeterminism, i.e. with the sign $\Box$, the game-tree branches out according to the possible choices. The cases *any* and *every* differ from one another simply whether player A or B moves. The correctness of the above definition is ensured by the following

Lemma 5.4. For an arbitrary program $p\in NP_\vartheta$, $\mathfrak{A}\in Md(Ax^*)$ and q: $\mathrm{Var}\, p\to A$ there is a unique trace of p in $\mathfrak{A}$ starting with q and there is a unique associated q-game-frame GF_q. Moreover, the parameters of the formulas that define the trace f_q and the game-frame GF_q are the values of q and these definitions are universal (i.e. in each model of Ax^* they are defined by the same formula).

Proof. The uniqueness means that the associated game-frame GF_q and the corresponding trace f_q are unique. Let GF'_q, and f'_q be another q-game-frame and another trace for which conditions 1—3 of Definition 5.3 hold. Let v be an element of V with minimal length such that $v\notin V'$ or $v\in V'$ but $f_q(v)\neq f'_q(v)$. This minimum exists because both (GF_q, f_q) and (GF'_q, f'_q) are parametrically definable. Since $v\neq \Lambda$ so $w=left\,(v)\in V$ and by the minimality of v we have $w\in V'$ and $f(w)=f'(w)$. If so then by using conditions 1—3 we have that $v\in V'$ also holds and $f(v)=f'(v)$ which is a contradiction. To prove the existence of the trace we construct a recursive equation, the solution of which gives a trace of p in $\mathfrak{A}$ starting with q.

$$\varrho(v, l, y_1, \dots, y_k, a_1, \dots, a_k)\leftrightarrow \Big(l = i_0 \wedge v = \Lambda \wedge \bigwedge_{j=1}^{k} y_j = a_j\Big)$$

$$\vee\, \exists v^*, l^*, y_1^*, \dots, y_k^* \{\varrho(v^*, l^*, y_1^*, \dots, y_k^*, a_1, \dots, a_k) \wedge l^* \in \{i_m | m \leqq n\} \wedge \xi^*(\mathit{length}\, v) \wedge$$

$$\bigwedge_{\substack{m=0\\ u_m = y_s\leftarrow\tau}}^{n} l^* = i_m \to \Big\{v = (v^*, 0) \wedge \bigwedge_{\substack{i\neq s\\ i=1}}^{k} y_i^* = y_i \wedge y_s = \tau[\vec{y}^*/\vec{y}] \wedge l = i_{m+1}\Big\} \wedge$$

$$\bigwedge_{m=0}^{n} l^* = i_m \to \Big\{v = (v^*, y_s) \wedge \bigwedge_{\substack{i\neq s\\ i=1}}^{k} y_i^* = y_i \wedge y_s = \tau[\vec{y}^*/\vec{y}] \wedge l = i_{m+1}\Big\} \wedge$$

$$\bigwedge_{\substack{m=0\\ u_m = \mathit{if}\ \chi\ \mathit{then}\ \Box k_1, \dots, k_r}}^{n} l^* = i_m \to \Big[\bigwedge_{i=1}^{k} y_i^* = y_i \wedge (\neg\chi \to v = (v^*, 0) \wedge l = i_{m+1}) \wedge$$

$$(\chi \to \bigvee_{i=1}^{r} v = (v^*, k_i) \wedge l = k_i)\Big]\Big\}.$$

Applying Theorem 3.15 this equation has a minimal solution say $\varrho^* \in F_{\vartheta^*}$. It is eviden that

$$\Omega_V(w, a) \overset{\mathrm{d}}{=} \exists l, y_1, \ldots, y_k \, \varrho^*(w, l, y_1, \ldots, y_k, a_1, \ldots, a_k)$$

defines a tree in $\mathfrak{A}$ and if

$$\Omega_C(w, \vec{a}) \overset{\mathrm{d}}{=} \exists l, y_1, \ldots, y_k \left[\varrho^*(v, l, y_1, \ldots, y_k, a_1, \ldots, a_k) \wedge \bigvee_{\substack{m=0 \\ u_m = any\ y_s \\ u_m = if\ \chi\ then\ any\ k_1, \ldots, k_r}}^{n} l = i_m \right]$$

then (Ω_V, Ω_C) defines the associated q-game-frame GF_q and ϱ^* parametrically defines the trace of p in $\mathfrak{A}$ starting with q.

Note that since the solution of the above recursive equation is the same in any model of Ax^* thus ϱ^* defines the trace of p in any model of Ax^*. The universality of the definition of GF_q is evident from the construction. □

6. Descriptive language

Let Ax^* be a ϑ-type system and $\mathfrak{A} \in Md(Ax^*)$ an arbitrary model. Let $p = (i_0\colon u_0, \ldots, i_n\colon u_n) \in NP_\vartheta$ and $q\colon \mathrm{Var}\, p \to A$ an arbitrary input. We emphasize that the input q is arbitrary but fixed. The q-game-frame associated with the program p is $GF_q = (V_q, C_q)$.

Let Γ_B be an arbitrary set of winning paths of player B in game-frame GF_q and consider the game $(V_q, C_q, \emptyset, \Gamma_B)$. From the point of view of the theory of programming it is of great significance to consider whether player A has a non-losing strategy in the game $(V_q, C_q, \emptyset, \Gamma_B)$. The meaning of this consideration for the theory depends on what set Γ_B is selected. To illustrate the significance of the existence of a not losing strategy of A we consider the game $(V_q, C_q, \emptyset, \Gamma_B)$ with different Γ_B.

A. Let Γ_B be the set of those definable paths of the game-frame GF_q in which there exists a situation (node) v such that $l(v) \notin \{i_m \mid m \leqq n\}$, i.e. Γ_B consists of the terminating definable paths of V_q.

In this case "to have a not losing strategy of the player A" means that A is able to ensure that the program execution never "dies", i.e. it runs infinitely long (more precisely ζ-long). □

B. Let $\psi \in F_\vartheta^{X_p}$ be an arbitrary formula that is called the output condition and let Γ_B be the set of those definable paths of V_q each of that contains a node v such that

(i) $l(v) \notin \{i_m \mid m \leqq n\}$ and
(ii) $\mathfrak{A} \not\models \psi[s(v)]$.

So Γ_B consists of those executions of the program p that terminate without satisfying the output condition ψ.

In this case the non-losing strategy of A means the partial correctness of the program p with respect to the output condition ψ (while the input condition is any tautology e.g. $x = x$). □

C. Let $\varphi, \psi \in F_{\vartheta}^{X_p}$ be arbitrary formulas that are called input and output conditions respectively and let Γ_B be the set of those definable paths of V_q for each of which:

(i) $\mathfrak{A} \models \varphi[s(0)]$,

(ii) there exists a node (in the path) such that

$$l(v) \notin \{i_m | m \leqq n\} \quad \text{and} \quad \mathfrak{A} \not\models \psi[s(v)].$$

In this case the non-losing strategy of A means the partial correctness of the program p w.r.t. the input condition φ and output condition ψ. Note that we can suppose that if the program executed by input q which does not satisfy the condition φ then $\Gamma_B = \emptyset$. □

D. Let $\varphi, \psi \in F_{\vartheta}^{X_p}$ be arbitrary formulas and let Γ_B be the set of definable paths of V_q such that each satisfies both

(i) $\mathfrak{A} \models \varphi[s(0)]$ and

(ii) either it does not terminate or it has a node v such that $l(v) \notin \{i_m | m \leqq n\}$ and $\mathfrak{A} \not\models \psi[s(v)]$.

Now the non-losing strategy of A means the total correctness of the program p w.r.t. the input condition φ and output condition ψ. □

Before proceeding to the next cases we introduce further notations. The value of the input is subject to change while executing a program, i.e. during the corresponding computing process. Thus in order to describe program properties during execution we need both the input values and the actual values of program variables. For any fixed program p we distinguish a set $X_p \subset Y$ of variable symbols that duplicates the variable symbols Y_p and refers to the input values of the latter.

If $Y_p = \{y_1, \dots, y_k\}$ then $X_p = \{x_1, \dots, x_k\}$ and for any $i \in [0, n]$, x_i duplicates the corresponding y_i.

Moreover let t be a distinguished variable symbol of Y which will be used to describe time conditions. A formula $\varphi \in F_{\vartheta *}^{Y}$ is said to be an input condition for a fixed program p iff $\mathrm{Var}\, \varphi \subset X_p$. Moreover, a formula $\psi \subset F_{\vartheta *}^{Y}$ is called output condition for the program p iff $\mathrm{Var}\, \psi \subset X_p \cup Y_p \cup \{t\}$.

Now let us consider the further cases.

E. Let φ and ψ be input and output conditions for a fixed program p respectively and let Γ_B be the set of those definable paths of V_q which satisfy $\mathfrak{A} \models \varphi[q]$ and have a node v such that $l(v) \notin \{i_m | m \leqq n\}$ and $\mathfrak{A} \not\models \psi[q, s(v), \mathit{length}\,(v)]$.

In this case the existence of a non-losing strategy of player A is of the following meaning. An execution of program p carried out in accordance whith the strategy is such that it starts with input q that satisfies condition φ and when it stops it satisfies condition ψ. □

F. Let φ and ψ be input and output conditions for program p respectively and let Γ_B be the set of such definable paths of V_q that each of them satisfies $\mathfrak{A} \models \varphi[q]$ and it is either infinite or there exists a node v such that $l(v) \in \{i_m | m \leqq n\}$ and $\mathfrak{A} \not\models \psi[q, s(v), \mathit{length}\,(v)]$.

In this case the existence of a non-losing strategy of player A means the following. An execution of program p done in accordance with the strategy starts with input q satisfying condition φ and it terminates by satisfying condition ψ. □

We could go on listing cases infinitely, but we hope that the aboves represent the basic idea well.

However each of the above cases represents its own situation in the theory of programming, having its application in the mentioned way. From theoretical point of view it would be very useful to have a unique descriptive language for any version of the set Γ_B, i.e. this language has to be suitable to define sets of paths in trees. But for the time being we have not such a universal descriptive language. Thus for each Γ_B we have to introduce a new descriptive language remaining within the same definable game-frame (V_q, C_q).

Corresponding to aboves a theory of nondeterministic programming though having one programming language has to possess several descriptive languages each of which fits in to certain conditions corresponding to some pragmatical aims.

To illustrate this theory we give the appropriate descriptive language for the cases E and F. This language will be common for both cases. We have chosen these cases because in literature time consideration is hardly investigated.

First we introduce the syntax of the descriptive language.

Definition 6.1. $\mathbf{F}_\vartheta \overset{d}{=} \{(\varphi, p, \psi) | p \in NP_\vartheta$, φ and ψ are input and output conditions for p respectively$\} \cup \{[\varphi, p, \psi] | p \in NP_\vartheta$, φ and ψ are input and output conditions for p respectively$\}$. □

When we are interested only in the triple of φ, p, ψ without specifying the type of brackets that include them we write $|\varphi, p, \psi|$.

Now we are going to define the semantics of the descriptive language.

Let $\chi = |\varphi, p, \psi| \in \mathbf{F}_\vartheta$ and let $\mathfrak{A} \in Md(Ax^*)$. According to Lemma 5.4 for any q: $\mathrm{Var}\, p \to A$ there is a game-frame $G_p = (V_q, C_q)$ and a function $f_q = (l, s)$: $V_q \to A$ and they are definable in Ax^*. Starting out from the game-frame G_q and taking the formula χ into account the following game can be constructed:

(i) $\Gamma_A^\chi = \emptyset$,

(ii) if $\mathfrak{A} \not\models \varphi[q]$ then let $\Gamma_B^\chi = \emptyset$. In opposite case there are two possibilities:

a) First if $\chi = (\varphi, p, \psi)$ then $\Gamma_B^\chi \overset{d}{=} \{\gamma$ is a definable path in $V_q |$there is a $v \in \gamma$ such that $l(v) \notin \{i_m | m \leqq n\}$ and $\mathfrak{A} \not\models \psi[q, s(v), \mathit{length}\,(v)]\}$.

b) Secondly if $\chi = [\varphi, p, \psi]$ then let $\Gamma_B^\chi \overset{d}{=} \{\gamma$ is a definable path in $V_q | \gamma$ is either infinite or there is a $v \in \gamma$ such that $l(v) \notin \{i_m | m \leqq n\}$ and $\mathfrak{A} \not\models \psi[q, s(v), \mathit{length}\,(v)]\}$.

Thus we have got a game $G_q^\chi = (V_q, C_q, \emptyset, \Gamma_B^\chi)$ which is called *the associated q-game generated by χ.*

By using the technique used in the proof of Theorem 4.12 it is not difficult to verify that the set Γ_B^χ is parametrically definable in Ax^* in both of the above cases a) and b). Moreover, the value of q are used only as parameters in the defining formulas. Thus we have the following

Lemma 6.2. Let $\mathfrak{A} \in Md(Ax^*)$ be arbitrary, $\chi \in \mathbf{F}_\vartheta$ and q: $X_p \to A$. Then the associated q-game $(V_q, C_q, \emptyset, \Gamma_B^\chi)$ is parametrically definable in Ax^*. □

Now we turn to the definition of the semantics of the descriptive language to be defined.

Definition 6.3. Let $\mathfrak{A}\in Md(Ax^*)$ and $\chi\in\mathbf{F}_\vartheta$ be arbitrary. $\mathfrak{A}|\!\equiv\chi$ iff for any q: Var $p\to A$ player A has a non-losing definable strategy in the associated q-game G^χ_q. □

Note that if $\chi=(\varphi, p, \psi)$ $(\chi=[\varphi, p, \psi])$ then $\mathfrak{A}|\!\equiv\chi$ means the partial (respectively total) correctness of the program p in the model $\mathfrak{A}$ w.r.t. the input condition φ and output condition ψ.

Definition 6.4. For an arbitrary ϑ-type system Ax^* the descriptive language expressing partial and total correctness and describing the corresponding time condition is

$$\mathbf{D}_\vartheta^{Ax^*}\overset{\mathrm{d}}{=}(\mathbf{F}_\vartheta, Md(Ax^*), |\!\equiv).$$

Remark 6.5. Similarly to the fact that the associated q-game is parametrically definable, if $\mathfrak{A}|\!\equiv\chi$ then the non-losing strategy and the corresponding run are also parametrically definable in $\mathfrak{A}$.

What is the connection between the validity relation introduced above and the validity relation of Definition 3.5. The answer to this and the expressibility of some notions introduced above by the first order language is given by the following

Theorem 6.6. For any $\chi\in\mathbf{F}_\vartheta$ there is a formula $\chi^*\in F^X_{\vartheta*}$ such that for any $\mathfrak{A}\in Md(Ax^*)$, $\mathfrak{A}|\!\equiv\chi$ iff $\mathfrak{A}\models\chi^*$.

The function rendering χ^* to χ is recursive.

Moreover, there is a formula $\sigma_\chi(\vec{a}, v, w)\in F^X_{\vartheta*}$ such that if $\mathfrak{A}|\!\equiv\chi$ then for any q: Var $p\to A$, $\sigma_\chi[q]$ defines a non-losing strategy of player A in the associated q-game G^χ_q.

Proof. The existence of the formula χ^* for a given $\chi\in\mathbf{F}_\vartheta$ is an easy consequence of Theorem 4.9. The recursiveness of the function that renders the corresponding χ^* to each of them can be established by using Gödel-numbering. We omit the details. □

Let $E_0^{Ax^*}$ denote the set of all formulas that are true in the models of Ax^*, i.e. $E_0^{Ax}\overset{\mathrm{d}}{=}\{\chi\mid Ax^*|\!\equiv\chi\}$.

