tanulmányok

158/1984

MTA Számitástechnikai és Automatizálási Kutató Intézet Budapest

MAGYAR TUDOMÁNYOS AKADÉMIA
SZÁMITÁSTECHNIKAI ÉS AUTOMATIZÁLÁSI KUTATÓ INTÉZETE

PROCEEDINGS

OF THE

3RD INTERNATIONAL MEETING OF YOUNG COMPUTER SCIENTISTS

Held at Smolenice Castle, Czechoslovakia,
October 22 -26, 1984

Edited by:

J. DEMETROVICS, Hungarian Academy of Sciences, Budapest,
and
J. KELEMEN, Comenius University, Bratislava

Published by Computer and Automation Institute of
the Hungarian Academy of Sciences, Budapest, in 1984

Tanulmányok 158/1984

A kiadásért felelős:

DR. VÁMOS TIBOR

Főosztályvezető

DEMETROVICS JÁNOS

ISBN 963 311 175 7
ISSN 0324-2951

Hozott anyagról sokszorosítva

8414941 MTA Sokszorosító, Budapest. F. v.: dr. Héczey Lászlóné

PREFACE

This volume contains papers contributed for presentation at the 3rd International Meeting of Young Computer Scientists (formerly the Czechoslovak - Soviet Conference of Young Computer Scientists), held at Smolenice Castle, Czechoslovakia, October 22 - 26, 1984.

The conference was organized by the Association of Slovak Mathematicians and Physicists with the aim to promote research in various areas of the computer science and to provide an opportunity for young computer scientists of exchanging their ideas and establishing professional relations.

The Proceedings include 4 invited and 42 submitted papers. The contributed texts have been completed by the authors in the camera ready form (except for Aladyev's paper) and have not been formally refereed. However, the members of the programme committee and the editors spent considerable time discussing a number of the papers and improving the formal level of several of them. It is anticipated, that most of the papers will appear in a more polished and completed form in scientific periodicals.

The members of the programme committee (S.K.Dulin,Moscow; J.Hvorecký, Bratislava; A.Kelemenová, Bratislava; P.Mikulecký, Bratislava; J.Pittl, Prague; M.Szijártó, Győr) and the editors are deeply indebted to all the contributors to the programme of the conference and to the Proceedings.

We wish to express our gratitude to the Computer and Automation Institute of the Hungarian Academy of Science for publishing the Proceedings. Special thanks go to Peter Mikulecký, the programme chairman of the conference for his substantial technical support and help in editing the Proceedings.

Budapest and Bratislava, June 1984

János Demetrovics
Jozef Kelemen

CONTENTS

Invited Papers

Theoretical Computer Science

Artificial Intelligence

Software Projects

Dialogue Systems

Miscellaneous

• INVITED PAPERS •

NEW RESULTS IN THE THEORY OF HOMOGENEOUS STRUCTURES

Victor Aladyev
Tallinn 200035, Paldinski mnt 171-26, USSR

During the recent years there has been a considerable interest in the theory of homogeneous structures /HS/ about which many interesting results have been obtained. Much of this work has been motivated by the growing interest in computer science and biological modelling. The HS is a formalization of the concept of an infinite regular array of identical finite-state machines uniformly interconnected in that sense that each machine can directly receive information via interconnecting wires from a finite number of neighbouring machines where the spatial arrangement of these neighbouring machines is the same relative to each machine in the array. Each machine can synchronously change its state at discrete time steps as a function of the states of the neighbouring machines. This function can change from time to time step,but will be identical for each machine in the array at any given time step. The simultaneous action of these local functions will define global functions which will act on the entire array changing configurations /CF/ of machine states in the array to other configurations. A d-dimensional HS /d-HS/ is an ordered quadruple $HS = \langle Z^d, A, \tau^{(n)}, X \rangle$, where

/1/ Z^d is a d-dimensional regular array
/2/ $A = \{0,1,...,a-1\}$ is the alphabet of the HS
/3/ $\tau^{(n)}$ is the global transition function of d-HS
/4/ X is the neighbourhood index of HS
/5/ n is the number of neighbouring machines in HS.

Such models have been applied in such diverse areas as pattern recognition, machine self-reproduction, cellular differentiation, evolution and development theories, theory of morphoge-

nesis, adaptive systems, parallel algorithms and parallel processing and so on [1]-[4]. HS can serve as the basis for modelling of many discrete processes and they present enough interesting independent objects of investigation as well. This paper considers such problems as modelling in HS, decomposability of the special global maps in HS, the complexity of configurations and global maps in HS, employment of HS in parallel programming, apparatus of investigations in the HS theory and so on. The latest results in this directions in detail can be characterizrd as follows. All definitions,notations and designations can be found in [1].

The problem of modelling in HS plays a very important role in the theory of HS itself and in its applications. A number of scientists have dealt with the problem, but it is necessary to note that neither the neighouring nor the stateset reduction techniques are necessarily optimal, and here we made an attempt to obtain the optimal technique along these lines. On the basis of our approaches we can formulate the following theorems [4].

Theorem 1. For an arbitrary d-dimensional ($d \geq 1$) HS there exists a binary HS′ of the same dimensionality, which 1-models it and whose template satisfies the following condition:

$$L=(I_1+1)^{d-1}(I_d+1)\prod_{1}^{d}(p_i+1),$$

where $p_1 \times p_2 \times \ldots \times p_d$ is the template of HS, $\log_2\log_2 4(a-1) \geq d$ and

$$I_1 = B + 2, \text{ where } B=\sqrt[d]{\log_2(a-1)}$$

$$I_d = \begin{cases} I_1 & \text{if } B-I_1 < 0,5 \\ I_1 + 1 & \text{otherwise.} \end{cases}$$

Theorem 2. For an arbitrary 1-dimensional HS with alphabet A there exists a 1-dimensional binary HS′ which 1-models it and whose template is $L=(n+1)(\log_2 a+n+E)$, $(E=4)$ if $a \leq 2^{19}$, otherwise $E=5$), where n is the size of the template of HS.

On the basis of a special simulation technique of this theorem the following theorem can be proved.

Theorem 3. A 1-dimensional HS is said to be universal if it models a universal Turing machine. There exists a universal 1-dimensional binary HS with a template of size n=17 .

To our knowledge this result is the best of its kind.

Theorem 4. For an arbitrary d-dimensional HS ($d \geq 1$) with the alphabet A there exists a d-dimensional binary HS´ which 1-models it and whose template is

$$L = [(p_1 + 1)(\log_2 a + E + p_1)] \times p_2 \times p_3 \times \ldots \times p_d ,$$

where $p_1 \times \ldots \times p_d$ is the template of HS, E=4 for $a < 2^{19}$ and E=5 otherwise.

With modeling in HS the problem of reliability of HS is linked . It is said that HS is real structure if at any moment $t>0$ the computation of a new state of an elementary automaton in HS (on the basis of local transition function $L^{(n)}(x_1,\ldots, x_n) = x_1'$) can be exposed to breaches, i.e. $L^{(n)}(x_1,\ldots,x_n)=x_1'' \neq x_1'$. Real HS is self-restoring if it is capable of above-mentioned breaches in the process of functioning. It is said that a real d-HS has $(1-1/K^d)\times 100\%$ - reliability if in any d-dimensional hypercube with edge of size K can be exposed to breaches any more than $p \geq 1$ elementary automata. With a glance to this suppositions the following results can be formulated [4].

Theorem 5. For any 2-dimensional real HS with $(1-1/K^2)\times 100\%$ - reliability ($d=2, K \geq 3$) and an alphabet A there exists a self-restoring HS of the same dimensionality which 2-models it and has an alphabet $A' = A \cup \{M\}$, and a global transition function $\tau(q)\tau(p)$, where is a reliable function with Moore´s neighbourhood index.

Theorem 6. Let an elementary automaton of the real HS with alphabet A within the limits of any t steps can perpetrate any more than $p \leq m$ breaches ($m < |t/2|$). Then there exists a self-restoring HS´ of the same dimensionality with the same neighbourhood index and alphabet A which (t+1)-models it.

In our book [1] we quite justly noted constructive defects of HS with symmetrical local functions. On the other hand, from our results in modelling in HS may be drawn that HS with symmetrical and asymmetrical local functions are equivalent with respect to computability.

Theorem 7. For any Turing machine with s symbols on tape and q internal states /MT $^{s}_{q}$/ there exists a 1-HS with alphabet A of cardinality 2(s+2q+1), Moore´s neighbourhood index and symmetrical function which 4-models it.

On the basis of this result the following theorem can be formulated.

Theorem 8. For any 1-HS with the alphabet A and Moore´s neighbourhood index there exists a 1-HS with symmetrical local function, alphabet A´ of cardinality 2(4a+5a+12) and Moore´s neighbourhood index, which 4W-models it, where W is the length of the rewritten finite configuration.

Two questions present undoubted interest: - Can an arbitrary HS be embedded in a reversible one of the same dimensionality? - Are there at all computation- and construction-universal reversible 1-dimensional HS ?

Furthermore, I am inclined to the opinion, that both problems have a negative solution. Indeed, enough wide classes of modelling of one HS by another of the same dimensionality corroborate this conclusion. To this effect we introduced two concepts of modelling: WM- and W- modelling in HS, which embrace a wide class of methods of modelling. On the basis of investigations of the above concepts of modelling we can formulate the following results [4].

Theorem 9. For any integer $d \geq 1$ there does not exist d-HS without NCF, which WM- or W-models it.

In our above-mentioned book [1] we discussed the question of community of the classical concept of HS. G.E.Tseitlin in connection with the further generalization of this concept in-

troduced heterogeneous periodically defined transformations /HPDT/. It can be verified that any HPDT is equivalent to some classical d-HS [4] .

Theorem 10. For any HPDT defined on d-dimensional abstract register R in alphabet A there exists a classical d-HS ($d \geq 1$) with alphabet A $\cup$ {b} ($b \notin A$), which 1-models it.

Thus, these and a number of other widenings of the classical concept of HS show that this concept possesses the sufficient degree of community. In that connection the question arises about the complexity of global functions in HS. We give an answer in the terms of the theory of recursive functions.

Theorem 11. Any global transition function $\tau^{(n)}$ of HS defined on the set $\overline{C}_A$ of finite configurations is a primitive recursive word function.

This result defines the position of the class of global functions of HS in the hierarchy of all recursive functions.

In the book [1] the following problem is discussed: Can any global function of HS of special type be presented in the form of composition of the finite number of more simple global functions of the same type? It is said that a global function $\tau^{(n)}$ in an alphabet A is more simple than a global function $\tau^{(m)}$ in the same alphabet if $n<m$. Such a problem is called composition problem.

On the basis of Yamada-Amoroso's results on completeness problems the negative solution of the general composition problem is presented [4]. There exist enough interesting classes of global functions, for which the decomposition problem is algorithmically solvable.

Theorem 12. Let MB be the set of all d-dimensional global functions such that for any $\tau^{(n)} \in MB$ and any CF $c \in \overline{C}_A$ $|c| < |c\tau^{(n)}|$ where $|c|$ is a minimum size of CF c. In the class BM of global functions the composition problem is algorithmically solvable.

Theorem 13. Let M be the set of all binary 1-dimensional global functions $\tau^{(n)}$ in HS /$n \geq 2$/. There exist global functions

$\tau^{(n)}$ of M which cannot be presented as a composition of the finite number of functions $\tau^{(n)} \in M$ $/n_i<n$, $i=1,\ldots,k./$.

On the basis of Yamada-Kaoru´s result on the completeness problem in 1-dimensional binary HS we have the following result.

Theorem 14. The general decomposition problem in the class of binary 1-dimensional injective global functions $\tau^{(n)}$ has a negative solution.

Further, the following interesting problem is discussed: Is it decidable whether an arbitrary global function over an alphabet A can be presented as composition of the finite number of more simple global functions in the same alphabet ? Let a global function $\tau^{(n)}$ in 1-dimensional HS with alphabet A be defined by the local function:

$$L^{(n)}(x_1,\ldots,x_n)=x_1 + x_n + \sum_{j=1}^{k} x_{i_j} \pmod a \qquad /1/$$

for $0\leq k\leq n-2$ and $i_j \in \{2,3,\ldots,n-1\}$.

It is well-known that in such HS any finite configuration is self-reproducing. The relation between the concept of complexity and decomposition problem in HS is stated. Then the relation between A(X) and other famous measures of complexity is presented [4] .

The classical concept of HS is very bulky for modelling of complex processes and phenomena, in a number of cases. The modelling itself in such HS becomes complex, boundness and loses sometimes any sense. In [1] we discuss a class of HS which to a certain extent is similar to nervous tissue. In such HS /HS´/ each machine can directly receive information from its immediate neighbours and each machine can synchronously change its state and outcome impulses at discrete time steps as a function of its state and incomung impulses. Such HS allow to obtain extremely lucid picture of information streams which operate by functioning of algorithms realized in HS.HS´, by definition, is a quadruple $HS´=<Z^d,A,I,\psi>$,where Z^d and A are defined as for classical HS, I is a set of impulses, and ψ is

a functional algorithm /FA/ of HS´. The next result can be easily verified: Any HS´=$<Z^1,A,I,\psi>$ can be constructively embedded in the classical HS= $<Z^1,A,\tau^{(3)},X>$. Furthermore under certain conditions HS´ can be embedded in HS = $<Z^1,A \cup I,\tau^{(3)},X>$, where X is Moore´s neighbourhood index. Such conditions for HS´ are discussed. The following result can be shown [4].

Theorem 15. Any HS´ = $<Z^1,A,I,\psi>$ can be embedded into classical HS = $<Z^1,A \cup I,\tau^{(7)},X>$, where X is the neighbourhood index.

Formal language theory is by its very essence an interdisciplinary area of science: the need for a formal grammatical or machine description of languages used arises in various scientific disciplines. Therfore, influences from outside the mathematical theory itself have often enriched the theory of formal languages.

Perhaps the most prominent examples of such an outside stimulation is provided by the theory of L-systems and τ_n-grammars. L-systems were originated by A.Lindenmayer in connection with biological considerations in 1968. We defined τ_n-grammars in 1974 [1]. Two main novel features brought about by the theory of L-systems and τ_n-grammars. From its very beginning are:
/1/ parallelism in the rewriting process and
/2/ the notion of a grammar conceived as a description of a dynamic /i.e., taking place in time / rather than a static entity.

The later feature /2/ initiated an intensive study of sequences /in contrast to sets / of words, as well as of grammars without nonterminal letters. During the past five-year period, the research in the area of L-systems and τ_n-grammars has been most active. For a systematic presentation of the essentials of τ_n-grammars mathematical theory we refer to [1]. Here we present only some problems in τ_n-grammars theory and discuss non-deterministic τ_n-grammars. A number of results present decisions of the previous open problems in τ_n-grammars[4].

Definition 1. A τ_n-grammar is an ordered quadruple $\tau_n = < n,A,\tau^{(n)},c_0 >$, where

/1/ n is the size of the template of n-HS,

/2/ A is the alphabet of the HS,

/3/ $\tau^{(n)}$ is the global transition function of 1-HS / production in grammar /,

/4/ $c_0 \varepsilon \overline{C}_A$ is an axiom / initial CF in 1-HS/.

The language generated by τ_n-grammar is the set $L(\tau_n) = \{c \varepsilon \overline{C}_A : c_0 \xrightarrow{\tau} c\}$.

The following results in such languages can be presented [4].

Theorem 16. There exist $L(\tau_n)$-languages and a set of words /CF/ $C = \{c_1, \ldots, c_d\}$ $(c_i \varepsilon \overline{C}_A - \{\emptyset\}$ $i = 1, \ldots, d)$ such that sets $L(\tau_n)$-C and $L(\tau_n) \cup C$ are not $L(\tau_m)$-languages.

Theorem 17. There exist $L(\tau_n)$-languages for which the reversions $L^{-1}(\tau_n)$ are not $L(\tau_m)$-languages.

Theorem 18. The problem of nonempty intersection for arbitrary $L(\tau_n)$ -languages is undecidable.

Definition 2. A nondeterministic T-grammar is an ordered quadruple $T = \langle d, A, \tau, c_0 \rangle$, where

/1/ A is the alphabet of the grammar

/2/ d is the dimensionality of words /CF/

/3/ $c_0 \varepsilon \overline{C}_A$ is an axiom

/4/ τ is admitted finite set of global functions /set of productions/.

The language generated by T-grammar is the set $L(T) = \{c \varepsilon \overline{C}_A : c_0 \xrightarrow{\tau} c\}$.

The following results in L(T)-languages can be presented [4] .

Theorem 19. There exist L(T)-languages which are not $L(\tau_n)$-languages. There exist regular languages which cannot be generated by τ_n- or T-grammars.

Theorem 20. Any finite set of words in an alphabet A can be generated by some T-grammar. For any integer m>2 there exist regular sets of words which cannot be generated by T_m-grammars, but can be generated by some T_{m+1}-grammar and τ_{m+1}-grammar, even.

Theorem 21. L(T)-languages are not closed under intersection with regular sets.

Theorem 22. L(T)-languages are closed with respect to the operation of inversion and are not closed with respect to the operations supplement and intersection.

Theorem 23. The membership, equivalence and finiteness problems and the problem of nonempty intersection for two arbitrary L(T)-languages are undecidable.

Theorem 24. Let W be a finite transformation and L be an L(T)-language. There exist W such that W(L) cannot be generated by some T-grammar. Similar result take place for sets $\tau(L)$, H(L), where τ and H are global transformation and homomorphism, respectively.

A correlation between L-systems and 1-HS can be stated [4] .

Theorem 25. Any L-system can be simulated by a 1-dimensional HS (broadly speaking not in real time), and vice versa.

Now we go over to the discussion of some possibilities of HS in parallel processing. In [4] we present some approaches in utilization of HS in parallel programming. Above all,the problem of non-contradictoriness of algorithms of parallel substitutions is discussed. This problem has the vital importance for parallel microprogramming. In this direction the following result can be proved.

Theorem 26. The problem of non-contradictoriness of algorithms of parallel substitutions

$$KF^{j}(m_i^1,\ldots,m_i^d) \rightarrow KF^{j}(m_{i_1}^1,\ldots,m_{i_1}^d) \quad [X_1^i,\ldots,X_d^i]$$

is constructively solvable.

The established correlation between modified Post-algebras and HS allow to use the HS in the applied algorithm theory [4] . The above mentioned general decomposition problem

(GDP) of global transition functions in HS is very important. The problem was solved by V.Aladyev [1] with help of nonconstructibility approaches in HS. In [4] the problem receives the further decision on the basis of other interesting approaches. In the first place, with the help of Shannon´s function we present a solution of GDP for binary general HS.

Theorem 27. The general decomposition problem for binary d-dimmensional ($d \geq 1$) global function $\tau^{(n)}$ has a negative solution.

On the basis of results in k-valued logic we have the following theorem [4].

Theorem 28. The general decomposition problem for d-dimensional ($d \geq 1$) global functions $\tau^{(n)}$ in alphabet A ($a \geq 3$) has a negative solution.

For a solution of the GDP the algebraical approach can be used.

Theorem 29. Let $L(a,d)$ be a semigroup of all d-dimensional maps $\tau^{(n)}: C_A \to C_A$. $L(a,d)$ can be presented in the form of union of subsemigroups A_i ($i=1,\ldots,4$) $\{(\forall i)(\forall j)(i \neq j \to A_i A_j = \emptyset\}$, which has no finite systems of generators and a maximum group.

The absence of the finite system of generators for subsemigroup A_4 was received on the basis of investigations of the special types of the infinite mutually erasable configurations in HS.

Theorem 30. The semigroup of all 1-dimensional maps $\tau^{(n)}$ has not a finite system of generators.

The utilization of the possibility of representation of local transition functions $L^{(n)}$ in the form of polynoms in (mod a) allow to receive a number of interesting results on the GDP in HS.

Theorem 31. For any prime number a there exist global functions which cannot be presented in the form of composition of the finite number of more simple global functions in the same alphabet A. For any integer $n \geq 2$ there exists a binary global function $\tau^{(n)}$ which has a negative solution of the GDP.

At the end of this results constructive approaches to the solution of the GDP for some classes of global functions are presented. An approach is based on the following result.

Theorem 32. A global function has a positive solution of the GDP iff the corresponding polynom P_n (mod a) can be presented in the form of superposition of polynom $P_{n_k}(P_{n_{k-1}}...(P_{n_1})...)$ (mod a) for $n_i<n$ $(i=1,...,k)$.

Since up to this point there do not exist enough general own methods of investigations of HS, along with attracting for these purposes the methods of other mathematical areas, the working out of such methods would be extremely desirable. The present survey of methods allow, in my opinion, to use some little-known methods by many investigators and to intense the working out of one´s own methods of investigations of HS. Here we shall only be content with giving a list of topics which are used for investigations of HS:

/0/ basic level / it contains one´s own methods of investigation of HS /
/1/ group, semigroup and algebra theories,
/2/ k-valued logic and Boolean algebra
/3/ structural approach
/4/ simulation
/5/ theory of the shift dynamical systems
/6/ graph-topological approach
/7/ theory of recursive functions
/8/ modelling
/9/ formal language theory
/10/ number theory
/11/ computer modelling
/12/ general system theory

I hope that this survey will help to clear up some aspects of the methods of investigation of HS as well as giving some information about modern methods to scientists working on this topic.

References

[1] Aladyev V., Mathematical Theory of Homogeneous Structures and Their Applications, Valgus Press,Tallinn,1980

[2] Parallel Processing and Parallel Algorithms.(Ed.V.Aladyev), Valgus Press,Tallinn,1981

[3] Aladyev V. et al., Mathematical Developmental Biology, Science, Moscow, 1982 (in Russian)

[4] Parallel Processing Systems (Ed. V. Aladyev), Valgus Press, Tallinn,1983

IMYCS'84

ON THE DESCRIPTION OF LANGUAGES BY GRAMMARS WITH REGULATED REWRITING

Jürgen Dassow
Technological University Magdeburg
Department of Mathematics and Physics
DDR-3040 Magdeburg, PSF 124
German Democratic Republic

Gheorghe Paun
University of Bucharest
Faculty of Mathematics
R-70109 Bucuresti
Str. Academiei 14
Romania

1. Introduction

One of the most important and well investigated subjects in formal language theory is the descriptional complexity of languages with respect to such measures as the number of variables required for the generation of the language, the index, etc. One of the early results in this direction is the fact that the regular languages form an infinite hierarchy with respect to the number of variables necessary for the generation by context-free grammars,/Gr/. The same holds for some other classes of languages and grammars with respect to the index.

In this note we summarize some results of the authors on the number of variables and productions, respectively, which is required for the generation of languages by matrix grammars, programmed grammars, and random context grammars. Especially, we give contributions to the following problems:

- Compare the complexities of the language descriptions by grammars of different types.
- Give uniform estimations of the complexity of languages in a

given class of languages.

2. Definitions and notations

We assume that the reader is familiar with basic notions and results in formal language theory especially concerning regulated rewriting (see /Ma/, /P1/, /Sa/). Here we recall only some definitions informally and specify some notations.

In all cases we use the nonterminal alphabet V_N, the terminal alphabet V_T, and the axiom $S \in V_N$.

A matrix grammar is a construct $G = (V_N, V_T, S, M, F)$ with V_N, V_T, S as above, and M and F denoting the set of matrices (sequences of context-free rules $A \to w$, $A \in V_N$, $w \in (V_N \cup V_T)^*$) and the set of occurrences of rules in M which are used in the appearance checking mode. In a step of a derivation we have sequentially to use all the rules of a matrix; if a rule appears in F, and it is not applicable to the current string, then it can be overpassed.

For any non-matrix grammar G we denote the set of rewriting rules by P.

The rules of a programmed grammar $G = (V_N, V_T, S, P)$ are of the form $(b, A \to w, E, F)$ where b is the label of this rule, $A \in V_N$, $w \in (V_N \cup V_T)^*$, E is the successfield, and F is the failure field (sets of labels). If the core production $A \to w$ is applicable then, after using it, we have to apply a rule with label in E; if $A \to w$ is not applicable, then we continue with a rule whose label belongs to the failure field F.

The rules of a random context grammar $G = (V_N, V_T, S, P)$ are of the form $(A \to w, R, Q)$ where $R, Q \subseteq V_N$ are the set of forbidden and permitting letters, respectively. The core rule $A \to w$ can be used only for the rewriting of sentential forms uAv such that uv do not contain any symbol of R and uv contains all letters of Q.

By CF, M, PR, RC we denote the classes of context-free, matrix, programmed, and random context grammars (with erasing rules), respectively,

For a class X of grammars, let $\mathfrak{L}(X)$ be the family of languages L(G) generated by grammars G of X. By $\mathfrak{L}(RE)$ we denote the family of recursively enumerable languages. It is known that

$$\mathfrak{L}(RE) = \mathfrak{L}(M) = \mathfrak{L}(PR) = \mathfrak{L}(RC) .$$

For a grammar G and a language L, we define

$$\text{Var}(G) = \text{card}(V_N),$$

$$\text{Var}_X(L) = \inf \{\text{Var}(G) : G \in X, L(G) = L\} ,$$

$$\text{Prod}(G) = \text{card}(\{A \to w : A \to w \text{ occurs in a rule/matrix of } G\}),$$

$$\text{Prod}_X(L) = \inf \{\text{Prod}(G) : G \in X, L(G) = L\} .$$

Further we put

$$\mathfrak{L}_X(n) = \{L : L \in \mathfrak{L}(X), \text{Var}_X(L) \leq n\} .$$

3. Comparison results

By definitions, we obtain

$$\text{Var}_X(L) \leq \text{Var}_{CF}(L) \quad \text{and} \quad \text{Prod}_X(L) \leq \text{Prod}_{CF}(L)$$

for $X \in \{M, PR, RC\}$ and $L \in \mathfrak{L}(CF)$. The following theorem indicates that the description by regulated context-free grammars can be as more economic as you like compared with the use of context-free grammars.

Theorem 1. (/Da/, /DP1/, /BCMW/) There are sequences of context-free languages L_n, M_n, N_n, O_n, $n \in N$, such that

i) $\text{Var}_M(L_n) \leq 2$, $\text{Var}_{CF}(L_n) = n$,

ii) $\text{Var}_{PR}(M_n) = 1$, $\text{Var}_{CF}(M_n) = n$,

iii) $\text{Var}_{RC}(N_n) \leq 8$, $\text{Var}_{CF}(N_n) = n$,

iv) $\text{Prod}_{PR}(O_n) \leq 5$, $\text{Prod}_M(O_n) \leq 10$, $\text{Prod}_{RC}(O_n) \leq c$ (where c is a constant), and $\text{Prod}_{CF}(O_n) \geq \log(n) + 1$.

The results on the comparison between matrix and programmed grammars are summarized in the following theorem.

Theorem 2.(/Da/, /DP1/, /DP2/) For each $L \in \mathfrak{L}(RE)$,

i) $Var_M(L) \leq Var_{PR}(L) + 1$, $Var_{PR}(L) \leq Var_M(L) + 2$,

ii) $Prod_M(L) \leq Prod_{PR}(L) + 5$, $Prod_{PR}(L) \leq Prod_M(L) + 1$.

Concerning the optimality of the estimations we mention

Theorem 3. (/DP2/) There are context-free languages L and K such that

i) $Var_{PR}(L) = 1$, $Var_M(L) = 2$,

ii) $Var_M(K) = 1$, $Var_{PR}(K) = 2$.

Random context grammars form a class with greater descriptional complexities than the other two regulation mechanisms as can be seen by

Theorem 4. (/Da/) i) For each $L \in \mathfrak{L}(RE)$,

$$Var_{PR}(L) \leq Var_{RC}(L) + 1, \quad Var_M(L) \leq Var_{RC}(L) + 1 .$$

ii) For each $n \in N$, there exist regular languages R_n and S_n such that

$$Var_{PR}(R_n) = 1, \quad Var_{RC}(R_n) \geq n,$$
$$Var_M(S_n) \leq 3, \quad Var_{RC}(S_n) \geq n.$$

4. Uniform estimations of language families

Theorem 5. (/P2/, /DP1/)

$$\mathfrak{L}_M(6) = \mathfrak{L}_{PR}(8) = \mathfrak{L}(RE) .$$

It is an open problem whether or not the values of Theorem 5 are optimal.

Some special families require only a fewer number of nonterminals. A context-free grammar $G = (V_N, V_T, S, P)$ is called

- linear if all productions of P are of the form

$$A \to u, \quad A \to uBv \qquad (1)$$

where $A, B \in V_N$, $u, v \in V_T^*$, and

- metalinear if all productions are of the form $S \to w$, $w \in (V_N \cup V_T)^*$, or of the form (1) and S does not occur at the right side of a production.

By LIN and MLIN we denote the families of linear and metalinear grammars, respectively.

Theorem 6. (/DP1/) i) $\mathfrak{L}(LIN) \subseteq \mathfrak{L}_M(2)$, $\mathfrak{L}(LIN) \subseteq \mathfrak{L}_{PR}(2)$,

ii) $\mathfrak{L}(MLIN) \subseteq \mathfrak{L}_M(3)$, $\mathfrak{L}(MLIN) \subseteq \mathfrak{L}_{PR}(3)$.

The optimality of these relations is shown by the following result.

Theorem 7. (/DP1/) i) There are regular languages U and V such that

$$Var_M(U) = 2 \quad \text{and} \quad Var_{PR}(V) = 2.$$

ii) There is a metalinear language W with $Var_M(W) = 3$.

By definition, each sentential form of a linear (metalinear) grammar contains at most one nonterminal (a bounded number of nonterminals). This is the characteristic property of the language family which will be defined now.

By $\#_X(w)$ we denote the number of occurrences of letters of the set X in the word w. For a context-free grammar G with the set V_N of nonterminals, a word $w \in L(G)$, a derivation

$$D : S = w_1 \Rightarrow w_2 \Rightarrow \ldots \Rightarrow w_{n-1} \Rightarrow w_n = w$$

of w, and a context-free language L we define

$$Ind(D) = \max \{\#_{V_N}(w_i) : 1 \leq i \leq n\},$$

$$Ind(w,G) = \min \{Ind(D) : D \text{ is a derivation of } w \text{ in } G\},$$

$$Ind(G) = \sup \{Ind(w,G) : w \in L(G)\},$$

$$Ind(L) = \inf \{Ind(G) : L(G) = L\}.$$

Further let

$$\mathfrak{L}_{FIN}(CF) = \{L : L \in \mathfrak{L}(CF), Ind(L) < \infty\}$$

be the family of context-free languages with finite index.

However, in order to obtain a generalization of Theorem 6 we consider matrix grammars with leftmost restriction, i.e. the productions of the matrices have to be applied to the leftmost occurrence of their left side in the current string. This class

of grammars is denoted by M_1. The class PR_1 of leftmost restric stricted programmed grammars is defined analogously. (Note that this leftmost restriction differs from that given in /Sa/ and /P1/.)

Theorem 8. (/DP2/) i) $\mathfrak{L}_{FIN}(CF) \subseteq \mathfrak{L}_{M_1}(3)$,

ii) $\mathfrak{L}_{FIN}(CF) \subseteq \mathfrak{L}_{PR_1}(3)$.

We mention that Theorem 8 can be generalized to matrix/programmed languages of finite index.

For random context grammars such results (as Theorem 5 -8) are not possible since already the regular languages form an infinite hierarchy as it can be seen by the following theorem.

Theorem 9. For each $n \geq 1$, $\mathfrak{L}_{RC}(n+1) \setminus \mathfrak{L}_{RC}(n)$ contains a regular language.

Further we note that
- all languages in $\mathfrak{L}_M(1)$ are semilinear,
- there is a non-semilinear language L with $Var_M(L)=Var_{PR}(L)=3$,
- $\mathfrak{L}_M(1)$ contains non-context-free languages.

With respect to the measure Prod we have only estimations for language families over a fixed alphabet V.

Theorem 10. (/DP2/) i) $Prod_M(L) \leq 13 + card(V)$,

ii) $Prod_{PR}(L) \leq 15 + card(V)$.

References

/BCMW/ W.Bucher, K.Culik II, H.A.Maurer, D.Wotschke, Concise description of finite languages. Bericht 32, Institut für Informationsverarbeitung, Technische Universität Graz, 1979.

/Da/ J.Dassow, Remarks on the complexity of regulated rewriting. To appear in Fundamenta Informaticae.

/DP1/ J.Dassow, Gh.Paun, Further remarks on the complexity of regulated rewriting. Submitted for publication.

/DP2/ J.Dassow, Gh.Paun, Some notes on the complexity of regulated rewriting. Submitted for publication.

/Gr/ J.Gruska, Some classifications of context-free languages. Inform. Control 14 (1969) 152-179.

/Ma/ O.Mayer, Some restrictive devices for context free grammars. Inform. Control 20 (1972) 69-92.

/P1/ Gh.Paun, Gramatici matriciale. Bucuresti, 1981.

/P2/ Gh.Paun, Six nonterminals are enough for generating all recursively enumerable languages by a matrix grammar. Submitted for publication.

/Sa/ A.Salomaa, Formal Languages. New York, 1973.

IMYCS'84

ON ω-SEQUENCES OBTAINED BY ITERATING MORPHISMS

Matti Linna
Department of Mathematics
University of Turku
Turku, Finland

1. Introduction

Since the work of Thue, [15], infinite words have been investigated from different points of view in theoretical computer science, see e.g. [1,2,6,12].

The purpose of this paper is to discuss some recent results and open problems concerning infinite words obtained by iterating morphisms, the main emphasis being on some periodicity questions.

After preliminaries in Section 2 we recall the DOL prefix problem [10] and some other related results. In Section 3 we shall first study equations of the form $h(x) = x^n$, $n = 2,3,\ldots$, where h is a given endomorphism on a finitely generated free monoid. It turns out that all the solutions of these are obtained as powers of finitely many primitive words. Then we turn to consider the DOL periodicity problem: Is there an algorithm for deciding whether the limit of a given DOL language consists of ultimately periodic infinite words? In the last section we state some further results and discuss some open problems.

2. Preliminaries

Let A be a finite alphabet and A^* the free monoid generated by A. We denote by 1 the identity (the empty word) in A^* and by A^+ the free semigroup $A^* \setminus \{1\}$. For a word $w \in A^*$, $|w|$ denotes the length of w, while $|A|$ is the cardinality of A.

A word $w \in A^*$ is <u>primitive</u> if it is not a power of another word. Every word is a power of a primitive word, denoted by $\sqrt{w}$. Given two words w and v

we say that w is a prefix of v in case $v = ww_1$ for some $w_1 \in A^*$. Also, w and v are conjugates if one finds words u_1 and u_2 such that $w = u_1u_2$ and $v = u_2u_1$.

In what follows we are interested in iterating a morphism $h: A^* \to A^*$ starting with a given word u. This iteration gives us a sequence

$$u, h(u), h^2(u), \ldots \tag{1}$$

of words. The pair (h,u) is called a DOL system in the literature and its language is the set $L(h,u) = \{h^i(u) \mid i \geq 0\}$, which may be finite, of course.

Given a morphism $h: A^* \to A^*$ we call a letter $b \in A$ finite if L(h,b) is a finite set. Otherwise b is an infinite letter.

The limit, lim L(h,u), of the set L(h,u) consists of all infinite words $\alpha = a_1a_2\ldots$, $a_i \in A$, such that for all n, α possesses a prefix longer than n belonging to (1). The adherence, adh L(h,u), of the set L(h,u) consists of all infinite words α such that for every prefix w of α, there is a word x such that wx is in L(h,u).

It is easy to verify that adh $L(h,u) \neq \phi$ if and only if the language L(h,u) is infinite. So the emptiness problem for the adherences of DOL languages is decidable. The same holds true also for the limits as was shown in [10].

Theorem 1. The emptiness problem for the limits of DOL languages is decidable.

The proof presented in [10] is mainly based on the so called defect theorem, see e.g. [12].

We note also that if $\lim L(h,u) \neq \phi$ then one can effectively find integers p and q such that $h^p(u)$ is a proper prefix of $h^{p+q}(u)$. In this case

$$\lim L(h,u) = \bigcup_{i=0}^{q-1} \lim L(h^q, h^{p+i}(u)),$$

where moreover $|\lim L(h^q, h^{p+i}(u))| = 1$ for each $i = 0,1,\ldots,q-1$. We can thus separate the case (1) in an effective way into a finite number of special cases where the limit of the sequence exists uniquely.

We finally mention two recent generalizations of the ordinary DOL sequence equivalence result. The first is obtained by Culik II and Harju [4] and the second by Head [9].

Theorem 2. There is an algorithm for deciding whether or not two given DOL systems generate the same limit.

Theorem 3. There is an algorithm for deciding whether or not two given DOL systems generate the same adherence.

3. On the periodicity

As discussed in the preceding section, we can restrict ourselves to DOL systems (h,u), where (if the limit exists) $h(u) = ux$ for some $x \in A^*$. This kind of a system defines the infinite word

$$h^\omega(u) = uxh(x)h^2(x)\ldots .$$

Besides Theorem 1, one of the crucial questions concerning infinite words obtained by iterating morphisms is the following. Is it decidable whether or not a given prefix preserving morphism h defines an ultimately periodic infinite word, that is, whether or not

$$h^\omega(u) = vw^\omega$$

for some words v and w? Here w^ω denotes the infinite word $ww\ldots$. Some special cases of the problem were solved in [9] and [11]. In [9] a partial solution to the problem was used to solve the adherence equivalence problem for DOL systems (Theorem 3).

The ultimate periodicity problem, shown to be decidable in [8], comes into use also in solving the ω-regularity problem for the limits of DOL languages. The ordinary regularity problem for DOL languages was shown to be decidable in [14]. The corresponding problem for infinite words is just another formulation for the DOL periodicity problem. In the following we shall present the main ideas of the solution.

First we shall consider the equations

$$h(x) = x^n, \quad n = 2,3,\ldots, \tag{2}$$

where $h: A^* \to A^*$ is a given morphism. It turns out that all the solutions of (2) can be effectively found.

Given a solution, $h(w) = w^n$ for some $n \geq 2$, we note that $(\sqrt{w})^p$ is also a solution for all $p \geq 0$. Thus we need to search for the primitive solutions only. With this in mind we define

$$P_h = \{w \in A^+ \mid w \text{ primitive and } h(w) = w^n \text{ for some } n \geq 2\}.$$

Let $A = A_F \cup A_I$, where A_F is the set of finite letters and A_I the set of

infinite letters with respect to h. One can prove

Theorem 4. For a given $h: A^* \to A^*$ there are only finitely many primitive words w for which $h(w) = w^n$ for some $n \geq 2$. In fact there is a partition $A_1,\ldots,A_r$ of A_I such that

$$P_h \subseteq \bigcup_{i=1}^{r} (A_F \cup A_I)^*$$

and the words in $P_h \cap (A_F \cup A_i)^*$ are conjugates $(i = 1,\ldots,r)$.

Given two words v_1 and v_2 it is decidable whether or not $h^i(v_1) = h^i(v_2)$ for some integer i, cf. [3] or [5]. Using this result together with Theorem 4 one can prove

Theorem 5. The set P_h can be constructed effectively for a given morphism $h: A^* \to A^*$.

The following decidability result is an immediate consequence.

Theorem 6. It is decidable whether or not the equations $h(x) = x^n$, $n = 2,3,\ldots,$ possess a nontrivial solution.

This result can also be given in a somewhat stronger form.

Theorem 7. For a given h it is decidable whether or not there exists a nontrivial word x such that $h^m(x) = x^n$ for some $m \geq 1$ and $n \geq 2$.

Now using Theorems 1 and 5 one obtains

Theorem 8. The ultimate periodicity problem is decidable for DOL systems.

Proof. Let us be given a morphism $h: A^* \to A^*$ and a word $u \in A^*$ such that $h(u) = ux$ for some $x \in A^+$. Denote by A_1 the subset of A which consists of the infinite letters occurring infinitely many times in $h^\omega(u)$. Clearly A_1 is an effective set.

In case $A_1 = \phi$ there appears only one infinite letter b which is isolated, that is, no letter produces b. This case is thus easy, since the period comes out from $h(b) = u_0 bv$.

Assume now that $A_1 \neq \phi$ and let b be the first letter of $h^\omega(u)$ from A_1. Then for some $i \leq |A|$ and $y \in A^*$

$$h^i(yb) = yby_1 \text{ and } h^{|A|}(y) = 1.$$

For otherwise $h^\omega(u)$ is not ultimately periodic. We may assume that $i = 1$

since otherwise we consider the pair (h^i,u) instead of (h,u). Thus

$$h(yb) = yby_1 \text{ and } h^{|A|}(y) = 1.$$

Let us write now

$$h^\omega(u) = u_1 ybu_2u_3\cdots,$$

where $h(u_1) = u_1 ybu_2$ and $u_3 \in A^*A_1A^*$.

By Theorem 5 we may test whether there exists a primitive w such that yb is a prefix of w and $h(w) = w^n$ for some $n \geq 2$. If no such w can be found then $h^\omega(u)$ is not ultimately periodic by above. Assume then that we have found such a word w.

Claim. $h^\omega(u)$ is ultimately periodic iff $h^\omega(ybu_2u_3) = w^\omega$ iff (h,ybu_2u_3) defines an infinite word.

Proof of the claim. Assume $h^\omega(ybu_2u_3)$ is defined. Then $h^\omega(ybu_2u_3) = h^\omega(yb) = w^\omega$ (since $b \in A_1$ and yb is a prefix of w). We have also for all $i \geq 1$

$$h^i(ybu_2u_3) = u_1\cdot ybu_2\cdot\ldots\cdot h^i(ybu_2)\cdot h^i(u_3)\cdot$$

and so $h^i(u_3)$ is a prefix of $h^{i+1}(ybu_2u_3)$. Suppose i is here already so large that $|h^i(u_3)| > |w|$ and $h^j(ybu_2u_3)$ is a prefix of w^ω for all $j \geq i$. Then also w is a prefix of $h^i(u_3)$ and hence $h^i(ybu_2) \in w^*$. Now $h^\omega(u) = h^\omega(u_1ybu_2u_3)$ implies that $h^\omega(u)$ is ultimately periodic. The converse of the claim is trivial.

We note that the last statement is decidable in the claim and so is the first one. This completes the proof of the theorem.

Remark. Pansiot [13] has recently given another quite different proof (based on simplifiable morphisms) to the above theorem.

4. Discussion

Theorems 4 and 5 or their proofs in [8] do not give the primitive solutions w explicitely. In the binary case, $|A| = 2$, one can, however, obtain a very effective characterization to the set P_h as well as to the morphism h, [7]:

Theorem 9. Let w be a primitive word in $\{a,b\}^*$ and let h be an endomorphism on $\{a,b\}^*$. Then $h(w) = w^n$ for some $n \geq 2$ iff at least one of the letters, say a,

is infinite and

(i) $w = \sqrt{h(a)}$ and either

1^{o}. $h(b) = 1$ and, $|w|_a \geq 2$ or $h(a)$ is not primitive;

or

2^{o}. $w = \sqrt{h(b)}$;

or

3^{o}. $h(a) \in a^2a^*$;

or

(ii) $w = b^{s_1}ab^{s_2}$ and $h(b) = b$, $h(a) \in (ab^{s_1+s_2})^+a$;

or

(iii) $w = ab$ and $h(a) \in (ab)^+a$, $h(b) \in b(ab)^+$.

We note that in above all the primitive solutions w are of length at most $\max\{|h(a)|,|h(b)|\}$. It is a matter for remark that this is not so in alphabets of larger size. We mention just a simple example: $h(a) = ab$, $h(b) = ca$, $h(c) = bc$. Here $h(abc) = (abc)^2$.

The finiteness result in Theorem 4 is characteristic to free semigroups. Namely, this property fails already in <u>one-relator semigroups</u>. As an example we mention the free commutative semigroup $\langle a,b;\ ab = ba\rangle$, where the morphism h defined as $h(a) = a^2$, $h(b) = b^2$ possesses infinitely many 'primitive' solutions of the form a^ib.

Also it is worthwile noting that this failure concerns <u>free groups</u> as well. To see this consider a word w in a finitely generated free group F such that $h(w) = w^n$ for an endomorphism h on F. Now, let G be a disjoint finitely generated free group and define h to be the identity on G We have $h(uwu^{-1}) = uw^nu^{-1} = (uwu^{-1})^n$ for all u in G and so h has infinitely many solutions of the form uwu^{-1} in the free product G*F.

We still mention one open problem. Is it decidable, for given morphisms $h,g: A^* \to A^*$ and given words $u,v \in A^*$, whether there exists a word $w \in A^*$ such that $h^{\omega}(u) = wg^{\omega}(v)$? In the special case h = g the problem reduces to the ultimate periodicity problem for DOL systems which is decidable by Theorem 8. We mention also that the positive solution of the general problem would easily imply the solution of the limit equivalence problem for DOL systems (Theorem 2).

Acknowledgements: I like to thank Tero Harju for helpful comments.

References

[1] Berstel, J.: Some recent results on squarefree words, Proceedings of a Symposium of Theoretical Aspects of Computer Science, Paris 1984, Springer Lecture Notes in Computer Science 166 (1984), 14-25.

[2] Boasson, L. and Nivat, M.: Adherences of languages, J. Comput. Syst. Sci. 20 (1980), 285-309.

[3] Culik, K., II: The ultimate equivalence problem for DOL systems, Acta Informatica 10 (1978), 79-84.

[4] Culik, K., II and Harju, T.: The ω-sequence equivalence problem for DOL systems is decidable, J. Assoc. Comput. Mach. 31 (1984).

[5] Ehrenfeucht, A. and Rozenberg, G.: On simplifications of PDOL systems, Proceedings of a Conference on Theoretical Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 1977.

[6] Eilenberg, S.: Automata, Languages and Machines, Vol. A, Academic Press 1974.

[7] Harju, T. and Linna, M.: The equations $h(w) = w^n$ in binary alphabets, submitted for publication, 1984.

[8] Harju, T. and Linna, M.: On the periodicity of morphisms on free monoids, submitted for publication, 1984.

[9] Head, T.: Adherence equivalence is decidable for DOL languages, Proceedings of a Symposium of Theoretical Aspects of Computer Science, Paris 1984, Springer Lecture Notes in Computer Science 166 (1984), 241-248.

[10] Linna, M.: The decidability of the DOL prefix problem, Intern. J. Comput. Math. 6 (1977), 127-142.

[11] Linna, M.: On periodic ω-sequences obtained by iterating morphisms, Ann. Univ. Turkuensis, Ser. A I 186 (1984),to appear.

[12] Lothaire, M.: Combinatorics on Words, Addison-Wesley 1983.

[13] Pansiot, J.: Decidability of periodicity for infinite words, a manuscript, 1984.

[14] Salomaa, A.: Comparative decision problems between sequential and parallel rewriting, Proc. Symp. Uniformly Structured Automata and Logic, Tokyo (1975), 62-66.

[15] Thue, A.: Uber unendliche Zeichenreihen, Videnskapsselsk. Skrifter I. Kristiania (1906), 1-22.

IMYCS'84

Understanding the Bugs and Misconceptions of Novice Programmers: An Overview

Elliot Soloway

Department of Computer Science
Yale University
New Haven, Connecticut 06520 U.S.A.

1. Introduction: Motivation and Goals[1]

In this document we provide an overview of the research being conducted at the Cognition and Programming Project in the Computer Science Department at Yale University that deals with understanding novice programming. Since there are a number of available reports that examine our research efforts in some detail, this report will present only highlights to that body of research, and pointers to particular papers. In particular, I will focus on 3 topics that we view as key in the process of understanding novice programming:

- The knowledge base underlying programming.
- The types of bugs and misconceptions that novice programmers exhibit.
- The relationship of programming language constructs to the cognitive strategies of programmers.

The thread that ties the above topics together is that of being sensitive to the underlying knowledge representation and processing strategies that novices (and experts) use when they program. Clearly, the above topics do not exhaust the area of novice progarmming; however, we believe that our research efforts have been able to tap into some of the important cognitive aspects of programming.

2. Beyond Syntax and Semantics: Plan Knowledge and Discourse Rules

While knowledge of the syntax and semantics of a programming language is important, this knowledge alone will not permit one to effectively read/write computer programs. We have identified two types of *deep-structure knowledge* that expert programmers seem to have and use:

- *programming plans* which represent the stereotypic action sequences in programming. (Also see [11, 2].)
- *rules of programming discourse* which guide the composition of plans into understandable and executable programs.

We have carried out a number of empirical studies with novice and advanced programmers to

[1]This work was sponsored by the National Science Foundation, under NSF Grant MCS-8302382.

evaluate the above claims; data do in fact support these claims [14, 5, 15, 16, 17].

2.1. Programming Plans

```
Problem:  Read in numbers, taking their sum, until the number 99999 is seen.
Report the average.  Do not include the final 99999 in the average.


                    PROGRAM OrangeAlpha;
                    VAR Sum, Count, Num : INTEGER;
                        Average : REAL;
Counter Variable    BEGIN
Plan          ----------> Count := 0;
              |     ---> Sum := 0;                  Running Total Loop Plan
              |     |    Read(Num); <-------------------
Running Total |     |    WHILE Num <> 99999 DO <-------|
Variable Plan |     |      BEGIN                       |
              |     -------> Sum := Sum + Num; <-------|
              -------------> Count := Count + 1;       |
                             Read(Num);       <--------
                           END                          Skip Guard Plan
                         IF Count > 0 THEN  <---------------------------
                                       BEGIN <-------------------------|
                                        Average := Sum/Count; <--------|
                                        Writeln( Average); <-----------|
                                       END <--------------------------|
                                     ELSE  <--------------------------|
                                       Writeln( 'no legal inputs'); <-|
                    END.
```

Figure 1: Examples of Programming Plans

We can identify two types of programming plans in the program in Figure 1: control flow plans and variable plans.[2] For example, the RUNNING TOTAL LOOP PLAN and the SKIP GUARD PLAN are two control flow plans in this program. The former plan repeatedly reads in some values and accumulates their total. The latter plan is also a common one: it protects the average computation from an illegal division by 0. The RUNNING TOTAL VARIABLE PLAN and the COUNTER VARIABLE PLAN are two common variable plans used in programming. Notice that in these plans, the variable's initialization is explicitly tied to its update; while textbooks don't emphasize this point, we have found that programmers use this relationship in understanding programs ([5]). In Figure 2 we depict a network of programming plans that captures the tacit

[2]Variable plans are related to, but are richer than, the computer science notion of *abstract data types*, in that plans have more properties (e.g., relatedness, goal) than are usually associated with abstract data types.

knowledge underlying a portion of introductory programming [14].

There is already substantial empirical evidence that provides support *for* the claim that programmers have and use programming plans. Shneiderman [13], Adelson [1], and McKeithen et al. [9] showed that expert programmers had better recall of meaningful programs than did novice programmers, but that both groups performed about the same on nonsense programs (randomly composed lines of code). We have built on this work by identifying specific pieces of programming plan knowledge (e.g., Figure 2). Moreover, in empirical studies we carried out, we have gathered supportive evidence for the existence of these specific knowledge units [15, 14, 5].

2.2. Rules of Programming Discourse

Rules of programming discourse specify the conventions in programming, e.g., *the name of a variable should agree with its function*; these rules set up expectations in the minds of the programmers about what should be in the program. They are analogous to discourse rules in conversation. Under our theory, programs are composed from programming plans that have been modified to fit the needs of the specific problem. The composition of those plans is governed by rules of programming discourse. Thus, a program can be correct from the perspective of the problem, but be difficult to write and/or read because it doesn't follow the rules of discourse, i.e., the plans in the program are composed in ways that violate some discourse rule(s).

In Figure 3 we depict a set of programming discourse rules that we have identified. Individually, they look innocuous enough, and one would hardly disagree with them. While these rules typically are not written down nor taught explicitly (except, for example in [7]), we claim that programmers have and use these rules in the construction and comprehension of programs. Moreover, if programmers do use these rules and expect other programmers to also use these rules, then we predict that programs that violate these rules should be harder to understand than programs that do not. Data we have recently collected from a number of empirical studies bear out this prediction [17, 16].

We call programs that are consistent with the discourse rules *plan-like programs*, and ones that are not *unplan-like* programs. For example, in Figure 4, Version Alpha is the plan-like version of a program that finds the maximum of some numbers. In our plan jargon, it uses the MAXIMUM SEARCH LOOP PLAN which in turns uses a RESULT VARIABLE PLAN. Notice that the RESULT VARIABLE is appropriately named `Max`, i.e., the name of the variable is consistent with the plan's function. In contrast, Version Beta is unplan-like since it uses a MINIMUM SEARCH LOOP PLAN in which the RESULT VARIABLE is inconsistent with the plan's function: the program computes

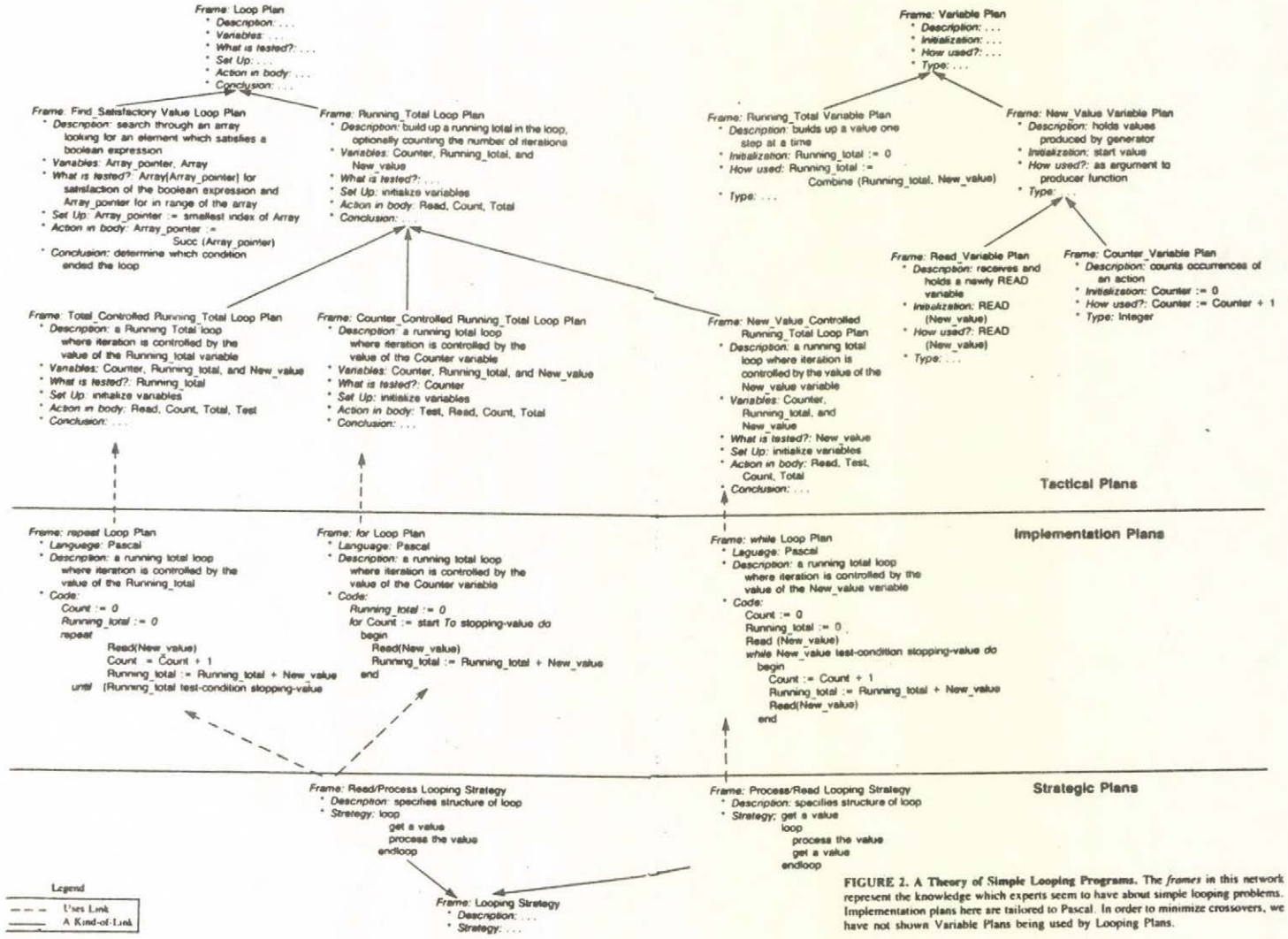


FIGURE 2. A Theory of Simple Looping Programs. The *frames* in this network represent the knowledge which experts seem to have about simple looping problems. Implementation plans here are tailored to Pascal. In order to minimize crossovers, we have not shown Variable Plans being used by Looping Plans.

the minimum of some numbers using a variable named Max. To create the Beta version, we violated the first rule of programming discourse in Figure 3: *Variable names should reflect function.*

```
(1)  The name of variable should reflect its function in the program.

(2)  Don't include code that won't be used.

     (2a) If there is a test for a condition, then the condition must have the
          potential of being true.

(3)  A variable that is initialized via an assignment statement
     should be updated via an assignment statement.

(4)  Don't do double duty with code in a  non-obvious way.

(5)  An  IF  should  be  used  when a statement body is guaranteed to
     be executed only once, and a WHILE used when a  statement  body  may
     need to be repeatedly executed.
```

Figure 3: Examples: Rules of Programming Discourse

3. Understanding the Bugs and Misconceptions of Novice Programmers

Bug and misconceptions, to say the least, abound.[3] In order to more systematically explore the large space of programming bugs and misconceptions, we have developed a categorization for types of bugs and types of misconceptions. In what follows we first describe the categories themselves; we then demonstrate the explanatory power of our theory of programming knowledge, and bug and misconception categories by examining actual buggy novice programs.

3.1. Types of Bugs

We are exploring the bugs and misconceptions that novice programmers have as they try to map their understanding of the problem into a programming language. From this broad category of bugs we have isolated the following two subcategories:

- *Bug Type 1:* bugs that are the result of a discrepancy between the stated problem and its program realization,
- *Bug Type 2:* bugs that don't interfere with the running of the program and appear from the program's output to give correct results, but that nonetheless indicate a misconception on the student's part.

[3]By way of terminology: a *bug* is an error in a computer program, while a *misconception* is some conceptualization in the student's mind that can lead to a program bug.

Version Alpha

```
PROGRAM Magenta(input, output);
VAR Max, I, Num : INTEGER;
BEGIN
    Max := 0;
    FOR I := 1 TO 10 DO
        BEGIN
          READLN(Num);
          If Num  >  Max THEN Max := Num
        END;
    WRITELN(Max);
END.
```

Version Beta

```
PROGRAM Purple(input, output);
VAR Max, I, Num : INTEGER;
BEGIN
    Max := 999999;
    FOR I := 1 TO 10 DO
        BEGIN
          READLN(Num);
          If Num  <   Max THEN Max := Num
        END;
    WRITELN(Max);
END.
```

Basic plan:	search plan (max, min)
Discourse rule:	A variable's name should reflect its function (1) (Figure 3)
Alpha case:	variable name agrees with search function
To construct Beta version:	violate discourse rule (1)
Beta case:	variable name does NOT agree with search function

Figure 4: Discourse Rules: Plan-like and Unplan-like Programs

3.2. Types of Misconceptions

We can also identify different types of misconceptions that manifest themselves as program bugs:

1. *A clash between what students bring to programming and what students are supposed to learn in programming.* Oftentimes a student's preprogramming knowledge will override the student's budding computer knowledge resulting in a bug.
2. *Faulty/incomplete understanding of programming concepts.* This is a general category of misconceptions that can be further broken down:
    a. *Overgeneralization:* Students have difficult in discerning the specific context in which a concept is appropriate. This sometimes results in the overgeneralization of a concept.
    b. *Hazy understanding of a concept.* Programming knowledge is cumulative: one

builds on what one knows. Students often have a hazy notion of a concept that does not manifest itself in the simple programs, but rather comes to light only when more advanced topics are introduced.

c. *Simply not knowing the rules of programming discourse.* In conversation (and text) there are tacit rules of communication that must be obeyed in order that understanding be facilitated. These rules are built up with experience. Since, by definition, novices do not have much programming experience, they typically have not acquired these rules of programming discourse. The problem is compounded since the rules are usually not taught explicitly. One goal of our research is to identify these otherwise tacit rules.

3. *Difficulties arising from the coordination of multiple constructs.* That is, a student may well understand each individual construct, however, since this knowledge may not be as routinized as other knowledge, the complexity that results from having to coordinate many new pieces of knowledge can result in program bugs.
4. *Students may decompose the problem differently from that which was intended.* The result is usually a buggy program.

3.3. Example: Bug Type 1 & Misconception Type 2.a: Intention/Realization Difference

The program in Figure 5 is an attempt to solve the problem also given in Figure 5. The programmer intended to write a program that reads in a sequence of numbers and takes their average. However, the programmer has written a program that reads in a number and then computes the average of all the numbers between it and 99999, in integer increments. Thus the bug in this program is of type 1: there is a discrepancy between the intended goal of the problem and its realization.

Averaging Problem: Read in numbers, taking their sum, until the number 99999 is seen. Report the average. Do not include the final 99999 in the average.

```
 1    PROGRAM Average( input, output );
 2    VAR Sum, Count, New, Avg: REAL;
 3    BEGIN
 4      Sum := 0;
 5      Count := 0;
 6      Read( New );
 7      WHILE New<>99999 DO
 8        BEGIN
 9          Sum := Sum+New;
10          Count := Count+1;
11          New := New+1
12        END;
13      Avg := Sum/Count;
14      Writeln( 'The average is ', avg );
15   END;
```

Figure 5: Example: A Buggy Program

One might be, as we were, baffled by this bug: what would prompt someone to write such an unusual program? However, by appealing (1) to our theory of programming knowledge and (2) to the sources of misconceptions, we can provide a cogent analysis. We would argue that the misconception that the student was most likely laboring under is an overgeneralization of the COUNTER VARIABLE PLAN: if incrementing Count gets the next INTEGER value, then incrementing New should get the next input value. From within the context of our theory, then, this bug becomes less surprising: the COUNTER VARIABLE PLAN and the READ VARIABLE PLAN are both types of (i.e., members of the same class) NEW VALUE VARIABLE PLANS (Figure 2), since both of the former plans result in the same function, i.e., the production of the next value.

3.4. Example: Bug Type 2 & Misconception Type 1: "But My Program Runs..."

Consider the program in Figure 6; it too is an attempt to solve the problem in Figure 5. While this program runs, and produces a correct result, the "fractured" running total update indicates that the student may not really understand the assignment statement: the use of the variable Y to hold an intermediate result is unnecessary. Note we are *not* making an efficiency argument; the fact that an extra variable is used, and thus may be more costly computationally, is not the issue. Rather, we suggest that this misconception is a result of a clash between well routinized pre-programming knowledge and nascent programming knowledge: in algebra, students know full well that one doesn't have the same variable on both sides of an equation.

```
program Student21_Problem3;
    var C, X, Y, Z : integer;
    begin
    C := 0;
    Z := 0;
    while X < 99999 do
        begin
        Read (X);
        C := C + 1;
        Y := X + Z;
        Z := Y
        end;
    A := Y div C;
    Writeln (A, ' Average')
    end.
```

The above program contains a fractured RUNNING TOTAL assignment statement in the body of the while loop.

Figure 6: A Fractured RUNNING TOTAL Update

The fact that the student *did* form the counter update correctly might appear to be evidence against our analysis. However, we would argue that a counter update is viewed as an indivisible whole, not as an assignment statement with two variables and a constant. In fact, we believe that novices perceive the RUNNING TOTAL VARIABLE as being different from the COUNTER VARIABLE because the models underlying these two concepts are quite different. That is, calculating a running total is not a common technique: when one wants a sum, one writes all the numbers down and then adds them by columns. This *pre*-programming method does not fit well with the programming technique. In contrast, counters in everyday experience go up by 1, and thus the realization of counters matches quite well with the programming method (e.g., `count := count + 1`). As support for the claim that novices do perceive these two types of variables as different, we cite three pieces of evidence:

- In [14] we report on a study in which novices were asked to write a program to solve the Averaging Problem (Figure 5); 100% of the novices wrote a correct COUNTER VARIABLE update, while only 83% wrote a correct RUNNING TOTAL VARIABLE update (this difference is statistically significant at the .05 level).
- In looking more closely at the programs gathered in the study cited above, we found a number of programs similar to the one in Figure 6. If students perceived the COUNTER VARIABLE update to be of the same sort as the RUNNING TOTAL VARIABLE update, then we would expect to see "fractured updates" used in both the COUNTER VARIABLE update and the RUNNING TOTAL VARIABLE update. Fractured updates were observed in 7 programs from the novice group. However, in *all* instances this occurred to a RUNNING TOTAL VARIABLE update, not a COUNTER VARIABLE update.
- In Figure 7 we present a fragment of a verbal protocol taken by a colleague, J. Bonar [3]in which a student openly discusses the difference in perception he has of the RUNNING TOTAL VARIABLE and the COUNTER VARIABLE.

Students resist being told about bugs of this sort. They typically respond by saying that their program works, so there really is no problem. They apparently view programs as yes/no answers rather than as "English compositions". However, programs with type 2 bugs indicate some form of misconception on the student's part and need to be remedied at some point. Data we have collected clearly shows that while students may initially get by with a hazy notion, say of a variable, when arrays and subroutines (with parameter passing) are introduced, they will certainly have trouble.

Interestingly enough, the computer itself helps support the student's claims that bugs of this type are not harmful. That is, when the student uses incorrect syntax or when the algorithm itself is incorrect, the student gets immediate feedback: the program doesn't run at all, or it produces spurious results. This type of interaction is one of the positive features of computing (e.g., [10]). However, students come to believe that an *absence* of feedback means that all is fine with their

Problem : Write a program which reads in 10 integers and prints the average of those integers.

After working on the problem for a few minutes, the subject had written the following:

```
Repeat
(1) Read a number (Num)
        (1a) Count := Count + 1
(2) Add the number to Sum
        (2a) Sum := Sum + Num
(3) until Count := 10
(4) Average := Sum div Num
(5) writeln ('average = ', Average)
```

Below we give a portion of the verbal interview of the student who wrote the above code.

Interviewer:
Steps 1a and 2a: are those the same kinds of statements?

Subject:
How's that, are they the same *kind*. Ahhh, ummm, not exactly, because with this [1a] you are adding - you initialize it at zero and you're adding one to it [points to the right side of 1a], which is just a constant kind of thing.

Interviewer:
Yes

Subject:
[points to 2a] Sum, initialized to, uhh Sum to Sum plus Num, ahh - thats [points to left side of 2a] storing two values in one, two variables [points to Sum and Num on the right side of 2a]. Thats [now points to 1a] a counter, thats what keeps the whole loop under control. Whereas this thing [points to 2a], this was probably the most interesting thing ... about Pascal when I hit it. That you could have the same, you sorta have the same thing here [points to 1a], it was interesting that you could have, you could save space by having the Sum re-storing information on the left with two different things there [points to right side of 2a], so I didn't need to have two. No, they're different to me.

Interviewer:
So -- in summary, how do you think of 1a ?

Subject:
I think of this [point to 1a] as just a constant, something that keeps the loop under control. And this [points to 2a] has something to do with something that you are gonna, that stores more kinds of information that you are going to take out of the loop with you.

Figure 7: Example: Verbal Protocol

program: if the *computer* doesn't complain then the program must be correct. Since, as we said above, these seemingly harmless bugs can cause significant problems later, we feel it important to identify bugs of this type and the misconceptions that cause them.[4]

4. Cognitively Appropriate Progamming Language Constructs

In the above we have tried to lay out some of the knowledge and processing strategies that novices and experts employ. However, for the most part, these sorts of observations have not been used in the design of programming languages. This is unfortunate, since a unnecssary barrier to learning has been erected if a language construct is "not matched" with the cognitive strategy that underlies it. In what follows we will identify one such language construct --- Pascal's `while` loop --- in which there is a clear mismatch between how people prefer to solve problems and how the language construct forces them to solve problems. The upshot, which is not surprising, is that people's performance using this construct is unacceptably poor. (An expanded version of this work appears in [15].)

Consider then, the problem and its program solution given in Figure 8A. Notice that the problem is neither esoteric nor tricky; certainly one would expect novice programmers to be able to write a correct solution to this problem. However, we have found that performance on this problem is generally quite poor; in one study where we asked students to write a program, at their desks, for this problem, we found only 40% could write correct programs.[5] We have argued that a major stumbling block is the unusual model of looping required by Pascal's `while` construct. Stepping back from the code, the strategy that this program embodies can be characterized as:

```
Read (first value)
WHILE Test (i'th value)
    DO BEGIN
        Process (i'th value)
        Read (i+1st value)
    END
```

Since the loop may not be executed if the first value read is 99999, a `Read` outside the loop is necessary in order to get the loop started. However, this results in the loop processing being one step behind the `Read`; on the ith pass through the loop, the ith value is processed and *then* the ith + 1 value is read in. We call this strategy "process i/read next-i" (henceforth referred to as PROCESS/READ). In effect, processing in the loop would be "out of sync" with reading in the

[4]Because we see that bugs of this sort often "fall through the cracks", PROUST, our program that finds non-syntactic bugs in novice programs, has been explicitly designed to provide comments to the student on just such bugs [6].

[5]We did not count off for incorrect syntax.

loop.

In contrast, consider the program displayed in Figure 8B that solves the Averaging Problem using a variant of Ada's `loop...exit` construct. The strategy underlying the use of the `loop...leave...again` construct can be abstracted as follows:

```
LOOP
    DO BEGIN
        Read (i'th value)
        IF Test (i'th value) LEAVE
        Process (i'th value)
    AGAIN
```

That is, on the ith pass through the loop read the ith value and process it; we call this the "read i/process i" strategy (henceforth referred to as READ/PROCESS). We argue that the difficulty of this strategy arises from the extra burden that it places on memory and processing resources, in comparison to that placed by a READ/PROCESS strategy.

In a study reported in [15] we tested the following hypothesis:

> *People will write correct programs more often when the language facilitates their preferred strategy.*

Subjects were novice, intermediate and advanced programmers. In the first part of the study, we asked subjects to write a *plan* --- not a program --- to solve the Averaging Problem. We found that when programmers were not constrained by a particular programming language they overwhelmingly used a READ/PROCESS strategy in their plans. In the second part of the study, half the subjects were asked to write the program using standard Pascal, the language that they already knew; the other half were asked to write the same program using Pascal-L, which is Pascal with only the `loop...leave...again` construct. Both groups were given a one page description describing how the language's looping construct worked. We found that programmers using the `loop...leave...again` construct wrote correct programs significantly more often than those using the `while` construct -- the increase in performance was approximately 20%. Note that the programmers had never seen or used the `loop...leave...again` construct before, whereas they had been using the `while` construct for up to 5 semesters.

Strong claims have been made against a construct that permits an exit from the middle of the loop; it is argued that one should exit a loop from the top or the bottom, not the middle [18, 4, 8]. It is further claimed that the readability of a program is hampered if exits from the middle of the loop are allowed. Our study did not examine the readability claim, since we looked only at program generation. However, a series of studies by Sheppard et al. [12] suggest that in fact a construct that permits an exit in the middle does not interfere with readability. Thus, there appears to be empirical evidence that an exit from the middle of the loop is not as harmful as was

The Averaging Problem: Write a program that repeatedly reads in integers, until it reads the integer 99999. After seeing 99999, it should print out the **correct** average. That is, it should not count the final 99999.

{A} A Stylistically Correct Pascal Solution to Averaging Problem

```
PROGRAM STUDENT6 PROBLEM3;
       VAR COUNT, SUM, NUMBER : INTEGER; AVERAGE : REAL;
       BEGIN
         COUNT := 0;
         SUM := 0;
         READ (NUMBER);
         WHILE NUMBER <> 99999 DO
                 BEGIN
                   SUM := SUM + NUMBER;
                   COUNT := COUNT + 1;
                   READ (NUMBER)
                 END;
         IF COUNT > 0 THEN
                 BEGIN
                   AVERAGE := SUM / COUNT;
                   WRITELN (AVERAGE);
                 END
           ELSE WRITELN ('NO NUMBERS INPUT - AVERAGE UNDEFINED');
       END
```

{B} The Averaging Problem using Pascal-L

```
PROGRAM PASCAL-L;
   VAR COUNT, SUM, NEWVALUE : INTEGER;
         AVERAGE : REAL;
   BEGIN
     COUNT := 0;
     SUM := 0;
     LOOP
        READ (NEWVALUE);
        IF NEWVALUE = 99999 THEN LEAVE;
        SUM := SUM + NEWVALUE;
        COUNT := COUNT + 1;
     AGAIN
     IF COUNT > 0 THEN
        BEGIN
          AVERAGE := SUM / COUNT;
          WRITELN (AVERAGE);
        END
        ELSE WRITELN ('NO NUMBERS INPUT - AVERAGE UNDEFINED');
   END
```

Figure 8: Pascal and Pascal-L Solutions

conjectured. Finally, we hope that this sort of study indicates how one might proceed to analyze the relationship between language constructs and cognitive strategies.

5. Concluding Remarks

In the above, we have described several aspects of our research into novice programming. As we have mentioned several times, the key to our insights has been the sensitivity to the programmer --- to the novice and to the expert: we have not focused solely on external factors, e.g., on a debate between Pascal or BASIC, because that puts the problem outside the programmer. If one is to really make some headway on that particular debate, for example, one needs to argue about how and why programmers use the constructs in the two languages --- a debate divorced from the programmer just is not sufficiently convincing. Thus, more than the details we tried to get across, we hope to have conveyed a sense a how powerful the cognitive approach is to gaining insight into the workings of the novice programmer.

ACKNOWLEDGEMENT

A number of people have made significant contributions to the research reported here: Kate Ehrlich, John Black, Jeff Bonar, Lewis Johnson, Saj-Nicole Joni, Valerie Abbott, Beth Adelson, David Littman, Eric Gold, Ben Cutler, and Robert Goldman.

References

1. Adelson, B. "Problem Solving and the Development of Abstract Categories in Programming Languages." *Memory and Cognition 9* (1981), 422-433.

2. Barstow, David. *Knowledge-Based Program Construction*. Elsevier North Holland Inc., 1979.

3. Bonar, J. and Soloway, E. Uncovering Principles of Novice Programming. SIGPLAN-SIGACT Tenth Symposium on the Principles of Programming Languages, in press.

4. Dijkstra, E. W. Notes on Structured Programming. In *Structured Programming*, O. J. Dahl, E. W. Dijkstra, C. A. R. Hoare, (Eds.), Academic Press, New York, N.Y., 1972.

5. Ehrlich, K., Soloway, E. An Empirical Investigation of the Tacit Plan Knowledge in Programming. in Human Factors in Computer Systems , J. Thomas and M.L. Schneider (Eds.), Ablex Inc., in press.

6. Johnson, W. L., Soloway, E. PROUST: Knowledge-Based Program Understanding. Tech. Rept. 295, Dept. of Computer Science, Yale University, 1983.

7. Kernighan, B., Plauger, P.. *The Elements of Style*. McGraw Hill Co., New York, 1978.

8. Ledgard, H.F., Marcotty, M. "A Genealogy of Control Structures." *Communications of the ACM 18* (1975), 629-638.

9. McKeithen, K.B., Reitman, J.S., Rueter, H.H., Hirtle, S.C. "Knowledge Organization and Skill Differences in Computer Programmers." *Cognitive Psychology 13* (1981), 307-325.

10. Papert, S.. *Mindstorms, Children, Computers and Powerful Ideas*. Basic Books, 1980.

11. Rich, C. Inspection Methods in Programming. Tech. Rept. AI-TR-604, MIT AI Lab, 1981.

12. Sheppard, S.B., Curtis, B., Milliman, P. and Love, T. "Modern Coding Practices and Programmer Performance." *Computer December* (1979), 41-49.

13. Shneiderman, B. "Exploratory Experiments in Programmer Behavior." *International Journal of Computer and Information Sciences 5,2* (1976), 123-143.

14. Soloway, E., Ehrlich, K., Bonar, J., Greenspan, J. What Do Novices Know About Programming? In A. Badre, B. Shneiderman, Ed., *Directions in Human-Computer Interactions*, Ablex, Inc., 1982.

15. Soloway, E., Bonar, J., Ehrlich, K. . Cognitive Strategies and Looping Constructs: An Empirical Study. Communications of the ACM, in press.

16. Soloway, E., Ehrlich, K. What DO Programmers Reuse? Theory and Experiment. Proceedings of the Workshop on Reusability in Programming, ITT, Providence, R.I., 1983.

17. Soloway, E., Ehrlich, K. What Does an Expert Programmer Know? Tech. Rept. 294, Dept. of Computer Science, Yale University, 1983.

18. Wirth, N. "On the Composition of Well-Structured Programs." *ACM Computing Surveys 6*, 4 (1974).

• THEORETICAL COMPUTER SCIENCE •

IMYCS'84

СИНХРОСЕТИ - СРЕДСТВО ОПИСАНИЯ ВЗАИМОДЕЙСТВИЯ АСИНХРОННЫХ ПРОЦЕССОВ

Быстров А.В.
Вычислительный центр
Новосибирск, 630090
СССР

1. Введение

В последнее время проблема описания параллельных и распределенных систем и исследования их свойств привлекает большое внимание. Одной из наиболее широко распространенных и активно изучаемых формальных моделей таких систем являются сети Петри [3]. В предложенной В.Е. Котовым алгебре сетей Петри [1] сети записываются в виде формул - конструкций обычных для языка программирования. Это дало возможность с помощью удобного и наглядного аппарата сетей описывать структуру управления параллельных программ. При этом каждому модулю или оператору программы сопоставляется переход сети, а необходимым условием для его исполнения является возможность срабатывания соответствующего перехода. В то же время, описание сложных иерархических систем, в которых есть взаимодействия по управлению между разными уровнями часто требует гло-

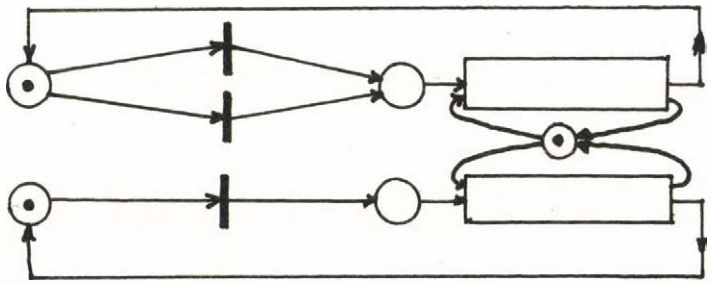
бального рассмотрения управляющей сети.

Рассматриваемые в данной работе синхросети, являющиеся обобщением сетей Петри, позволяют выделить в структуре управления определенные слои. Каждый из этих слоев описывается сетью, а их взаимодействие строго регламентировано. Слой может описывать в частности некоторую типичную синхронизацию, например, взаимное исключение модулей работающих с одним ресурсом. Такое локализованное описание типовой структуры управления, родственное понятию абстрактного типа данных оказывается весьма полезным при изучении параллельных систем.

2. Синхросети

Сетью Петри называется ориентированный двудольный граф, в котором вершины одного типа называются местами и изображаются кружками, а вершины другого типа называются переходами и изображаются вертикальными черточками или прямоугольниками. Каждому месту сопоставлено неотрицательное целое число, называемое разметкой. Разметка изображается соответствующим числом точек или фишек в вершинах. Функционирование сети Петри заключается в срабатывании переходов. Переход может сработать, если во всех его входных местах есть хотя бы по одной фишке. Срабатывая, переход изымает из каждого своего входного места по одной фишке и добавляет по одной фишке во все выходные места. В работе сети допустим параллелизм и автопараллелизм. Имееется в виду, что в промежутке между изъятием и рассылкой фишек могут срабатывать другие переходы, а также, если позволяет разметка, повторно этот же переход. Отметим, что сети с (авто)параллелизмом легко моделируются обычными сетями и наоборот.

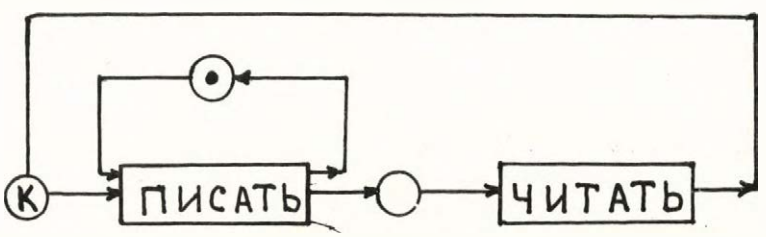
Пример сети, описывающей два параллельных циклических процесса с критическими интервалами:



Синхросеть представляет собой набор сетей Петри, называемых слоями. Слои функционируют параллельно но не независимо. Переход одного слоя может быть синхронизован переходом другого слоя, который также может быть синхронизован и т.д. Вместе они образуют связку. Это означает, что для того, чтобы сработал один из этих переходов необходимо, чтобы мог сработать и другой, причем срабатывают эти переходы вместе.

Можно рассматривать вариант синхросетей, в котором одним переходом могут быть синхронизованы несколько переходов. В этом случае при срабатывании связки срабатывает только один из этих нескольких переходов.

Приведем пример синхросети, описывающей типичную работу с буфером ограниченного объема. В буфер можно записывать и из него можно считывать. Если буфер пуст, то считывать нельзя, если же в нем уже есть K записей, то нельзя в него записывать.В системе много процессов, работающих с буфером, причем записывать в буфер может одновременно только один процесс, читать разрешается параллельно. Процессы, в которых используются операции чтения и записи в буфер могут быть сколь угодно сложными, возможно изменяющимися, их структура управления может быть произвольной, она совершенно не важна. Для того чтобы описать требуемую синхронизацию, достаточно воспользоваться изображенной ниже сетью. Ее переходы синхронизуют соответствующие операции работы с буфером.



В правилах функционирования синхросети было сказано, что синхронизованные переходы срабатывают вместе. Это внешне простое определение, однако, предполагает определенную синхронизацию самих механизмов работы слоев: проверок условий готовности, изменения разметки и т.п. Для корректного определения семантики синхросетей в условиях распределенного управления требуется зафиксировать некоторый протокол взаимодействия слоев. Опишем один из возможных протоколов в предположе-

нии, что на множестве слоев есть частичный порядок и отношение синхронизации с ним согласовано, т.е. все переходы одной связки можно линейно упорядочить. Это имеет место, в частности, для иерархических систем, в которых синхронизумые переходы находятся в иерерхии ниже уровня синхронизующих. Протокол основан на известном способе распределения ресурсов, исключающем дедлоки: все ресурсы упорядочиваются и процесс запрашивает ресурс следующего уровня только получив ресурсы предыдущих уровней. В данном случае ресурс - это фишки, требуемые для срабатывания перехода. Они изымаются из входных мест и только после этого выясняется, может ли сработать в своем слое следующий переход связки. Если может, то из его входных мест изымаются фишки, и это повторяется для всех переходов связки. Если переход не может сработать, то вниз по связке передается сигнал инициирующий возвращение фишек во входные места. В случае, когда описанным способом выяснена готовность всей связки и она срабатывает, каждый переход рассылает фишки в свои выходные места.

3. Заключение

Рассмотренные в данной работе синхросети составили основу средств описания структуры управления программ в Базовом языке программирования для многопроцессорных вычислительных систем, разработанном в Вычислительном центре СО АН СССР [2]

Литература

1. Котов В.Е. Алгебра сетей Петри. - Кибернетика, 1980, №5, с.10-18.
2. Kotov V.E. On Basic Parallel Language, Proceedings of IFIP Congress 80, North-Holl.Co., Amsterdam, 1980.
3. Petry C.A. Kommunication mit Automaten, Univ. Bonn, 1962.

IMYCS'84

GRAPHIC CONTROLLED TABLE LINDENMAYER SYSTEMS

Erika Braunsteinerová
Juraj Hromkovič

Department of Theoretical Cybernetics
Komensky University
842 15 Bratislava

1. Introduction

Theory of Lindenmayer systems also referred to as the theory of developmental systems is the result of an infusion of ideas from developmental biology into formal language theory. This theory has become a very actively investigated topic in the last few years. It has produced many results of interest both for the formal language theorists and the ones various types of real systems. Lindenmayer systems, short L systems, were introduced by Lindenmayer (1968).

We shall consider a special type of L systems, called TOL systems. The reason for introduction of TOL languages was the fact that developmental behavior of many organizms depends on environmental conditions (such as dark, light, cold, warm, etc.). To describe the development of such an organizm one has to provide different sets of developmental rules corresponding to the different environmental conditions, with the assumption that at each moment of time one such set is obeyed.

We will study graphic controlled versions of TOL systems in this paper. The main reason for introducing them was the fact that many real systems behave according to conditions whose changing is in nature of a law (day and night, spring, summer, autumn and winter, etc.) or it can be beforhand fixed in which order this conditions set in. We introduce 4 versions of graphic controlled table L systems, briefly graphic table L systems (we hope that it will cause no confusion), in dependence on considering the determinizm with respect to the graph (the order of the conditions) or to the TOL system.

The graphic TOL systems differ from TOL systems with determining the order of using different tables by deterministic or nondeterministic way. This order is determinated by directed graph in which each edge corresponds to using some table. In deterministic graphic TOL system exactly one edge goes out from each vertex.

This paper is organized as follows. Section 2 involves the basic definitions of GTOL systems. The hierarchy and closure properties of language families defined by GTOL, DGTOL, GDTOL and DGDTOL systems is given in Section 3. In Section 4 we shall give the definition of growth functions of DGTOL systems and make analysis and synthesis of these functions. Finishing Section 4 and this paper too we obtain some results according to growth equivalences.

2. Basic definitions

In order to establish our notation we will first review the definitions for TOL systems.

Definition: A TOL scheme is a pair $G = \langle \Sigma, \mathcal{R} \rangle$ where Σ is a finite, nonempty set called the alphabet, $\mathcal{R}$ (the set of tables) is a finite, nonempty set. Each element R of $\mathcal{R}$, called a table, is a finite, nonempty subset of $\Sigma \times \Sigma^*$ such that for all a in Σ there exists α in Σ^* that (a, α) belongs to R. We use $a \xrightarrow[G]{} \alpha$ or $a \rightarrow \alpha$ to denote (a, α) is in R.

Definition: A TOL system is a triple $S = \langle \Sigma, \mathcal{R}, x \rangle$ where $G = \langle \Sigma, \mathcal{R} \rangle$ is a TOL scheme and x, called the axiom of S, is a word over Σ^*.

Definition: A graphic TOL system, GTOL system, is a 5-tuple $\langle \Sigma, \mathcal{P}, G, \mathcal{R}, x \rangle$ where

Σ is a finite nonempty set of symbols called the alphabet,

$\mathcal{P}$ is a finite nonempty set of tables. Each element P of $\mathcal{P}$, called a table, is finite, nonempty subset of $\Sigma \times \Sigma^*$ (called productions) such that for all a in Σ and for all P in $\mathcal{P}$ there is α in Σ^* that (a, α) belongs to P,

$\mathcal{R} \subseteq \mathcal{P}$ is a nonempty set, called the initial tables,

$x \in \Sigma^*$ is an axiom,

$G \subseteq \mathcal{P} \times \mathcal{P}$ is a graph such that for all P in $\mathcal{P}$ there is at least one table P' in $\mathcal{P}$ such that (P, P') is in G.

Definition: Let $S = \langle \Sigma, \mathcal{P}, G, \mathcal{R}, x \rangle$ be a GTOL system. Let

$y = a_1 \ldots a_m$ with $m \geq 1$ and a_j in Σ for $j = 1,\ldots,m$ and let z be in Σ^*. Then we say that y directly derives z in S, denoted by $y \overset{G}{\Rightarrow} z$, iff there exists P in $\mathcal{P}$ and $\alpha_1,\ldots,\alpha_m$ in Σ^* such that $a_1 \xrightarrow{P} \alpha_1$, ... , $a_m \xrightarrow{P} \alpha_m$ and $y = \alpha_1 \ldots \alpha_m$. In this case we also write $x \overset{P}{\Rightarrow} y$.

Definition: For any GTOL system $S = \langle \Sigma, \mathcal{P}, G, \mathcal{R}, x \rangle$ we define a table set sequence $\{\mathcal{R}_n\}_{n=1}$ by induction on n in the following way $\mathcal{R}_1 = \mathcal{R}$

$$\mathcal{R}_n = \{P' \mid (P,P') \in G, \quad P \in \mathcal{R}_{n-1}\}$$

Definition: For any GTOL system $S = \langle \Sigma, \mathcal{P}, G, \mathcal{R}, x \rangle$ we define the finite language $A_n(S)$ by induction on n: $A_0(S) = \{x\}$

$A_1(S) = \bigcup_{P \in \mathcal{R}_1} A_{P,1}(S)$ $\qquad A_{P,1}(S) = \{y \mid x \overset{P}{\Rightarrow} y\}$

$A_n(S) = \bigcup_{P \in \mathcal{R}_n} A_{P,n}(S)$ $\qquad A_{P,n}(S) = \{y \mid$ there is z in $A_{P',n-1}(S)$ and $(P',P) \in G$ such that $z \overset{P}{\Rightarrow} y\}$.

We say that x derives y in S ($x \overset{*}{\underset{S}{\Rightarrow}} y$ or only $x \overset{*}{\Rightarrow} y$) if y is in $A_n(S)$ for some n.

Definition: Let $S = \langle \Sigma, \mathcal{P}, G, \mathcal{R}, x \rangle$ be a GTOL system. The language generated by S, denoted L(S), is defined as $L(S) = \{y \mid x \overset{*}{\underset{S}{\Rightarrow}} y\}$.

Definition: A deterministic graphic TOL system, DGTOL system, is a GTOL system $S = \langle \Sigma, \mathcal{P}, G, \mathcal{R}, x \rangle$ such that for all P in $\mathcal{P}$ there exists exactly one P' in $\mathcal{P}$ such that (P,P') is in G and $\mathcal{R}$ is an one-element set.

Definition: A graphic DTOL system, GDTOL system, is a GTOL system with deterministic tables (for each $a \in \Sigma$ there is exactly one production $a \to \alpha$ in each table $P \in \mathcal{P}$.)

Definition: A deterministic graphic DTOL system, DGDTOL system, is a DGTOL system with deterministic tables.

Note: Because DGTOL, GDTOL and DGDTOL systems are special cases of GTOL systems, all notion defined for GTOL systems applies for DGTOL, GDTOL and DGDTOL systems too.

Note: It is easy to see that a table set sequence of DGTOL system consists of one-element subsets of $\mathcal{P}$ (i.e. any $\mathcal{R}_i$ contains exactly one $P \in \mathcal{P}$). So, for DGTOL and DGDTOL systems we define a set $\{R_i\}_{i=1}^{\infty}$ called the table sequence where $R_i = P \iff P \in \mathcal{R}_i$. Because the set of tables in system is finite, it is clearly that G contains a cycle.

3. Hierarchy of language families and closure properties

To show the hierarchy of classes of languages generated by graphic table L-systems we give at first any assertions about languages which will be used.

Theorem 1. Let $L_1 = \{a^2, a^4\}$.
Then $L_1 \in \mathcal{L}(\text{DGDTOL})$, $L_1 \notin \mathcal{L}(\text{TOL})$.

Corollary 1:
$L_1 \in \mathcal{L}(\text{GDTOL}), \mathcal{L}(\text{DGTOL}), \mathcal{L}(\text{GTOL})$
$L_1 \notin \mathcal{L}(\text{DTOL}), \mathcal{L}(\text{OL}), \mathcal{L}(\text{DOL})$.

Theorem 2. Let $L_2 = \{a^2, a^4, a^6\}$.
Then $L_2 \in \mathcal{L}(\text{GDTOL})$, $L_2 \notin \mathcal{L}(\text{DGTOL})$.

Corollary 2:
$L_2 \in \mathcal{L}(\text{GTOL})$
$L_2 \notin \mathcal{L}(\text{TOL}), \mathcal{L}(\text{DGDTOL})$.

Theorem 3. Let $L_3 = \{a^{3^i 10^j} \quad i,j = 0,1,\ldots\}$.
Then $L_3 \in \mathcal{L}(\text{DTOL})$, $L_3 \notin \mathcal{L}(\text{DGTOL})$.

Corollary 3:
$L_3 \in \mathcal{L}(\text{TOL})$
$L_3 \notin \mathcal{L}(\text{DGDTOL})$.

Theorem 4. Let $L_4 = \{a^{2k-1} - \{a^7\} , k=1,2,\ldots\}$.
Then $L_4 \in \mathcal{L}(\text{OL})$, $L_4 \notin \mathcal{L}(\text{GDTOL})$.

Corollary 4:
$L_4 \in \mathcal{L}(\text{TOL}), \mathcal{L}(\text{GTOL}), \mathcal{L}(\text{DGTOL})$
$L_4 \notin \mathcal{L}(\text{DGDTOL})$.

For the sake of easy survey the relations between this classes, we give three graphes. In these we note the relations $\mathcal{L}(A) \subset \neq \mathcal{L}(B)$ with $A \longrightarrow B$ and $\mathcal{L}(A) \not\subseteq \mathcal{L}(B)$ with $A \dashleftarrow\!\dashrightarrow B$.

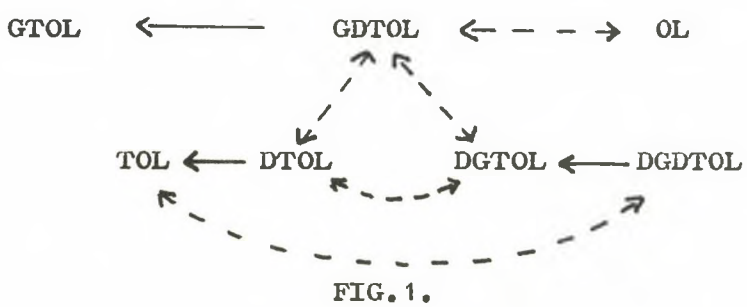


FIG. 1.

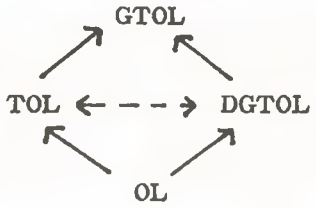


FIG.2.

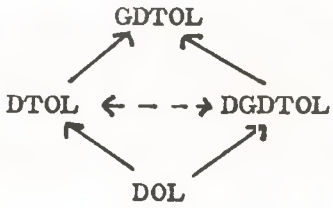


FIG.3.

For closure characteristics the families generated by systems defined in this paper we have a following theorem.

Theorem 5. The families of DGTOL, DGDTOL, GDTOL and GTOL languages are not closed under

a) union

b) intersection with regular languages

c) ε -free homomorphism

d) inverse homomorphism.

4. The growth functions of deterministic graphic TOL systems

In this part we shall give a definition, analysis and synthesis of growth functions of DGTOL systems.

As a conclusion we shall give some assertions about growth equivalence of these systems.

Definition: Let $S = \langle \Sigma, P, G, R, x \rangle$ be a DGTOL system. The growth function of S, denoted f_S, is defined for $n \geq 0$ as follows

$$f_S(n) = \frac{1}{|A_n(S)|} \cdot \sum_{x \in A_n(S)} |x|$$

where $|A_n(S)|$ is a cardinality of $A_n(S)$, $|x|$ is the lenght of x.

So, f_S is a mapping from natural numbers to the nonnegative rationals such that $f_S(n)$ is the expected word lenght after n steps of derivation of S.

Before given the algorithm for computing of growth function we define still some important notions.

Definition: Let $S = \langle \Sigma, P, G, R, x \rangle$ be a DGTOL system and $\Sigma = \{a_1, a_2, \ldots, a_t\}$. Let $\# a_j(x)$ be the number of occurences a_j in x.

The Parikh vector of x, denoted v(x), is the following t-dimensional row vector $(\# a_1(x), \ldots, \# a_t(x))$.

The growth matrix of table P, denoted $G_P = (g_{ij})$ is the t by t matrix where

$$g_{ij} = \frac{1}{num(a_i,P)} \cdot \sum_{\alpha} \#a_j(\alpha),$$

the sum is over all α such that $a_i \to \alpha$ is in P and $num(a_i,P)$ is a number of all productions in P with a_i in left hand.
The sum vector $\overline{1}$ is a t-dimensional column vector with all components equal to 1.

Theorem 6. For any DGTOL system $S = \langle \Sigma, P, G, R, x \rangle$ and any natural number n, the value $f_S(n)$ of the growth function can be computed by

$$f_S(n) = v(x) \cdot \prod_{i=1}^{n} G_{R_i} \cdot \overline{1}$$

Theorem 7. For any DGTOL system with n tables and a cycle of n-k+1 lenght there are matrixes $M_1, \ldots, M_n, G_1, G_2$ such that

$$f(k-1+a(n-k+1)+b) = v \cdot G_1 \cdot G_2^a \cdot M_k \ldots M_{k+b-1} \cdot \overline{1}$$

for all $a = 0,1,\ldots$ and all b in $\{0,1,\ldots,n-k\}$.

Theorem 8. There is an algorithm which, given a DGTOL system S with t symbols, n tables and with cycle of lenght n-k+1, will construct a system of n-k+1 linear reccurence equations of degree at most t such that the growth function f_S of S fulfils this equations.

Theorem 9. For any DGTOL system $S = \langle \Sigma, P, G, R, x \rangle$ with t symbols, n tables, with cycle of lenght n-k+1 and with growth function f_S there is an algorithm, which, given the first $k-1+t(n-k+1)$ values of f_S, will construct a system of n-k+1 linear reccurence equations for f_S such that all values of f can be computed from these initial values and the reccurence equations.

Definition: By $\mathcal{F}(DGTOL)$ we denote the class of all functions which can be obtained as growth functions of arbitrary DGTOL system, i.e. $\mathcal{F}(DGTOL) = \{f_S \mid S \in DGTOL\}$.

Theorem 10. Let f(n) be in $\mathcal{F}(DGTOL)$ and c be a natural number. Then the function $g(n) = c.f(n)$ is in $\mathcal{F}(DGTOL)$.

Theorem 11. Let f(n) and g(n) are in $\mathcal{F}(DGTOL)$. Then the function f(n) + g(n) is in $\mathcal{F}(DGTOL)$.

Definition: Let $S = \langle \Sigma, P, G, R, x_1 \rangle$ and $G = \langle \Sigma, P, G, R, x_2 \rangle$ be two DGTOL systems. Let f_S and f_G be the growth functions of DGTOL systems S and G respectively. We say that x_1 is k growth equivalent to x_2, if $f_S(n) = f_G(n)$ for all $n=0,\ldots,k$. We say that x_1 is growth equivalent to x_2 provided $f_S(n) = f_G(n)$ for all $n \geq 0$.

Theorem 12. Let $S = \langle \Sigma, P, G, R, x_1 \rangle$, $G = \langle \Sigma, P, G, R, x_2 \rangle$ be a DGTOL systems with t symbols, n tables and with cycle of lenght n-k+1. Then x_1 and x_2 are growth equivalent iff they are $k-1+t(n-k)$ growth equivalent.

Theorem 13. Let $S_1 = \langle \Sigma_1, P_1, G_1, Q_1, x_1 \rangle$ resp. $S_2 = \langle \Sigma_2, P_2, G_2, Q_2, x_2 \rangle$ be a DGTOL systems with t_1 resp. t_2 symbols, n_1 resp. n_2 tables and with cycle of length c_1 resp. c_2. Then S_1 and S_2 are growth equivalent iff they are n-c+t.c growth equivalent where

c = smallest common multiple of c_1 and c_2,

$n = \max \{n_1-c_1, n_2-c_2\} + c$,

$t = t_1+t_2$.

ACKNOWLEDGEMENT

We would like to thank Branislav Rovan for his comments concerning this work.

REFERENCES

1. Herman,G. and Rozenberg,G. (1975), Developmental Systems and Languages, North-Holland, Amsterdam
2. Hromkovič,J. and Mertan,J. (1982), Stochastic Table Lindenmayer Systems, Technical Report, Komensky University, Bratislava
3. Lindenmayer,A. (1968), Mathematical Models of Cellular Interaction in Development, J. Theoret. Biol. 18, 280-299

TAG SEQUENCES AND SUBSTITUTIONS

Anton Černý

Department of Theoretical Cybernetics
Faculty of Mathematics and Physics
Comenius University
Mlynská dolina, 842 15 Bratislava
Czechoslovakia

1. Introduction

Infinite words serve often as a useful tool in the investigation of the structural and combinatorial properties of words. A wide class of infinite words is formed by the so-called tag sequences (Cobham, 1972). This class is quite natural since each (uniform) tag sequence can be characterized by a finite automaton.

In the present paper a characterization of uniform tag sequences by means of substitutions is given. Moreover, new kinds of operations on infinite words - a stepwise compression and a stepwise substitution - are described. It is shown that the family of all (uniform) tag sequences is closed under stepwise compression and that the family of all infinite words obtainable by iteration of stepwise substitutions is a proper subfamily of all (uniform) tag sequences.

2. Basic notions

Let $N = \{0,1,2,\dots\}$ and for $p \in N$ let $[p] = \{0,1,\dots,p-1\}$. For a finite alphabet Σ let Σ^p denote the set of all mappings (finite words over Σ of length p) $W: [p] \longrightarrow \Sigma$ and let Σ^N denote the set of all mappings (infinite words over Σ) $W: N \longrightarrow \Sigma$. Moreover denote $\Sigma^* = \bigcup_{p=0}^{\infty} \Sigma^p$, $\Sigma^\infty = \Sigma^* \cup \Sigma^N$ and ε the empty word. We shall identify Σ and Σ^1.

Σ^* together with the usual concatenation of words form a free monoid over Σ, moreover the concatenation can be par-

tially extended to Σ^∞ - finite words can be concatenated with infinite words but not vice versa. Denote for $W \in \Sigma^*$ $|W|$ to be the length of W and for $W \in \Sigma^N$ set $|W| = \infty$.

For a word $W \in \Sigma^\infty$ and for $i,j \in N$, $j \geq 1$ let

$$\mathrm{Sub}^i_j(W) = \varepsilon \quad \text{if } i+j > |W|$$
$$= W(i)W(i+1)\ldots W(i+j-1) \quad \text{otherwise}$$

Let Σ, Γ be alphabets. A mapping $\varphi : \Sigma^* \longrightarrow \Gamma^*$ is called <u>morphism</u> if for all $X,Y \in \Sigma^*$ $\varphi(XY) = \varphi(X)\,\varphi(Y)$. φ is uniquely determined by its values on Σ and can be extended in a natural way to map Σ^∞ to Γ^∞. φ is called <u>p-uniform</u> for $p \in N$ iff for all $a \in \Sigma$ $|\varphi(a)| = p$.

A word W over Σ is called <u>simple tag sequence</u> iff there is a morphism $\varphi : \Sigma^\infty \longrightarrow \Sigma^*$ and a letter $a \in \Sigma$ such that $W = \lim_{n\to\infty} \varphi^n(a)$, i.e. each $\varphi^n(a)$, $n \geq 0$, is a (mapping) restriction of W. This is possible iff the first letter of $\varphi(a)$ is a, φ is called <u>prolongable in a</u> in this case.

<u>Example 2.1:</u> The infinite word T = abbabaabbaababba... generated by the morphism $\varphi(a) = ab$, $\varphi(b) = ba$ is a simple tag sequence - the well-known sequence of Thue (Thue 1912) containing no cubes (nonempty subwords w^3).

A word W over Γ is called <u>tag sequence</u> iff there is some alphabet Σ, a simple tag sequence $\overline{W}$ over Σ (generated by some morphism φ prolongable in a) and a 1-uniform morphism $\psi : \Sigma^* \longrightarrow \Gamma^*$ such that $W = \psi(\overline{W})$. We shall write $W = T(\Sigma, a, \varphi, \Gamma, \psi)$. If φ is a p-uniform morphism then W is called <u>p-uniform tag sequence</u>.

We have the following properties of p-uniform tag sequences (Cobham, 1972)

<u>Proposition 2.1:</u> W is a p-uniform tag sequence iff there are some integers $0 \leq r < s$ such that for all $i,j \geq 0$ $\mathrm{Sub}^i_{p^r}(W) = \mathrm{Sub}^j_{p^r}(W)$ implies $\mathrm{Sub}^i_{p^s}(W) = \mathrm{Sub}^j_{p^s}(W)$.

<u>Proposition 2.2:</u> W is p-uniform iff W is p^r-uniform (r being arbitrary, $r \geq 1$, and W a tag sequence).

Proposition 2.3: The family of all p-uniform tag sequences is closed under

2.3.1 q-uniform morphisms, $q \geq 1$

2.3.2 inverse of injective q-uniform morphisms, $q \geq 1$

3. Substitutions

In (Christol et al., 1980) a notion of substitution has been introduced. An (i,j)-substitution, $1 \leq i < j$, is a partial mapping $\mu : \Sigma^* \longrightarrow \Gamma^*$ defined as $\mu = \beta \circ \gamma^{-1}$ where $\gamma : \Delta^* \longrightarrow \Sigma^*$ is an injective i-uniform morphism and $\beta : \Delta^* \longrightarrow \Gamma^*$ is a j-uniform morphism. Substitutions can be extended in a natural way to infinite words. An easy observation gives us the following lemma.

Lemma 3.1: Let $X \in \Sigma^N$ and let $\mu : \Sigma^\infty \longrightarrow \Sigma^\infty$ be a (q,p.q)-substitution, $q \geq 1$, $p \geq 2$. Then $\mu(X) = X$ iff $X = \lim_{n \to \infty} \mu^n(W)$ for some $W \in \Sigma^q$.

We can give a characterization of p-uniform tag sequences by means of substitutions:

Theorem 3.1: An infinite word $X \in \Sigma^N$ is a p-uniform tag sequence ($p \geq 2$) iff $X = \mu(X)$ for some $(q,p^m.q)$-substitution μ, $q \geq 1$, $m \geq 1$.

Proof: 1. The only-if-part follows from Proposition 2.1 and Proposition 2.2.

2. The if-part. Let $X = \mu(X)$ for a $(q,p^m.q)$-substitution μ. Let $\mu = \beta \circ \gamma^{-1}$ where β and γ are as in the definition of substitution. Then $\gamma^{-1} \circ \beta$ is a p^m-uniform morphism and

$$\gamma^{-1} \circ \beta (\gamma^{-1}(X)) = \gamma^{-1}(\beta \circ \gamma^{-1}(X)) = \gamma^{-1}(X)$$

According to Lemma 3.1 (the case $q = 1$) $\gamma^{-1}(X)$ is a p^m-uniform tag sequence. Proposition 2.2 and 2.3.1 of Proposition 2.3 imply that X is a p-uniform tag sequence.

The operation from 2.3.2 of Proposition 2.3 is often called the block compression. However, one can imagine another

way how to compress the blocks - the stepwise compression - when each block of q consecutive letters is compressed. Formally, let $\Sigma_q = \{\langle W\rangle \mid W \in \Sigma^q\}$, $q \geq 1$, and let $\delta_q : \Sigma_q \longrightarrow \Sigma^q$ be the bijection defined by $\delta_q(\langle W\rangle) = W$. Furtheron, denote $C_q = \{ W \in \Sigma_q^\infty - \{\varepsilon\} \mid$ for all $1 \leq i < |W|$ $\mathrm{Sub}^0_{q-1}(\delta_q(W(i))) = \mathrm{Sub}^1_{q-1}(\delta_q(W(i-1)))\}$. δ_q can be extended to the bijection

$$\delta_q : C_q \longrightarrow \Sigma^q\Sigma^\infty$$

defined inductively for $W \in C_q$, $2 \leq |W| < \infty$ by

$$\delta_q(W) = \delta_q(\mathrm{Sub}^0_{|W|-1}(W)).\mathrm{Sub}^{q-1}_1(\delta_q(\mathrm{Sub}^{|W|-1}_1(W)))$$

and for infinite words from C_q by

$$\delta_q(\lim_{n\to\infty} W_n) = \lim_{n\to\infty} \delta_q(W_n)$$

δ_q is called the <u>stepwise q-block compression</u>.

<u>Lemma 3.2</u>: Let $X \in \Sigma^N$ be a simple (p-uniform) tag sequence. Then $\delta_q^{-1}(X) \in \Sigma_q^N$ is a simple (p-uniform) tag sequence.

<u>Proof</u> (sketch): Let $X = \lim_{n\to\infty} \varphi^n(a)$ where φ is a (p-uniform) morphism. Denote $\nu : \Sigma_q^* \longrightarrow \Sigma_q^*$ the (p-uniform) morphism defined for $A \in \Sigma_q$ by

$$[\nu(A)](i) = \delta_q^{-1}(\mathrm{Sub}^i_q(\varphi \circ \delta_q(A))) \quad \text{for } i \in [n_A]$$

where for $Z \in C_q$ $n_Z = |\varphi(\mathrm{Sub}^0_{|Z|}(\delta_q(Z)))|$.
One can (in a rather technical way) prove $\nu(\delta_q^{-1}(X)) = \delta_q^{-1}(X)$. Lemma 3.1 implies that $\delta_q^{-1}(X)$ is a (p-uniform) tag sequence.

<u>Theorem 3.2</u>: Let $W \in \Sigma^N$, $p,q \geq 2$. Then W is a (p-uniform) tag sequence if and only if $\delta_q^{-1}(W)$ is a (p-uniform) tag sequence.

<u>Proof</u>: Considering Proposition 2.3 one can see that it is sufficient to prove the theorem for simple tag sequences. The only-if-part follows from Lemma 3.2. The if-part follows from the fact that $W = \psi(\delta_q^{-1}(W))$ where $\psi : \Sigma_q^* \longrightarrow \Sigma^*$ is a morphism defined for $A \in \Sigma_q$ by $\psi(A) = [\delta_q(A)](0)$.

As an analogue of an (i,j)-substitution we can define the (i,j)-stepwise substitution ($i \geq 1$, $j \geq 2$) to be a mapping $\mu = \beta \circ \delta_i^{-1}$ where δ_i is the stepwise i-block compression as above and $\beta : \Sigma_i^* \longrightarrow \Gamma^*$ is a j-uniform morphism. An analogue of Lemma 3.1 is valid for arbitrary stepwise sub-

stitutions.

Our main result concerning stepwise substitutions is stated in the following theorem.

Theorem 3.3: If for some $X \in \Sigma^N$ and for some (i,j)-stepwise substitution μ on Σ one has $\mu(X) = X$ then X is a j-uniform tag sequence.

Proof (sketch): Let s be an integer, $s \geq \max\{2, (i+j-2)/(j-1)\}$ Let $\gamma : (\Sigma_i)_s^* \longrightarrow (\Sigma_i)_s^*$ be a j-uniform morphism defined for $A \in (\Sigma_i)_s$ by

$$\gamma(A) = \delta_s^{-1}(\mathrm{Sub}^0_{j+s-1}(\delta_i^{-1} \circ \beta \circ \delta_s(A))) .$$

One can prove

$$\gamma(\delta_s^{-1} \circ \delta_i^{-1}(X)) = \delta_s^{-1} \circ \delta_i^{-1}(X)$$

Hence the latter is a j-uniform tag sequence and so is X according to Theorem 3.2 and Proposition 2.3.

Obviously, each simple p-uniform tag sequence is a fixpoint of some (i,p)-stepwise substitution, for each $i \geq 1$. This is not true for p-uniform tag sequences, as shown in Example 3.1. On the other hand, in Example 3.2 a tag sequence is shown which is not simple but it is a fixpoint of a stepwise substitution.

Example 3.1: Let $X = \psi(\bar{X})$, $\bar{X} = \lim_{n\to\infty} \varphi^n(a)$ where $\varphi(a) = ab$, $\varphi(b) = ac$, $\varphi(c) = cc$, $\psi(b) = \psi(c) = 0$, $\psi(a) = 1$. Then

$\bar{X}$ = abacabccabacccccabacabcccccc...

X = 10101000101000001010100000000...

Suppose there is a (i,j)-stepwise substitution generating X. $\bar{X}$ contains arbitrarily long factors bc^{i-1} and c^i. The corresponding factors 0^i of X should be mapped to 10^{j-1} and 0^j, respectively - a contradiction.

Example 3.2: For the stepwise substitution μ defined by $\mu(aa) = \mu(ab) = \mu(ba) = ab$, $\mu(bb) = ba$ the infinite word X = abbaababab... is a fixpoint. Obviously, X is not a simple 2-uniform tag sequence.

The result can be summarized as follows.

Theorem 3.4: The set of all infinite fixpoints of (i,p)-stepwise substitutions, $i \geq 1$, properly contains the set of all infinite simple p-uniform tag sequences, and it is properly contained in the set of all infinite p-uniform tag sequences.

We are not able to give an analogical theorem for (i,j)-substitutions. Our conjecture is, that the set of all uniform tag sequences is properly contained in the set of all fixpoints of arbitrary substitutions. The proof will not be very easy since the necessary conditions for a sequence to be a uniform tag sequence (Cobham, 1972) seem to be valid for the above fixpoints, too. These sequences can be characterized as "uniform with rational modulus".

4. References

Christol, G., Kamae, T., Mendes-France, M., Rauzy, G.: Suites algébriques, automates et substitutions, Bull. Soc. math. France 108 (1980), 401 - 419

Cobham, A.: Uniform tag sequences, Math. Syst. Theory 6 (1972), 164 - 192

Thue, A.: Über die gegenseitige Lage gleichen Teile gewisser Zeichenreihen, Videnskapsselskapets Skifter, I. Mat. - naturv. Klasse, Kristiania 1912, N° 1, 1 - 67

SOME THEOREMS ON k-BOUNDED INTERPRETATIONS OF FINITE LANGUAGE FORMS

Erzsébet Csuhaj-Varju
Computer and Automation Institute
Hungarian Academy of Sciences
1132 Budapest
Victor Hugo u. 18-22.
Hungary

1. INTRODUCTION

The notion of an interpretation of a finite language /finite form/ was introduced in [2] . The concept was motivated by the notion of an interpretation of a grammar form and some important properties of finite forms and of the corresponding language family were examined.

In this paper we define a restricted version of the interpretation a finite form, namely the notion of a k-bounded interpretation. This concept has a deep combinatorial motivation. We present some interesting properties of the k-bounded interpretation family, as we study the structural decomposability of it.

2. BASIC TERMINOLOGY AND RESULTS

We first review the notion of an interpretation of a finite form [2] and introduce the notion of a k-bounded interpretation and a k-th inflation of it. We state some simple properties concerning them.

In the following let Σ be an infinite countable set of abstract symbols.

DEFINITION 1.
A finite language $L_1 \subset \Sigma_1^*$ is said to be a finite form iff $\Sigma_1 \subset \Sigma$.

DEFINITION 2.

Let L_1 and L_2 be finite forms with $L_i \subseteq \Sigma_i^*$, $\Sigma_i \subset \Sigma$, i=1,2. We say that L_1 is an interpretation of L_2 modulo μ, written $L_1 \lhd L_2(\mu)$, if μ is a finite substitution $\mu : \Sigma_2^* \to 2^{\Sigma_1^*}$ satisfying

/1/ $\mu(a) \subseteq \Sigma_1$, for all $a \in \Sigma_2$;

/2/ $\mu(a) \cap \mu(b) = \emptyset$, for all a,b in Σ_2, $a \neq b$;

/3/ $L_1 \subseteq \mu(L_2)$.

NOTATION 1.
Let L_1 be a finite form. The set of all interpretations of

L_1 is denoted by $L(L_1)$.

Next we define the notion of a k-bounded interpretation of a finite form. This notion is motivated by the notion of a k-bounded interpretation of a grammar form introduced in [3].

DEFINITION 3.

Let L_1 and L_2 be finite forms and let $L_2 \in L(L_1)$. Let $H=\{\mu \mid L_2 \lhd L_1(\mu)\}$. L_2 is called a k-bounded interpretation of L_1 if

$$\min_{\mu \in H} \max_{\substack{a \in \Sigma, \\ a \text{ occurs in at least one word in } L_1}} \operatorname{card}\{b \mid b \in \mu(a), \ b \text{ occurs in at least one element of } L_2\} = k .$$

The set of j-bounded interpretations of L_1, where $j \leq k$, is said to be the k-bounded interpretation family / grammatical family/ of L_1 and is denoted by $L_k(L_1)$.

The next notion which is analogous to the notion of the k-th inflation of a grammar form expresses maximality.

DEFINITION 4.

Let L_2 be an interpretation of a finite form L_1. We say that L_2 is a k-th inflation of L_1 if there exists a finite substitution μ such that $L_2 \lhd L_1(\mu)$ and $\operatorname{card}\{\mu(a)\}=k$ for all $a \in \Sigma$ and $L_2=\mu(L_1)$. The set of all k-th inflations of

L_1 is denoted by $K(L_1)$.

The following two theorems can be obtained by some simple considerations [4] .

THEOREM 1.

Let L_1 and L_2 be two finite forms such that $L_1(L_1)=L_1(L_2)$. Then L_1 and L_2 are isomorphic.

THEOREM 2.

For all finite forms L and for all positive integers k $L_k(L)= L_1(K(L_1))$.

3. DECOMPOSITION

In this section we define the notion of a decomposable finite form and that of a k-th order decomposition of it. We present some theorems concerning the structure of the k-bounded interpretation family.

DEFINITION 5.

A finite form L_1 is said to be k-th order decomposable if there exists a finite form L_2 such that L_1 is a k-th inflation of L_2 . L_2 is said to be a k-th order decomposition of it. A finite form L is said to be decomposable if there exists a positive integer $k > 1$ such that L is k -th order decomposable. The number k is called a

number of decomposablity of L. The set of decomposablity numbers of L is said to be its spectrum and is denoted by $SP(L)$.

Next we define a concept showing the degree of decomposability of a finite form.

DEFINITION 6.

Let L be a decomposable finite form and let L_1 be a k-th order decomposition of it, where $k=\max SP(L)$. Then L_1 is said to be a kernel of L. The set of all kernels of L is denoted by $KER(L)$.

The next theorem is of special importance and needs combinatorial considerations. The proof of it can be found in [5].

THEOREM 3.

Let L be a decomposable finite form. Then all kernels of it are isomorphic.

The following theorem shows that the operation of forming kernels is insensitive to the operation of forming k -th inflations.

THEOREM 4.

For all finite forms L and for all positive integers k

$KER(K(L))=KER(L)$.

PROOF.

Let L be a finite form and let k be an arbitrary positive integer. Let $L_1 \in K(L)$. First, assume that max $SP(L)=n$, max $SP(L_1)=l$. By simple considerations we obtain that $l \geq k \cdot n$. First, let $l=k \cdot n$. Let L′ be a kernel of L . Then by the corresponding definition $L_1 \in K(L) \subseteq K(N(L))$. So, $L' \in KER(L_1)$. Applying Theorem 3. and the corresponding definitions we obtain that in this case $KER(L)=KER(L_1)$. Let $l > k \cdot n$ and let L′ and L_1' be arbitrary kernels of L and L_1 , respectively. It comes by technical considerations that the number of symbols occurring in L is equal to the number of symbols occurring in L_1 and the number of words of L is equal to the number of words of L_1 . Moreover, L′ and L_1' are 1-bounded interpretations of each other. Then they are isomorphic and by Theorem 3. it means that $KER(L)= KER(L_1)$.
Hence the result.

The next theorem characterizes the spectrum.

THEOREM 5.

Let L be a finite form. Then $SP(L)$ consists of all divisors of max $SP(L)$.

The proof can be found in [5] .

The last theorem is about the equality of bounded interpretation families.

THEOREM 6.

Let L_1 and L_2 be different finite forms and let k and l be different positive integers such that $L_k(L_1) = L_l(L_2)$. Then

/1/ at least one of L_1 and L_2 is decomposable;

/2/ $KER(L_1) = KER(L_2)$;

/3/

$$k = \frac{l.c.m.(\max SP(L_1), \max SP(L_2))}{\max SP(L_2)} \cdot j$$

$$l = \frac{l.c.m.(\max SP(L_1), \max SP(L_2))}{\max SP(L_2)} \cdot j$$

where j is a positive integer.

The proof can be found in [5] .

REFERENCES

Maurer, H.A., Salomaa A., Wood D., /1980/, Context-free grammar forms with strict interpretations

IMYCS'84

A NEW ERROR RECOVERY METHOD FOR OPTIMIZED LR PARSERS

Peter Forbrig

Wilhelm-Pieck-Universitat Rostock
Sektion Informationsverarbeitung
Albert-Einstein-Str. 21
DDR - 2500 Rostock

1. Introduction

During the last ten years an increasing activity in developing error handling methods for syntaxtic analysis algorithms can be observed. This is doubtless a consequence of the low developed theory of error handling in comparision with the high developed theory of language translation.

But also the development of more and more special languages and their compilers asks for general strategies in handling syntactic errors.

The error handling routine is a main part of the syntactic analysis, because most programs, which have to be analysed, are affected with errors. So it is necessary to detect all syntactic errors in the first run. Otherwise a sequence of further fruitless translation attempts could be the consequence.

Error handling methods can be broken up into error recovery and error correction schemes.
The only aim of error recovery algorithms is to continue parsing after the error for discovering as many further errors as possible. The errors themselves are not corrected.
Error correction methods try to transform the erroneous part of a program into a syntactically correct one.

In spite of numerous promising statements up to now there is no general solution, which meets all practical requirements.

In most cases the known methods have disadvantages in error detection, space or computation.
Many existing compilers perform language dependent actions during error handling. Already slight changes in language design can destroy the used concept.
Modern compiler construction by generating systems needs general methods.

A project is under way at the Wilhelm-Pieck-University of Rostock to implement the compiler generating system RUEGEN. It is based on the principle of the grammar of syntactic functions (see (Rie 76) and (Rie 83)), which is a kind of an attributed grammar.

The already implemented LR (1) parser generator uses the algorithm of Pager (Pag 77), which is based on the idea of Knuth (Knu 68). It produces an optimized LR (1) parser.

This part of the system is now improved by a slightly modified version of the algorithm of Pager (Pag 73), which eliminates unit productions without semantic functions of the form A ::= B (A, B are nonterminals).
The algorithm and its comparision with (Pag 73) is described in (For 82 B).

The compiler generating system has also an error recovery routine for the optimized LR parser.
The used method is a combination of the algorithm of Leinius (Lei 70) and my own method (For 80).

It was suggested in (For 80) to use specieal error rules in the grammar. This decreases the number of states in the generated parser, but gives a good possibility of error diagnostics. In this paper I propose a new method with a reduced number of error rules, but with the same quality of error handling.
(A detailed comparision of numerous methods can be found in (For 82 A). A large bibliography can be found also in (Cie 79).)

2. LR error recovery

It is assumed that the reader is familiar with principles of LR parsers. For detailed information see (Aho 73), (Aho 74) or (Aho 77).

The proposed error recovery method tries to restart the Syntactical analysis after detecting an error as soon as possible in a correct way.
This is achieved also with the help of new error rules in the grammar.
Error rules will be generated for all production rules with terminals which have the character of brackets.

If a production rule has the form

$$A ::= X_1 \ldots X_n \text{ "(" } X_{n+2} \ldots X_m \text{")" } X_{m+2} \ldots X_k$$

A is a nonterminal
X_i is a nonterminal or terminal for $1 \leq i \leq k$
"(" is an "opening" bracket"
")" is a "closing bracket"

$$0 \leq n \leq m - 2 \leq k - 4$$

the error rule

$$A ::= \text{"ERROR" ")" } X_{m+2} \ldots X_k$$

will be generated.

"ERROR" is a new terminal, which has not been an element of the grammar.

For the grammatical rule

```
statement ::= "IF" condition "THEN" statementlist
                             "ELSE" statementlist "FI".
```

for example the following error rules will be produced.

```
statement ::= "ERROR" "THEN" statementlist
                      "ELSE" statmentlist "FI".
statement ::= "ERROR" "ELSE" statementlist "FI".
statement ::= "ERROR" " FI".
```

Beside "brackets" some other terminals are choosen to be potential start points after errors. These terminals should have a ambigious syntactical meaning in the programming language to guarantee a correct restart for the parser. Some error handling systems use identifiers as restart terminals. Our tests have shown that identifiers are too weak for this purpose. They have an unambigious sysntactical meaning in most programming languages. Therefore it is better to restrict the set of potential restart terminals to some (or all) keywords, operators and delimiters.

In our method the potential restart terminals are broken up into three groups.

(A) All opening terminals of syntactcal constructions.
    (e. g. "IF", "CASE", "BEGIN", "(")
(B) All brackets in closing position.
    (e. g. "END", "THEN", "ELSE", "FI", ")")
(C) All other restart terminals.
    (e. g. "," ";", "+", "=")

After detecting a syntactical error the following algorithm is proposed.

I. The parser skips the remaining input-string to the next restart terminal. Go to II.

II. If the restart terminal belongs to group A then go to VII else go to III.

III. All elements are popped from the stack until a nonterminal successor of the top element exists, which accepts the restart terminal.
If such a nonterminal successor exists then go to IV else go to V.

IV. The nonterminal transition will be performed and the syntactical analysis can go on. The error handling method has finished.

V. If the restart terminal belongs to group B then go to VI else the actual restart terminal is skiped and the error routine begins again with I.

VI. In these cases the bracket structure of the input-string is incorrect. Now it is the task of the error rules to allow a further analysis.
The stack is popped until the top of the stack accepts ("ERROR "restart terminal"). If this attempt is successfull, the error handling process finishes and the analysis can go on
else the restart terminal will be skipped and the error routine begins again with I.

VII. The stack is popped until the top of the stack accepts the restart terminal.
If the stack is empty another attempt is performed with the current restart terminal. go to III.
Otherwise the syntactical analysis can go on and the error handling method has finished.

3. Summary

The error recovery method explained in this paper has the same quality in error detection as the method in (For 80), but needs less space. It has a better error detection quality than (Lei 70), (Aho 74) and other algorithms. For detailed comparisions with some other methods see (For 82 A). A bibliography of error handling methods can also be found in (Cie 79).

The proposed method has the advantage that "important terminals" are treated correspondingly and "insecure" terminals are not used.

The correct bracket structure will never be destroyed. This is an advantage over many other methods (e. g. (Lei 70), (Aho 74)).

The method can be used in compiler generating systems, because error rules can be constructed automatically. Only three groups of restart terminals explained in the paper are needed.

References

(Aho 73) Aho,A. V.; Ullmann, J.: The Theory of Parsing, Translation and Compiling, Prentice Hall 1973

(Aho 74) Aho,A. V.; Johnson, C. C.: LR parsing Computing Surveys, Vol. 6, No. 2, 1974, p. 99-124

(Aho 77) Aho,A. V.: Ullmann, J.: Principles of Compiler Design, Addison Wesley 1977

(Cie 79) Ciesinger, J.: A Bibliography of Error Handling, SIGPLAN Notices, Vol. 14, No. 1, 1979

(For 80) Forbrig, P.: Untersuchungen zur Fehlerbehandlung und Effektivitatssteigerung bei LR (1) Syntaxanalyseverfahren, Ph. d. Thesis, WPU Rostock 1980

(For 82 A) Forbrig, P.: Untersuchungen zur Fehlerbehandlung bei der syntaktischen Analyse in LR (k) Verfahren. WPU Rostock, Rechenzentrum Report WPU-PS-Nr. (1982)

(For 82 B) Forbrig, P.: Steigerung der Effektivitat der LR (1) Syntaxanalyse durch Eliminerung von Einheitsreduktionen. WPU Rostock, Rechenzentrum Report WPU-PS-Nr. 2 (1982)

(Lei 70) Leinius, R. P.: Error Detection and Recovery for Syntax directed compiller Systems. Ph. d. Thesis, University of Wisconsin, 1970

(Pag 73) Pager, D.: Eleminating Unit Reductions from LR Parsers, University of Hawaii, Techn, Report 1973

(Pag 77) Pager, D.: A Practical General Method for Constructing LR (k) Parsers. Acta Informatica, Vol. 7, Fasc. 3, 1977

(Rie 76) Fiedewald, G.: Grammatiky syntaktickych funkci a jejich pozivani v kompilatorech, Acta polytechnica 4 (1976) 2/6, p. 89 - 103

(Rie 83) Riedewald, G.: Maluszynski, J.: Dembinski, P.: Formale Beschreibung von Programmiersprachen: Eine Einführung in die Semantik, Akademie-Verlag, Berlin, 1983
(also Oldenbourg Verlag München Wien)

FORESTS p-LANGUAGES AND p-REGULAR SYSTEMS

Maria Forys

Institute of Computer Science
Jagellonian University
Kopernika 27
Kraków
Poland

1. Introduction

In this paper the languages composed of forests are considered and a new framework for these considerations is presented. Forests p-languages there are languages in which any word is a p-tuple of trees i.e. forest. We assume that from each node of any tree of a forest at most p branches are going out. The matrix representation of such forest is defined and it simplified our considerations. We define some operations on the family of all forests and then on p-languages : a catenation, a j-catenation and their closures. We introduce systems which generate forests p-languages and then check some basic properties. This paper is a first step on our purpose to construct an algebra of regular expressions for forests p-languages.

2. Definitions and notations

Now we introduce the basic notions and notations to be used in this paper. $\mathbb{N}$ denotes the set of non-negative integers that is $\mathbb{N} = \{0,1,2,\dots\}$. The set of all positive integers is denoted by $[\omega]$. For any $n \in [\omega]$ by $[n]$ is denoted the set $\{1,\dots,n\}$. Any non-void finite set is called an alphabet. A stratified alphabet is a pair (A,r) where A is an alphabet and r is a function $r : A \longrightarrow \mathbb{N}$. For any set X /finite or not/ a free monoid over X is denoted by X^* .

Definition 1. Any finite and non-void set $V \subset [\omega]^*$ is a tree if and only if :

1^0 $\lambda \in V$ / λ - an identity of $[\omega]^*$ /

2^0 $\forall\, v \in V$, $\forall\, w \in [\omega]^*$, $\exists\, u \in [\omega]^*$ such that if $v = wu$ then $w \in V$

3^0 $\forall\, v \in V$, $\forall\, j \in [\omega]$ if $vj \in V$ then $vi \in V$ for $i \in [j]$.

Each $v \in V$ is called a node of a tree V .

Definition 2. Let $n \in [\omega]$. A forest composed of n trees is any set $W \subset [n] \times [\omega]^*$ such that for each $i \in [n]$ a set W_i is a tree where $W_i = \{ v \in [\omega]^* : iv \in W \}$.

Definition 3. The frontier of a forest W it is the following set

$$\operatorname{fr} W = \{ w \in W : w1 \notin W \} .$$

Let us assume that a pair (Σ, r) is any stratified alphabet such that the set $[p]$ is contained in $\Sigma_0 / \Sigma_0 = r^{-1}(0)$, p is any fixed positive integer/. Now we define a valued forest of (n,p) type.

Definition 4. Let $n,p \in [\omega]$. A valued forest of (n,p) type over stratified alphabet (Σ, r) is a pair $1 = (W, \eta)$ where W is any forest composed of n trees and $\eta : W \longrightarrow \Sigma$ is a function such that $r(\eta(v)) = \max \{ i \in \mathbb{N} : vi \in W \}$.

Any valued forest 1 of (n,p) type is denoted as $1 : n \longrightarrow p$. For simplicity any forest $1 : 1 \longrightarrow p$ is identified to valued tree. For any $1 : n \longrightarrow p$ t_i denotes an i-th tree of 1 . If $1 = (W, \eta)$ then $t_i = (W_i, \eta_i) : 1 \longrightarrow p$ such that $\eta_i(v) = \eta(iv)$, W_i as in definition 2. This gives a reason for the denotation $1 = (t_1, \ldots, t_n)$. For any $j \in [p]$ the symbol $\mathbf{j}$ denotes the tree $(W_j, \eta_j) : 1 \longrightarrow p$ such that $W_j = \{\lambda\}$, $\eta_j(\lambda) = j$. Λ_p denotes the forest (W, η) of (p,p) type such that $W = [p] \times \{\lambda\}$, $\eta(j) = j$ for $j \in [p]$. Hence $\Lambda_p = (\mathbf{1}, \ldots, \mathbf{p})$. To any valued forest of (n,p) type we associate some matrix representation defined below .

Definition 5. 2 Let $1 = (W, \eta) : n \longrightarrow p$ be any valued forest over (Σ, r) . A matrix representation of 1 it is an aug - mented matrix $(A;a)$ where A is a $n \times p$ matrix and a is a $n \times 1$ matrix defined for any $i \in [n]$, $j \in [p]$ as follows :

$$A_{ij} = \{ \eta(i) i_1 \ldots i_{k-1} \eta(ii_1 \ldots i_{k-1}) i_k : v = ii_1 \ldots i_k \in \operatorname{fr} W, \eta(v) = j, i_1 \ldots i_k \neq \lambda \} .$$

$\{\lambda\}$ is added to A_{ij} if $v = i$ and $\eta(v) = j$.

$$a_i = \{ \eta(i) i_1 \ldots i_{k-1} \eta(ii_1 \ldots i_{k-1}) i_k \eta(iv) : v = ii_1 \ldots i_k \in \operatorname{fr} W, \eta(v) \in \Sigma_0 - [p] \} .$$

It is easy to see that any i-th row of $A = [A_{ij}]$ represents an i-th tree of a forest 1 . For example a tree $\mathbf{j}$ has the following matrix representation $[\emptyset, \ldots, \underset{j}{\{\lambda\}}, \ldots \emptyset][\emptyset]$ and a forest Λ_p is represented by

$$\begin{bmatrix} \{\lambda\} & & \emptyset \\ & \ddots & \\ \emptyset & & \{\lambda\} \end{bmatrix} \begin{bmatrix} \emptyset \\ \vdots \\ \emptyset \end{bmatrix}$$

Now some operations on valued forests are introduced.

Definition 6. Let $l_1 = (W_1, \eta_1) : n \longrightarrow p$ and $l_2 = (W_2, \eta_2) : p \longrightarrow q$ be any valued forests. A catenation of two forests l_1 and l_2 / $l_1 \downarrow l_2$ / gives as the result a valued forest de - fined below :

$$l_1 \downarrow l_2 = (W, \eta) : n \longrightarrow q \qquad \text{where}$$

$W = W_1 - \eta_1^{-1}([p]) \cup \bigcup_{i=1}^{p} \eta_1^{-1}(i) W_{2i}$, $\eta|_{W_1 - \eta_1^{-1}([p])} = \eta_1$, and $\eta(vw) = \eta_2(w)$ for $v \in \eta_1^{-1}(i)$, $w \in W_{2i}$.

If $(A;a)$ and $(B;b)$ are matrix representations of l_1 and l_2 respectively then the matrix representation of $l_1 \downarrow l_2$ is of the form $(A \cdot B \ ; \ a + A \cdot b)$ / "+" a set theoretic sum, "." string catenation / .

Definition 7. Let $j \in [n]$ and $l_1 = (t_1, \ldots, t_n) : n \longrightarrow p$, $l_2 = (W_2, \eta_2) : p \longrightarrow q$ be any valued forests over (Σ, r) . A j-ca tenation of two forests l_1 and l_2 / $l_1 \downarrow_j l_2$ / gives as the result a valued forest defined below :

$$l_1 \downarrow_j l_2 = (t_1, \ldots, t_j \downarrow l_2, \ldots, t_n) : n \longrightarrow \max\{p,q\} .$$

As above if $(A;a)$ and $(B;b)$ are matrix representations of l_1 and l_2 the forest $l_1 \downarrow_j l_2$ has the following matrix repre - sentation :

$$\begin{bmatrix} A_1 \\ \vdots \\ A_j B \\ \vdots \\ A_n \end{bmatrix} ; \begin{bmatrix} a_1 \\ \vdots \\ a_j + A_j b \\ \vdots \\ a_n \end{bmatrix} \overset{\text{not}}{=} (A;a) \cdot_j (B;b) \quad \text{where} \quad A = \begin{bmatrix} A_1 \\ \vdots \\ A_n \end{bmatrix}, \ a = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix}$$

A catenation of forests is an associative operation but a j-ca tenation hasn't this property.

3. p-Languages and p-systems

Let $n,p \in [\omega]$ and (Σ, r) be any stratified alphabet such that $r(\sigma) \in [p]$ for any $\sigma \in \Sigma$. $F^{\Sigma}_{n,p}$ denotes the family of all valued forests of (n,p) type over (Σ, r) . In the case $n = p$ this family is denoted by F^{Σ}_{p} .

Definition 8. A p-forests language /a p-language for short/ it is any subset L of F^{Σ}_{p} .

The set F^{Σ}_{p} with a catenation of forests forms a monoid with Λ_p as an identity.

The following operations are defined on the class of p-lan guages :

1. a catenation

$$L_1 \downarrow L_2 = \{ l_1 \downarrow l_2 : l_1 \in L_1 , l_2 \in L_2 \}$$

2. a j-catenation for any $j \in [p]$

$$L_1 \downarrow_j L_2 = \{ l_1 \downarrow_j l_2 : l_1 \in L_1 , l_2 \in L_2 \}$$

3. a catenation closure

$$L^* = \bigcup_{k=0}^{\infty} X_k \quad \text{where} \quad X_0 = \{\Lambda_p\} , \quad \text{and} \quad X_{k+1} = L \downarrow X_k$$

4. a j-catenation closure for any $j \in [p]$

$$L^{*j} = \bigcup_{k=0}^{\infty} X_k^j \quad \text{where} \quad X_0^j = \{\Lambda_p\} \quad \text{and} \quad X_{k+1}^j = L \downarrow_j X_k^j$$

The equality $(L^{*i})^{*j} = (L^{*j})^{*i}$ doesn't hold.

Now we introduce a system generating a p-language.

Definition 9. Let (Σ , r) be any non-void stratified alphabet such that $[p] \subset \Sigma_c$ and $r(\sigma) \in [p]$ for any $\sigma \in \Sigma$. A p-regular system over (Σ , r) it is a 4-tuple $S = (V, \Sigma , \Gamma , P)$ where $V \cap \Sigma = \emptyset$ and

(V,s) is a nonterminal alphabet and $s(X) \in [p]$ for any $X \in V$,

$\Gamma \subset F_p^{V \cup \Sigma}$ is a finite set of initial axioms ,

$P \subset F_{1,p}^{V \cup \Sigma} \times F_{1,p}^{V \cup \Sigma}$ is a finite set of productions.

Now the process of generation in a p-regular system S is defined. We use the following denotations :

1° for any $t = (W, \eta) \in F_{1,p}^{\Sigma}$ and any $w \in W$

$t|w = (U, \tau)$

where $U = W|w = \{ u \in [\omega]^* : wu \in W \}$ and $\tau(u) = \eta(wu)$ for $u \in U$.

So $t|w$ it is a subtree of t detached at the point w .

2° for any $l = (W, \eta) \in F_p^{\Sigma}$, $l = (t_1, \ldots, t_p)$, $\bar{t} = (\bar{W}, \bar{\eta}) \in F_{1,p}^{\Sigma}$ and any $w \in W_i$

$$l(iw \leftarrow i\bar{t}) = (t_1, .., t_i|(w \leftarrow \bar{t}), .., t_p)$$

where $t_i|(w \leftarrow \bar{t}) = (U, \tau)$ and $U = \{ u \in W_i : \forall v \in [\omega]^* \; u \neq wv \} \cup w\bar{W}$

$\tau(u) = \eta_i(u)$ for $u \in W_i$, $\tau(u) = \bar{\eta}(v)$ for $u = wv$, $v \in \bar{W}$.

It means that in i-th tree of l at the point w a tree $\bar{t}$ is attached.

Definition 10. Let $l, \bar{l} \in F_p^{\Sigma}$. Let $S = (V, \Sigma , \Gamma , P)$ be any p-regular system. A forest l generates $\bar{l}$ in S / $l \xrightarrow{*} \bar{l}$/ if and only if there exist $l_1, \ldots, l_k \in F_p^{\Sigma}$ such that $l_1 = l$,

$1_k = \bar{1}$ and $1_i \longrightarrow 1_{i+1}$ in S for $i \in [k-1]$ where
$1 \longrightarrow \bar{1}$ in S iff it exists $w \in W$ and $1 \xrightarrow{w} \bar{1}$ in S and
$1 \xrightarrow{w} \bar{1}$ in S iff it exists $(t,\bar{t}) \in P$ such that $(j \downarrow 1)|w = t$,
$1 | (jw \leftarrow j\bar{t}) = \bar{1}$

Definition 11. Let $S = (V, \Sigma, \Gamma, P)$ be any p-regular system. A p-regular forests language generated by S it is a following set of p-forests :

$$L(S) = \left\{ 1 \in F_p^{\Sigma} : \exists\, 1_o \in \Gamma \ , \ 1_o \xrightarrow{*} 1 \text{ in } S \right\} .$$

Any two p-regular systems S and S´ are equivalent if and only if $L(S) = L(S')$.

4. Results

Let us consider a p-regular system $S = (V, \Sigma, \{1_o\}, P)$. In this case a p-language L(S) can be treated as cartesian product of p tree regular languages. It means that $L(S) = L_1 \times \ldots \times L_p$ and for each $i \in [p]$ $L_i = L(S_i)$ where $S_i = (V, \Sigma, \{i \downarrow 1_o\}, P)$ is a regular system in the sense of Brainerd [1] . It is essential to note that an alphabet (Σ, r) fulfils the condition $[p] \subset \Sigma_o$. This condition is very important in the definitions of operations on forests. There is no such assumption in Brainerd [1]. We emphasize that our operations on forests are different from the introduced in Thatcher [3].