From the point of view of the usability of the descriptive language $\mathbf{D}_\vartheta^{Ax^*}$ for the theory of programming the handlebility of $E_0^{Ax^*}$ is very important. This is established by the following

Theorem 6.7. (Completeness.) If the system Ax^* is recursively enumerable then so is set of formulas $E_0^{Ax^*}$, i.e. the descriptive language $\mathbf{D}_\vartheta^{Ax^*}$ is complete.

Proof. Immediate from 6.6. □

Remark 6.8. If $\mathfrak{A}|\!\equiv\chi$ would be defined so that for any input data q in the game G^χ_q player A had a non-losing strategy (without the assumption of definability) then the language $\mathbf{D}_\vartheta^{Ax^*}$ would lose the completeness, even more, it would be neither ω-complete nor complete relative to arithmetic in contrast with the case of deterministic programming (see Banachowski et al (1977) and Cook (1978), Pratt (1978) respectively).

7. A complete calculus

Being aware of the completeness of the language $\mathbf{D}_\vartheta^{Ax^*}$ we are going to introduce such a complete calculus which is close to the programmer's intuition. This calculus has to be convenient to prove about a given program p either its partial or total correctness w.r.t. given input and output conditions. To provide such a calculus is significant from the point of view of both theory and practice.

Theorem 6.6 shows that the syntax $\mathbf{F}_\vartheta$ can be coded in the syntax F_ϑ^Y however the proofs in the latter can not be interpreted directly by programming situations.

Below we introduce a calculus in the spirit of Floyd and Hoare.

First we introduce the following

Notation 7.1. Let *Lab*: $NP_\vartheta \to P(\mathrm{N})$ be the function rendering to each nondeterministic program $p=(i_0\colon u_0, \dots, i_n\colon u_n)\in NP_\vartheta$ the set of labels occuring in it as follows.

$Lab\,(p)=\{i_m \mid m\leqq n+1\}\cup\{k \mid$ there is an m such that $u_m = if\ \varkappa\ then\ \square \dots k \dots$ $\square\in\{\text{any, every}\}\}$.

Definition 7.2. Let us fix a ϑ-type system Ax^* and let $\chi=|\varphi, p, \psi|$ be an arbitrary formula of $\mathbf{F}_\vartheta$. Let Φ: $Lab\,(p)\to F_{\vartheta^*}$ be a function which to each label of the program p renders a formula $\Phi(i)$ with variable symbols $\vec{x}, \vec{y}, t\in Y$. Remember that $\vec{x}$ serves to duplicate the program variable $\vec{y}$ and t refers to time. Further on we write Φ_i instead of $\Phi(i)$. The function Φ is said to be the description of χ w.r.t. Ax^* iff it satisfies the following conditions:

(i) $Ax^*\models \varphi\to\Phi_{i_0}[0/t, \vec{x}/\vec{y}]$ (input rule)

(ii) if i_m: $y\leftarrow\tau$ then

$Ax^*\models \Phi_{i_m}\to\Phi_{i_{m+1}}[\tau/y, t+1/t]$ (assignment rule)

(iii) if i_m: *any* $y\leqq\tau$ then

$Ax^*\models \Phi_{i_m}\to\exists z\,(z\leqq\tau\wedge\Phi_{i_{m+1}}[z/y, t+1/t])$ (assignment rule of A's choice)

(iv) if i_m: *every* $y\leqq\tau$ then

$Ax^*\models \Phi_{i_m}\to\forall z\,(z\leqq\tau\wedge\Phi_{i_{m+1}}[z/y, t+1/t])$ (assignment rule of B's choice)

(v) if i_m: *if* $\varkappa$ *then any* $k_1, \dots, k_r$ then

$Ax^*\models \Phi_{i_m}\wedge\neg\varkappa\to\Phi_{i_{m+1}}[t+1/t]$, $\quad Ax^*\models \Phi_{i_m}\wedge\varkappa\to\bigvee_{j=1}^{r}\Phi_{k_j}[t+1/t]$

(rule of conditional jump of A's choice)

(vi) if i_m: *if* $\varkappa$ *then every* $k_1, \dots, k_r$ then

$Ax^*\models \Phi_{i_m}\wedge\neg\varkappa\to\Phi_{i_{m+1}}[t+1/t]$, $\quad Ax^*\models \Phi_{i_m}\wedge\varkappa\to\bigvee_{j=1}^{r}\Phi_{k_j}[t+1/t]$

(rule of conditional jump of B's choice)

(vii) $Ax^*\models \Phi_z\to\psi$ (output rule for $z\notin\{i_m \mid m\leqq n\}$)

Moreover if $\chi=[\varphi, p, \psi]$ then
(viii) for any $z\in\{i_m \mid m\leqq n\}$
$Ax^*\models\exists v\,\forall\vec{y}[\Phi_z\rightarrow v\geqq t]$ (rule of termination).
We denote by $Ax^*\vdash\chi$ the fact that the formula χ has a description w.r.t. Ax^*.
□

Now let us see some remarks on the above definition:

Each formula Φ_i $(0\leqq i\leqq n)$ shows the conditions the data should satisfy before executing the command labelled by i and the corresponding time conditions. We note that ζ can be used in any formula Φ_z.

The rule (i) shows that the execution of the program starts at the moment 0.

In the rules (ii)—(vi) the substitution $[t+1/t]$ denotes unambiguously that the execution of each command happens during one unit of time however complex e.g. the term τ is. This has already been supposed in the definition of the q-game associated to the program (see 5.3).

This assumption can be generalized without any trouble in the following way. We render to any command i_m: u_m an execution time t_m which can also depend on the data $\vec{y}$. Thus the game associated to the program continues a unique path during t_m time-units. Now this corresponds to the fact that the game stays in one and the same state. Using this generalization in the description of χ in every rule (ii)—(vi) instead of the substitution $[t+1/t]$ we write $[t+t_m/t]$.

The rule (vii) says that if the execution of the program p stops then it stops forever i.e. its time "stops" — the process "dies".

The rule (viii) is needed only to prove the total correctness of p.

In the descriptive formulas the usage of separate variable symbols x is allowed in order to refer to the input values. These are needed only in the formulas used in the description of total correctness. While proving the completeness we shall get a formula independent from x for the case $\chi=(\varphi, p, \psi)$, by using the definition $\Phi_z^*\overset{d}{=}\exists x\,\Phi_z$ for each descriptive formula Φ_z.

It is interesting that in the case of nondeterministic programming total correctness is not equal to partial correctness plus termination in contrast with the deterministic sequential case. Thus in our case the total correctness cannot be established by proving the partial correctness and the termination separately.

Now we show that the calculus introduced above is complete.

Theorem 7.3. (Completeness.) For any ϑ-type system Ax^* and $\chi\in\mathbf{F}_\vartheta$, $Ax^*\models\equiv\chi$ iff $Ax^*\vdash\chi$.

Proof. First we show that if $Ax^*\vdash\chi$ then $Ax^*\models\equiv\chi$.

Let us fix a $\chi=|\varphi, p, \psi|$ and let Φ: $Lab\,p\rightarrow F_{\vartheta*}^X$ be a description of χ w.r.t. Ax^*. Let $\mathfrak{A}\in Md(Ax^*)$ and q: $\mathrm{Var}\,p\rightarrow A$ be arbitrary. We must prove that in the q-game $G_q^\chi=(V_q, C_q, \emptyset, \Gamma_B^\chi)$ player A has a non-losing strategy.

Let (l, s): $V_q\rightarrow\mathbf{N}\times{}^{\mathrm{Var}\,p}A$ be the trace of p in $\mathfrak{A}$ starting with q. First we define a strategy for player A.

Let $v\in C_q$ be arbitrary. Since $v\in C_q$ for an $m\in[1, n]$ we have $l(v)=i_m$ and $u_m=$ $=$*any* $y\leqq\tau$ or $u_m=$*if* $\varkappa$ *then any* $k_1, \ldots, k_r$. If $\mathfrak{A}\not\models\Phi_{l(v)}[q, s(v), \mathit{length}\,v]$ then let $str(v)$ be the minimal element of $S_{V_q}(v)$. (It might be arbitrary but we have to be sure that *str* is parametrically definable.) So let us suppose that $\mathfrak{A} \models \Phi_{l(v)}[q, s(v), \mathit{length}(v)]$. First let $u_m = \mathit{any}\ y \leqq \tau$. By our assumption with $i_m = l(v)$

$$\mathfrak{A} \models \Phi_{i_m} \to \exists z (z \leqq \tau \wedge \Phi_{i_{m+1}}[\tau/y, t+1/t],$$

$\mathfrak{A} \models \Phi_{i_m}[q, s(v), \mathit{length}\, v]$ and thus

$$\mathfrak{A} \models \exists z (z \leqq \tau \wedge \Phi_{i_{m+1}}[\tau/y, t+1/t][q, s(v), \mathit{length}(v)].$$

Now let z^* be the minimal $\underset{\sim}{z} \in A$ such that

$$\mathfrak{A} \models \underset{\sim}{z} \leqq \tau \wedge \Phi_{i_{m+1}}[\tau/y, t+1/t][q, s(v), \mathit{length}(v), \underset{\sim}{z}].$$

It is clear that $\mathit{pair}(v, z^*) \in S_{V_q}(v)$ and so let

$$\mathit{str}(v) \overset{\mathrm{d}}{=} \mathit{pair}(v, z^*).$$

Obviously

$$\mathfrak{A} \models \Phi_{i_{m+1}}[q, s(\mathit{str}(v)), \mathit{length}(\mathit{str}(v))] \tag{1}$$

In the end let $u_m = \mathit{if}\ \varkappa\ \mathit{then\ any}\ k_1, \dots, k_r$. Since $v \in C_q$ so $\mathfrak{A} \models \varkappa[s(v)]$. Using $\mathfrak{A} \models \Phi_{i_m} \wedge \varkappa \to \bigvee_{j=1}^{r} \Phi_{k_j}[t+1/t]$ and $\mathfrak{A} \models \Phi_{i_m} \wedge \varkappa[q, s(v), \mathit{length}(v)]$ we have

$$\mathfrak{A} \models \bigvee_{j=1}^{r} \Phi_{k_j}[t+1/t][q, s(v), \mathit{length}(v)].$$

Let k^* be the minimal $k \in \{k_1, \dots, k_r\}$ for which

$$\mathfrak{A} \models \Phi_k[t+1/t][q, s(v), \mathit{length}(v)] \tag{2}$$

and take $\mathit{str}(v) \overset{\mathrm{d}}{=} \mathit{pair}(v, k^*)$.

Thus the strategy *str* is defined. Since G_q is definable and the method used in the aboves can be defined in $F^X_{\mathfrak{Z}^*}$ so *str* is also parametrically definable. The fact, that *str* is a non-losing strategy follows from the following

Lemma 7.4. Let π be a parametrically definable path in the run R_{str} of the strategy *str*. For every $v \in \pi$

$$\mathfrak{A} \models \Phi_{l(v)}[q, s(v), \mathit{length}(v)].$$

In fact if $l(v) \notin \{i_m | m \leqq n\}$ then

$$\mathfrak{A} \models \psi[q, s(v), \mathit{length}(v)].$$

Moreover, if $\chi = [\varphi, p, \psi]$ then π terminates.

Proof. The proof goes by induction on the *length* (v). If $v = \Lambda$ and $\mathit{length}(v) = 0$ then by using $\mathfrak{A} \models \varphi[q]$ and (i) of Definition 7.2, $\mathfrak{A} \models \Phi_{l(\Lambda)}[q, s(\Lambda), 0]$. The inductive step can be done by using (1), (2) and (ii)—(vi) of Definition 7.2. If $l(v) \notin \{i_m | m \leqq n\}$ then from (vii) of Definition 7.2 we get $\mathfrak{A} \models \psi[q, s(v), \mathit{length}(v)]$.

If $\chi = [\varphi, p, \psi]$ then let us apply (viii) of Definition 7.2 and we get a $\underset{\sim}{t} \in A$ such that if $\mathit{length}(v) > \underset{\sim}{t}$ then for any $z \in \{i_m | m \leqq n\}$, $\mathfrak{A} \models \Phi_z[q, s(v), \mathit{length}(v)]$. Hence using $\mathfrak{A} \models \Phi_{l(v)}[q, s(v), \mathit{length}(v)]$ we have $l(v) \notin \{i_m | m \leqq n\}$.

The proof of the lemma is completed. □

Now the proof of the theorem is continued by showing that if $Ax^* \models \chi$ then $Ax^* \vdash \chi$.

Let $\chi = |\varphi, p, \psi|$ be such that $Ax^* \models \chi$. Let $\Omega_V(\vec{a}, v)$ and $\Omega_C(\vec{a}, w)$ be the formulas which parametrically define the game-frame $GF=(V, C)$ in Ax^* generated by χ. By the second part of Theorem 6.6 there is a formula $\sigma_\chi(\vec{a}, v, w)$ with the same parameters as Ω_V, Ω_C which in Ax^* parametrically defines a non-losing strategy for player A in the game G_q^χ. Hence there is a formula $\varrho_\chi(\vec{a}, r) \in F_{\mathfrak{Z}^*}^X$ which defines in Ax^* the run of the strategy defined by σ_χ. Now we give a description of χ in Ax^*:

$$\Phi_z(\vec{x}, \vec{y}, t) \overset{\mathrm{d}}{=} \varphi(\vec{x}) \land \exists v\, \{\varrho_\chi(\vec{x}, v) \land l(v) = z \land \vec{y} = s(v) \land \mathit{length}(v) = t\}.$$

Since s is parametrically definable in Ax^* by using only the parameter $\vec{x}$ so $\Phi_z \in F_{\mathfrak{Z}^*}^X$. It is clear that $\mathrm{Var}\, \Phi_z \subset \{x_1, \ldots, x_k, y_1, \ldots, y_k, t\}$.

The following lemma is enough to complete the proof.

Lemma 7.5. The above function Φ is a description of χ w.r.t. Ax^*.

Proof. Let $\mathfrak{A} \in Md(Ax^*)$ and $q\colon \mathrm{Var}\, p \to A$ be arbitrary. We have to prove that for any $\vec{y}$, $\underset{\sim}{t} \in A$ the rules in Definition 7.2 hold. (i)—(vii) can be proved in the same way so e.g. we prove (iii). Let $\vec{y}, \underset{\sim}{t} \in A$ be such that $\mathfrak{A} \models \Phi_{i_m}[q, \underset{\sim}{\vec{y}}, \underset{\sim}{t}]$ and $u_m = \mathit{every}\ y \leqq \tau$. Let $G_q^\chi = (V_q, C_q, \emptyset, \Gamma_B^\chi)$ be the associated q-game generated by χ and let (l, s) be the trace of p in $\mathfrak{A}$. By the definition of Φ_{i_m} there is a $v \in V_q$ such that $\underset{\sim}{\vec{y}} = s(v)$ and $t = \mathit{length}(v)$. Moreover $v \in R_{str}$ where str is the non-losing strategy of player A defined by σ and q. Let $\underset{\sim}{z} \leqq \tau_{\mathfrak{A}}[s(v)]$ be arbitrary. Then $w = \mathit{pair}(v, \underset{\sim}{z}) \in R_{str}$, $\mathit{length}(w) = \underset{\sim}{t} + 1$,

$$s(w)(x) = \begin{cases} \underset{\sim}{z} & \text{if} \quad x = y, \\ s(v)(x) & \text{otherwise}; \end{cases}$$

and $l(v) = i_{m+1}$.