The following properties of forests p-languages and p-systems are obtained /we have the use of results of [1] / :

1^o Let $L \subset F_p^{\Sigma}$. It exists a p-regular system $S = (V, \Sigma, \Gamma, P)$ such that $L = L(S)$ iff it exists a finite number of p-regular languages $L^1, \ldots, L^k$ such that $L = \bigcup_{i=1}^{k} L^i$ and for any $i \in [k]$ $L^i = L_1^i \times \ldots \times L_p^i$ where $L_j^i \subset F_{1,p}^{\Sigma}$ and is a regular tree language for $j \in [p]$. We note that it holds an equality $k = \mathrm{card}\,\Gamma$.

2^o For any p-regular system it can be effectively constructed an equivalent p-regular simple system. It means the system in which every production $(t, \bar{t})$ is of the form :
$t = (\{\lambda\}, \eta)$, $\eta(\lambda) \in V_o$ and
$\bar{t} = (\bar{W}, \bar{\eta})$, $\bar{W} = \{\lambda\} \cup [r(\bar{\eta}(\lambda))]$, $\bar{\eta}(\lambda) \in \Sigma$, $\bar{\eta}(i) \in V_o$ for $i \in [r(\bar{\eta}(\lambda))]$
Such production is written as :

$$x \longrightarrow \overset{\sigma}{\underset{x_1 \ \ldots \ x_m}{\swarrow \ \searrow}} \qquad \text{if} \quad \eta(\lambda) = x \ , \ \bar{\eta}(\lambda) = \sigma, \ m = r(\sigma) \ , \ \eta(i) = x_i \ \text{ for } i \in [m]$$

An equivalent p-regular system with a single initial axiom not always exists.

3^{o} Each singleton of F_p^{Σ} is a p-regular language.

4^{o} A class of p-regular languages is closed under the set-theoretic sum.

5^{o} A class of p-regular languages is closed under the j-catenation and its closure for any $j \in [p]$.

6^{o} A class of p-regular languages is not closed under the catenation and the catenation closure.

For the construction of the language generated by any p-regular system S one can use the matrix representation of a forest. It is possible to determine a matrix representation of each forest obtained during the process of generation using the following algorithm :

Let $S = (V, \Sigma, \Gamma, P)$ be any p-regular system and $l_o \in \Gamma$ where $l_o = (X_1^o, \ldots, X_p^o)$. Let $l_o \longrightarrow l_1 \longrightarrow \ldots \longrightarrow l_n$ be any derivation of l_n in S. A matrix representation of an initial forest l_o is $(A^o; a^o)$ defined as below

$A^o = [\phi] \;/\; A_{mj}^o = \phi$ for $m, j \in [p]/$, $a_m^o = \{X_m^o\}$ for $m \in [p]$.

Let us assume that $(A^k; a^k)$ is a matrix representation of a forest l_k where

$$A^k = \begin{bmatrix} A_1^k \\ \vdots \\ A_p^k \end{bmatrix} \quad , \qquad a^k = \begin{bmatrix} a_1^k \\ \vdots \\ a_p^k \end{bmatrix} .$$

Now we define a matrix representation $(A^{k+1}; a^{k+1})$ of a forest l_{k+1} under the assumption that l_{k+1} is obtained from l_k by a production (X, t).

1^{o} if $k < n$ then it exists $v \in a_m^k$ such that $v = \bar{v}X$ and there are possible 3 subcases :

a/ if $t = j$ for $j \in [p]$ then

$A_{mj}^{k+1} = A_{mj}^k \cup \{\bar{v}\}$, $A_{mi}^{k+1} = A_{mi}^k$ for $i \in [p]$ and $i \neq j$,

$a_m^{k+1} = a_m^k - \{v\}$

b/ if $t = \sigma$, $\sigma \in \Sigma_c$ then

$A_m^{k+1} = A_m^k$, $a_m^{k+1} = a_m^k - \{v\} \cup \{\bar{v}\sigma\}$

c/ if $t = \sigma$ (with $x_1 \ldots x_s$), $s = r(\sigma) \geq 1$ then

$A_m^{k+1} = A_m^k$, $a_m^{k+1} = a_m^k - \{v\} \cup \{\bar{v}1X_1, \ldots, \bar{v}sX_s\}$

2^{o} if $k = n$ then for any $v \in a_m^k$ it holds $v = \bar{v}\sigma$ or $a_m^k = \emptyset$.

This paper is a first step on our attempts to construct the regular expressions for forests p-languages. We consider an abstract algebra RE_f with a set theoretic sum, a j-catenation and a j-catenation closure as operations and with a support composed of some special kind of 2p-forests /not trivial only in one place of their 2p-tuples/. Our preliminary investigations suggest the following :

Conjecture : The algebra RE_f is the algebra of regular expressions for forests p-languages .

REFERENCES

[1] Brainerd W.S. Tree Generating Regular Systems, Inf.Control 14 , 1969

[2] Elgot C.C.,BloomS.L.,Tindell R. On the Algebraic Structure of Rooted Trees , J.Comp.Syst.Sc 16 , 1978

[3] Thatcher J.W. , Wright J.B. Generalized Finite Automata Theory with an Application... Math.Syst.Th. 2 ,1968

COFINITE LANGUAGES - A FIXED POINT CHARACTERIZATION

Wit Forys

Institute of Computer Science
Jagellonian University
Kopernika 27
Kraków
Poland

We consider the family of cofinite languages over some alphabet A and give a fixed point characterization of it in the following way. We define the family $\mathcal{F}_A$ of functions such that the correspondent family of fixed point languages is equall COFIN A . Then we define two abstract algebras with COFIN A and $\mathcal{F}_A$ as supports. It is established some morphical interdependence beetwen these two algebras.

The following notations will be used. Let A denotes an alphabet that is any finite non-empty set. A^* is a free monoid generated by A .A language $L \subset A^*$ such that $A^* - L$ is finite is called a cofinite language. The class of all cofinite languages over A is denoted as COFIN A . For any $L \subset A^*$ by $\mathrm{card}(L)$ is denoted its cardinality. The cardinality of the set of integers is denoted as $\aleph_0$. For any function $f:A^* \longrightarrow A^*$ and any $L \subset A^*$ the restriction of f to L is denoted $f|_L$.

Definition 1. The class $\mathcal{F}_A$ consists of all functions $f:A^* \longrightarrow A^*$ such that for any $L \subset A^*$ for which $\mathrm{card}(L) = \aleph_0$ it holds $f(L) \cap L \neq \emptyset$.

Definition 2. Let $f:A^* \longrightarrow A^*$ be any function. The set of all fixed words of f is defined as

$$\mathrm{Fp}\ f = \{ w \in A^* : f(w) = w \} .$$

THEOREM 1. Let A be any alphabet. The class COFIN A is identical to the class $\mathrm{Fp}\ \mathcal{F}_A$.

Proof. Let us assume,for an indirect proof, that for some $f \in \mathcal{F}_A$ it holds $\mathrm{card}(A^* - \mathrm{Fp}\ f) = \aleph_0$. So it is possible to

define the following sequence :
$w_1 \in A^* - \mathrm{Fp}\ f$ and
$w_{n+1} \in (A^* - \mathrm{Fp}\ f) - \bigcup_{i=1}^{n} f(w_i)$, $f(w_{n+1}) \neq w_i$ for $i = 1,\ldots,n$
This sequence of words may be finite or not. In the first case it follows that there exists an infinite language $L \subset (A^* - \mathrm{Fp}\ f) - \bigcup_{i=1}^{N} f(w_i)$ mapped by f on one of the words of this finite sequence $(w_1,\ldots,w_N)$. For this language L the equallity $L \cap f(L) = \emptyset$ holds. In the second case it is obtained an infinite sequence of words from $A^* - \mathrm{Fp}\ f$. It forms an infinite language L such that $L \cap f(L) = \emptyset$. Hence in both cases we come to a contradiction. So the language $\mathrm{Fp}\ f$ belongs to the class COFIN A .

Conversely let L belongs to COFIN A with $A^* - L = \{u_1,..,u_n\}$. If $n > 1$ it is obvious that the function f_L defined below belongs to the class $\mathcal{F}_A$ and that $\mathrm{Fp}\ f = L$:

$$f_L(w) = \begin{cases} w & \text{if } w \in L \\ u_{i+1 (\mathrm{mod}\ n)} & \text{if } w = u_i \notin L \end{cases}$$

If $n = 1$ we can put $f(w) = v$ where v is any word from L for $w = u_1$ and $f(w) = w$ for each $w \in L$. Of course in this case $f \in \mathcal{F}_A$ and $\mathrm{Fp}\ f = L$ also. So the proof is complete.

Let us define two abstract algebras :
$\mathrm{COFIN} = \langle \mathrm{COFIN\ A} ; \cup, \cap, *, \leftarrow \rangle$ and
$\mathcal{F} = \langle \mathcal{F}_A ; \square, \circ, \circledast, \leftrightarrow \rangle$ in the following way. Symbols $\cup$, $\cap$ denote respectively set-theoretical sum and intersection that is two binary operations on COFIN A and $*$, $\leftarrow$ denote Kleene star and reversal - unary operations on COFIN A . In the algebra $\mathcal{F}$ symbols $\square$, $\circ$ denote binary operations defined on $\mathcal{F}_A$ as follows:

for any f , $g \in \mathcal{F}_A$ such that $\mathrm{Fp}\ f = L_1$, $\mathrm{Fp}\ g = L_2$

$$f \square g = f|_{L_1} \cup g|_{L_2} \cup f|_Z \qquad \text{where } Z = A^* - (L_1 \cup L_2) ,$$

$$f \circ g = (f \cap g) \cup f|_{Z_1} \cup g|_{Z_2 - (Z_1 \cap Z_2)} \cup f|_{L_2 - D} \cup g|_{L_1 - D}$$

where $Z_1 = A^* - L_1$, $Z_2 = A^* - L_2$, $D = \{ w \in A^* : f(w) = g(w) \}$.
Two unary operations $\circledast$ and $\leftrightarrow$ are defined in $\mathcal{F}$ as below.
For any $f \in \mathcal{F}_A$ we put
$f^{\circledast} = \bigcup_{n=0}^{\infty} f^n|_{L^*} \cup f|_{A^* - L^*}$
where $f^0 = \{(1,1)\}$ and $f^n(w) = f(w_1) \ldots f(w_n)$ if there are existed $w_i \in L$ such that $w = w_1 \ldots w_n$ /for any other $v_j \in L$ such that

$w = v_1 \ldots v_m$ the value of f^n is the same / .

For any $f \in \mathcal{F}_A$ $\overleftrightarrow{f}$ is a function defined as $\overleftrightarrow{f}(w) = \overleftarrow{f}(w)$.

It is obvious now that the function $H : \mathcal{F} \longrightarrow$ COFIN defined for any $f \in \mathcal{F}$ as $H(f) = Fp\ f$ is an epimorphism of two defined above algebras. Now let us define a binary relation ϱ on the set $\mathcal{F}_A$ the support of $\mathcal{F}$ in the following way:

for any $f, g \in \mathcal{F}_A$ we put

$$f \varrho g \quad \text{iff} \quad Fp\ f = Fp\ g \ .$$

It is easy to check that ϱ is a congruention. So it is possible to define a quotient algebra $\mathcal{F}/\varrho$ and to formulate :

THEOREM 2. The algebras $\mathcal{F}/\varrho$ and COFIN are isomorphic.

ITERATIVE SEMIRINGS WITH DIVERGENCE

Karol Habart

Technische Universitat Dresden
Sektion Mathematik
Dresden
DDR

0. I n t r o d u c t i o n

Two specific classes of semirings are to be concerned in this paper, the class of semirings with divergence and the class of the iterative semirings. The definition of these structures is justified by the effort to describe the semantics of a programming language as an algebra. Considering the Dijkstra-language (a simple theoretical language) one can see, that its semantics can be described by the set of all correspondences in $\underline{Z}$, where $\underline{Z}$ is the set of all states, i.e. the set of all mappings from the set of all identifiers into the set $\mathbb{Z}$ of integers. (Assumpted, that using the Dijkstra-language one can use integers only.) The following operations in $\underline{Z}$ are needed in this describtion:

- The composition of correspondences:

0.1. $\varrho_1 \circ \varrho_2 := \{[x,y] \in \underline{Z} \times \underline{Z} \mid \exists z \in \underline{Z} ([x,z] \in \varrho_1 \wedge [z,y] \in \varrho_2)\}$

- The union of correspondences:

0.2 $\varrho_1 \cup \varrho_2 := \{[x,y] \in \underline{Z} \times \underline{Z} \mid [x,y] \in \varrho_1 \vee [x,y] \in \varrho_2\}$

- The divergence of correspondences:

0.3. $\mathrm{Div}\,\varrho := \{[x,x] \mid \forall y \in \underline{Z} ([x,y] \notin \varrho)\}$

- The iteration of correspondences, i.e. the transitive reflexive closure of correspondences:

0.4. $\varrho^* := \bigcup\{\varrho^n \mid n \in \mathbb{N}\}$ where $\varrho^0 := 1 := \{[x,x] \mid x \in \underline{Z}\}, \varrho^{n+1} := \varrho^n \circ \varrho$

The set $\underline{Z}$ is together with the operations composition and union a semiring with 0 and 1. Therefore one can generalize the operations divergence and iteration and define them in an arbitrary semiring by axioms.

1. Semirings with divergence

Definition: The semiring R with 0 and 1 is to be called a semiring with divergence if and only if there is an operation div. (divergence) in R and the following rules are holding:

D1 $\quad \operatorname{div} a \,.\, a = 0 \qquad (a \in R)$

D2 $\quad \operatorname{div} a + \operatorname{div} \operatorname{div} a = 1 \qquad (a \in R)$

D3 $\quad \operatorname{div}(a.\operatorname{div} \operatorname{div} b) = \operatorname{div}(a.b) \qquad (a,b \in R)$

The axioms D1, D2, D3 are independent. The following propositions are holding:

1.1. $\quad \operatorname{div} \operatorname{div} a \,.\, a = a \qquad (a \in R)$

1.2. $\quad \operatorname{div} 0 = 1 \quad \wedge \quad \operatorname{div} 1 = 0$

1.3. $\quad \operatorname{div} x = 1 \quad \Rightarrow \quad x = 0 \qquad (x \in R)$

1.4. $\quad \operatorname{div}^{2n-1} a = \operatorname{div} a, \ \operatorname{div}^{2n} a = \operatorname{div} \operatorname{div} a \qquad (a \in R, n \in \mathbb{N})$

Theorem 1: In the semiring of all correspondences $\varphi \subset \underline{Z} \times \underline{Z}$ there is no operation holding D1,D2,D3 different from Div. Defined in 0.3.

Proof: Let a be an arbitrary element of the Boolean $\mathcal{B}(\underline{Z} \times \underline{Z})$ of $\underline{Z} \times \underline{Z}$. According to D2 is $\operatorname{div} a \subset 1$, i.e. $\operatorname{div} a = \{[x,x] \mid x \in A\}$ where $A \subset \underline{Z}$ is well-defined. On D1 follows:

1.5. $\quad \operatorname{div} a \subset \operatorname{Div} a$

By D1 and D3 holds:
$\operatorname{div}(\operatorname{Div} a \,.\, \operatorname{div} \operatorname{div} a) = \operatorname{div}(\operatorname{Div} a \,.\, a) = \operatorname{div}(0) = 1.$
On 1.3. is then:
$\operatorname{Div} a \,.\, \operatorname{div} \operatorname{div} a = 0$
and by D2:
$\operatorname{Div} a = \operatorname{Div} a \,.\, \operatorname{div} a,$
hence
$\operatorname{Div} a + \operatorname{div} a = \operatorname{Div} a \,.\, \operatorname{div} a + \operatorname{div} a = \operatorname{div} a$
and hence $\operatorname{Div} a \subset \operatorname{div} a$ and then follows according to 1.5.:

$\operatorname{Div} a = \operatorname{div} a.$

q.e.d.

Let R be an idempotent semiring, i.e. let it be for every $a \in R$, that:

$a + a = a.$

Then is R ordered by the relation:

$a < b \quad :\Leftrightarrow \quad a + b = b.$ $\qquad (a,b \in R)$

Lemma 1: Let R be an idempotent semiring with divergence. Then:

$\operatorname{div} a < \operatorname{div} b \Leftrightarrow \operatorname{div} a \cdot \operatorname{div} b = \operatorname{div} a \Leftrightarrow$

$\Leftrightarrow \operatorname{div} b \cdot \operatorname{div} a = \operatorname{div} a.$ $\qquad (a,b \in R)$

Lemma 2: In an idempotent semiring with divergence holds:

$a > b \Rightarrow \operatorname{div} a < \operatorname{div} b.$

Theorem 2: In an idempotent semiring with divergence holds:

1.6. $\operatorname{div}(a + b) = \operatorname{div} a \cdot \operatorname{div} b.$ $\qquad (a,b \in R)$

Proof: By lemma 2 holds:

$\operatorname{div}(a + b) < \operatorname{div} a \quad \wedge \quad \operatorname{div}(a + b) < \operatorname{div} b.$

Hence follows on D2:

$\operatorname{div}(a+b) + \operatorname{div} a \cdot \operatorname{div} b = \operatorname{div}(a + b)(\operatorname{div} b + \operatorname{div} \operatorname{div} b) +$

$+ \operatorname{div} a \cdot \operatorname{div} b = (\operatorname{div}(a + b) + \operatorname{div} a) \cdot \operatorname{div} b +$

$+ \operatorname{div}(a + b)\operatorname{div} \operatorname{div} b = \operatorname{div} a \cdot \operatorname{div} b + \operatorname{div}(a + b)\operatorname{div} \operatorname{div} b <$

$< \operatorname{div} a \cdot \operatorname{div} b + \operatorname{div} b \cdot \operatorname{div} \operatorname{div} b.$

According to D1 follows then:

$\operatorname{div}(a + b) + \operatorname{div} a \cdot \operatorname{div} b < \operatorname{div} a \cdot \operatorname{div} b$

hence:

1.7. $\operatorname{div}(a + b) < \operatorname{div} a \cdot \operatorname{div} b.$

On the other hand, on axiom D1 and lemma 2:

$\operatorname{div} a \cdot \operatorname{div} b \cdot (a + b) = \operatorname{div} a \cdot \operatorname{div} b \cdot a + \operatorname{div} a \operatorname{div} b \cdot b =$

$= \operatorname{div} a \cdot \operatorname{div} b \cdot a < \operatorname{div} a \cdot a = 0,$

i.e.:

$\operatorname{div} a \cdot \operatorname{div} b \cdot (a + b) = 0.$

Hence follows on D3:

$\operatorname{div}(\operatorname{div} a \cdot \operatorname{div} b \cdot \operatorname{div} \operatorname{div}(a + b)) =$

$= \operatorname{div}(\operatorname{div} a \cdot \operatorname{div} b \cdot (a + b)) = \operatorname{div}(0) = 1$

and then follows on 1.3.:

$\operatorname{div} a \cdot \operatorname{div} b \cdot \operatorname{div} \operatorname{div}(a + b) = 0.$

Hence follows by D2:

$\operatorname{div} a \cdot \operatorname{div} b = \operatorname{div} a \cdot \operatorname{div} b \cdot \operatorname{div}(a + b),$

i.e. according to lemma 1 holds:

$\operatorname{div} a \cdot \operatorname{div} b < \operatorname{div}(a + b).$

Because of 1.7. is then:

$\operatorname{div} a \cdot \operatorname{div} b = \operatorname{div}(a + b).$

q.e.d.

Notion: In idempotent semirings with divergence holds then:

1.8. div a . div a = div a. $(a \in R)$

Lemma 3: In semirings with divergence holding 1.6. holds:

div div(a + b) =

= div(div(a + b) + a) + div(div(a + b) + div a) =

= div(div(a + b) + b) + div(div(a + b) + div b).

Theorem 3: In an idempotent semiring with divergence holds:

div div(a + b) = div div a + div div b.

Proof: According to lemma 3 and 1.8. holds:

div div(a + b) =

(div(div(a + b) + a) + div(div(a + b) + div a)).

. (div(div(a + b) + b) + div(div(a + b) + div b)).

Hence by theorem 2:

div div(a + b) = div(div(a + b) + a + b) +

+ div(div(a + b) + a + div b) + div(div(a + b) + div a + b) +

+ div(div(a + b) + div a + div b).

Then it follows on theorem 2 again:

div div(a + b) = div div(a + b).div(a + b) +

+ div(div a . div b + a + div b) +

+ div(div a. div b+ div a + b) +

+ div(div a . div b + div a + div b).

According to lemma 1 and axiom D1 follows then:

div div(a + b) = div(a + div b) +

+ div(div a + b) + div(div a + div b).

Hence by theorem 2:

div div(a + b) = div a . div div b + div div a . div b +

+ div div a . div div b = (div a + di div a)div div b +

+ div div a . (div b + div div b).

According axiom D2 follows at least:

div div(a + b) = div div a + div div b.

q.e.d.

Notion: In the same (idempotent) semiring more than one operation divergence (holding D1,D2,D3) can exist. The image div [R] of the mapping "divergence" is a Boolean algebra in an idempotent semiring, where:

$\bar{a}^1$:= div a. $(a \in R)$

2. Iterative semirings

Definition: The semiring R with 0 and 1 is to be called iterative semiring if and only if there is an operation $.^{\circ}$ (iteration) in R and following rules are holding:

I1 $(a.b)^{\circ} = 1 + a.(b.a)^{\circ}.b$ $\quad (a,b \in R)$

I2 $(a + b)^{\circ} = (a^{\circ}.b)^{\circ}.a^{\circ}$ $\quad (a,b \in R)$

I3 $(a^{\circ})^{\circ} = a^{\circ}$ $\quad (a \in R)$

The axioms I1, I2 and I3 are independent!

Let Rel denote the semiring of all correspondences $\rho \subset Z \times Z$. Let $.^{\circ}$ be an iteration in Rel holding I1,I2,I3. Let $.^{*}$ denote the operation iteration defined in 0.4. Then holds:

Lemma 4: For arbitrary $a \in$ Rel and $b := \{[x,y]\} \in$ Rel holds:

$(ab)^{\circ} = 1 + ab.$

Proof: It holds:

2.1. $ab = \{[z,y] \mid z \in A\}$ with well-defined $A \subset Z$.

2.2. $(ab)^{\circ} ab = \{[z,y] \mid z \in B\}$ with well-defined $B \subset Z$.

According to axiom I1:

2.3. $(ab)^{\circ} = 1 + ab(ab)^{\circ} = 1 + (ab)^{\circ} ab = 1 + ab + abab(ab)^{\circ}$.

Using 2.1., 2.2. and 2.3. one can show, that if $[u,v] \in (ab)^{\circ}$ and $u \neq v$, then the following implications are holding:

2.4. $\begin{cases} [u,v] \in (ab)^{\circ} ab \Rightarrow v = y \\ [u,v] \in ab(ab)^{\circ} \Rightarrow u \in A. \end{cases}$

The pair $[u,y]$ is element of $(ab)^{\circ}$ for arbitrary $u \in A$, because $(ab)^{\circ} = 1 + ab + abab(ab)^{\circ}$ (see 2.3.) Therefore:

2.5. $(ab)^{\circ} \supset ab.$

On 2.1.,2.3.,2.4. and 2.5. follows:

$(ab)^{\circ} = 1 + ab.$

q.e.d.

Lemma 5: Let $M \subset$ Rel be defined as follows:

$M := \{a \in \text{Rel} \mid \forall p(p \subset a \Rightarrow p^{\circ} = p^{*})\}.$

Then holds:

$(a \in M \wedge [x,y] \in Z \times Z \wedge [x,y] \notin a) \Rightarrow (a \cup \{[x,y]\} \in M).$

Proof: Let be: $= \{[x,y]\} \in \underline{Rel}$, $q \subset a + b$ be arbitrarily choosed. If $[x,y] \notin q$, then $q \subset a$ and because of $a \in M$ follows $q^{\circ} = q^{*}$. This holds also if $[x,y] \in q$, because then $q = p + b$ where $p \subset a$ is well-defined. Using axiom I2 follows:

$$q^{\circ} = (p + b)^{\circ} = (p^{\circ}b)^{\circ}p^{\circ} = (p^{*}b)^{\circ}\, p^{*}$$

hence:

$$q^{\circ} = (p^{*}b)^{*}p^{*} = (p + b)^{*} = q^{*}$$

because by lemma 4 follows $(p^{*}b)^{\circ} = (p^{*}b)^{*}$.

q.e.d.

Theorem 4: In the semiring Rel no operation holding I1,I2, I3 is different from $.^{*}$ defined in 0.4., i.e.:

$\forall a \in \underline{Rel}(a^{\circ} = a^{*})$.

Proof: The set M defined in lemma 5 is not empty, because $0^{\circ} = 0^{*} = 1$. Furthermore, the set M is ordered by inclusion in that way, that Zorn's lemma can be applied. Therefore there exist a maximal element a of M and because of lemma 5 is $a = \underline{Z} \times \underline{Z}$. Then for arbitrary $p \subset \underline{Z} \times \underline{Z}$ holds:

$$p^{\circ} = p^{*}$$

q.e.d.

Notion: Generally there can exist more than one iteration holding I1,I2,I3 in a (idempotent) semiring.

STOCHASTIC TABLE LINDENMAYER SYSTEMS

J. Hromkovič

Department of Theoretical Cybernetics
Komensky University
842 15 Bratislava
Czechoslovakia

J. Mertan

Department of Microelectronics
USIP Trenčín
911 05 Trenčín
Czechoslovakia

1. Introduction

Theory of Lindenmayer systems,also referred to as theory of developmental systems,is the result of an infusion of ideas from developmental biology into formal language theory.This theory has become a very actively investigated topic in the last few yers,and it has produced many results of interest both for the formal language theorists and the ones modelling various types of real systems,briefly L systems,were introduced by Lindenmayer [6].We shall study the stochastic versions of L systems introduced by Jürgensen [4,5] and Schäffler [7] in this paper.

We introduce three versions of stochastic table OL systems. The motivation for introduction the table OL systems,called TOL systems,was in the fact that the developmental behavior of many organisms depends on environmental conditions.To describe the development of such organism one has to provide different sets

of developmental rules corresponding to the different environmental conditions.

The stochastic versions of TOL systems introduced here differ from TOL systems in two ways.Instead of a single axiom the stochastic TOL systems have a probability distribution over a finite set of possible axioms.In a STSOL system we add a probability distribution to the set of tables and each table has a probability distribution over the rewrite rules in the same way as stochastic OL system.Clearly,one of the meanings of introduction stochastic TOL systems can be the same as the motivation for introduction stochastic OL systems.But,the most important meaning which we consider is in the fact that the occurence probability of various types of environmental conditions can be different.And this fact is simply described in the probability distribution over the tables of an STSOL system.

This paper is organized as follows.Section 2 contains the basic definitions for OL systems,TOL systems and their stochastic versions.Topics including techniques for computing of a growth functions of a stochastic L systems are in Section 3 and hierarchy of the classes of growth functions defined by stochastic L systems is studied in Section 4.Synthesis of growth functions and classification of growth rate are investigated in Section 5 and Section 6 respectively.Section 7 involves the results concerning several types of growth equivalences.

2. Basic Definitions

In order to establish our notation,we will rewiev the definitions for OL systems.

DEFINITION.An OL scheme is a pair $G=\langle\sum,R\rangle$ where $\sum$ is a finite set of symbols called alphabet and $R\subseteq\sum\times\sum$ is a finite set of rewrite rules/called productions/ such that for each a in $\sum$ there exists at least one α in $\sum^*$ that (a,α) is in R.We use $a\xrightarrow[G]{}\alpha$ to denote (a,α) is in R.

DEFINITION.A OL system is a triple $S=\langle\sum,R,x\rangle$ where $\langle\sum,R\rangle$ is OL scheme and x is an element of $\sum^*$ called the axiom.

DEFINITION.A TOL scheme is a pair $G=\langle\sum,\mathcal{R}\rangle$ where $\sum$ is a finite

nonempty set called the alphabet, $\mathcal{R}$ /the set of tables/ is a finite ,nonempty set.Each element R of $\mathcal{R}$,called a table,is a finite nonempty subset of $\sum\times\sum^*$ such that for all a in $\sum$ there exists α in $\sum^*$ that (a,α) belongs to R.We use $a\underset{G}{\rightarrow}\alpha$ or $a\underset{R}{\rightarrow}\alpha$ to denote (a,α) is in R.

DEFINITION.A TOL system is a triple $S=\langle\sum,\mathcal{R},x\rangle$,where $G=\langle\sum,\mathcal{R}\rangle$ is a TOL scheme and x,called the axiom of S,is a word over $\sum^*$.

Since the OL systems are the special cases of TOL systems which have only one table we define the following notion only for TOL systems.

DEFINITION.Let $G=\langle\sum,\mathcal{R}\rangle$ be a TOL scheme,let $x=a_1\ldots a_m$ with $m\geq 1$ and a_j in $\sum$ for $j=1,\ldots,m$ and let y be in $\sum^*$.Thenwe say that <u>x directly derives y</u> in G,denoted as $x\underset{G}{\Rightarrow}y$,if and only if there exists R in $\mathcal{R}$ such that there exists $\alpha_1,\ldots,\alpha_m$ in $\sum^*$ such that $a_1\underset{R}{\rightarrow}\alpha_1,\ldots,a_m\underset{R}{\rightarrow}\alpha_m$ and $y=\alpha_1\ldots\alpha_m$.In this case we also write $x\underset{R}{\Rightarrow}y$.

DEFINITION.For any TOL scheme $G=\langle\sum,\mathcal{R}\rangle$,for any word x in $\sum^*$ and for any nonnegative integer n,we define the finite language $L_n(G,x)$ by induction on n.

$L_0(G,x)=\{x\}$

$L_{n+1}(G,x)=\{y \mid$ there exists z in $L_n(G,x)$ and R in $\mathcal{R}$ such that $z\underset{R}{\Rightarrow}y\}$

We say that <u>x derives y</u> in $G/x\overset{*}{\underset{G}{\Rightarrow}}y$ or only $x\overset{*}{\Rightarrow}y/$ if y is in $L_n(G,x)$ for some n.If y is in $L_k(G,x)$ we write $x\overset{k}{\underset{G}{\Rightarrow}}y/x\overset{k}{\Rightarrow}y/$.

DEFINITION.Let $S=\langle\sum,\mathcal{R},x\rangle$ be a TOL system.<u>The language generated by S</u>,denoted L(S),is defined as $L(S)=\{y \mid x\overset{*}{\underset{S}{\Rightarrow}}y\}$.

Now we give the basic definition for stochastic L systems.

DEFINITION.<u>Astochastic OL scheme</u>,SOL scheme,is a triple $\langle\sum,R,h\rangle$ where $\langle\sum,R\rangle$ is a OL scheme and h is mapping from $\sum$ to the half open interval $(0,1\rangle$ of real numbers such that,for each a in $\sum$, $\sum_{\alpha}h(a\rightarrow\alpha)=1$,where the sum is over all productions with a on the left-hand side.<u>A stochastic OL system</u> ,SOL system,is a four tuple $S=\langle\sum,R,h,\omega\rangle$,where $\langle\sum,R,h\rangle$ is SOL scheme and ω,called <u>axiom distribution</u> is a mapping from $\sum^+$ to the closed interval $\langle 0,1\rangle$ of real numbers such that there are only finitely many x in $\sum^+$ with $\omega(x)>0$ and such that $\sum_x\omega(x)=1$,where the sum is over all x in $\sum^+$.

DEFINITION. A stochastic TOL scheme, STOL scheme, is a triple $G=\langle \sum, \mathcal{R}, H \rangle$ where $\langle \sum, \mathcal{R} \rangle$ is TOL scheme and H called the probability distribution on $\mathcal{R}$ is a mapping from $\mathcal{R}$ to half open interval $(0,1\rangle$ such that $\sum_R H(R)=1$, where the sum is over all R in $\mathcal{R}$. A stochastic TOL system, STOL system, is four tuple $S=\langle \sum, \mathcal{R}, H, \omega \rangle$ where $\langle \sum, \mathcal{R}, H \rangle$ is a TOL scheme and ω is a axiom distribution over $\sum^+$.

DEFINITION. A table stochastic OL scheme, TSOL scheme, is a triple $G=\langle \sum, \mathcal{R}, \mathcal{H} \rangle$ where $\langle \sum, \mathcal{R} \rangle$ is TOL scheme and $\mathcal{H}$ is finite, nonempty set such that if $\mathcal{R}=\{R_1, R_2, \ldots, R_k\}$ then $\mathcal{H}=\{h_1, h_2, \ldots, h_k\}$ and $\langle \sum, R_i, h_i \rangle$ is SOL scheme for $i=1,\ldots,k$. A table stochastic OL system, TSOL system, is a four tuple $S=\langle \sum, \mathcal{R}, \mathcal{H}, \omega \rangle$ where $\langle \sum, \mathcal{H}, \omega \rangle$ is TSOL scheme and ω is axiom distribution over $\sum^+$.

DEFINITION. A substochastic TOL scheme, STSOL scheme, is a four tuple $G=\langle \sum, \mathcal{R}, H, \mathcal{H} \rangle$ where $\langle \sum, \mathcal{R}, H \rangle$ is a STOL scheme and $\langle \sum, \mathcal{R}, \mathcal{H} \rangle$ is a TSOL scheme. A substochastic TOL system, STSOL system, is five tuple $S=\langle \sum, \mathcal{R}, H, \mathcal{H}, \omega \rangle$ where $\langle \sum, \mathcal{R}, H, \mathcal{H} \rangle$ is STSOL scheme and ω is a axiom distribution over $\sum^+$.

We define following notion only for STSOL system because SOL, STOL and TSOL systems are special cases of STSOL system and so the notion defined for STSOL system will be defined by this way for other stochastic systems too.

DEFINITION. Let $G=\langle \sum, \mathcal{R}, H, \mathcal{H} \rangle$ be a STSOL scheme and $d=w_1 \overset{P_1}{\Rightarrow} w_2 \overset{P_2}{\Rightarrow} \ldots \overset{P_{n-1}}{\Rightarrow} w_n$ is a derivation in $\langle \sum, \mathcal{R} \rangle$. Let $w_i=a_1 \ldots a_k$ and by the derivation $w_i \overset{P_i}{\Rightarrow} w_{i+1}$ are used the productions $a_1 \overset{P_i}{\rightarrow} \alpha_1, \ldots, a_k \overset{P_i}{\rightarrow} \alpha_k$. We define

$$P(w_i \overset{P_i}{\Rightarrow} w_{i+1}) = H(P_i) \cdot \prod_{j=1}^{k} h_i((a_j, {}_j))$$

and the probability that $w_1 \overset{*}{\Rightarrow} w_n$ according the derivation d is $P(w_1 \overset{*}{\Rightarrow} w_n, d) = \prod_{i=1}^{n-1} P(w_i \overset{P_i}{\Rightarrow} w_{i+1})$. Let $T(n,x,y)$ denote the set of all derivation of y from x having lenght exactly n. Let x,y be in $\sum^+$. The probability of deriving y from x in exactly n steps is denoted $P(n,x,y)$ and defined $P(n,x,y)=\sum_d P(x \overset{*}{\Rightarrow} y, d)$, where the sum is over all d in $T(n,x,y)$.

DEFINITION. Let $S=\langle \sum, \mathcal{R}, H, \mathcal{H}, \omega \rangle$ be a STSOL system. Let $y_1, y_2, \ldots, y_m$ be all words over $\sum^+$ such that $\omega(y_i)>0$ and let x be an element of $\sum^*$. The probability of deriving x from ω in exactly n steps is

denoted $P(n,\omega,x)$ and defined by

$$P(n,\omega,x) = \sum_{i=1}^{m} \omega(y_i) \cdot P(n,y_i,x) .$$

Then the growth function of S, denoted f_S, is defined for $n \geq 0$ as follows: $f_S(n) = \sum_{x} |x| R(n,\omega,x)$, where the sum is over all x in $\sum^*$ and $|x|$ denoted the lenght of x. So f_S is a mapping from natural numbers to the nonnegative reals such that $f_S(n)$ is expected word lenght after n steps of a derivation of S.

3. Analysis of Growth Functions

Before we give the algorithms for computing of growth functions we define still some important notions.

DEFINITION. Let $G=\langle \sum, \mathcal{R}, H, \mathcal{H} \rangle$ be a STSOL scheme and a be in $\sum$, α in $\sum^*$, R_i in $\mathcal{R}$ for $i=1,\ldots,k$. We define $p(a \xrightarrow[R_j]{} \alpha) = H(R_j) \cdot h_j((a,\alpha))$ if (a,α) is in R_j and $p(a \xrightarrow[R_j]{} \alpha) = 0$ if (a,α) does not belong to R_j. Then we determine the probability of use of the production $a \longrightarrow \alpha$ by

$$p(a \longrightarrow \alpha) = \sum_{i=1}^{k} p(a \xrightarrow[P_i]{} \alpha) .$$

DEFINITION. Let $S=\langle \sum, \mathcal{R}, H, \mathcal{H}, \omega \rangle$ be a STSOL system and $\sum = \{a_1,\ldots,a_t\}$. Let $\#a_j(x)$ be the number of occurences a_j in x. <u>The Parikh vector of x</u>, denoted v(x), is the following t dimensional row vector $(\#a_1(x),\ldots \#a_t(x))$. <u>The growth matrix of S</u>, denoted $G_S=(g_{ij})$, is the t by t matrix where $g_{ij} = \sum_{\alpha} \#a_j(\alpha) \cdot p(a_i \longrightarrow \alpha)$, where the sum is over all α such that $a_i \longrightarrow \alpha$ is in $\bigcup_{R \in \mathcal{R}} R$.

Let $y_1, y_2, \ldots, y_m$ be an enumeration of all strings y_i in $\sum^+$ such that $\omega(y_i) > 0$. <u>The initial vector of S</u> - z_ω is defined as the following sum of Parikh vectors.

$$z_\omega = \sum_{i=1}^{m} \omega(y_i) \cdot v(y_i) .$$

<u>Thesum vector $\overline{1}$</u> is t-dimensional column vector with all components equal to 1.

Our first assertion is a extension of the analysis theorem for deterministic OL systems made by Herman and Rozenberg [3].

THEOREM 1. For any STSOL system $S=\langle \sum, \mathcal{R}, H, \mathcal{H}, \omega \rangle$ and any natural number n, the value $f_S(n)$ of the growth function at n can be computed by

$$f_S(n) = z_\omega \cdot G_S^n \cdot \overline{1} .$$

The next theorem gives us the posibility to compute the growth functions of STSOL systems by means of algorithms made for growth functions of SOL systems.

THEOREM 2.For any STSOL system S there exists an algorithm which constructs SOL system S′such that $f_S(n)=f_{S'}(n)$ for all natural n.

The next two results were shown by Eichhorst and Savitch 2 for SOL systems and using Theorem 2 we can formulated them for STSOL systems.

THEOREM 3.There is an algorithm which,given any STSOL system S with t symbols,will compute a homogeneous difference equation of order t,with real coefficient,such that the growth function f_S of S satisfies this equation.

THEOREM 4.Let f be a function known to be realizable as the growth function of some STSOL system with at most t symbols. There is an algorithm which ,given the first 2t values of such an f,will construct a linear reccurence equation for f such that all values of f can be computed from these initial values and the recurrence formula.

THEOREM 5.There is an algorithm which,given a STSOL system S with t symbols,will construct a linear recurrence equation of the degree at most t such that the growth function f_S of S fulfils this equation.

4. Hierarchy of Growth Functions

In this section we study the relation between the classes of of growth functions determined by stochastic L systems.Let us first define some basic notion.

DEFINITION.Let A be some class of L systems/for example OL or STOL/ .Then by $\mathcal{F}(A)$ we denote the class of all functions which can be obtained as growth functions of arbitrary A systems,i.e. $\mathcal{F}(A)=\{f_S \mid S\in A\}$.

THEOREM 6.There is an algorithm,given any SOL system,will construct a STOL system S′such that $f_S(n)=f_{S'}(n)$ for all natural n.

THEOREM 7. $\mathcal{F}(SOL) = \mathcal{F}(STOL) = \mathcal{F}(TSOL) = \mathcal{F}(STSOL)$.

We conclude this section by two theorems which show the

relation between the growth functions of stochastic and nonstochastic L systems.

THEOREM 8.There exists such stochastic L system S that the growth function cannot be a growth function of any nonstochastic L system.

THEOREM 9.There exists such nonstochastic L system S that the growth function f_S of such system cannot be a growth function of any STSOL system.

5. Synthesis of Growth Functions

Now we introduce a theorem whose results we prove using the results made by Eichhorst and Savitch [2].This theorem study the closure properties of the growth functions of stochastic L systems too.

THEOREM 10. (1) Let c be arbitrary natural number and f be in $\mathcal{F}$(SOL).Then the function $g(n)= f(n)^c$ is in $\mathcal{F}$(SOL).

(2) If $f_1,f_2,\ldots,f_t$ are in $\mathcal{F}$(SOL) and $c_1,c_2,\ldots,c_t$ are nonnegative real numbers such that $\sum_{i=1}^{t} c_i \geq 1$,then the function $f(n)=\sum_{i=1}^{t} c_i f_i(n)$ is in $\mathcal{F}$(SOL).

(3) Let g be in $\mathcal{F}$(SOL) and let c be a nonnegative real number.Then there exists SOL system S with such growth function f_S that $f_S(n)=cg(n)$ for all $n\geq 1$.

(4) Let g_1,g_2 be in $\mathcal{F}$(SOL) and let c,d be arbitrary nonnegative real numbers.Then there exists SOL system S such that $f_S(n)=cg_1(n) + dg_2(n)$ for all $n\geq 1$.

6. Classification of Growth Functions

We classify STSOL schemas according to their growth rate in this section.First we define some basic notion.

DEFINITION.Let f_S be the growth function of a STSOL system S.The growth rate of S is said to be <u>terminating</u> if there exists such natural number N that $f_S(n)=0$ for all $n>N$.The growth rate of S is said to be <u>limited</u> if there exists a polynomial p(n) such that $f_S(n)\not\leq p(n)$ for all n.If the growth rate of S is not limited, then it is said to be <u>explosive</u>.A STSOL scheme $G=\langle \sum,\mathcal{A},H,\mathcal{K} \rangle$ is said to be <u>Terminating/limited</u>,<u>explosive</u>/provided that for each axiom distribution over $\sum^+$ the STSOL system $S=\langle \sum,\mathcal{A},H,\mathcal{K},\omega \rangle$ has

terminanting /limited,explosive/ growth rate respectively.

THEOREM 11.Let S be a STSOL scheme with growth matrix G_S.Let $p(z)=\det(G_S - zI)$ be the characteristic polynomial of G_S.Then

(1) S is terminating if and only if all roots of p(z) are zero

(2) S has limited growth rate iff all roots of p(z) have the absolute values less than equal to one

(3) S has explosive growth rate iff at least one root of p(z) has the absolute value strictly greater than one.

7. Growth Equicalences

We introduce and study several equivalence problems for all types of stochastic L system in this section.Using the assertion of theorem 2 in this paper and the decidability results made by Eichhorst and Savitch [2] for SOL schemas and SOL systems we should formulate theorem as theorem10 in [2] in the similar way for STSOL schemas and STSOL systems.We omit this formulation and we shall define the other types of growth equivalences for stochastic L systems.

DEFINITION.Let $S_1=\langle\sum_1,\mathcal{R}_1,\mathcal{H}_1,\omega_1\rangle$ and $S_2=\langle\sum_2,\mathcal{R}_2,\mathcal{H}_2,\omega_2\rangle$ be a TSOL systems.Let $A_1=\langle\sum_1,\mathcal{R}_1,H_1,\mathcal{H}_1,\omega_1\rangle$ and $A_2=\langle\sum_2,\mathcal{R}_2,H_2,\mathcal{H}_2,\omega_2\rangle$ be STSOL systems with growth functions f_{A_1} and f_{A_2}.We say that H_1 is growth equivalent to H_2 for S_1,denoted $H_1\sim H_2$,iff $f_{A_1}(n)=f_{A_2}(n)$ for all $n\geq 0$.We say that S_1 is more powerful in H completion as S_2,denoted $S_1\xrightarrow{H}S_2$,iff for every H_2 such that $M_2=\langle\sum_2,\mathcal{R}_2,H_2,\mathcal{H}_2,\omega_2\rangle$ is a STSOL system,there exists such H_1 that the growth function f_{M_1} of STSOL system $M_1=\langle\sum_1,\mathcal{R}_1,H_1,\mathcal{H}_1,\omega_1\rangle$ is the same as the growth function f_{M_2} of M_2.Two TSOL systems S_1 and S_2,are said to be H growth equivalent,denoted $S_1\xleftrightarrow{H}S_2$,provided $S_1\xrightarrow{H}S_2$ and $S_2\xrightarrow{H}S_1$.

DEFINITION.Let $S_1=\langle\sum_1,\mathcal{R}_1,H_1,\omega_1\rangle$ and $S_2=\langle\sum_2,\mathcal{R}_2,H_2,\omega_2\rangle$ be two STOL systems.Let $A_1=\langle\sum_1,\mathcal{R}_1,H_1,\mathcal{H}_1,\omega_1\rangle$ and $A_2=\langle\sum_2,\mathcal{R}_2,H_2,\mathcal{H}_2,\omega_2\rangle$ be STSOL systems with growth function f_{A_1} and f_{A_2}.We say that $\mathcal{H}_1$ is growth equivalent to $\mathcal{H}_2$ for S_1,denoted $\mathcal{H}_1\sim\mathcal{H}_2$,iff $f_{A_1}(n)=f_{A_2}(n)$ for all $n\geq 0$.We say that S_1 is more powerful in $\mathcal{H}$ completion as S_2,denoted $S_1\xrightarrow{\mathcal{H}}S_2$,iff for each $\mathcal{H}_2$ such that $M_2=\langle\sum_2,\mathcal{R}_2,H_2,\mathcal{H}_2,\omega_2\rangle$ is a STSOL system,there exists such $\mathcal{H}_1$ that the growth function f_{M_1} of STSOL system $M_1=\langle\sum_1,\mathcal{R}_1,H_1,\mathcal{H}_1,\omega_1\rangle$ is the same as the growth function

f_{M_2} of M_2. Two STOL systems S_1 and S_2 are said to be $\mathcal{R}$ growth equivalent, denoted $S_1 \overset{\mathcal{R}}{\leftrightarrow} S_2$, provided $S_1 \overset{\mathcal{R}}{\Rightarrow} S_2$ and $S_2 \overset{\mathcal{R}}{\Rightarrow} S_1$.

DEFINITION. Let $S=\langle \sum, R, h, \omega \rangle$ be a SOL system. Then the triple $G=\langle \sum, R, \omega \rangle$ we call SOL subscheme. Let $S_1=\langle \sum_1, R_1, \omega_1 \rangle$ and $S_2=\langle \sum_2, R_2, \omega_2 \rangle$ be SOL subschemas. Let $A_1=\langle \sum_1, R_1, h_1, \omega_1 \rangle$ and $A_2=\langle \sum_1, R_1, h_2, \omega_1 \rangle$ be SOL systems. We say that h_1 is growth equivalent to h_2 for S_1, denoted $h_1 \sim h_2$, iff $f_{A_1}(n)=f_{A_2}(n)$ for all $n \geq 0$. We say that S_1 is more powerful in h completion as S_2, denoted $S_1 \overset{h}{\Rightarrow} S_2$, iff for each h_2 such that $M_2=\langle \sum_2, R_2, h_2, \omega_2 \rangle$ is a SOL system there exists such h_i that the growth function f_{M_1} of SOL system $M_1=\langle \sum_1, R_1, h_1, \omega_1 \rangle$ is the same as the growth function f_{M_2} of M_2. Two SOL subschemas S_1 and S_2 are said to be h growth equivalent, denoted $S_1 \overset{h}{\leftrightarrow} S_2$, provided $S_1 \overset{h}{\Rightarrow} S_2$ and $S_2 \overset{h}{\Rightarrow} S_1$.

THEOREM 12. There are algorithms to decide each of following yes-no problems. (1) Given a TSOL system $S=\langle \sum, \mathcal{R}, \mathcal{H}, \omega \rangle$ and H_i such that $A_i=\langle \sum, \mathcal{R}, H_i, \mathcal{H}, \omega \rangle$ are STSOL systems for i=1,2. Determine whether or not $H_1 \sim H_2$ for S. (2) Given a STOL system $S=\langle \sum, \mathcal{R}, H, \omega \rangle$ and $\mathcal{R}_i$ such that $A_i=\langle \sum, \mathcal{R}, H, \mathcal{R}_i, \omega \rangle$ are STSOL systems for i=1,2. Determine whether or not $\mathcal{R}_1 \sim \mathcal{R}_2$ for S. (3) Given a SOL subscheme $S=\langle \sum, R, \omega \rangle$ and h_i such that $A_i=\langle \sum, R, h_i, \omega \rangle$ are SOL systems for i=1,2. Determine whether or not $h_1 \sim h_2$ for S.

OPEN PROBLEMS. Are there algorithms to decide some of following yes-no problems?

(i) Let S_1 and S_2 be two TSOL systems. Determine whether or not $S_1 \overset{H}{\Rightarrow} S_2$ $(S_1 \overset{H}{\leftrightarrow} S_2)$.

(ii) Let S_1 and S_2 be two STOL systems. Determine whether or not $S_1 \overset{\mathcal{R}}{\Rightarrow} S_2$ $(S_1 \overset{\mathcal{R}}{\leftrightarrow} S_2)$.

(iii) Let S_1 and S_2 be two SOL subschemas. Determine whether or not $S_1 \overset{h}{\Rightarrow} S_2$ $(S_1 \overset{h}{\leftrightarrow} S_2)$.

We conclude this paper with some interesting remarks concerning the open problems. We conjecture that the open problems (i), (ii) and (iii) are equivalent.

REFERENCES

[1] P.Eichhorst,Stochastic Lindenmayer Systems,Ph.D.Thesis, Department of Applied Physics and Information Science, University of California, San Diego (1977).

[2] P.Eichhorst and W.J.Savitch,Growth Functions of Stochastic Lindenmayer Systems,Inform.Contr. 45 (1980) 217-228.

[3] G.Herman and G.Rozenberg,Developmental Systems and Languages,North-Holland,Amsterdam (1975).

[4] H.Jürgensen,Stochastische L-systeme,Bericht 7501, Universität Kiel (1975).

[5] H.Jürgensen,Probabilistic L-systems,Automata,Languages, Development,North Holland,Amsterdam (1976) 211-225.

[6] A.Lindenmayer,Mathematical Models of Cellular Interaction in Development,J.Theoret.Biol. 18 (1968) 280-299.

[7] G.Schäffler,Stochastische OL-systeme,Bericht NR 21, Institut für Informatik,Universität Hamburg (1975).

ON k-REPETITION FREENESS OF LENGTH UNIFORM MORPHISMS OVER A BINARY ALPHABET

Veikko Keränen

Department of Mathematics, University of Oulu
Linnanmaa, SF-90570 Oulu, Finland

Abstract. A word is called k-repetition free, if it contains no subword of the form P^k, where k is a natural number and $P \neq \lambda$. Let h be a length uniform morphism over the alphabet $\{a,b\}$ (meaning that $|h(a)| = |h(b)|$) and let $h(a) \neq h(b)$. In this case we can give an optimal upper bound for the length of P^n, $n \geq k$, where P^n is a subword of $h(w)$ such that w is k-repetition free. Also, we give outlines for the proof of the following result. When deciding whether a given morphism h, of the form mentioned above, is k-repetition free, one has only to examine the words $h(w_o)$, where the length of w_o is ≤ 4 (or, in some special cases, even less). Finally, we consider the k-repetition freeness of DOL and NDOL sequences and obtain, for example, the following result. If h generates a cube P^3 in a DOL sequence, then it does it in three steps.

1. INTRODUCTION

Our study originates from L. Wegner's problem, solved by many writers in [1]. In our article there (pp. 24 - 31) we were considering the word sequence $S = s_1, s_2, s_3, \ldots$, defined recursively as follows:

(1)
$$\begin{cases} s_1 = u \\ s_{i+1} = s_i s_i s_i^R \qquad (i \geq 1), \end{cases}$$

where $u \neq \lambda$ is a word over a finite alphabet Y and s_i^R is the reversal (mirror image) of s_i. Originally, in Wegner's sequence $u = ab$, and the broblem was to decide whether this sequence is cube-free. In the sequel, let u and v be different nonempty words of equal length and consider the NDOL system

(2)
$$G = (X, Y, g, h, a),$$

where $X = \{a,b\}$, the endomorphism $g: X^* \to X^*$ is defined by $g(a) = aab$, $g(b) = abb$, and the length uniform morphism $h: X^* \to Y^*$ is defined by $h(a) = u$, $h(b) = v$. We showed in [1] that $S = E(G)$, if $v = u^R$, and conjectured that if P^3 is a subword of a word in the sequence S, then the length of P is $\leq \frac{4}{3}|u|$. We can now show that this (optimal result) is true for S and also for any NDOL sequence $E(G)$ generated by G in (2). This in turn implies that all cubes P^3 in the sequence $E(G)$ can be found in the fourth word. Also, by the fact that $S = E(G)$, Theorem 7 below implies in a trivial way that the Wegner's sequence ($u = v^R = ab$ in (1)) is cube-free. Moreover, in this article we generalize these considerations to concern also general k-repetition of words.

2. PRELIMINARIES

For the very basic notations and definitions (concerning, for example, L systems) the reader is referred to [3], or to [2] and [4], which also contain results concerning repetitions.

A word u is called a subword [an inner subword] of w, if $w = u_1uu_2$ for some words [nonempty words] u_1 and u_2. The notation SW(w) [ISW(w)] denotes the set of all subwords [inner subwords] of w. Moreover, we write $XSW(w_1,\dots,w_n) = XSW(w_1) \cup \cdots \cup XSW(w_n)$ for words w_i and $X = \lambda$ or I. A word w is primitive, if $w = w_0^n$ implies $n = 1$ (w is only a trivial power of another word w_0).

Let k be a natural number. Then we say that a word is [strongly] k-repetition free, if it contains no subword of the form $P^k \neq \lambda$ [$P^{k-1}a$, where a is the first letter of $P \neq \lambda$]. A word sequence or a language is k-repetition free, if all words in it are k-repetition free, otherwise we say that there is a k-repetition in it. A morphism $h: X^* \to Y^*$ is k-repetition free, if $h(w)$ is k-repetition free for every k-repetition free w in X^*.

Finally, if not otherwise mentioned, Y is a finite alphabet, u, v are different nonempty words of equal length and $h: \{a,b\}^* \to Y^*$ is a length uniform morphism such that $h(a) = u$ and $h(b) = v$. In the case of endomorphism we sometimes write g instead of h.

3. MAIN LEMMAS

The proofs of our main results are quite long containing many lemmas, from which we present the most interesting and fundamental ones.

Lemma 1. Let u, v and w be nonempty words of equal length such that $u \neq v$. Then $uv \notin ISW(u^2v, vu^2, w^3)$ and $w^2 \notin SW(uvu)$. Furthermore, if $uv \in ISW(uvu, vuv)$, then $uv = w_0^r$ for some w_0 and $r \geq 3$.

By Lemma 1 it is easy to prove

Lemma 2. Let u and v be different nonempty words of equal length and $A = \{u^2v, vu^2, v^2u, uv^2\}$. Then $A \cap ISW(\alpha\beta\gamma\delta) = \emptyset$ for α, β, γ and δ in $\{u,v\}$.

Lemma 2 tells us that if a word w is in $\{u,v\}^*$ and $w = w_1w_2w_3$, with w_2 in A, then also w_1 and w_3 are in $\{u,v\}^*$. This implies the following fact. If a word w in $X^* = \{a,b\}^*$ is k-repetition free, then $h(w)$ contains only short subwords of the form P^k such that $P \notin [h(X)]^*$. (Note that $P^k = P^{k-1}P = PP^{k-1}$ and every long P^{k-1} would contain, as a subword, a word in A.) On the other hand, if here $P \in [h(X)]^*$, then $P^k = \lambda$, as the following lemma shows. In this lemma we also consider the case $|X| > 2$.

Lemma 3. Let X and Y be finite alphabets with $|X| \geq 2$ and $|Y| \geq 2$, and let a morphism $h: X^* \to Y^*$ be such that $h(a)$ and $h(b)$ are different nonempty words of equal length for every different a and b in X. If a word w in X^* is strongly k-repetition free and $h(w)$ has a subword Q of the form $pref(P^{k+1})$, where $P \in [h(X)]^+$, then $|Q| < |P^{k-1}| + 2|h(a)|$ in the case $|X| \geq 3$, $k \geq 2$, and $|Q| < |P^{k-1}| + |h(a)|$ in the case $|X| = 2$, $k \geq 3$. Furthermore, if $|X| = 2$ and a word w is k-repetition free $(k \geq 3)$, then $h(w)$ has no subword of the form P^k, where $P \in [h(X)]^+$.

4. MORPHISMS

By the previous lemmas we have obtained the following optimal result.

Theorem 4. If a word w in $\{a,b\}^*$ is k-repetition free $(k \geq 3)$ and $h(w)$ has a subword of the form P^n, where $n \geq k$, then

$|P^n| < (2k - 1)|u|$. If, in addition neither $a(ba)^{k-1}$ nor $b(ab)^{k-1}$ is a subword of w , then $|P^n| \leq 2(k - 1)|u|$.

As an example of the optimality of the upper bound for $|P^n|$, consider the case $n = k = 3$, $u = h(a) = cdc$ and $v = h(b) = dcd$. Choosing $P^3 = h(abab) = (cdcd)^3$, we get $|P^3| = 4|u| = 2(k - 1)|u|$.

As a corollary of Theorem 4 we get

Corollary 5. Let a word w in $\{a,b\}^*$ be k-repetition free ($k \geq 3$), and let w_o be a subword of w . Then $h(w)$ is k-repetition free if and only if $h(w_o)$ is k-repetition free for every w_o of length $\leq 2k - 1$.

This corollary gives us a test set for checking whether a given length uniform morphism $h: \{a,b\}^* \to Y^*$ is k-repetition free. However, Corollary 5 is more interesting from the point of general k-repetition freeness of DOL and NDOL sequences. This is the case, because in Theorem 7 we obtain, quite independently of the proof of Theorem 4 and Corollary 5, a very efficient test set (optimal in general) for testing the k-repetition freeness of h .

Lemma 6. Let the words u , v , uv and vu be primitive and, if $k = 3$, the words u and v cube-free. Let a word w in $\{a,b\}^*$ be k-repetition free ($k \geq 3$) and P^k a subword of $h(w)$, say $h(w) = w_1 P^k w_2$. Then $w_1 P^k w_2 \neq w_1 \alpha \beta \gamma w_2$ whenever $w_1\alpha$, γw_2 are in $\{u,v\}^*$ and β is in $\{uu, vv, uvu, vuv\}$. Consequently, P^k is a subword of $h(w_o)$, where w_o is a subword of w and $|w_o| \leq 4$.

Using Lemma 6 one can easily prove

Theorem 7. A length uniform morphism $h: \{a,b\}^* \to Y^*$ is k-repetition free ($k \geq 3$) if and only if

(i) u, v, uv and vu are primitive;

(ii) $h(w_0)$ is k-repetition free for every w_0 in $\{a,b\}^+$ of length ≤ 3 (differing from a^3 and b^3 if $k = 3$); and

(iii) $h(w_0)$ is k-repetition free for every w_0 in $\{xxyx, xxyy, xyxx, xyxy \mid x, y \text{ in } \{a,b\} \text{ and } x \neq y\}$.

By Theorem 7 it is easy to decide, for any k, whether a given morphism h, of the considered form, is k-repetition free. Because the words w_0 (in the test set) are always of length ≤ 4, the amount of work, needed to solve the problem, does not grow as k gets greater. In fact we get

Corollary 8. Let $|u| = |v| \geq 2$, and let u, v, uv and vu be primitive. Then h is $(2|u| - 1)$-repetition free.

As an example, let $u = h(a) = aab$ and $v = h(b) = baa$. Then, by Corollary 8, h is 5-repetition free.

5. ENDOMORPHISMS

Concerning short words u and v in Theorem 7, one can still improve the upper bounds for the length of w_0. In the case of endomorphism ($X = Y = \{a,b\}$) we have obtained the following results using UNIVAC 1100/22 computer and, in spite of an efficient program, hours of CPU time. In the case $k = 3$ we found out that

(3) $g(a) = abbaababaababba$, $g(b) = baababbaababaab$

is the first (in alphabetical order with respect to $h(a)h(b)$) binary length uniform endomorphism g such that $g(w_0)$ is cube-free for every cube-free w_0 of length ≤ 3, but $g(w_0)$ contains a cube for a cube-free w_0 of length $= 4$. Here $g(abaa)$ contains the cube $(babaababbaa)^3$ (but $g(w_0)$ is cube-free for other cube-free $w_0 \neq abaa$ of length $= 4$). So we have

Theorem 9. In the case $|u| = |v| \leq 14$ a length uniform endomorphism g over the alphabet $\{a,b\}$ is cube-free if and only if

(i) $u \neq v$;

(ii) u and v are primitive; and

(iii) $g(w_0)$ is cube-free for every cube-free w_0 in $\{a,b\}^+$ of length ≤ 3.

Also, for $k = 3$, 4 and 5, the following endomorphisms

$k = 3$: $g(a) = abaabba$, $g(b) = baabaab$;

$k = 4$: $g(a) = aabba$, $g(b) = babab$;

$k = 5$: $g(a) = aaabbba$, $g(b) = bababab$

are the first such that g fulfils the primitiveness condition (i) in Theorem 7 and $g(w_0)$ is k-repetition free ($k = 3$, 4 or 5) for every w_0 of length ≤ 2, but $g(w_0)$ contains a k-repetition for a w_0 of length 3 or 4 (consider the words $g(aba)$). These results mean that

Theorem 10. In the case $k = 3$ [$k = 4$, $k = 5$] and $|u| = |v| \leq 6$ [≤ 4, ≤ 6] a length uniform endomorphism g over the alphabet $\{a,b\}$ is k-repetition free if and only if

(i) u , v , uv and vu are primitive; and

(ii) uu , vv , uv and vu are k-repetition free.

Using these results one can now enumerate in alphabetical order k-repetition free endomorphisms for any k . As an example we consider the lengths $|u| = |v| = 1, \ldots, 6$, the repetitions $k = 3, 4, 5$ and give for each $|u|$ and k the first endomorphism g such that g is k-repetition free but not $(k-1)$ - repetition free. By Theorems 10 and 7 the work is here very easy also without any computer program.

		g_1	g_2	g_3	g_4	g_5	g_6
$k = 3$:	a	a	ab	aab	abba	abaab	aababb
	b	b	ba	abb	baab	babba	aabbab
$k = 4$:	a			aab	aaab	aaabb	aaabab
	b			aba	aabb	aabab	aaabbb
$k = 5$:	a			aab	aaab	aaaab	aaaabb
	b			baa	aaba	aaabb	aaabab

6. DOL AND NDOL SEQUENCES

Here we consider the decidability of k-repetition freeness ($k \geq 3$) of DOL and NDOL sequences generated by length uniform morphisms. It is possible to generate a k-repetition free DOL sequence, although the iterated endomorphism g itself is not k-repetition free. For example, if

$$g(a) = ab\ , \quad g(b) = aa\ , \tag{4}$$

then $G = (\{a,b\}, g, a)$ generates a 4-repetition free DOL sequence, as is straightforward to see using Corollary 5. However, here $g(b^{k-1}) = a^{2(k-1)} = a^k a^{k-2}$ showing that g is not k-repetition free for any $k \geq 3$.

In general it is decidable, for any k, whether a given endomorphism g generates a k-repetition free DOL sequence (see [2], p. 291). In our case this result is easy to prove using Corollary 5. A similar positive decidability result holds true also for NDOL sequences, generated by NDOL systems $G = (\{a,b\}, Y, g, h, \omega)$, where the morphisms g and h are length uniform.

In deciding the k-repetition freeness of a DOL [NDOL] sequence generated by a given length uniform g [g and h] we, at first, decide by Theorem 7 whether the given morphism[s] are k-repetition free. If the answer is affirmative, then we are ready. Otherwise we have to check whether there is a subword w_0 in the DOL sequence such that $g(w_0)$ [$g(w_0)$ or $h(w_0)$] is not k-repetition free. In this work there are two possibilities: either u, v, uv and vu are primitive for g [g and h] or they are not. If not, then we have to use Corollary 5 instead of Lemma 6 that otherwise would be simpler to use. (For example, consider g in (4).) However, here Lemmas 1, 2 and 3 may be very useful.

In the sequel (except in Theorem 12) we consider the situation, in which u, v, uv and vu are primitive. We obtain optimal upper bounds for the number of words needed to be tested in the beginning of a given DOL or NDOL sequence in order to solve the k-repetition freeness problem for this sequence. At first we consider DOL sequences. Here, using Lemma 6 we get

Theorem 11. Let g be a length uniform endomorphism over the alphabet $\{a,b\}$ such that $g(a)$, $g(b)$, $g(ab)$ and $g(ba)$ are primitive. Then a DOL sequence $E(G)$ generated by $G = (\{a,b\}, g, \omega)$ is k-repetition free if and only if the set $\{g^n(\omega) \mid n \leq 4\}$ is k-repetition free.

In general this result is optimal as indicated by the endomorphism $g(a) = babab$, $g(b) = aabaa$, for which $g^4(a)$ contains a 4-repetition (a^4), but the set $\{g^n(a) \mid n \leq 3\}$ is 4-repetition free. Using Theorem 10 one immediately sees that this g is 5-repetition free.

Although Theorem 11 is optimal in general, we get in Theorem 12 still stronger result for the case $k = 3$. This case is quite exceptional: if $|u| = |v|$ and g does not fulfil the primitiveness condition (i) in Theorem 7, then (see [2], p. 282) g never generates a cube-free DOL sequence. In the case $k \geq 4$ the situation is very different as indicated by g in (4). This special property makes the cube-freeness problem for DOL sequences (generated by binary length uniform endomorphisms) very easy to solve, at least for short words u and v. In fact we know that, if $2 \leq |g(a)| = |g(b)| \leq 14$, then the DOL sequence generated by $G = (\{a,b\}, g, \omega)$ is cube-free if and only if g and ω are cube-free. It is possible that instead of 14 we

may have even 32 , but not more, since the endomorphism g , defined by

$$g(a) = aabaabbaabbabbaabaababbabbaabaabb ,$$

$$g(b) = aabbabbaabaababbabbaabaabbaabbabb$$

is such that $g(w_0)$ is cube-free for every cube-free $w_0 \neq baba$ of length ≤ 4 , but $g(baba)$ contains the cube

$$P^3 = (abbabbaabaabbaabbabbaabaab)^3 .$$

Furthermore, baba is not in $SW\{ g(w_0) \mid |w_0| \leq 2\}$, and so $E(G)$ is cube-free for every DOL system $G = (\{a,b\}, g, \omega)$, where ω is cube-free and baba not a subword of ω . (Note also that g fulfils part (i) in Theorem 7.)

In [2] it is shown that, in general, binary endomorphism (not necessarily length uniform) generates a cube in a DOL sequence if and only if it does it in less than 11 steps. Using results in [2] we can in our case prove the following Theorem 12. This result is optimal as seen by considering (once again) the endomorphism g in (4) .

Theorem 12. Let g be a length uniform endomorphism over the alphabet $\{a,b\}$. Then a DOL sequence $E(G)$ generated by $G = (\{a,b\}, g, \omega)$ is cube-free if and only if the set $\{g^n(\omega) \mid n \leq 3\}$ is cube-free.

Finally we consider NDOL sequences. The following theorem is straightforward to prove using Theorem 11, Lemmas 6 and 2 and the fact that a word w in {aa, bb, ab, ba} is a subword of $E(G)$ generated by $G = (\{a,b\}, g, \omega)$, where g is length uniform, if and only if w is in $SW\{ g^n(\omega) \mid n \leq 3\}$.

Theorem 13. Let g and h be length uniform morphisms over the alphabet $\{a,b\}$ such that the image words of a, b, ab and ba are primitive for both g and h [for g and $h(a) \neq h(b)$]. Then an NDOL sequence $E(G)$ generated by $G = (\{a,b\}, Y, g, h, \omega)$ is k-repetition free if and only if the set $\{ h(g^n(\omega)) \mid n \leq 4 \ [n \leq 5]\}$ is k-repetition free.

By Theorem 11 the nonbracketed part of this result is optimal. The same upper bound 4 for n is optimal also in the case, where $k = 3$ and the underlying DOL sequence is cube-free. This is seen by considering the case: $\omega = b$, $g(a) = ab$, $g(b) = ba$ and h as the morphism g in (3).

In our last theorem we consider the case $k = 3$. Here the proof goes straightforwardly using Theorem 12, Lemma 6 and the fact that all the words aa, bb, ab and ba are subwords of $g^2(\omega)$ whenever the DOL sequence $E(G)$ generated by $G = (\{a,b\}, g, \omega \neq \lambda)$ is cube-free and $|g(a)| = |g(b)| \geq 3$.

Theorem 14. Let g and h be length uniform morphisms over the alphabet $\{a,b\}$ such that $|g(a)| = |g(b)| \geq 3$. Then an NDOL sequence $E(G)$ generated by $G = (\{a,b\}, Y, g, h, \omega)$ is cube-free if and only if the set $\{ h(g^n(\omega)) \mid n \leq 3\}$ is cube-free.

By choosing $\omega = b$, $g(a) = aab$, $g(b) = abb$ and h as the morphism in (3), we see that this result is optimal, too.

ACKNOWLEDGEMENTS

The author wants to thank Professor Paavo Turakainen for encouragement and many valuable comments. Also, thanks are due to Jussi Mattila, whose contribution to computer programs was of fundamental importance.

REFERENCES

[1] E.A.T.C.S. Bulletin 19 (1983).

[2] J. Karhumäki, On cube-free ω-words generated by binary morphisms, Discrete Appl. Math. 5 (1983) 279-297.

[3] G. Rozenberg and A. Salomaa, The Mathematical Theory of L Systems (Academic Press, London, 1980).

[4] A. Salomaa, Jewels of Formal Language Theory (Computer Science Press, 1981).

THE COMPLEXITY OF THE CONSENSUS BETWEEN HIERARCHICAL TREES

Mirko Křivánek
VÚMS Praha
Loretánské nám.3
118 55 Praha 1
Československo

1. INTRODUCTION

The construction of the optimium hierarchical tree is the kernel of hierarchical clustering. This is difficult, in fact NP-hard [6], problem. Because of wide range of applications both in social and natural sciences, a lot of efficient (but approximate) procedures have been developed and used in hierarchical clustering [3]. These procedures sometimes lead to completely different hierarchical trees and also for this reason we need to study the consensus between them. In subsequent sections we review some basic definitions that are necessary for introducing three open (as to complexity) decision computational problems arising when we want to compare several hierarchical trees from the poit of view of medians [1], minimum length sequence metric [2] and nearest neighbour interchange metric [8]. All these problems are shown to be NP-complete. (Our NP-completeness theoretical terminology is that of [5]).