This means that

$$\mathfrak{A} \models \Phi_{i_m}[q, s(w), l(w)].$$

Hence

$$\mathfrak{A} \models \Phi_{i_m+1}[z/y, t+1/t][q, \underset{\sim}{\vec{y}}, \underset{\sim}{t}].$$

Summarizing:

$$\mathfrak{A} \models \Phi_{i_m} \to \forall z\, (z \leqq \tau \land \Phi_{i_{m+1}}[z/y, t+1/t]).$$

If $\chi = [\varphi, p, \psi]$ we must prove that for any $z \notin \{i_m \mid m \leqq n\}$,

$$\mathfrak{A} \models \exists T\, \forall \vec{y}\, \forall t\, [\Phi_z(\vec{y}, t) \to t \leqq T].$$

By the definition of G_q^χ and R_{str} in R_{str} any definable path terminates. It is clear that R_{str} is finitary and thus König's Lemma can be applied: there is a $\underset{\sim}{T} \in A$ such that for any $v \in R_{str}$, $\mathit{length}(v) \leqq \underset{\sim}{T}$. As in the first part of the proof for any $(\underset{\sim}{\vec{y}}, \underset{\sim}{t}) \in A$ with

$$\mathfrak{A} \models \Phi_z[q, \underset{\sim}{\vec{y}}, \underset{\sim}{t}]$$

there is a $v \in R_{str}$ such that $s(v) = \vec{y}$ and $\mathit{length}(v) = \underset{\sim}{t}$. Hence $\underset{\sim}{t} \leqq \underset{\sim}{T}$ and thus

$$\mathfrak{A} \models \forall \vec{y}\, \forall t\, [\Phi_z(\vec{y}, t) \to t \in T][q, \underset{\sim}{T}].$$

The proof of the lemma is completed and so is the proof of the theorem. □

8. Examples

8.1. The least common multiplier

Let us consider the following nondeterministic program that we would like to use for the computation of the least common multiplier of two positive integers:

$p_1 \overset{d}{=}$

```
0: any $y_1 \leqq \max(a, b)$
1: if $a|y_2 \wedge b|y_2 \wedge y_2 \neq 0$ then 5
2: if $y_2 > ab$ then 5
3: $y_2 \leftarrow y_2 + y_1$
4: goto 1
```

The input and output conditions are respectively

$$\varphi \overset{d}{=} a > 0 \wedge b > 0 \wedge y_2 = 0$$

and

$$\psi \overset{d}{=} y_2 = [a, b].$$

Let Ax be the set of axioms, which contains PA and the axiomatic definition of $|$, max, $<$, $\leqq$, and $[., .]$. Let

$$Ax^* \overset{d}{=} Ax \cup \{\forall x \zeta(x)\}.$$

We are going to prove that

$$Ax^* \models (\varphi, p_1, \psi).$$

i.e. in the label 0 the value of y_1 can be chosen so that if the program terminates then $y_2 = [a, b]$. Indeed let us consider the following description of χ w.r.t. Ax^*:

$$\Phi_0 \overset{d}{=} a > 0 \wedge b > 0 \wedge y_2 = 0$$

$$\Phi_1 \overset{d}{=} y_1 > 0 \wedge (y_1|a \vee y_1|b) \wedge \forall z (z \leqq y_2 \wedge y_1|z \rightarrow \neg z|a \vee \neg z|b)$$

$$\Phi_2 \overset{d}{=} y_1 > 0 \wedge (y_1|a \vee y_1|b)$$

$$\Phi_3 \overset{d}{=} \Phi_2 \wedge y_2 \leqq ab$$

$$\Phi_4 \overset{d}{=} \Phi_1$$

$$\Phi_5 \overset{d}{=} y_2 = [a, b]$$

It is easy to verify that the function described above is really the description of χ w.r.t. Ax^*.

It is obvious that the greater value is chosen for y_1 from that of satisfying the condition Φ_1 the sooner termination of the program occurs. That is why from the output condition we claim the expression of early termination. So let

$$\tilde{\psi} \overset{\mathrm{d}}{=} y_2 = [a, b] \wedge t \leqq \frac{4([a, b]+1)}{\max(a, b)}.$$

Now let us consider the description $\tilde{\Phi}$ of $\tilde{\chi}=[\varphi, p, \tilde{\psi}]$ w.r.t. Ax^*, where the time conditions are to be expressed in the descriptive formulas $\tilde{\Phi}_i$ $(0 \leqq i \leqq 5)$. The description $\tilde{\Phi}$ is a light modification of Φ as follows:

$$\tilde{\Phi}_0 \overset{\mathrm{d}}{=} a > 0 \wedge b > 0 \wedge y_2 = 0$$

$$\tilde{\Phi}_1 \overset{\mathrm{d}}{=} y_1 = \max(a, b) \wedge y_2 = \frac{t-1}{4} y_1 \wedge y_2 \leqq ab \wedge \forall z (z \leqq y_2 \wedge y_1 | z \rightarrow \neg z | a \vee \neg z | b)$$

$$\tilde{\Phi}_2 \overset{\mathrm{d}}{=} y_1 = \max(a, b) \wedge y_2 = \frac{t-2}{4} y_1 \wedge y_2 \leqq ab \wedge \forall z (z \leqq y_2 \wedge y_1 | z \rightarrow \neg z | a \vee \neg z | b)$$

$$\tilde{\Phi}_3 \overset{\mathrm{d}}{=} \tilde{\Phi}_2 [t-3/t]$$

$$\tilde{\Phi}_4 \overset{\mathrm{d}}{=} \tilde{\Phi}_1 [t+1/t]$$

$$\tilde{\Phi}_5 \overset{\mathrm{d}}{=} y_2 = [a, b] \wedge t \leqq \frac{4([a, b]+1)}{\max(a, b)}.$$

From the descriptions Φ and $\tilde{\Phi}$ it immediately follows that the command 2: *if* $y_2 > ab$ *then* 5 is really unnecessary. It is good exercise to prove that if in the program p_1 we write *every* instead of each occurence of *any* we get a program p_1' such that

$$Ax^* \models [\varphi, p_1', t \leqq 4ab+4].$$

8.2 Pattern matching

Now we show how the nondeterministic programming language NP_ϑ can be used for handling the problem of pattern matching. First let us specify what pattern-matching is.

Let $\leqq$ be a definable partial ordering in a fixed system Ax^*. Let $\mathfrak{A} \in Md(Ax^*)$. If for some $a, b \in A$, $a \leqq b$ then we say that b is defined at least that much as a. Let us give a vector $\langle X(j) \rangle_{j \leqq n}$ of $n+1$ elements. We would like to match to this vector a pattern from the matrix $\langle A(i, j) \rangle_{i \leqq m, j \leqq n}$.

More precisely, we would like to find such a row in the matrix each element of which is defined at least that much as the corresponding element of the vector $\langle X(j) \rangle_{j \leqq n}$.

Since vectors and matrices are not allowed in our language NP_ϑ explicitly we have to suppose that the type ϑ contains the function symbols *vcomp* and *mcomp*, for which $\vartheta(vcomp)=2$ and $\vartheta(mcomp)=3$. The function symbol *vcomp* provides the i-th component of the vector having the code $\bar{X}$. Analoguously *mcomp* provides the (i, j)-th component of the matrix having the code $\bar{A}$.

Let us consider the following program the input data of which are A, x, n, m.

```
$p_2 \overset{d}{=}$
    0: any $i \leqq m$
    1: every $j \leqq n$
    2: if $vcomp(x, j) \leqq mcomp(A, i, j)$ then 5
    3: $u=0$
    4: goto 6
    5: $u=1$
```

Let $u=1$ be the output condition of the program. It is a useful exercise to prove the fact that the execution of p_2 w.r.t. the input values $\bar{A}, x, n, m$ is correct iff such a row s can be chosen for the label 0 that can be matched to $\bar{X}$.

8.3 A NIM-game

According to the definition the execution of programs is represented by an associated game. Of course this can be reversed as follows: the properties of a game can be represented by an appropriate nondeterministic program.

Let us consider the following version of the game NIM. There are two players A and B and there is a single pile of n chips. The two players take turns alternately and at each move a player must pick up at least one and at most k chips from the pile. This is the game-frame. We add to this the following: the player who picks up the last chip wins. Thus we have the game G.

Let player A move first. By using the calculus of Definition 7.2 and Theorem 7.3 we show that player A has a winning strategy if n is divisible by $k+1$ i.e. if $k+1|n$.

Let us take the same system Ax^* as in 8.1 and consider the following program.

```
$p_3 \overset{d}{=}$
     0: any $y \leqq k$
     1: if $y=0$ then 9
     2: if $y \geqq n$ then 11
     3: $n \leftarrow n-y$
     4: every $y \leqq k$
     5: if $y=0$ then 11
     6: if $y \geqq n$ then 9
     7: $n \leftarrow n-y$
     8: goto 0
     9: $u=0$
    10: goto 12
    11: $u=1$
```

Note that the commands of labels 0 and 4 allow to choose 0 chips in contrast with the rules of the game G. This is connected with our programming language $NP_{\mathfrak{J}}$, where for the sake of simplicity we introduced the assignment commands with choices in the form of i: $\Box y \leqq t$, $\Box \in \{any, every\}$ .However we could have given them in the following form i: $\Box\ \tau_1 \leqq y \leqq \tau_2$ or i: $\Box\ y \in \varphi$ (where $\tau_1, \tau_2 \in T_{\mathfrak{J}}^Y$ and $\varphi \in F_{\mathfrak{J}}^Y$ and it has at least one free variable symbol). Thus to keep up with the game rules we need the commands labelled by 1 and 5. These commands assure that if one of the players on his turn chooses 0 chips that this move will be considered as a cheat and he loses the game immediately.

It is obvious that player A has winning strategy in the game G iff the program p_3 is correct w.r.t. the output condition $\psi \overset{\mathrm{d}}{=} u = 1$. Since in the game G drow is not possible so a non-losing strategy of player A is, at the same time, his winning strategy. Thus it is enough to show that $Ax^* \vdash \chi$ where

$$\chi = [\neg k+1|n, p_3, u = 1],$$

i.e. it is enough to give a description of the formula χ w.r.t. Ax^*. It is easy to verify that the formulas below define a description of χ w.r.t. Ax^*:

$$\Phi_0 \overset{\mathrm{d}}{=} \neg k+1|n$$

$$\Phi_1 \overset{\mathrm{d}}{=} 1 \leqq y \leqq k \wedge y \equiv n \bmod (k+1)$$

$$\Phi_2 \overset{\mathrm{d}}{=} 1 \leqq y \leqq k \wedge y \equiv n \bmod (k+1)$$

$$\Phi_3 \overset{\mathrm{d}}{=} y < n \wedge y \equiv n \bmod (k+1)$$

$$\Phi_4 \overset{\mathrm{d}}{=} k+1|n \wedge n > 0$$

$$\Phi_5 \overset{\mathrm{d}}{=} y \leqq k \wedge n > 0 \wedge k+1|n$$

$$\Phi_6 \overset{\mathrm{d}}{=} 1 \leqq y \leqq k \wedge k+1|n \wedge n > 0$$

$$\Phi_7 \overset{\mathrm{d}}{=} 1 \leqq y \leqq k \wedge k+1|n \wedge n > 0$$

$$\Phi_8 \overset{\mathrm{d}}{=} \neg k+1|n$$

$$\Phi_9 \overset{\mathrm{d}}{=} \mathit{false}$$

$$\Phi_{10} \overset{\mathrm{d}}{=} \mathit{false}$$

$$\Phi_{11} \overset{\mathrm{d}}{=} \mathit{true}$$

$$\Phi_{12} \overset{\mathrm{d}}{=} u = 1.$$

Abstract (to Part 2)

Games are adequate to nondeterministic programming shown in Part 1 and the theory of nondeterministic programming is intended to be developed within the frame of classical first order logic. So in Section 4 the representation of games is given by considering definable games within this frame. The nondeterministic programming language is introduced in Section 5. In Section 6 a descriptive language is introduced which, beside the classical data consideration, handles time conditions as well. It is shown (in Theorem 6.7) that this language is complete. Section 7 contains a calculus in the spirit of Floyd and Hoare, the usage of which is illustrated in the last section by examples.

RESEARCH INSTITUTE FOR
APPLIED COMPUTER SCIENCE
CSALOGÁNY U. 30—32.
BUDAPEST, HUNGARY
H—1536

References

[1] BANACHOWSKI, L., A. KRECZMAR, G. MIRKOWSKA, H. RASIOWA and A. SAŁWICKI, An introduction to algorithmic logic, *Mathematical Foundations of Computer Science,* Banach Center Publications, Warszawa, v. 2, 1977, pp. 7—99.

[2] COOK, S. A., Soundness and completeness of an axiom system for program verification, *SIAM J. Comput.* v. 7, 1978, pp. 73—92.

[3] GERGELY, T. and L. ÚRY, Nondeterministic programming within the frame of first order classical logic, Part 1, *Acta Cybernet.*, v. 4, 1980, pp. 333—354.

[4] HAREL, D., Arithmetical completeness in logics of programs, *Automata, Languages and Programming,* Eds, G. Ausiello and C. Böhm, Springer-Verlag, Heidelberg, 1978.

[5] MENDELSON, E., *Introduction to mathematical logic,* Van Nostrand, N. Y. 1964.

(Received Sept. 7, 1979)

Completeness in non-simple and stable modal logics

By K. Tóth

In my work [1] I have defined the syntax and semantics of modal logics. Also, inference systems and completeness theorems for simple, non-stable logics have been included. Unfortunately, the methods used there cannot apply directly to non-simple and stable logics. In this paper I give a modification of the method and prove completeness theorems for the cases not covered in [1]. In fact, this paper is a continuation to [1], all non-common notions and notations are introduced there.

§ 1. Completeness in non-simple logics

The notion of consistency is defined in [1].

Definition. The set of formulae is complete if the following conditions are satisfied:

(i) α is consistent;

(ii) If $\mathcal{A}$ contains variables only from $\pi(\alpha)$, then either $A \in \alpha$ or $\sim\mathcal{A} \in \alpha$;

(iii) Let $\mathcal{A}$ contain variables only from $\pi(\alpha)$. If $\exists x \mathcal{A} \in \alpha$, then there exists a variable $a \in \pi(\alpha)$ such that a is free for x and $\mathcal{A}[x/a] \in \alpha$;

(iv) Let f be n-argument function symbol and let $x_1, \dots, x_n \in \pi(\alpha)$. There exists a variable $a \in \pi(\alpha)$ such that for all classical formula $\mathcal{A}$ the fact $f(x_1, \dots, x_n)$ is free for a in $\mathcal{A}$ implies that the two assertions $\mathcal{A} \in \alpha$ and $\mathcal{A}[a/f(x_1, \dots, x_n)] \in \alpha$ are equivalent.

Theorem 1. If α is consistent, then there exists a complete set β such that $\alpha \subseteq \beta$.

Proof. Parallel to the proof of Theorem 5 in [1] using the following Lemma.

Lemma. Let f be an n-argument function symbol, α a consistent set and $a \notin \pi(\alpha)$. Moreover, let $\alpha' = \alpha \cup \{\mathcal{A} : \mathcal{A}$ is a classical formula, $f(x_1, \dots, x_n)$ is free for a in $\mathcal{A}$ and $\mathcal{A}[a/f(x_1, \dots, x_n)] \in \alpha\}$. Then α' is consistent.

Proof. In contrary, let us suppose that there exist the formulae $\mathcal{A}_1, \dots, \mathcal{A}_k$, $\mathcal{B}_1, \dots, \mathcal{B}_l$ such that $\mathcal{A}_1, \dots, \mathcal{A}_k \in \alpha$, $\mathcal{B}_1[a/f(x_1, \dots, x_n)], \dots, \mathcal{B}_l[a/f(x_1, \dots, x_n)] \in \alpha$

and $\vdash \sim(\mathscr{A}_1 \wedge \ldots \wedge \mathscr{A}_k \wedge \mathscr{B}_1 \wedge \ldots \wedge \mathscr{B}_l)$. Applying R2 and A6.b we have

$$\vdash \forall a \sim (\mathscr{A}_1 \wedge \ldots \wedge \mathscr{A}_k \wedge \mathscr{B}_1 \wedge \ldots \wedge \mathscr{B}_l)$$

$$\vdash \sim(\mathscr{A}_1 \wedge \ldots \wedge \mathscr{A}_k \wedge \mathscr{B}_1[a/f(x_1, \ldots, x_n)] \wedge \ldots \wedge \mathscr{B}_l[a/f(x_1, \ldots, x_n)])$$

which is a contradiction.

Definition of a complete system of formula-sets is just the same as in [1]-however, item (iii) can be omitted by the remark above.

The theorem remains valid for the new concept:

Theorem 2. If α is a complete set of formulae, then there exists a complete system of sets M such that $\alpha \in M$.

The completeness result follows easily from this theorem.

Theorem 3. Let a non-simple, non-stable modal logic be given. If $\mathscr{A}$ cannot be derived in this logic, then $\sim\mathscr{A}$ is satisfiable.

Proof. By the previous theorems, there exist a complete set α and a complete system of sets M such that $\sim\mathscr{A} \in \alpha$, $\alpha \in M$. We assume, by the definition of a complete set, that is a function for which the following property holds: if $\alpha \in M$, f is n-argument function symbol, $x_1, \ldots, x_n \in \pi(\alpha)$, then $v(\alpha, f(x_1, \ldots, x_n))$ is a variable, such that for all classical formula $\mathscr{B}$, if $f(x_1, \ldots, x_n)$ is free for $v(\alpha, f(x_1, \ldots, x_n))$ in $\mathscr{B}$, then the two assertions $\mathscr{B} \in \alpha$ and $\mathscr{B}[v(\alpha, f(x_1, \ldots, x_n))/f(x_1, \ldots, x_n)] \in \alpha$ are equivalent.

Let us introduce the notations:

$$N = \{\beta\colon\ \beta \in M \text{ and } \beta^+ \neq \emptyset\};$$

$$\text{If } \beta, \gamma \in M, \quad \text{then} \quad \beta R \gamma \Leftrightarrow ((\beta^+ \subseteqq \gamma \text{ and } \beta^+ \neq \emptyset) \quad \text{or} \quad (\beta^+ = \emptyset \text{ and } \gamma = \beta));$$

$$|P(\beta)| = \pi(\beta);$$

$$f_{P(\beta)}(x_1, \ldots, x_n) = v(\beta, f(x_1, \ldots, x_n)), \quad \text{where} \quad x_1, \ldots, x_n \in \pi(\beta);$$

$$r_{P(\beta)}(x_1, \ldots, x_n) \Leftrightarrow r(x_1, \ldots, x_n) \in \beta, \quad \text{where} \quad x_1, \ldots, x_n \in \pi(\beta).$$

It is clear that $\langle M, N, \alpha, R, P\rangle$ is a model. Let us extend the domain of v as follows: let $v(\beta, x) = x$, where $x \in \pi(\beta)$; and let $v(\beta, f(\tau_1, \ldots, \tau_n)) = v(\beta, f(v(\beta, \tau_1), \ldots, \ldots, v(\beta, \tau_n)))$, where $\tau_1, \ldots, \tau_n$ are terms containing variables from $\pi(\beta)$ exclusively. The following assertions can be proved by (the usual) induction:

Let k be an interpretation and $\varkappa$ the corresponding valuation.

(i) If $\tau \in \mathscr{H}_k(\beta)$, then $\varkappa(\tau, \beta) = v(\beta, \tau[x_1, \ldots, x_m/k(x_1), \ldots, k(x_m)])$, where $x_1, \ldots, x_m$ are all variables occuring in τ.

(ii) If $\mathscr{B} \in \mathscr{H}_k(\beta)$, then $\beta \models \mathscr{B}[k] \Leftrightarrow \mathscr{B}[x_1, \ldots, x_m/k(x_1), \ldots, k(x_m)] \in \beta$; where $x_1, \ldots, x_m$ are all variables occuring in $\mathscr{B}$.

In particular, it follows that $\sim\mathscr{A}$ is valid in the model $\langle M, N, \alpha, R, P\rangle$.

Properties K1—K3 can be proved just as in [1, Theorem 7].

§ 2. Completeness in stable logics

Theorem 4. If α is complete, then there exist a complete set β and a complete system of sets M such that $\alpha\subseteq\beta$, $\beta\in M$ and for every $\gamma\in M$, $\pi(\beta)=\pi(\gamma)$.

Before proving this theorem we give the completeness result for stable logics.

Theorem 5. Let a stable modal logic be given. If the formula $\mathscr{A}$ cannot be derived in this logic, then $\sim\mathscr{A}$ is satisfiable.

Proof. Very similar to the proof of Theorem 3 above or Theorem 7 of [1] provided complete system of sets M, given by Theorem 4, is used in the construction.

§ 3. Proof of Theorem 4

We introduce the following notations: let α be a set of formulae. By $\psi(\alpha)$ we shall mean the set of all formulae which contain variables only from $\pi(\alpha)$.

Let R be a two-argument relation. We define the relation R^n, n finite, by the following recurrence: R^0 is the identity relation and let R^{n+1} be defined by $AR^{n+1}B$ if there exists C such that AR^nC and CRB.

Then, $\bar{R}=\bigcup\limits_{n=0}^{\infty}R^n$, where $\bar{R}$ is the reflexive, transitive closure of R.

In the following we shall deal with certain ordered triplets $\langle\alpha, M, R\rangle$. Without further mentioning we always suppose that the following conditions hold for $\langle\alpha, M, R\rangle$:

(i) M is a set of complete sets, $\alpha\in M$, R is a binary relation on M.

(ii) For every $\beta\in M$, $\alpha\bar{R}\beta$ and if $S\subseteq R$ and for all $\beta\in M$ $\alpha\bar{S}\beta$, then $R=S$.

(iii) If $a\in\pi(\beta)$, then there exists γ such that $a\in\pi(\delta)$ if and only if $\gamma\bar{R}\delta$.

(iv) a) If $\beta R\gamma$ then $\beta^+\cap\psi(\gamma)\subseteq\gamma$ and $\beta^+\neq\emptyset$.

b) Let $\beta\in M$, $\langle\rangle\mathscr{A}\in\beta$. If there is a $\gamma\in M$, such that $\beta R\gamma$ and $\mathscr{A}\in\psi(\gamma)$, then there also is a $\gamma\in M$ with $\beta R\gamma$ and $\mathscr{A}\in\gamma$.

Assertion 1. For arbitrary triplet $\langle\alpha, M, R\rangle$ there is no $\beta\in M$ such that $\beta R\alpha$. If $\beta R\delta$ and $\gamma R\delta$ then $\beta=\gamma$.

Proof. (a) Consider the triplet $\langle\alpha, M, S\rangle$, where S is defined by $\beta S\gamma$ if and only if $\beta R\gamma$ and $\gamma\neq\alpha$. By the second clause (ii) above, $R=S$.

(b) Let us suppose that $\beta R\delta$, $\gamma R\delta$ and $\beta\neq\gamma$. Let S be defined as $\beta_1S\beta_2$ if and only if $\beta_1R\beta_2$ and $(\beta_1,\beta_2)\neq(\beta,\delta)$. Then conditions above will hold for $\langle\alpha, M, S\rangle$, but $S\subseteq R$ and $S\neq R$ which contradicts the second condition (ii).

Definition. $\langle\alpha, M, R\rangle$ is called n-th order triplet if for every $\beta\in M$ there is a k $(0\leqq k\leqq n)$ such that $\alpha R^k\beta$. $\langle\alpha, M, R\rangle$ is totally n-th order triplet if it is an n-th order triplet and if $0\leqq k<n$, $\alpha R^k\beta$, $\langle\rangle\mathscr{A}\in\beta$, $\beta^+\neq\emptyset$ then there exists $\gamma\in M$ for which $\beta R\gamma$ and $\mathscr{A}\in\gamma$.

It is clear, that for every m $(m\leqq n)$ the fact $\langle\alpha, M, R\rangle$ is an n-th order triplet implies that $\langle\alpha, M, R\rangle$ is an m-th order triplet too. Similarly, if $\langle\alpha, M, R\rangle$ is a zero order triplet, then $M=\{\alpha\}$, $R=\emptyset$, thus $\langle\alpha, M, R\rangle$ is totally zero order.

Let $\langle\alpha, M, R\rangle$ be arbitrary, $\beta\in M$. Let us set

$$M/\beta=\{\gamma\colon\ \beta\bar{R}\gamma\},\quad R/\beta=R\cap(M/\beta\times M/\beta).$$

Assertion 2. If $\langle\alpha, M, R\rangle$ is an n-th order triplet and $\alpha R\beta$ then $\langle\beta, M/\beta, R/\beta\rangle$ is an $(n-1)$-th order triplet.

DEFINITION. Let us define the operation L by the following items: if $\langle\alpha, M, R\rangle$ is a 0-order triplet, then $L\langle\alpha, M, R\rangle=\alpha$, if $n>0$ and $\langle\alpha, M, R\rangle$ is an n-th order triplet, then let $L\langle\alpha, M, R\rangle\doteq\alpha\cup\bigcup\limits_{\alpha R\beta}\{\Diamond\mathscr{A}$: $\mathscr{A}$ is a conjunction of formulae from the set $L\langle\beta, M/\beta, R/\beta\rangle\}$.

Theorem 6. Let $\langle\alpha, M, R\rangle$ be an n-th order triplet. Then $L\langle\alpha, M, R\rangle$ is consistent.

Proof. We proceed by induction on n. If $n=0$ then the assertion clearly holds. Let $n>0$, and assume the contrary, i.e. there exist $\mathscr{A}_1, \mathscr{A}_2, \ldots$ and a conjunction $\mathscr{B}_1$ of formulae from $L\langle\beta_1, M/\beta_1, R/\beta_1\rangle$, a conjunction $\mathscr{B}_2$ of formulae from $L\langle\beta_2, M/\beta_2, R/\beta_2\rangle$ etc., such that

$$\vdash\sim(\mathscr{A}_1\wedge\mathscr{A}_2\wedge\ldots\wedge\Diamond\mathscr{B}_1\wedge\Diamond\mathscr{B}_2\wedge\ldots)$$

that is

$$\vdash\sim\mathscr{A}_1\vee\sim\mathscr{A}_2\vee\ldots\vee\Box\sim\mathscr{B}_1\vee\Box\sim\mathscr{B}_2\vee\ldots.$$

We can assume that all β_i, β_j are distinct, for if not, then can apply

$$\vdash\Box\sim\mathscr{B}_i\vee\Box\sim\mathscr{B}_j\rightarrow\Box\sim(\mathscr{B}_i\wedge\mathscr{B}_j).$$

Hence we obtain a form in which all sets β_i, β_j are distinct. Let x be a variable of $\mathscr{B}_1$ such that $x\notin\pi(\alpha)$. It follows from Assertion 1 and condition (iii) that x does not occur in the formulae $\mathscr{A}_1, \mathscr{A}_2, \ldots$ or $\mathscr{B}_2, \ldots$.

Apply rule R2 for all variables not occuring in $\pi(\alpha)$:

$$\vdash\sim\mathscr{A}_1\vee\sim\mathscr{A}_2\vee\ldots\vee\forall x_{11}\,\forall x_{12}\ldots\Box\sim\mathscr{B}_1\vee\forall x_{21}\,\forall x_{22}\ldots\Box\sim\mathscr{B}_2\vee\ldots.$$

Since the fixed logic is stable we can repeatedly apply the axiom $\forall x\Box\mathscr{A}\rightarrow\Box\forall x\mathscr{A}$ and obtain

$$\vdash\sim\mathscr{A}_1\vee\sim\mathscr{A}_2\vee\ldots\vee\Box\forall x_{11}\,\forall x_{12}\ldots\sim\mathscr{B}_1\vee\Box\forall x_{21}\,\forall x_{22}\ldots\sim\mathscr{B}_2\vee\ldots$$

where all free variables are from $\pi(\alpha)$. Since bound variables can be substitued by suitable ones from $\pi(\alpha)$ we have

$$\vdash\sim\mathscr{A}_1\vee\sim\mathscr{A}_2\vee\ldots\vee\Box\forall x'_{11}\,\forall x'_{12}\ldots\sim\mathscr{B}'_1\vee\Box\forall x'_{21}\,\forall x'_{22}\ldots\sim\mathscr{B}'_2\vee\ldots$$

α complete, so this possible only when some disjunctive terms, e.g. $\Box\forall x'_{11}\,\forall x'_{12}\ldots$ $\ldots\sim\mathscr{B}'_1\in\alpha$. (For if $\sim\mathscr{A}_1\in\alpha$, then $\mathscr{A}_1\in\alpha$ which contradicts the completeness of α.) So

$$\forall x'_{11}\,\forall x'_{12}\ldots\sim\mathscr{B}'_1\in\beta_1\subseteqq L\langle\beta_1, M/\beta_1, R/\beta_1\rangle$$

and

$$\vdash\sim(\forall x'_{11}\,\forall x'_{12}\ldots\sim\mathscr{B}'_1\wedge\mathscr{B}_1).$$

We concluded that $L\langle\beta_1, M/\beta_1, R/\beta_1\rangle$ is consistent, wich proves the theorem.

Theorem 7. If $\langle\alpha, M, R\rangle$ is an n-th order triplet, $\alpha R\gamma$ and β is a complete set such that $L\langle\alpha, M, R\rangle\subseteqq\beta$ then $\beta^+\cup L\langle\gamma, M/\gamma, R/\gamma\rangle$ is consistent.

Proof. In contrary, let us assume that there are formulae $\mathscr{B}\in\beta^+$ and $\mathscr{C}$, a conjunction of elements from $L\langle\gamma, M/\gamma, R/\gamma\rangle$ such that $\vdash\sim(\mathscr{B}\wedge\mathscr{C})$ i.e. $\vdash\mathscr{B}\to\sim\mathscr{C}$. By R3, we obtain $\vdash\Box\mathscr{B}\to\Box\sim\mathscr{C}$ i.e. $\vdash\sim(\Box\mathscr{B}\wedge\langle\rangle\mathscr{C})$. In accordance with our conditions, $\Box\mathscr{B}\in\beta$ and so $\langle\rangle\mathscr{C}\in\beta$ which contradicts the completeness of β.

Definition. We say that $\langle\alpha, M, R\rangle$ is a continuation of $\langle\beta, N, S\rangle$ if there exists a function $f\colon N\to M$, such that $f(\beta)=\alpha$, if $\gamma\in N$ then $\gamma\subseteqq f(\gamma)$, and if $\gamma S\delta$ then $f(\gamma)Rf(\delta)$.

Theorem 8. If $\langle\alpha, M, R\rangle$ is an n-th order triplet and β is a complete set for which $L\langle\alpha, M, R\rangle\subseteqq\beta$, then there exists a totally n-th order triplet $\langle\beta, N, S\rangle$ which is a continuation of $\langle\alpha, M, R\rangle$.