2. BACKGROUND

In this section we give a brief summary of relevant definitions [cf.7] which are also visualized in Figure 1.

Throughout this paper let $X = \{x_1,\ldots,x_n\}$ be a finite set. Hierarchical tree H_X' (or simply hierarchy) on X is every subset of $\mathcal{P}(\mathcal{P}(X))$ [††] such that

(1) $X \in H$

(2) $\emptyset \in H$

(3) $(\forall x \in X) \quad \{x\} \in H$

(4) $(\forall h,h' \in H) \quad h \cap h' \in \{h,h',\emptyset\}$

[†] Usually droping the subscript X when it is clear from the context

[††] $\mathcal{P}(X)$ denotes the power set of the set X

$X = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}\}$

$H = \{\{x_1\}, \{x_2\}, \{x_3\}, \{x_4\}, \{x_5\}, \{x_6\}, \{x_7\}, \{x_8\}, \{x_9\}, \{x_{10}\},$
$\{x_1, x_2, x_3\}, \{x_6, x_7\}, \{x_5, x_6, x_7\}, \{x_4, x_5, x_6, x_7\}\ (= i),$
$\{x_1, x_2, x_3, x_4, x_5, x_6, x_7\}, \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9\}, X\}$

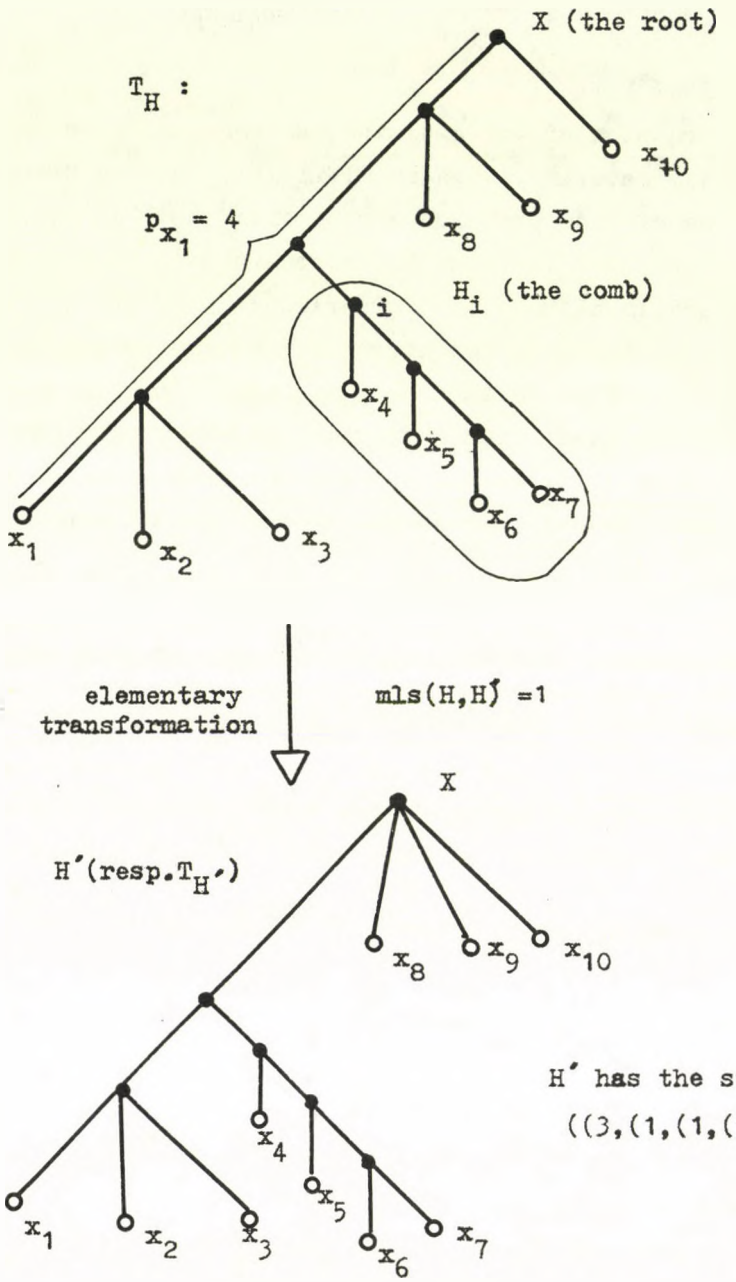


Figure 1.

Every hierarchy H is in 1-1 correspondence with the tree $T_H = (H, U_H)$, where $U_H = \{ (h,h') \in H \times H \ / \ h' \subseteq h \ \& \ (h'' \subseteq h \rightarrow h'' = h') \}$. By p_h we denote the length of the path between X and h in T_H. The degree of the vertex h of T_H will be denoted by s_h.

Hereafter we shall use the following notation concerning H_X :

$\{X\}$ † ... the root of H_X

$L = \{\{x\} \ / \ x \in X\}$ ... hierarchy H_X is labeled by the set of leaves L

$I = H - L$ .... the set of internal vertices of H_X

$H_h = \{ h' \in H_X \ / \ h' \subseteq h \}$ ... the subhierarchy of H_X induced by an internal vertex $h \in I$

$n_h = |h|$.

The hierarchy H is described by means of the following indices :

Height	$\varrho(H) = \max \{ p_x \ / \ x \in X \}$	
Width	$\delta(H) = \sum_{x \in X} p_x$	$(= \sum_{i \in I} n_i)$
Size	$\gamma(H) = \lvert I \rvert - 1$	
Density	$\eta(H) = \sum_{i \in I} s_i$	

We shall express the structure of a hierarchy H_X as the word W_X in the alphabet $N \cup (,,,)$ with the aid of the following rules :

(1) $h = \{x\} \in L \rightarrow W_h = (1)$

(2) $h \in I \ \& \ (h,h_j) \in U_H \ , \ j = 1,\ldots,s_h$

$$\rightarrow W_h = \begin{cases} \text{either } (n_{h_1},\ldots,n_{h_{s_h}}) \\ \text{or } (W_{h_1},\ldots,W_{h_{s_h}}) . \end{cases}$$

When the integer a appears b times in the structure $(a,\ldots a)$ it will be abbreviated by $(b \times a)$.

We say that the hierarchy H is <u>binary</u> iff $s_X = 2$ and $(\forall \ i \in I - X)$ $s_i = 3$. The binary hierarchy H is said to be a <u>comb</u> (on X) iff each subhierarchy H_i of H ($i \in I$) has the structure $(n_i - 1, 1)$.

The dissimilarity of two hierarchical trees is mesured by the length of the sequence of elementary transformations converting one hierarchy into the other. We say that the hierarchy H' on X is obtained by an elementary transformation from the hierarchy H on X iff $1 \leq |H \Delta H'| \leq 2$.

(Δ stands for the symmetric difference of sets). Note that the intersection of two hierarchies on X is the hierarchy on X and that $|B \Delta B'| > 1$ B,B' being two different binary hierarchies.

† Usually writing X instead of $\{X\}$.

By mls (H,H') we denote the minimum number of elementary transformations needed to convert the hierarchy H into the hierarchy H'. B and B' being binary hierarchies on X, mls (B,B') is exactly the nearest neighbour interchange metric of [8] (see [7]) and so we shall write nni (B,B') instead of mls (B,B').

3. RESULTS

First we shall consider a family of median-type computational problems in hierarchical clustering [1] :

MEDIAN : Instance : Hierarchies $H_1,\ldots,H_m$ on X, integer k, $\iota \in \{\rho,\delta,\gamma,\eta\}$

Question : Is $\sum_{j=1,\ldots,m} \iota(H \cap H_j) \geq k$?

We show that the MEDIAN is NP-complete problem for the index γ. The proof for the other indices is similar and thus omitted.

THEOREM 1

The MEDIAN is NP-complete.

Proof.

We first note that MEDIAN is certainly a member of NP since a nondeterministic algorithm can just guess a hierarchy H_X and then check if $\sum_{j=1,\ldots,m} \gamma(H \cap H_j) \geq k$, which can be done in polynomial time. Now we polynomially transform the known NP-complete problem (X3C) [5,p.221] to MEDIAN.

EXACT COVER BY 3-SETS (X3C) : Instance : Set X with $|X| = 3q$ and a collection C of 3-element subsets of X.

Question : Does C contain an exact cover for X, i.e. a subcollection $C' \subseteq C$ such that every element of X occurs in exactly one member of C'?

So let X,C be an arbitrary instance of (X3C). The corresponding instance of MEDIAN is illustrated in Figure 2.

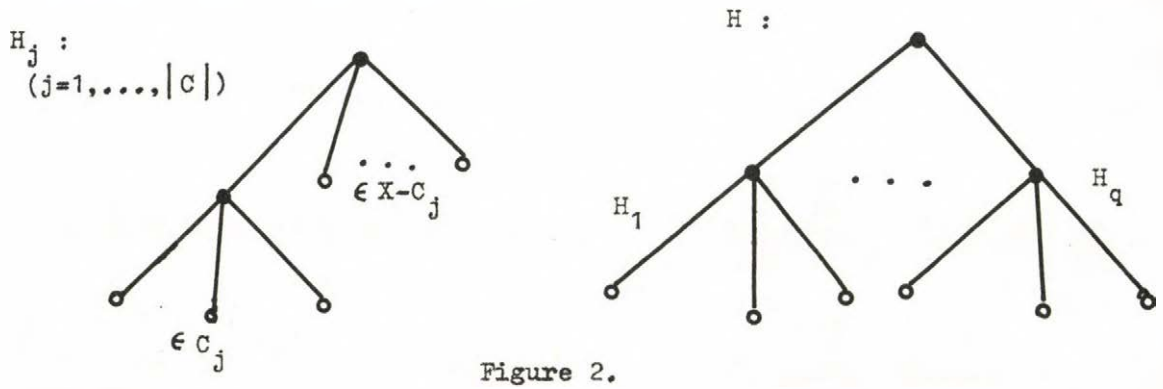


Figure 2.

We complete the proof by verifying the following equivalence :

$\sum_{j=1,\ldots,|C|} \gamma(H \cap H_j) = q$ iff (X3C) has an exact cover for X.

Indeed, let $C_1,\ldots,C_q$ be the exact cover for X. We construct the hierarchy H (on X) consisting exactly of q subhierarchies H_i such that each H_i is labeled by C_i, i = 1,...,q. Then H has the structure (q × 3) and the equality $\sum_{j=1,\ldots,|C|} \gamma(H \cap H_j) = q$ is satisfied (cf.Figure 2). Similarly the converse is also true, QED.

Our second problem belongs to the minimum length sequence problems introduced by W.Day [2] :

MLS : Instance : Two hierarchies H_1, H_2 on X, integer k

Question : Is $mls(H_1,H_2) \leq k$?

We find the following solution :

THEOREM 2

The MLS is NP-complete.

Proof.

It is easy to see that MLS is in NP. So we only have to give a polynomial transformation from the known NP-complete problem to it. We choose the transformation from PARTITION , [5,p.223] .

PARTITION : Instance : Finite set A, size $s(a) \in Z^+$ for each $a \in A$.

Question : Is there a subset $A' \subseteq A$, $|A'| = |A|/2$, such that $\sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a)$?

Let $\{ s(a_1),\ldots,s(a_n) \}$ be an arbitrary instance of PARTITION and let $B = 1/2 \sum_{a \in A} s(a)$. The corresponding instance of MLS is constituted by a hierarchy H_1 that has the structure $(s(a_1),\ldots,s(a_n))$ and H_2 has the structure (B,B). Now it is immediate that $mls(H_1,H_2) = n - 2$ iff PARTITION has "yes"-solution, QED.

Similarly we solve the analogous problem in the case of binary hierarchies. The following problem is often referred to as the problem of nearest neighbour interchange :

NNI : Instance : Two binary hierarchy B_1, B_2 , integer k

Question : Is $nni(B_1,B_2) \leq k$?

THEOREM 3

The NNI is NP-complete.

Proof (Outline).

The problem NNI is obviously in NP ; furthemore we can polynomially

transform PARTITION to it. So taking any instance $\{s(a_1),\ldots,s(a_n)\}$ of PARTITION we construct the instance of NNI as it is implied in Figure 3.

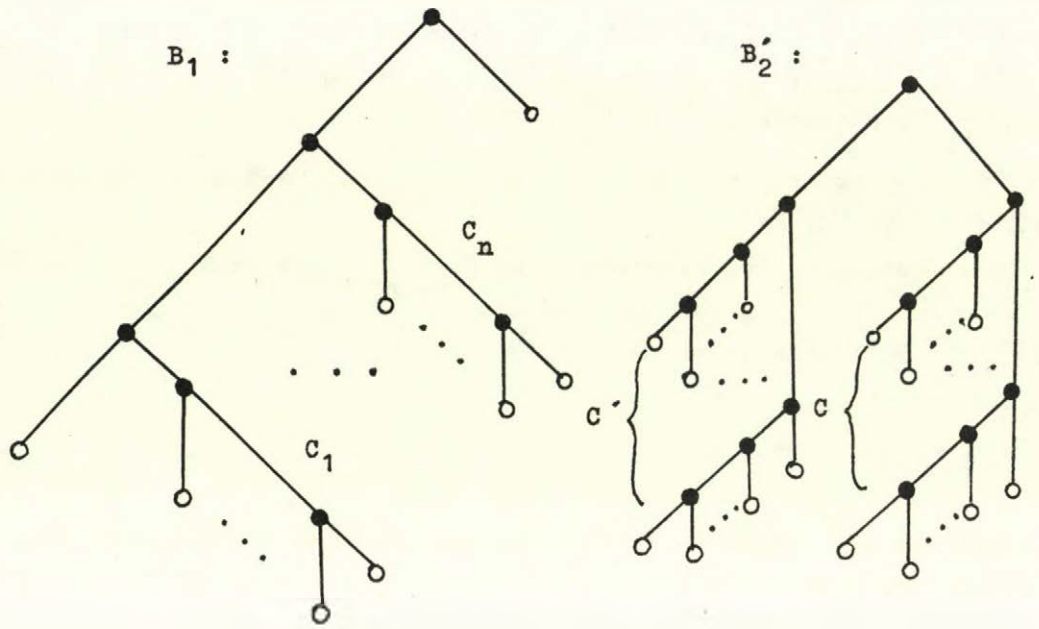


Figure 3.

Let $B = n^2/2 \sum_{i=1}^{n} s(a_i)$. Then the binary hierarchy B_1 has the structure $(((\ldots((1,n^2s(a_1)),n^2s(a_2)),\ldots),n^2s(a_n)),1)$ and the binary hierarchy B_2 has the structure $(B+1,B+1)$. Note that the hierarchy B_1 consists of n combs C_i and each C_i has $n^2s(a_i)$ leaves $(i=1,\ldots,n)$. To conclude the proof it is sufficient to show that $nni(B_1,B_2) \leqq n^2$ iff PARTITION has "yes"-solution. So let A' be the solution of PARTITION. Let us divide the set of combs $\{C_1,\ldots,C_n\}$ from the hierarchy B_1 into two sets C and C' as follows : $C_i \in C \iff a_i \in A'$, $C_i \in C' \iff a_i \in A-A'$, $i=1,\ldots,n$. Then the hierarchy B_2' (see Figure 3) can be obtained from the hierarchy B_1 by transforming $|A'|$ combs of C over the root in B_1. The hierarchy B_2' has the required structure and clearly $nni(B_1,B_2') \leqq n^2$. Conversely suppose that PARTITION has "no"-solution and that $nni(B_1,B_2) \leq n^2$. This yields $|\sum_{a \in A} n^2s(a) - \sum_{a \in A-A'} n^2s(a)| \geq n^2$ for each $A' \subseteq A$. Consequently we need at least one elementary transformation of one comb over the root in B_1 and n^2 elementary transformations of n^2 leaves over the root of B_1 to shape the structure $(B+1,B+1)$ of the hierararchy B_2- - a contradiction, QED.

REFERENCES

[1] Barthelemy J.P.,Monjardet B.: The Median Procedure in Cluster Analysis and Social Choice Theory. Math.Soc.Sci.1(1981),235-267.

[2] Day W.H.E.: The Complexity of Computing Metric Distances Between Partitions.Math.Soc.Sci.1(1981),269-287.

[3] Day W.H.E.,Edelsbrunner H.: Efficient Algorithms for Agglomerative Hierarchical Clustering Methods. Report F 121, Technische Universität Graz,1983.

[4] Diday E.:Croisements, Orders et Ultramétriques : Application à la recherche de consensus en classification automatique. Rapport de recherche N°144, INRIA Rocquencourt,1982.

[5] Garey M.R., Johnson D.S.: Computers and Intractability : a Guide to the THeory of NP-completeness. W.H.Freeman,San Francisco,1979.

[6] Křivánek M., Morávek J.: ON NP-Hardness in Hierarchical Clustering. To appear in COMPSTAT´84 Proceedings,Physica-Verlag,Vienna,1984.

[7] Leclerc B.: Description, Evaluation et Comparaison des Hierarchies de Parties. Rapport de Recherche de Centre d´Analyse et de Mathématique Sociale, Paris 1982.

[8] Waterman M.S., Smith T.F.: On the Similarity of Dendrograms. J.Theor.Biology 73(1978),789-800.

ON GRAMMARS WITH DOUBLE RESTRICTIONS IN DERIVATION

Gheorghe PĂUN
Institute of Mathematics
Str. Academiei 14, Bucureşti
70109 ROMANIA

1. Introduction

In order to generate non-context-free languages by grammars using context-free core rules, many restrictions were considered about the derivation in Chomsky grammars. In this frame, an interesting question is to study combinations of these restrictions, grammars with two or more regulating devices. Some such combinations were already investigated: extensions of the so-called Indian and Russian grammars were considered in /2/, /3/, respectively, and similar extensions of Friš ordered grammars /5/ were examined in /4/. The present paper continues this program by investigating the generative capacity of certain extensions of random context grammars. We consider random context matrix, programmed and scattered context grammars, as well as matrix, programmed, controlled and scattered context grammars. The main conclusion of this study is that, in these cases, two restrictions are as powerful as one (as the most powerful of the two restrictions involved), possibly excepting the case of scattered context grammars where new characterizations of context sensitive languages family are

obtained (without knowing whether the λ-free scattered context grammars generate all context sensitive languages /6/).

2. Notations

We assume the reader familiar with basic notions of formal language theory, including definitions of matrix, programmed, controlled, random context, scattered context grammars, etc. Here we shall give only informal definitions for double restriction grammars.

Generally, a class of doubly restricted grammars is denoted by $(X_\alpha, Y^{\delta}_{\beta})$, where Y is the core restriction and X is the extra restriction, acting on rules/matrices of type Y. Throughout, the subscripts α, β are either equal to ac (indicating appearance checking features) or to the empty word (no appearance checking), whereas δ is either λ (λ-rules allowed) or the empty word (no λ-rules). In this context, $\max(\alpha, \beta)$ is equal to ac iff at least one of α, β is equal to ac.

The restrictions in derivation are denoted as follows: M = = matrix, PR = programmed, RC = random context, C=(regular) control, SC = scattered context, O = ordered, I = Indian. Also, we denote by CS the class of context sensitive grammars. The family of languages generated by grammars in a class X is denoted by $\mathcal{L}(X)$.

3. Extensions of ordered grammars

We report here without proofs the results in /4/, where the next classes of grammars were considered:

i) (O, M_β) - matrix grammars with an order relation on the set of matrices,

ii) (O, PR_β) - programmed grammars with an order relation,

iii) (O, RC_β) - random context grammars with an order relation,

iv) (0,I) - ordered Indian grammars,

v) (0,SC) - ordered scattered context grammars.

The following results are proved in /4/ (they show that an order relation is of the same power as the appearance checking):

THEOREM 1. a) $\mathcal{L}(X_{ac}) = \mathcal{L}(0,X) = \mathcal{L}(0,X_{ac})$, for all $X \in \{M, PR, RC\}$;

b) $\mathcal{L}(0,I) = \mathcal{L}(EDTOL)$;

c) $\mathcal{L}(0,SC) = \mathcal{L}(CS)$.

Similar results are true if the superscript λ is added to X in the first statement and to I, SC, CS in the last two.

4. Extensions of random context grammars

4.1. Random context control imposed to other restrictions

We consider here the following classes of grammars:

i) $(RC_{\alpha}, M^{\delta}_{\beta})$ - grammars with matrices of the form $((r_1,\ldots,r_k),R,Q)$, where $r_1,\ldots,r_k$ are context-free rules and R,Q are sets of nonterminals; such a matrix is applicable to a string x in order to effectively rewrite some symbols $X_1,\ldots,X_t$, $0 \leqslant t \leqslant k$, $x = x_1X_1x_2\ldots x_tX_tx_{t+1}$, only if the string $x_1x_2\ldots x_{t+1}$ contains all symbols in R and no symbol in Q.

In all the paper, we consider each pair (R,Q), of associated sets of nonterminals, to contain disjoint sets (otherwise the related matrix/rule cannot be ever applied).

ii) $(RC_{\alpha}, PR^{\delta}_{\beta})$ - grammars with rules of the form $((r,A \longrightarrow x, \sigma_r, \varphi_r),R,Q)$; the rule $(r,A \longrightarrow x, \sigma_r, \varphi_r)$ is applied to a string w only if the restrictions imposed by R,Q are observed (when the application is effective, $w = uAv$, uv contains

all symbols in R and no symbol in Q, and when the application is not effective, then w contains the symbols in R and no symbol in Q),

iii) $(RC_\alpha, SC_\beta^\delta)$ - similar to $(RC_\alpha, M_\beta^\delta)$, with scattered context-type derivation, namely $((A_1 \longrightarrow x_1, \ldots, A_k \longrightarrow x_k), R, Q)$ is applied to $u_1A_1u_2A_2\ldots u_kA_ku_{k+1}$ only if $u_1u_2\ldots u_{k+1}$ contains all symbols in R and no symbol in Q.

The appearance checking in SC_{ac} means to pass over some rule $A_i \longrightarrow x_i$ iff it appears in a distinguished rules set F and A_i does not appear in the subword of the current sentential form bounded by the nearest neighbouring rules which are effectively applied (or the end of the sentential form when no rule in the left/right hand of $A_i \longrightarrow x_i$ is effectively applied). See discussions about SC_{ac} in /1/.

In what follows we examine the families $\mathcal{L}(X,Y)$, (X,Y) as above, compared to $\mathcal{L}(Y)$. All the proofs are constructive (they will be omitted here), by mutual simulation of involved grammars. Although some constructions are straightforward, sometimes they involve certain delicate points which have to be carefully checked. Moreover, generally, $\mathcal{L}(X_\alpha^\delta) \subseteq \mathcal{L}(Y_\beta, X_\alpha^\delta)$ and $\mathcal{L}(X_\alpha^\delta) \subseteq \mathcal{L}(X_\alpha, Y_\beta^\delta)$ (we reduce the restriction Y to the null case), and some inclusions directly follows from equalities $\mathcal{L}(M_{ac}^\delta) = \mathcal{L}(PR_{ac}^\delta) = \mathcal{L}(RC_{ac}^\delta)$, proved in /7/.

THEOREM 2. a) $\mathcal{L}(RC_\alpha, M_\beta^\delta) = \mathcal{L}(M_{max(\alpha,\beta)}^\delta)$;

b) $\mathcal{L}(RC_\alpha, PR_\beta^\delta) = \mathcal{L}(PR_{max(\alpha,\beta)}^\delta)$;

c) $\mathcal{L}(RC_\alpha, SC_\beta^\delta) = \mathcal{L}(SC_{max(\alpha,\beta)}^\delta)$.

4.2. Restrictions imposed to random context grammars

We consider now the following types of grammars:

i) $(M_\alpha, RC_\beta^\delta)$ - grammars with matrices of random context rules: $((A_1 \longrightarrow x_1, R_1, Q_1), \ldots, (A_k \longrightarrow x_k, R_k, Q_k))$,

ii) $(PR_\alpha, RC_\beta^\delta)$ - grammars with rules of the form $(r, A \longrightarrow x, R, Q), \sigma_r, \varphi_r)$; if $(A \longrightarrow x, R, Q)$ is effectively applicable, then we pass to σ_r, if $(A \longrightarrow x, R, Q)$ cannot be applied (either A does not occur in the current string, or the restrictions imposed by R,Q are not observed), then we pass to φ_r,

iii) $(C_\alpha, RC_\beta^\delta)$ - random context grammars with regular control languages associated to them,

iv) $(SC_\alpha, RC_\beta^\delta)$ - as at $(M_\alpha, RC_\beta^\delta)$, with scattered context derivation.

THEOREM 3. a) $\mathcal{L}(M_\alpha, RC_\beta^\delta) = \mathcal{L}(M^\delta_{max(\alpha,\beta)})$;

b) $\mathcal{L}(PR_\alpha, RC_\beta^\delta) = \mathcal{L}(PR^\delta_{max(\alpha,\beta)})$;

c) $\mathcal{L}(C_\alpha, RC_\beta^\delta) = \mathcal{L}(C^\delta_{max(\alpha,\beta)})$;

d) $\mathcal{L}(SC_\alpha, RC_\beta^\delta) = \mathcal{L}(SC^\delta_{max(\alpha,\beta)})$.

5. Final remarks

As two superposed restrictions generally behave as only one as generative capacity, three or more superposed restrictions are not of interest. The same result can be obtained for other pairs of regulated devices. Such possible pairs (of compatible restrictions) are (PR,M), (C,M), (PR,SC), (C,SC); using the usual mutual simulation procedures of M, PR, C we can easily obtain equalities of the form $\mathcal{L}(X_\alpha, Y_\beta^\delta) = \mathcal{L}(Y^\delta_{max(\alpha,\beta)})$ for (X,Y) as above.

Interesting results can be probably obtained by imposing further restrictions on valence grammars in /8/; as the valence restriction does not determine the order of the rule application,

all restrictions on order (M, PR, C, O) or on context (RC, SC) can be added.

As the generative capacity is not increased by imposing a further restriction to a regulated device, it remains to look for a possible utility of such a hybrid from other points of view, for instance, taking into account the descriptional complexity of languages.

REFERENCES

1. A.B. CREMERS, Normal forms for context sensitive grammars, Acta Informatica, 3 (1973), 59 - 73.

2. J. DASSOW, On some extensions of Indian parallel context-free grammars, Acta Cybernetica, 4 (1980), 303 - 310.

3. J. DASSOW, On some extensions of Russian parallel context-free grammars, Acta Cybernetics, to appear.

4. J. DASSOW, GH. PĂUN, On extensions of some grammars by order relations, Elektr. Inform. und Kybern. EIK, to appear.

5. J. FRIS, Grammars with partial ordering of the rules, Information and Control, 12 (1968), 415 - 425.

6. S.A. GREIBACH, J.E. HOPCROFT, Scattered context grammars, Journal of Computer and Systems Science, 3 (1969), 233 - 247.

7. O. MAYER, Some restrictive devices for context-free grammars, Information and Control, 20 (1972), 69 - 92.

8. GH. PĂUN, A new type of generative device: valence grammars, Revue Roumaine Math. Pures et Appl., 25 (1980), 911 - 924.

9. GH. PĂUN, Matrix grammars, The Scientific and Encyclopaedic Publ. House, Bucharest, 1981 (in Rom.).

10. A. SALOMAA, Formal languages, Academic Press, New York, London, 1973.

ON THE ANALYSIS AND SYNTHESIS OF FINITE MEALY-AUTOMATA

Csaba Puskás
Institute of Mathematics and Computer Science
K. Marx University of Economics, Budapest

1. Abstract

There is given a new method for describing automaton mappings induced by finite Mealy-automata and, conversely, for constructing finite cyclic reduced Mealy-automata realizing every element of given finite sets of finite automaton mappings.

2. Introduction

A *Mealy-automaton* is a quintuple $\mathfrak{A} = (A,X,Y,\delta,\lambda)$, where A is the set of states, X is the set of inputs, Y is the set of outputs, $\delta : A \times X \to A$ is the transition function and $\lambda : A \times X \to Y$ is the output function. We shall assume that δ and λ are extended in form

$$\delta : A \times X^* \to A \quad \text{and} \quad \lambda : A \times X^* \to Y^*$$

in the usual way, where X^* denotes the free monoid over the set X. Let $X^+ = X^* - \langle e \rangle$, where e is the empty word and for $p \in X^+$ let $\overline{p}$ denote the last letter of p. Moreover, for $p \in X^*$ $|p|$ means the length of p. $\mathfrak{A}$ is said to be *finite* if A,X and Y are finite sets. It is supposed that all of

automata considered in this paper are finite ones. We say that $\mathfrak{A}$ is *cyclic* if it has a *generator state* $a_o \in A$ for which

$$(\forall\, a \in A;\ \exists p \in X^*)\colon\ a = \delta(a_o, p).$$

We correspond to $\mathfrak{A}$ a set of automaton mappings $\Phi_{\mathfrak{A}} = \langle \alpha_a : X^* \to Y^* \mid a \in A \rangle$, where α_a is defined by

$$\forall\, p \in X^* :\ \alpha_a(p) = \lambda(a,p)\ .$$

$\mathfrak{A}$ is called *reduced* if the implication

$$\alpha_a = \alpha_b \Rightarrow a = b \qquad (a,b \in A)$$

is true. We say that an alphabetic mapping $\alpha : X^* \to Y^*$ is a *finite automaton mapping* if there is a finite automaton $\mathfrak{A}$, for which $\alpha \in \Phi_{\mathfrak{A}}$ holds. It is well known that the alphabetic mapping $\alpha\colon X^* \to Y^*$, with finite sets X and $Y = \langle y_1,\ldots,y_m \rangle$ is finite automaton mapping if and only if it satisfies the following conditions:

(i) $\forall\, p \in X^* :\ |\alpha(p)| = |p|$,

(ii) $\forall\, p,q \in X^* :\ \alpha(pq) = \alpha(p)\alpha_p(q)$,

(iii) $\forall\, y_i \in Y :\ L_{y_i} = \langle p \in X^+ \mid \overbrace{\alpha(p)} = y_i \rangle$ is a regular language.

A finite automaton mapping $\alpha : X^* \to Y^*$ can be given by a complete regular language vector (briefly ℓ-vector) $\underline{a} = [a_1,\ldots,a_m]$ over X. The *regularity* of $\underline{a}$ means that every component of $\underline{a}$ is a regular language and we say that $\underline{a}$ is *complete* if

$$i \neq j \Rightarrow a_i \cap a_j = \emptyset \quad (1 \leq i,j \leq m) \text{ and } \sum_{i=1}^{m} a_i = X^+$$

hold. The fact that there is a one-to-one correspondence between finite automaton mappings and complete regular ℓ-vectors can be verified as follows:

If $\alpha: X^* \to Y^*$ is finite automaton mapping then the corresponding complete regular ℓ-vector $\underline{a} = [a_1,\ldots,a_m]$ is determined by the connection

$$a_i = L_{y_i} = \langle p \in X^+ \mid \overline{\alpha(p)} = y_i \rangle \qquad (i=1,\ldots,m)$$

and conversely the complete regular ℓ-vector $\underline{a} = [a_1,\ldots,a_m]$ is just corresponded to the finite automaton mapping $\alpha : X^* \to Y^*$, for which $\alpha(e) = e$ and if $p = x_1x_2\ldots x_k \in X^+$ then

$$\alpha(p) = y_{i_1}y_{i_2}\ldots y_{i_k} \iff x_1 \in a_{i_1},\ x_1x_2 \in a_{i_2},\ldots,x_1x_2\ldots x_k \in a_{i_k}$$

$(1 \le i_1,i_2,\ldots,i_k \le m)$ hold.

For the notation and notions that will not be defined here we refer to [2,5].

3. Analysis of finite Mealy-automata

Let $A = \langle a_1,\ldots,a_n \rangle$ and $Y = \langle y_1,\ldots,y_m \rangle$ be the state set, respectively the output set of the finite Mealy-automaton $\mathfrak{A} = (A,X,Y,\delta,\lambda)$. By the *transition matrix* of $\mathfrak{A}$ we mean the language matrix (briefly ℓ-matrix) $\underline{N}$ of $n \times n$ type defined by

$$(\underline{N})_{ij} = \sum_{\delta(a_i,x)=a_j} x \qquad (x \in X;\ i,j=1,\ldots,n).$$

Similarly, the *output matrix* of $\mathfrak{A}$ is the ℓ-matrix $\underline{P}$ of $n \times m$ type given by

$$(\underline{P})_{ik} = \sum_{\lambda(a_i,x)=y_k} x \qquad (x \in X;\ i=1,\ldots,n;\ k=1,\ldots,m).$$

The following theorem shows that the analysis of $\mathfrak{A}$ is equivalent to the problem of solving the ℓ-matrix equation

$$\underline{Q} = \underline{N}\,\underline{Q} + \underline{P}\ .$$

THEOREM 1. *Let* $\mathfrak{A} = (A,X.Y,\delta,\lambda)$ *be a finite Mealy-automaton with state set* $A = \langle a_1,\ldots,a_n\rangle$ *and output set* $Y = \langle y_1,\ldots,y_m\rangle$. *Let* $\underline{N}$ *and* $\underline{P}$ *be the transition matrix and respectively the output matrix of* $\mathfrak{A}$. *Then the complete regular* ℓ*-vector* $\underline{a}_i$ *corresponding to the finite automaton mapping* α_{a_i} $(1 \leq i \leq n)$ *is just the* i*-th row in the solution matrix of the* ℓ*-matrix equation* $\underline{Q} = \underline{N}\,\underline{Q} + \underline{P}$.

P r o o f : Let $\underline{A}$ be the ℓ-matrix of $n \times m$ type defined by

$$(\underline{A})_{ik} = \langle p \in X^+ \mid \overbrace{\alpha_{a_i}(p)} = y_k\rangle \qquad (i=1,\ldots,n;\ k=1,\ldots,m),$$

that is, the i-th row of $\underline{A}$ is just the ℓ-vector $\underline{a}_i$ $(i=1,\ldots,n)$. We prove that $\underline{A} = \underline{N}\,\underline{A} + \underline{P}$ holds. Let $(\underline{A})_{ik}$ be an arbitrary component of $\underline{A}$ and let us consider an arbitrary word $p \subseteq X^+$. If $|p| = 1$ then $p = x$ for some $x \in X$ and we have that

$$x \in (\underline{A})_{ik} \iff \lambda(a_i,x) = \alpha_{a_i}(x) = y_k \iff x \in (\underline{P})_{ik} \iff$$

$$\iff x \in (\underline{N}\,\underline{A} + \underline{P})_{ik}\ .$$

If $|p| > 1$ then $p = xq$ with $x \in X$ and $q \in X^+$, therefore we obtain that

$$p \in (\underline{A})_{ik} \iff \overbrace{\lambda(a_i,p)} = \overbrace{\lambda(\delta(a_i,x),q)} = \overbrace{\alpha_{\delta(a_i,x)}(q)} = y_k \iff$$

$$\iff \exists j \ (1 \le j \le n) : x \in (\underline{N})_{ij} \ \& \ q \in (\underline{A})_{jk} \iff xq = p \in (\underline{N}\,\underline{A})_{ik} \iff$$

$$\iff p \in (\underline{N}\,\underline{A} + \underline{P})_{ik} ,$$

where j is the index of the state $\delta(a_i,x)$. On the other hand, the ℓ-matrix equation $\underline{Q} = \underline{N}\,\underline{Q} + \underline{P}$ has a unique solution. This fact can be obtained by a slight generalization of a Bodnarčuk's result on the solution of systems of linear equations in the algebra of languages (see [1] and [4] or [5]). □

In [5] was pointed out that the solution of the ℓ-matrix equation $\underline{Q} = \underline{N}\,\underline{Q} + \underline{P}$ can be determined by subsequent elimination of unknown rows of $\underline{Q}$ and at the same place an illustrative example can be found as well. Thus, by the previous theorem we can assert that the algorithm for solving the ℓ-matrix equation $\underline{Q} = \underline{N}\,\underline{Q} + \underline{P}$ is an algorithm for the analysis of $\mathfrak{M}$.

4. Generalized synthesis problem of finite Mealy-automata

Everywhere in this paragraph X and Y will denote finite nonempty sets, with $Y = \langle y_1,\ldots,y_m\rangle$. As in the introduction it was shown any finite automaton mapping $\alpha : X^* \to Y^*$ can be given by a complete regular ℓ-vector $\underline{a}$. In [5] we have

defined the *left side e-free derivations* of ℓ-vectors with respect to a word $p \in X^*$ as follows; the left side e-free derivation of $\underline{a} = [a_1,\ldots,a_m]$ with respect to the word $p \in X^*$ is the ℓ-vector ${}_\ell D_p^-(\underline{a}) = [{}_\ell D_p^-(a_1),\ldots,{}_\ell D_p^-(a_m)]$, where

$$ {}_\ell D_p^-(a_i) = \langle q \in X^+ \mid pq \in a_i \rangle \quad (i=1,\ldots,m) . $$

It can be easily verified that if $\underline{a}$ is complete and regular then ${}_\ell D_p^-(\underline{a})$ is also complete and regular for all $p \in X^*$. Moreover, the regularity of $\underline{a}$ implies that the set of all different left side e-free derivations of $\underline{a}$ is a finite set.

In [5] was proved the following

THEOREM 2. *Let* $\alpha : X^* \to Y^*$ *be a finite automaton mapping and let* $\underline{a} = [a_1,\ldots,a_m]$ *be the corresponding complete regular ℓ-vector. Then* α *can be induced by the finite cyclic reduced Mealy-automaton* $\mathfrak{M} = (A,X,Y,\delta,\lambda)$, *with* $A = \langle {}_\ell D_p^-(\underline{a}) \mid p \in X^* \rangle$, *in which* $\underline{a} = {}_\ell D_e^-(\underline{a})$ *is the generator state and the functions* δ *and* λ *are defined for all* ${}_\ell D_p^-(\underline{a}) \in A$ *and* $x \in X$ *by*

$$ \delta({}_\ell D_p^-(\underline{a}),x) = {}_\ell D_{px}^-(\underline{a}) \text{ and } \lambda({}_\ell D_p^-(\underline{a}),x) = y_k \iff x \in {}_\ell D_p^-(a_k) \quad (1 \le k \le m). $$

More precisely $\alpha = \alpha_{\underline{a}}$ *holds.*

Now we consider a finite set $\langle \alpha_i : X^* \to Y^* \mid i \in I \rangle$ of finite automaton mappings and verify that a finite cyclic reduced Mealy-automaton can be constructed which induces every mapping α_i $(i \in I)$. For all $i \in I$ let $\underline{a}_i = [a_{i1},\ldots,a_{im}]$ be the complete regular ℓ-vector corresponding

to α_i. By the previous theorem it is enough to prove that there is a complete regular ℓ-vector $\underline{b} = [b_1,\ldots,b_m]$ such that every $\underline{a}_i$ $(i \in I)$ can be obtained as a left side e-free derivation of $\underline{b}$ with respect to some word $p \in X^*$. This is, that will be proved in the following theorem.

THEOREM 3. *Let* $\langle \underline{a}_i = [a_{i1},\ldots,a_{im}] \mid i \in I\rangle$ *be a finite set of complete regular* ℓ*-vectors over* X. *Then there exists a complete regular* ℓ*-vector* $\underline{b} = [b_1,\ldots,b_m]$ *over* X *such that for all* $i \in I$

$$\underline{a}_i = {}_{\ell}D_p^{-}(\underline{b})$$

holds with some $p \in X^*$.

P r o o f: Let n be the less natural number for which $\mathrm{card}(I) \leq \mathrm{card}(X^n)$, where X^n denotes the set of all words $p \in X^*$ with $|p| = n$. Moreover, let $\langle L_i \mid i \in I\rangle$ be a set of pairwise disjunct languages such that

$$\sum_{i\in I} L_i = X^n .$$

Finally, let $\underline{b}' = [b_1',\ldots,b_m']$ be an ℓ-vector over X for which

$$j \neq k \Rightarrow b_j' \cap b_k' = \emptyset \quad (1 \leq j,k \leq m) \text{ and } \sum_{j=1}^{m} b_j' = X+\ldots+X^n .$$

We form the left side linear combination of ℓ-vectors $\underline{a}_i$ $(i \in I)$ with languages L_i $(i \in I)$ as follows:

$$\sum_{i\in I} L_i\underline{a}_i = \Big[\sum_{i\in I} L_i a_{i1},\ldots,\sum_{i\in I} L_i a_{im}\Big],$$

where $L_i a_{ij}$ $(1 \leq j \leq m)$ means the catenation of languages L_i and a_{ij} and $\sum_{i \in I} L_i a_{ij}$ means the sum of languages $L_i a_{ij}$ $(i \in I)$.

Let $\underline{b}$ be the ℓ-vector given by

$$\underline{b} = \sum_{i \in I} L_i \underline{a}_i + \underline{b}' .$$

We shall prove that $\underline{b}$ satisfies the demanded requirements.

It is obvious that $\underline{b}$ is regular since the catenation and the sum of regular languages is regular. Let us assume that $j \neq k$ but $p \in b_j \cap b_k$ holds for some $p \in X^+$. Since $j \neq k$ implies that $b'_j \cap b'_k = \emptyset$, the length of p must be greater than n and

$$p \in \sum_{i \in I} L_i a_{ij} \cap \sum_{i \in I} L_i a_{ik} .$$

But $|p| > n$ implies that $p = qr$ $(q,r \in X^+)$ with $|q| = n$ and taking into account that $\langle L_i \mid i \in I \rangle$ is a partition of X^n,

$$p = qr \in \sum_{i \in I} L_i a_{ij} \cap \sum_{i \in I} L_i a_{ik}$$

implies that there exists a unique $i \in I$ such that

$$p = qr \in L_i a_{ij} \cap L_i a_{ik} .$$

Since $L_i \subseteq X^n$ it follows that

$$q \in L_i \quad \text{and} \quad r \in a_{ij} \cap a_{ik} ,$$

which contradicts to the completeness of the ℓ-vector $\underline{a}_i$.

On the other hand,

$$\sum_{j=1}^{m} b_j = \sum_{j=1}^{m} \Big(\sum_{i\in I} L_i a_{ij} + b'_j \Big) = \sum_{i\in I} L_i \sum_{j=1}^{m} a_{ij} + X + \ldots + X^n =$$

$$= \sum_{i\in I} L_i X^+ + X + \ldots + X^n = X+\ldots+X^n+X^nX^+ = X^+.$$

Thus we have got that $\underline{b}$ is a complete regular ℓ-vector. To verify that every $\underline{a}_i$ $(i \in I)$ can be obtained as a left side e-free derivation of $\underline{b}$, let p be any word from L_i for arbitrarily fixed $i \in I$. Then, taking into account that $|p| = n$ implies that for all $j(=1,\ldots,m)$ ${}_\ell D_p^-(b'_j) = \emptyset$, and for arbitrary languages L and $\hat{L}$

$${}_\ell D_p^-(L.\hat{L}) = {}_\ell D_p^-(L)\hat{L} + \sum_{qr=p} \delta_q(L)\, {}_\ell D_r^-(\hat{L})$$

holds, where

$$\delta_q(L) = \begin{cases} e & \text{if } q \in L, \\ \emptyset & \text{if } q \notin L, \end{cases}$$

we have got that

$${}_\ell D_q^-(\underline{b}) = \Big[{}_\ell D_p^- \Big(\sum_{i\in I} L_i a_{i1} + b'_1 \Big), \ldots, {}_\ell D_p^- \Big(\sum_{i\in I} L_i a_{im} + b'_m \Big) \Big] =$$

$$= \Big[\sum_{i\in I} {}_\ell D_p^-(L_i a_{i1}), \ldots, \sum_{i\in I} {}_\ell D_p^-(L_i a_{im}) \Big] =$$

$$= [a_{i1},\ldots,a_{im}] = \underline{a}_i \ . \ \square$$

Constructing the finite cyclic reduced Mealy-automaton inducing the automaton mapping determined by the ℓ-vector $\underline{b}$ according to the Theorem 2 it will induces every finite automaton mapping α_i of the given set $\langle \alpha_i : X^* \to Y^* \mid i \in I \rangle$.

REFERENCES

[1] BODNARČUK, V. G., Системы уравнений в алгебре событий, Журн. вичисл. матем. и матем. физ., 3(1963), 1077-1088.

[2] GÉCSEG, F. and PEÁK, I., Algebraic Theory of Automata, Akadémiai Kiadó, Budapest, 1972.

[3] PEÁK, I., Bevezetés az automaták elméletébe.II. /Introduction to the Theory of Automata, Vol.II, in Hungarian/ Tankönyvkiadó, Budapest, 1978.

[4] PUSKÁS, CS., Matrix equations in the algebra of languages with applications to automata, Papers on Automata Theory, IV., K. Marx Univ. of Economics, Dept. of Math., Budapest, 1982, No. DM 82-1, 77-89.

[5] PUSKÁS, CS., A common method for analysis of finite deterministic and non-deterministic automata, Papers on Automata Theory, V., K. Marx Univ. of Economics, Dept. of Math., Budapest, 1983, No. DM 83-3, 77-90.

• ARTIFICIAL INTELLIGENCE •

CONSONANCE OF ACTIVE KNOWLEDGE BASE WITH FUZZY RELATIONS

A.N.Averkin, S.K.Dulin

The Computer Center of the Academy of Sciences of the USSR
Moscow
U S S R

The paper deal with one of the ways to maintain the integrity of data base in the sense of Heider's consonance of special ternary relations on data base components. The consonance of data base is expecially important in the case of active data base, i.e. when it includes some procedures of knowledge generation. The verification of active data base consonance may be realized using ordinary, fuzzy or linguistic relations between components.

INTRODUCTION

The procedure of data base designing suggests the definition of informational model of the problem environment to a precision of the names of the object's types and to a precision of the specification of links between them. The name of the object's type defines several objects of this type distinguished by their attributes. The structuration of data base is based on fixation of links between the pairs of objects. Omitting the problems of data base designing we'll point out several important distinctions between data base organization and knowledge base organization. At first, knowledge simulation and knowledge representation have special forms, which reflect knowledge-dependent inner structure of every knowledge type. At second, knowledge structuration suggests much more strong inter-knowledge relations and in general case they can't be presented only by binary relations.

The inner knowledge representation is difficult to discuss without problem environment description. But as far as general structuration problem is concerning, the problem-environment independent areas exist. One of such areas is the knowledge base integrity problem which is unsolvable without elimination of contradictions inside of the knowledge set, i.e. without transformation of the knowledge base into consonance state.

TYPES OF RELATIONS

By analogy with data base we define knowledge base as interconnected set of knowledge pieces. We consider knowledge base as further development of data base from the point of view of introduction of new relations on data types. As a result fragments of declarative and procedural knowledge arise and they produce semantic network. In formal way we may present the knowledge base as quintuple (T, L, R, G, P), where T - a set of terms; L,R and G - local, regional and global relations; P - a set of knowledge base management proce-

dures. The global relations define logical structure of knowledge base and they have the same semantic meaning as they have in data base, i.e. they provide the general scheme of the system on the base of hierarchy, priority and membership of system components. The local relations are closely connected with concrete data types (terms) and may be regarded as undivided integrity with such data. For example, the local relations are used for generation of frame knowledge bases. Frame models [1] greatly improve the adequacy and dynamic of information support in comparison with data bases, where any change of structure is impossible without participance of data administrator. The modification of frame model suggests the human interference in connection of inner frame structure. Active inner term-level regeneration of knowledge system is possible only without strictly plotted pieces of problem-oriented knowledge.

Otherwise we ought to consider such pieces of knowledge as terms. The knowledge generation needs the special tools for generation of inner structure, which are based on the knowledge system relation, In this case the relations on data types become regional and take part in logical-procedural representation. Here we may point out the possibility of uncertain knowledge representation using fuzzy sets theory. Fuzzy relations can be used on the frame level and on the level of regional relations. In general case each frame needs it's own linguistic scale and it's own rules of membership function evaluation (semantic rules). On the contrary the regional relations can be universal. This fact allows to construct a number of rather simple and effective consonance procedures using fuzzy models.

APPROACH TO CONSONANCE PROCEDURES

One of the central questions of knowledge base generation is the support of knowledge base in consonance state independently on every regional relation. The support of knowledge base consonance is the generalization of the procedure of data integrity support in data base. As it was mentioned above, it can't be based only on the analysis of binary relations between pieces of knowledge. To investigate active knowledge base consonance we use the method which was developped for determination of internal inconsistensy of the system [2]

Consonance ternary relations were defined by Heider [3] using consonance functions $\phi(x,y,z) = xyz \vee \bar{x}\bar{y}z \vee \bar{x}y\bar{z} \vee x\bar{y}\bar{z}$, where x,y and z are boolean variables, which characterised the existense of given relation between all pairs of components of certain triple. Ternary relation is called consonance if $\phi(x,y,z) = 1$. The system is called consonance if all the triples of components gives the consonance relations. Here we point out two types of binary relations used in definition of consonance. The first type is based on threshold estimation of relation between elements and correspond to ordinary boolean matrix. The second type suggest the existence of fuzzy relations between elements, that corresponds to matrix with coefficients from [0,1] or from the term-set of given linguistic scale. For the second type ternary relation is called consonance, if $\phi(x,y,z) \geq 0,5$, where x, y, z - fuzzy grades of links between elements or linguistic approximation LA $[\phi(x,y,z)]$ $\in$ {true, not very true, very true, very very true etc.} in the case of linguistic values of variables.

The introduction of fuzzy relations especially linguistic rela-

tions, needs several special procedures of fuzzy information processing (so-called direct and inverse linguistic approximation, storing of the meanings of fuzzy linguistic terms, the adaptation, of the knowledge base to the new problem area). This adaptation is closely connected with transforamtion to universal scale [4] or with the invariancy principle for fuzzy linguistic scale given by modelling relation [5]. Inspite of the fact that nonfuzzy model is the special case of fuzzy ones, algorithms of consonance approximation, using maxmin criteria instead of additive ones, are much more simple.

The problem of consonance approximation of the relation with minimal number of changed links between elements is equivalent to the problem of maximal cut of the graph and has no effective algorithms In [6] it is described one of the heuristic consonance approximation algorithms which allows to reduce the search to find the nearest consonance relation to the given relation. The description of several such algorithms is given in [7], where the procedures of transitive closure and transitive approximation are used. They are based on following ideas.

It is easy to show that relation is consonance if and only if it induces the partition on the objects and forms two disjoint classes, so that $\phi(x,y,z) \geq 0,5$ for the objects inside the classes and $\phi(x,y,z) < 0,5$ otherwise. In particular, fuzzy equivalence relation is consonance, if it forms exactly two classes for $\alpha = 0,5$. In this case the transitive closure of similaruty relation S gives also the algorithm of construction of fuzzy consonance relation on the object set. The set of all α, such that α - cut $\hat{S}_\alpha$ induces exactly two classes of equivalence is an interval $[\alpha_1, \alpha_2]$ and in general case it doesn't include 0,5. But we can monotonically transform the values of the relation S, so that the new relation S will give such interval $[\alpha_1', \alpha_1']$, that $0,5 \in [\alpha_1', \alpha_1']$. As far as the results of classification are invariant under the monotonic transformations, the obtained relation will be fuzzy consonance relation.

If we use the standart method of transformation of the relation into consonance state, i.e. the replacement the values s_{ij} of the relation S by the opposites ones, $\bar{s}_{ij} = 1 - s_{ij}$, we can't use the algorithm of transitive closure. But we'll base on the assumption, that the most rational variant for the system is to change the values with minimal amplitude. Hence we'll minimize functional $\max_{s_{ij} \in S^*} | s_{ij} - s_{ij} |$, where S* - the set of all values, changed during the transformation into consonance state. The suggested algorithm searchs for cut of the set M, which includes the most "heavy" links s_{ij} among $s_{ij} \leq 0,5$. If the search is failed, then the algorithm additionally considers the links $0,5 \leq s_{ij} \leq 0,5 + \alpha$. The value of α monotonically increases until the α-cut will be found.

We need much more complex algorithm, if the fuzzy relation S_2 of type 2 is given. Using the algorithm of transitive approximation with some additional conditions [7] we get such fuzzy equivalence relation S*, that $\min_{ij} \mu_{ij}(t^*_{ij})$ is the largest consonance relation among all consonance relations with the same partition.

The above mentioned procedures may also use information, given as

modelling relation [5] (of type 1 or of type 2), if the values s_{ij} can't be obtained directly. Then S can be defined by the formula $S = R \circ R'$ or $S = R \tilde{\circ} R'$, where $\circ$ and $\tilde{\circ}$ - maximum and generalizated maxmin operations. Thus, modelling relation may be used for knowledge representation in active knowledge bases.

REFERENCES

1. Minsky M., A Framework for Representing Knowledge, Massachusetts Institute of Technology, Cambridge, 1974.
2. Dulin S.K., Consonance Function Application for the Analysis of Inner Contradiction State of an Object Set, Second International Conference "Artificial Intelligence and Information-Control Systems of Robots", Bratislava, 1982 (in Russian).
3. Dulin S.K., Research into Dissonance Networks, Technical Cybernetics, 1982, N°5 (In Russian).
4. Ezhkova I.V., Pospelov D.A., Decision Making in Fuzzy Environment. The Universal Scale, Technical Cybernetics, 1977, N°6 (In Russian).
5. Averkin A.N., Fuzzy Modelling Relation in Sheduling Systems of Robot Behaviour, Second International Conference "Artificial Intelligence and Information-Control Systems of Robots", Bratislava, 1982 (In Russian).
6. Averkin A.N., Dulin S.K., Active Knowledge Systems, Second Czechoslovak-Soviet Conference of Young Computer Scientists, Bratislava, 1982 (In Russian).
7. Averkin A.N., Dulin S.K., Fuzzy Modelling Relation for Robots, International Symposium on Artificial Intelligence, Leningrad, 1983 (In Russian).

ПРИНЦИПЫ ОРГАНИЗАЦИИ ВНУТРЕННЕГО ЯЗЫКА СИСТЕМЫ РАСПРЕДЕЛЁННОЙ ОБРАБОТКИ ИНФОРМАЦИИ.

Барилов А.А.
Ленинград, Менделеевская линия I, ЛНИВЦ АН СССР
Торгашев В.А.
Ленинград, Менделеевская линия I, ЛНИВЦ АН СССР

В настоящее время при решении задач с помощью ЭВМ обычно используется алгоритмический подход, заключающийся в том, что для задачи составляется алгоритм её решения или, иными словами, явно определённая последовательность действий, которые необходимо выполнить для преобразования исходных данных в результат. Такой подход имеет многовековую историю, являясь естественным для сознательного человеческого мышления, хорошо согласуется со структурой современных машин традиционного типа и достаточно эффективно применяется при решении научно-технических задач, в которых, в основном, обрабатывается числовая информация.

В то же время существует много практически важных задач (в частности, большинство задач управления, планирования и проектирования, различного рода моделирование и любые задачи, связанные со структурными преобразованиями), для которых применение алгоритмического подхода приводит к неоправдано сложным решениям. Поэтому в последние годы большое внимание уделяется альтернативным подходам, использующим представление задачи в виде структурного описания предметной области, к которой относится задача, с помощью функциональных семантических сетей или фреймов. Такой подход позволяет существенно тупросить программирование сложных задач, сделав его доступным для непрофессиональных программистов [1,2]. Однако, при его реализациях возникает ряд серьёзных проблем, обусловленных тем, что для решения задачи на машине традиционного типа необходимо иметь программу, представленную в алгоритмической форме. Переход от семантических сетей к алгоритмам требует сложных трансляторов и значительного объёма оперативной памяти для размещения системных средств. Кроме того структура современных ЭВМ практически не позволяет использовать естественный параллелизм задач, полностью сохраняемый в семантических сетях (в отличие от алгоритмического подхода) для сокращения времени решения задачи. Поэтому возникает

проблема разработки таких ЭВМ, которые могли бы непосредственно реализовывать структурный метод представления задач без перехода к алгоритмам.

В работах 3,4 была предложена новая модель вычислений, обладающая высокой конструктивностью и позволяющая создавать высокоэффективные ЭВМ в основе которых лежит представление задач близкое к семантическим сетям или фреймам. В основе этой модели лежат динамические автоматные сети (ДАС), обеспечивающие представление любой задачи в виде динамической структуры, то есть в виде множества объектов, связанных отношениями. Как объекты, так и отношения сами могут являться структурами. Динамизм структуры заключается в том, что она наделяется способностью к автотрансформации, то есть в состав структуры включаются такие элементы, которые способны изменять её. Исходной программе соответствует некоторая начальная структура, к трансформации которой и сводится решение задачи. Задача считается решённой, если в программной структуре не остаётся ни одного элемента, способного изменять эту структуру. В ДАС каждому элементу структуры сопоставляется конечный автомат (исключение составляют лишь некоторые примитивные отношения, такие как ПРИНАДЛЕЖНОСТЬ, СЛЕДОВАНИЕ, ЭКВИВАЛЕНТНОСТЬ и ряд других, которым сопоставляются межавтоматные связи) , который не только преобразует входные сигналы в выходные, но и обеспечивает формирование таких воздействий на коммутационную среду (обеспечивающую межавтоматные связи), которые позволяют изменять связи автомата с другими автоматами сети, ликвидировать все связи, что эквивалентно уничтожению автомата, включать в состав сети новые автоматы, обеспечивая тем самым её развитие.

Любой ДАС можно сопоставить виртуальную машину, архитектура которой динамически изменяется в соответсвии с изменением структуры ДАС в ходе решения задачи. Такая машина называется машиной с динамической архитектурой (МДА).

Любая МДА состоит из двух основных частей - операционной и коммутационной. Операционная часть соответствует множеству автоматов конечного числа типов, из которых и строятся любые сети, а коммутационная часть соответствует коммутационному полю, обеспечивающему динамическое изменение межавтоматных связей. Наиболее естественной формой такого поля является комутационная автоматная сеть с регулярной структурой, состоящая из специализированных коммутационных автоматов. На данном этапе развития технологии целесообразно сопоставлять реальное устройство не отдельно взятому автомату а множеству автоматов. Так множество операционных автоматов отображается в вычислительные модули (ВМ), каждый из которых соответствует вычислитель-

ной машине, включающей в свой состав процессоры, обеспечивающие реализацию автоматных функций, оперативную и внешнюю память, обеспечивающую хранение описаний автоматов и их состояний, и каналы ввода вывода, осуществляющие взаимодействие автоматов с внешней средой (периферийными устройствами). Множество коммутационных автоматов отображается в коммутационные модули (КМ), соответствующие специализированным процессорам, по своим функциям близким к связным процессорам, используемым в вычислительных сетях. В целом по своей структуре МДА соответствует сети ЭВМ, однако по своей архитектуре любой вычислительный модуль принципиально отличается от традиционных машин. Ниже будут рассмотрены основные черты внутреннего языка МДА, определяющего в целом архитектуру таких машин, имеющих высокий уровень внутреннего "интеллекта".

Внутренние языки любых современных машин, включая нетрадиционные, содержат элементы двух принципиально различных классов - команды и данные. Программа, определяющая алгоритм решения задачи представляет из себя жёстко определённую последовательность команд, остающихся неизменными в ходе её решения. Данные, являющиеся объектами преобразования со стороны программы, непосредственно не входят в её состав. Несмотря на принципиальное различие (команды не могут являться объектом преобразования, а данные не могут выполняться) команды и данные не содержат в своём составе признаков,позволяющих отличить их друг от друга.

Внутренний язык МДА (называемый в дальнейшем ДАР) состоит из программных элементов (ПЭ) достаточно близких друг к другу как по структуре, так и по использованию в машине, поскольку каждый из этих элементов соответствует автомату ДАС. Любой элемент может выполняться и, в свою очередь, быть объектом преобразования. Программа, представленная в МДА, соответствует сети программных элементов, и процесс решения задачи заключается в преобразовании исходной сети в конечную сеть, соответствующую результату.

Каждый ПЭ состоит из трёх основных частей: описания, состояния и связей. Описание определяет функцию, выполняемую автоматом, соответствующим ПЭ, а также определяет синтаксис ПЭ. Смысл остальных частей непосредственно определяется их названием.

Связи ПЭ соответствуют примитивным отношениям, связывающим данный элемент с другими ПЭ. Обычно в машинных языках используются два функциональных отношения АРГУМЕНТ и РЕЗУЛЬТАТ, связывающих команду с данными и неявное отношение ПРЕДШЕСТВОВАНИЕ (СЛЕДОВАНИЕ), определяемое с помощью соответствующего размещения элементов в памяти машины. В некоторых машинах с высоким уровнем внутреннего языка ис-

пользуется также отношение ПРИНАДЛЕЖНОСТЬ (ВКЛЮЧЕНИЕ), позволяющее определять древовидные структуры для данных или блочные структуры для команд. Поскольку язык ДАР ориентирован на непосредственное представление в машине произвольных структур, в его состав целесообразно включить помимо перечисленных выше ещё несколько примитивных отношений, таких как ЭКВИВАЛЕНТНОСТЬ, СООТВЕТСТВИЕ и РАВЕНСТВО. Любое примитивное отношение, связывающее ПЭ с другими элементами, является для ПЭ входящим или исходящим (симметричное отношение, например, РАВЕНСТВО, является одновременно и тем и другим). Примитивные отношения по своей структуре похожи на ПЭ, то есть они состоят из описания, определяющего тип отношения, состояния и связи.

В состоянии ПЭ можно выделить три основных элемента: состояние корректности (СК), состояние активности элемента (СА) и собственно состояние программного элемента (СЭ), соответствующее его значению.

Состояние корректности определяет степень правильности СЭ и может принимать 4 логических значения: ЛОЖЬ, НЕОПРЕДЕЛЁННЫЙ, БЕЗРАЗЛИЧНЫЙ и ИСТИНА (следует заметить, что в языке ДАР используется четырёхзначная логика). Для отношений СК совпадает с СЭ, причём отношение считается неопределённым, если элемент, для которого отношение является исходящим находится в неопределённом состоянии, и безразличным, если отношение неприменимо к элементам, которые оно связывает. Другие ПЭ переходят в ложное состояние при возникновении ошибок в ходе хранения или выполнения. Неопределённое состояние используется для целей управления вычислительным процессом в МДА. Безразличное состояние любых ПЭ (кроме отношений) используется для решения частично определённых задач.

Состояние активности определяет различные стадии выполнения ПЭ. Если в обычных машинных языках выполняемый элемент (команда) может находится в двух состояниях - ВЫПОЛНЕНИЕ и ХРАНЕНИЕ, то в ДАРе таких состояний имеется четыре: ХРАНЕНИЕ, АКТИВАЦИЯ, ВОЗБУЖДЕНИЕ И ГОТОВНОСТЬ. Состояние ХРАНЕНИЕ определяет отсутствие каких либо действий, выполняемых соответствующим автоматом. ПЭ, находящийся в этом состоянии хранится в оперативной или внешней памяти ВМ. При изменении СК любого примитивного отношения, входящего в ПЭ и находящегося в СА ХРАНЕНИЕ, ПЭ переходит в состояние АКТИВАЦИЯ и заносится в очередь на выполнение, состоящую из ПЭ, которые находятся в таком же состоянии. Выполнение любого ПЭ, находящегося в данном состоянии заключается в проверке состояний отношений, входящих в ПЭ. Если все эти отношения истинны, то ПЭ переходит в состояние ВОЗБУЖДЕНИЕ и ставится уже в другую очередь на выполнение, состоящую из ПЭ, находящихся в том же состоянии. Выполнение возбуждённого ПЭ

зависит от класса и типа элемента, а также от размещения других ПЭ (например, связанных с данным ПЭ отношением АРГУМЕНТ), участвующих в преобразовании, и либо осуществляется одним из процессоров ВМ, в котором размещена соответствующая очередь, либо передаётся в очередь на вывод в тот ВМ, где находятся соответствующие аргументы или аргумент. Аналогичная ситуация возникает и тогда когда длина очереди в текущем ВМ превышает некоторую пороговую величину.

После завершения выполнения элемент переходит в состояние ГОТОВНОСТЬ и попадает ещё в одну очередь. Выполнение любого элемента, содержащегося в этой очереди, заключается в переводе в состояние АКТИВАЦИЯ всех ПЭ, с которыми данный элемент связан исходящими отношениями. После завершения этого действия элемент переходит в состояние ХРАНЕНИЕ либо уничтожается (в зависмости от класса, типа и ряда вспомогательных признаков). Элементы, находящиеся в состоянии ХРАНЕНИЕ обычно находятся в очереди на передачу во внешнюю память. При любых перемещениях ПЭ внутри ВМ или в другие ВМ осуществляется соответствующее изменение связей во всех элементах связанных какими либо отношениями с перемещаемым ПЭ.

Каждой из перечисленных выше очередей можно сопоставить свой процессор, обеспечивающий выполнение элементов, находящихся в соответствующей очереди, то есть наиболее естественная структура ВМ соответствует многопроцессорной системе с относительно простыми функционально-ориентированными процессорами. Любой из этих процессоров берёт первый элемент из своей очереди, выполняет его и помещает либо в одну из других очередей, либо в свою собственную очередь, если выполнение элемента не было завершено, либо просто помещает в память. Поскольку порядок элементов в очереди является безразличным для выполнения программы и отстутствуют причины, изменяющие порядок ПЭ в очереди, то в процессорах ВМ можно организовать эффективную конвейерную обработку, обеспечив тем самым высокую производительность машины.

Все элементы внутреннего языка ДАР делятся на 5 основных классов: значения, операторы, отношения, ссылки и ресурсы.

Значения в целом соответствуют данным и обеспечивают представление на машинном уровне практически любых структур, применяемых в современных языках программирования.

Операторы языка ДАР обеспечивают произвольные структурные преобразования, включая и обычную числовую обработку, причём, как правило, в ходе решения задачи сначала происходит рост программы за счёт размножения операторов и связанных с ними значений, а к концу решения задачи программа уменьшается за счёт уничтожения отработавших

операторов и использованных значений. Все операторы на теоретическом уровне могут быть определены через единственный примитивный оператор структурной подстановки.

Отношения имеют тот же смысл, что и в семантических сетях и подобно любым другим ПЭ могут соответствовать произвольным структурам. Выполнение любого отношения соответствует проверке его правильности путём определения состояния корректности. С любым значением отношения может быть связана стандартная или определяемая пользователем программа реакции на указанное значение.

Ссылки являются исключительно мощным средством повышения эффективности языка и компактности программ. Они используются для определения примитивных отношений над неявно определёнными ПЭ либо сразу над множеством ПЭ. Как правило ссылки указывают на элементы некоторой структуры (то есть такие ПЭ, которые связаны с элементом, соответствующим собственно структуре, отношением ПРИНАДЛЕЖНОСТЬ) с помощью индексов (для последовательно упорядоченных структур), имён или образцов (ассоциативных признаков). В качестве элементов ссылок могут использоватья также кванторы ВСЕ и ЛЮБОЙ. Например, программа записанная на внешнем языке в виде выражения: (ВСЕ=папа ИЗ текст ЗАМЕНИТЬ НА ЛЮБОЙ ИЗ (папаша,отец,батя)), обеспечивающая замену всех слов "папа" на любой из указанных синонимов в заданном тексте, при представлении на языке РЯД содержит один оператор ЗАМЕНИТЬ (ПОДСТАНОВКА), связанный с аргументом и результатом с помощью ссылок, каждая из которых соответствует одному ПЭ.

Ресурсы являются элементами ДАР, используемыми для эффективного отображения программы на структуру реальной машины. Например, эти ПЭ позволяют обеспечить выполнение программы в конкретном модуле или процессоре или в заданной группе ВМ, использовать в ходе вычислений регистровую память для хранения промежуточных результатов, выдать информацию на конкретное внешнее устройство и т.д.

В заключение следует отметить, что как показали проведённые теоретические и экспериментальные исследования предложенный выше подход позволяет на порядок уменьшить объём системного и прикладного программного обеспечения при одновременном повышении эффективной производительности ЭВМ на I-2 порядка даже в случае использования одной и той же элементной и конструктивной базы за счёт существеного увеличения числа одновременно работающих процессоров.

Л И Т Е Р А Т У Р А

I. Кохра М.И., Колве А.П., Тыугу Э.Х. Инструментальная система программмирования ЕС ЭВМ ("ПРИЗ"). - "Финансы и статистика" М. I98I.

2. Тыугу Э.Х. Вычислительные фреймы и структурный синтез программ. - Техническая кибернетика, I982, №6, с.I2-20.

3. Торгашёв В.А. Управление вычислительным процессом и машина с динамической архитектурой. - В кн.: Вычислительные системы и методы автоматизации исследований и управления. М.:Наука, I982.

4. Пономарев В.М., Плюснин В.У., Торгашёв В.А. Распределённые вычисления и машины с динамической архитектурой: Препринт №54. Ленинград: ЛНИВЦ АН СССР, I982

IS THE IDEALIZED LOGIC PROGRAMMING FEASIBLE ?

Rudolf Fiby
Sviatoslav Molnar
Imrich Weigl

Institute of Technical Cybernetics
Slovak Academy of Sciences
Dubravska cesta 9
842 37 Bratislava
Czechoslovakia

Since the computer language PROLOG (PROgramming in LOGic), based on symbolic logic, has been designed and implemented by Colmerauer and Roussel in Marseille in 1972, there has been increasing research activity on such areas as expert systems, natural language understanding, plan formation, computer aided building design, compiler construction, data base description and guery, the solution of mechanics problems, natural language processing, e.t.c. PROLOG is not the ultimate version of the logic programming language. Several languages based upon logic have been created. However, in the actual implementations the logic programming languages are not yet completed.

What we expect from logic programming is that

- running a program, we make a logical proof of the solution of the given problem, that means, we do not have to care about the correctness of the program.
- the computation to be done forward and backward depending on the given environment.
  The formula $F(x,y)$ can be interpreted as $y = f(x)$, when y is the unknown value or $x = g(y)$, when the unknown value is x instead of y.

- both the order of the literals in the clause and the order of the clauses to be irrevelant to the correctness of the logic program, that means, the logic program could be seen as a collection of pieces of known facts.

Autors inform about the first version of the experimental logic programming system EXL implemented in Institute that has features as follows:

- EXL is based on the first-order Horn clauses.
- EXL always halts for programs not containing terms with functional symbols.
  None of the known PROLOG system satisfies that condition.
- unification is complete.
  Most of PROLOG implementations has logically incomplete unifier to increase the speed of computation.
- computation is bidirectional.
  Usually, PROLOG programs contain too many extralogical operators to make the program work efficiently, that effects one-directional computation only.
- the order of literals in clause as well as the order of clauses in programs is not critical.
  Changing the order of clauses and the order of literals in clause does not change the logic of the problem, still the execution of PROLOG programs leads to nonterminating loop or an incorrect solution is obtained.
- EXL is a built-in function of LISP 1.10 for PDP-11/40.
- EXL has its proper input language LPL similar to PROLOG.
- the cut operator is eliminated.
- the obscure parameter evaluation is avoided.
- the side effects as input/output primitives and variable/value assignment executions are eliminated.

For those abovementioned reasons the program is transparent.

To stress the deductive power of EXL a small but strong example is shown.

Suppose we have a single fact, say

REL(const1,const1) <- ,

and we define the transitive closure by rule

REL(var1,var3) <- REL(var1,var2),REL(var2,var3) .

The preceding clauses constitute a complete specification of the problem. Asking

<- REL(const1,const2) ,

whether const1 is in relation REL with const2, we can see that answer is NO. But running PROLOG, the execution leads to nonterminating loop that represents the incompleteness of the PROLOG proof procedure. The similar behavior have the other dialects of PROLOG and LOGLISP.

EXL halts returning the correct answer.

REFERENCES

1. K.L. Clark - S.A. Tarnlund : Logic Programming, Academic Press, London, New York, Paris, San Diego, San Francisco, Sao Paolo, Sydney, Tokyo, Toronto, 1982.
2. W.F. Cloksin - C.S. Mellish : Programming in Prolog, Springer-Verlag, Berlin, Heidelberg, New York, 1981.
3. R. Fiby - S. Molnar - I. Weigl : Sprava83, Internal report, Institute of Technical Cybernetics, Slovak Academy of Sciences, Bratislava, Czechoslovakia, 1983.
4. First International Logic Programming Conference, Faculte des Sciences de Luiny, Marseille, France, September, 14-17th, 1982.

THE INTELLIGENT APPLIED PROGRAM PACKAGE FOR INDUSTRIAL SYSTEM DEVELOPMENT - "PROJECT"

R.S.Rodin, N.A.Tsarevsky

The Computer Center of the Academy of Sciences of the USSR
Moscow
U S S R

This paper describes designing principles and structure of a program package for the planning of Industrial Systems capacities development. The planning offices personnel interacts with the package on its professional language. When solving the package uses original methods to handle with large-scale boolean programming problems.

The planning offices personnel (POP) engaged in industrial planning from the final goals usually encounters difficulties caused by the very large amounts of information to be taken into account.

Dealing with the large-scale arrays of input data characterizing the planning situation POP has to produce a number of alternative plans, evaluate them on the basis of the final goals and to choose the best one. Large dimensions of the planning problems, limitations on the resources to be allocated and the variety of industrial links make these problems very hard for the analysis and decision making.

It is generally accepted nowdays that for these purposes modern computers must be used. The formalization of different aspects of planning gave rise the development of mathematical methods while the widespread of programming languages provides their efficient computer realization. Also, to deal with highly structured and complexly organized data, the Data Base (DB) representation of information and the Data Base Management Systems (DBMS) were created. The computer development brings into play the possibilities to accumulate and store practically unlimited amounts of data and programs which may now be considered as knowledge accessable for automatic use for solving of applied problems.

Though POP are professionals in their field, usually are just dabblers in programming, applied mathematics and computer science.

Thus, an urgent necessity arises to create the intellectual systems which ehables users to describe the problems under consideration on his professionally oriented language and to solve them with the help of applied mathematics, programming and computer science technique.

The traditional approach to handle this problem is the developing of Program Packages (PP) technology [2,3] . Due to the complex inner structure PP's possesses quite new properties as compared to the isolated programs and information arrays. The principal merit

is the ability to solve problems formulated in professional terms of some applied area.

This report deals with the principles, structure and architecture of the PP PROJECT, the design of the Computer Center of the Academy of Sciences of the USSR.

PP PROJECT is developed for solving the project choice problems while planning new industrial capacities and, in particular, for finding the starting moments of their building and construction. The target plan must be balanced as for the resources as well as for the product output program. Such problems arise in planning procedures for the industrial systems (IS). The package under design is specially aimed for the mixed-boolean problem with block-trianguler constraints to which theses planning problem can be reduced.

The main components of the package are: problem area (PA), functial and system components.

1. The problem area description comprises the POP's knowledge about problem area concepts and connections among them, about mathematical class of problems under solvement, about routines of functional component, data structures and man-machine comunication language.

The description of the PA is arranged into three levels of description: the domain-oriented one, the mathematical and program levels [4].

The concepts of the domain-oriented level are as follows: planning period, resource, industrial capacities, industrial system, block of projects, investment and capacity development projects, finished product and complementing product, complementation matrix, task, plan, optimal plan.

Interrelations between the terms are caused by the fact, that on the domain-oriented level of PP the industrial system is described by the blocks of projects of capacity development intended for the product outputs. The connection between the finished and complementing products is given by complementation matrix. The industrial system, blocks, block's projects with their numerical characteristics, the goal of the IS, the IS program define the planning situation model. The plan for IS determines the moments of the beginning of the projects and, consequently, the summarized industrial expenditures and total volumes of finished products from these new capacities. The optimal plan is the plan which corresponds in a best way (in the sence of MIN-MAX) to the task of the industrial output (or to the task of capacities development, or expenditure of resources).

The main operations on the domain-oriented level are as follows: model construction operations (the description of the block of projects and the description of the industrial tasks), problem formulation and problem solving operations, data access operations and the report generation.

The operations on model description and manipulation with the information arrays can be, on the other hand, implemented by means of data management operations. The concepts in problem area and their interrelation can be represented by means of the data description

language. In this way the description of the problem area, models and the data manipulation operations can be realized by the standard DBMS means of the PP.

The package is intended, first of all, for the planning problems solving and gives to user the possibility to find the solution for the following problems:
- PROBLEM 1.
Find the plan of capacity development which under given level of available resources gives the best approximation (MAX-MIN) for the industry task of finished outputs.

- PROBLEM 2.

Find the plan which under the given level of available resources gives the best approximation for the industrial task of the capacity development.

- PROBLEM 3.

Find the plan of capacity development which under given level of finished product outputs gives the best way of resources utilization (MIN-MAX).

- PROBLEM 4.

The statement of this problem is analogeous to the Problem 1, but it takes in the consideration an inventry of product outputs.

- PROBLEM 5.

Find the plan of a block capacity development which minimizes the reduced resource.

These planning tasks are reduced to the mixed-integer boolean programming problems with block-triangular constrainsts.

On the mathematical level of problem area the description of these problems and algorithms for solving them are represented in the form of OR/AND graph. This graph is called subproblems network and has block structure. Each block of network corresponds to one of problems mentioned above and represents the algorithm for solving this problem. The subproblems with their types make up the set of verticies in the graph. There are three typical classes of subproblems in PP: mathematical subproblems (linear and boolean programming problems, linear algebra subproblems), DB-problems (subproblems of access to data corresponding to IS, blocks, resources) and subproblems for decision making (subproblems for methods selection or alternative problem solving way). Each type of subproblems has its name and attributes. For example, the linear programming problem has following attributes: the matrix of constraints, constraints vector, vector of variables linear criterion. The set of the arcs of the subproblems network is divided into two subsets: Γ-arcs and D-arcs. Γ-arcs define the logic of the algorithm and D-arcs describe exchanges of data between subproblems.

Problem statement and problem solving operations form the main operations of the mathematical level. User can stand the problem on planning situation model, points out the type of problem (1 - 5) and, possibly, changes input data. The executer of PP organizes the task solving process by using corresponding block of subproblems network. The relationship between domain-oriented and mathematical

levels of PA is performed by DB-problems, which carry out the interface between the data base and subproblems attributes.

The program modules descriptions constitute the program level of problem area. Each module has its name and input/output parameters.

The relationship between mathematical and program levels consists of correspondence descriptions between the names of subproblems and modules and the correspondence between subproblem's attributes and modules's parameters.

The representation of domain-oriented mathematical and program knowledges in PA gives the possibility for easy changing of a scheme of data base, introducing of new types of standard problems and adding modules without essential modification of PP.

2. The functional component of the PP consists of modules for typical subproblems solving. Standing and solving of the typical problems (1 - 5) demands a large amount of data, but the reception of precise desicion is difficult and not always need. Thus, the PP utilizes an iterative methods which are based on Danzig-Wolf decompositional scheme, applied to approximational problems for problems 1 - 4 and "branch and bound" method for problem 5. Search scheme for solving the standard problems 1 - 4 is based on sequential dealing with problem 5, [5] . The subproblems of mathematical level are the steps of solving algorithms for these standard problems. The linear programming problems in PP are handled by the Direct Simplex method or by the Danzig-Wolf decomposition method. For solving the boolean problems the method based on analysis of a search tree, "knapsack" algorithm and heuristic method for generation solutions for linear system of inequalities with boolean variables [6] are used. For dealing with linear algebra problems in package linear algebra routines are used.

3. The main functions of the system's component of the program package are following: a maintaining of communication between the user and PP, an algorithm generation and a controlling of problems solving process by using the modules from the functional component of the PP.

The monitor of PP accomplishes the monitoring functions and calls the subsystems of package (module definition, data access, report generation subsystems and executer).

First of all, PP construct the algorithm for solving of the planning problem which is described on the professionaly-restricted language. In the PP an algorithm is generated in accordance with a correspondent block of the subproblem network. The data base problems supply the mathematical subproblems with information.

The executer organizes the solving process by interpreting the correspondent block of subproblem network and calls a necessary modules from the system library, adjusting and supplying them by data.

The user interacts with the package on a rather simple language which concludes the professional terms from the planning area. The language has an hierarhical structure. The basic construction of the language is an instruction. Each instruction has defining key word and a list of parameters. The language instructions form three levels. The first level instructions give to user the possibilities

of model definition, problem stating, data accessing and description of forms for report generation. Performing these instructions monitor calls corresponding subsystems which languages constitute the second level of PP's language. For the dialogue with a program modules the third language level is used.

To illustrate the possibilities of the PP PROJECT let's consider typical dialogue with the system. First of all, the user id determing the planning situation model, describing the blocks, the projects of IS, task and resources of IS. The information about projects has been loaded to the data base under PP adjusting. On the next step of dialogue the user can modify the part of data state one of the standard PP's problems and initiate its solving. During a solving process the user is interacting with the functional modules and taking an active participence in a process of an optimal plan reseption. Has been recieved the plan the user, applying his own professional practice, analyses it and then modyfies an input data or describes and put into solving a new task. The solving of a practical planning problems is not always possible in dialogue mode because of large processing time. In this case the PP enaibles user to formulate a task and to fulfil it then in batch mode. The user can print an obtained solution in form of variable tables. The formats of these tables are described in generating report subsystem.

In this way PP PROJECT anables to the POP the possibilities for analyzing of many variants of capacities development plans and for reception of the well-grounded plan.

The program package PROJECT is under the design in the Computer Center, the Academy of Sciences of the USSR, as a part of Dialogue Industrial Planning System by using MAVR System [7] (Instrumental System for the program packages developing) and DBMS COMPAS [8] .

REFERENCES:

1. Problems of programmed goal planning and management., G.S.Pospelov Ed., Moscow, 'Nauka', 1981 (In Russian).
2. B.G.Tamm, E.Kh.Tyugu, Software packages, Izv. Akad. Nauk SSSR. Tekhnicheskaya Kibernetika, 1977, N°5 (In Russian).
3. A.P.Ershov, V.P.Ilyin, Software packages as a methodology of problem solving. In'Applied Software Packages, problems and perspectives', Moscow, 'Nauka', 1982 (In Russian).
4. S.R.Rodin, A.P.Shvalev, A.I.Erlikh, MAVR - Interactive system for modelling of alternatives and decision making. In 'Artificial Intelligence and Information - Control Systems of Robots', Smolenice, Institute of Technical Cybernetics of the Slovak Academy of Sciences, 1982 (In Russian).
5. N.A.Tsarevsky, Dynamic problem of development of an Industrial System Capacities., Moscow, Vych. Cent. Akad. Nauk SSSR, 1979 (In Russian).
6. N.A.Tsarevsky, An approximational algorithm for combinatorial problem of Industrial System Capacities development with block-triangular constraints., To appear (In Russian).
7. R.S.Rodin, MAVR System, Monitor of a Problem, To appear (In Russian).
8. V.I.Filippov, Data Base Management System COMPAS, Moscow, Vych. Cent. Akad. Nauk SSSR, 1982 (In Russian).

HYBRID TEXTURE RECOGNITION METHOD

Milan Šonka

Department of Control
Czech Technical University
Karlovo nám. 13
121 35 Praha 2
Czechoslovakia

1. Introduction

This contribution deals with the automatic recognition of textures. Texture can be found in all images from multispectral scanner images obtained from aircraft or satellite platforms to microscopic images of cell cultures or tissue samples. Despite its importance and ubiquity in image data, a formal approach or precise definition of texture does not exist. We think of texture as an organized area phenomena. When it is decomposable it has two basic dimensions on which it may be described. The first dimension is for describing the primitives out of which the image texture is composed, the second dimension is for the description of the spatial dependence among the primitives of an image texture. The primitives are maximally connected sets of pixels having a given tonal property.

Problems solved in this work are following -

- to describe a new method of texture recognition based on primitive extraction and on texture description by the number and types of extracted primitives and spatial organization of primitives
- to show the function of this method on real textures .

Methods of texture description are usually separated into two large groups. The first one includes symptomatic methods, the second one syntactic methods /3/,/4/,/5/,/6/. We do not

mention these two approaches to be of competition. We accept both of them. Proposed texture recognition system is hybrid and hierarchic, in one level symptomatic in the other syntactic approach predominates.

2. Hybrid Texture Recognition Method

2.1. Approach

Presented texture recognition method is neither symptomatic nor syntactic. This method /7/ is multiple-level hybrid one connecting advantages of both basic approaches. Hybrid method gives relatively simple computation and operational objectiveness as same as symptomatic methods. In the same time primitives are exactly defined what is a good property of syntactic methods.

In case texture recognition is made this way, the proposed method must content several following steps. In the first step it is necessary to extract primitives from digitized texture images, to describe and classify these primitives. After finishing this step classifier is able to classify any primitive described in given manner. In the second step texture samples are scanned, primitives are extracted, classified and spatial dependence among primitive kinds is computed. On the base of the spatial dependence feature vector classifier for texture sort discrimination is learned. The third step represents classification of unknown textures entering texture recognition system.

Presented method consequently has two levels of learning, in the recognition level classification is performed. In the first level classifier I is learned to classify primitives. The learning proceeds by the cluster analysis method - learning without teacher. It means classifier only needs examples of different primitives and information about the number of primitive classes. Important reason for using learning without teacher is impossibility to give right information about class of all primitives because of their great number.

In the second level the recognition system classifies primitives and on the base of this classification and computed spatial dependence among primitive kinds classifier II (working on a principle of minimum distance) is learned to classify textures. It is necessary to give an information about texture sort to the system now. This information is easy to give - the name of texture.

In the recognition level unknown texture is scanned, texture primitives are extracted, their characteristics are described and primitives are classified. On the base of primitive kinds the spatial dependence is computed and the texture sort is determined.

2.2. Primitive Extraction

The goal of primitive extraction is to segment texture image into primitives and background. Proposed hybrid texture recognition method is based on texture description by the number and types of extracted primitives and by spatial organization of primitives. For effective primitive extraction was rather modified Hong's et al. method /2/, based on grouping edges into primitive boundaries by joining facing pairs of edge points. This approach to primitive extraction realizes requirements for texture primitives very well. Our modification was designed to separate touching primitives.

2.3. Primitive Description

After extraction of primitives each primitive is an isolated object. To decide about similarity of primitives it is necessary to find their information-bearing features and to define a system of quantitative description of these qualitative properties.

We have found following suitable features for primitive description
- area
- average gray tone

- gray tone dispersion
- elongatedness
- rectangularity
- direction
- noncompactness
- straigtness of a boundary .

All algorithms for computation of these features are given in /7/. Further it is necessary to provide all primitives with identification numbers and to determine their position.

2.4. Spatial Dependence among Primitives

After foregoing operations the kinds of all primitives in the processed texture are known, the position of all primitives is known, too. For each primitive in every 45° cut the nearest neighbouring primitive is determined. On the base of nearest neighbours for all texture primitives a spatial dependence matrices M_j for every primitive class j is computed. It means the number of spatial dependence matrices is the same as the number of primitive classes. The value $M_j(i,k)$ tells, for instance, how often in processed texture primitive kind k was the nearest neighbour of primitive kind j in the i-th cut, etc.

Information of each spatial dependence matrix is reduced into a scalar. Each texture sample is described by the K-dimensional vector $\underline{p} = (p_1, p_2, \ldots, p_K)$, where K is the number of primitive classes

$$p_j = \frac{\sum_{i=1}^{8} \sum_{k=1}^{K} \left(M_j(i,k) \right)^2}{\sum_{j=1}^{K} \sum_{i=1}^{8} \sum_{k=1}^{K} M_j(i,k)} \quad . \qquad (1)$$

3. Experimental Results

Property of designed method was determined on real Brodatz texture samples /1/ and on TIROS satellite meteorological snaps.

On a set of four Brodatz textures - oriental straw cloth, heringbone weave, pressed cork and plastic pellets, a 93 % correct identification was obtained. The training set for primitive classifier learning contained 455 elements - primitives, primitives were classified into 8 classes. Training set for texture classification contained 59 texture samples. During recognition another set of 116 texture samples was processed. When the problem was reduced and the first three texture sorts - oriental cloth, heringbone weave and cork were processed, a 97 % correct classification was achieved. Another solved problem was to recognize two very similar textures - raffia and straw matting, a 92 % correct identification was obtained. In all up to now discussed problems the processed 50x50 image entered the minicomputer through the on-line connected TV system.

Next work was to identify three basic kinds of clouds (stratus, stratocumulus, cumulus) on a limited set of TIROS satellite meteorological data. Processed image 128x128 pixels was scanned in B2 channel (0.75 - 1.00 μm). In this task bright primitives were extracted because of bright clouds in the meteorological snaps. Primitives were classified into 10 classes. There was achieved a 96 % correct identification. It is very interesting only a little worse results were obtained with dark primitives without physical interpretation.

4. Conclusion

Each of the up to now existing texture recognition methods tends to emphesize tone or structure/3/. Presented hybrid texture recognition method treats both aspects equally. Tone is the basic information for primitive extraction,

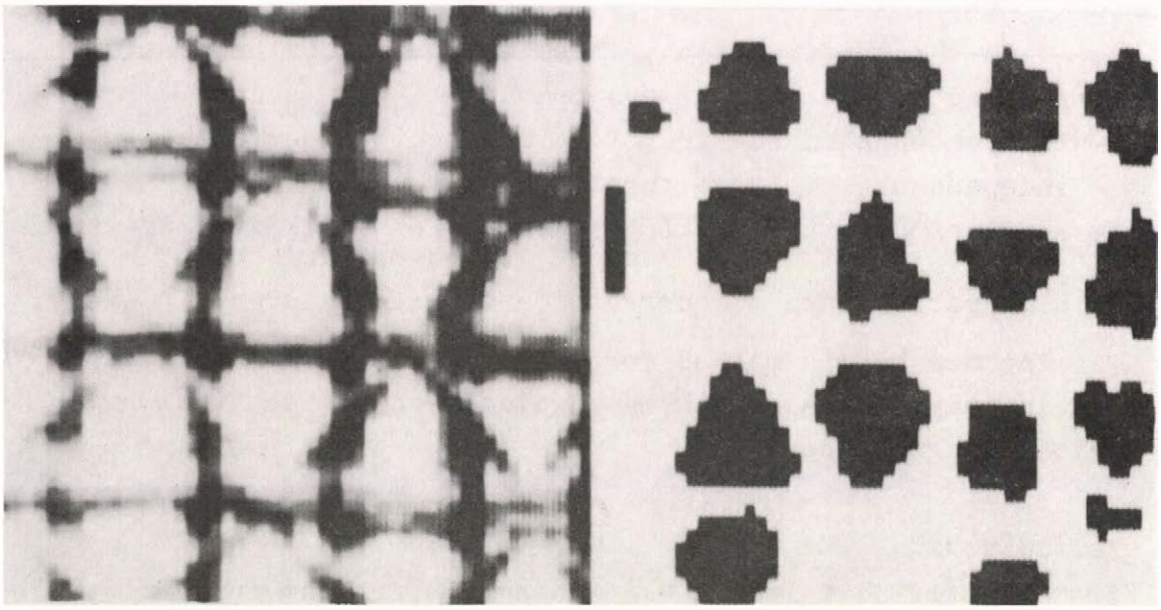


Fig.1 - Oriental straw cloth (a) Digitized Image
(b) Bright primitives extracted

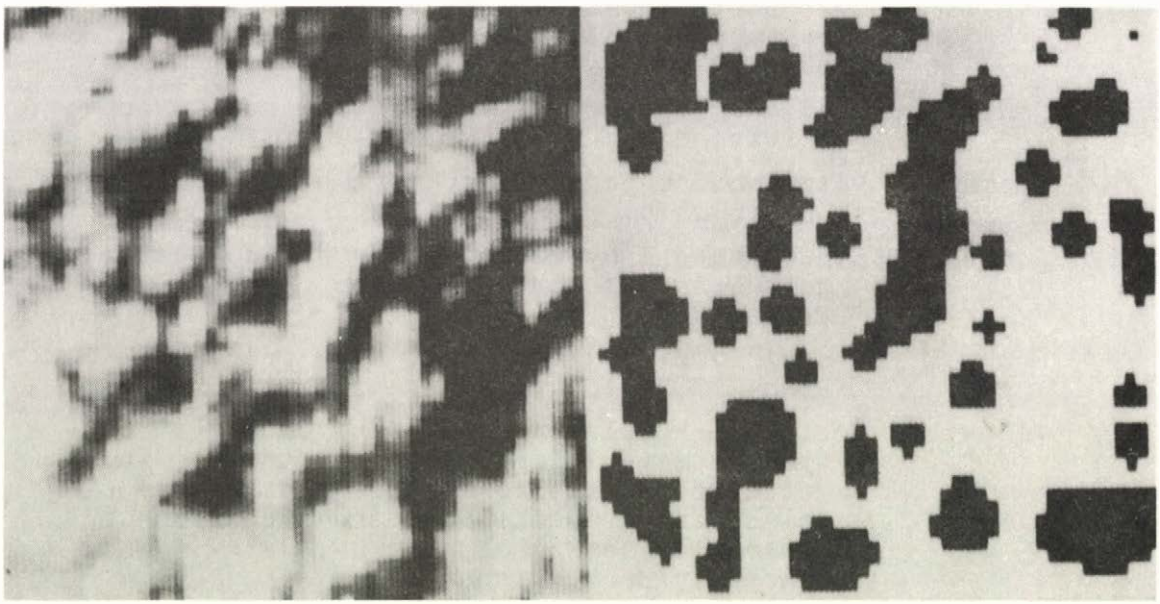


Fig.2 - Pressed cork (a) Digitized image
(b) Bright primitives extracted

structure is studied with spatial dependence among primitives.

With the verification , these supposed good properties of proposed hybrid method were proved -

- independence of the gray tone monotonic transformation
- independence of image turning
- short processing time (50 seconds for an 128x128 image and EC 1040 computer)
- high performance of correct identified textures .

The new hybrid method for texture recognition will be used in remote sensing, in meteorology and in technical and medical diagnostics.

5. Literature

/1/ Brodatz,P.: Textures - A Photographic Album for Artists and Designers, 1.ed., Dover New York 1966, 132 p.

/2/ Hong,T.H., Dyer,Ch.R., Rosenfeld,A.: Texture Primitive Extraction Using an Edge Based Approach, IEEE Trans. System Man Cybern., Vol. SMC-10, No 10, pp. 659-675, 1980.

/3/ Haralick,R.M.: Statistical and Structural Approaches to Texture, Proc. IEEE, Vol. 67, No 5, pp. 786-804, 1979.

/4/ Šonka,M.: Úvod do rozpoznávání texturních obrazů, Acta Polytechnica, No 9, pp. 85-94, SPN Praha 1983.

/5/ Šonka,M.: Příznakové metody popisu textur, Acta Polytechnica, No 5, pp.61-74, SPN Praha 1983.

/6/ Šonka,M.: Syntaktické a hybridní metody popisu textur, Acta Polytechnica, No 18, pp. 89-98, SPN Praha 1983.

/7/ Šonka,M.: Rozpoznávání textur, kand. dis. práce ČVUT FEL Praha 1983, 109 p.

/8/ Vlčák,L., Šonka,M.: Kompleksnyj algoritm obrabotki dannych iz meteorologičeskich sputnikov, in: Metody objektivnogo analiza i prognoza polej oblačnosti i osadkov na osnove dannych iz meteorologičeskich sputnikov, Ústav fyziky atmosféry ČSAV, Praha 1982.

• SOFTWARE PROJECTS •

АРХИТЕКТУРА И ЯЗЫКОВАЯ ОРГАНИЗАЦИЯ СИСТЕМЫ РАСПРЕДЕЛЁННОЙ ОБРАБОТКИ

Афанасьев С.В.
Ленинград, Менделеевская линия I, ЛНИВЦ АН СССР
Барилов А.А.
Ленинград, Менделеевская линия I, ЛНИВЦ АН СССР

I. Введение.

В настоящее время проблема распределённой обработки информации занимает центральное место в вычислительной технике. Всё большее значение приобретает поиск новых принципов организации распределённых вычислений и перспективных архитектурных решений, отвечающих требованиям эффективности и мобильности в условиях быстрого развития технологической базы.

Обычно, разработка распределённых систем диктуется текущими потребностями, сводящимися к одному: получение больших возможностей путём объединения существующих вычислительных средств в систему, и производится по методу "снизу вверх". При этом основные усилия направляются на решение проблемы стыковки различных программно-аппаратных средств (при помощи стандартных программно-аппаратных интерфейсов,протоколов связи и т.д.), организацию распределённых баз данных и, затем, разработку операционной системы,которая осуществляет управление ресурсами и процессами,как правило, централизованным образом

Основным недостатком при таком подходе является отсутствие единой методики разработки систем распределённой обработки, что приводит с одной стороны к уникальности разработки, с другой стороны к сложной адаптации на новые приложения.

Альтернативный подход, нашедший своё воплощение в создании машины с динамической архитектурой(МДА) I , заключается в разработке виртуальной системы, реализующей вычислительную среду, которая поддерживает функционирование сложных информационных структур на основе внутреннего языка высокого уровня,позволяющего достаточно эффективно описывать организацию и динамику изменения иерархических программных структур, являющихся объектами вычислительной среды.

Такой подход отвечает требованиям долгосрочности и универсальности, а также обладает другими достоинствами. Во-первых, представление задачи в структурном виде является наиболее естественной фор-

мой описания. Во-вторых, виртуальная система с точки зрения архитектуры однородна и обладает возможностью регулярного наращивания. В-третьих, высокий уровень организации виртуальной вычислительной среды и внутреннего языка упрощает отображение сложных задач распределённой обработки на уровне системы. В-четвёртых, реализация системы на конкретных вычислительных средствах сводится к реализации достаточно простого ядра (базовых механизмов интерпретации внутреннего языка).

2.Принципы организации системы распределённой обработки

Разработка системы сводится к последовательной реализации следующих основных этапов :

-выбор абстрактной модели вычислений,отвечающей требованиям эффективной распределённой обработки,

-организация вычислительной среды, обеспечивающей реализацию абстрактной модели и внешнего языка высокого уровня,

-определение принципов организации виртуальной системы, поддерживающей функционирование вычислительной среды и реализующей внутренний язык,

-реализация модели,заключающаяся в программно-аппаратной реализации виртуальной архитектуры, внутреннего языка и базовых механизмов на конкретных вычислительных средствах.

В качестве абстрактной модели вычислений выбраны Динамические Автоматные Сети (ДАС) 2 , ориентированные на представление задач в виде динамических распределённых автоматных структур. В отличие от других моделей вычислений, автоматы ДАС наделены возможностью изменять свои связи с другими автоматами в процессе функционирования. Любое вычисление, выполняемое на ДАС, сводится к изменению структуры исходной сети и структуры автоматов, входящих в её состав, т.е. трансформации в сеть, соответствующую результату. Сложность автоматных структур на уровне абстрактной модели не ограничена.

Для реализации абстрактной модели необходима вычислительная среда в которую погружаются объекты (автоматы ДАС). На уровне вычислительной среды ДАС представляет собой структуру, содержащую множество объектов с определённым над ними множеством отношений. Среда обеспечивает возможность порождения новых и изменения существующих структур путём порождения и уничтожения объектов и изменения отношений. Другими словами, среда обеспечивает "жизнедеятельность" объектов. Объекты среды и отношения между ними описываются на языке высокого уровня РЯД I . Описание конкретной вычислительной структуры соответствует программе.

При реализации модели вычислений возникает проблема определе-

ния виртуальной системы, реализующей вычислительную среду и ее объекты. На этом этапе определяются следующие основные понятия:

- архитектура виртуальной системы (или виртуальная архитектура),
- внутренний язык и
- базовые механизмы виртуальной системы.

Виртуальная система с точки зрения архитектуры представляет собой множество вычислителей - вычислительных модулей (вычислительное поле) и множество коммутаторов - коммутационных модулей (коммутационное поле) соединенных регулярным образом и допускающими возможность регулярного и неограниченного наращивания системы.

Если каждому объекту среды сопоставить соответствующее устройство, то в результате получится вычислительная система полностью реализующая описываемую вычислительную структуру. Однако, с точки зрения эффективности целесообразно множество объектов разбить на непересекающиеся подмножества, каждое из которых реализуется одним вычислителем.

Внутренний язык служит для описания объектов вычислительной среды и отношений между ними на уровне виртуальной системы.

Базовые механизмы обеспечивают функционирование виртуальной системы. Любой объект (автомат) может находиться в одном из следующих состояний: управение, вычисление, коммутация и хранения. Соответственно выделяются и аппаратно реализуются базовые механизмы управления, вычисления и коммутации.

Реализация системы заключается в програмно-аппаратной реализации архитектуры и базовых механизмов виртуальной системы, что и является тем ядром которое подвергается изменению при переносе на другие средства обработки информации.

3.Принцины внутренней языковой организации

Разработка внутреннего языка является наиболее конфликтным этапом в общем процессе разработки, так как требуется удовлетворить два, в общем-то, противоречивых требования: с одной стороны, наиболее полное отображение свойств модели вычислений, что предполагает язык высокого уровня, с другой стороны достаточно эффективная реализация, что предполагает простые механизмы интерпретации языка.

Концептуальную основу традиционного программирования образует понятие алгоритма - точного описания последовательнсти действий (команд). Множество команд разбивается на два основных класса: исполнительные или операторы (команды обработки, ввода-вывода) и управляющие (команды передачи управления). Программа на внутреннем языке является описанием алгоритма и представляет собой неявно упорядоченное множество команд на котором определены отношения двух базо-

вых типов : информационные и следования.

Информационные отношения обладают невысокой степенью динамизма и определяются адресными полями и порядком выполнения операторов. Они достаточно просто прослеживаются на линейных (не содержащих команд передачи управления) участках программ. Отношения следования более динамичны и определяются управляющими примитивами, которые образуют структуру управления, что и позволяет в ходе выполнения программы организовать вычислительный процесс.

В принципе, выполнение программы можно рассматривать как процесс, заключающийся в построении и функционирования некоторого информационного графа, содержащего только операторы связанные только информационными отношениями. Вычислительная система является устройством реализующим процесс построения и функционирования информационного графа на основе заданной программы. Именно информационный граф отражает естественную структуру задачи. Способ его построения и обеспечения функционирования представляет удобную основу для анализа и классификации различных вычислительных систем.

Одним из основных недостатков традиционной фон-неймановской модели вычислений является низкий уровень внутреннего языка, что влечёт за собой необходимость сложных преобразований при отображении задачи с внешнего на внутренний язык. Стуктура программы на уровне системы не отражает структуры задачи и черезвычайно усложнена дополнительными "внутренними" отношениями, что сильно затрудняет отладку и приводит к потере производительности.

Известные алгоритмические языки и мультипрограмные ОС являются неудобными для решения широкого класса задач распределённой обработки в силу сложности управления. Попытки распараллеливания вычислительного процесса как правило эффективны лишь для обработки элементарных регулярных фрагментов - несложных линейных участков программ (системы с опережающим просмотром), обработки векторов и матриц (векторные и матричные системы). Это в какой-то мере позволяет повысить уровень внутреннего языка за счёт введения операторов над простыми структурами данных, но в то же время значительно усложняется структура управления программ.

Ещё в большей степени это характерно для многопроцессорных систем, во внутренний язык которых вводятся дополнительные синхронизирующие примитивы. Организация вычислительного процесса требует предельно сложных структур управления, связанных с распределением физических ресурсов и синхронизации процессов. Кроме того, описание задач в терминах языков параллельного программирования является достаточно сложным.

Выходом является отказ от традиционной программной организации в пользу моделей с вырожденным управлением. К ним относятся сети Петри, потоковые модели и др. 3 4 . Программа для таких систем описывает операторную схему, реализующую задачу. Порядок выполнения операторов определяется условиями готовности. Условия готовности берут на себя всю организацию управления, так что можно не разделять средства управления на средства формирующие информационный граф и на средства синхронизации. Недостатком существующих моделей подобного типа является отсутствие элементов динамики. Схемы статичны и управление, как правило, осуществляется путём введения соответствующих элементов (селекторов, арбитров), выполняющих коммутацию потоков информации.

Этих недостатков лишена предлагаемая модель вычислений. Введение отношений и элементов динамизма во внутреннюю языковую организацию позволяет ещё более упростить структуру управления при решении сложных задач (в частности обрабтки сложных структур данных и введение элементов рекурсии) и упростить отображение инормационных структур на структуру системы. Таким образом, программа на внутреннем языке представляет собой граф, вершины которого - объекты и отношения, а дуги - связи между ними.

Любой объект вычислительной среды (автомат ДАС) представляется на уровне внутреннего языка соответствующим програмным объектом, содержащим полное его описание и состоящим из трёх основных частей: связи, функция и состояние. В силу необходимости эффективности описания произвольно сложных объектов требуется определить функционально полный набор базовых языковых элементов и правил их структурной композиции. Исходя из требований ко внутреннему языку можно сказать, что базовый набор должен быть достаточно полным для представления произвольно сложных программных конструкций внешнего языка, и, во-вторых, должен эффективно реализовываться, то есть языковые объекты базового набора должны быть достаточно простыми.

Основной единицей внутреннего языка является програмный элемент (ПЭ)- семантически законченная языковая конструкция. Программа, отображенная со внешнего на внутренний язык представляет собой структуру. В процессе решения исходная структура-программа интерпретируется и динамически реконфигурируется, в конечном итоге образуя структуру, соответствующую результату.

Множество базовых типов ПЭ можно разбить на следующие основные группы: отношения, операторы, значения, указатели

Во внутреннем языке должны присутствовать средства структурного описания. Для этого вводится понятия терминальности ПЭ и примитив-

ные стуктурные отношения. ПЭ могут быть нетерминальными и терминальными. Нетерминальные ПЭ служат для определения объектов, являющихся структурами. Терминальное ПЭ являются примитивными, т.е. не требующими структурной декомпозиции. Структурными отношениями являются отношения иерархии и следования. Семантика ПЭ (его тип и структурность) может определяться либо из описания (дискриптора), находящимся непосредственно в самом ПЭ, либо из отношений с другими ПЭ структуры.

ПЭ могут находиться в различного рода отношениях с другими ПЭ. Основными типами отношений являются структурные, функциональные и именные. Отношения могут быть структурами произвольной сложности.

ПЭ типа операторы небходимы для описания функции объекта и соответствуют операторам традиционных языков програмирования. На уровне внутреннего языка определяется множество арифметических и логических операций, а так же ряд операторов, результатом выполнения которых является преобразование структур. Терминальный ПЭ-оператор содержит имя (или указатель на имя) функции .

ПЭ типа значения соответствуют данным в традиционных языках программирования. В этой группе 3 типа: значения, значения с форматом и общие значения. Значения используются для представления линейных массивов данных, семантика которых определена на более высоком уровне программы, то есть, зависит от программы, обрабатывающей данный элемент. Значение представляется в общем случае сильно-связанным списком. Значение в какой-то мере соотвеоствует представлению данных в машинных языках традиционных ЭВМ. Значения с форматом служат для представления простых структур данных(в виде строк), в которых определена их семантика (тип, формат). Общее значение необходимо для представления сложных древовидных структур данных. Этот ПЭ содержит информацию о компонентах, структуре, связях и другую служебную информацию.

При отображении програмных структур на уровне системы приходится решать ряд проблем, связнных с представлением программ в памяти и организацией связей между ПЭ. Так как програмные структуры - динамические, то возникает проблема динамического распределения памяти в процессе вычисления. Это определяет максимальный размер ПЭ . С одной стороны, большой объём памяти, отводимой под ПЭ, приводит к нерациональному её использованию и порождает трудности упаковки ПЭ. С другой стороны, малый объём памяти не позволяет сформировать ПЭ, как семантически законченную языковую конструкцию и потребовал бы введения массы служебной информации для организации внутренних ссылок. Таким образом, внутренний язык использует страничную организа-

ции памяти. В процессе выполнения задачи при размещении и удалении ПЭ память выделяется и освобождается динамически страницами. Проблема "сборки мусора" решается путём определения двухсторонних связей между ПЭ, что так же повышает надёжность вычислений.

Распределённость структур в системе связана с проблемами организации связей между ПЭ, находящимися в различных частях системы. Например, в пределах одного вычислительного модуля ПЭ могут находиться в оперативной, сверхоперативной, постоянной или вторичной (дисковой) памяти. Эти проблемы решаются путём использования различного типа указателей. Указатели бывают простыми и сложными. Простой указатель занимает одно слово и может находиться в ПЭ различных типов. Сложные указатели занимают целую страницу памяти и составляют отдельную группу ПЭ. Необходимость введения отдельных типов ПЭ-указателей обусловлена, с одной стороны, необходимостью определения сложных структурных связей, а с другой - ограниченным размером ПЭ. С помощью указателей реализуется сложный аппарат отношений, широко использующийся во внутреннем языке. Тип указателя может оределяться явно из дескриптора или неявно, например, по номеру слова в котором находится указатель.

Связи между ПЭ, находящимися в различных вычислительных модулях определяются с помощью полных внешних указателей . По сути, это двойной указатель, содержащий указатель на внешний объект и указатель на ПЭ в этом объекте.

Перемещение программных структур между модулями и различного типа памятью требует достаточно сложной обработки указателей. С целью упрощения механизмов перемещения во внутреннем языке определяются специального тип ПЭ - частичные внешние указатели. Он содержит только указатель(и) на ПЭ во внешнем объекте. Сам же указатель на внешний объект находится в ПЭ более высокого уровня.

Литература

I. Торгашёв В.А. Управление вычислительным процессом и машина с динамической архитектурой. - В кн: Вычислительные системы и методы автоматизации исследований и управления. М.: Наука, 1982.

2. Пономарёв В.М., Плюсин В.У., Торгашёв В.А. Распределённые вычисления и машины с динамической архитектурой. Препринт №54, Ленинград: ЛНИВЦ АН СССР, 1982.

3. Adams D.A. A Computational Model With Data Flow Sequencing Computer Science Dept., School of Humanities and Sciences. Stanford University, Technical Report CS 117 (December 1968).

4. Peterson J.L. Petri Nets. ACM Surveys 9,3 (Sept. 1977).

СПЕЦИАЛИЗИРОВАННАЯ СИСТЕМА ПРОГРАММИРОВАНИЯ

А. Беляускас
Вильнюс, Университет, ВЦ

I. Введение

Одной из важнейших задач в радиоизмерительной технике в последнее время является задача автоматизации контрольно-поверочных работ радиоизмерительных приборов (РИП), что вызвано усложнением РИП, ростом трудоемкости и стоимости контроля.

В общем случае эта задача решается с помощью автоматизированных систем контроля (АСК).

Для АСК, использующих ЭВМ, весьма сложной и дорогостоящей задачей является разработка программного обеспечения. Для повышения производительности работы программиста автоматизированной системы разрабатываются специализированные средства автоматизации программирования, применение которых позволяет достигнуть:

- ускорения и упрощения подготовки и поддержки прикладного программного обеспечения;
- повышения живучести и надежности прикладной системы;
- удобного диалога между человеком и машиной;
- повышения производительности системы;
- простоты адаптации к меняющимся условиям применения;
- стандартизации и типизации решений при разработке прикладных систем;
- переносимости программного обеспечения.

Представляемая специализированная система программирования предназначена для автоматизации подготовки и поддержки прикладного программного обеспечения многопостовых АСК РИП с децентрализованным управлением.

2. Краткая характеристика многопостовой АСК

Многопостовая АСК с децентрализованным управлением построена по принципу двухуровневой иерархической локальной сети с одной центральной мини-ЭВМ и некоторого количества периферийных микро-ЭВМ - по одной в составе каждого рабочего поста (РП). Работа такой многомашинной АСК в общих чертах выглядит следующим образом: центральная ЭВМ по запросу периферийной ЭВМ передает через систему межмашинной связи необходимую программу проверки в

память ЭВМ РП АСК; РП автономно выполняет программу проверки и передает полученные результаты для анализа, хранения и распечатки в центральную ЭВМ. Центральная ЭВМ, кроме того, используется для подготовки и хранения программ проверки.

В настоящей работе в качестве центральной ЭВМ используется СМ-4, а в качестве периферийной ЭВМ - микро-ЭВМ "Электроника-60М". Рабочие посты АСК построены с использованием аппаратуры имеющей выход на стык стандартного приборного интерфейса (ГОСТ 26.003-80, IEEE-488, МЭК-625). Есть возможность использования в АСК устройств с интерфейсом отличным от приборного интерфейса.

Минимальная конфигурация центральной ЭВМ включает:

- 32 К слов оперативной памяти;
- устройство внешней памяти прямого доступа;
- два алфавитно-цифровых видеотерминала;
- устройство печати;
- устройство связи с периферийной ЭВМ.

Минимальная конфигурация периферийной ЭВМ включает устройство связи с программно-управляемой технологической аппаратурой и устройство связи с центральной ЭВМ.

Децентрализацией управляющих функций АСК - введением автономного процессора в состав каждого РП АСК - обеспечивается ряд преимуществ перед однопроцессорной системой с централизованным управлением. Наиболее очевидными преимуществами являются:

- повышение надежности (отказ одного из автономных постов не остановит работу всей системы в целом);
- возможность регулирования производительности всей системы путем изменения количества периферийных постов; большую систему можно вводить в эксплуатацию по частям постепенно наращивая конфигурацию;
- удобство разделения функций между отдельными частями системы;
- снижение себестоимости;
- ускорение времени реакции на внешние воздействия благодаря параллельной работе более одного процессора;
- удобство обслуживания, удобство адаптации к изменениям условий применения благодаря модульности построения.

Для полного проявления преимуществ децентрализованного управления необходимо соответствующее программное обеспечение,

при подготовке которого возникает необходимость решать сложную задачу взаимодействия между процессами, выполняемыми различными процессорами.

3. Система программирования многопостовой АСК

Представляемая специализированная система программирования предоставляет пользователю широкий круг возможностей, в том числе:

- составление программ на специализированном языке высокого уровня, ориентированном на задачи проверки параметров РИП и учитывающем аппаратурные особенности АСК;
- сборку программ проверки из отдельных модулей с обеспечением взаимодействия между ними, причем отдельные модули могут быть написаны как на входном языке настоящей системы, так и на некотором другом языке системы программирования РАФОС, например, фортран IV или макроассемблер;
- использование специального отладочного режима удобного как для отладки программ, так и для отладки аппаратуры АСК;
- независимость от аппаратурной конфигурации АСК;
- одновременную независимую работу постов АСК с различными программами проверки;
- замену программы проверки в каждом рабочем посту в любой момент сеанса работы АСК;
- выдачу оператору АСК информации о состоянии системы в процессе выполнения программы проверки объекта контроля;
- защиту программ проверки и аппаратуры от неправильных действий оператора, извещение оператора о сбоях в системе, их местонахождении и причинах.

Система программирования построена по модульному принципу. Ее компоненты работают под управлением операционной системы РАФОС. Модули системы программирования АСК написаны на языках макроассемблер и фортран IV.

Система программирования АСК состоит из следующих компонент:

- распределенной исполнительной системы;
- транслятора языка программирования АСК;
- средств ведения базы описаний АСК;
- библиотеки базовых подпрограмм;
- вспомогательных средств.

Программное обеспечение распределенной исполнительной сис-

темы состоит из двух составных частей: программного обеспечения центральной ЭВМ и программного обеспечения периферийной ЭВМ.

Компонента распределенной исполнительной системы в центральной ЭВМ выполняет следующие функции:

- инициирует и завершает сеанс работы АСК;
- управляет распределением общих ресурсов в сеансе работы;
- обслуживает запросы периферийных ЭВМ;
- поддерживает связь между ЭВМ;
- управляет базой данных АСК;
- наблюдает за состоянием всей системы и обрабатывает особые ситуации.

Центральная ЭВМ в течении сеанса работы многопостовой АСК может работать в двухпрограммном режиме: оперативный раздел предназначен для работы исполнительной системы АСК, а фоновый раздел одновременно может быть использован для выполнения других менее приоритетных задач, например, подготовки программного обеспечения. Однако удовлетворительная работа фонового раздела возможна лишь в случае не слишком интенсивных запросов на обслуживание от периферийных ЭВМ.

Отдельная копия компоненты исполнительной системы периферийной ЭВМ хранится в памяти ЭВМ каждого рабочего поста и выполняет следующие основные функции:

- управляет выполнением программ проверки, выполняемых в этом рабочем посту;
- обеспечивает связь с оператором, проводящим контроль;
- обеспечивает связь с центральной ЭВМ;
- обрабатывает особые ситуации.

Исполнительная система периферийной ЭВМ загружается из центральной ЭВМ начальным загрузчиком, который хранится в устройстве постоянной памяти периферийной ЭВМ.

Транслятор предназначен для преобразования программ на исходном языке системы программирования многопостовой АСК в программы на языке макроассемблер. Начальная полготовка текстов программ на исходном языке и дальнейшая ее обработка на макроассемблере производится средствами операционной системы РАФОС.

Транслятор языка АСК производит объектные программы двух типов: отладочного и рабочего. Отладочный вариант любой программы может быть выполнен автономно с возможностью оперативного изменения исходных данных. Рабочий вариант каждой программы пред-

назначен для использования при работе многопостовой АСК в комплексе в автоматическом режиме.

Входной язык системы программирования АСК является проблемно-ориентированным языком высокого уровня для записи алгоритмов управления процессами контроля РИП и обработки информации в АСК.

Целью разработки настоящего языка программирования явилось ускорение и упрощение процесса написания и отладки программ проверки РИП, обеспечение удобства документирования контроля.

Язык программирования АСК предоставляет пользователю средства для описания операций, осуществляемых при проведении проверки объекта контроля в рабочем посту АСК:

- подсоединение определенных входов и выходов приборов, входящих в состав АСК;
- установка режимов работы приборов;
- подача испытательных сигналов;
- чтение информации с измерительных приборов;
- обработка полученных результатов;
- управление выполнением программы;
- выдача требуемой информации в желаемой форме.

Язык программирования АСК легко осваивается пользователем-непрограммистом.

В реализованном варианте языка содержатся следующие операторы (в алфавитном порядке):

ВЫПОЛНИТЬ - выполнить подпрограмму, обозначенную указанным именем;

ЖДАТЬ - не начинать выполнения следующего оператора до выполнения определенного условия;

ЗАДЕРЖАТЬ - задержать выполнение следующего по порядку оператора до истечения определенного интервала времени;

ЗАПУСТИТЬ - запустить определенный прибор для выполнения измерения;

ИЗМЕРИТЬ - выполнить операцию измерения (равноценно ЗАПУСТИТЬ плюс ЧИТАТЬ);

ИНДИЦИРОВАТЬ - представить информацию оператору АСК с помощью световых табло и цифровых индикаторов специализированного пульта управления рабочего поста АСК;

НАЧАТЬ - первый оператор программной единицы;

ОБОЗНАЧИТЬ - вычислить результат арифметического выражения и обозначить его именем (т.е. присвоить переменной значение);

ОИ - очистить интерфейс;

ОКОНЧИТЬ - последний оператор программной единицы;

ОСТАНОВИТЬ - приостановить выполнение программы для выполнения оператором ручного управления;

ПЕРЕЙТИ - изменить порядок выполнения операторов по какому-либо условию или без условия;

ПЕЧАТАТЬ - печатать алфавитно-цифровую информацию с помощью печатающего устройства АСК или вывести на экран дисплея;

ПОВТОРИТЬ - выполнить повторно определенное количество раз один или несколько предыдущих операторов программы;

СБРОСИТЬ - установить некоторые указанные или все приборы рабочего поста АСК в исходное состояние;

СОЕДИНИТЬ - соединить определенные входы и выходы коммутатора либо определенные входы и выходы каких-либо устройств, входящих в состав рабочего поста АСК;

УСТАНОВИТЬ - установить режим работы определенного устройства из состава рабочего поста АСК;

ЧИТАТЬ - считать информацию определенного устройства и хранить это значение в заданных единицах в оперативной памяти.

Информационная часть системы - база описаний - состоит из:

- описаний приборной части АСК;
- описаний конфигурации АСК;
- описаний размерностей физических величин АСК.

В описаниях приборной части АСК содержатся данные о всех приборах, которые могут быть использованы в АСК. Описание прибора состоит из наименования прибора, наименований выполняемых им функций, наименований значений каждой из функций и управляющей информации, при помощи которой кодируются эти функции и значения функций.

В исходных текстах программ на языке программирования АСК используемые приборы и ими выполняемые функции и значения функций указываются в виде символьных наименований, например, функция - частота, значение функции - 100 герц и т.п.

Описание конфигурации АСК состоит из таблицы адресов приборов, входящих в состав конкретной АСК.

Описания размерностей физических величин содержат символьные наименования размерностей, используемых в программах, кодировку размерностей для отдельных приборов и масштабные коэффициенты для групп соответствующих размерностей.

В тексте программы на языке АСК указывается символьное наименование размерности физической величины, в единицах которой необходимо хранить результат выполненного измерения независимо от того, в каких единицах из этой группы измерительный прибор передаст программе результат.

Использование базы описаний АСК обеспечивает простую адаптацию системы программирования АСК к условиям применения и независимость системы от любой частной АСК. Для применения настоящей системы программирования в конкретной АСК достаточно поместить ее характеристики в базу описаний.

Средства ведения базы описаний АСК состоят из программ, предназначенных для:

- создания файлов базы описаний АСК;
- помещения новых описаний в базу;
- удаления описаний из базы;
- коррекции существующих описаний;
- копирования данных базы описаний;
- просмотра и распечатки содержимого базы описаний.

Описания используются транслятором для формирования управляющей информации в прикладных программах, а также исполнительной системой при формировании сообщений о причинах сбоев во время работы АСК.

Библиотека базовых подпрограмм включает подпрограммы транслятора языка системы программирования и стандартные рабочие подпрограммы. В их число входят:

- подпрограммы управления устройствами АСК;
- подпрограммы управления вводом/выводом;
- подпрограммы преобразования информации;
- подпрограммы математической обработки данных;
- подпрограммы взаимодействия с базой описаний АСК;
- подпрограммы связи с операционной системой;
- подпрограммы обработки прерываний;
- подпрограммы обработки особых ситуаций и т.д.

К вспомогательным средствам системы программирования АСК относятся контрольные примеры, программы самоконтроля АСК, программы, выполняющие некоторые сервисные функции.

ONE STEP MULTIDERIVATIVES METHODS FOR SOLVING DIFFERENTIAL EQUATIONS OF THE FIRST ORDER

Jozef Dančo

Department of Numerical and Optimalization Methods
Faculty of Mathematics and Physics
Komenský University
842 15 Bratislava
Czechoslovakia

The subject mattrer of this paper is usualy formulated as a special numerical problem and concerns the problems of obtaining coefficients for this special class of numerical method for solving ordinary differential equations. These coefficients were computed by computer as wellas the particular methods which were programmed for computer. So, computer finds not only the solution of ordinary differential equations but also the coefficients of the method. This seems to me be an advantage for numerical solution of the above mentioned problem and for use of computer as well.

1. Introduction

One step multiderivatives methods are a means for numerical solution of a first order differential equation

$$y' = f(x,y)$$
$$y(x_0) = y_0 \qquad (1)$$

at the point x_n. In this paper we suppose, that function $f(x,y)$ satisfy the conditions of the existence and unit theorem of the solution of differential equation. This theorem has been proved in [1] . Numerical methods for solving the above mentioned problem are using also the derivatives of the function $f(x,y)$. The problem is solved in the interval $\langle a,b\rangle$, where $x_0=a$. Interval $\langle a,b\rangle$ is devided into parts separet d by points $x_1, x_2, \ldots x_{n-1}$ and $x_n=b$. A step we define as $h_n=x_{n+1}-x_n$ and in this paper we supose $h_n=h$.

The basic idea of these methods is very simple. To find the next value of solution, the values of function f(x,y) and derivatives of this function in points (x_n,y_n) , (x_{n+1},y_{n+1}) has to be used. This can be written as follous:

$$y_{n+1} = y_n + \sum_{i=0}^{k} h^{i+1}(a_i f_n^{(i)} + b_i f_{n+1}^{(i)}) \qquad (2)$$

where a_i,b_i for i=0,1, ... k are unknown coefficients of the method

h is a step, $h=x_{n+1}-x_n$

and

$$f_n^{(i)} = \frac{d^i}{dx^i} f(x,y(x))\Big|_{x=x_n} \qquad (3)$$

As can be seen the methods defines by (2) are implicit.

2. The general form for finding the unknown coefficients of the methods.

In this part we shall concentrate on the general form for counting the unknown coefficients a_i,b_i i=0,1, ... k for any value of k. If we expand y_{n+1}, $f_n^{(i)}$, $f_{n+1}^{(i)}$ to Taylor's serie and compare the coefficients of the derivatives of f(x,y) and povers of h we get the linear equations system of the following form:

$$Q\,a = y \qquad (4)$$

where a is the unknown coefficients vector

$$a = (a_0,a_1, \ldots a_k,b_0,b_1, \ldots b_k)^T$$

y is a vector, the right hand side of the equation and can be written in the form

$$y_i = \frac{1}{i!} \qquad \text{for } i=1,2, \ldots 2k+2$$

Q denotes the matrix of dimension 2k+2 and has the form:

$$Q_{i,j} = \delta_{ij} \qquad \text{for } i=1,2, \ldots 2k+2 \quad j=1,2, \ldots k+1$$

$$Q_{i,j} = \frac{1}{(i-j+k+1)!} \qquad \text{for } i \geq j-k-1,\ i \neq 1,2, \ldots 2k+2,\ j=k+2, \ldots 2k+2$$

$$Q_{i,j} = 0 \qquad \text{for } i < j-k-1$$

Solution of linear equation system satisfy the condition

$$a_i = (-1)^i b_i \quad \text{for } i=0,1, \ldots k \qquad (5)$$

This fact is very important, since the coefficients computation requires only the solution of a smaller linear equation system of the form:

$$P\,b = z \qquad (6)$$

where b is the vector of unknowns

$$b = (b_0, b_1, \ldots b_k)^T$$

vector z can be written as follows:

$$z_i = y_{i+k+1} \qquad \text{for } i=1,2, \ldots k+1$$

and matrix P is a reduced form matrix from matrix Q as

$$P_{i,j} = Q_{i+k+1,j+k+1} \quad \text{for } i,j = 1,2, \ldots k+1$$

That means that for finding the unknown coefficients the right down quadrant of the matrix Q is used. By solving the linear equations system (6) the unknown coefficients are obtained. Special computer program in FORTRAN has been written for this part of solution. This program solves the system of linear equations in fraction term. The algorithm used in this program is a Gaussian elimination method in fraction term.

3. Local truncation error and the approximation order.

Suppose we have a general form of the methods:

$$y_n = h\, G_f(x_n, y_n, y_{n+1}, h) \tag{7}$$

than the local truncation error t_{n+1} in the point x_{n+1} is given by:

$$t_{n+1} = y(x_n) - h\, G_f(x_n, y(x_n), y(x_{n+1}), h)$$

where $y(x_n)$ is exact solution of differential equation (1). In our case for methods definited by (2) the local truncation error is given by:

$$t_{n+1} = c_k \frac{h^{2k+3}}{(k+2)!}\, y^{(2k+3)}(x_n) + O(h^{2k+4}) \tag{8}$$

Where the constant term c_k is equal to:

$$c_k = \frac{1}{(k+3)^{[k+1]}} - \sum_{i=0}^{k} \frac{b_{k-i}}{(k+3)^{[i]}} \tag{9}$$

and $n^{[k]}$ is the k-th raising factorial from n

b_i are unknown coefficients of the method

Approximation order can be as simply as possible defined as the greatest power of polynom $G(x)=g_0+g_1x+ \dots g_nx^n$, which exactly sutisfy the method. From expression (8) we can see that the method described in this paper is of order $2k+2^{nd}$

Suppose we have a more general scheme that than of (2). this form is as follows:

$$\sum_{j=-1}^{q} \sum_{i=-1}^{k} h^{i+1}\, d_{i,j}\, f_{n-j}^{(i)} \tag{10}$$

and

$$f_{n-j}^{(-1)} = y_{n-j}$$

This scheme is for multiderivative multistep methods. Former

scheme (2) is only a special case of this scheme where q=0. In general, one can say that the methods described by the scheme (10) are not stable. Daniel and Moore prooved that the multiderivative multistep methods are stable only if the order is $2k+2$, also that the methods described by scheme (2) are A-stable (See Dalquist, 1963).

3. Particular methods

The coefficients from k=0 to k=8 are tabulated in the Table 1. Here are written only the a_i coefficients for i=0,1, ... k and the coefficients b_i for i=0,1, ... k, multiplied by -1 or 1 , can be computed as we do it in (5). Table 1 consists olso the constant term c_k for local truncation error, but the exact local truncation error is given by (9).

In the case when k=0, it is a well-known Euler's method. When k=1 we got the method described in [2] , but there is not the procedure for finding the coefficients.

4. Numerical results

The methods described in this paper are acceptable for solving the stiff differential equations as a corrector of one step method. As for predictor the clasic Runge-Kutta method of 4^{th} order could be used. The method can be written in the form:

0				
$\frac{1}{2}$	$\frac{1}{2}$			
$\frac{1}{2}$	0	$\frac{1}{2}$		
1	0	0	1	
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

By the clasic Runge-Kutta method we find a first value of y_{n+1}. Then using the method described in this paper we

k	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	c_k
0	$\frac{1}{2}$									$\frac{1}{6}$
1	$\frac{1}{2}$	$\frac{1}{12}$								$\frac{0}{120}$
2	$\frac{1}{2}$	$\frac{1}{10}$	$\frac{1}{120}$							$\frac{1}{4200}$
3	$\frac{1}{2}$	$\frac{3}{28}$	$\frac{1}{84}$	$\frac{1}{1680}$						$\frac{1}{211680}$
4	$\frac{1}{2}$	$\frac{1}{9}$	$\frac{1}{72}$	$\frac{1}{1008}$	$\frac{1}{30240}$					$\frac{1}{13970880}$
5	$\frac{1}{2}$	$\frac{5}{44}$	$\frac{1}{66}$	$\frac{1}{792}$	$\frac{1}{15840}$	$\frac{1}{665280}$				$\frac{1}{1141620480}$
6	$\frac{1}{2}$	$\frac{3}{26}$	$\frac{5}{312}$	$\frac{5}{3432}$	$\frac{1}{11440}$	$\frac{1}{308880}$	$\frac{1}{17297280}$			$\frac{1}{111307996800}$
7	$\frac{1}{2}$	$\frac{7}{60}$	$\frac{1}{60}$	$\frac{1}{624}$	$\frac{1}{9360}$	$\frac{1}{205920}$	$\frac{1}{7207200}$	$\frac{1}{518918400}$		$\frac{1}{12614906304000}$
8	$\frac{1}{2}$	$\frac{2}{17}$	$\frac{7}{408}$	$\frac{7}{4080}$	$\frac{1}{8160}$	$\frac{1}{159120}$	$\frac{1}{4455360}$	$\frac{1}{196035840}$	$\frac{1}{17643225600}$	$\frac{1}{1629845894476800}$

Table 1.

correct this first value of y_{n+1}. The general formula of those methods can be written as:

$$P\ (\ E^{k+1}\ C\ )^m$$

where P is the predictor (in our case the clasic Runge-Kutta method of 4-th order)
E is the evaluation of right hand side of differential equation and the derivatives (counting k+1 times)
C is the corrector of the method
m is the mumber of repeating use of the corrector

A kind of disadvantage of these methods is that we have to find the derivative of the right hand side of differential equation. The first two derivatives are as follows:

$$\frac{d}{dx} f(x,y) = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y} f$$

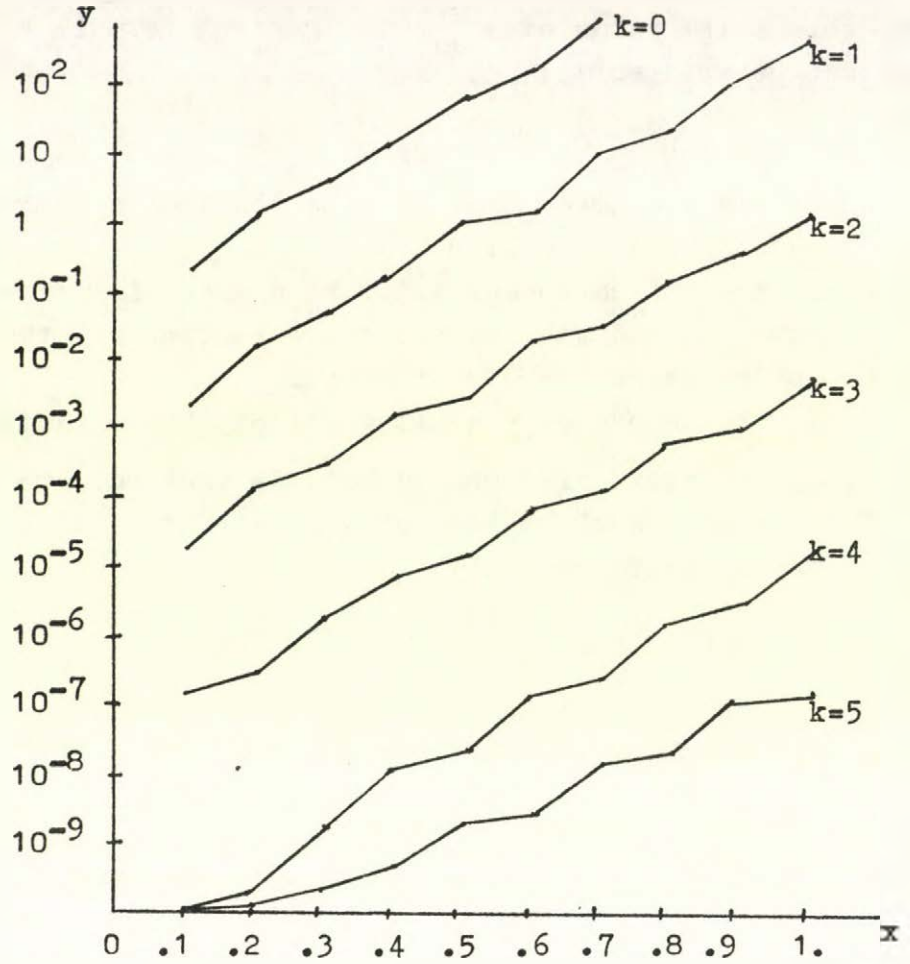
$$\frac{d^2}{dx^2} f(x,y) = \frac{\partial^2 f}{\partial x^2} + 2f \frac{\partial^2 f}{\partial x \partial y} + f^2 \frac{\partial^2 f}{\partial y^2} + \frac{\partial f}{\partial y}\left(\frac{\partial f}{\partial x} + f \frac{\partial f}{\partial y} \right)$$

Supose we have a differential equation of the form

$$y' = 10y$$
$$y(0) = 1.$$

with exact solution $y(x)=e^{10x}$. Function $y(x)$ is steeply increasing with growing x. One step explicit methods are not able to take into account this sharp increase. If the methods described in this paper are used for solution of differential equations with step $h = 0.1$ in the interval $\langle 0,1 \rangle$, there is a substantial difference in accuracy.The next graph demonstrate the error term (axis y) in absolute value in relation to an increasing x (axis x) and could be used to prove our methods for this kind of initial value problem.

Methods proposed in this paper are a bit complicated because of derivative, but by this algorithm more accurate results could be found for stiff differential equations.



For solving the coefficients of these methods as well as methods for solving differential equations computer programme is available in FORTRAN. The only limitation of this programm was a maximal word bit-lenght, but when using computer with apropriate word lenght, further extention of these methods is available and accuracy of differential equations solution considerably increases.

Bibliography.

[1] Kluvánek I., Mišík L., Švec M.: Matematika I and II, SVTL Praha 1961

[2] Hall G., Watt J.M.: Modern numerical Methods of Ordinary Differential Equations, Oxford 1978

[3] Wanner G,: On the integration of stiff differential equations, ISNM 37 Birkhäuser Verlag, 1977

ГРАФИЧЕСКИЙ ПАКЕТ ГРАФОР: КОНЦЕПЦИИ И ВОЗМОЖНОСТИ

Владимир А. Галактионов
Институт прикладной математики им.М.В.Келдыша АН СССР
I25047 СССР, Москва, Миусская пл., 4.

I. Введение.

Машинная графика по достоинству оценена учеными, конструкторами, технологами как мощное средство взаимодействия между человеком и машиной. Применение графических устройств существенно расширяет потенциальные возможности вычислительных машин, позволяет работать с такими объемами информации, освоить которые старыми методами очень трудно, а порой и невозможно. Например, в задаче трехмерного численного моделирования процесса возникновения и эволюции крупномасштабной структуры Вселенной исследуется эволюция 32^3 частиц под действием собственных сил тяготения /2/. Удобным способом представления результатов вычислений является фильм, каждый кадр которого показывает распределение плотности частиц в некоторый момент времени.

Можно выделить еще один широкий класс задач, решать которые, не имея развитых средств машинной графики, практически невозможно. Это прежде всего задачи автоматизации проектирования и производства, связанные с обработкой геометрической информации, описанием геометрий сложных форм и т.п. На решение этих задач (в том числе и на разработку соответствующих графических средств) направлены усилия многих научных и производственных коллективов. Достаточно сказать, что компания Форд, одна из ведущих автомобилестроительных фирм мира, к I987 году планирует производить 90% всех проектно-конструторских работ при непосредственном использовании средств машинной графики /5/.

Одной из развитых систем машинной графики, предоставляющих широкие "изобразительные" возможности, является графический пакет

Графор, разработанный в ИПМ им.М.В.Келдыша АН СССР /I/.

2. Структура и состав комплекса Графор.

По своей сути Графор является библиотекой графических подпрограмм и функций (их более 400), написанных преимущественно на Фортране. Тем самым решаются многие проблемы, связанные с переносом пакета на другие машины и операционные системы. Зависимость от графических устройств в Графоре локализована так, что при подключении нового устройства не требуется сколь-нибудь значительных переделок. Фортранная природа Графора, выбор узкого базиса пакета, рациональное деление на уровни способствовали переносу Графора на многие ЭВМ (БЭСМ-6, ЕС ЭВМ, СМ-4, М-6000, PDP-11, NORD, ECLIPSE и др.) и адаптации на различные графопостроители и дисплеи (ЕС 7051÷÷7054, ЕС 7064, CALCOMP, BENSON, VU-2000(SINTRA), TEKTRONIX и др.). В результате Графор получил очень широкое распространение и стал практически самым популярным графическим пакетом в стране. Рассмотрим кратко основные возможности, предоставляемые пользователю этим пакетом.

2.I. Средства построения изображения.

Базисный (нижний) уровень пакета составляют программы построения "графических примитивов": отрезков прямых линий, дуг окружностей и эллипсов, строк текста, маркеров и др. На их основе строятся более сложные изображения, а также другие уровни пакета.

На практике часто приходится сталкиваться с изображениями, которые содержат однотипные элементы (подкартинки), отличающиеся друг от друга только местоположением, ориентацией, размером. В таких случаях удобно воспользоваться программами, реализующими линейные (аффинные) преобразования. Каждое такое преобразование описывается матрицей. Результирующее преобразование вычисляется путем умножения (возможно с использованием скобок в случае изображений, имеющих иерархическую структуру, где с каждым уровнем может быть связано свое преобразование) матриц основных преобразований поворота, переноса, масштабирования.

Используя программы экранирования, можно запретить рисование внутри (или вне) любой области (в том числе и многосвязной), имеющей форму многоугольника. Возможно одновременное задание нескольких экранов (до I6), тогда на изображение будет действовать результирующий экран, являющийся объединением отдельных экранов.

В Графоре имеются средства, позволяющие формировать так назы-

ваемый след пера, т.е. с некоторой дискретностью отбирать точки, принадлежащие траектории пера (луча). Рисование при этом может и не производиться, т.е. перемещение пишущего инструмента может быть как реальным, так и "воображаемым". Зафисированную в памяти картинку можно в дальнейшем рисовать, не повторяя вычислений. Она может быть также подвергнута произвольному, в том числе и нелинейному, преобразованию. След пера оказывается полезным еще и для формирования границ участков, которые впоследствии будут заэкранированы или заштрихованы.

2.2. Геометрические вычисления.

В составе Графора имеются программы, позволяющие производить многие геометрические вычисления, с которыми приходится сталкиваться при автоматическом формировании рисунков и чертежей, а также при подготовке программ для станков с числовым программным управлением. К таким задачам относятся, например, проведение прямых, составляющих друг с другом заданный угол или касающихся заданных окружностей, построение концентрических окружностей, окружностей, касающихся друг друга или заданной прямой, построение точек пересечения прямых и окружностей, выполнение разного рода сопряжений, скруглений и т.п.

2.3. Построение графиков функций. Аппроксимация и сглаживание.

В Графоре предусмотрены довольно развитые средства, позволяющие изображать различные функциональные зависимости. В простейшем случае – это построение кривых, заданных табличными значениями, и проведение осей координат. Отображение математического пространства пользователя на физическое пространство листа бумаги, а также разметка осей, выполняются автоматически. График может быть построен в декартовой и в полярной системах координат, причем в первой из них могут использоваться логарифмические шкалы по одной или обеим осям.