Proof. We proceed by induction on n. If $n=0$, the assertion follows. Let $n>0$. If $\alpha^+=\emptyset$, then for all $\mathscr{A}\in\psi(\alpha)$, $\langle\rangle\mathscr{A}\in\alpha$, so $\langle\rangle\mathscr{A}\in\beta$. In particular if $\mathscr{A}$ is a negation of a tautology, then $\vdash\langle\rangle\mathscr{A}\to\langle\rangle\mathscr{B}$, thus for every $\mathscr{B}\in\psi(\beta)$, $\langle\rangle\mathscr{B}\in\beta$, i.e. $\beta^+=\emptyset$. It is impossible that $\beta S\gamma$, by definition, hence $\langle\beta, \{\beta\}, \emptyset\rangle$ is a totally n-th order triplet and this is a continuation of $\langle\alpha, M, R\rangle=\langle\alpha, \{\alpha\}, \emptyset\rangle$.

Let $\alpha^+\neq\emptyset$, and so $\beta^+\neq\emptyset$. As we see $\langle\rangle\mathscr{A}\in\beta$ implies the consistency of $\beta^+\cup\{\mathscr{A}\}$, hence we can assume that $\beta^+\cup\{\mathscr{A}\}\subseteqq\delta_{\mathscr{A}}$, $\delta_{\mathscr{A}}$ is complete. By the previous theorem, $\beta^+\cup L\langle\gamma, M/\gamma, R/\gamma\rangle$ is consistent, too, provided $\alpha R\gamma$, thus there is a complete set δ_γ such that $\beta^+\cup L\langle\gamma, M/\gamma, R/\gamma\rangle\subseteqq\delta_\gamma$.

It is clear, that the new variables, introduced in these steps, may be chosen so that the sets $\pi(\delta_{\mathscr{A}})\setminus\pi(\beta), \ldots, \pi(\delta_\gamma)\setminus\pi(\beta), \ldots$ are pairwise disjoint. As $\langle\delta_{\mathscr{A}}, \{\delta_{\mathscr{A}}\}, \emptyset\rangle$ is an $(n-1)$-th order triplet, and since $L\langle\delta_{\mathscr{A}}, \{\delta_{\mathscr{A}}\}, \emptyset\rangle\subseteqq\delta_{\mathscr{A}}$ it follows that a totally $(n-1)$-th order continuation $\langle\delta_{\mathscr{A}}, M_{\mathscr{A}}, R_{\mathscr{A}}\rangle$ of $\langle\delta_{\mathscr{A}}, \{\delta_{\mathscr{A}}\}, \emptyset\rangle$ exists. Since $L\langle\gamma, M/\gamma, R/\gamma\rangle\subseteqq\delta_\gamma$ by the induction hypothesis, it follows that there exist M_γ, R_γ such that $\langle\delta_\gamma, M_\gamma, R_\gamma\rangle$ is a totally $(n-1)$-th order triplet and it is a continuation of $\langle\gamma, M/\gamma, R/\gamma\rangle$.

We may assume that the common variables of any two sets $\pi(\bigcup M_{\mathscr{A}}), \ldots, \ldots, \pi(\bigcup M_\gamma), \ldots$ are contanied in $\pi(\beta)$.

Let $\delta\in N$, provided $\delta=\beta$, or
  if there is an $\mathscr{A}$, such that $\langle\rangle\mathscr{A}\in\beta$ and $\delta\in M_{\mathscr{A}}$, or
  if there is a γ, such that $\alpha R\gamma$ and $\delta\in M_\gamma$.
Let $\delta_1, \delta_2\in N$ and $\delta_1 S\delta_2$ provided
  if $\delta_1=\beta$ and there is an $\mathscr{A}$ such that $\langle\rangle\mathscr{A}\in\beta$ and $\delta_2=\delta_{\mathscr{A}}$, or
  if $\delta_1=\beta$ and there is an γ for which $\alpha R\gamma$ and $\delta_2=\delta_\gamma$, or
  if there is an $\mathscr{A}$, such that $\langle\rangle\mathscr{A}\in\beta$ and $\delta_1 R_{\mathscr{A}}\delta_2$, or
  if there is an γ, such that $\alpha R\gamma$ and $\delta_1 R_\gamma\delta_2$.
It is obvious, that the conditions (i)—(iv) hold for $\langle\beta, N, S\rangle$. Also, it is a totally n-th order triplet and is a continuation of $\langle\alpha, M, R\rangle$.

Now we can return to the proof of Theorem 4: Let $\langle\alpha_0, M_0, R_0\rangle=\langle\alpha, \{\alpha\}, \emptyset\rangle$ i.e. a totally 0-order triplet. Let us suppose, that for some n, a totally n-th order triplet $\langle\alpha_n, M_n, R_n\rangle$ is defined. By Theorem 6, $L\langle\alpha_n, M_n, R_n\rangle$ is consistent, and hence there is a complete set α_{n+1}, such that $L\langle\alpha_n, M_n, R_n\rangle\subseteqq\alpha_{n+1}$. By Theorem 8, there exist M_{n+1}, R_{n+1} such that $\langle\alpha_{n+1}, M_{n+1}, R_{n+1}\rangle$ is a totally $(n+1)$-th order triplet and it is a continuation of $\langle\alpha_n, M_n, R_n\rangle$. Thus, there exists a function $f_n\colon M_n\to M_{n+1}$ such that $f_n(\alpha_n)=\alpha_{n+1}$ and $\beta\in M_n$ implies $\beta\subseteqq f_n(\beta)$ and if $\beta R_n\gamma$, then $f_n(\beta)R_{n+1}f_n(\gamma)$.

Let $\beta=\bigcup\limits_{n=0}^{\infty}\alpha_n$ and $M=\left\{\bigcup\limits_{n=k}^{\infty}\gamma_n\colon \gamma_k\in M_k \text{ and for all } i\geqq k,\ \gamma_{i+1}=f_i(\gamma_i)\right\}$. Since union of increasing complete sets is also complete we have that every element of M is complete.

Let $\gamma\in M$, $\gamma^+\neq\emptyset$, and $\langle\rangle\mathscr{A}\in\gamma$. For $\gamma=\bigcup\limits_{n=k}^{\infty}\gamma_n$, there exists an l, such that $\langle\rangle\mathscr{A}\in\gamma_l$ and hence there also exists δ_l for which $\gamma_l R_l \delta_l$ and $\mathscr{A}\in\delta_l$. Let $\delta=\bigcup\limits_{n=l}^{\infty}\delta_n$. $\gamma^+=\bigcup\limits_{n=l}^{\infty}\gamma_n^+\subseteqq\bigcup\limits_{n=l}^{\infty}\delta_n=\delta$ and $\mathscr{A}\in\delta$, thus M is a complete system of sets.

Let $a\in\pi(\beta)$. For some k, $a\in\pi(\alpha_k)$, and if $l>k$, then $a\in\pi(\alpha_k)$, too. If $\delta\in M$, then for some i, $\delta=\bigcup\limits_{n=i}^{\infty}\delta_n$. We may assume that $k<i$ and so $a\in\pi(\alpha_i)$. Since $\langle\alpha_i, M_i, R_i\rangle$ is a totally i-th order triplet, we have $\pi(\alpha_i)\subseteqq\pi(\delta_i)$, and thus $a\in\pi(\delta)$.

Let $a\in\pi(\delta)$ for some $\delta\in M$. We may assume that $\delta=\bigcup\limits_{n=k}^{\infty}\delta_k$ and $a\in\pi(\delta_k)$. For $L\langle\alpha_k, M_k, R_k\rangle\subseteqq\alpha_{k+1}$, $a\in\pi(\alpha_{k+1})$ and hence $a\in\pi(\beta)$.

We gained, that for every $\delta\in M$, $\pi(\beta)=\pi(\delta)$ which completes the proof of Theorem 4 and also the completeness theorem.

References

[1] Tóth, K., Modal logics with function symbols, *Acta Cybernet.*, v. 4, 1979, pp. 291—302.

(Received July 13, 1979)

Enumeration of certain words *

By K. H. Kim and F. W. Roush

1. Introduction

Both the content and methods of this paper are closely related to those of [2]. Let F denote a free semigroup on generators $x_1, x_2, \ldots, x_m$. We wish to enumerate, for fixed n and k, the number of length n words which do not have any segment which is the square of a length k word.

One reason for considering this problem is that it is related to the more difficult problem of enumerating words not containing any segment which is a square word. Goulden and Jackson [1, Corollary 4.1] have obtained our Theorem 1 by completely different methods.

2. A recursion formula

Definition 1. Let F^1 be a *free monoid.* Let $w_1, w_2 \in F^1$.
(1) $w_1 | w_2$ if $x w_1 y = w_2$ for some $x, y \in F^1$.
(2) $w_1 |_i w_2$ if $w_2 = w_1 y$ for some $y \in F^1$.
(3) $w_1 |_f w_2$ if $w_2 = x w_1$ for some $x \in F^1$.
(4) $|w_1|$ denotes the length of w_1; $|1|=0$.
(5) If $w_1 \neq 1, \hat{w}_1$ is the unique word such that

$$\hat{w}_1 |_i w_1, \; |\hat{w}_1| = |w_1| - 1.$$

Definition 2. Let F be a *free semigroup* on $m>1$ generators and F^1 the associated free monoid. Fix $k>0$.

To any word w in F^1 we assign a length $k-1$ (0, 1)-vector v as follows. The number v_j is 1 if and only if the $n-k+j+1$ letter of w equals the $n-2k+j+1$ letter of w, and $n-2k+j+1>0$. We assign an integer $a(w)$ to w by stating that $a(w)$ is the length of the longest terminal sequence of 1 components of v. Let $S(n, c)$ for $0 \leqq c \leqq k-1$ denote the set of length n words w_1 of F^1 such that $a(w_1)=c$ and if $|w_2|=k$ it is false that $(w_2)^2 | w_1$.

* This work was supported by Alabama State University Faculty Research Grant R—78—6.

In other terms, given a word w_1 consider the terminal segment w_3 of length $2k-1$ of w_1. Then $a(w_1)$ is the length of the longest initial segment of w_3 which equals a final segment of w_3, or is $k-1$ if this length exceeds $k-1$. The set $S(n, c)$ is the set of words w of length n such that $a(w)=c$ and w is not divisible by the square of a word of length k.

Theorem 1. *For $n>k$, $0<c<k$,*

$$|S(n, c)| = |S(n-1, c-1)|,$$

$$|S(n, 0)| = \sum_{j=0}^{k-1} (m-1)|S(n-1, j)|.$$

Furthermore, $|S(n, 0)|=m^n$, $|S(n, c)|=0$ *for* $n\leqq k$, $0<c<k$.

Proof. Suppose $x\in S(n, c)$. Then $\hat{x}\in S(n, c-1)$. (Note that if $c=k-1$, the $n-k$ and $n-2k$ letters of x must differ, else x would be divisible by the square of a length k word.) And if $y\in S(n, c-1)$ there is one and only one $x\in S(n, c)$ such that $\hat{x}=y$. Namely the last letter of x must equal the $n-k$ letter of y. Thus $|S(n, c)|=|S(n, c-1)|$. Also the function $x\to\hat{x}$ maps $S(n, 0)$ into $\bigcup_{j=0}^{k-1} S(n-1, j)$. Each element y of the latter set arises from exactly $(m-1)$ elements of $S(n, 0)$. Namely we can add to y any letter except the $(n-k)^{\text{th}}$. Therefore

$$|S(n, 0)| = (m-1) \sum_{j=0}^{k-1} |S(n-1, j)|.$$

This proves the theorem.

This formula can be recast into a matrix form. Let M be the $k\times k$ matrix

$$\begin{bmatrix} m-1 & 1 & 0\dots 0 \\ m-1 & 0 & 1\dots 0 \\ m-1 & 0 & 0\dots 0 \\ & \dots & \\ m-1 & 0 & 0\dots 0 \end{bmatrix}.$$

Let u be the vector $(m^k, 0, 0, \dots, 0)$. Then $|S(n, c)|$ is the $c+1$ component of uM^{n-k}, for $n\geqq k$. The characteristic polynomial of M is

$$P(z) = z^k-(m-1)(z^{k-1}+z^{k-2}+\dots+z+1).$$

Definition 3. For $n<k$ put $f(n)=0$. For $n\geqq k$, let $f(n)$ be

$$\sum_{j=0}^{k-1} |S(n, j)|.$$

Theorem 2. *The generating function of $f(n)$ is*

$$\frac{m^k z^k(1-z^k)}{1-mz+(m-1)z^{k+1}}.$$

Proof. As in [2], the generating function of $f(n)$ must have the form

$$\frac{q(z)z^k}{z^k p\left(\frac{1}{z}\right)}$$

where $q(z)$ is some polynomial of degree at most $k-1$. In degree less than $2k$, this quotient must be

$$z^k m^k(1+mz+\ldots+m^{k-1}z^{k-1}).$$

Therefore $q(z)$ is the portion of

$$m^k(1-(m-1)z-\ldots-(m-1)z^k)(1+mz+\ldots+m^{k-1}z^{k-1})$$

having degrees no more than k. A computation gives the formula above (note that $(1-z)$ can be cancelled from numerator and denominator).

3. Asymptotic values of $f(n)$

Lemma 3. *The equation $P(z)=0$ has a unique positive real root r_k of multiplicity* 1. *This root exceeds the absolute value of any other root. We have $r_k>r_{k-1}$ and*

$$\lim_{k\to\infty} r_k = m.$$

Moreover

$$m-\frac{m-1}{m^k} > r_k > m-\frac{m-1}{r_j^k}$$

for $j<k$.

Proof. If $P(z)=0$ then $u=\frac{1}{z}$ satisfies

$$u^{k+1}=\frac{mu-1}{m-1}$$

as does $u=1$. However no straight line can cut the curve $y=x^{k+1}$, $x>0$ in more than two places since $y=x^{k+1}$ is concave upwards. Therefore 1 and $\frac{1}{r_k}$ are the only positive solutions of

$$u^{k+1}=\frac{mu-1}{m-1}.$$

Therefore $P(z)=0$ has only one positive real solution r_k and $r_k>1$. Differentiation shows that r_k has multiplicity 1. Let z be a root of $P(z)=0$ which is negative or complex. Then

$$z^k=(m-1)(z^{k-1}+z^{k-2}+\ldots+1)$$

implies

$$|z|^k < (m-1)(|z|^{k-1}+|z|^{k-2}+\dots+1)\ .$$

So $|z|<r_k$ because for $|z|\geqq r_k$ we have $P(|z|)>0$. We have

$$r_k^{k-1} = (m-1)\left(r_k^{k-2}+\dots+1+\frac{1}{r_k}\right) > (r_k^{k-2}+\dots+1)(m-1)$$

and

$$r_{k-1}^{k-1} = (m-1)(r_{k-1}^{k-2}+\dots+1).$$

This implies $r_k>r_{k-1}$. It also implies that

$$r = \lim_{k\to\infty} r_k$$

satisfies

$$r = (m-1)\frac{1}{1-\frac{1}{r}}.$$

Therefore $r=m$.

From

$$z^k = (m-1)\frac{z^k-1}{z-1}$$

at $z=r_k$ we have

$$z = 1+(m-1)\left(1-\frac{1}{z^k}\right).$$

This implies the last inequality of the lemma. This proves the lemma.

Theorem 4. *The asymptotic value of $f(n)$ is*

$$\frac{m^k(1-u^k)u^{k-n-1}}{m-(m-1)(k+1)u^k}$$

where $u=\frac{1}{r_k}$.

Proof. Expand the generating function in partial fractions. All other terms will be insignificant compared with the term

$$\frac{A}{1-r_k z}.$$

This term can computed by letting z approach $\frac{1}{r_k}$ in the generating function. This proves the theorem..

Abstract

We study the number of words of length n, in m generators, divisible by the square of a length k word. We find a recursion formula, the generating function, and the asymptotic value of this number.

MATHEMATICS RESEARCH GROUP
ALABAMA STATE UNIVERSITY
MONTGOMERY, ALABAMA 36101
U.S.A.