При выводе информации на графические устройства, а также при графическом вводе, часто возникает необходимость в ее предварительной обработке (восполнении, приближении, сглаживании). В результате улучшается как математическое, так и эстетическое представление информации. В рамках Графора реализовано несколько методов сплайн-интерполяции, а также различные способы сглаживания функций (построенные на основе метода наименьших квадратов, рядов Фурье, ортогональных многочленов, линейного фильтра и др.).

В системах автоматизации проектирования и производства мето-

ды аппроксимации применяются прежде всего в целях конструирования кривых и поверхностей, когда практически утрачивает смысл такой математический критерий как точность аппроксимации, и главную роль начинают играть такие критерии как внешний вид и гладкость кривой, отсутствие осцилляций и т.п. В этих случаях чрезвычайно удобными и эффективными оказываются методы аппроксимации Безье и В-сплайнов (произвольной степени).

2.4. Изображение функций двух переменных.

При решении многих реальных практических задач приходится иметь дело с большими массивами чисел, которые являются значениями функций двух переменных. Для их изображения чаще всего применяются два способа: карты изолиний и проекции поверхностей. Для каждого из этих способов в Графоре имеется несколько отличающихся друг от друга реализаций.

Построение линий уровня может выполняться с использованием как линейной, так и бикубической интерполяции. Последний способ целесообразно применять там, где сетка значений "редкая". Процесс построения каждой изолинии разбивается по-существу на три этапа: поиск изолинии (определение начальной точки на изолинии), отслеживание изолинии (нахождение на ней последовательности точек) и, наконец, оформление изолинии (вписывание значений в разрывы изолиний, нумерация, расстановка берг-штрихов).

Графор позволяет строить произвольные аксонометрические и перспективные проекции поверхностей. Для достижения большей наглядности те части проекции объекта, которые "невидимы" наблюдателю, стираются. Применяемые алгоритмы удаления невидимых линий основываются на упорядочении "элементов" поверхности, будь то сечения поверхности параллельными плоскостями или ячейки криволинейной сетки.

Если сетка, на которой задана функция, оказывается слишком редкой, то функцию всегда можно довосполнить в узлах новой, более частой сетки, путем аппроксимации кусочно-многочленными функциями гладкости 0,1,2,3 или бикубическими многочленами. Если же функция задана в узлах, не образующих регулярную прямоугольную сетку, то поверхность аппроксимируется с помощью треугольной сетки, формируемой по заданной границе области определения функции и узлам внутри нее с помощью триангуляции.

Для изображения объектов более общих, чем однозначные функции, может использоваться метод "ореола", когда в дальние (невидимые)

отрезки вносятся разрывы, вызванные воображаемыми непрозрачными ореолами вокруг более близких к наблюдателю отрезков. Этот метод позволяет работать с неструктурированной графической информацией, т.е. он предназначен для изображения объектов, представленных в виде неупорядоченной совокупности отрезков прямых.

3. Графор и унификация машинной графики.

В настоящее время в области машинной графики чрезвычайно актуальной является задача разработки унифицированного графического обеспечения. Стандартизация способствует созданию мобильных систем машинной графики, а, следовательно, и мобильных прикладных программ, относительно легко переносимых с одной аппаратной конфигурации на другую. На основе анализа общих черт ряда графических пакетов были разработаны унифицированные графические системы CORE /6/ и GKS /4/.

В последние годы в ИПМ АН СССР на базе предложений, содержащихся в проекте CORE, был создан и состыкован с Графором Базовый графический пакет, позволяющий организовать графический диалог с программами, написанными на Фортране /3/. Таким образом, Графор оказался дополненным интерактивными средствами, т.е. появилась возможность работать с графическим дисплеем в режиме диалога. Однако в связи с принятием системы GKS в качестве международного стандарта ISO весьма актуальными становятся также задачи реализации этой системы и сопряжения с ней Графора.

ЛИТЕРАТУРА

1. Баяковский Ю.М., Галактионов В.А., Михайлова Т.Н. ГРАФОР: комплекс графических программ на Фортране. Части I и 2. - Москва: ИПМ АН СССР, 1983. - 182с. и 176с.

2. Клыпин А.А., Шандарин С.Ф. Трехмерная численная модель образования крупномасштабной структуры Вселенной. - Москва: Препринт ИПМ АН СССР, 1982, №136. - 27с.

3. Лацис А.О., Романенко С.А. ГРАФОР-БГП: Интерактивная версия системы Графор. - Москва: Препринт ИПМ АН СССР, 1983, №89. - 26с.

4. Enderle G., Kansy K., Pfaff G. Computer Graphics Programming. GKS - The Graphics Standard. - Springer-Verlag, Berlin, 1984. -542p.

5. Parrott R.W. The Ford Graphics System: Local Processing, Central Control. - AUTOFACT 4 Conf. Proc., Philadelphia, 1982. North-Holland, Amsterdam, 1982. - p.4.56-4.60.

6. Status report of the GSPC of ACM/SIGGRAPH.-Comput.Graph., 1979, vol.13,N°3. - 179p.

ПРОГРАММНЫЙ КОМПЛЕКС НЕПТУН
В ПАКЕТЕ ПРИКЛАДНЫХ ПРОГРАММ САФРА.

Герасимов Б.П.

Институт прикладной математики им.М.В.Келдыша
Академии наук СССР, Москва.

I. Математическая постановка задачи.

Конвективное движение несжимаемой вязкой жидкости описывается системой уравнений Навье–Стокса в приближении Буссинеска:

$$(I)\quad \frac{\partial \vec{V}}{\partial t} + (\vec{V}\nabla)\vec{V} = -\frac{1}{\rho_0}\nabla p + \nu \Delta \vec{V} - \vec{G}T$$

$$\frac{\partial T}{\partial t} + (\vec{V}\nabla)T = \varkappa \Delta T + Q$$

$$\nabla \vec{V} = 0$$

Здесь $\vec{V} = (u, v)$ – скорость течения, T – отклонение темпепатуры от среднего постоянного значения $\overline{T}$, p – отклонение давления от гидродинамического $\overline{p}$, соответствующего постоянной температуре $\overline{T}$, Q – мощность тепловых источников, ν – коэффициент кинематической вязкости, $\varkappa$ – коэффициент температуропроводности, $\vec{G} = \beta\vec{g}$, β – коэффициент температурного расширения вещества, $\vec{g}$ – ускорение свободного падения. Для системы уравнений (I) в переменных функция тока ψ и вихрь ω рассматривается внутренняя краевая задача в прямоугольнике. В зависимости от конкретных граничных условий рассматриваемая область может быть либо замкнутой полостью, либо проточной. При обезразмеривании уравнения (I) сохраняют свой вид, однако смысл входящих туда параметров $\nu, \varkappa, G, Q$, а также масштабы основных величин различны для замкнутой и проточной полостей.

Решение системы (I) должно удовлетворяться следующим граничным условиям: на стенках выполняются условия прилипания и непротекания (для пористой стенки задается скорость просачивания) и задано распределение температур или тепловых потоков; на входном сечении трубы параметры втекающего потока известны (например, могут быть заданы распределение температуры и функции тока); на выходном сече-

нии распределение скорости, температуры и других параметров обычно неизвестно, поэтому предполагается, что линии тока параллельны, а значения всех переменных считаются постоянными вдоль линий тока; граничные условия для вихря ω на твердых стенках опре - деляются в конечно-разностном виде. В задачах с симметрией рассматривается половина области.

2. Разностная схема.

Для численного решения системы (I) используются разностные методы и вводится неравномерная сетка по пространству и равномерная по времени. Все параметры считаются определенными в узлах сетки.

Уравнения для вихря ω и температуры T аппроксимируются на сетке консервативной продольно-поперечной схемой [3-5] .

Для решения уравнения для функции тока ψ используются два метода:

а) итерационный метод установления с параметрами, оптимизированными по Жордану;

б) метод разделения переменных с применением быстрого преобразования Фурье.

Получаемая система разностных уравнений решается последовательными прогонками. Сначала по известным значениям T, ω и ψ в результате прогонок находится значение температуры на новом временном слое $\hat{T}$. Подставляя найденное $\hat{T}$ в уравнение для ω , получаем новое $\hat{\omega}$ во внутренних точках области. Затем, решая уравнение для функции тока, определяем $\hat{\psi}$. Далее вычисляются новые значения вихря на границе. На этом заканчивается расчет одного шага по времени. Описанный алгоритм зафиксирован в схеме счета программы и, в частности, в программе управления расчетом временного шага ⟨2.I0⟩ *STEPON*.

Расчет проводится до установления температуры или вихря. В программе проверка на окончание расчета проводится по одному из усло - вий; какое из них выбрать, определяет пользователь, исходя из физических параметров задачи. Шаг по времени выбирается автоматически.

3. Структурное описание программного комплекса.

Программный комплекс *NEPTUN* написан на языке ФОРТРАН-ДУБНА в рамках системы *OLYMPUS* [I]. Стандартизация структуры, имен и идентификаторов в этой системе обеспечивает простоту освоения программ новыми пользователями и упрощает внесение изменений. Являясь частью функционального наполнения пакета прикладных программ

САФРА (Система Автоматизации Физических РАсчетов) [2], программный комплекс NEPTUN существенно использует возможность автоматизации сборки модулей.

Все переменные, необходимые для работы программы, хранятся в оперативной памяти машины и группируются в блоки общей памяти. Вывод информации на внешний носитель ведется только с целью записи результатов расчета.

Каждой программной единице присваивается номер, указывающий класс, которому она принадлежит, а также ее место в данном классе. Например, запись ⟨I.5⟩ указывает на пятую подпрограмму из первого класса (класса пролога). Аналогичное соглашение имеется и для общих блоков. Опишем некоторые подпрограммы комплекса.

Класс 0. Управление расчетом.

Этот класс включает основную программную единицу MAIN и четыре подпрограммы: BASIC, MODIFY, COTROL, EXPERT. Назначение этих программ определено системой OLYMPUS [I]. В подпрограмме MODIFY реализован ввод основных переменных системы OLYMPUS.

Класс I. Пролог.

В этот класс входят подпрограммы, имена и функциональное назначение которых совпадают с именами и назначениями, описанными в OLYMPUS [I,8].

Класс 2. Вычисления.

⟨2.I0⟩ STEPON – управляет расчетом значений функций на $n+1$ временном слое по заданным значениям на n -ом слое:
а) вызовом ⟨2.20⟩ SPEED по значению функции тока вычисляются значения скоростей, используемые далее в аппроксимации конвективных членов при решении уравнений для температуры и вихря;
б) вызовом ⟨2.30⟩ TEMPER решается уравнение для температуры и находится новое значение температуры на верхнем временном слое. Программа TEMPER вырабатывает числовой признак, который в случае получения отрицательной температуры получает значение -I. В STEPON анализируется этот признак. В случае получения отрицательной температуры вызовом ⟨2.35⟩ TIMSTP производится дробление шага по времени и расчет повторяется с начала с выдачей соответствующего сообщения. Программа TEMPER вырабаты-

вает параметр $DELTEM = \max_{i,j} |\hat{T}_{i,j} - T_{i,j}| / \tau$, по которому определяется установление решения по температуре.

в) вызовом ⟨2.40⟩ VORTEX решается уравнение для вихря и находится новое значение вихря на верхнем временном слое во внутренних точках. Программа VORTEX вырабатывает параметр $DELCUR = \max_{i,j} |\hat{\omega}_{i,j} - \omega_{i,j}| / \tau$, по которому определяется установление решения по вихрю.

г) вызовом ⟨2.50⟩ STRFUN решается уравнение для функции тока, что дает новое значение для функции тока на верхнем временном слое. Эта программа обращается к программе ⟨2.55⟩ POISON , решающей уравнение для функции тока итерационным методом, до тех пор, пока норма невязки в пространстве L_2 не станет меньше заданной. Число итераций при решении уравнения для функции тока на каждом шаге вычислений ограничено сверху максимально допустимым, при превышении которого выдается сообщение об ошибке и расчет прекращается.

д) вызовом ⟨2.60⟩ BOUNDY рассчитываются новые граничные значения вихря на верхнем временном слое.

На этом программа STEPON свою работу заканчивает. Пользуясь соглашениями системы OLYMPUS , любую из перечисленных программ 2-го класса можно выключить из расчета.

⟨2.70⟩ PRESS рассчитывает значение давления на верхнем временном слое. Вычисление давления при расчете не является обязательным на каждом временном слое, поэтому обращение к программе PRESS , принадлежащей по смыслу к классу вычислений, производится из программы управления выводом OUTPUT , и только на тех временных слоях, когда есть вывод.

К классу 3 относятся программы ⟨3.10⟩ OUTPUT, ⟨3.20⟩ MPRINT , ⟨3.30⟩ RECORD , обеспечивающие вывод информации на внешние устройства.

Класс 4. Эпилог.

⟨4.10⟩ TESEND - вырабатывает признак окончания расчета в следующих случаях:

а) при достижении заданного числа шагов;

б) при достижении заданного времени;

в) за TZERO сек. до исчерпания ресурса времени задачи;

г) при установлении температуры заданной с точностью;

д) при установлении вихря с заданной точностью.

Класс 5. Диагностика.

<5.40> ERROR – программа выдает сообщение о фатальных ошибках и заканчивает расчет.

Класс U . Утиллиты.

<U.01> APRINT – печать двумерного массива действительных чисел в указанном диапазоне и с указанным шагом по индексам;

<U.02> JORDAN – вычисляет оптимальный по Жордану набор итерационных параметров;

<U.11> DEFAC – определяет массив косинусов, используемых при вычислении конечных сумм в методе быстрого преобразования Фурье;

<U.12> FOUR81 – методом быстрого преобразования Фурье вычисляет конечные суммы Фурье по синусам.

4. Примеры применения программ.

Созданный программный комплекс много лет успешно эксплуатируется для решения различных физико-технических задач [5-7] .

I. Гайфулин С.А., Карпов В.Я., Мищенко Т.В. САФРА. Функциональное наполнение. Система OLYMPUS .Препринт ИПМ АН СССР, № 27, М., I980.

2. Горбунов-Посадов М.М., Карпов В.Я., Корягин Д.А., Красотченко В.В., Мартынюк В.В. Пакет прикладных программ САФРА. Системное наполнение . Препринт ИПМ АН СССР, № 85, М., I977.

3. Самарский А.А., Теория разностных схем. Наука, М., I977.

4. Герасимов Б.П., Один метод расчета задач конвекции несжимаемой жидкости. Препринт ИПМ АН СССР, № I3I, I974 г.

5. Герасимов Б.П., Калачинская И.С. Численное исследование тепло-массообмена в химических реакторах. Препринт ИПМ АН СССР № I97, I979 г.

6. Герасимов Б.П., Елизарова Т.Г. Численное исследование влияния свободной конвекции на тепловое просветление аэрозолей. Препринт ИПМ АН СССР № 69, I979 г.

7. Герасимов Б.П., Елизарова Т.Г. О тепловом самовоздействии светового пучка в присутствии свободной конвекции. Препринт ИПМ АН СССР № 83, I98I г.

8. М.И.Бакирова, Б.П.Герасимов и др. Программа расчета уравнений Навье-Стокса в приближении Буссинеска. Препринт ИПМ АН СССР № I3, I983 г.

ПАКЕТ ПРОГРАММ ДЛЯ ПРОЕКТИРОВАНИЯ ЭЛЕКТРОННО-ОПТИЧЕСКИХ СИСТЕМ

В.А.Катешов

Вычислительный центр СО АН СССР

I. Введение

Пакет ЭФИР предназначен для решения следующих задач.

а).Расчета двумерных плоских или осесимметричных электростатических полей.

б).Расчета осесимметричных полей с незначительным искажением геометрии [4, II, I2]. Учитываются возмущения первого порядка относительно параметров геометрических искажений следующих типов: сдвиг, поворот или перекос оси симметрии отдельных деталей, эллиптическая их деформация. Подобные задачи, например, возникают при расчете допусков электронно-оптических систем (ЭОС).

в).Расчета характеристик ЭОС. К ним относятся увеличение изображения, положение плоскости Гаусса, плоскости кроссовера, геометрические искажения изображения (дисторсия), коэффициенты пространственных, сферических и сферохроматических аберраций второго порядка, разрешающая способность приборов и др. см. [5, 6, 7].

г).Оптимизации ЭОС, требующих нахождения минимума функционала при одновременном удовлетворении некоторым ограничениям за счет варьирования геометрии области и краевых условий. Минимизируемый функционал и ограничения задаются в виде комбинации характеристик, описанных в п. "в". Варьирование геометрии допускает сдвиг, поворот, сжатие, растяжение отдельных частей границы области [I0].

Пакет разрабатывается с учетом использования его инженерами-разработчиками и техниками, непосредственно занимающимися созданием приборов. Поэтому наряду с обеспечением точности решения за-

дачи, выбора экономических алгоритмов реализации, будет уделено внимание проблеме автоматизации процесса решения задачи, в котором можно выделить этапы:

а) Автоматизация задания и обработки входной информации. Для этого разработан проблемно-ориентированный язык ВК и реализован транслятор с него, позволяющие в удобной и наглядной форме описывать постановку задачи.

б) Автоматизация построения алгоритма решения поставленной задачи. Производится путем реализации для каждого типа задач своей схемы работы алгоритмических модулей.

в) Оперативная обработка выходной информации. В пакете выходной документ можно получить в традиционном числовом виде, в виде рисунков, графиков на АЦПУ или графопостроителе. Реализуется такая возможность с помощью специального сервисного комплекса СЕРВИС, работа которого управляется директивами входного языка. Система СЕРВИС позволяет выводить расчетную область, расчитывать линии равного потенциала или равного модуля напряженности, силовые линии, строить векторные поля,графики различных функциональных зависимостей [8].

г) Хранение информации. Входная информация, результаты оптимизации, результаты решения полевых задач хранятся в базе данных ППП ЭФИР, работа с которой также производится с помощью входного языка ВК.

В данной работе приводятся численные алгоритмы, реализованные в пакете, структура пакета, описание сервисного комплекса и базы данных.

Пакет основан на использовании метода интегральных уравнений для потенциала простого слоя при аппроксимации плотности потенциала с помощью B - сплайнов различных порядков. Система алгебраических уравнений строится относительно коэффициентов сплайнового разложения. Вспомогательные интегралы вычисляются с помощью гауссовых квадратур с учетом асимптотического поведения в окрестности особых точек аналогично [I, 3]. Для решения системы алгебраических уравнений используется метод Гаусса с выбором главного элемента либо метод ортогональных преобразований отражения, реализованный в коплексе процедур КОПЛА [9]. Особенность последнего состоит в том, что процедуры по апостериорной информации выдают гарантированные оценки погрешности решения линейной системы, вызванные плохой обусловленностью матрицы.

Пакет позволяет проводить расчеты для областей, составленных из различного количества отрезков прямых и окружностей. На различных участках границы могут задаваться краевые условия I-го, 2-го, 3-го родов или условия сопряжения на поверхности раздела сред с различными диэлектрическими свойствами. Граница области может быть одно- или многосвязанной, замкнутой или разомкнутой.

Решение нелинейных задач оптимизации с ограничениями основано на применении квадратичной функции штрафов или алгоритмов модифицированной функции Лагранжа. Для решения безусловной минимизации используются алгоритмы наискорейшего спуска или алгоритмы сопряженных градиентов [2].

ППП ЭФИР полностью реализован на языке ФОРТРАН-IУ и адаптирован на ЭВМ серии ЕС и БЭСМ-6.

2. Решение задач оптимизации

Задача оптимизации осесимметричных ЭОС сводится к нахождению такого осевого распределения потенциала, при котором значение некоторого функционала $\mathcal{F}$, характеризующего оптические свойства ЭОС, достигает минимума.

Для решения задачи потенциал представляется в виде

$$\Phi(z) = \Phi_0(z) + \sum_{\kappa} \Phi_\kappa(z)\delta\beta_\kappa ,$$

где $\Phi_0(z)$ – осевое распределение для начальной краевой задачи (начальное приближение граничных условий и геометрии области), $\Phi_\kappa(z)$ – функции возмущения (влияния) – есть производные $\Phi_\kappa = \partial\Phi/\partial\beta_\kappa$ осевого потенциала, по параметрам определяющим возмущения геометрии границы или граничных условий, $\delta\beta_\kappa$ – величина данного (κ-го) возмущения.

Таким образом, оптимизационная задача сводится к отысканию возмущений $\delta\beta_\kappa$, при котором $\mathcal{F}$ достигает минимума.

Минимизируемый функционал представляется в форме

$\mathcal{F} = \sum_{i=1}^{N} a_i (J_i - J_{0,i})^2$, где $N \in \{1,2,3\}$, a_i, $J_{0,i}$ – заданные числа, J_i – характеристики электронно-оптического изображения, перечисленные во введении п. "в" и являющиеся функциями от осевого распределения $J_i(\Phi(z))$.

Минимизация функционала производится при линейных ограничениях (типа равенств или неравенств) на переменные $\delta\beta_\kappa$

$$\vec{\alpha} \le A\vec{x} \le \vec{\beta} ,$$

и при, возможно, функциональных ограничениях вида

$$A_k \le J_k \le B_k \quad , \quad k = 1, 2, \ldots$$

Здесь $\vec{\alpha} = (\alpha_1, \ldots, \alpha_m), \vec{\beta} = (\beta_1, \ldots, \beta_m), \vec{x} = (\delta\beta_1, \ldots, \delta\beta_m); \alpha_i, \beta_i, A_k, B_k$ – некоторые константы, A – матрица порядка m ; $m = NP + NG$, где NP – число функций влияния, вызванных вариациями потенциала, NG – число функций влияния, вызванных вариациями геометрии.

Производные от функционала и от функциональных ограничений определяются с помощью центральных разностей.

3. Описание пакета

В состав ППП ЭФИР входят следующие компоненты:

а) транслятор с входного языка,

б) набор модулей, реализующих численные алгоритмы,

в) набор модулей, реализующих обработку и вывод результатов счета,

г) база данных.

Структура пакета может быть представлена в виде блок-схемы на рис. I, где альтернативность выполнения работ показана с помощью параллельного соединения блоков, а последовательность выполнения – в виде последовательного соединения.

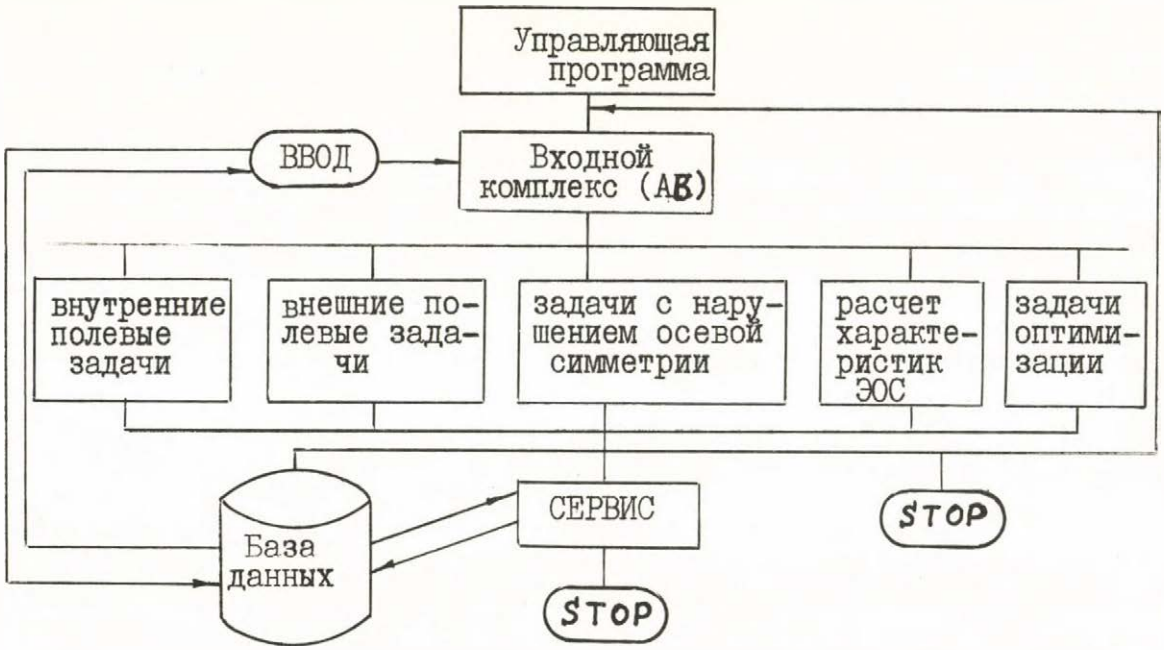


Рис.I

Литература

I.О.Ф.Антоненко. Численное решение задачи Дирихле для незамкнутых поверхностей вращения. Сб. "Вычислительные системы", вып. I2, Новосибирск, Наука, I968.

2.Г.И.Забиняко. Математическое обеспечение, применение и анализ выпуклого программирования. Автореферат дис. на соиск. уч.степени канд. физ.-мат.наук. - Новосибирск, I980, I5 с.

3.В.Н.Иванов, В.П.Ильин. Решение смешанных краевых задач для уравнения Лапласа методом интегральных уравнений. - Сб. "Типовые программы решения задач математической физики. - Новосибирск, изд-во ВЦ СО АН СССР, I975, с.5-35.

4.В.П.Ильин, В.А.Катешов. Пакет программ ЭФИР для расчета потенциалов и их возмущений. Автометрия, Новосибирск, № 4, I982.

5.А.Н.Игнатьев, Ю.В.Куликов. Радиотехника и электроника, I978, 23, II, 2470.

6.Ю.В.Куликов и др. Радиотехника и электроника, I978, 23, I, I67.

7.Ю.В.Куликов. Радиотехника и электроника, I975, 20, 6, I249.

8.В.А.Катешов. Сервисные программы пакета ЭФИР. Сб. Методы расчета электронно-оптических систем, часть П, изд-во ВЦ СО АН СССР, I983, I64...I7I.

9.З.А.Ляпидевская. Комплекс программ по линейной алгебре. Новосибирск:I980 (Препринт/АН СССР, Сиб.отд-ние, ВЦ; № 256).

I0.М.А.Монастырский. Интегральные уравнения в задачах оптимизации электронно-оптических систем. П.Осесимметричный случай. - Сб. Численные методы решения задач электронной оптики, Новосибирск, изд-во ВЦ СО АН СССР, I979, I09...I20.

II. W.Glaser, P.Schiske. Zeitschrift fur Angew. Phys., 1953, v.5,9,329...339.

I2. P.A.Sturrock. Phil. Trans., Ser.A, 1951, v.243, N868,387...429.

СИСТЕМА КОНСТРУИРОВАНИЯ ПРОСТРАНСТВЕННЫХ ГЕОМЕТРИЧЕСКИХ ОБЪЕКТОВ ГЕТРИЗ

Кислюк О.С.

СССР, Владивосток, ИАПУ ДВНЦ АН СССР

Системы конструирования сложных геометрических объектов являются ядром многих систем автоматизации проектирования. В работе описывается система конструирования сложных трехмерных геометрических объектов, обладающая следующими отличительными чертами.

Для описания топологии объектов предложена специальная схема представления, дающая возможность выделить множество правильно построенных объектов, доказать корректность введенных над ними операций.

Для построения поверхностей сложной формы используются рекурсивные дискретные B-сплайны.

Создан специальный язык и набор операций ориентированных на реализацию мощных методов конструирования таких как: переход от грубого описания к более точному, сочетания анализа и синтеза в процессе конструирования и др.

Рассмотрим описание системы более подробно. Введем множество описаний объектов. Объекты в системе описываются через их границу. Многогранник описывается набором многоугольников. Многоугольник набором ребер (и поверхностью, в которой они лежат в случае криволинейного многоугольника), отрезок описывается набором точек (и кривой, в которой они лежат в случае криволинейного отрезка).

Над множеством описаний вводится предикат $R(X,S)$, истинный, если точка принадлежит объекту и ложный, в противном случае. $R(X,S)$ истинен (ложен), если любой луч, выходящий из X пересекает границу S нечетное (четное) число раз. В случае

криволинейного многоугольника или отрезка вместо лучей рассматриваются образы лучей проведенных в параметрическом пространстве функций, задающих поверхность и кривую.

Если $R(X,S)$ определен для любой точки X, описание S является семантически корректным, поскольку R ставит ему в соответствие некоторое множество точек. Множество осмысленных описаний весьма обширно. В него входят, например, многоугольники и произвольным количеством дыр или многоугольники с самопересекающейся границей. Хотя определение интуитивно ясно, оно неконструктивно, поэтому мы переходим от семантической корректности к синтаксической корректности – свойству описания, которое легко проверяется.

Преимущество такого подхода к описанию объектов по сравнению с наиболее известным подходом Брэйда [I] состоит в том, что явно определена функция, ставящая в соответствие каждому допустимому описанию множество точек. Это гарантирует отсутствие бессмысленных описаний и сводит доказательство корректности аналогов теоретико-множественных операций к доказательству их гомоморфизма относительно R.

В качестве описаний объектов используются ориентированные ациклические графы. Каждой вершине ставится в соответствие некоторая геометрическая сущность: объект, многогранник, многоугольник, отрезок, точка, кривая, поверхность. Поверхности и кривые задаются параметрически. Соответствующие функции должны быть гладкими и взаимно-однозначными. Многогранник соединен дугами с многоугольниками, которые задают его границу, многоугольник с отрезками и возможно поверхностью, отрезок с точками и возможно кривой. Каждый граф имеет одну корневую вершину, которой сопоставляется весь объект. Она соединена дугами со своими подобъектами: многогранниками, многоугольниками, отрезками или точками.

Определение I. Граф с корневой вершиной S является семантически допустимым, если $R(X,S)$ определен для любой точки X.

Определение 2. Граф с корневой вершиной S является синтаксически допустимым, если:

I. Любую вершину типа многогранник и любую вершину типа отрезок соединяет четное число путей.

2. Любую вершину типа многоугольник и любую вершину типа

точка соединяет четное число путей.

3. Из любой вершины типа отрезок исходит четное число дуг к вершине типа точка.

4. Все граничные отрезки многоугольника лежат на поверхности, которой принадлежит многоугольник.

5. Все точки, являющиеся границей отрезка, лежат на кривой, которой принадлежит отрезок.

Предложение I. Любой синтаксически допустимый граф является семантически допустимым.

Предложение 2. Существует алгоритм, строящий по данному семантически допустимому графу эквивалентный ему синтаксически допустимый граф.

Над введенными описаниями геометрических объектов определены аналоги теоретико-множественных операций: объединение, пересечение, вычитание, а также ряд других операций, о которых будет рассказано ниже. Для теоретико-множественных операций определен гомоморфизм относительно R. Например, если $\tilde{\cap}$ аналог операции пересечения, S, S_1, S_2 корневые вершины графов, а X и Y точки, то

$$R(S_1 \tilde{\cap} S_2, X) = R(S_1, X) \cap R(S_2, X) \qquad (I)$$

Доказательство предложений I, 2 и равенства (I) можно найти в работе [3] . В качестве базовых поверхностей в системе используются плоскости и частично поверхности второго порядка.

Рассмотрим теперь важный вопрос описания поверхностей произвольной геометрической формы. Одним из наиболее удобных способов описания поверхностей являются кубические B-сплайны. К их достоинствам можно отнести:

а). Естественность параметризации. Сплайн задается точками в вершинах прямоугольной сетки в параметрическом пространстве (характеристический многогранник).

б). Возможность локальной модификации формы поверхности.

в). Отсутствие избыточной волнистости.

Однако классическая B-сплайновая поверхность может быть определена только над прямоугольной сеткой. Обобщением B-сплайнов являются дискретные рекурсивные B-сплайны [2] , которые задаются над произвольной сеткой. Если классические сплайны зада-

ются как функция

$$\sum_{j=0}^{k}\sum_{i=0}^{p} \bar{r}_{ij} N_{ij}(u,v)$$

над вершинами $\bar{r}_{ij}$ характеристического многогранника, то дискретные рекурсивные сплайны задаются функционалом $F(S)$ (где сетка), который позволяет по сетке S_i строить сетку S_{i+1}. С помощью такого функционала мы получаем последовательность сеток, каждая из которых является все более точным приближением искомой поверхности.

Хранение сетки S_i целиком не обязательно, поскольку в силу свойства локальности B-сплайнов удается построить алгоритм для создания лишь малого участка поверхности.

Использование дискретных рекурсивных сплайнов позволяет создать удобный аппарат конструирования сложных криволинейных объектов: вначале создается грубое приближение, а затем оно уточняется при помощи функционала .

Рассмотрим теперь сам процесс конструирования. Отличительной чертой описываемой системы является возможность использования нескольких весьма мощных методов конструирования.

Разработан специальный диалоговый язык и для обеспечения использования методов конструирования, причем используются не только наборы процедур, но и языковые средства, например, возможность недетерминированных вычислений (программирование с возвратами).

Основные методы конструирования, используемые в системе, это:

I. Сочетания процесса конструирования с процессом анализа.

Для конструирования объектов используются: теоретико-множественные операции, преобразование объекта или его части оператором, построение обобщенного цилиндра (например, тела вращения), построение тела по оболочке.

Для анализа объектов используется языковый механизм недетерминированных вычислений и возможность существования многозначных функций. Это позволяет строить "образцы", которые можно накладывать на исследуемый объект. При этом используются многозначные функции, строящие все подобъекты данного объекта равные объекту S, все подобъекты данного объекта на которые указывает световое перо, топологически связанные подобъекты данного объекта и др.

2. Переход от грубого описания к уточнению.

Осуществление этого механизма конструирования обеспечивается созданием произвольной сетки, а затем построением по ней рекурсивных дискретных В-сплайнов.

3. Неявное задание операции перемещения объектов.

Этот механизм конструирования обеспечивается: во-первых, специальной процедурой, строящей оператор смещения или оператор смещения - поворота по его действию на плоскости, прямые, точки и объекты. Во-вторых, набором процедур анализа строящих плоскости, прямые и центры симметрии, плоскости и прямые, в которых лежит объект, и некоторых других.

4. Непосредственное создание объекта.

Этот механизм обеспечивается наличием специального объекта "плавающий вектор", которым можно пользоваться как "чертежным инструментом".

При разработке данной системы были созданы несколько алгоритмов. Наибольший интерес представляют:

алгоритм построения векторных изображений с удалением невидимых линий,

алгоритм построения пересечения и объединения объектов,

алгоритм построения рекурсивных дискретных В-сплайнов.

Литература

1. Braid I.C. Notes on a Geometric Modeler C.A.D. Group Document N°101: University of Cambrige, 1980
2. Catmull E. and J. Clark "Recursively Generated B-spline Surfaces on Arbitrary Topologically Meshes", Computer Aided Design, Vol. 10, No. 6, 1978, p. 350-355
3. Кислюк О.С. Синтез геометрических объектов. Обобщенные многогранники. Препринт. Владивосток: ИАПУ ДВНЦ АН СССР, 1982.

ПРИНЦИПЫ ПОСТРОЕНИЯ И ОСОБЕННОСТИ РЕАЛИЗАЦИИ КОМПЛЕКСА ПРОГРАММ РАДИУС-2

Климачев С.Н.
СССР, г. Москва

Дорри М.Х.
СССР, г. Москва

I. Введение

Проектирование систем автоматического управления представляет собой сложную задачу, требующую проведения большого объема вычислений для объективного обоснования проектных решений. В современных условиях остро встает проблема повышения производительности инженерного труда путем рациональной организации взаимодействия проектировщика с ЭВМ.

Эта проблема имеет три главных аспекта:

- создание специального входного языка, допускающего удобный способ задания вычислительной машине модели исследуемой системы в привычных для проектировщика терминах ;

- алгоритмизация и программирование методов теории автоматического управления ;

- согласование и стыковка отдельных алгоритмов и программ между собой.

В докладе излагается комплексный подход к решению этой проблемы, учитывающий все три ее перечисленные аспекта, и его реализация в комплексе программ РАДИУС-2 (РАсчет ДИнамических Управляющих Систем), предназначенном для автоматизации процессов исследования и проектирования структур систем управления сложными техническими объектами.

2. Принципы построения

При выработке принципов построения комплекса программ

РАДИУС-2 отправной точкой послужил анализ процесса подготовки и исследования модели с помощью ЭВМ, схематически изображенного на рис.I. Отдельные блоки соответствуют различным этапам этого процесса, а в правой части рисунка выписаны принципы построения комплекса, вытекающие из особенностей каждого этапа.

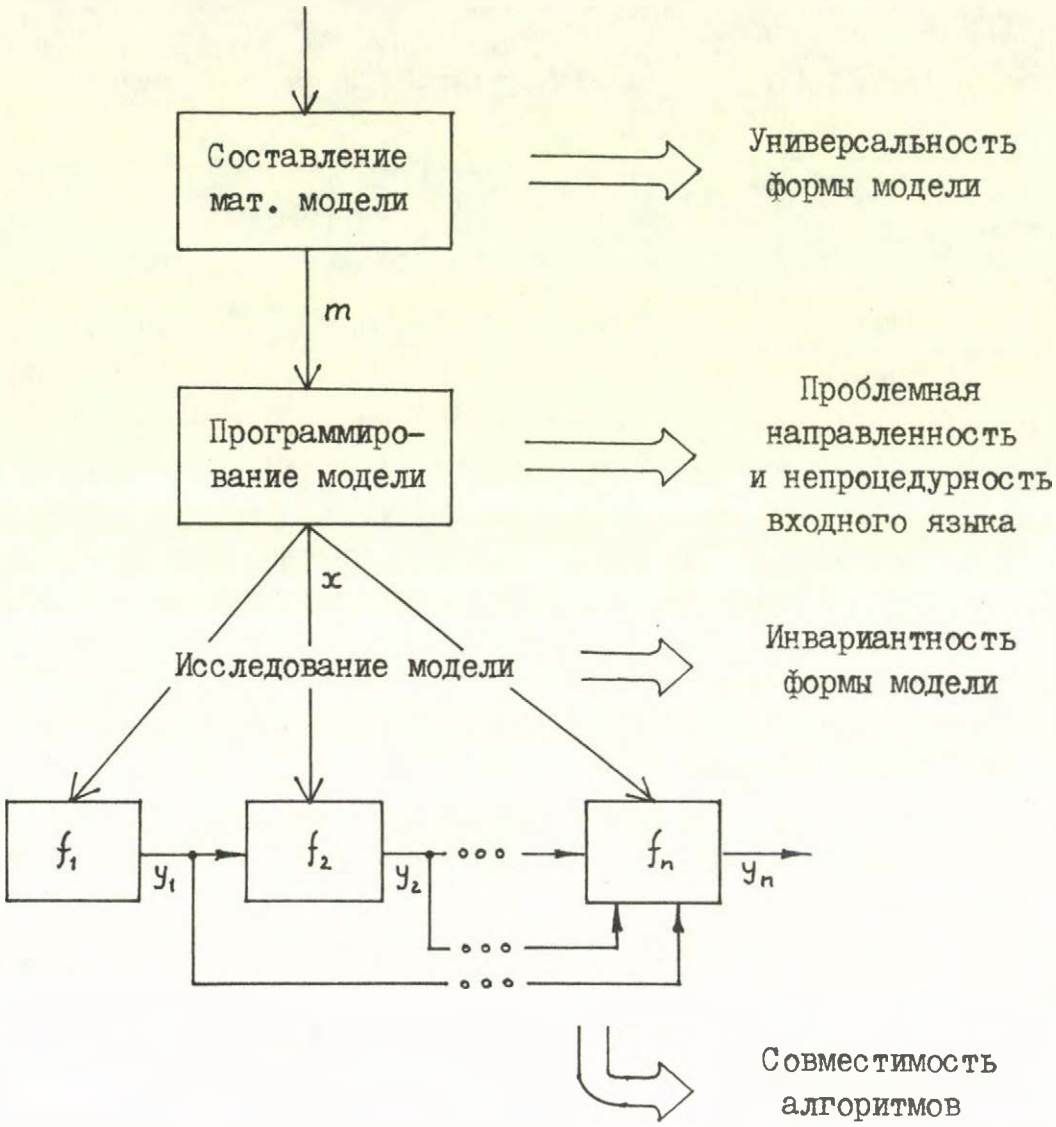


Рис.I

Благодаря универсальности формы модели удалось охватить достаточно широкий класс исследуемых систем. В качестве основных форм модели приняты два наиболее общих способа описания систем автоматического управления: структурная схема и система уравнений в пространстве состояний.

На этапе программирования модели важнейшими принципами являются проблемная направленность и непроцедурность входного языка. Проблемная направленность позволяет программировать модель в терминах, принятых в данной прикладной области деятельности. Непроцедурность дает возможность задавать в программе только модель исследуемого объекта и системы управления, не расписывая алгоритм ее обработки. В области программного обеспечения для исследования динамических систем свойства проблемной направленности и непроцедурности наиболее полно воплощены в языках моделирования, наиболее развитые из которых послужили прототипом при разработке входного языка комплекса РАДИУС-2, который отличается от них двумя важными качествами. Во-первых, эти свойства реализованы не за счет разработки специализированного языка и соответствующего транслятора, а в рамках Фортрана-4 путем его семантического расширения. Во-вторых, помимо средств моделирования динамических процессов комплекс включает алгоритмы и программы, реализующие различные методы анализа и синтеза автоматических систем, такие как расчет устойчивости, оптимизация по заданному критерию и др.

Характерно, что все расчетные алгоритмы допускают запись исследуемой модели в одной и той же форме. Иначе говоря, выбранная форма модели обладает свойством инвариантности относительно алгоритмов расчета. В результате этого возникает возможность независимого применения нескольких алгоритмов $f_1, f_2, \ldots, f_n$ к одной и той же модели x:

$$[f_1, f_2, \ldots, f_n] : x = \langle f_1 : x, f_2 : x, \ldots, f_n : x \rangle. \qquad (1)$$

Поскольку, кроме того, алгоритмы совместимы друг с другом по входной и выходной информации, то можно выполнять комплексные расчеты, предполагающие последовательное применение различных алгоритмов $f_1, f_2, \ldots, f_n$, каждый из которых использует результаты работы предыдущих:

$$(f_1 \circ f_2 \circ \ldots \circ f_n) : x = f_1 : (f_2 : (\ldots (f_n : x))). \qquad (2)$$

При такой формализации напрашивается аналогия с функциональным стилем программирования: выражение (1) можно трактовать как конструкцию алгоритмов $f_1, f_2, \ldots, f_n$, а выражение (2) - как их композицию. Набор элементарных функций включает далеко не элементарные алгоритмы анализа и синтеза систем управления.

В качестве объекта выступает модель исследуемой системы x, которая, как правило, состоит из тройки разделов p, o и s, определяющих соответственно параметры, выводимые переменные и структуру модели:

$$x = (p, o, s).$$

Структура модели кодируется с помощью функциональных блоков. Имеются функциональные блоки трех разновидностей: математические соотношения (дифференциальные и алгебраические уравнения, генераторы функций и др.), элементы систем автоматики (типовые динамические звенья, однозначные и неоднозначные нелинейности и др.) и модели технологических объектов (таких как теплообменник, турбина и др.). Программа пользователя состоит из описания одной или нескольких моделей $x_1, x_2, \ldots, x_m$ и директив на проведение расчетов по алгоритмам $f_1, f_2, \ldots, f_n$. На рис.2 показан пример организации программы пользователя.

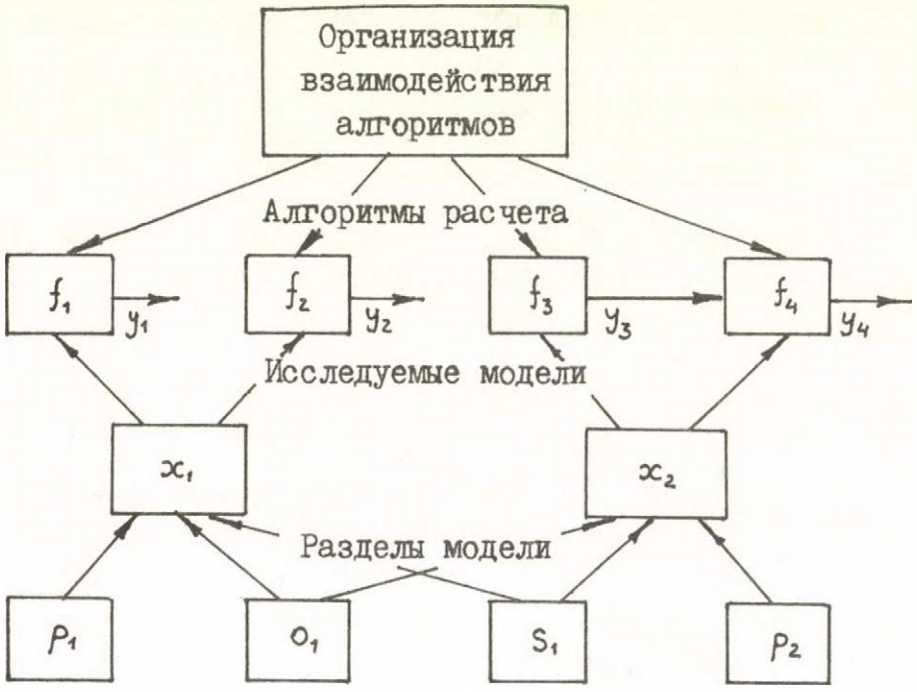


Рис.2

В этой программе исследуются модели x_1 и x_2 с помощью алгоритмов f_1, f_2, f_3 и f_4. Алгоритмы f_1 и f_2 обрабатывают модель x_1 :

$$[f_1, f_2] : x_1 = \langle f_1 : x_1, f_2 : x_2 \rangle,$$

а алгоритмы f_3 и f_4 – модель x_2:

$$(f_4 \circ f_3) : x_2 = f_4 : (f_3 : x_2),$$

причем расчеты f_1 и f_2 выполняются независимо, а расчет f_4 использует результаты расчета f_3. Модели x_1 и x_2 имеют одинаковую структуру и операции вывода, отличаясь только параметрами:

$$x_1 = (p_1, o_1, s_1); \quad x_2 = (p_2, o_1, s_1).$$

3. Особенности реализации

РАДИУС-2 представляет собой сложный программный комплекс, насчитывающий более 100 подпрограмм общим объемом более 10 тыс. операторов.

Основными подсистемами комплекса являются:

- входной язык, к которому примыкает библиотека функциональных блоков;
- набор алгоритмов расчета;
- подсистема вывода информации.

Входной язык предназначен для задания ЭВМ исследуемой модели и цели исследования в виде последовательности директив, определяющих тип и параметры проводимых расчетов. Входной язык комплекса РАДИУС-2 реализован в синтаксических рамках Фортрана-4 за счет его семантического расширения. Набор алгоритмов реализует методы теории автоматического управления и другие численные методы, используемые при исследовании и проектировании систем управления. Подсистема вывода информации обеспечивает выдачу результатов расчета на внешние устройства в удобном для восприятия и последующего анализа виде, как цифровом, так и графическом.

Комплекс целиком реализован на универсальном языке программирования Фортран-4. К настоящему времени он прошел апробацию на ЭВМ ICL 4-70 (с операционной системой DOS-J), а также М-4030 и различных моделях ЕС ЭВМ (с различными версиями операционных систем ОС ЕС и ДОС ЕС).

4. Заключение

В докладе описаны принципы построения и особенности реализации комплекса программ РАДИУС-2, предназначенного для автоматизированного проектирования структур систем управления.

Благодаря заложенным в него принципам комплекс чрезвычайно удобен в эксплуатации. Так, вследствие проблемной направленности входного языка пользователь вводит в машину модель исследуемой системы буквально с листа, не производя предварительной перекодировки. Непроцедурность языка освобождает пользователя от необходимости расписывать алгоритм расчета и позволяет ему сконцентрировать все внимание на решении основной задачи. Благодаря инвариантности формы модели отпадает необходимость каждый раз переписывать модель при переходе от одного алгоритма к другому, т.к. все преобразования модели из одного вида в другой выполняются автоматически. Наконец, совместимость алгоритмов позволяет осуществлять сложные программы исследования модели путем последовательного применения к ней различных алгоритмов, каждый из которых может использовать результаты работы предыдущих.

С точки зрения реализации комплекса РАДИУС-2 чисто фортранное исполнение обеспечивает его мобильность, легкость освоения и наращивания, а также возможность использования многолетнего опыта программирования, материализованного в многочисленных программах и пакетах программ.

AUTOMATED LINKING OF PROGRAMS IN THE SYSTEM OF BPS

Igor Klačanský

Computing Research Centre
Dúbravská 3
Bratislava
Czechoslovakia

The paper discusses the problems concerned with the automated linking of programs in the system with modular language. It contains a brief description of the BPS system, of the BPS/L language and the definition of the module as the fundamental unit of the language. Further, the structure of the program is defined and the requirements for the structure are given as the result of the principles of modular and structural programming. The subsystem Linker, which ensures the automated linking of programs in BPS, is adduced as an example of the practical implementation.

0. INTRODUCTION

If we follow the development of programming means we may state that the centre of gravity of the development was shifted from the field of programming languages to the integrated programming environments - to the systems supporting the program during its whole life cycle and making the programming easier in several aspects. The means allowing an automated linking of programs is an important part of such a system.

It is possible to say that the progressive programming methodologies left the university campus and through programming environments have been applied in practice.

The first part briefly characterizes the BPS system on SM4-20 computer which in many aspects is likely to become a programming environment. The second part deals with the problems of

linking the programs in BPS and with the possibility of automating this activity. The third part describes the automated tool LINKER.

1. A BRIEF CHARACTERIZATION OF BPS

BPS - the Bratislava Programming System /1/ has been developed in the Computing Research Centre in Bratislava and is implemented on the IBM, CDC, HP, PDP computers and their equivalents. In the following part of this paper we shall be concerned with the version on SM4-20.

BPS is an interactive programming system supporting the program during its whole life cycle /exclusive of semantic specifications/, i. e. the subsystems providing

- designing and debugging of the program structure,
- encoding, translating and debugging of source texts,
- linking of programs,
- debugging of programs,
- maintenance of programs and their modification,
- generating of output documentation,

are a component part of BPS.

BPS is aimed at more efficient, easier and more rapid building of large programs, namely by means of

- automation or the automated performing of mechanical works,
- making the user to apply up-to-date programming methods and techniques,
- integrated approach to the creation and maintenance of the programs.

BPS makes it possible to program in two languages of different levels:

BPS/L

macroassembler.

A Short Characterization of the BPS/L Language

BPS/L /2/ is a modular language based on the principles of structured programming. It ranks among the Pascal-like languages. Originally it was designed for the system programming. In the beginning it followed the MODULA language. Successively, according to the requirements of practice, it was extended in several stages to further language tools so that it has become a universal programming tool. It contains current control and data structures. Besides, it provides rich means for text-processing, pointers, all basic arithmetics, predefining of operators in expressions, parallel constructions - monitors, etc. BPS support a strong type check. The language compiler is completed by the standard modules which the user can change or create:

- I/O operations,
- communication with the operating system,
- interlink with the RMS system,
- other run-time supports

ensuring communication with the environment.

The module is a basic, separately compilable language unit /and of the whole system, too/. It consists of two parts:

- procedure bodies.

The specifications contain the definitions and declarations of the global and local module objects /all the objects used in the module/, namely

- constants,
- data types,
- variables,
- procedures.

The global objects represent a single means of communication with the environment and /with other modules/. The local objects cannot be seen outside the module. The state of the module can be changed outside the module only by means of its

global procedures. This principle allows to hide the implementation details from the public - the implementation changes are present only in the given module. The concept of the module is very close to the concept of the abstract data type /3/.

2. LINKING OF THE MODULES IN BPS

The hierarchical program structure

The specifications of the module contain a USE clause, the list of the module names, the global objects of which can be used by the module - we say that the given module can see these modules. The global objects of other modules are not accessible for the given module.

The relation of visibility over the program modules is antisymmetrical, i. e. if A can see B, then B cannot see A. The visibility relation partially orders the program modules so that the "higher" modules can see the "lower" ones and the program contains a single highest module labelled as MAIN. The program is started from this module. In this way we have defined the hierarchical structure in the program. If we assign nodes to the modules, oriented edges to the visibility relations, the program structure can be represented by a root acyclically oriented labeled graph. This graph is called the visibility graph of the program. The graph evaluation /higher value is assigned to a "higher" module/ is dividing the graph into levels. The modules of one level cannot see one other.

Important note

Practice has proved that visibility graph is an important characteristics of the program. The quality of the program is dependent on how well has the visibility graph been designed. Designing of a suitable program structure is the matter of considerable experience and sensibility. BPS supports the program structure design by allowing to compile,

link and document the modules created only by the global specifications. The possible errors can be easier removed than in a completed program. It is more expensive in the latter case.

Check of the hierarchical structure of the program

The acyclic character of the visibility graph and the uniqueness of the MAIN type module represent inevitable conditions imposed on the visibility graph. To this the condition of consistency can be added.

Every module contains internal information in the specifications which help to characterize the module. We are interested here in the referential number of the module which should be there - how many times the global specifications were changed /the number increases at every such change/. Apart from it, there are the referential numbers of all the modules which can be seen by the given module, more preciselly referential numbers which are current in time of creating the specifications. If so remembered referential numbers are not consistent with the current ones in time of linking - the global objects of the modules, which can be seen by the modul, have changed since the former compilation - the modul is inconsistent. In the opposite case the module is consistent.

In order to provide a proper linking of the module with its environment, it is necessary to recompile the inconsistent module.

The above said shows us the work of the automated tool for linking of programs. It analyzes the USE lists successively in all program modules - it starts with the MAIN type module, the program is linked and the given requirements are checked at the same time. One has to do with searching through the visibility graph in width /at simultaneous creating top-down/.

If we now assign a similar referential number to every USE list, we can considerably simplify the check-up of the already linked programs. If the library contains a stored list of modules /with the referential numbers of the USE clauses/ forming the program, it is sufficient then at the repeated linking of the program to check only the subgraph of the visibility graph "under" the module signalling the change of the USE list. /It is reflected mainly with the large /segmented/ programs as considerable time-saving.

3. LINKER - THE AUTOMATED SUBSYSTEM

The above given principles apply also to all the systems resembling BPS. In this part we shall illustrate the subsystem of the BPS system - LINKER which was written by the author of this article.

LINKER performs all the activities described above as well as those that are given by the BPS system and by the DOS RV environment on SM4-20. It supports the linking of simple and segmented programs /composed of the segments which overlap in memory/. Its work is automated in such respect that the input represents only the name of the "main" module. Other activities are performed with the user´s absence. It serves as a preprocessor for the system task builder TKB /4/. It generates and compiles input information and the control files for TKB. If necessary, BPS ensures a start of TKB immediately after LINKER completes its work, i. e. it provides the creation of task image of the program.

LINKER performs all checks either during compiling the visibility graph or when crossing it. The graph is compiled in such a way that the USE list of the main program module or of its segment is analyzed and this analysis is repeated on all modules which can be seen by the given module, atc. /search through the visibility graph in width/, at the same time LINKER records the analyzed modules. If they occur in the visibility graph again, their USE list are no more

analyzed. At the same time it should be checked if the main module of the program is of the MAIN type, if all the modules are present /the specifications of modules/. When the visibility graph has been compiled, it is searched within its depth /every path is searched which leads from the main module/ and the acyclic character and consistency are being checked. In every path leading from the graph root every module can occur no more than once, the opposite case indicates that the graph contains a cycle. The referential number of every module must be consistent with the referential number which is given at its occurence in a USE list. If this does not hold true, the program is inconsistent. If during search LINKER comes across the cycle, it stops working.

The things are more complicated when the program is overlapped. Only the main module of the main segment must be of the MAIN type /the main module of the segment is a module which is at the highest level in segment/. Correctness is ascertained separately for each segment. The whole activity of the LINKER program is directed in two degrees. In the first degree it is directed by the graph of segments of the program, in the second degree by the visibility graph of the just analyzed segment. The acyclic character is tested only within the branch of the graph segments /paths from root to leaves/ as in large programs LINKER would not be able to keep information on all program modules. At the same time the information on the already analyzed segments, which are not a component of the just searched branch, gets lost. The visibility graph of the whole program is not immediately in memory.

Apart from it, LINKER provides

- a check of the occurence of the mutually excluding standard modules,
- linking of other implicit modules,
- linking of debugging means and the RMS system, etc.

The dokumentational tools of BPS are also based on LINKER.

4. CONCLUSION

The article contains a brief characterization of the BPS system with the BPS/L programming language. The technique of linking the programs in the systems resembling BPS is analyzed here. For illustration the actually applied automated system LINKER implemented on SM4-20 is described.

/1/ BPS, Reference Manual, VVS Bratislava, 1982

/2/ BPS/L, Reference Manual, VVS Bratislava, 1980

/3/ GOGUEN, J. A.: An initial algebra approach to the specification... In Current Trends in Programming Methodology, Prentice Hall, 1978

/4/ RSX-11M Task Builder, Reference Manual, Dec. 1978

ОДИН ПОДХОД К ПОСТРОЕНИЮ РАСПРЕДЕЛЕННОЙ СИСТЕМЫ МОДЕЛИРОВАНИЯ

Пополитов В.Н.
Институт проблем управления, Москва, СССР

I. Введение

Моделирование на ЭВМ является одним из наиболее удобных и гибких методов исследования и проектирования сложных динамических объектов. В последние годы все более актуальными становятся задачи моделирования, решение которых эффективно только при использовании возможностей распределенных (многомашинных) вычислительных комплексов (РВК). В частности, особую важность развитие распределенных систем моделирования (СМ) приобретает в связи с необходимостью построения сложных моделирующих комплексов с ЭВМ в контуре управления.

Недостаточное развитие СМ высокого уровня для РВК значительно усложняет задачу исследователей и ограничивает возможности активного использования методов имитационного моделирования, так как перед программистом (и тем более специалистом, выполняющим постановку эксперимента) возникает ряд специфических проблем, таких как:

- описание динамической модели объекта с учетом того, что ее отдельные подсистемы могут быть реализованы на различных ЭВМ (в том числе разнородных);
- распределение и перераспределение подсистем модели между отдельными компонентами РВК (ЭВМ и реальной аппаратурой);
- синхронизация развития и организация информационно-управляющего взаимодействия подсистем динамической модели, реализованных на различных ЭВМ.

Пока многомашинные комплексы использовались редко и для решения отдельных частных задач, связанных с постановкой и исследованием уникальных динамических моделей, а создаваемые модели использовались многократно практически без существенных изменений, была оправдана технология "автокодного программирования": большая часть многомашинных моделей непосредственно базируется на машинно-зависимых системах автокодного типа и сетевых пакетах, обеспечивающих передачу информации между несколькими ЭВМ. При этом разработчики моделей должны быть весьма квалифицированными программистами.

Отмеченная тенденция расширения области активного применения многомашинных СМ требует привлечения к созданию программ моделирования больших коллективов пользователей (большинство разработчиков имитационных моделей являются системными исследователями и лишь по необходимости знакомятся с программированием). А это, в свою очередь, требует разработки распределенных СМ высокого уровня.

В качестве примера рассмотрим ряд задач, возникающих при разработке систем управления сложным динамическим объектом с ЭВМ в контуре. Проектирование и отработка таких систем часто проводится на многомашинных моделирующих стендах, позволяющих подключать к ним реальную аппаратуру. Предположим, что реализована полная программная модель системы, включающая модели объекта и подсистемы управления. Использование традиционных систем моделирования [1,3] позволяет в этом случае выполнять в основном качественное исследование алгоритмов управления. При этом в стороне остаются такие вопросы как моделирование собственно управляющих ЭВМ, временные характеристики реальной аппаратуры и другие важные факторы исследования моделей рассматриваемого класса динамических объектов. Кроме того существующие методы и системы моделирования не позволяют непосредственно использовать для более детального исследования исходную программную модель без значительной перестройки. Чтобы убедиться в этом, рассмотрим два варианта: аппаратную реализацию компонент объекта и включение ЭВМ в состав системы управления. В первом случае необходима модификация остающейся программной части модели (по крайней мере везде, где имеются связи "программа - реальная аппаратура"). Во втором случае должна дорабатываться модель объекта с целью обеспечения правильного взаимодействия с управляющей ЭВМ (и соответственно перекодировать-

ся должна программа управления).

Указанная перестройка весьма трудоемка и требует высокой профессиональной подготовки разработчика. Для эффективного решения указанных задач необходимы как повышение уровня СМ, так и обеспечение распределенности на уровне логической структуры многомашинной СМ.

Исследованию одного подхода к построению таких СМ и посвящена данная работа.

2. Концептуальная схема СМ

Основу построения формальной модели СМ, отражающей особенности реализации на РВК, составляют следующие предположения:
- модель рассматривается как совокупность развивающихся и взаимодействующих процессов, каждый из которых представляет автономный программный модуль, реализуемый на одной из ЭВМ РВК;
- динамика модели полностью контролируется подсистемой управления, в которую "погружены" процессы. Управляющая подсистема обеспечивает продвижение системного времени моделирования и согласованное информационно-управляющее взаимодействие процессов в рамках РВК.

Состояние динамической модели в каждый момент полностью определяется подсистемой управления (ADS^{MOD}), состоянием процессов, входящих в состав модели (P^{MOD}) и механизмом информационного взаимодействия между ними (L^{MOD}).

Основу определяемой схемы составляет структурный процесс, описываемый двумя компонентами: "внешней" ($Int=(X,Y,U)$), задающей информационно-управляющие связи процесса с остальной частью модели, и "внутренней" ($Fun=(V,W)$), полностью характеризующей конкретную функциональную реализацию процесса, при этом:

1) X и Y - порты, через которые осуществляются все возможные информационные связи процесса в виде сообщений;

2) в области управления U задаются параметры, определяющие текущее (в том числе исходное) состояние процесса и его влияние на динамику модели, потребности процесса в системных ресурсах и связь "внешней" и "внутренней" компонент;

3) область свойств V содержит полную спецификацию всех атрибутов (собственных параметров) процесса;

4) область структуры W состоит из конечного множества этапов активности. Каждый этап реализуется как вычислимая функция на множестве собственных параметров процесса.

Относительно структурных процессов предполагается, что они согласованы по информационно-управляющим связям и могут быть (в зависимости от своего состояния, анализируемого на уровне ADS^{MOD}) активны или пассивны.

В результате задача эффективного представления распределенной СМ решается в предлагаемом подходе на системном уровне [2] путем (а) явного разделения множества процессов на управляющие, составляющие ADS^{MOD}, и функциональные; (б) структуризацией функциональных процессов; (в) выделением автономной подсистемы, поддерживающей взаимодействие процессов, реализованных на ЭВМ РВК.

3. Логическая модель распределенной СМ

Базовая (физическая) модель, отражающая принципиальную реализуемость распределенной модели некоторой реальной системы, предполагает:

- декомпозицию моделируемой системы на множество автономно развивающихся и взаимодействующих подсистем-процессов;
- локализацию функционального развития и децентрализацию информационно-управляющего взаимодействия по отдельным процессам модели;
- выделение межпроцессного взаимодействия как основного свойства, характеризующего распределенность обработки информации.

Таким образом распределенность СМ реализуется на уровне процессов и определяется схемой их взаимодействия. Предполагается, что динамика связей любых процессов ρ_i и ρ_j может быть полностью охарактеризована набором "помеченных" сообщений (m), который назовем "полной историей взаимодействия ρ_i и ρ_j ":

$S_{ij} = \{(t_1, m_1), \ldots, (t_k, m_k)\}$, где $(\forall \ell, n)(\ell \neq n \Rightarrow t_\ell \neq t_n \,\&\, \ell > n \Rightarrow t_\ell > t_n)$,

$[t_1, t_k]$ - интервал взаимодействия ρ_i и ρ_j .

"Динамической историей взаимодействия ρ_i и ρ_j " назовем

$$h_{ij}(t) = \begin{cases} ( ) & , \text{если } t < t_1 \\ \{(t_1, m_1), \ldots, (t_\ell, m_\ell)\} & , \text{если } t_\ell \leq t < t_{\ell+1} \\ S_{ij} & , \text{если } t \geq t_k \end{cases}$$

Логическая модель распределенной СМ строится как совокупность асинхронно развивающихся во времени моделирования логических процессов, имитирующих функции и взаимодействие процессов физической модели.

Предлагается конструктивная процедура, обеспечивающая постро-

ение для любой физической модели функционально эквивалентной ей логической модели. Основу этой процедуры составляет алгоритм формирования динамических историй взаимодействия по выходным каналам (портам) каждого логического процесса. Доказывается корректность предложенного алгоритма.

Основным результатом проведенной работы является Утверждение. Построенная логическая модель представляет управляемую потоком данных корректную реализацию корректной физической модели.

4. Опыт реализации СМ

Проведенные исследования позволили перейти к практической реализации СМ распределенного типа для локальной вычислительной сети иерархической структуры. В частности,были реализованы:
- система непрерывного моделирования универсального типа;
- ядро монитора реального времени для управляющей микроЭВМ.

Реализация одномашинной версии СМ непрерывных систем(МОНЕС) позволила проверить возможности использования предложенного подхода при построении СМ высокого уровня (организация подсистемы управления, выбор входного языка и структуры системы трансляции).

Реализованный диспетчер реального времени является центральной функциональной компонентой программного обеспечения локальной вычислительной сети (фактически, это операционная система терминальной ЭВМ). В качестве инструментальной системы был выбран язык *PASCAL* , а основной единицей работы параллельного расширения *PASCAL* - процесс. Процесс рассматривается как программная ветвь, выполняющаяся логически параллельно с другими процессами. Для организации работы с процессами был реализован ряд механизмов, составляющих базовое ядро и ядро реального времени.

Каждый процесс имеет приоритет, от которого зависит его переход из состояния "готовности" в "активное" состояние. Активным считается процесс, стоящий первым в системной очереди процессов, готовых к выполнению. Для синхронизации процессов используются два механизма: семафоры и события. Семафоры и связанные с ними операции (*WAIT* и *SIGNAL*) позволяют работать с ресурсами взаимного исключения, тогда как события и соответствующие операции (*AWAIT* и *CAUSE*) обеспечивают возможность процессу реагировать на некоторую описываемую программным событием ситуацию.

Специально выделен механизм работы с прерываниями от реальной

аппаратуры: процесс может заказать ожидание такого прерывания и возобновить свою активность только после его прихода. Имеется также возможность отложить исполнение текущего процесса на некоторое время, по истечении которого соответствующий процесс снова переходит в состояние готовности.

Двухуровневая структура ядра позволила сделать систему достаточно простой и мобильной. Изменения, внесенные в исполнительную PASCAL -библиотеку, сделали генерируемый код системно независимым и позволили программам, написанным с использованием ядра, выполняться на любой микроЭВМ с системой команд СМ-4 без операционной системы.

5. Заключение

Введенная общая структурная схема СМ обобщает свойства известных систем и обеспечивает моделирование широкого класса динамических объектов.

Предложенная физическая модель выделяет конкретный класс распределенных СМ, а построенная функционально эквивалентная этому классу логическая модель соответствует распределенной СМ, управляемой потоком сообщений.

В рамках проекта многомашинной СМ реализованы цифровая система непрерывного моделирования с входным языком высокого уровня и ядро параллельного программирования, выполняющее функции операционной системы терминальной ЭВМ локальной вычислительной сети.

Полученные результаты могут быть использованы при построении распределенных СМ динамических объектов и позволяют перейти на качественно новый уровень постановки и проведения имитационных экспериментов на РВК.

Литература

I. Cristopher T., Evens M., Gargeya R.R., Leonhardt T. Structure of a distributed simulation system. 3rd Int.Conf.Distrib. Comput.Syst.,Miami/Ft.Landerdale,Fla,Oct.,18-22,1982, Silver Spring, Md, 1982, 584-589.

2. Месарович М., Такахара Я. Общая теория систем: математические основы. М., Мир, 1978.

3. Oren T.I. Software for simulation of combined continuous and discrete systems. Simulation, 1977, 28, No.2, 33-45.

BASIC TECHNICAL INFORMATION ON Ada* PROJECT IN CZECHOSLOVAKIA

Jaroslav Vladík

Kancelářské stroje,k.ú.o.
28. října 15
Praha 1
Czechoslovakia

The strategy of approach to design of a compilation system for the Ada language was markedly influenced by the fact that the first version of the compilation system is envisaged for minicomputers, a typical feature of which is a limited memory size. From this point of view the method of multi-task approach was selected. That is from one Ada-program several task images (i.e. .TSK files) are generated.

At first a compilation system methodology was worked out, various degrees of dependences between the compilation units were determined and the resultant rules for obtention of data and instructional contributions to individual program sections were stated. At this stage the questions of separate compilation and the questions of sharing of variables were primarily considered [Mů183].

In view of multiple execution time increasing for the interpreted product and, apart from this, in view of a narrowing-down of the utility space for data and programs by the interpret's own body, the originally selected method of interpretation of the A-code was rejected and it was decided to generate the target code for the SMEP computers.

For this solution procedure a BELA (Back-end Language for Ada) was designed. That is intermediate code produced by the front-end of a compiler and passed on to the TCG (Target Code Generator). The front-end of a compiler contains the usual

*Ada is a registered trademark of the U.S. Government, Ada Joint Program Office

components for lexical and syntactic analysis and checking of the static semantics conditions. Static information is concentrated in compiling environment with the use of entity descriptions. First approach to solution is described in [Vla84] with the use of Ada. Aspecial form of the syntactic tree (in the sense of the AS2 Danish method of solution [Bjø80]) is passed on to the BEGE (BELA Generator) which already processes the complex of problems of dynamic semantics. In an intermediate product of the BELA [Mül83a] problems of addressing are solved a BELA product, however, still has a tree-like structure. The products of TCG are target modules (in .OBJ or .OLB form) that are processed by ADB (Ada-builder) into an arrangement which enables a execution under the DOS-RV operating system Based on structural information, ADB forms indirect command files for a TKB (Task Builder) program that is started up iteratively.

One .TSK program is built up for each task type and for the main program too. At the same time a tree of task interdependences is worked out. That serves for appropriate task activations and terminations. For each task-object an execution of the corresponding .TSK program is initialized. Other ADB functions are related to the selected approach to memory division. This approach assumes that the logical memory of an individual process does not exceed 32K words and that the available physical memory has a larger size (128K words). The approach outlined hereinafter utilizes the possibilities of mapping for the SMEP computers.

For each process it is assumed that it has its code region, a local data region (also comprising stacks) and, finally, that each process is mapped to a common region that contains two subregions.:

- SSD (Static Shared Data) in which are stored all common visible objects

- HEAP which serves as a classicial heap for meeting the demands for the dynamic store (including interprocess communication).

The size of this whole common mapped region shall be determined by ADB from the sizes of individual .TSK programs as a supplement of the longest one among them up to 32K (rounded to 4K).

The above approach to solution makes it possible, as a matter of fact, to substitute a segmentation of extensive programs by division of partial problems among individual tasks.

In connection with the memory management a heap structure was designed together with basic operations on the heap and with an elementary procedure of heap compaction. Next to this the contents of activating records, their chaining and static and dynamic links for subroutine calls were specified. For addressing in nested subroutines there was not selected a display mechanism, but a mechanism of static and dynamic chaining that enables effective addressing for depth of nesting up to two. The time delay at greater depths of nesting is not regarded as a limiting factor, as such greater nestings are not recommended.

The addressing mechanism was designed so as to enable straight-line addressability also for objects of dynamically variable size. These objects are described by their descriptors and are situated outside the activating record. An arrangement makes it possible to form addresses by means of offsets already at the time of compilation.

Particular attention was paid to the problems of paralelism for which a specification of variou forms of rendezvous, including general forms of select statement, with a regard to the means of the DOS-RV operating system was worked out [Pla84].

From this point of view too,the multi-program conception appears to be a very advantageous one, since the solution of the problems of parallelism (with the questions of waiting, suspendations etc.) is of a straight-line type, with utilization of both AST concept and a mechanism of sending of messages connected with local flags.

For the concept of exceptions a separately chained structure was designed that enables the required reactions for the exceptions raised during the elaboration of declaration parts, the exceptions raised during execution of the given unit, or exceptions raised in the exception handler.

In the I/O sphere the required functions are provided for both the text inputs and outputs (including the corresponding conversions) and for general inputs and outputs specified as sequential or direct ones.

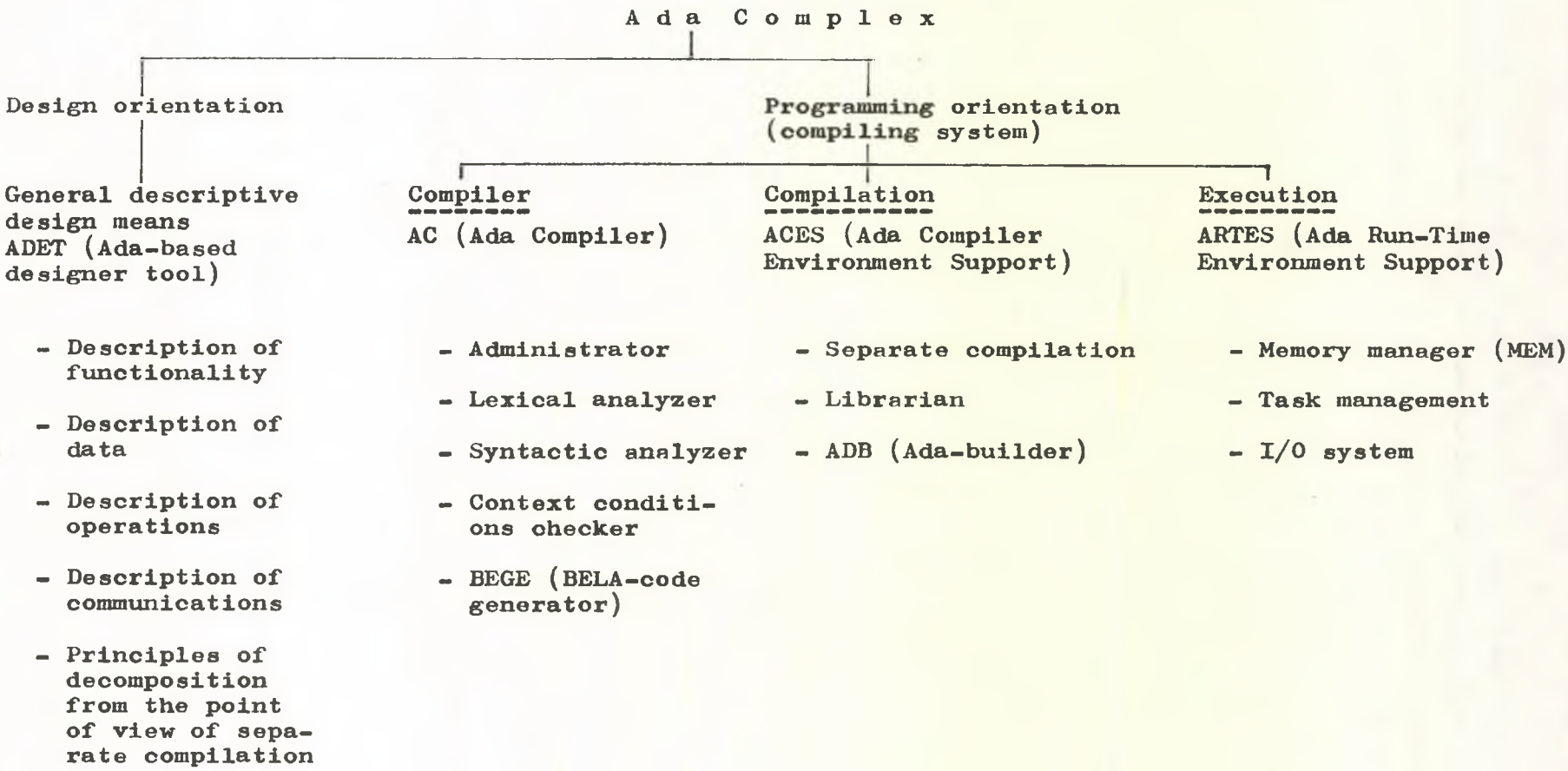
Possibilities of a supporting FCS means of the DOS-RV operating system were analyzed in order to ensure the required Ada functions. In relation to FCS default parametr values (variability of the record size etc.) and structures of FORM parameter were specified.

The Ada project started in ČSSR in 1982 on the base of preliminary studies ([Vla81]) and first version of Ada_subset compiler will appear in 1985.

In conclusion of this much abridged informative description it should be pointed out that the Ada project is in Czechoslovakia oriented not only on the problems of compilation systems, but that it considers an extension of the Ada concepts also to the sphere of system projects. Along these lines the first version of the ADET (Ada-based Designer Tool) has been designed and is at present an object of experimental vericification [Pet82]. There is endeavoured the maximum possible approximation between the means of expression used by the design and programming engineers.

A global idea of division of the whole complex of problems dealt with can be formed on hand of the enclosed diagrammatic representation.

Graphic Representation of Decomposition of Problems Considered



Literature

[Bjø80] Bjørner D., Oest O.N.: Towards a Formal Description of Ada; Lecture Notes in Computer Science, Vol. 98; 1980

[Mü183] Müller K.: Ada compiler structure; technical research report; 1983 (in Czech: Návrh kompilátoru jazyka Ada)

[Mü183] Müller K.: BELA - Back_end language for Ada; technical research report; 1983 (in Czech);

[Pet82] Petera k.: ADET - Ada_based designer tool; technical research report; 1982 (in Czech);

[Pla84] Plášil F.: Ada tasking; technical research report; 1984 (in Czech);

[Vla81] Vladík J.: Design and programming tools based on Ada definition; technical research report; 1981 (in Czech: Projekční a programovací prostředky na bázi definice jazyka Ada);

[Vla82] Vladík J.: Ada basic overview; manual; 1982 (in Czech: Základní charakteristiky jazyka Ada);

[Vla83] Vladík J.: Ada'83 textbook; manual; 1983 (in Czech: Ada'83 učební texty s hodnoceními)

[Vla84] Vladík J.: Compiling Environment; work papers; 1984 (in Czech: Kompilační prostředí)

• DIALOGUE SYSTEMS •

IMACS'84

ДИАЛОГОВАЯ СИСТЕМА ФОРМИРОВАНИЯ И ВЫБОРА РЕШЕНИЙ В УСЛОВИЯХ МНОГОКРИТЕРИАЛЬНОСТИ

Игорь И.Эрлих
СССР 103009 МОСКВА
Ул.Неждановой, 2а

I. Введение

Эффективное решение сложных задач управления разного рода операциями, экономического планирования и проектирования на современном этапе становится невозможным без использования формальных моделей и методов, среди которых все большую роль начинают играть математические модели задач выбора вариантов решения.

Природа таких задач зачастую допускает модельное представление их формализуемой части в виде задач многокритериальной оптимизации. С формальной точки зрения решение таких задач сводится к поиску лучшего или наиболее предпочтительного с точки зрения пользователя (специалиста в прикладной области, решающего задачу) решения из всего множества допустимых решений X. Множество X, а точнее, система функциональных соотношений, задающих это множество, представляет собой математическую модель вариантов исходов проведения операции, плана или проекта. С каждым допустимым вариантом $x \in X$ связан набор критериев (показателей) $y_i(x)$, $i = 1, \ldots, k$, который позволяет пользователю оценивать качество этого варианта решения. Оценка пользователем вариантов решений и соответствующих им значений показателей определяется некоторым отношением предпочтения $R \subseteq Y \times Y$ на множестве значений критериев $Y \subseteq E^k$ - образе множества X.

Любая процедура отыскания наиболее предпочтительного решения основывается на использовании информации, которая может быть

получена от пользователя в процессе решения задачи. Традиционно делаются также дополнительные предположения о свойствах самого отношения предпочтения.

Процедура выбора вариантов, применяемая в рассматриваемой диалоговой системе, использует информацию о попарных сравнениях вариантов по значениям их оценочных показателей, а для допустимого варианта, возможно, и о направлении его желательного улучшения. Само отношение предпочтения R считается непрерывным и строго выпуклым. Предполагается, кроме того, что множество X выпукло и рассматривается важный для приложений [1,3] случай линейных оценочных показателей $y_i(x)=\langle c_i, x\rangle$, $i=1,\dots,\kappa$. Используемая процедура выбора вариантов [5] основана на модификации метода эллипсоидов для многокритериальных задач, который, как известно [2], обладает хорошими оценками сходимости.

2. Схема формирования и выбора вариантов решения

Во многих прикладных задачах параметры модели, входящие в функции-ограничения, задающие множество X, по существу, можно разделить на две группы [1].

Первая группа - это неуправляемые параметры. Числовые значения неуправляемых параметров модели зависят только от конкретной реализации моделируемой задачи и обусловливаются только формализуемыми факторами. Изменение значений неуправляемых параметров по желанию пользователя, использующего модель для решения конкретной задачи, не допускаются, так как это может привести к существенному нарушению адекватности модели и задачи.

Вторая группа - управляемые параметры. Числовые значения этих параметров модели не определяются однозначно конкретной реализацией моделируемой задачи. Пользователь, использующий модель для решения конкретной задачи, может в некоторых пределах, обусловленных как формализуемыми, так и неформализуемыми факторами, выбирать на свое усмотрение значения управляемых параметров. Очевидно, что в общем случае разным значениям управляемых параметров соответствуют разные варианты решения задачи выбора.

Таким образом, множество вариантов X для каждой конкретной задачи выбора определяется с точностью до числовых значений управляемых параметров модели этой задачи, т.е. $X=X_u$ а, следовательно, и $Y=Y_u$. Здесь через u обозначен век-

тор управляемых параметров модели, принадлежащий некоторому множеству U, определяемому как формальными, так и неформальными факторами, и потому явно не заданному.

Присутствие в модели управляемых параметров $u \in U$ по существу означает, что эта модель представляет собой не одну задачу, а совокупность задач многокритериальной оптимизации. Каждому фиксированному набору допустимых значений управляемых параметров в общем случае соответствует своя задача.

Такой подход к решению задачи выбора определяет необходимость диалогового (человеко-машинного) взаимодействия пользователя с моделью решаемой им задачи. Действительно, пользователь осуществляет выбор, анализируя промежуточные варианты решения, а поиск наиболее предпочтительного варианта ведет, путем воздействия на управляемые параметры задачи и уточнения сведений о своем отношении предпочтения.

Процесс формирования и выбора варианта решения, осуществляемый с использованием рассматриваемой в работе диалоговой системы представляет собой последовательность однотипных шагов следующего содержания.

Получив очередной вариант решения, соответствующий заданным значениям параметров модели и имеющейся в системе текущей информации об отношении предпочтения, - пользователь, осуществляющий выбор, проводит анализ этого варианта. Целью анализа является, во-первых, определение соответствия рассматриваемого варианта неформализуемым факторам решаемой задачи и, во-вторых, выбор новых значений параметров модели, который, возможно, дополняется уточнением ведущейся системой модели его отношения предпочтения, необходимой для дальнейшего сужения множества допустимых решений. Если в результате проведенного анализа выясняется, что рассмотренный вариант решения в достаточной степени удовлетворяет субъективным требованиям, предъявляемым к решению пользователем, то диалоговый процесс выбора заканчивается, и данный вариант принимается в качестве решения. В противном случае система позволяет пользователю производить следующие действия:

- давать указания о своем отношении предпочтения на основе попарного сравнения вариантов по их оценочным показателям и (или) в виде направления желательного улучшения вектора-показателя допустимого варианта решения, и (или) в виде приемлемых значений отдельных характеристик вариантов;

- изменять или уточнять характеристики вариантов, значения параметров функций-ограничений, задающих множество вариантов решения, т.е. осуществлять корректировку компонент модели;

- просматривать последовательность уже полученных вариантов и своих действий с целью корректировки своих указаний.

Следует отметить, что анализ вариантов решения и корректировка компонент модели в диалоговом процессе формирования и выбора вариантов решения в конечном итоге представляет собой чисто творческий процесс, основывающийся, главным образом на знаниях, опыте и интуиции пользователя. При этом, конечно, предполагается возможность использования вспомогательных формальных методов анализа в тех случаях, когда пользователь может выразить свое отношение к рассматриваемому варианту решения в каких-либо формализуемых понятиях.

Возможности применения вспомогательных формальных методов анализа для случая, когда в качестве управляемых параметров выступают имеющиеся в распоряжении пользователя ресурсы - правые части функций-ограничений, задающих множество X_u, рассматривалась в [4]. В этой работе описан алгоритм формирования оценки первого порядка изменения управляемых параметров для класса линейных задач, который включен как вспомогательный в диалоговую систему формирования и выбора вариантов.

3. Основные компоненты системы

В состав диалоговой системы входят следующие блоки:

- блок отсева вариантов, в котором производится сужение текущего множества допустимых решений на основе локальной информации об отношении предпочтения, полученной в результате анализа пользователем очередного сформированного варианта решения;

- блок генерации вариантов, в котором осуществляется формирование для анализа пользователем допустимого варианта решения такого, что возможное сужение множества значений критериев на следующем шаге диалога было бы максимальным;

- блок решения задач выпуклого (линейного) программирования, предназначенный для построения аппроксимации множества значений критериев на основе информации о модели и характеристиках вариантов, а также для проверки сформированного варианта на допустимость;

- блок управления генерацией и отсевом вариантов, обеспечи-

вающий согласованное функционирование блоков отсева, генерации и решения задач выпуклого программирования;

- блок корректировки компонент задачи, который позволяет уточнять исходную информацию о параметрах функций-ограничений, задающих множество допустимых решений и характеристик этих вариантов, а также дает возможность пользователю формировать различные наборы показателей из имеющихся;

- блок человеко-машинного взаимодействия, который представляет собой диалоговый монитор, позволяющий на проблемно-ориентированном языке, понятном пользователю, производить ему следующие действия: а) давать системе указания о своих предпочтениях; б) осуществлять манипуляции со значениями параметров функций-ограничений, с набором и значениями характеристик вариантов, со сформированными вариантами решений;

- блок протоколирования, обеспечивающий запись действий пользователя и реакций системы и предназначенный для анализа выработки решения и контроля процесса со стороны пользователя.

В состав системы входят, кроме того, блоки информационного обеспечения:

- локальная база данных модели множества допустимых вариантов решений, содержащая информацию о функциях-ограничениях, задающих это множество, параметрах этих функций, о диапазонах изменения переменных, их количестве и т.п.;

- локальная база данных характеристик вариантов, содержащая информацию об имеющихся в распоряжении пользователя критериях, их размерностях, о функциональных или алгоритмических зависимостях от переменных;

- локальная база данных форматов отображений, содержащая сведения о возможных форматах представления информации пользователю при различных режимах работы системы;

- локальная база данных сгенерированных вариантов решения, предназначенная для накопления информации о сформированных вариантах решения.

Особенностью системы является гибкое, модульное ее построение, что дает возможность варьирования как постановок решаемых с ее помощью задач, так и режима диалогового взаимодействия. Формы подготовки исходной информации и отображения результатов также могут изменяться в зависимости от режимов работы системы.

Литература

I. Проблемы программно-целевого планирования. Под ред.Поспелова Г.С. - М.: Наука, I98I.
2. Хачиян Л.Г. Полиномиальные алгоритмы в линейном программировании. - Журнал вычислительной математики и математической физики, I980, т.20, № I.
3. Хог Э., Арора Я. Прикладное оптимальное проектирование. - М.: Мир, I983.
4. Эрлих И.И. Диалоговый процесс решения одного вида плохоформализуемых задач линейного программирования. - В сб.Математические методы оптимизации и их приложения в больших экономических и технических системах. - М.: ЦЭМИ, I980.
5. Эрлих И.И. О возможностях использования одного класса методов выпуклого программирования в диалоговых процессах планирования. - Изв.АН СССР. Техническая кибернетика, I98I, № 4.

IMYCS'84

ДИАЛОГОВАЯ СИСТЕМА РЕДАКТИРОВАНИЯ ЧЕРТЕЖЕЙ

К.П.ГОЛИКОВ
Вычислительный центр АН СССР, Москва

Диалоговая система Редграф (редактор графический) – это автономный комплекс программных средств, предназначенный для автоматизации выпуска чертежно-конструкторской документации. Система экономит время и повышает качество графических работ, избавляя конструктора от большинства рутинных повторяющихся операций.

I. Логический базис.

Диалоговый редактор работает на основе аппаратно-независимого мобильного пакета программ на Фортране. Пакет Редграф создает иерархическую структуру данных для представления чертежа в форме сегментированного файла графических элементов.

Файл состоит из 6 типов графических примитивов: отрезок, окружность, дуга, кривая, текст и маркер. Каждый примитив снабжен парой статических атрибутов: цветом и видом линии. Цвет может принимать 31 значение, вид линии 7 (тонкая, штриховая, штрихпунктирная, основная и т.д.). Кроме примитивов чертеж включает фрагменты – символы хранимых файлов графической базы данных. Любой чертеж, созданный пользователем в форме рабочего файла, может быть преобразован в более компактную хранимую форму и записан в базу данных.

Фрагмент – это ссылка на хранимый файл и совокупность параметров привязки. Привязка включает произвольную композицию аффинных преобразований (масштабирование, поворот, перенос и симметрия), а также дополнительное преобразование – фильтрацию по значениям статических атрибутов. Фильтр определяет подмножество значений атрибутов и может быть "прозрачен" для какого-то из них или "непрозрачен". Считается, что фрагмент состоит только из тех образов примитивов хранимого файла, для которых фильтр

привязки "прозрачен". Фильтрация позволяет расслаивать чертеж на независимые подмножества примитивов и выбирать только нужный слой. Например, из фрагмента можно исключить все размерные линии и текст, или все линии штриховки и т.д.

Графические примитивы и фрагменты можно объединять в группы, которые в последующих преобразованиях будут выступать как единое целое. Допускается до I5 уровней вложенности групп и фрагментов друг в друга. Редграф позволяет идентифицировать любой примитив,как независимый, так и содержащийся в группе или фрагменте, и затем извлечь его параметры для геометрических вычислений при построении новых примитивов.

На основе приведенной информационной модели в пакете Редграф определен широкий набор процедур координатного расчета и преобразования чертежа. Преобразования включают удаление и копирование отдельных графических элементов и групп, в том числе удаление и копирование участков линий между двумя точками, вытягивание линий, изменение цвета или вида линии, аффинные преобразования выделенных совокупностей элементов, экранирование произвольной многосвязной областью. Имеется набор процедур, позволяющих автоматизировать ряд наиболее трудоемких этапов создания чертажа. К ним относятся: вычисление точек пересечения линий, построение касательных и сопряжений, построение гладких контуров, в том числе лекальных кривых, эквидистант, штриховка областей. Отдельную группу составляют операторы для построения размерных линий и нанесения размерных чисел и надписей.

В число основных функций пакета Редграф входит управление архивом чертежей - графической базой данных. Чертеж помещается в базу в форме компактного хранимого файла, называемого кадром.

Логическая схема базы данных представлена однородной иерархической сетью, отражающей два типа отношений между кадрами. Отношение "содержит" связывает кадр со всеми, которые вызваны на него в качестве фрагментов. Отношение "содержится" связывает кадр со всеми, на которые он вызван в качестве фрагмента.

Аппарат привязок фрагментов (символов) существенно сокращает общий объем хранимой информации и уменьшает ее избыточность.

Модификация кадра вызывает согласованные изменения во всех чертежах, на которые он вызван.

Наличие перекрестных ссылок позволяет повысить сохранность и достоверность данных. Если кадр содержится в других кадрах, то операция удаления не выполняется, а при модификации запрещается увеличивать его размеры. В силу последнего ограничения фрагмент всегда остается в границах того чертежа, на который он вызван. Архивная система позволяет выгрузить чертеж вместе с его транзитивным замыканием по ссылкам в другую локальную базу или на магнитную ленту. Таким образом производится обмен графическими данными между разными группами разработчиков.

Наличие архивной системы позволяет связать диалоговый редактор с другими компонентами системы автоматизированного проектирования и конструирования. Например, система геометрического моделирования в трехмерном пространстве может синтезировать графический файл, содержащий совокупность плоских проекций, сечений и разрезов объекта, и поместить этот файл в базу данных. Затем, в процессе диалогового редактирования можно скомпоновать полученные полуфабрикаты на поле чертежа, нанести технологическую информацию и выполнить целый ряд других интеллектуальных операций, необходимых для преобразования геометрических проекций в условный язык конструкторского документа.

2. Диалоговый интерфейс.

Взаимодействие с оператором происходит под управлением диалогового монитора. В его функции входит: ввод управляющих директив и параметров, вызов исполняющих модулей пакета Редграф, отображение чертежа и всех его изменений на экране графического дисплея, контроль и выдача сообщений об ошибках.

Монитор построен по модульному принципу, причем все модули написаны на Фортране.

Операции ввода/вывода определены в терминах логических устройств, которые могут отображаться на различные физические устройства. Оборудование, необходимое для организации рабочего места Редграфа, включает следующие устройства:

- алфавитно-цифровой экран с клавиатурой для вывода сообщений системы и ввода чисел и текстов ;

- графический экран с высокой разрешающей способностью, желательно не менее 1024 x 1024 точек ; все сложные элементы изображения генерируются программно, поэтому от дисплея требуется только возможность построения векторов ;
- логическое устройство ввода координат (локатор) для указания точек на графическом экране ;
- логическое устройство ввода типа меню ; ядро диалогового монитора требует не менее 300 функциональных "кнопок" ;
- графопостроитель для получения чертежа на бумаге.

Возможно подключение дополнительного локатора, например, цифрового планшета для сколки уже имеющихся чертежей.

Все управляющие директивы вводятся при помощи меню. Параметры вводятся либо в числовом виде с клавиатуры, либо при помощи локатора. На базе локатора программно реализованы процедуры указания, позволяющие идентифицировать ранее построенные точки и элементы чертежа. Зона захвата указки автоматически изменяется в зависимости от масштаба отображения чертежа на графический экран.

В процессе работы чертеж разделяется на два плана - передний и задний, каждый из них представлен отдельным графическим файлом. На заднем плане накапливаются уже готовые элементы чертежа. На переднем плане выполняются вспомогательные построения, накапливаются элементы для объединения в грушу или для аффинных преобразований. В любой момент времени все содержимое переднего плана можно присоединить к заднему, или наоборот, любой элемент заднего плана извлечь на передний. Извлекать элементы можно по отдельности, простым указанием на них, или при помощи прямоугольной рамки. В последнем случае на переднем плане появляется либо все множество элементов, полностью лежащих внутри рамки, либо дополнение к этому множеству.

В процессе работы чертеж можно наблюдать целиком, но в том масштабе, который допускают размеры графического экрана, либо выбрать прямоугольное окно и разглядывать участок чертежа в любом масштабе. Окно устанавливается в произвольном месте чертежа указанием двух точек или центра. В последнем случае размеры окна автоматически выбираются так, чтобы его

содержимое отображалось на весь экран в заданном масштабе.

Кроме перечисленных, монитор использует и другие традиционные методы интерактивной машинной графики: установка дискретных сеток, "резиновые нити" и т.д.

3. Реализация.

Диалоговая система Редграф реализована на ЭВМ "Электроника I00-25" (64К оперативной памяти) в системе программирования Фортран ОС Рафос. Используется графическое рабочее место, включающее алфавитно-цифровой дисплей, графический дисплей на запоминающей трубке (I024 x I024), два цифровых планшета и графопостроитель.

Литература.

I. Голиков К.П., Педанов И.Е. Редграф- система редактирования чертежной документации, М., ВЦ АН СССР, I98I.

SPECIFICATION OF DIALOGUE SYSTEMS USING ATTRIBUTED GRAMMARS

Uwe Lämmel

Wilhelm-Pieck-Universität Rostock
Sektion Informationsverarbeitung
Albert-Einstein-Str. 21
DDR 2500 Rostock

0. Introduction

The user's view of a dialogue system is strongly influenced by its user-interface. Wether the user accepts the system or not depends on the quality of this part of the system. Instead of menus command languages are more and more used for the man-machine-communication.

Translator construction techniques can be used for describing and realizing command languages, even for graphical dialogues, because such languages are a kind of programming languages at all. Grammars of Syntactical Functions(GSF), a type of attributed grammar, define the language's syntax and semantics and a translator writing system based on these grammars supports the implementation.

Working out a standard form of a command language will be discussed within this paper.

Looking at the software-development-process characterized for instance by Green /GREEN82/ 1/ we can say our method gives some directions and aids for the specification and implementation phases. This will be explained by an example:

A user wants to design plans for parks. The plans are two-dimensional drawings with circles and rectangles denoting trees

1/ 1) requirements
2) specification
3) design
4) implementation
5) testing
6) maintenance

and houses respectively. It should be possible to place such objects into a plan or to delete them. Otherwise plans can be combined together.
Now we develop a command language for the problem solving process.

1. The command language development process

1.1 Requirements

In the first phase the problem will be analysed in order to find out what kinds of objects the user wants to manipulate. We call those object-types because real objects in this phase don't still exist.
According to data types in programming languages like PASCAL we define each objekt-type by ordered sets of object-types and use the standards: integer, real, text, point, name.
Here is the set of objekt-types for the example without mentioning the standards:

```
T = { PLAN,TREE,HOUSE,PATH,PLACE,BROAD,TOP,GROUND,POINTSEQUENCE,
      UPPERPOINT,LOWERPOINT }   ,
PLAN : NAME * UPPERPOINT * LOWERPOINT ;
TREE : NAME * PLACE * TOP ;
PATH : NAME * POINTSEQUENCE * BROAD * GROUND ;
POINTSEQUENCE : NAME * POINT , POINTSEQUENCE * POINT ;
BROAD : REAL ;
```

The notation means that for instance a triple of a name, a place, and a top define a tree.
In a second step the operations the user wants to perform on the objekts must be identified. They are written down in a similar way:

```
Q = { PUT,DELETE,PLUS }  ,
PUT : PLAN * (TREE,PATH,HOUSE) -> PLAN ;
DELETE : PLAN * (TREE,PATH,HOUSE) -> PLAN ;
PLUS : PLAN * PLAN -> PLAN ;
```

The left hand side of "-> " describes the domain and the right hand side the range of an operation. (A,B) is the union set of A and B.
The PUT operation sets a tree-, path-, or house-object into a given plan. The result is a new plan. DELETE works in the reversed way. With PLUS plans can be put together.

Elements of a third set are relations existing between different objects. In our example we define one relation: SAME-PLACE. Two objects of type house, path, or tree are in relation if there aren't any points belonging to both objects.

SAME-PLACE : (TREE,PATH,HOUSE) * (TREE,PATH,HOUSE) ;

1.2 Specification

During the specification phase a " formal description of the external behavior of the program is produced." /GREEN82/ In this sense the language description by a GSF can be the result of the specification. Some parts of the command-language-GSF are produced automatically using results of the requirements-phase. These parts contain GSF-rules describing commands for each operation and for declaring objects of some types. For an object-type T

$$T : T_{11} * T_{12} * \dots * T_{1n_1} ,$$
$$T_{21} * T_{22} * \dots * T_{2n_2} ,$$
$$\dots$$
$$T_{m1} * T_{m2} * \dots * T_{mn_m} ;$$

the following rules were produced:

OBJECT-DEFINITION(RES,"T") : 'T', T-PARM(ADR,KIND)
& T(RES,ADR,KIND).

T-PARM(ADR,j) : Tj1(ID1), Tj2(ID2), ... , Tjnj(IDnj)
& TPARMj(ADR,ID1,ID2,...,IDnj).

... , j=1..m .

OBJECT-DEFINITION, T-PARM, Tji, j=1..m, i=1..nj are nonterminals. Terminals are included in quotation marks. Semicolon separates left from right hand side of a rule. The attributes of nonterminals and may be terminals are set in parantheses. After "&" the semantic functions belonging to a rule can be found. The names of attributes are only of local interest within a rule. If T_{ji} is a union of types then a new nonterminal TPARMji(IDi, TYPEi) is built and then stands for Tji(IDi) in the rule T-PARMj(ADR,j). TYPEi became another input parameter of the semantic function TPARMj . Rules like

TPARMji(ID,"K") : K(ID) . where K is a union element will be added. Now the rules for the object-type TREE from the example:

OBJECT-DEFINITION(RES,"TREE") : 'TREE', TREE-PARM(ADR,KIND)
& TREE(RES,ADR,KIND).

```
TREE-PARM(ADR1) : PLACE(ID1),TOP(ID2) & TREEPARM1(ADR,ID1,ID2).
```

In a similar way rules describing the operations are built:

```
 PUT-COMMAND(RES) : NAME(ID), 'PUT', PUT-PARM(ADR,KIND)
                              & PUT(RES,ID,ADR,KIND).
 PUT-PARM(ADR,1) : PUTPARM11(ID1,TYPE1), PLAN(ID2)
                              & PUTPARM1(ADR,ID1,ID2,TYPE1).
```

The first part of the specification phase is the construction of these rules done automatically by a special program.
In a second step the author could do some changes in the rules to reach some more effectivity or transparency of the language.
The PUT-command could have the following definition afterwards:

```
 PUT-COMMAND(RES) : NAME(ID), 'PUT', PUTPARM(ID1,TYPE), 'INTO',
                    PLAN(ID2)  & PUT(RES,ID,ID1,ID2,TYPE).
```

The last part contains a lot to do for the author. He has to give a description of the global structure of a dialogue session. The given grammar-root USER-SYSTEM(RC) must be divided in a hierarchy of nonterminals using GSF-rules until OBJECT-DECLARATION and name-COMMAND are on the left hand side of some rules. This leads to the definition of a control-structure of the system.
Unfortunately a small example like ours can't illustrate this possibility in a sufficient way: any sequence of previously specified commands builds our problem-oriented command-language.

```
 USER-SYTEM(RC) : COMMANDSEQUENCE(RC).
 COMMANDSEQUENCE(RC) : COMMAND(RC).
 COMMANDSEQUENCE(RC) : COMMANDSEQUENCE(RC1),COMMAND(RC2)
                     &  CONNECT(RC,RC1,RC2).
 COMMAND(RC) : OBJECT-DECLARATION(RC).
 OBJECT-DECLARATION(RC) : NAME(ID),'=',OBJECT-DEFINITION(DEF,T)
                        & OBJECT(RC,ID,DEF,T).
 COMMAND(RC) : PUT-COMMAND(RC).
     ...
```

For large systems this could be a powerfull mechanism for defining system structur.

1.3 Design

This step lies a bit outside of the scope of this paper. Within the design phase the algorithms and data structures realizing

the semantic functions of the rules were developed.

1.4 Implementation

At first we look at some problem-dependent properties of the system. Every user can work with a standard set of system-commands available at any moment during the dialogue. These are commands for opening and closing a session, for simple input and output, and of course a HELP-command. Moreover a mechanism for defining new commands during the dialogue can be used. The problem-dependent language will then be added to the kernel. The command-language-description, the GSF, will be transformed by a translator writing system, called RÜGEN (Rostocker Übersetzer GENerator). The internal description of the language syntax and semantics will be produced. The interpreter needs the information to be able to process the problem-oriented language commands. Unfortunately not all the implementations can be done automatically. That's why the author has to implement procedures realizing the semantic functions. The procedures must be build in the standard form the system can handle.

2. Conclusion

We described a way to define problem-oriented command-languages - even for graphical dialogue- by using a special kind of attributed grammars (GSF). In our case we presented a method supporting the languege-development-process by the computer itself. The implementation of the command-language-interpreter is done by a translator writing system based on GSF.

3. Literature

/GREEN82/ Green,M.: A specification language and design notation for graphical user interfaces. McMaster University Hamilton, 1982, TR 81-CS-09

/LÄMMEL83/ Lämmel,U.: Attributierte Grammatiken als Hilfsmittel zur Beschreibung von Dialogsystemen. in: ALGOL'83 Tagung, TU Dresden, Schriftenreihe des WBZ/MKR , in preparation

/RIEDEWALD83/ Riedewald,G.; Maluszinski,J.; Dembinski,P.: Formale Beschreibung von Programmiersprachen. Berlin, 1983

IMYCS'84

ДИАЛОГОВАЯ СИСТЕМА ПОСТРОЕНИЯ И КОРРЕКЦИИ ДИНАМИЧЕСКИХ МОДЕЛЕЙ

Перфильев К.Г.
СССР, Москва, ВНИИ системных исследований

Соколов А.В.
СССР, Москва, ВНИИ системных исследований

I. Введение

Накопленный к настоящему времени опыт построения и анализа математических моделей сложных систем позволяет выделить следующие этапы работы: концептуальное описание системы, разработка и анализ логической структуры модели, разработка и анализ формального описания, алгоритмизация и разработка программы для ЭВМ, получение результатов и их анализ.

В настоящей работе предлагается описание диалоговой системы автоматизации процесса построения и коррекции моделей **GIN**, отражающей в той или иной мере каждый из указанных этапов исследования. При этом предполагается, что система **GIN** должна обеспечивать работу с детерминированными моделями, описываемыми обыкновенными дифференциальными уравнениями первого порядка, разрешенными относительно производной. Для такого класса моделей существуют общепринятые понятия уровня, темпа, вспомогательной переменной, временного ряда, постоянного коэффициента, которые будут далее активно использоваться.

2. Общая характеристика системы **GIN**

Структурная схема диалоговой системы построения и коррекции динамических моделей **GIN** представлена на рис. I. Здесь в агрегированном виде отражена как структура взаимодействия отдельных составляющих системы **GIN**, так и структура их взаимодействия с операционной

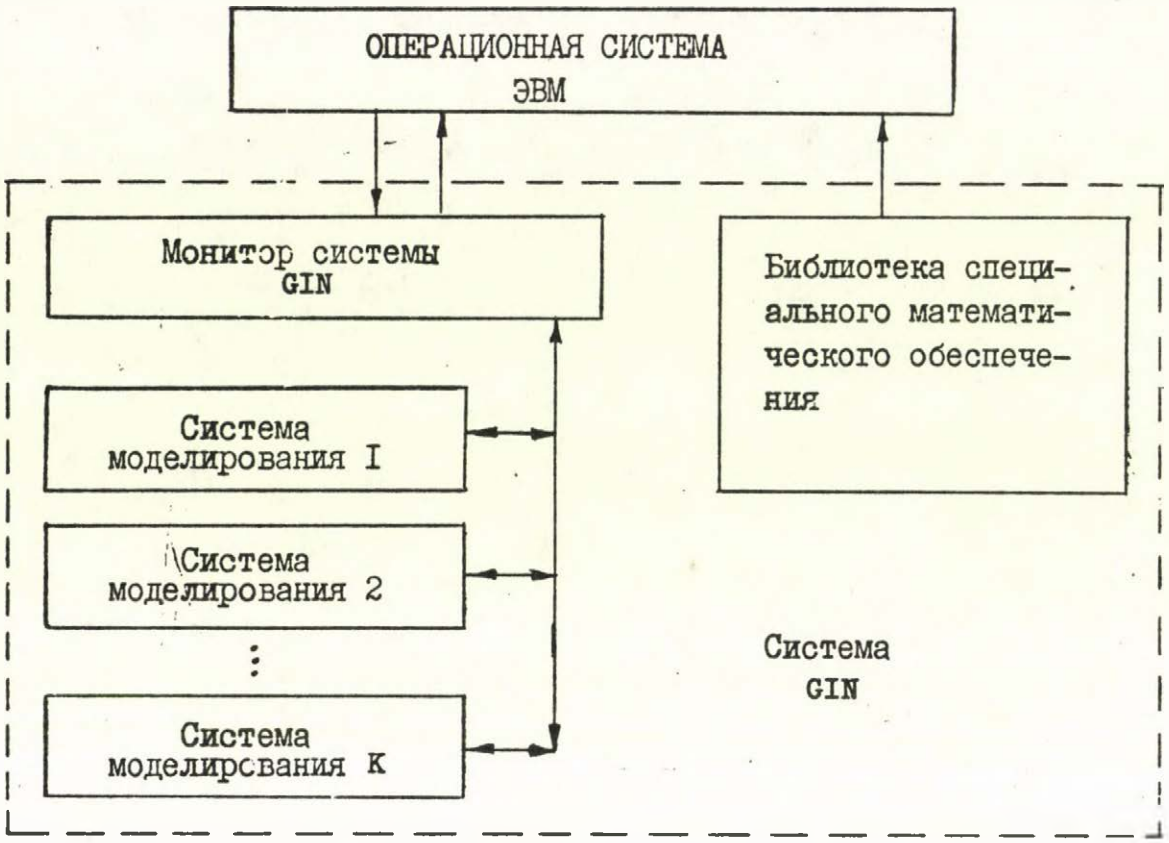


Рис. I. Структурная схема системы GIN.

системой ЭВМ. В соответствии со схемой система состоит из управляющего модуля (монитора), совокупности независящих друг от друга систем моделирования (СМ) и библиотеки специального программного обеспечения.

Монитор системы. Основная функция монитора состоит в диалоговом поэтапном управлении процессом построения или коррекции (модификации) динамической модели конкретного объекта. Кроме того, монитор предоставляет дополнительные возможности получения информации об общем состоянии системы и процесса построения, а также информации о допустимых на данном этапе командах. Взаимодействие системы с операционной системой ЭВМ осуществляется, как правило, только через монитор.

Система моделирования. Единственное назначение СМ состоит в хранении информации об объекте моделирования, как в структуризованном, так и в неструктуризованном виде. Структурная схема СМ приводится на рис. 2.

Библиотека специального программного обеспечения. Эта составляющая системы используется на этапе формирования программных модулей моделей и включает стандартные сервисные подпрограммы интегрирования системы обыкновенных дифференциальных уравнений, аппроксимации функций одной или нескольких переменных, различных форм ввода-вывода информации и т.д.

3. Принципы хранения информации в системе моделирования

Как уже отмечалось, СМ является средством хранения информации о моделируемом объекте (или объектах), вводимой пользователем в процессе диалога с монитором. Поэтому, с точки зрения функционирования всей системы GIN, СМ является единственной зависящей от пользователя частью. Зависимость эта определяется в первую очередь спецификой моделируемого объекта и интерпретацией информации о нем исследователем.

Вместе с тем это не исключает возможность реализации единого для всех СМ подхода к организации хранения необходимой информации.

Каждая система моделирования состоит из

- словарной подсистемы (хранение неструктуризованной информации), включающей словарь M имен блоков, моделей и программных модулей (каталог СМ), словарь V наименований переменных состояния и процессов моделируемого объекта, словарь R дифференциальных связей, словарь F функциональных зависимостей;

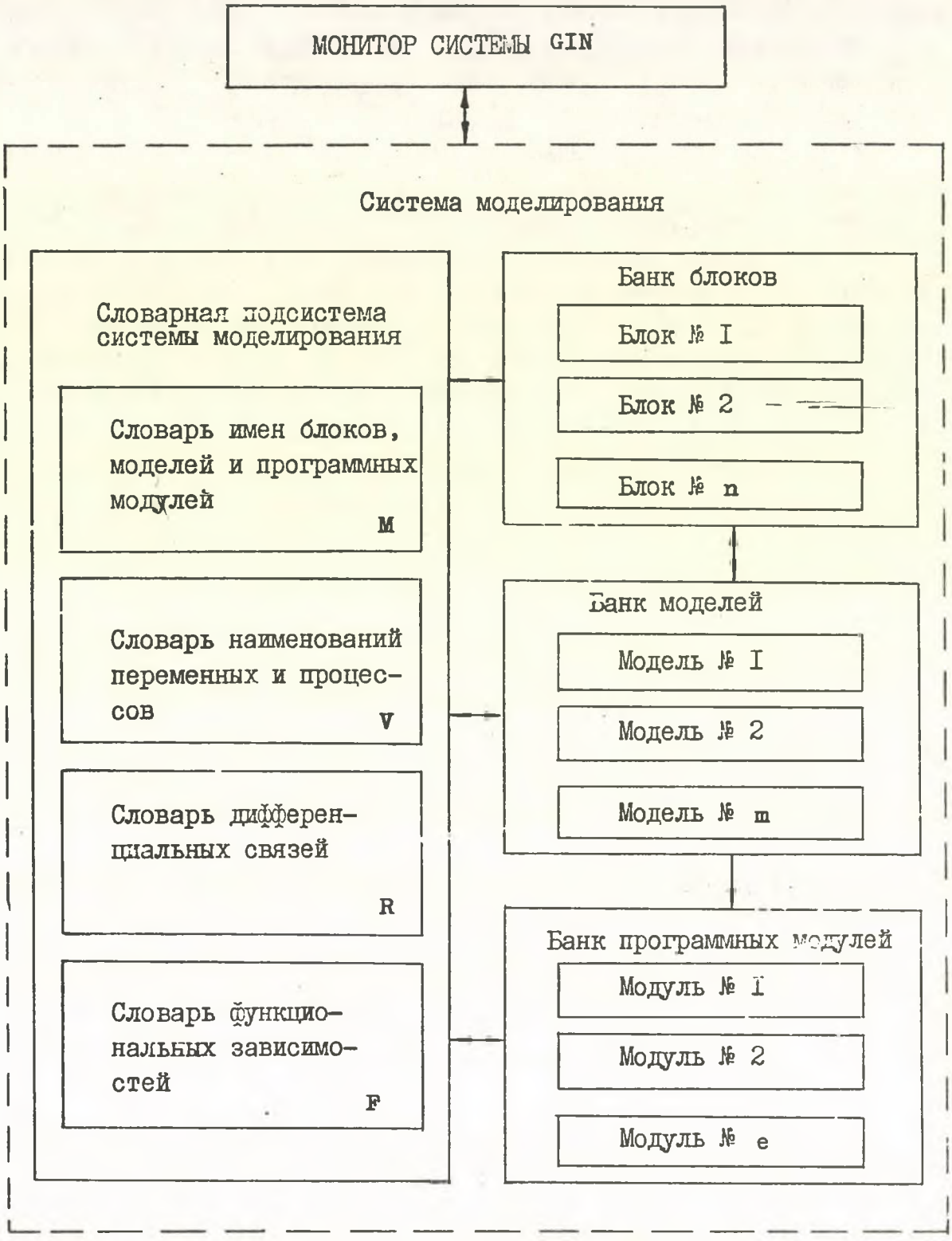


Рис. 2. Структурная схема системы моделирования.

- банка блоков моделей;
- банка моделей;
- банка программных модулей моделей.

Рассмотрим подробнее каждую из составляющих СМ.

M-словарь словарной подсистемы СМ является каталогом структурных единиц системы моделирования и содержит информацию об их имени, дате и времени создания, имени раздела в соответствующем банке.

V-словарь является каталогом терминов, используемых в системе моделирования, включая имена процессов и параметров моделей, и содержит информацию об их смысловом значении в текстовом виде. Введение этого словаря облегчает организацию диалогового режима построения и модификации моделей и позволяет пользователю постоянно "помнить" о смысловой интерпретации введенных понятий.

Третья составляющая словарной подсистемы, R-словарь, содержит информацию о структуре правых частей дифференциальных уравнений, определяющих динамику моделируемого объекта. При этом предполагается, что пользователь может указать совокупность процессов, характеризующих состояние динамической модели. Таким образом, каждая запись R-словаря содержит номер некоторой переменной, которому соответствует определенная запись в V-словаре, и номера процессов, определяющих изменение данной переменной, которым также соответствуют некоторые записи V-словаря.

Наконец, F-словарь содержит информацию о функциональных зависимостях. Каждая запись его состоит из номера определямой переменной, списка номеров переменных-аргументов (из V-словаря) и конкретного вида формулы, в которой могут быть использованы знаки элементарных арифметических операций, числовые константы и указатели элементарных функций, а также указатели функций, задаваемых таблично.

Кроме того, для удобства реализации СМ и работы с ней каждый из указанных словарей содержит вспомогательную информацию о некоторых характеристиках словарной подсистемы (в частности, их объем).

Выделение банков блоков и моделей в рамках СМ связано с традиционным подходом к моделированию сложных систем, заключающемся в разбиении моделируемого объекта на взаимосвязанные подсистемы (блоки). С этой точки зрения каждая модель представляет собой объединение некоторой совокупности блоков.

В отличие от словарной подсистемы СМ каждый раздел банка блоков содержит структуризованную информацию о некоторой подсистеме объекта моделирования. С целью избежания дублирования информации раздел блока включает в основном лишь номера записей соответствующих сло-

варей. Структура информации в разделе банка блоков базируется на разбиении всей совокупности параметров, характеризующих состояние описываемой подсистемы на множества уровневых, темповых и вспомогательных переменных, а также временных рядов и коэффициентов. Выделенные множества образуют в разделе соответствующие массивы, каждый элемент которых представляет собой ссылки на записи словарей V, R или F. Массив темповых переменных формируется на основе информации об уровневых переменных и определяющих их динамику процессах автоматически.

Банк моделей СМ по принципам своей организации аналогичен банку блоков. Каждый раздел банка моделей включает три информационных массива, содержащих имена блоков модели, постоянных коэффициентов, и временных рядов.

Наконец, банк программных модулей представляет собой средство хранения конечных результатов работы системы GIN по созданию алгоритмизированных моделей исследуемых объектов в виде программ на языке высокого уровня, а также информацию об используемых численных алгоритмах в виде обращений к подпрограммам библиотеки специального программного обеспечения.

4. Принципы функционирования системы

В соответствии с отмеченными ранее этапами построения и анализа моделей сложных объектов представляется целесообразным выделить следующие режимы работы с системой: создание новой СМ, настройка на существующую СМ, создание или коррекция какого-либо блока модели, создание или коррекция собственно модели объекта, создание программного модуля модели.

Первый из перечисленных режимов заключается в создании новой СМ для конкретного пользователя или группы пользователей и сводится, по сути дела, к формированию соответствующей словарной подсистемы из четырех пустых словарей.

Второй режим предназначен для организации взаимосвязи монитора с конкретной уже существующей СМ и является обязательным в каждом сеансе работы. Здесь же предоставляется возможность ликвидации всей СМ или некоторой части содержащейся в ней информации, необходимость сохранения которой в дальнейшем отпала.

Назначение третьего режима состоит в организации управления процессом построения и модификации каждого блока модели. В процессе построения первоначально определяется символическое имя создаваемо-

го блока и формируется соответствующая идентифицирующая запись М-словаря. После этого монитор системы обеспечивает последовательное заполнение информационных массивов раздела банка блоков, предоставляя одновременно возможность просмотра и (при необходимости) пополнения словарей словарной подсистемы СМ.

Для удобства контроля и предоставления исследователю возможности осуществления процедуры построения на протяжении нескольких сеансов работы с системой GIN в разделе банка блоков предусмотрены специальные средства хранения сведений о степени завершенности заполнения информационных массивов.

Процесс коррекции заключается во внесении изменений в какие-либо из информационных массивов. В свою очередь эти изменения могут потребовать дальнейшей модификации блока для обеспечения замкнутости описания, что и осуществляется монитором в диалоговом режиме принудительно.

Процедуры построения и коррекции моделей в рамках системы GIN в целом аналогичны соответствующим процедурам работы с блоками и сводятся к последовательному формированию необходимых информационных массивов. После завершения их формирования (или модификации) система автоматически проводит анализ корректности и замкнутости описания всей модели. В частности, осуществляется проверка возможности последовательного вычисления всех вспомогательных и темповых переменных и устанавливается порядок этих вычислений.

Наконец, последний режим работы системы GIN, осуществляемый в основном автоматически, служит для построения программного модуля модели, который в дальнейшем совместно с библиотекой специального программного обеспечения используется для формирования загрузочного модуля программы модели вне системы GIN средствами операционной системы ЭВМ.

Литература

I. Перфильев К.Г., Соколов А.В. GIN - диалоговая система построения и коррекции моделей.- В сб.: Моделирование процессов экологического развития. Вып. 2.- Москва: ВНИИСИ, 1982, с.116-123.
2. Форрестер Дж. Мировая динамика.- Москва: Наука, 1978.

РЕАЛИЗАЦИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ НА ПЕРСОНАЛЬНОМ КОМПЬЮТЕРЕ

Т.В.Ускова

ВЦ АН СССР
ул.Вавилова 40
Москва, СССР

I. Рассматривается метод реализации диалоговой информационной системы (ДИС) на персональном компьютере, соединенном линией связи с большой ЭВМ. Описываемый подход характеризуется следующим:

а) Наличием специальных средств, дающих возможность создателям информационной системы конструировать схему диалога в соответствии с требованиями пользователей.

б) Полным отделением процесса создания ДИС от её использования. Пользователь при этом полностью избавлен от программирования в обычном смысле, используя простые управляющие средства для получения нужной информации из базы данных.

2. Взаимодействие человека и информационной системы определяется схемой диалога, описываемой формально как ориентированный граф $\Gamma = (S, T)$, где S - множество состояний диалога, а T - множество переходов между состояниями. Состояния могут быть нескольких типов: меню /М/, запрос значения /А/, промежуточное. В описываемой системе используются два типа меню:

- простое позиционное /$М_р$/ и
- позиционное с запросом значения /$М_а$/.

В обоих случаях на экран выводится видеообъект, состоящий из:

I) графических элементов
2) текста, накладываемого на графическое изображение
3) курсора, перемещаемого под воздействием управляющих клавиш по фиксированной траектории.

В состоянии $М_р$ выбор пункта меню производится с помощью специальных клавиш, подведением курсора в соответствующую позицию на экране, после чего нажимается клавиша "Исполнение" и система осуществляет заранее запрограммированные действия.

В состоянии $М_а$ курсор подводится к нужному пункту меню и вводится запрашиваемое текстовое или числовое значение, которое затем обрабатывается соответствующей прикладной программой. Так, например, ДИС, предназначенная для поддержки научно-организационной работы содержит информацию о :

I) комиссиях
2) исследованиях
3) учреждениях
4) ведомствах
5) оборудовании

Для получения необходимой информации используются оба вида меню: $М_р$, $М_а$. В частности, для получения списка учреждений используется меню типа $М_р$. После выбора в главном меню позиции "Учреждения" на экране возникает список учреждений, который может просто изучаться пользователем или служит основой для дальнейших операций. Чтобы получить список членов какой-либо комиссии, используется меню типа $М_а$. Система при этом запрашивает номер комиссии, который должен быть введен пользователем в соответствующей позиции на экране, после чего прикладная программа извлечет из базы данных список всех членов, указанной комиссии и выведет его на экран.

3. При создании вышеописанной системы отдельно выполняются следующие этапы:

а) формируется база данных
б) создается схема диалога
в) составляются прикладные программы, ассоциируемые со схемой диалога и служащие для извлечения информации из базы данных.

База данных представляет собой совокупность файлов, каждый из которых содержит информацию об объектах одного типа, например, учреждениях, исследованиях, оборудовании и т.д. Ввод данных осуществляется простыми средствами текстового редактирования.

Другой этап создания ДИС, формирование схемы диалога, осуществляется с помощью "Конструктора диалоговых систем" . Эта

программа позволяет разбивать экранные изображения графа Г на фрагменты $Г_i \subset Г$, в которых фиксируются отдельные связанные, группы состояний, их атрибуты, переходы между состояниями. Атрибуты задают действия, выполняемые в каждом состоянии, такие, как вывод видеообъекта, запуск прикладной программы и т.д.

Видеообъекты строятся специальной программой "Видеоредактор". Эта программа позволяет создавать изображения, состоящие из произвольных линий, элементарных геометрических фигур, цвета, текста. Эти средства построения схем диалога позволяют осуществить их модификацию и расширение, приспосабливая к конкретным потребностям пользователей.

4. Следует отметить, что база данных ДИС может быть как угодно большой, поскольку часть информации может находиться в большой ЭВМ. С помощью специальной программы, связывающей персональный компьютер с большой ЭВМ, осуществляется доступ к базе данных в соответствующих состояниях диалога.

В описываемой системе в большой ЭВМ хранятся данные о членах комиссий (адреса, телефоны, должности), учреждениях (адреса, телексы, фамилии руководителей) и др.

5. Отделение этапа конструирования диалога от функциональной и содержательной части ДИС обеспечивает эффективную настройку на требования пользователей, освобождает прикладные программы от функции управления графикой, текстового редактирования и других универсальных операций. ДИС, реализованная на персональном компьютере, является новым эффективным средством информационного обслуживания.

ЛИТЕРАТУРА

I. В.М.Брябрин, "Адаптивный диалог - основа персональной вычислительной системы" в сб.: Лингвистические процессы и представление знаний, ВЦ СО АН СССР, 1981, с. 22-40.

2. Г.В.Сенин, О.А.Гончаров, "Средства организации диалога в экспертных системах на микро-ЭВМ", Материалы 4-го Всесоюзного симпозиума: Системное и теоретическое программирование, Кишенев, 1983, с. 340-341.

3. Т.В.Ускова, "Фреймовый подход к машинному представлению словарных энциклопедических статей", Материалы 2-го Международного семинара по машинному переводу, Москва, 1979.

IMACS'84

ДИАЛОГОВОЕ МАТЕМАТИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ

Викторов Л.П.
СССР, Москва, ВНИИ системных исследований

Одно из современных направлений в математическом обеспечении имитационного моделирования - создание систем управления моделью. Имитационные исследования требуют многократного прогона модели с различными значениями начальных данных и сценарных переменных, определяющих тот или иной рассматриваемый вариант. Необходимые для этого средства управления моделью и изменения значений управляющих переменных на ранних этапах развития имитационого моделирования встраивались в язык моделирования. Таким образом, помимо собственно программы модели заранее описывались так же и варианты моделирования. С появлением и все большим распространением интерактивного режима использования ЭВМ стало ясно, что функции задания вариантов следует отделить от описания модели и вынести в отдельную часть - диалоговый монитор. В этом случае пользователь просчитывает различные варианты в интерактивном режиме, непосредственно видит их результаты, может легко оценить различные стратегии. Основные функции диалогового монитора включают управление моделью, взаимодействие с терминалом, задание управляющих воздействий, сбор и вывод результатов.

В докладе рассматриваются основные концепции и возможности монитора для интерактивного моделирования (МИМ) [I], разработанного для построения и исследования имитационных моделей на ЭВМ. Особенно эффективно использование монитора в задачах моделирования сложных систем, таких как социально-экономические системы. Монитор дает возможность проводить работы с моделью в диалоговом режиме как создателям модели, так и специалистам, не имеющим опыта использования ЭВМ. Диалоговый монитор МИМ может использоваться как самостоятельная система, а так же служить основой для построения проблемно-ори-

ентированных систем моделирования.

Диалоговый монитор МИМ применим к широкому классу имитационных моделей, написанных на Фортране. Модель рассматривается в общем виде, как оператор перехода от состояния моделируемого объекта в момент T1 к состоянию в момент T2, т.е.

$$Y(T2) = MY(T1),$$

где $Y(T)$ - вектор переменных модели в момент T.

При таком подходе допустимо использование монитора для различных моделей, в том числе описываемых системами функциональных, конечно-разностных и дифференциальных уравнений. В мониторе не накладывается ограничений на математические методы, применяемые в модели, например на способы интегрирования. Диалоговый монитор обеспечивает работу как со всей моделью, так и с различными комбинациями подмоделей.

В процессе разработки модели исследователь может отлаживать программу модели с помощью разработанного автором и включенного в состав МИМа символьного интерактивного отладчика [2], который позволяет в диалоговом режиме устанавливать точки прерывания, распечатывать и изменять значения переменных. Вся работа с отладчиком ведется в терминах Фортрана, т.е. указываются имена подпрограмм, номера операторов, имена переменных, а не их адреса.

Во время работы можно вводить дополнительные переменные, значения которых вычисляются в процессе моделирования по определяющим их арифметическим выражениям. Эту возможность удобно использовать для интерактивного внесения изменений в модель, добавления формул, получения дополнительных результатов (например, вычисления суммарного значения какой-либо величины на всем временном интервале моделирования).

Много внимания уделено представлению результатов моделирования, реализованы алгоритмы накопления, сохранения и получения информации. Монитор обеспечивает вывод как в табличном, так и в графическом виде на дисплей, графопостроитель и АЦПУ. Для сравнения альтернативных вариантов предусмотрен одновременный вывод результатов этих вариантов на одном графике или таблице. Для графического вывода можно задать различные режим интерполяции и масштабирования.

Реализован двухуровневый диалог : для специалистов, имеющих опыт работы с МИМом и начинающих пользователей. Во втором случае активная роль принадлежит монитору, исследователь выбирает вид работы из предлагаемого меню или вводит спрашиваемый параметр. Взаимодействие с начинающими пользователями реализовано на специально

разработанном языке управления диалогом; опытные пользователи могут, используя этот язык, изменять структуру диалога.

Опыт работы с моделями сложных систем, таких как социально-экономические показал необходимость специального аппарата формирования сценариев. Целесообразным является предварительное создание "заготовок" сценариев, описывающих в общих чертах какой-либо вариант. Во время работы с моделью, в диалоговом режиме, исследователь выбирает интересующую его заготовку сценария, возможно переопределяет ряд параметров, не изменяющих суть сценария, и просчитывает сформированный вариант. Блок формирования сценариев в МИМе допускает структуризацию сценариев и предоставляет различные способы задания управлений, в том числе графический и в виде функций от времени и переменных модели.

Остановимся теперь на основных принципах реализации диалогового монитора на ЭВМ. С самого начала разработки было решено сделать МИМ как можно более машиннонезависимым. Поэтому, основным языком программирования был выбран Фортран, трансляторы с которого существуют на всех современных ЭВМ.

Базовая ЭВМ, для которой разрабатывался МИМ - это мини-ЭВМ PDP-11/70. Самым существенным ограничением для сложных систем на любой мини-ЭВМ является ограничение на память задачи (для ЭВМ PDP-11/70 - 64 Кб.) Чтобы обойти это ограничение, система была реализована в виде комплекса задач. Основная задача - собственно МИМ управляет работой подзадач. В число подзадач входит задача-модель, а так же блоки МИМа, требующие для исполнения много памяти и слабо связанные по параметрам с остальными блоками - подзадачи формирования сценариев и графического вывода. Разработана стандартная процедура для подключения подзадач, следовательно, таким способом можно на основе МИМа как центрального управляющего блока строить проблемно-ориентированные системы моделирования. Заметим, что подзадачи могут быть написаны на любом языке, поддерживаемом операционной системой.

Важным принципом реализации является включение в состав МИМа отладочных средств. Во-первых, символьный интерактивный отладчик может использоваться не только для отладки модели, но и для отладки самого диалогового монитора. Во-вторых, в МИМ встроены средства для сбора внутренней информации о работе монитора. В процессе сеанса работы с монитором можно включать и отключать сбор информации о работе блоков синтаксического анализа, запоминания результатов, интерпретатора арифметических выражений и программ межзадачной связи.

Введение отладочных средств значительно упрощает важную и трудную задачу поддержки программных систем в процессе эксплуатации.

Изложенные принципы обеспечивают в большой степени переносимость диалогового монитора. При переносе на ЭВМ без существенных ограничений на память задачи вся система моделирования может быть реализована в виде одной задачи. Такая возможность учитывалась при разработке МИМа; модули, обеспечивающие межзадачную связь, легко могут быть исключены. Встроенные отладочные средства так же оказывают неоценимую помощь при переносе системы на другие ЭВМ.

Более чем пятилетний опыт эксплуатации МИМа показал, что его использование значительно облегчает процесс исследования моделей; особенно эффективно его применение при исследовании сложных систем. Удобные для анализа формы представления результатов и ориентация диалога на разный уровень пользователей дали возможность применять МИМ для задач обучения и демонстрации моделей различных социально-экономических систем.

ЛИТЕРАТУРА

I. Викторов Л.П. МИМ - Монитор для Интерактивного Моделирования. Препринт. М., ВНИИСИ, 1980

2. Викторов Л.П. Система отладки фортрановских программ для PDP-11/70. Препринт. М., ВНИИСИ, 1980

• MISCELLANEOUS •

IMYCS'84

ПОСТРОЕНИЕ НЕКОТОРЫХ ИНВАРИАНТНЫХ КЛАССОВ И ИХ СЛОЖНОСТЬ

Л.Г. Асатрян

СССР г.Ереван-44, ул.П.Севака I,

ВЦ АН Арм.ССР

I. Введение

Задача синтеза управляющих систем [2] является одной из основных задач кибернетики. Для многих задач в этой области важно не только построить устройство, написать программу и т.д., но еще и добиться экономичности этих объектов. В частности широко известна задача построения минимальных схем (в классе контактных схем или булевых формул, или схем из функциональных элементов) реализующих булевы функции. О.Б.Лупановым [I] было доказано, что имеет место эффект Шеннона: почти все функции из P_2^n (булевы функции зависящие от n переменных) имеют асимптотически одинаковую сложность. Эта сложность асимптотически наихудшая. Поэтому, и потому, что на практике возникают не все булевы функции, острой открытой проблемой является выделение подклассов класса булевых функций и исследование их сложностей . Широкий класс таких функций (инвариантные классы) в параметризованном виде описал С.В.Яблонский [3] и нашел асимптотическую сложность ненулевых инвариантных классов (инвариантный класс Q_σ называется ненулевым, если $\sigma \neq 0$, где параметр σ определяется из соотношения $\sqrt[2^n]{P_{Q_\sigma}(n)} \longrightarrow 2^\sigma$, $0 \leq \sigma \leq 1$, а $P_{Q_\sigma}(n)$ - число функций из Q_σ , зависящих от n переменных) в классе контакных схем. Но для всех практически интересных инвариантных классов Q_σ - $\sigma = 0$. Поэтому остается открытым вопрос о том, как возрастает функция $P_{Q_\sigma}(n)$ в случае $\sigma = 0$ и, в зависимости от этого, какова сложность данного инвариантного класса например в классе схем из функциональных элементов. В этой работе

описывается способ, с помощью которого строятся нулевые инвариантные классы и находится их сложность в классе схем из функциональных элементов.

2. Формулировка основных результатов.

Обозначим через P_2 множество всех булевых функций.

Определение. Множество функций $Q \subseteq P_2$ называется инвариантным классом, если, наряду с каждой функцией $f \in Q$, оно содержит все функции, получающиеся из f применением следующих трех операций:

1) добавление и изъятие фиктивных переменных,

2) переименование переменных (без отождествления),

3) подстановка констант на места некоторых переменных.

Основными результатами данной работы являются

Теорема 1. Для любой монотонно неубывающей целочисленной функции $\Psi(n)$, удовлетворяющей следующим трем соотношениям

1) существует натуральное число n' такое, что $\Psi(n') < n'$,

2) $\dfrac{\min \Psi^{-1}(k)}{k} \geqslant \dfrac{\min \Psi^{-1}(k-1)}{k-1}$,

3) $n = o\left(2^{\Psi(n)}\right)$,

можно построить инвариантный класс Q такой, что $\log P_Q(n) \sim 2^{\Psi(n)}$.

Инвариантные классы Q с $\log P_Q(n) \sim 2^{\Psi(n)}$, о которых говорится в теореме 1 назовем инвариантными классами типа $\langle \Psi(n) \rangle$.

В теореме 1 через $\Psi^{-1}(k)$ обозначается следующее множество $\Psi^{-1}(k) = \{n / \Psi(n) = k\}$.

Теорема 2. Сложность инвариантного класса Q типа $\langle \Psi(n) \rangle$ в классе схем из функциональных элементов асимптотически равна $\dfrac{2^{\Psi(n)}}{\Psi(n)}$.

Литература

1 Лупанов О.Б., О синтезе некоторых классов управляющих систем, Сб. "Проблемы кибернетики", вып. 10, М., 1963, 63 - 97.

2 Яблонский С.В., Основные понятия кибернетики, Сб. "Проблемы кибернетики", вып. 2, М., 1959, 7 - 38.

3 Яблонский С.В., Об алгоритмических трудностях синтеза минимальных контактных схем, Сб."Проблемы кибернетики", вып.2, М., 1959, 73 - 121.

IMYCS'84

ИССЛЕДОВАНИЕ ВОЗМОЖНОСТЕЙ ГЛОБАЛЬНОЙ ОПТИМИЗАЦИИ ПРОГРАММ НА УРОВНЕ ВХОДНОГО ЯЗЫКА

Ходулёв А.Б. ИПМ им. М.В.Келдыша АН СССР,
Москва, А-47, Миусская пл., 4

I. Введение

В настоящем докладе излагаются результаты работы по созданию и исследованию глобального оптимизатора Фортран-программ на уровне входного языка. Предыстория этой работы такова.

В ИПМ АН СССР был разработан и реализован оптимизирующий транслятор Форекс с языка Фортран-77. Транслятор Форекс выполняет квазилокальную оптимизацию транслируемой программы. Участком оптимизации в трансляторе Форекс является фрагмент программной компоненты, не содержащий внутри себя меток и циклов. В частности, участком оптимизации может быть целиком внутренний цикл, если его тело не содержит меток; участок оптимизации может включать разветвления, задаваемые условными инструкциями. Специально проведенные исследования /I/ и опыт эксплуатации транслятора Форекс показали высокое качество порождаемых им программ.

Дальнейшее повышение качества программы возможно при глобальной оптимизации, когда участком оптимизации является вся программа. Для выяснения величины дополнительного эффекта, который может достигаться сверх того, что обеспечивает транслятор Форекс, была предпринята реализация экспериментального глобального оптимизатора Глоп. Он выполняет оптимизацию программы на уровне входного языка, т.е. выдает оптимизированную программу на том же языке, что и входная программа - на Фортране. Такое решение, обеспечивая разделение фаз оптимизации и трансляции, упрощает реализацию оптимизатора и транслятора в сравнении со случаем оптимизирующей трансляции. Кроме того, оптимизатор на уровне входного языка может использоваться совместно с любым транслятором, воспринимающим стандартный Фортран, хотя при разработке оптимизатора Глоп выбор оптимизирующих преобразований

делался в расчете на использование транслятора Форекс.

Оптимизатор Глоп выполняет только автоматическую оптимизацию программы, т.е. он не воспринимает от пользователя какую-либо дополнительную информацию о свойствах программы. Участком оптимизации в оптимизаторе Глоп является целиком одна программная компонента.

2. Выполняемые преобразования и структура оптимизатора

Перечислим оптимизирующие преобразования, выполняемые оптимизатором Глоп.

I. Действия с константами. Это преобразование состоит в замене выражения с константными операндами на его значение, вычисляемое оптимизатором. Оптимизатор Глоп выполняет действия с константами в арифметических и логических выражениях, а также в условных инструкциях. Последнее означает удаление неисполняемых ветвей условных инструкций с константным (тождественно истинным или тождественно ложным) условием. В качестве константных операндов рассматриваются не только явно заданные в программе константы, но и переменные, значение которых может быть вычислено оптимизатором.

2. Вынесение из цикла инвариантных вычислений. Выражение, операндами которого являются константы или переменные, не изменяемые в цикле, называется инвариантным в данном цикле. Вычисление таких выражений переносится оптимизатором из тела в пролог цикла (отдельный линейный участок, вставляемый перед входом в цикл). В пролог цикла помещаются инструкции присваивания значений этих выражений вспомогательным переменным, создаваемым оптимизатором, а в теле цикла использование вынесенного выражения заменяется на использование этой временной переменной. Данному преобразованию подвергаются все циклы с одним входом, как заданные с помощью инструкции цикла, так и заданные неявно, с помощью условных переходов.

3. Экономия общих подвыражений. Два вхождения некоторого выражения считаются общими, если они принадлежат одному линейному участку и между ними нет инструкций, изменяющих операнды выражения, или если они принадлежат двум линейным участкам, один из которых доминирует над другим в управляющем графе программы и на всех путях, связывающих эти два вхождения, нет

инструкций, изменяющих операнды выражения. Преобразование состоит в том, что перед первым вхождением общего выражения помещается инструкция присваивания его значения некоторой вспомогательной переменной, а вхождения выражения заменяются на эту переменную.

4. Удаление неактивных присваиваний. Присваивание некоторой переменной называется неактивным, если на всех путях от этого присваивания до выхода из программной компоненты значение переменной не используется до ее переопределения (при этом считается, что значения элементов общих блоков и формальных параметров используются в выходных точках программной компоненты). Неактивные присваивания удаляются.
5. Удаление недостижимых частей программы.
6. Удаление неиспользуемых меток. Это преобразование приводит к улучшению программы за счет укрупнения участков оптимизации транслятора Форекс, что позволяет транслятору выполнять более полную оптимизацию.
7. Переупорядочение программы с целью уменьшения числа безусловных переходов.
8. Удаление несущественных разветвлений, т.е. таких условных инструкций, все ветви которых передают управление в одну точку программы.
9. Замена (если это возможно) арифметических условных инструкций на логические. Это преобразование приводит к появлению неиспользуемых меток и увеличивает таким образом возможность выполнения преобразования 6.

Работа оптимизатора Глоп состоит из трех последовательно выполняемых фаз:

I. Синтаксический анализ программы и перевод ее во внутреннее представление
2. Оптимизирующие преобразования внутреннего представления
3. Генерация выходного текста.

Фаза оптимизации, в свою очередь, включает выполнение четырех независимых программ, реализующих преобразования I-4.

Экспериментальный характер описываемой разработки предполагает возможность управления оптимизацией. Для исследования эффекта отдельных преобразований имеется возможность отключать любые из этих программ или менять порядок их выполнения, в том числе выполнять какие-либо преобразования многократно. Указания

по управлению оптимизирующими преобразованиями записываются в виде комментариев специального вида в тексте исходной программы.

Преобразования 5-9 выполняются на фазах синтаксического анализа и генерации, а также параллельно с преобразованиями I-4. Пользователь имеет возможность только отключать выполнение преобразований 8,9; выполнение преобразований 5-7 не контролируется пользователем.

В некоторых случаях имеется возможность более тонкого управления. В частности, для преобразования экономии общих подвыражений можно отдельно отключить локальную экономию (т.е. экономию в пределах одного линейного участка).