Reference

[1] GOULDEN, I. M. and D. M. JACKSON, An inversion theorem for cluster decompositions of sequences with distinguished subsequences, Univ. of Waterloo, Dept. of Combinatorics and Optimization Research Report CORR 78/24, August 1978.

[2] KIM, K. H., M. S. PUTCHA and F. W. ROUSH, Some combinatorial properties of free semigroups, *J. London Math. Soc.* (2), v. 16, 1977, pp. 397—402.

(Received Nov. 9, 1978)

Groupoids of pseudoautomata

By F. Ferenci

Introduction

It is known [7] that to a unary universal algebra (universal algebra [6] with unary operations only) there corresponds a monoid (semigroup with identity). An automaton without outputs [4], or shortly automaton, can be obtained from a unary universal algebra by selecting an element and a subset from its base set, for the initial state and the final state set of automaton, respectively [8]. The above mentioned monoid is associated with this automaton, as well [5].

The concept of a tree automaton [1] is such a generalization of that of an automaton, when the corresponding universal algebra is not necessarily unary [12], [11]. The purpose of this paper is to show that there is an other way of generalization obtained by replacing the monoid by an arbitrary groupoid. Then the notions corresponding to those of the unary universal algebra and the automaton are the pseudoalgebra and the pseudoautomaton (which is a kind of tree automata, as well), respectively (see Conclusion at the end of the paper). These notions are introduced in the paper [10].

The method used here for representation of formal languages [9] by a set of trees is analogous to that in the author's paper [2].

1. Trees and algebraic structures

By ω we denote the set of all nonnegative integers, i.e., $\omega=\{0, 1, 2, \dots\}$, and by π the set of two parentheses (and), i.e., $\pi=\{(,)\}$. Furthermore, we suppose that π is disjoint from each other sets used here. For a set A, A^* is the set of all finite strings on A including the empty string λ and $A^+=A^*-\{\lambda\}$. If p is a finite string on A, then $\lg(p)$ denotes the length of p, i.e., the number of occurences of symbols from A in p. An *alphabet* is allways a finite nonempty set.

Let V be an alphabet and X a set of symbols disjoint from V (X may be infinite, finite, nonempty or empty). *The set of trees of type V over the set X,* in notation $[V, X]$, is a subset of $(V\cup X\cup\pi)^+$ defined as follows:

1.1. (1) $x \in [V, X]$ for all $x \in X$;

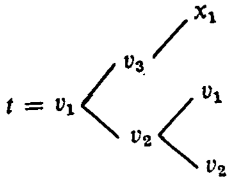
(2) if $v \in V$, $k \in \omega$ and $t_i \in [V, X]$ for $1 \leqq i \leqq k$; then $v(t_1)(t_2)\ldots(t_k) \in [V, X]$ (in the special case $k=0$, $v \in [V, X]$);

(3) the elements of $[V, X]$ are those and only those which we get from (1) and (2) in a finite number of steps.

It can be proved that for every $t=v(t_1)(t_2)\ldots(t_k) \in [V, X]$, the components $v \in V$, $k \in \omega$ and $t_i \in [V, X]$ $(1 \leqq i \leqq k)$ are uniquely determined.

If $X=\emptyset$ then we write $[V]$ for $[V, X]$, i.e., $[V]=[V, \emptyset]$.

Example 1.1. If $V=\{v_1, v_2, v_3, v_4\}$ and $X=\{x_1, x_2\}$, then $t=v_1(v_3(x_1))(v_2(v_1)(v_2))$ is an element of $[V, X]$. This tree is represented graphically in the next figure:



(It can be noticed from our definition of trees, that the use of parentheses differ from usually manner, see [10], [12], [1], etc. Our method is taken from [3].)

The *word function* W: $[V] \to V^+$ we define in the next way:

1.2. (1) if $t=v$ for some $v \in V$, then $W(t)=v$;

(2) if $t=v(t_1)(t_2)\ldots(t_k)$ where $v \in V$, $k \geqq 1$, $t_i \in [V]$, $1 \leqq i \leqq k$, then $W(t)= =vW(t_1)W(t_2)\ldots W(t_k)$.

It can be seen that for a $t \in [V]$, $W(t)$ is the word over the alphabet V [9] which is obtained from t by erasing all parentheses.

Example 1.2. If $V=\{v_1, v_2, v_3\}$ and $t=v_1(v_2)(v_1(v_2)(v_2))$, then $W(t)= =v_1v_2v_1v_2v_2$.

For $T \subseteqq [V]$, $W(T)$ is the set $\{W(t) \mid t \in T\}$ and it is a λ-free language over V [9].

Let V be an alphabet. A *pseudoalgebra of type* V is a system $\mathbf{A}=\alpha\langle V, A\rangle$ where A is a nonvoid set disjoint from V, the *base set* of $\mathbf{A}$, and α is an operator which for each v from V determines a mapping αv: $A^* \to A$ [10]. The pseudoalgebra $\mathbf{A}$ is *finite* iff A is finite. A pseudoalgebra $\mathbf{B}=\beta\langle V, B\rangle$ of the same type V is called a *subpseudoalgebra* of $\mathbf{A}$ iff $B \subseteqq A$ and $\beta v(p)=\alpha v(p) \in B$ for every $v \in V$ and $p \in B^*$. Let $\mathbf{C}=\gamma\langle V, C\rangle$ be a pseudoalgebra and h a mapping of C into A. When for arbitrary $v \in V$ and $c_1c_2\ldots c_k \in C^*$ $(k \in \omega$, $c_i \in C$, $1 \leqq i \leqq k)$ $h(\gamma v(c_1c_2\ldots c_k))=\alpha v(h(c_1)h(c_2)\ldots \ldots h(c_k))$ holds, then h is a *homomorphism* of $\mathbf{C}$ into $\mathbf{A}$. If in addition h is *onto* then $\mathbf{A}$ is a *homomorphic image* of $\mathbf{C}$. Moreover, if h is an onto and one-to-one homomorphism then it is an *isomorphism*, and $\mathbf{A}$ and $\mathbf{C}$ are called *isomorphic* pseudoalgebras.

Let us consider a pseudoalgebra $\mathbf{A}=\alpha\langle V, A\rangle$ and the set of trees $[V, A]$. By α we define a mapping $\bar{\alpha}$ from $[V, A]$ into A in the following way:

1.3. (1) if $t=a$ for an $a \in A$, then $\bar{\alpha}(t)=a$;

(2) if $t=v(t_1)(t_2)\ldots(t_k)$ where $v \in V$ and $t_i \in [V, A]$ for $1 \leqq i \leqq k$, then $\bar{\alpha}(t)= =\alpha v(\bar{\alpha}(t_1)\bar{\alpha}(t_2)\ldots\bar{\alpha}(t_k))$.

The next lemma expresses a property of homomorphisms.

Lemma 1.1. If $\mathbf{A}=\alpha\langle V, A\rangle$ and $\mathbf{B}=\beta\langle V, B\rangle$ are pseudoalgebras and $h\colon B\to A$ a homomorphism of $\mathbf{B}$ into $\mathbf{A}$, then $h(\bar{\beta}(t))=\bar{\alpha}(t)$ holds for every t from $[V]$.

Proof. The proof is by induction on $\lg(t)$. First, it can be shown that if $t=v$ for some $v\in V$, the assertion is true. Then, supposing that the assertion is true for t_i, $1\leqq i\leqq k$, $k\geqq 1$, one can prove, that it is true for $t=v(t_1)(t_2)\dots(t_k)$.

A pseudoalgebra $\mathbf{A}=\alpha\langle V, A\rangle$ is *connected* iff $\bar{\alpha}([V])=A$.

The next statement is a consequence of the previous lemma.

Lemma 1.2. If $\mathbf{A}$ and $\mathbf{B}$ are connected pseudoalgebras of the same type and there exists a homomorphism h of $\mathbf{B}$ into $\mathbf{A}$ then h is uniquelly determined and $\mathbf{A}$ is a homomorphic image of $\mathbf{B}$.

A nonvoid set A with a binary operation "multiplication" defined on A is a *groupoid* $\tilde{A}$. The result of the multiplication of two elements a_1 and a_2 from A (their "product") will be denoted by $(a_1\cdot a_2)$, but expressions obtained by a successive application of multiplication can be simplified in the known way, i.e., by erasing the outer parentheses. For example, instead of $(a\cdot b)$ and $(a\cdot(b\cdot c))$ we shall write $a\cdot b$ and $a\cdot(b\cdot c)$, respectively.

If to a groupoid $\tilde{A}$ we add an alphabet V disjoint from A and introduce a mapping $\xi\colon V\to A$, we get a *designed groupoid* $\mathcal{A}$, in symbols, $\mathcal{A}=\langle\tilde{A}, V, \xi\rangle$. The designed groupoid $\mathcal{A}$ is *finite* iff $\tilde{A}$ (i.e. A) is finite. It is *connected* iff $\tilde{A}$ is generated by the set $\xi(V)=\{\xi(v)|v\in V\}$. If $\mathcal{B}=\langle\tilde{B}, V, \eta\rangle$ is a designed groupoid too, then the mapping $h\colon B\to A$ is termed *homomorphism* of $\mathcal{B}$ into $\mathcal{A}$ iff it is a homomorphism of $\tilde{B}$ into $\tilde{A}$ and $h(\eta(v))=\xi(v)$ holds for arbitrary $v\in V$. "Onto" homomorphism and isomorphism are defined in the natural way.

Using a designed groupoid $\mathcal{A}=\langle\tilde{A}, V, \xi\rangle$ one can consruct a pseudoalgebra which is denoted by ind $\mathcal{A}$ *(pseudoalgebra induced by $\mathcal{A}$)* in the next way: ind $\mathcal{A}=$ $=\alpha\langle V, A\rangle$, i.e. it is of type V, its base set is A, and for every $v\in V$, $p\in A^*$ and $a\in A$ holds

1.4. (1) $\alpha v(\lambda)=\xi(v)$;

(2) $\alpha v(pa)=\alpha v(p)\cdot a$.

In other terms, for $p=a_1a_2\dots a_k$ $(a_i\in A,\ 1\leqq i\leqq k)$ $\alpha v(p)=(\dots((\xi(v)\cdot a_1)\cdot a_2)\cdot$ $\cdot\dots)\cdot a_k$.

The following lemma will be useful in some proofs.

Lemma 1.3. Let $\mathcal{A}$ be a designed groupoid. Then ind $\mathcal{A}$ is connected iff $\mathcal{A}$ is connected.

Proof. Let $\mathcal{A}=\langle\tilde{A}, V, \xi\rangle$ and ind $\mathcal{A}=\alpha\langle V, A\rangle$. We shall show that an $a\in A$ is a product of some elements from $\xi(V)$ iff there exists a $t\in[V]$ with $\bar{\alpha}(t)=a$. We proceed by induction.

When the number of factors in the product is one, i.e. $a=\xi(v)$ for some $v\in V$, then it is equivalent to $a=\bar{\alpha}(t)$ for $t=v\in[V]$. If we suppose that the element a_i from A is a product of elements from $\xi(V)$ and $a_i=\bar{\alpha}(t_i)$ holds for some $t_i\in[V]$, where $1\leqq i\leqq k$, $k\geqq 1$, then $a=(\dots((\xi(v)\cdot a_1)\cdot a_2)\cdot\dots)\cdot a_k$ iff $a=\alpha v(a_1a_2\dots a_k)=$ $=\bar{\alpha}(t)$ for $t=v(t_1)(t_2)\dots(t_k)\in[V]$.

2. The groupoid of a pseudoalgebra

Let θ be a binary relation on a set X ($\theta \subseteq X^2$). Then we write $x_1 \theta x_2$ iff $(x_1, x_2) \in \theta$. If θ is an equivalence relation, then X/θ denote the partition of X induced by θ, i.e. the set of all equivalence classes modulo θ. For an $x \in X$ the equivalence class containing x will be denoted by $\theta(x)$.

Let $[V, A]$ be the set of trees of type V over a nonempty set A. We define the subsets $[V, A]'$ and $[V, A]''$ of this set in the following way:

2.1. $$[V, A]' = [V, A] - A$$

and

2.2.
$$[V, A]'' = \{t \mid t \in [V, A],\ t = v(a_1)(a_2)\dots(a_k) \text{ for } v \in V \text{ and } a_1 a_2 \dots a_k \in A^* \ (a_i \in A,\ 1 \leqq i \leqq k)\}.$$

If $t \in [V, A]'$ and $p = a_1 a_2 \dots a_k$ is an element from A^* ($a_i \in A$, $1 \leqq i \leqq k$), then let $t\bar{p} = t(a_1)(a_2)\dots(a_k)$ ($\in [V, A]'$).

Let us suppose that $\mathbf{A} = \alpha\langle V, A\rangle$ is a pseudoalgebra of type V. On the set $[V, A]'$ we define the relation $\varrho' \subseteq ([V, A]')^2$ in the next way:

2.3. for arbitrary trees t_1 and t_2 from $[V, A]'$ $t_1 \varrho' t_2$ holds iff $\bar{\alpha}(t_1\bar{p}) = \bar{\alpha}(t_2\bar{p})$ is satisfied for each $p \in A^*$.

The relation ϱ' is evidently an equivalence relation on the set $[V, A]'$. Into the set $[V, A]'/\varrho'$ of equivalence classes modulo ϱ' one can introduce a binary operation — multiplication — in the next manner: for arbitrary t_1 and t_2 from $[V, A]'$

2.4. $$\varrho'(t_1) \cdot \varrho'(t_2) = \varrho'(t) \quad \text{where} \quad t = t_1(t_2).$$

One can easily prove that this operation is well defined.

Now, we have a groupoid $\overgroup{[V, A]'/\varrho'}$ whose multiplication is defined by 2.4. The designed groupoid $\langle \overgroup{[V, A]'/\varrho'}, V, \alpha'\rangle$ where $\alpha'(v) = \varrho'(v)$ holds for each $v \in V$, we call the *groupoid of the pseudoalgebra* $\mathbf{A}$ and denote it by $\mathcal{G}(\mathbf{A})$.

Since $\varrho'' = \varrho' \cap ([V, A]'')^2$ is an equivalence relation too, however now on the set $[V, A]''$ ($\subseteq [V, A]'$), and each equivalence class modulo ϱ' contains elements from $[V, A]''$ (if $t_1 = v(r_1)(r_2)\dots(r_k)$ and $\bar{\alpha}(r_i) = a_i \in A$, $1 \leqq i \leqq k$, then $t_1 \varrho' t_2$, where $t_2 = v(a_1)(a_2)\dots(a_k) \in [V, A]''$). If on the set $[V, A]''/\varrho''$ we define the multiplication by

2.5. $$\varrho''(t_1) \cdot \varrho''(t_2) = \varrho''(t), \quad \text{where} \quad t = t_1(\bar{\alpha}(t_2)),$$

we get a groupoid $\overgroup{[V, A]''/\varrho''}$ isomorphic to $\overgroup{[V, A]'/\varrho'}$. Namely, it can be easily shown that the mapping $f\colon [V, A]''/\varrho'' \to [V, A]'/\varrho'$, which is defined by $f((\varrho''(t)) = \varrho'(t)$, is an isomorphism between these groupoids. For this reason the designed groupoid $\langle \overgroup{[V, A]''/\varrho''}, V, \alpha''\rangle$ where $\alpha''(v) = \varrho''(v)$ holds for each $v \in V$, is isomorphic to $\mathcal{G}(\mathbf{A})$. That means that we can consider this designed groupoid to be equal to $\mathcal{G}(\mathbf{A})$.

We investigate now the nature of the elements of $\mathcal{G}(\mathbf{A})$. For this purpose, for every $t \in [V, A]'$ let us introduce a mapping $\tilde{\alpha}t\colon A^* \to A$ in the next way: for arbitrary $p \in A^*$ let $\tilde{\alpha}t(p) = \bar{\alpha}(t\bar{p})$. Since for $t = v$ ($v \in V$), $\tilde{\alpha}t(p) = \alpha v(p)$ holds $\tilde{\alpha}$ is an extension of α. This mapping $\tilde{\alpha}t$ can be called the *mapping induced by t in* $\mathbf{A}$. From 2.3 we conclude now that *the set of elements of* $\mathcal{G}(\mathbf{A})$, up to notation, *is the same as the set of all different mappins induced by trees from* $[V, A]'$ *(or* $[V, A]''$*) in* $\mathbf{A}$.

In a similar way as we obtained the relation ϱ'' we can get the relation $\dot{\varrho}=\varrho'\cap\cap([V])^2$. The set $[V]/\dot{\varrho}$ of equivalence classes modulo $\dot{\varrho}$ equals to the set of all mappings induced by trees from $[V]$ in $\mathbf{A}$, and it is a subset of all mappings induced by trees from $[V, A]'$. This set forms a groupoid with the multiplication defined as in 2.4, and this groupoid is the subgroupoid of $\overline{[V, A]'}/\varrho'$ generated by the set $\{\varrho'(v)|v\in V\}$. (For each t from $[V]$, $\varrho'(t)$ is a product of elements from $\{\varrho'(v)|v\in V\}$. This can be proved by induction on $\lg(t)$.) If $\mathbf{A}$ is connected then $\overline{[V, A]'}/\varrho'$ is generated by the set $\{\varrho'(v)|v\in V\}$ (since then for every t from $[V, A]'$ there is an r from $[V]$ for which $r\varrho' t$ is valid; r can be getted from t by substituting each $a\in A$ with an r_a from $[V]$ for which $\bar{\alpha}(r_a)=a$). Now the next lemma, by the Lemma 1.3, follows from the fact that the set $\{\varrho'(v)|v\in V\}$ equals to the set $\alpha'(V)$.

Lemma 2.1. If the pseudoalgebra $\mathbf{A}$ is connected, then so is ind $\mathcal{G}(\mathbf{A})$.

The following theorem gives a connection between a pseudoalgebra and its groupoid.

Theorem 2.1. Let $\mathbf{A}=\alpha\langle V, A\rangle$ be a pseudoalgebra and $\mathcal{G}(\mathbf{A})$ its groupoid. Then the following assertions are valid:

1° There exists a homomorphism h from ind $\mathcal{G}(\mathbf{A})$ into $\mathbf{A}$.

2° If $\mathbf{A}$ is connected then

(a) the homomorphism h is completely determined and it is an *onto* homomorphism;

(b) if for some connected designed groupoid $\mathcal{B}=\langle\tilde{B}, V, \eta\rangle$ there exists a homomorphism from ind $\mathcal{B}$ into $\mathbf{A}$, then $\mathcal{G}(\mathbf{A})$ is a homomorphic image of $\mathcal{B}$ and ind $\mathcal{G}(\mathbf{A})$ is a homomorphic image of ind $\mathcal{B}$;

(c) if $\mathbf{A}=$ ind $\mathcal{A}$ for some designed groupoid $\mathcal{A}$, then $\mathcal{A}$ is isomorphic to $\mathcal{G}(\mathbf{A})$ and therefore, $\mathbf{A}$ is isomorphic to ind $\mathcal{G}(\mathbf{A})$.

Proof. 1° Since $\mathcal{G}(\mathbf{A})=\langle\overline{[V, A]'}/\varrho', V, \alpha'\rangle$, the mapping $h\colon [V, A]'/\varrho'\to A$ for which $h(\varrho'(t))=\bar{\alpha}(t)$ holds, where t is an arbitrary element from $[V, A]'$, is well defined by 2.3. It can be easily shown that h is a homomorphism from ind $\mathcal{G}(\mathbf{A})$ into $\mathbf{A}$.

2° (a) It follows from 1°, and Lemmas 2.1 and 1.2.

(b) Let ind $\mathcal{B}=\beta\langle V, B\rangle$. By Lemma 1.3, ind $\mathcal{B}$ is connected and, by Lemma 1.2, $\mathbf{A}$ is a homomorphic image of ind $\mathcal{B}$ under the mapping $h\colon B\to A$ which is determined by Lemma 1.1. From these facts it follows that the mapping $f\colon B\to\to[V, A]'/\varrho'$, where $f(\bar{\beta}(t))=\varrho'(t)$ holds for arbitrary $t\in[V]$, is well defined. Indeed, if for $t_1, t_2\in[V]$, $\bar{\beta}(t_1)=\bar{\beta}(t_2)$ holds, then $\varrho'(t_1)=\varrho'(t_2)$ holds too. It can be checked as follows.

Since h is a homomorphism from ind $\mathcal{B}$ onto $\mathbf{A}$, then for $a_1, a_2, \dots, a_k$ from A there exists $b_1, b_2, \dots, b_k$ from B such, that $h(b_i)=a_i$ for $1\leqq i\leqq k$. Then

$$\begin{aligned}\bar{\alpha}(t_1(a_1)(a_2)\dots(a_k)) &= h(\bar{\beta}(t_1(b_1)(b_2)\dots(b_k))) =\\ = h((\dots(\bar{\beta}(t_1)\cdot b_1)\cdot b_2)\cdot\dots)\cdot b_k) &= h((\dots(\bar{\beta}(t_2)\cdot b_1)\cdot b_2)\cdot\dots)\cdot b_k) =\\ = h(\bar{\beta}(t_2(b_1)(b_2)\dots(b_k))) &= \bar{\alpha}(t_2(a_1)(a_2)\dots(a_k)) \quad \text{(see 2.3).}\end{aligned}$$

Moreover, the mapping f is a homomorphism from ind $\mathscr{B}$ into ind $\mathscr{G}(\mathbf{A})$, and since ind $\mathscr{B}$ is connected then from Lemmas 2.1 and 1.2 it follows that f is an onto homomorphism.

The same mapping f is a homomorphism from $\mathscr{B}$ onto $\mathscr{G}(\mathbf{A})$.

(c) The proof is a routine computation.

From the assertion 2° (c) of the previous theorem, it follows that each connected designed groupoid is the groupoid of a pseudoalgebra.

3. Finite pseudoalgebras

Let us suppose that the pseudoalgebra $\mathbf{A}=\alpha\langle V, A\rangle$ is finite, i.e., A is a finite set. For arbitrary $v\in V$ and $a\in A$, let $\mathscr{L}(v, a)$ denote the set $\{p\,|\,p\in A^*, \alpha v(p)=a\}$. It is evident, that $\mathscr{L}(v, a)$ is a language over A. We shall say that the pseudoalgebra $\mathbf{A}$ is *regular* iff $\mathscr{L}(v, a)$ is a regular language over A [9] for each $v\in V$ and $a\in A$.

The next theorem is of great importance for our approach.

Theorem 3.1. The groupoid of a finite pseudoalgebra $\mathbf{A}$ is finite iff $\mathbf{A}$ is regular.

Proof. Let $\mathbf{A}=\alpha\langle V, A\rangle$. It is known from the previous chapter, that $\mathscr{G}(\mathbf{A})$ is isomorphic to the designed groupoid $\langle [V, A]''/\varrho'', V, \alpha''\rangle$. Therefore, $\mathscr{G}(\mathbf{A})$ is finite iff there are finitely many equivalence classes modulo ϱ''.

For arbitrary $v\in V$ let $\varrho''_v=\varrho''\cap([\{v\}, A]'')^2$. The equivalence relations ϱ''_v, when v is running through V, have the property that each equivalence class modulo ϱ'' is the union of some equivalence classes modulo ϱ''_v such that for each $v\in V$ at most one equivalence class modulo ϱ''_v occurs in this union. Consequently, for the finiteness of V, there are finitely many equivalence classes modulo ϱ'' iff there are finitely many equivalence classes modulo ϱ''_v, i.e. $[\{v\}, A]''/\varrho''_v$ is finite, for each $v\in V$.

We give now a necessary and sufficient condition for the finiteness of $[\{v\}, A]''/\varrho''_v$. From 2.3 we have that for arbitrary $t_1=v\bar{p}$ and $t_2=v\bar{q}$ from $[\{v\}, A]''$, where p and q are in A^*, $t_1\varrho''_v t_2$ holds iff $\bar{\alpha}(t_1\bar{r})=\bar{\alpha}(t_2\bar{r})$ is valid for every $r\in A^*$, which is the same as

$$3.1. \qquad \alpha v(pr) = \alpha v(qr) \quad \text{for every} \quad r\in A^*.$$

Now, we can induce an equivalence relation σ_v on the set A^* in the next way: $p\sigma_v q$ holds iff 3.1 is valid. It is obvious that A^*/σ_v is finite iff $[\{v\}, A]''/\varrho''_v$ is finite.

An other equivalence relation $\bar{\sigma}_v$ on A^* can be defined by: $p\bar{\sigma}_v q$ iff $\alpha v(p)=$ $=\alpha v(q)$. In the members of the partition $A^*/\bar{\sigma}_v$ we can recognise the sets $\mathscr{L}(v, a)$ for those $a\in A$ for which $\mathscr{L}(v, a)$ is nonempty. Moreover, the partition A^*/σ_v is a refinement of $A^*/\bar{\sigma}_v$. From the definitions of σ_v and $\bar{\sigma}_v$ it can be seen that the partition A^*/σ_v is the maximal right automaton-partition of the set A^* written into the partition $A^*/\bar{\sigma}_v$ (see [5]). It is finite iff each member of $A^*/\bar{\sigma}_v$ is a regular language over the alphabet A, in other terms, iff each $\mathscr{L}(v, a)$ is regular.

4. Pseudoautomata

Let $\mathbf{A}=\alpha\langle V, A\rangle$ be a pseudoalgebra of type V. Selecting a subset A_F of A, we get from $\mathbf{A}$ a *pseudoautomaton* $\overline{\mathbf{A}}=(\mathbf{A}, A_F)=(\alpha\langle V, A\rangle, A_F)$. Sets V, A and A_F are the sets of *inputs, states* and *final states* of $\overline{\mathbf{A}}$, respectively, while α can be called the *transition function* of $\overline{\mathbf{A}}$ (see [10]). The pseudoautomaton $\overline{\mathbf{A}}$ is *finite, regular* or *connected* iff the pseudoalgebra $\mathbf{A}$ is finite, regular or connected, respectively. The *set of trees represented* by $\overline{\mathbf{A}}$, in symbols $\mathscr{T}(\overline{\mathbf{A}})$, is a subset of $[V]$ defined by

4.1. $$\mathscr{T}(\overline{\mathbf{A}}) = \{t \mid t\in[V],\, \bar{\alpha}(t)\in A_F\}.$$

For an alphabet V, a subset T of trees from $[V]$ will be called *recognizable* iff it is represented by a finite pseudoautomaton. In the case when the pseudoautomaton is regular, T is called *pseudoregular.*

The *language represented* by a pseudoautomaton $\overline{\mathbf{A}}$, in symbols $\mathscr{L}(\overline{\mathbf{A}})$, is defined by

4.2. $$\mathscr{L}(\overline{\mathbf{A}}) = W(\mathscr{T}(\overline{\mathbf{A}})) = \{W(t) \mid t\in\mathscr{T}(\overline{\mathbf{A}})\}.$$

The set $\mathscr{L}(\overline{\mathbf{A}})$ is, obviously, a language over the set of inputs of $\overline{\mathbf{A}}$.

Let $\overline{\mathbf{A}}=(\mathbf{A}, A_F)$ and $\overline{\mathbf{B}}=(\mathbf{B}, B_F)$ be two pseudoautomata, where $\mathbf{A}=\alpha\langle V, A\rangle$ and $\mathbf{B}=\beta\langle V, B\rangle$ are pseudoalgebras of the same type V. If there is a mapping $h\colon B\to A$ which is a homomorphism from $\mathbf{B}$ into $\mathbf{A}$, and in addition for every b from B, $b\in B_F$ iff $h(b)\in A_F$ then h is a *homomorphism from* $\overline{\mathbf{B}}$ *into* $\overline{\mathbf{A}}$. If h is also a homomorphism (isomorphism) of $\mathbf{B}$ onto $\mathbf{A}$, then we say that h is a *homomorphism (isomorphism) of* $\overline{\mathbf{B}}$ *onto* $\overline{\mathbf{A}}$. In this case $\overline{\mathbf{A}}$ *is a homomorphic image of* $\overline{\mathbf{B}}$ (h is an *isomorphism between* $\overline{\mathbf{A}}$ *and* $\overline{\mathbf{B}}$). We shall say that the pseudoautomaton $\overline{\mathbf{B}}$ is a *subpseudoautomaton* of $\overline{\mathbf{A}}$ iff $\mathbf{B}$ is a subpseudoalgebra of $\mathbf{A}$ and $B_F=A_F\cap B$. The pseudoautomaton $\overline{\mathbf{B}}$ is the *trunk* of the pseudoautomaton $\overline{\mathbf{A}}$ iff $\overline{\mathbf{B}}$ is the connected subpseudoautomaton of $\overline{\mathbf{A}}$. (Note that the trunk is completely determined and $B=\{\bar{\alpha}(t) \mid t\in[V]\}$.)

Two pseudoautomata $\overline{\mathbf{A}}$ and $\overline{\mathbf{B}}$ will be called *equivalent* iff $\mathscr{T}(\overline{\mathbf{A}})=\mathscr{T}(\overline{\mathbf{B}})$. It is evident, that for equivalent pseudoautomata $\overline{\mathbf{A}}$ and $\overline{\mathbf{B}}$, $\mathscr{L}(\overline{\mathbf{A}})=\mathscr{L}(\overline{\mathbf{B}})$ also holds (the opposite is not true).

The next result is a direct consequence of our definitions and Lemma 1.1.

Theorem 4.1. If there exists a homomorphism of the pseudoautomaton $\overline{\mathbf{B}}$ into the pseudoautomaton $\overline{\mathbf{A}}$, then $\overline{\mathbf{A}}$ and $\overline{\mathbf{B}}$ are equivalent. Consequently, $\mathscr{L}(\overline{\mathbf{A}})=$ $=\mathscr{L}(\overline{\mathbf{B}})$.

Moreover, we obviously have

Theorem 4.2. Any pseudoautomaton is equivalent to each of its subpseudoautomata. Especially, a pseudoautomaton is equivalent to its trunk.

The second part of the previous theorem shows that connected pseudoautomata are of special interest.

From Lemma 1.2 we can get the next theorem.

Theorem 4.3. Let $\overline{\mathbf{A}}$ and $\overline{\mathbf{B}}$ be connected pseudoautomata with a common set of inputs. Assume that h is a homomorphism of $\overline{\mathbf{B}}$ into $\overline{\mathbf{A}}$. Then h is uniquelly determined and $\overline{\mathbf{A}}$ is a homomorphic image of $\overline{\mathbf{B}}$.

By *the groupoid of a pseudoautomaton* $\overline{\mathbf{A}}=(\mathbf{A}, A_F)$, in symbols $\mathcal{G}(\overline{\mathbf{A}})$ we mean the designed groupoid $\mathcal{G}(\mathbf{A})$.