Реализация оптимизатора Глоп подробно описана в /2/.

Критерием качества программы при оптимизации в данной работе принято, как обычно, время ее работы. Оно, конечно, зависит от применяемго после оптимизации транслятора и ЭВМ, на которой будет выполняться программа. При разработке оптимизатора Глоп выбор преобразований основывался на предположении об использовании транслятора Форекс для БЭСМ-6. Однако, лишь два преобразования (6 и 9) существенно опираются на специфику транслятора Форекс. Остальные преобразования, как можно предполагать, будут оптимизирующими и для других трансляторов. В любом случае можно отключить те преобразования, которые не будут улучшать программу.

3. Оценка эффективности оптимизирующих преобразований

Для измерения эффективности какого-либо оптимизирующего преобразования (i) или их группы следует оттранслировать некоторую программу (p) в двух вариантах - исходном и оптимизированном, причем при оптимизации следует включить только исследуемые преобразования. ВВедя обозначения $t_u(p)$ и $t_{0,i}(p)$ для времен работы, соответственно, исходной и оптимизированной программы, используем для оценки эффективности преобразования относительное изменение времени счета

$$e_i(p) = \frac{t_u(p) - t_{0,i}(p)}{t_u(p)}$$

Величина $e_i(p)$ может сильно меняться в зависимости от программы p. Естественно поэтому усреднить $e_i(p)$ по программам p некоторого класса, использовав в качестве оценки среднее значение $\bar{e}_i$.

Следующий важный вопрос - выбор программ для исследования. при оценке производительности ЭВМ и качества трансляторов часто используют синтетические тесты - специально составленные программы, включающие в нужной пропорции конструкции, используемые в реальных программах. При оценке глобального оптимизатора такой подход неприменим, поскольку глобальный оптимизатор учитывает далекие связи в программе и, следовательно, размер конструкций, из которых должен был бы состоять синтетический тест, оказывается весьма большим. Общее число таких конструкций будет чрезмерно велико, что не позволит обеспечить представительность теста. Единственная разумная возможность- исследование эффективности оптимизирующих преобразований непосредственно на реальных программах.

Для проведения такого исследования несколько реальных программ были обработаны оптимизатором Глоп с включением различных оптимизирующих преобразований. Полученные программы были оттранслированы транслятором Форекс. Сравнение времен выполнения исходной и оптимизированных программ показало низкую эффективность выполняемых оптимизирующих преобразований, величина $e_i(p)$ не превышала 10%. Более точные оценки не удалось получить из-за невысокой точности измерения времени ЦП на БЭСМ-6.

Для сравнения эффективности оптимизирующих преобразований была использована оценка их эффективности, основанная на статических характеристиках оттранслированных программ. Предположим, что

1) Время выполнения линейного участка программы пропорционально его длине в оттранслированной программе (числу команд)
2) Возможность выполнения какого-либо преобразования в каком-либо месте программы не зависит от частоты выполнения этого места.

Эти предположения, вероятно, приближенно выполняются для большинства программ и всех перечисленных выше оптимизирующих преобразований, за исключением преобразований 2 (вынесение из цикла) и 5 (удаление недостижимых частей). Из этих предположений легко выводится, что среднее по программам значение эффективности e_i совпадает со средним относительным уменьшением длины командной части программы в результате i-го преобразования, которое мы обозначим $\tilde{e}_i$. Преобразование 5 не влияет на время выполнения программы, а для преобразования вынесения из цикла в качестве величины $\tilde{e}_i$, приближающей e_i, использовалось среднее

относительное уменьшение длины внутренних циклов программы. Полученные таким образом оценки эффективности оптимизирующих преобразований приведены в таблице. В первом столбце указаны номера выполняемых преобразований, во втором - величина $\tilde{e}_i$ для этих преобразований в процентах, в третьем - максимальная по программам величина $\tilde{e}_i(\rho)$.

Таблица. Эффективность оптимизирующих преобразований

Преобр.	$\tilde{e}_i(\rho)$, проценты	
	сред.	макс.
6,7	0.5	2.5
8,9	2	8
I	3	8
2	0.5	I
3	0	0
4	0.2	0.6
8,9,I,4	6	I8

Полученные результаты позволяют сделать вывод о том, что в настоящее время глобальную оптимизацию программ на уровне входного языка нецелесообразно применять к реальным производственным программам, составленным вручную.

Литература

I. Вик.С. Штаркман. Сравнительный анализ транслятора FOREX. Препр. ИПМ АН СССР, I978г., №I29.

2. А.Б. Ходулев. Реализация глобального оптимизатора Фортран-программ. Препр. ИПМ АН СССР, I982 г., № 92.

IMYCS'84

THE COMBINATIONAL COMPLEXITY OF THE SYMMETRIC FUNCTIONS WITH SMALL/GREAT WORK NUMBERS

Milan Ftáčnik

Department of Theoretical Cybernetics
Faculty of Mathematics and Physics
Komenský University
842 15 Bratislava
Czechoslovakia

1. A Boolean function $f: E^n \longrightarrow E$ (where E denotes a set $\{0,1\}$) is symmetric if for all $c \in E^n$, $c = (c_1, c_2, \ldots, c_n)$ and for all permutations π of $\{1, 2, \ldots, n\}$

$$f(c_1, c_2, \ldots, c_n) = f(c_{\pi(1)}, c_{\pi(2)}, \ldots, c_{\pi(n)}). \qquad /1/$$

A Boolean function is symmetric if and only if there exist $w_0, w_1, \ldots, w_n \in E$ and for all $c \in E^n$

$$f(c_1, c_2, \ldots, c_n) = w_j \qquad /2/$$

where $j = \#\{i \mid c_i = 1\}$ and $\#S$ denotes the cardinality of set S. The symmetric function can be expanded as follows

$$f(x_1, x_2, \ldots, x_n) = w_0 \cdot \sigma_0(x_1, \ldots, x_n) + \ldots + \sigma_n(x_1, \ldots, x_n)$$

where $\cdot$ and $+$ denotes AND and OR and $\sigma_t: E^n \longrightarrow E$ is an elementary symmetric function

$$\sigma_t(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i = t \\ 0 & \text{otherwise} \end{cases} \qquad /3/$$

where $\sum$ denotes integer addition.

So symmetric function has the same value on all binary n-tuples containing the same number of 1's, that is, which have the same weight. The weights of those n-tuples on which the function has value 1 are called work numbers. They fully determine the

symmetric function. $S_N^{a_1,a_2,a_3}$ will denote the symmetric function $f: E^N \to E$ with working numbers a_1, a_2, a_3 i.e. the function for which only coordinates w_{a_1}, w_{a_2}, w_{a_3} have the value 1.

The combinational complexity of symmetric functions was analysed several times. In paper [2] it has been shown, that in base $\Omega_2 = \{\text{all Boolean functions of two arguments}\}$ is the upper bound of complexity (the asterisk denotes the evaluation without NOT-elements)

$$C^*_{\Omega_2}(S_n) \leqslant 8{,}5\, n . \qquad /4/$$

It can be shown [1] that in base $\Omega_0 = \{\text{AND, OR, NOT}\}$ holds

$$C_{\Omega_0}(S_n) \leqslant 12{,}5\, n . \qquad /5/$$

2. The aim of this paper is to present a method the use of which brings better upper bounds in base Ω_0 than /5/ for the symmetric functions with small or great work numbers. The definitions related to combinational complexity may be found e.g. in [2]. In all statements we omit the sign of the base that is Ω_0.

The method will be illustrated at first on the example of the symmetric function S_n^0, i.e. elementary symmetric function $\sigma_0(x_1, \ldots, x_n)$. The task of the recognition of zero-tuple among entry tuples is realised by the local encoding: two coordinates of n-tuple are composed in that way that the result of the composition is 0 if both were zero coordinates and 1 otherwise. The n/2-tuple of the results is obtained. By repeating this process, one-tuple is obtained which equals 0 if and only if all the coordinates of the entry n-tuple were zero. The requirements on the composition function are fulfilled by the logical sum.

<u>Theorem 1.</u> $n - 1 \leqslant C(S_n^0) < 2n$. /6/

Proof: Let $n = 2^\lambda$ ($\lambda = 1, 2, \ldots$). The scheme for S_n^0 in this case is constructed in accordance with the idea presented above (Fig. 1.). The NOT-element secures the right value of the function with respect to the result of recognition.

The scheme contains $n - 1$ OR-elements and one NOT-element. The complexity of the scheme is then equal to n.

In case $2^\lambda < n < 2^{\lambda+1}$ ($\lambda = 1, 2, \ldots$), the entry tuple is completed on the length $2^{\lambda+1}$ by constant zeros (this can be done with a complexity not greater than 2). Then the scheme is construc-

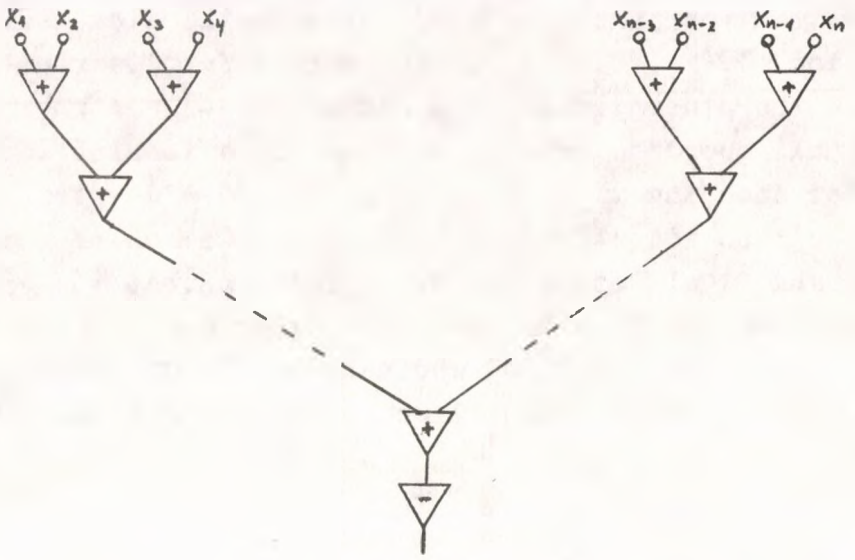


Fig. 1.

ted in accordance with Fig. 1. For its complexity there holds

$$C(S_n^0) < 2^{\lambda+1}(1 - 1/2^{\lambda}) + 2 + 1 = 2^{\lambda+1} + 1 < 2n .$$

Because S_n^0 essentially depends on all its variables, the natural lower bound holds for it.

Note 1: In the case $2^{\lambda} < n < 2^{\lambda+1}$ another construction is also possible. If n is even, the construction from Theorem 1 can be used in the first level. If n is odd, the entry n-tuple is completed on $n+1$-tuple by one constant zero and an "even" construction is used. In other levels if the tuple obtained is of even length, the construction is also "even", in case of the odd length there are two possibilities: a/ a single coordinate remains which cannot be composed with no other b/ a coordinate remains which can be composed with the analogy from the upper levels. The complexity of the scheme does not exceed $n + \log n$, but on the other hand the scheme depends on the concrete n and is not regular.

The scheme for S_n^n which is in some sense dual to S_n^0 is constructed analogically. Each OR in it is replaced by AND and each constant 0 by 1. It holds

$$C(S_n^n) < 2n \qquad /7/$$

and

$$C(S_n^n) = n - 1 \quad \text{if} \quad n = 2^{\lambda} .$$

Let us now take the function S_n^1, which is the elementary symmetric function $\sigma_1(x_1, \ldots, x_n)$. The scheme for its realisation is based on the generalised principle of the scheme for S_n^0. The number of unit coordinates of the entry tuple is also locally controlled. For encoding two bits are needed: 00 - denotes the absence of the unit coordinate, 10 or 01 - the presence of one such coordinate and 11 all other cases. It follows that the composition function cannot be a logical sum (defined coordinate by coordinate) because e.g. 01 OR 01 would be again 01, which contradicts the encoding. The composition function can be defined e.g. in this way

$$z_1 = x_1 \vee y_1 \vee (x_2 \wedge y_2)$$
$$z_2 = x_2 \vee y_2 \vee (x_1 \wedge y_1) \qquad /8/$$

where $z_1 z_2$ is the result and $x_1 x_2$ and $y_1 y_2$ are the 2-tuples composed.

<u>Theorem 2.</u> $2n - 3 \leqslant C(S_n^1) < 6n - 6$. /9/

Proof: Let us assume $n = 2^\lambda$ ($\lambda = 2, 3, \ldots$). If the function defined by equations /8/ is denoted by ○, then the scheme for S_n^1 is in Fig. 2. The last block of the scheme decodes the result of recognition to secure the correct value of the function.

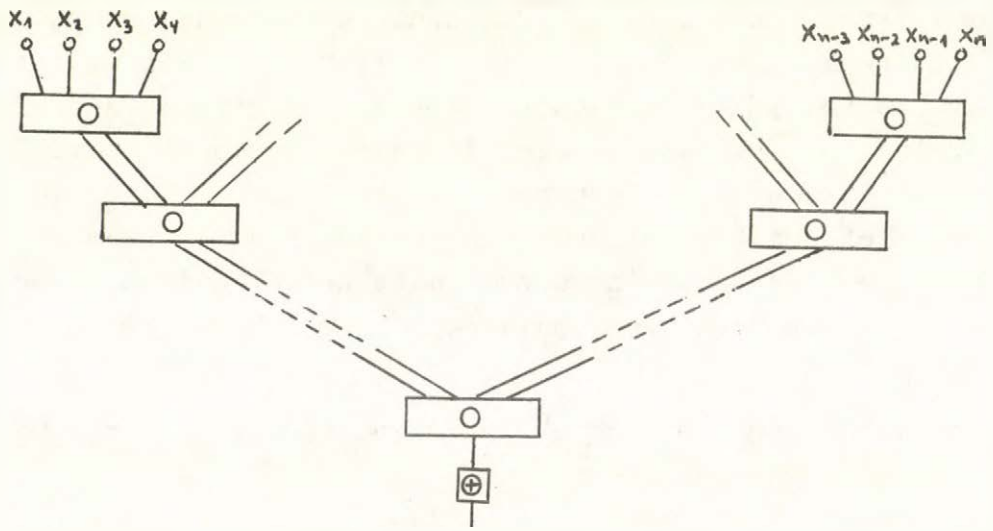


Fig. 2.

The number of blocks ○ in the scheme is

$n/4 + n/8 + \ldots + n/2^\lambda = n/2 - 1$.

According to /8/ every block $\bigcirc$ can be expressed by a subscheme containing 6 elements of the base. The complexity of the last block of the scheme is 4, so it holds

$$C(S_n^1) = 6(n/2 - 1) + 4 = 3n - 2 .$$

If $2^\lambda < n < 2^{\lambda+1}$, the entry tuple is again completed on the length $2^{\lambda+1}$ and the scheme is constructed in accordance with Fig. 2. In this case

$$C(S_n^1) < 6(2^{\lambda+1}(1/2 - 1/2^\lambda)) + 2 + 4 = 6(2^\lambda - 2) + 6 < 6n - 6 .$$

The lower bound is the consequence of the following theorem which was proved by Stockmeyer [3] .

Theorem 3. Let n and k be nonnegative integers and let us assume that the word $w_0w_1...w_n$ defining the symmetric function f has a form

$$E^{\geq k} . W . E^{\geq k}$$

where $.$ denotes the concatenation of the words, $E^{\geq k} = \{w \in E |$ the length of w greater than or equal to $k\}$ and W is a subset of E^4 containing three different subwords of length 2, that is $W = \{0100, 0010, 0110, 0011, 1101, 1001, 1011, 1100\}$.

Then

$$C_{\Omega_2}(f) \geq 2n + k - 3 . \qquad /10/$$

Corollary 1. For $i = 1, 2, ..., n-1$ and $j = \min\{i-1, n-i-1\}$ it holds

$$C_{\Omega_0}(S_n^i) \geq 2n + j - 3 . \qquad /11/$$

Proof: Every function S_n^i may be defined by the word $w_0w_1...w_n$ which has the form

$$\{0\}^{\geq j} . \{0100, 0010\} . \{0\}^{\geq j}$$

and obviously

$$C_{\Omega_0} \geq C_{\Omega_2} .$$

Analogical evaluations hold for S_n^{n-1} . The lower bound follows from corollary 1 and the upper bound from the scheme dual to that for S_n^1 .

Theorem 4. $2n - 2 \leqslant C(S_n^2) < 10n$. /12/

Proof: From the generalisation of the proofs of the theorems 1 and 2 it follows that in this case three coordinates are composed. The encoding is again natural: 000 denotes the absence of 1, 001, 010, 100 - the presence of one 1, 011, 101, 110 - the presence of two 1 and 111 denotes all other cases. The composition function over 3-tuples may be defined by following equations

$$z_1 = x_1 \vee y_1 \vee (x_2 \wedge y_2) \vee ((x_3 \wedge y_3) \wedge (x_2 \vee y_2))$$
$$z_2 = x_2 \vee y_2 \vee (x_3 \wedge y_3) \vee ((x_1 \wedge y_1) \wedge (x_3 \vee y_3)) \qquad /13/$$
$$z_3 = x_3 \vee y_3 \vee (x_1 \wedge y_1) \vee ((x_2 \wedge y_2) \wedge (x_1 \vee y_1))$$

This function may then be expressed by a subscheme containing 15 elements of the base.

Again two cases are distinguished. The first for $n = 3.2^{\lambda}$ ($\lambda = 1, 2, \ldots$), the second for $3.2^{\lambda} < n < 3.2^{\lambda+1}$. The scheme in the first case is constructed in accordance with the method described - its complexity is less than $5n$. In the second case the entry tuple is completed by zeros on the length $3.2^{\lambda+1}$. The last block - the decoder is a little more complicated (it may be constructed with the complexity less than 9).

Note 2: In the proofs of the theorems 1, 2 and 4 there was proved, besides the affirmations, that if for the symmetric functions S_n^i ($i = 1,2,3$) the condition $n = (i+1).2^{\lambda}$ ($\lambda = 1, 2, \ldots$) is fulfilled then the combinational complexity is approximately half of the case when it is not. It can be seen from the Note 1. that this "half" complexity can be reached asymptotically. An analogical note holds for S_n^i ($i = n, n-1, n-2$) and the condition $n = (n-i+1).2^{\lambda}$.

3. The method for obtaining the upper bounds for the symmetric functions with small/great work numbers was presented. The substance of the method is following: for i the greatest (the smallest in dual case) work number of the symmetric function is chosen and the condition $n = (i+1).2^{\lambda}$ ($n = (n-i+1).2^{\lambda}$) is controlled. The construction of the appertaining $i+1$ ($n-i+1$) equations is determined by the composition considered i.g. by remembering the unit coordinates - actually the shift of "double" unit to the left. From this follows the cyclic character of the

containing all easier cases for smaller (greater) i plus the specific term while the complexity is fast increasing with regard to the number of combinations at longer shift. It can be seen that the way of encoding enables to construct also the schemes for the symmetric functions with the combinations of work numbers e.g. $S_n^{0,1}$, $S_n^{1,2}$, $S_n^{n,n-1}$, $S_n^{n-1,n-2}$ etc.

I want to thank Doc. RNDr. I. Haverlík, CSc. for useful remarks to this work.

References

[1] Ftáčnik M., Haverlík I. : The Combinational Complexity of the Symmetric Functions in Different Bases, (submitted in Mathematica Slovaka 1984)

[2] Savage J.E.: The Complexity of Computing, Wiley Interscience New York, 1976

[3] Stockmeyer L.J.: On the Combinational Complexity of Certain Symmetric Boolean Functions, Mathematical Systems Theory, 10 (1977), 323 - 336

АЛГОРИТМИЧЕСКИЙ МЕТОД ИДЕНТИФИКАЦИИ ФАКТОРНО-НЕСТАЦИОНАРНЫХ ОБЪЕКТОВ

Каримов Ш.Т.
СССР, г.Ташкент, УзНПО "Кибернетика"
Салимов Н.Р.
СССР, г.Ташкент, ТашСХИ

Существует класс объектов, в которых всевозможные факторы $\Omega_T=\{\Omega_{T_1}, \Omega_{T_2}, \dots, \Omega_{T_m}\}$ и априорные информации $\mathcal{J}_T=\{K_T, E_T, M_T, L_T, \dots\}$ (например: $M_T=\{M_{T_1}, M_{T_2}, \dots\}$ - спектров структуры моделей $L_T=\{L_{T_1}, L_{T_2}, \dots\}$ - алгоритмы адаптации параметров модели для соответствующих моделей) нестационарные. Необходимо произвести целенаправленную группировку факторов $\Omega=\Omega(\Omega_T, \mathcal{J}_T)$, $\Omega=\{\Omega_1, \Omega_2, \dots, \Omega_n\}$ и тем самым упростить задачи $\mathcal{J}=\mathcal{J}(\Omega, \Omega_T, \mathcal{J}_T)$, $\mathcal{J}=\{K, E, M, L, \dots\}$, $n \leq m$: структурной и параметрической идентификации, адаптации параметров моделей и решение вопросов оптимизации без ущерба точности оптимального управления.

При исследовании факторно-нестационарных объектов системным анализом получили три типа квазистационарной области: $t_o \leq t \leq t_o+t_y$, Ω, $\mathcal{J}_T=const$, $t_o \leq t \leq t_o+t_M$, $\Omega^* \in \Omega$, $\mathcal{J}^* \in \mathcal{J}=const$, $t_o \leq t \leq t+t_n$, $A \leq const$, $M^* \in M$, $M^*=M(A)$,

где: t - время, t_o - начальный момент времени, t_o, t_M, t_n - интервал времени квазистационарных областей, $t_y \geq t_M \geq t_A$, A - параметры моделей, * - показаны выбранные из множеств характеристик, по определенным критериям оптимальности $C=\{C_1, C_2, \dots\}$ единственная наиболее подходящая.

Для решения этой проблемы было разработано следующее:

- создание автоматизированных информационных систем (АИС) существенно облегчит формализацию этой задачи. Информации $\mathcal{J}_T$, Ω_T, накапливаемые в информационной системе, представляют собой модель объекта некоторой области времени;

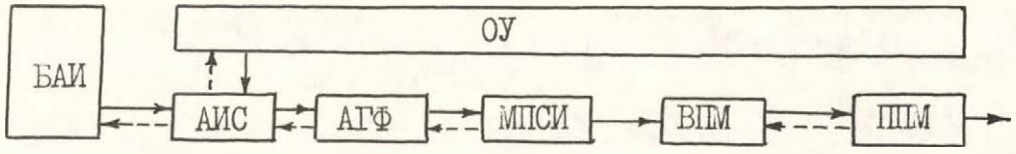
- алгоритм группировки факторов (АГФ) $\Omega=\Omega(\Omega_T, \mathcal{J}_T)$ разработан на основе нормального алгоритма Маркова;

- на основе $\mathcal{J}_\tau$, Ω_τ, Ω и используя разработанный матричный подход структурной идентификации (МПСИ) получаем спектр моделей $M \in \mathcal{J}^*$.

- на основе теории выбора и принятия решения по множеству критериев оптимальности C , осуществляем подбор $M^* \in M$ наиболее подходящую структуру модели (ВПМ) для данной области времени;

- для поиска параметров моделей (ППМ) (решение обратной задачи) с учетом нестационарности объекта управления, используем методы решения экстремальных задач и метод регуляризации Тихонова, где: функция регуляризации $\|a\|$, $A=\{a\}$; проверку модели на адекватность производили по критерию близости $\rho < y^э$, $y^\rho > \xi$ где: $y^э$, y^ρ - экспериментальные и расчетные выходные переменные, ξ - малое число.

Блок-схема принципа функционирования алгоритмического способа построения математических моделей факторно-нестационарных объектов приведена на рис.I.



- информация; - - - - запрос

Рис.I

В АИС поступают информации с объекта управления (ОУ) и из банка априорной информации (БАИ) по запросу, формирования запросов зависит от конкретной ситуации и требования моделей. БАИ обновляется в течении времени, поэтому является динамичным. Это диктует обновлению в свою очередь АИС.

В исследуемом классе химических нестационарных объектов, где участвуют сложные реакционные смеси, от полноты анализа априорной информации зависит решение общих проблем идентификации и оптимизации.

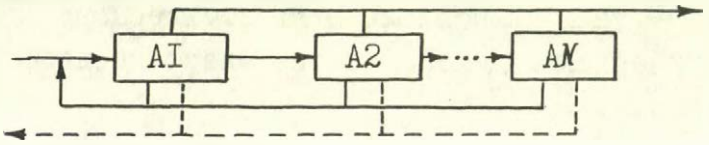
Так как состав сырья сложен, существует огромный объем информации, поэтому ее сбор и хранение - важный и трудоемкий этап в поиске структуры моделей. От удачности наглядности, полноты сбора и хранения информации зависят дальнейшие этапы решения за-

дачи идентификации.

При этом совокупность данных описывается как (иерархические файлы, наборы, древовидные) или как двумерный (плоский) файл, каждая запись которого имеет одинаковый набор полей и поэтому файл может быть представлен в виде двумерной матрицы.

Если порожденный элемент имеет более одного исходного элемента, то это отношение уже нельзя описать как древовидную или иерархическую структуру. Оно описывается в виде сетевой структуры, где любой элемент может быть связан с любым другим.

Необходимые данные поступают из АИС в АГФ. АГФ является результатом ограничений связи множества алгоритмов по определенным характеристикам объекта идентификации [1]. Алгоритмы имеют один вход и четыре выхода. Выходы: результат, запрос, реакция, продолжения. Блок-схема АГФ приведена на рис.2.



A1+ A2 ... AN - алгоритмы

Рис.2.

АГФ для исследуемого класса объектов состоит из четырех алгоритмов, связанных со следующими характеристиками процессов: химическим свойствам, трудоемкостью анализа, групп веществ,точности описания процесса, кинетическим данным.

Рассмотрим один из алгоритмов группировки веществ по химическим свойствам. Проверим все группы веществ по неравенству $Q(3, L8) > Q_{зад}$, $L8 = \overline{1,N8}$, где $Q_{зад}$ - заданный минимальный вес; - матрица химических свойств групп веществ.

Сверху вниз приведем семантический разбор дерева (Q) для исключения листьев и ветвей, не используемых в математических моделях, поместим номер листьев каждой $L10$ -й группы веществ в множества $\{q(L10)\}$, $L10 = \overline{1,N1}$.

МПСИ состоит из семи алгоритмов:

В существующей априорной информации о протекании реакции выделяются основные условия процесса (факторы).

Факторы в зависимости от исследуемого объекта могут влиять по разному. Поэтому принимаем экспертную процедуру для принятия решений.

Для ее решения используется экспертиза $Э$, где эксперты изолированы; обратная связь отсутствует: $V(L2) = \sum_{L3=1}^{N3} Э(L2,L3)/N3$, где $V(L2)$ - весовой коэффициент $L2$ -го фактора; $Э(L2,L3)$ - экспертной оценки $L3$ -го эксперта на $L2$ -му факту; $L3 = \overline{1,N3}$; $L2 = \overline{1,N2}$; $L4 = \overline{1,M}$;

$N2$ - всевозможные факторы, $N3$ - эксперты, M - основные факторы.

Алгоритм дает возможность выделить основные факторы и их весовые коэффициенты.

Алгоритм 2.

Выделяем и упорядочиваем по уменьшению веса весовых коэффициентов факторов по неравенству: $V_{min} < V(L2)$, $L2 = \overline{1,N2}$. Выделим M факторов и образуем вектор $OV(M)$ на основе технологического эксперимента.

Из априорной информации образуем O матрицу данных, где по столбцам - факторы, строкам - элементарные реакции. Полученная матрица имеет размер NN, M . Если априорная информация не существует, тогда элемент матрицы остается открытым.

На основе этих (O, OV) матриц образуем матрицу $A(NN, M)$ следующим образом: элементы каждой строки матрицы OV сравниваем с соответствующими элементами матрицы OV . Если они равны в соответствующей матрицы A записывается 0, если нет, то I.

Алгоритм 3.

$A1$ - вектор-столбец, показывающий протекание или не протекание реакции. Если реакция протекает, соответствующий его номеру элемент $A1$ = I, если нет, то 0.

Проведем пофакторный анализ протекания реакции по формуле $AP(L5,L4) = |A(L5,L4) \cdot V(L4) - A1(L5)|$, где $L5$ - номер элементарной реакции $L5 = \overline{1,NN}$, $L4 = \overline{1,M}$.

На основе полученных результатов проведем анализ протекания реакций по элементарным реакциям на основе формул:

$$PA(L5) = \begin{cases} \min\limits_{1 \le L4 \le M} |AP(L5,L4) - 0{,}5| + 0{,}5 , & \text{если } A1 = I, \\ \min |AP(L5,L4) - 0{,}5| - 0{,}5 , & \text{если } A1 = 0 \end{cases}$$

Для удобства использования вектор-столбец PA приведем к виду матрицы с размерами $N \times N$ и поместим в матрицу НР.

Алгоритм 4, где $Q(N)$ - вектор весовых характеристик всех групп веществ, подвергающихся весовому анализу.

Образуем матрицу $M(N,N)$ по формуле:

$$M(L6,L7)=\begin{cases} 0, & \text{если } HP(L6,L7)=0, \\ Q(L6) & \text{если } HP(L6,L7)\neq 0,\ L6<L7, \\ Q(L7) & \text{если } HP(L6,L7)\neq 0,\ L6>L7. \end{cases}$$

Алгоритм 5, где $Q(2,N)$ значения количества расходуемых и образуемых веществ, полученные на основе материального баланса с действующего объекта.

Проверим соблюдение материального баланса в матрице M .

Для этого должны выполняться условия

$$\sum_{L7=1}^{N} M(L6,L7) > Q2(1,L6), \quad a)$$

$$\sum_{L6=1}^{N} M(L6,L7) > Q2(2,L7). \quad \delta)$$

Если для какого-то a или δ условия не выполняются, тогда необходимо проверить априорную информацию, соответствующую $L6$ - строке или $L7$ -му столбцу.

Алгоритм 6, где PH - вероятность протекания реакций для группирования веществ определенной по формуле:

$$PH(L10,L11)\equiv \frac{\sum_{L6=1}^{N}\sum_{L7=1}^{N} M(L6,L7)\,HP(L6,L7)}{\sum_{L6=1}^{N}\sum_{L7=1}^{N} M(L6,L7)}$$

$$L6\in\{q(L10)\},\ L7\in\{q(L11)\}$$

$$L6\in\{q(L10)\},\ L7\in\{q(L10)\},\quad L10,L11=\overline{1,N1}.$$

Алгоритм 7. Входят матрицы $PH(N1,N1)$. На основе вероятностной матрицы РН образуем ряд (спектров) моделей. Для данного числа группированных веществ. Первая модель является упро-

щенной и по возрастанию номера постепенно усложняется. Определим максимальный элемент из матрицы РН и помещаем номер элемента в множество $\mathcal{Z}1$, т.е. определяем $max\{PH(l10, l11)\}$ полученный элемент проверяем, если $PH(l10,l11) < \xi$, $l11, l10 = \overline{1,N1}$; $l10, l11 \notin \mathcal{Z}1 \cup \mathcal{Z}$, поиск механизма реакции заканчивается, если нет проводим проверку материального баланса:

$$\sum_{l11=1}^{N1} M(l10,l11) > Q2(1,l11),\ l10,l11 \in \mathcal{Z} \cup \mathcal{Z}1,\ l10 = \overline{1,N1},$$

$$\sum_{l10=1}^{N1} M(l10,l11) > Q1(1,l10),\ l10,l11 \in \mathcal{Z} \cup \mathcal{Z}1,\ l11 = \overline{1,N1},$$

если условия (IO) не выполняются, тогда ($l10, l11$) заносится в множество $\mathcal{Z}1$ и процедура продолжается, если оба условия выполняются $(l10, l11)$ заносится в множество $\mathcal{Z}$ и процесс продолжается.

Далее полученные спектры моделей на основе ВПМ обрабатывается с целью выделения наиболее вероятной модели для данного момента времени.

Усложнение модели процесса приведет к более точному описанию исследуемого объекта, но при этом возникают следующие трудности:

- увеличивается группа веществ, подвергающих весовому анализу;
- усложняется алгоритм адаптации параметров модели;
- повышается частота съема данных из объекта управления;
- усложняется алгоритм оптимизации процесса.

Таким образом возникает задача. Задача принятия решения $< VM, ОП >$, где: VM - множество вариантов структур моделей; $ОП$ - принцип оптимальности.

Решением задачи $< VM, ОП >$ является множество $VM_{оп} \leq VM$, полученное с помощью принципа оптимальности ОП.

Оптимальный вариант структуры модели определяется по формуле $VM_{оп} = max\, VM(V)$, $1 \leq V \leq N12$.

Выбранная наиболее подходящая модель для данного момента времени подвергается иерархической идентификации на основе экспериментальных данных.

Параметрическая идентификация производится в квазистационарной области ($t_0 \leq t \leq t_0 + t_1$) при установившемся режиме

($dy/dt=0$) с использованием методов регуляризации.

Таким образом вышерассмотренные теоретические аспекты предлагаемой работы были использованы при построении конкретной математической модели гетерогенно-каталитического процесса гидрирования ксилозы, которая представлена в следующем виде:

$$\frac{dY_{\psi,f,q}}{d\tau_q} = \sum_{j=1}^{n} KC_{\psi,j,f,q}\, Y_{j,f,q} - \sum_{i=1}^{n} KC_{i,\psi,f,q}\, Y_{\psi,f,q},$$

$$KC_{i,j,f,q} = k_{i,j,q} \exp(-E_{i,j}/RT_{f,q})$$

$$k_{i,j,q} = k^{v}_{i,j,q} F_{i,j,q}(X_{1,q},\dots,X_{p,q},U_{1,q},\dots,U_{y,q}),$$

$$i,j,\psi = \overline{1,n}; \quad f = \overline{1,m}; \quad q = 1,2,\dots$$

где обозначены матрицы: Y - концентраций группированных веществ; X,U - входных и управляющих переменных; KC - констант скоростей реакций; k - предэкспоненциальных множителей; T - температур; E - энергии активации; F - переходных функций, а также: τ - вектор времени контактов ($\tau_q = v_q/e$) ; e - линейная координата реактора ($0 \le e \le L$); L - общая длина реактора; v - вектор линейных скоростей реакционных смесей в реакторе; R - газовая постоянная; n - число сгруппированных веществ; m - число секционных обогревателей; q - номер опыта; i,j - показывают, что j-й группы веществ получается i-я.

Применения разработанного алгоритмического метода идентификации факторно-нестационарных объектов сокращает время исследований примерно на 18-20%, что дает соответствующий экономический эффект.

Л И Т Е Р А Т У Р А

1. В.К.Кабулов. Алгоритмизация в механике сплошных сред. Ташкент, изд-во "Фан" УзССР, 1979, с.304.

2. В.К.Кабулов. Алгоритмическое направление в кибернетике и пакеты прикладных программ. "Изв.АН УзССР, серия техн.наук", 1976, № 6, с.3-10.

3.К.А.Ахметов. Докторская диссертация. Ташкент,1982,с.303.

ПЛАНИРОВАНИЯ ВЫЧИСЛЕНИЙ В РАСЧЕТНО-ЛОГИЧЕСКИХ СИСТЕМАХ

С. Р. Родин

ВЦ АН СССР

ул.Вавилова 40

Москва,СССР

В настоящее время разнообразные прикладные системы, предназначенные для решения различных инженерно-конструкторских задач, разрабатываются в виде пакетов программ (ПП), которые, по-существу, становятся современным способом организации проблемно-ориентированного математического обеспечения ЭВМ.

Благодаря своей сложной внутренней структуре, ПП позволяют решать задачи, сформулированные пользователем в своих содержательных терминах. Такая возможность, предоставляемая пакетам, позволяет существенно снизить процедурную нагрузку и, тем самым, существенно расширить круг потенциальных пользователей,позволяя им при решении своих задач неявно использовать знания из области прикладной математики и программирования, заложенные в ПП при его создании.

Работа пользователя с пакетом протекает следующим образом. Пользователь на своём профессионально-ограниченном языке описывает интересующую его проблемную ситуацию (модель), а затем исследует её, определяя значения одних атрибутов модели и интересуясь значениями других её атрибутов (ставит задачи на модели). Алгоритм решения задачи синтезируется Планировщиком пакета, который сводит поставленную пользователем задачу к решению ряда взаимосвязанных типовых для ПП задач, решаемых модулями функционального и методно-ориентированного наполнения пакета, а затем строит программу, реализующую этот алгоритм. Полученная программа выполняется Исполнительной подсистемой пакета, которая организует вычислительный процесс решения задачи.

Важной характеристикой пакета является мощность класса решаемых им задач, которая в значительной степени зависит от мощности методов синтеза алгоритмов решения.

Рассмотрим проблемы, возникающие при синтезе алгоритма решения задач расчёта и оптимизации стандартных режимов функционирования технических систем.

В качестве модели технической системы будем рассматривать вычислительную модель $\mathfrak{M}=(O,A)$, которая представляет собой систему отношений $O=(o^1,\dots,o^n)$, связанных общими атрибутами $A=(a_1,\dots,a_m)$. Эта модель формируется пакетом по данному пользователем описанию морфологии и функционального назначения системы на основе методов переформулирования, заложенных в ПП.

При решении задач расчёта стационарных режимов часто можно считать, что отношения $o^i(a^i), i=\overline{1,n}$ являются однородными, т.е. число атрибутов, значение которых может быть вычислено из такого отношения, не зависит от разбиения множества его атрибутов a^i на входные и выходные. При этом число выходных атрибутов называется рангом r_i однородного отношения o^i.

Если считать, что все отношения модели функционально независимы, то естественно потребовать, чтобы любая подсистема отношений была структурно-непротиворечивой, т.е. чтобы для любого подмножества отношений $O'\subseteq O$ выполнялось неравенство $\sum_{I_{O'}} r_i \leqslant \left|\bigcup_{I_{O'}} a^i\right|$, где $I_{O'}=\{i \mid o^i \in O'\}$. Это условие гарантирует, что ранг любой подсистемы отношений $O'\subseteq O$ равен суммарному рангу отношений этой подсистемы, т.е. любая подсистема отношений модели определяет некоторое новое однородное отношение.

Будем предполагать, что каждому отношению o^i модели $\mathfrak{M}$ соответствует набор явных функций $\varphi^i_l, l=\overline{1,n_i}$ разрешения этого отношения, причём каждая функция $\varphi\in\varphi^i_l$ имеет ранг r_i и реализуется некоторым программным модулем функционального наполнения ПП.

Вычислительную модель $\mathfrak{M}$, набор функций разрешения отношений модели, а также соответствие между функциями и математическими задачами с одной стороны и программно-реализующими их модулями с другой, будем использовать как математические и программные знания, используемые при синтезе алгоритма и при решении задач на модели.

Ограничимся рассмотрением проблем синтеза алгоритма расчётной задачи $З(A_{вх},A_{вых})$ на вычислительной модели $\mathfrak{M}$. Эта задача заключается в том, что среди множества атрибутов модели выделяется два непересекающихся подмножества: множество $A_{вх}$ исходных данных (входные атрибуты задачи) и множество $A_{вых}$ результатов (выходные атрибуты задачи). Решить расчётную задачу –

это значит при заданных значениях входных атрибутов найти значения выходных атрибутов, удовлетворяющие отношениям модели $\mathcal{M}$.

Важность расчётных задач заключается в том, что они являются подзадачами любой задачи расчёта (оптимизации) стационарных режимов функционирования технической системы, описываемой моделью $\mathcal{M}$.

В процессе синтеза алгоритма решения расчётной задачи естественно выделяются следующие четыре этапа:

I. Определить, разрешима ли задача, и если разрешима, то найти минимальную разрешающую подсистему отношений модели.
2. Для каждого отношения из найденной подсистемы выбрать функцию из набора функций разрешения этого отношения так, чтобы "сложность" алгоритма была минимальна.
3. Выделить вспомогательные подзадачи и построить алгоритм решения.
4. Сформировать программу решения расчётной задачи.

Особый практический интерес представляют первые два этапа, поэтому рассмотрим проблемы, возникающие при их реализации. Ограниченный объём статьи не позволяет привести подробный анализ и описание алгоритмов решения этих задач планирования. Перечислим лишь основные результаты.

I. Решение задачи планирования первого этапа. С расчётной задачей $З(A_{вх}, A_{вых})$ на модели $\mathcal{M} = (O, A)$ свяжем ориентированный взвешенный граф

$$K_{\mathcal{M}}(З) = (O \cup (A \setminus A_{вх}) \cup s \cup t, (V \setminus V_{вх}) \cup V_s \cup V_t, W),$$

где $O, A \setminus A_{вх}, \{s\}, \{t\}$ – вершины этого графа, а его дугам

$$V \setminus V_{вх} = \{(o^i, a_j) \mid o^i \in O, a_j \in a^i\} \setminus \{(o^i, a_j) \mid o^i \in O, a_j \in A_{вх}\},$$

$$V_s = \{(s, o) \mid o \in O\}, \quad V_t = \{(a, t) \mid a \in A \setminus A_{вх}\}$$

приписаны максимальные пропускные способности

$$W(v) = \begin{cases} 1, \text{ если } & v \in (V \setminus V_{вх}) \cup V_t \\ r_i, \text{ если } & v = (o^i, a), a \in A \setminus A_{вх} \end{cases}$$

Максимальный s-t поток в графе $K_{\mathcal{M}}(З)$ назовём полным, если он равен сумме рангов отношений модели $\mathcal{M}$.

Утверждение I.

Если произвольный максимальный поток в графе $K_{\mathcal{M}}(З)$ не

является полным, то задача 3 неразрешима.

Пусть u - произвольный полный s-t поток в графе $K_m(3)$. Он порождает ориентированный двудольный граф $D_m^u(3)=(O \cup (A \setminus A_{вых}), D)$ с множеством дуг D, получаемых из $V \setminus V_{вх}$ следующей ориентацией рёбер: ребро $v=(\sigma,a) \in V \setminus V_{вх}$ переходит в дугу (a,σ), если $u(v)=1$, либо в дугу (σ,a), если $u(v)=0$. Пусть $A_u \subseteq A \setminus A_{вх}$ - множество вершин, для которых в графе $D_m^u(3)$ существует дуга $(a,\sigma) \in D$.

Утверждение 2.

Если $A_{вых} \not\subseteq A_u$, то задача 3 неразрешима.

Через $C(A_{вых})$ обозначим множество вершин графа $D_m^u(3)$, предшествующих множеству выходных атрибутов задачи.

Утверждение 3.

Если $C(A_{вых}) \cap O \subseteq A_u$, то множество $C(A_{вых}) \cap (A \setminus A_{вых})$ является минимальной разрешающей системой для задачи 3. В противном случае задача 3 неразрешима.

Таким образом, решение задачи планирования первого этапа сводится к задаче нахождения полного потока в графе с целочисленными максимальными пропускными способностями и к задаче построения множества достижимости. Как известно, для решения этих задач существуют эффективные алгоритмы. Иначе обстоит дело при решении задач планирования второго этапа.

П. Решение задачи планирования второго этапа. Расчётную задачу, для которой существует разрешающая система, назовём разрешимой.

Утверждение 4.

Для решения произвольной разрешимой задачи для каждого отношения из минимальной разрешающей системы достаточно использовать ровно одну функцию.

Выбор допустимых функций разрешения $\Phi(\varphi^1_{j_1},\dots,\varphi^n_{j_n})$ определяет ориентированный двудольный граф

$$\Gamma_m^{\Phi}(3)=((C_0(A_{вых}) \cup O_{вх}) \cup (C_A(A_{вых}) \cup A_{вх}), \hat{D}),$$

где $C_0(A_{вых})=C(A_{вых}) \cap (A \setminus A_{вых})$, $C_A(A_{вых})=C(A_{вых}) \cap O$, $O_{вх}$ - отношение, значение атрибутов которого равны исходным данным задачи, а $\hat{D}=\{d \mid d=(\sigma,a)$, если a выходной атрибут для функции, разрешающей отношение $\sigma \in C_0(A_{вых}) \cup O_{вх}$ и $d=(a,\sigma)$, если a - выходной атрибут$\}$ - дуги графа.

Утверждение 5.

Решение расчётной задачи 3 сводится к решению следующей си-

стемы уравнений порядка n_3 :

$$F_{1i}(\vec{X}) = \ldots = F_{K_i+1}(\vec{X}), \qquad i = \overline{1, n_F} \qquad (I)$$

где F_{ji} – значение атрибутов $a \in C_A(A_{вых}) \cup A_{вх}$, для которых в графе $\Gamma^{\Phi}_{m}(3)$ полустепень захода K_i больше единицы,

$\vec{X}=(x_1,\ldots,x_{n_3})$ – значение атрибутов $a \in C_A(A_{вых}) \cup A_{вх}$, для которых в графе $\Gamma^{\Phi}_{m}(3)$ полустепень захода равна нулю,

$n_3 = \sum_{i=1}^{n_F} K_i$, а функции F вычисляются в силу отношений минимальной разрешающей системы.

Пусть $f = \max\limits_{i=\overline{1,n}} n_i$, а $\tau = \max\limits_{i=\overline{1,n}} \tau_i$.

Утверждение 6.

Задача выбора допустимого набора разрешений отношений, минимизирующего порядок n_3 системы (I) является NP –полной при $f \geqslant 2$ и $\tau \geqslant 2$.

Таким образом, для решения задачи планирования второго этапа необходимо использовать приближённые эвристические методы. Эти методы можно основывать на следующей специфике задачи.

Пусть часть функций для $O' \subset C_0(A_{вых})$ выбрана. Через A^i_+ обозначим множество выходных атрибутов функций φ^i_ℓ, $\ell = \overline{1, n_i}$

Рассмотрим взвешенный орграф

$$G^{O'}_{m}(3) = ((C_0(A_{вых}) \setminus O') \cup \hat{A} \cup s \cup t, V_s \cup V_t \cup \hat{V}, W),$$

где $\hat{A} = \bigcup\limits_{C_0(A_{вых})} A^i_+$ – множество выходных атрибутов функций разрешения отношений $C_0(A_{вых}) \setminus O'$, дуги $\hat{V} = \{(\sigma, a) \mid a \in \hat{A}\}$, а дуги V_s, V_t и веса W определяются также, как и для графа $K_m(3)$.

Утверждение 7.

При фиксированном допустимом наборе функций разрешения для части отношений $O' \subseteq C_0(A_{вых})$ нижняя оценка порядка системы (I) дается значением максимального s-t потока в графе $G^{O'}_{m}(3)$.

Это утверждение позволяет строить оценки ветвей дерева перебора при решении задачи планирования методами типа "ветвей и границ".

Рассмотренные принципы синтеза алгоритмов решения расчётных задач лежат в основе Планировщика инструментальной системы МАВР, разрабатываемой в Вычислительном Центре АН СССР [I - 2].

Литература

I. Родин С.Р., Эрлих А.И., Интерактивная система блочного моделирования технических систем. - Вопросы информационной теории и практики, № 46, М., ВИНИТИ, 1981.

2. Родин С.Р., Швалёв А.П., Эрлих А.И., МАВР - диалоговая система моделирования альтернатив и выбора решений, Preprint, 2nd International Conference "Artificial Intelligence and infomational control systems of robots", Smolenice, 1982.

IMYCS'84

ИСПОЛЬЗОВАНИЕ ОКРЕСТНОСТИ I ПОРЯДКА В ЛОКАЛЬНЫХ АЛГОРИТМАХ МИНИМИЗАЦИИ Д.Н.Ф.

Шестерова Н.А.
СССР, г.Ташкент, НПО "Кибернетика" АН УзССР

Среди множества различных алгоритмов построения оптимальных схем можно выделить класс локальных алгоритмов, для каждого элемента схемы проводящих анализ по "достаточно близким" к исследуемому элементам. Возможность решения в этом классе алгоритмов целого ряда экстремальных задач и, в частности, задач, связанных с проблемой минимизации дизъюнктивных нормальных форм (д.н.ф.) была показана Ю.И.Журавлевым / 2 /. В этой же работе впервые было введено понятие окрестности конъюнкции и доказано, что решение проблемы минимизации д.н.ф. в классе локальных алгоритмов невозможно, если использовать только информацию об окрестностях конечного порядка. Тем не менее,можно искать не обязательно минимальную (по количеству конъюнкций) д.н.ф., а достаточно близкую к ней, и проводить локальный анализ, например, по окрестности первого порядка, удаляя или закрепляя конъюнкции в д.н.ф. так,что длина полученной д.н.ф. будет достаточно близка к минимальной. Для оценки точности полученных результатов и трудоемкости используемых локальных процедур необходимо исследовать метрические (количественные) свойства окрестности первого порядка конъюнкций для большинства д.н.ф., реализующих булевы функции.

Д.н.ф. $\mathcal{D}$ представляет собой запись булевой функции f в виде дизъюнкции (многоместной функции "или") $K_1 \vee K_2 \vee \dots \vee K_m$, где каждый член K_j - элементарная конъюнкция (э.к.) - есть логическое произведение (многоместная функция "и") тех или иных переменных из совокупности $\{x_1, x_2, \dots, x_n\}$, взятых с отрицанием или без него. Каждая булева функция f имеет большое количество существенно различных д.н.ф., среди которых требуется найти минималь-

ную по длине. Очевидно, что минимальная д.н.ф. является неупрощаемой, т.е. перестает реализовывать булеву функцию f при удалении из нее любой э.к.

При изучении д.н.ф. часто прибегают к геометрической интерпретации, когда каждой булевой функции сопоставляется множество $\mathcal{N}_f$ точек $\tilde{\alpha}$ единичного n-мерного куба E^n таких, что $\tilde{\alpha} \in \mathcal{N}_f$, если $f(\tilde{\alpha}) = 1$, а каждой э.к. K - некоторый подкуб (интервал) $\mathcal{N}_K$ куба E^n. В этом случае задачу минимизации булевых функций в классе д.н.ф. можно заменить на эквивалентную ей задачу покрытия подмножества $\mathcal{N}_f$ куба E^n максимальными (не покрываемыми никаким другим интервалом из $\mathcal{N}_f$) интервалами. Необходимым и достаточным условием неупрощаемости покрытия будет существование в каждом интервале $\mathcal{N}_K$ этого покрытия хотя бы одной ядровой (не принадлежащей никакому другому интервалу данного покрытия) точки. Величина $d = n - r$, где r - количество входящих в э.к. K переменных, называется размерностью интервала $\mathcal{N}_K$. Очевидно, $|\mathcal{N}_K| = 2^d$.

Для каждой э.к. K окрестностью первого порядка относительно д.н.ф. D будет множество э.к. K_j из D таких, что $K \vee K_j \neq 0$ (или $\mathcal{N}_K \cap \mathcal{N}_{K_j} \neq \emptyset$).

Будем говорить, что почти все булевы функции обладают определенным свойством, если доля функций, не обладающих этим свойством, по отношению к общему количеству функций стремится к нулю.

Обозначим через $\mathcal{L}_{\kappa,t}(f)$ - число κ-мерных максимальных интервалов функции f, имеющих в пересечении с некоторым максимальным p-мерным интервалом из $\mathcal{N}_f$ интервал размерности t, $\overline{\mathcal{L}}_{\kappa,t}(n)$ - среднее значение этой величины по всем булевым функциям, содержащим данный p-мерный интервал (количество таких функций равно $(2^{2^p}-1)^{n-p} \cdot 2^{2^n - 2^p(n-p+1)}$). Тогда среднюю величину окрестности первого порядка по всем функциям f, содержащим данный p-мерный интервал, можно просчитать по формуле

$$\overline{\mathcal{L}}(n) = \sum_{t=0}^{p-1} \sum_{\kappa=t+1}^{n-p+t} \overline{\mathcal{L}}_{\kappa,t}(n).$$

ЛЕММА.
$$\overline{\mathcal{L}}_{\kappa,t}(n) = \frac{C_{n-p}^{\kappa-t} \cdot C_p^t \cdot 2^{p-t} \cdot 2^{2^p(n-p)}}{(2^{2^p}-1)^{n-p} \cdot 2^{2^\kappa - 2^t}} \left\{ \sum_{i=0}^{\kappa-t} (-1)^i \cdot C_{\kappa-t}^i \cdot 2^{(2^t-2^p)i} \times \right.$$
$$\left. \times \left(1 - \frac{1}{2^{2^\kappa - (i+1)2^t}}\right)^{p-t} \right\} \left(1 - 2^{-2^p} - 2^{-2^t} + 2^{2^t - 2^p - 2^\kappa}\right)^{n-p-\kappa+t}.$$

В связи с большим объемом выкладок, доказательство не приводится.

ТЕОРЕМА. Для почти всех булевых функций f от n переменных число $\mathcal{K}(f)$ э.к. в окрестности первого порядка произвольной конъюнкции, входящей в $\mathcal{N}_f$, удовлетворяет неравенству

$$\mathcal{K}(f) \leq n^{(1+\delta_n)\log\log n}, \quad \lim \delta_n = 0.$$

Доказательство. Для получения оценки используется известный / I / результат: доля функций, для которых $p(f) > \frac{1}{\varepsilon}\bar{p}(n)$ ($p(f)$ – значение некоторого параметра), не превосходит ε ($\varepsilon > 0$). Тогда $\mathcal{K}(f) \leq \frac{1}{\varepsilon}\bar{\mathcal{K}}(n) = \frac{1}{\varepsilon} \cdot n^{(1+\delta'_n)\log\log n}$, $\lim \delta'_n = 0$.

Полагая $\varepsilon = \frac{1}{n}$, получаем утверждение.

Полученная оценка означает, что применение локального анализа окрестности первого порядка в алгоритмах минимизации д.н.ф. увеличивает их сложность не более, чем в $n^{(1+\delta_n)\log\log n}$ раз ($\lim \delta_n = 0$). Однако, такой анализ в приближенных алгоритмах позволяет получать более оптимальное решение. Так, в широко известном градиентном алгоритме, который заключается в пошаговом отборе в покрытие $\mathcal{N}_f$ интервалов наибольшей размерности, конечным результатом для почти всех булевых функций является д.н.ф., отличающаяся по длине от минимальной не более, чем в $c_1 \cdot \log\log n$ раз (c_1 – константа) / 3 /, но, вообще говоря, данная д.н.ф. не будет неупрощаемой. Если из окрестности очередного рассматриваемого интервала $\mathcal{N}_\mathcal{K}$ удалять последовательно интервалы в порядке возрастания их размерностей до тех пор, пока в интервале $\mathcal{N}'_\mathcal{K}$ не появится хотя бы одна ядровая точка, и только после этого вносить интервал $\mathcal{N}'_\mathcal{K}$ в покрытие, а удаленные интервалы считать не входящими в минимальную д.н.ф., то результатом будет неупрощаемая д.н.ф., длина которой не более, чем в $c \cdot \log\log n$ раз (c – константа) больше длины минимальной для почти всех булевых функций.

Литература

1. Глаголев В.В. Некоторые оценки дизъюнктивных нормальных форм функций алгебры логики. Сб. Пробл. кибернетики, 1967, в.19, 75–94.
2. Журавлев Ю.И. Теоретико-множественные методы в алгебре логики. Сб. Пробл. кибернетики, 1962, в.8, 5–44.
3. Сапоженко А.А. О сложности дизъюнктивных нормальных форм, получаемых с помощью градиентного алгоритма. Сб. Дискретный анализ, 1972, в.21, 62–71.

AN INTERACTIVE IMPLEMENTATION OF KAREL THE ROBOT

Eleonóra Buriánová
Department of Theoretical Cybernetics
Faculty of Mathematics and Physics
Komensky University
842 15 Bratislava

Gabriela Syslová
Department of Theoretical Cybernetics
Faculty of Mathematics and Physics
Komensky University
842 15 Bratislava

0. Introduction

Karel the Robot as an introductory programming course has been introduced by R. Pattis in his famous book [1] . A slightly modified version of the language has been implemented at the Komensky University in Bratislava. The main distinction between the original and our version consists in different kinds of program processing. While Pattis have used a classical way, based on compiling of program followed by its execution, we have preferred a direct interpretation of each completed and syntactically correct statement. Since the text of program and Karel´s reactions can be observed on a screen simultaneously, the period between the writing of a statement and the observation of its execution is shortened as much as possible. Students are offered an ideal opportunity of discovering of errors, debugging and improving their programs.

The course based on our approach is assumed to be done just right at the terminals. To have the possibility of the parallel writing of program and of the presentation of its execution the screen of display is divided into three parts: two windows and a communication line.

The left window is used for writing of program, the right one shows Karel´s town and his moving there. A student write his program through keyboard, the system presents him the text of program and Karel´s reactions on the screen. The system also communicates with user through the communication line /detects the errors, asks for his agreement with Karel´s actions and so on/. In the case of any error, the text can be edited and it is executed once more /automatically/.

Because the problems solved are non-numerical and programming resembles a play, students formulate their own problems very easily and solve them as well. That way their activity is highly self-controlled and uses the comparison between the actions planned and provided for the self-education.

1. Building Karel´s town

"The play" with Karel consists of two parts: of building Karel´s town and of the programming Karel´s behaviour in the streets. In this section we describe how to build "the playground" - the world Karel will work in.

At the beginning the right window contains the rectangular net of points representing crossings of /horizontal/ streets and /vertical/ avenues. The user can put walls and beepers into the net. The walls are barriers on the route between two crossings, beepers are used as markers or counters. On each crossing from 0 to 9 beepers can be put and a wall can be laid between any two neighbour crossings.

Inserting of walls and beepers is interactive, too. The programmer leads a pointer on the desired point in the window and then places the object there /pressing a fixed key/. The instruction how to put and delete walls and beepers are written in the left window.

At the end of this part of play Karel the Robot is located at a particular crossing facing one of four cardinal points. The upper boundary of screen means the north and other cardinal points are oriented like on a map, too. Pressing a fixed key this part of "play" ends. The left window is cleaned / to be ready for writing program/, the right one remains unchanged.

2. Programming Karel´s behaviour

2. 1. Primitive instructions

In accordance with instructions Karel is able to move within his town and manipulate with beepers. He is not able go through walls, he must go round, Karel understands two primitive instructions that change his position. The first is

MOVE

which orders Karel to move forward to the next crossing. To avoid damaging himself, Karel will not move if he sees a wall or a boundary of window just before him. In that case he answers

I can´t go forward

in the communication line /CL/.

Note: If there are some beepers on the crossing Karel is standing on, there is no way to present Karel and beepers at the same point of screen simultaneously. Thus, while Karel stands on the crossing, the CL shows the number of beepers there.

The second primitive instruction is

TURNLEFT

This instruction changes the direction that Karel is facing, but does not alter his location.

The following instructions permit him to handle beepers. After

PICKBEEPER

he picks up a beeper from the crossing he is standing on. When there is none, Karel announces it in the CL.

Reversely, after the instruction

PUTBEEPER

he places a beeper on his current crossing. When there is nine beepers before the instruction, he reject the action with the announcement

There is no place for the next beeper.

Every instruction which can not be realized is assumed to be equal to the dummy one, i. e., the programmer can continue in writing program without any change in his program.

As you can see the reserved words are relatively long. That may cause problems to novice programmers who are not acquinted with the keyboard. For that reason, after pressing first three characters the system looks for an appropriate word in an inner table. When the word is found, the text on the screen is completed and the system asks for user's agreement with the choice in the CL.

When it is accepted the corresponding action is executed. In the opposite case system looks for another word. If there is no word beginning with those three letters in the table, the user is asked for a correction.

2. 2. Structured instruction

Since Karel is mentioned as an introduction to Pascal, algol-like structured statements are used.

Block structuring is accomplished by placing a sequence of instructions between the brackets BEGIN and END. Semicolons or other delimiters amoung instructions are not necessary, because words are completed by system and, consequently, the end of every syntactical unit is exactly known.

There are two kinds of loops Karel is able to do

ITERATE ⟨positive number⟩ TIMES ⟨instruction⟩

WHILE ⟨condition⟩ DO ⟨instruction⟩

and two conditional statements

IF ⟨condition⟩ THEN ⟨instruction-1⟩ ELSE ⟨instuction-2⟩

IF ⟨condition⟩ THEN ⟨instruction⟩

in their usual meanings. In the conditions we can test:
a/ Karel´s orientation: facing-north, not-facing-east etc.,
b/ Karel´s enviroment: front-is-blocked, left-is-clear,
next-to-a-beeper etc.

Again, as a help to the programmer, it is necessary to write only first three letters. Moreover, some words are filled up automatically - those ones which occurance is unaviodable in the given context. Say, TIMES after ITERATE and a positive number.

2. 3. Completion of program

The second part of "play" consists of the definition of a new instruction. The user writes his program into left window and can observe Karel reactions on the screen immediately. Every program has the form

DEFINE-NEW-INSTRUCTION ⟨new-name⟩ AS
BEGIN ⟨sequence-of-instructions⟩ END

The length of program can not exceed the size of window /16 lines per 38 characters/.

DEFINE-NEW-INSTRUCTION is written automatically directly as this part of play starts. The user has to write the whole name of instruction. The name is checked whether it is unique. If yes, the name is inserted into the table of reserved words and subroutine names /and later can be announced by its first three letters/. In the opposite case the user is asked for another name. The words AS BEGIN are filled up automatically after the specification of the new name. Then, the user programs a sequence of instructions and controls its correctness on the screen comparing the real activity of Karel with his planned one.

The user can use in his program any instruction defined before as a subroutine. Every user has his own library of programs, i. e., everybody "educates his own" Karel separately. The possibility of using subroutines allows to program very sophisticated programs. The recursion is not allowed.

In the case of any error the user can change his program using a build-in editor. If the error occurs in just written instruction /which has not been executed yet/, there are no problems. But, if the faulty instruction has been executed, after the correction the complete sequence of instructions is executed once more from the initial position. /Of course, the "complete sequence" means the part of program written till now./

After writting END the new defined instruction is inserted into user´s library and the user is asked to repeat the play. If he agrees, the process is repeated, otherwise the play ends.

3. Closing remarks

The implementation of Karel is based on a set of procedures which interpret the text of program/s/. The procedures that interpret structured statements and subroutine names are recursive and can call each other. The ones that interpret primitive instructions and communicate with user are non-recursive. The speed of execution is set up that way that Karel does not work lazy, but he is not very fast /to be observable/.

We have made experiments with a group of children ten years old who did not work with a computer before. Naturally, the dialoge was done in their native language, i. e., in the Slovak. They became fonds of Karel very quickly and even prefer him to other computer games. Nevertheless, the experiment has shown an interesting fact. Children have expected a major freedom in the language. They assumed Karel´s understanding of natural language constuctions like

MOVE AND TURNLEFT

or ITERATE WHILE ... and so on.

It seems to be resonable to allow them such combinations, so we want to build them in a future version. The goal should be to create a dialogue as natural as possible.

Reference:

[1] R. Pattis: Karel the Robot, John Wiley, New York, 1981

A GENERATING SYSTEM FOR PARTIAL RECURSIVE FUNCTIONS ON N*

by

Sándor Horváth

Eötvös Loránd University
Mathematical Institute
Department of Computer Science
Budapest, Múzeum körút 6-8
H-1088, Hungary.

Abstract. We give a simple, finite generating system for the class of $N^* \longrightarrow N^*$ type partial recursive functions, by adding a new operator ∇, called "iteration to zero", to the generating system of the class of $N^* \longrightarrow N^*$ type primitive recursive functions given by Mentrasti and Protasi (7).

We denote by N* the set of finite sequences over $N = \{0, 1, 2 \ldots\}$, including the empty sequence Λ too. Further, by PR_*^* and P_*^* we denote the classes of $N^* \longrightarrow N^*$ type primitive recursive and partial recursive functions, respectively (See (8) too). Mentrasti and Protasi (7) write SPR for PR_*^*, because they call the elements of PR_*^* sequence primitive recursive fuctions.) PR_n^m and P_n^m $(m, n \geq 1)$ will be the classes of (ordinary) primitive recursive and recursive functions, respectively.

We can define the class P_*^* in two plausible, equivalent ways. In the first way (followed e.g. in (5, 6) for some special subclasses of P_*^*) we identify P_*^* with the class of $N^* \longrightarrow N^*$ type partial functions computable by some abstract computer (e.g. Turing machine, RASP or RAM machine etc.) or Markov algorithm. In the second way (followed by Mentrasti and Protasi

(7) for PR_*^*) we define P_*^* to be $\{Q^{-1}\}\ P_1^1\ \{Q\}$ where $Q: N^* \longrightarrow N$ is the bijective "sequence encoding"

$$Q: (x_1, \ldots, x_k) \longmapsto \overline{1\underbrace{0\ \ldots\ 0}_{x_k}1\underbrace{0\ \ldots\ 0}_{x_{k-1}}10\ \ldots\ 01\underbrace{0\ \ldots\ 0}_{x_2}1\underbrace{0\ \ldots\ 0}_{x_1}}$$

$$Q: \Lambda \longmapsto 0$$

(the upper bar denotes the usual binary value). This sequence encoding Q was introduced independently in (3) and (4), and according to (5, 6) it is the simplest possible sequence encoding.

In general, if $f_1, f_2, \ldots$ are "base functions" and $0_1, 0_2, \ldots$ are "operators" then by $F(f_1, f_2, \ldots ; 0_1, 0_2, \ldots)$ we shall denote the class of functions "finitely generated" by the "genarating system" $(f_1, f_2, \ldots 0_1, 0_2, \ldots)$ (i.e. the smallest class of functions containing $f_1, f_2, \ldots$ and closed under $0_1, 0_2, \ldots$). Mentrasti and Protasi (7) proves that, defining the class PR_*^* as $\{Q^{-1}\}\ PR_1^1\ \{Q\}$,

$PR_*^* = F(0_r, S_r, \curvearrowleft, K;$ composition, repetition) where

$$0_r: (x_1, \ldots, x_k) \longmapsto (x_1, \ldots, x_k, 0)$$

$$S_r: \begin{cases} (x_1, \ldots, x_k, y) \longmapsto (x_1, \ldots, x_k, y+1) \\ \Lambda \longmapsto \Lambda \end{cases}$$

$$\curvearrowleft: \begin{cases} (x_1, \ldots, x_k, y) \longmapsto (y, x_1, \ldots, x_k) \\ \Lambda \longmapsto \Lambda \end{cases}$$

$$K: (x_1, \ldots, x_k) \longmapsto (x_1, \ldots, x_k, k)$$

$(k, x_1, \ldots, x_k, y \in N)$; the meaning of the operator "composition" is clear, and the operator "repetition" is defined for any $N^* \longmapsto N^*$ type partial function f as follows:

$$\text{repetition } (f): \begin{cases} (y, x_1, \ldots, x_k) \longmapsto f^{(y)}(x_1, \ldots, x_k) \\ \Lambda \longmapsto \Lambda \end{cases}$$

($f^{(y)}$ stands for $f \circ f \circ \ldots \circ f$ where the number of f's is y, and $f^{(0)}$ is the identity function).

Now we define the operator ∇, the operator "iteration to zero" on $N^* \longrightarrow N^*$ type partial functions f as

$$f^{\nabla} : (x_1, \ldots, x_v) \longmapsto (y_1, \ldots, y_w)$$

where $v, w, x_1, \ldots, x_v, y_1, \ldots, y_w \in N$,

$$k = \min\left\{k' \in N / f^{(k')} \text{ is of the form } (0, z_1, \ldots, z_q)\right\}$$

and

$$f^{(k)}(x_1, \ldots, x_v) = (0, y_1, \ldots, y_w)$$

(in case k is undefined, $f^{\nabla}(x_1, \ldots, x_v)$ is undefined too).

We introduce another operator, the operator Δ, the operator "iteration until odd" on $N \longrightarrow N$ type partial functions g as

$$g^{\Delta} : x \longmapsto y$$

where $x, y \in N$,

$$j = \min\left\{j' \in N / g^{(j')} \text{ is odd}\right\}$$

and

$$g^{(j)}(x) = y$$

Theorem 1. $P_1^1 = F(PR_1^1 ; \Delta, \text{composition})$.

Proof sketch. It is known from (1) that $P_1^1 = PR_1^1 (F(PR_2^2 ; \nabla)) PR_2^1$ (here, obviously, we restrict ∇ to $N^2 \longrightarrow N^2$ type partial functions). Actually, in (1) not exactly this restricted operator is used, but the modification is clear.) Now, defining the function $\gamma \in PR_1^2$ as

$$\gamma : \begin{cases} (0, y) \longmapsto 2y + 1 \\ (x + 1, y) \longmapsto 2(2^x (2y + 1) - 1) \end{cases} \qquad (x, y \in N)$$

clearly $\gamma^{-1} \in PR_2^1$ and γ has the property that $\gamma (x, y)$ is odd if $x = 0$. So, it can be seen easily that for any $f \in PR_2^2$, $f \nabla = U_2^2 \circ \gamma^{-1} \circ (\gamma \circ f \circ \gamma^{-1}) \Delta \circ \gamma$ (where $U_2^2 (z_1, z_2) = z_2$), from which the theorem already follows. Q. e. d.

Definition. Let $P_*^* = \{Q^{-1}\} P_1^1 \{Q\}$

Theorem 2. $P_*^* = F(PR_*^* ; \nabla, \text{composition})$.

Proof sketch. Observe that by the definition of Q, for any $x \in N^*$ $Q(x)$ is odd iff x is of the form $x = (x_1, \ldots, x_k)$ $(k \geq 1)$ where $x_1 = 0$. Now Theorem 2 fairly easily follows from the definition of the operator ∇ and Theorem 1. Q. e. d.

References

1. Brainerd, W. S. and Landweber, L. H.:
Theory of Computation. J. Wiley, New York, London, 1974.

2. Jacopini, G. and Mentrasti, P.:
Funzioni ricorsive di sequenza. Pub. I.A.C., Serie III, No. 104 (1975), 5 - 39.

3. Horváth, S.: Notes and lectures on abstract models of computers and programs, Eötvös Loránd University, Budapest, 1975 - 1980 (unpublished).

4. Mentrasti, P.: Sulle classi di funzioni elementari ed elementari inferiori in una variabile. Rend. Mat. (1), Vol. 9, Serie VI (1976), 37 - 56.

5. Horváth, S.: Complexity of sequence encodings. FCT'77, LNCS 56, Springer-Verlag, Berlin, Heidelberg, New York, 1977, 399 - 404.

6. Horváth, S.: Investigations in Computational Complexity of Abstract Computers. Dissertation, Eötvös Loránd University, Budapest, 1979 (in Hungarian).

7. Mentrasti, P. and Protasi, M.: Extended primitive recursive functions. RAIRO Informatique theorique/Theoretical Informatics, 16 (1982), 73 - 84.

8. Horváth, S.: On defining the Blum complexity of partial recursive sequence functions (extended abstract). EIK (Elektronische Informationsverarbeitung und Kybernetik) (to appear).