If for a pseudoautomaton $\overline{\mathbf{B}}=(\mathbf{B}, B_F)$ the pseudoalgebra $\mathbf{B}$ is of the form ind $\mathcal{B}$, where $\mathcal{B}$ is a designed groupoid, we shall call $\overline{\mathbf{B}}$ *groupoid pseudoautomaton*. The groupoid pseudoautomaton $\overline{\mathbf{B}}$ is *the groupoid pseudoautomaton belonging to* $\overline{\mathbf{A}}=(\alpha\langle V, A\rangle, A_F)$, iff $\mathcal{B}=\mathcal{G}(\overline{\mathbf{A}})=\langle [V, A]'/\varrho', V, \alpha'\rangle$ and $B_F=\{\varrho'(t)|t\in[V, A]', \bar{\alpha}(t)\in A_F\}$. In this case $\overline{\mathbf{B}}$ is denoted by $G(\overline{\mathbf{A}})$.

On account of our definitions and previous results we can state the following three theorems. The first of them is based on Lemma 2.1.

Theorem 4.4. If $\overline{\mathbf{A}}$ is a connected pseudoautomaton then $G(\overline{\mathbf{A}})$ is connected too.

The next theorem follows from Theorem 2.1, properties of the mapping h from the proof of the assertion 1° in this theorem and from our definitions.

Theorem 4.5. Let $\overline{\mathbf{A}}=(\mathbf{A}, A_F)$ be a pseudoautomaton and $G(\overline{\mathbf{A}})$ the groupoid pseudoautomaton belonging to $\overline{\mathbf{A}}$. The following assertions are valid:

1° There exists a homomorphism h from $G(\overline{\mathbf{A}})$ into $\overline{\mathbf{A}}$.

2° If $\overline{\mathbf{A}}$ is connected then

(a) $G(\overline{\mathbf{A}})$ is connected and h is completely determined *onto* homomorphism;

(b) if $\overline{\mathbf{B}}$ is a connected groupoid pseudoautomaton and there exists a homomorphism of $\overline{\mathbf{B}}$ into $\overline{\mathbf{A}}$ then $\overline{\mathbf{A}}$ is a homomorphic image of $\overline{\mathbf{B}}$ and $G(\overline{\mathbf{A}})$ is a homomorphic image of $\overline{\mathbf{B}}$;

(c) if $\overline{\mathbf{A}}$ is a groupoid pseudoautomaton then $G(\overline{\mathbf{A}})$ is isomorphic to $\overline{\mathbf{A}}$.

From the previous theorem it is clear, that the relation between a pseudoautomaton and its groupoid is similar to the relation between an automaton and its monoid (see [5]).

The following theorem is important in investigating finite pseudoautomata. It follows from Theorem 3.1.

Theorem 4.6. If $\overline{\mathbf{A}}$ is a finite pseudoautomaton then $G(\overline{\mathbf{A}})$ is finite iff $\overline{\mathbf{A}}$ is regular.

In the next theorem languages represented by regular pseudoautomata are characterised. (For language-theoretic terminology used here, see [9]. It should be emphasized that here in the definition of context-free grammar we take a set of initial letters instead of a single letter. Obviously, this modification does not alter the generative capacity of context-free grammars.)

Theorem 4.7. A language over an alphabet V is a λ-free context-free language iff it is represented by a regular pseudoautomaton with set of inputs V.

Proof. Since a finite groupoid pseudoautomaton is regular, by Theorems 4.1, 4.5 (assertion 1°) and 4.6, it will be sufficient to prove that a language is λ-free and context-free iff it is represented by a finite groupoid pseudoautomaton.

First we prove the sufficiency of the condition.

Let $\overline{\mathbf{A}}=(\mathbf{A}, A_F)$ be a finite groupoid pseudoautomaton, i.e. $\mathbf{A}=\text{ind}\,\mathcal{A}$, where $\mathcal{A}=\langle\tilde{A}, V, \xi\rangle$ is a finite designed groupoid. Using $\overline{\mathbf{A}}$ one can construct a λ-free context-free grammar $\Gamma(\overline{\mathbf{A}})$ of Chomsky normal form in the next way: $\Gamma(\overline{\mathbf{A}})=$ $=(A, V, A_F, \Pi)$, where A and V are the nonterminal and terminal alphabets, respectively. Moreover, A_F is the set of initial letters and Π is the set of productions for which $\Pi = \Pi_1 \cup \Pi_2$ where

$$\Pi_1 = \{a \to v | v \in V,\ a \in A,\ \xi(v) = a\},$$

and

$$\Pi_2 = \{a \to a_1 a_2 | a, a_1, a_2 \in A,\ a = a_1 \cdot a_2 \text{ in } \tilde{A}\}.$$

It can be shown that the language generated by $\Gamma(\bar{\mathbf{A}})$, in symbols $\mathscr{L}(\Gamma(\bar{\mathbf{A}}))$, equals to $\mathscr{L}(\bar{\mathbf{A}})$. For this purpose it is enough to show that for arbitrary $a \in A$ and $p \in V^+$, $a \Rightarrow^* p$ is valid iff there exists a t from $[V]$ for which $W(t) = p$ and $\bar{\alpha}(t) = a$ where α is the transition function of $\bar{\mathbf{A}}$. The proof is by induction. First, it can be seen easily that if $p = v$ and $t = v$ for an arbitrary v from V, then $a \Rightarrow^* p$ holds iff $\bar{\alpha}(t) = a$. Furthermore, let us suppose that for $p_i \in V^+$, $a_i \in A$, $1 \leqq i \leqq k$, $k \geqq 1$, it have been shown that $a_i \Rightarrow^* p_i$ is valid iff there exists a $t_i \in [V]$ for which $W(t_i) = p_i$ and $\bar{\alpha}(t_i) = a_i$ hold. But then, there is a sequence of productions from Π_2: $a \to b_k a_k$, $b_k \to b_{k-1} a_{k-1}, \ldots, b_2 \to b_1 a_1$, and a production from Π_1: $b_1 \to v$ such that by a succesive application of them we get the derivation $a \Rightarrow^* v a_1 a_2 \ldots a_k$, and by $a_i \Rightarrow^* p_i$, $a \Rightarrow^* v p_1 p_2 \ldots p_k = p$, $p \in V^+$ iff there is a sequence of identities $a = b_k \cdot a_k$, $b_k = b_{k-1} \cdot a_{k-1}, \ldots, b_2 = b_1 \cdot a_1$, $b_1 = \xi(v)$ in $\tilde{A}$, from which we get $a = (\ldots((\xi(v) \cdot a_1) \cdot a_2) \cdot \ldots) \cdot a_k = \alpha v(a_1 a_2 \ldots a_k)$, and for $a_i = \bar{\alpha}(t_i)$, $a = \bar{\alpha}(t)$, where $t = v(t_1)(t_2)\ldots(t_k)$, and moreover $W(t) = v p_1 p_2 \ldots p_k = p$.

To prove the necessity of the condition, let us suppose that L is a λ-free context-free language over V. Then there exists a grammar in Chomsky normal form generating L. From this grammar, by the method applied to the proof of Theorem 3.1 of Part Three in [9] (with the difference that here the set S_0' contains the empty subset of S_0, as well) we can get an equivalent grammar Γ of the form (A, V, A_F, Π). For the set of productions Π we have $\Pi = \Pi_1 \cup \Pi_2$, where Π_1 contains productions of the form $a \to v$ only ($a \in A, v \in V$) such that for each v from V there is exactly one production of this form, while Π_2 contains productions of the form $a \to a_1 a_2$ only ($a, a_1, a_2 \in A$) such that for each (a_1, a_2) from A^2 there is exactly one production of this form. From these properties of Γ it follows that there is a finite groupoid pseudoautomaton $\bar{\mathbf{A}}$ with $\Gamma = \Gamma(\bar{\mathbf{A}})$.

NOTE. If for a λ-free context-free grammar $\Gamma = (N, V, N', \Pi)$ we introduce the grammar $\Gamma^T = (N, V \cup \pi, N', \Pi^T)$ where Π^T is obtained from Π by substituting every production $a \to a_1 a_2 \ldots a_k$ ($a \in N, k \geqq 1, a_i \in N \cup V$, $1 \leqq i \leqq k$) in Π by the production $a \to a_1(a_2(\ldots(a_k)))$ then the language $\mathscr{L}(\Gamma^T)$ generated by Γ^T is a subset of $[V]$. Let us call $\mathscr{L}(\Gamma^T)$ *the set of trees generated by* Γ. It was shown in [3] that the following assertion is valid: *a set of trees is pseudoregular iff it is the set of trees generated by a λ-free context-free grammar.* Since $W(\mathscr{L}(\Gamma^T)) = \mathscr{L}(\Gamma)$ is valid, our Theorem 4.7 is now a consequence of this assertion. (Moreover, for $\Gamma(\bar{\mathbf{A}})$ from the proof of Theorem 4.7, $\mathscr{L}(\Gamma(\bar{\mathbf{A}})^T) = \mathscr{T}(\bar{\mathbf{A}})$ is valid.)

5. Relations between various types of the sets of trees

We shall say that a set of trees is *regular* iff it is represented by a such modification of our finite pseudoautomaton that if the transition function, the set of inputs and the set of states are denoted by α, V, and A, respectively, than for any $v \in V$, αv: $\{p | p \in A^*, \lg(p) \in K_v\} \to A$, where K_v is a finite nonempty subset of ω. It can be checked that this definition of regular sets of trees is equivalent to the definition of recognizable sets in [11].

Each regular set of trees is pseudoregular. (It can be seen by adding to our modified finite pseudoautomaton a new state b and mapping by αv all remaining words from $(A \cup \{b\})^*$ into it. The pseudoautomaton obtained by this procedure is regular.) The opposite is not true, i.e. there exist such sets of trees which are pseudoregular but not regular (for example $[V]$ for an alphabet V).

If a set of trees is pseudoregular it is recognizable by definition. However, there are recognizable sets of trees which are not pseudoregular. To show it, let us take an alphabet by a single letter v. Then each subset of the set

$$\{t \mid t = v\underbrace{(v)(v)\dots(v)}_{k \text{ times}} = v(v)^k,\ k \in \omega\},$$

is represented by a pseudoautomaton which has at most three states. Therefore, selecting a subset T of this set, for which $W(T)$ is not context-free, we get a recognizable set of trees and it is not pseudoregular by Theorem 4.7.

To finish these discussions, we demonstrate that for any alphabet V there are subsets of $[V]$ which are not recognizable. Let us suppose that v is an element of V and define a subset U of $[V]$ in the next way:

(1) $v \in U$;

(2) if $t \in U$, then $v(t) \in U$;

(3) the elements of U are those and only those which we get from (1) and (2) in a finite number of steps.

Every recognizable subset of U is regular, therefore, it is pseudoregular. Selecting from U a subset S for which $W(S)$ is not context-free, we get a set which is not recognizable.

6. Conclusion

From our point of view, we shall now answer to the question: what is the connection between pseudoautomata and automata?

The importance of connected pseudoautomata follows from Theorem 4.2. By the assertion 2° (a) of Theorem 4.5 every pseudoautomaton of this kind is a homomorphic image of a connected groupoid pseudoautomaton, and therefore (for Theorem 4.1) equivalent to it.

Let $\overline{\mathbf{A}} = (\alpha\langle V, A\rangle, A_F)$ be a connected groupoid pseudoautomaton, i.e. $\alpha\langle V, A\rangle = \operatorname{ind} \mathscr{A}$ for some connected designed groupoid $\mathscr{A} = \langle \tilde{A}, V, \xi\rangle$. Moreover, let $\tilde{A}$ be a monoid (semigroup with identity). Then, from the associativity it follows, that for each $p \in V^+$ the set $T(p) = \{t \mid t \in [V], W(t) = p\}$ has the property, that the whole set is represented by a single state of $\overline{\mathbf{A}}$. (This means that from $t_1, t_2 \in T(p)$ it follows $\bar{\alpha}(t_1) = \bar{\alpha}(t_2)$. It can be proved by induction on $\lg(p)$.) For this reason it may be chosen a representative from $T(p)$ which is simpler than other members of this set, and only it must be represented by the pseudoautomaton $\overline{\mathbf{A}}$. If $p = v_1 v_2 \dots v_k$ ($k \geqq 1$, $v_i \in V$, $1 \leqq i \leqq k$), then this representative can be $t = v_1(v_2(\dots(v_k)\dots))$ where arities of symbols $v_1, v_2, \dots, v_{k-1}$ equal to 1 and of v_k to 0. However, the situation becomes yet more simpler, if arities of each v from V equal to 1 and the other arities are ignored because they are unnecessary. But, it needs the introducing of a nullary symbol Λ which is not in V and whose realization is the identity e of the monoid $\tilde{A}$. Then the representative of $T(p)$ is the tree $v_1(v_2(\ldots(v_k(\Lambda))\ldots))$. By these modifications we got from $\bar{\mathbf{A}}$ a (connected) automaton in the sense of [4], [5], with initial state e. Now, an arbitrary homomorphic image of this automaton is a (connected) automaton too, its initial state is the image of e under the homomorphism, and moreover these automata are equivalent. (The first of them is a "monoid" automaton, but the second is an arbitrary one.)

By this interpretation, we got that the (ordinary) automaton is a simplification of the pseudoautomaton for the case when its groupoid is a monoid, and conversely, the concept of the pseudoautomaton is such a generalization of the concept of the automaton where its monoid is replaced with an arbitrary groupoid.

Abstract

The notion of a pseudoalgebra and that of a pseudoautomaton are introduced in a paper by Thatcher (1967). In this work it is shown that with a pseudoalgebra and with a pseudoautomaton a groupoid can be associated, in the same way as to a unary universal algebra and to an automaon a monoid can be corresponded.

FACULTY OF CIVIL ENGINEERING
24000 SUBOTICA
YUGOSLAVIA

References

[1] Brainerd, W. S., The minimalization of tree automata, *Inform. and Control*, v. *13*, 1968, pp. 484—491.

[2] Ferenci, F., A new representation of context-free languages by tree automata, *Found. Control Engrg.*, v. 1, 1976, pp. 217—222.

[3] Ferenci, F., *Generalized automata*, Dissertation, University of Novi Sad, 1977 (in Serbocroatian).

[4] Gécseg, F. and I. Peák, *Algebraic theory of automata*, Akadémiai Kiadó, Budapest, 1972.

[5] Глушков, В. М., Абстрактная теория автоматов, *Успехи матем. наук*, 16:5 (101), 1961, pp. 3—62.

[6] Grätzer, G. *Universal algebra*, D. van Nostrand Company, inc. Princeton, 1968.

[7] Мальцев, А. И., Алгебраические системы, Москва, 1970.

[8] Медведев, Ю. Т., О классе событий, допускающих представление в конечном автомате, *Сборник «Автоматы»*, Москва, 1956, pp. 385—401.

[9] Salomaa, A., *Formal languages*, Academic Press, New York, 1973.

[10] Thatcher, J. W., Characterizing derivation trees of context-free grammars through a generalization of finite automata theory, *J. Comput. System. Sci.*, v. 1, 1967, pp. 317—322.

[11] Thatcher, J. W., Generalized^2 sequential machine maps, *J. Comput. System. Sci.*, v. 4, 1970, pp. 257—287.

[12] Thatcher, J. W. and J. B. Wright, Generalized finite automata theory with an application to a decision-problem of second-order logic, *Math. Systems Theory*, v. 2, 1968, pp. 57—81.

(Received August 29, 1979)

INDEX — TARTALOM

ISSN 0324—721 X

Kiadja
Felelős szerkesztő: Gécseg Ferenc

79-5678 Szegedi Nyomda
Felelős vezető: Dobó József

Felelős kiadó: Gécseg Ferenc
Terjedelem 8,5 (A/5) ív
Készült az MSZ 5601 és MSZ 5602—55 szabvány szerint
A kézirat a nyomdába érkezett: 1979. november 30.