Comput. Math. Applic. Vol. 15, No. 9, pp. 723–742, 1988
Printed in Great Britain. All rights reserved

0097-4943/88 $3.00 + 0.00
Copyright © 1988 Pergamon Press plc

ATOMATICS AS MATHEMATICAL MODELLING AND KNOWLEDGE REPRESENTATION

APPLICATIONS TO SYSTEMS ANALYSIS IN COMPUTER AND INFORMATION SCIENCES

A. TUZHILIN
The College of Staten Island, CUNY, 130 Stuyvesant Place, Staten Island, NY 10301, U.S.A.

(*Received* 3 *September* 1987)

Communicated by E. Y. Rodin

1. INTRODUCTION

Application of mathematics to systems and situations in industry, business, and science begins with mathematical modelling of these systems and situations. Well established mathematical models of systems and situations are usually considered as representing knowledge about them. This type of knowledge representation is the most fundamental one on the level of contemporary knowledge. Overwhelming majority of contemporary mathematical models are related to systems and situations of the so-called natural type. That is, the real knowledge about these systems and situations is based on experimentations on them. Mathematical models for these systems and situations are based on different laws discovered in experiments. Examples of laws and models of this type we have in physics and chemistry. The corresponding mathematical field of modelling and analysis of models is usually called mathematical physics (in a broad sense). Mathematical models developed in mathematical physics are very often insufficient for practical applications. They describe usually some features of a real system, for example, mechanical or electromagnetic properties of a system. But a real system have usually all these properties simultaneously. Systems analysis appeared as an attempt to describe real objects more realistically.

Impetuous development of computer science gave rise to a new field for mathematical applications. At the beginning, it was obvious that the software part of computer science is a branch of mathematics. But this part is developing so dramatically fast by people so far from mathematics, that is seems that mathematics has very little in common with it. Moreover, contemporary software systems are so complex that there is no mathematical theory in contemporary mathematics being able to model and analyze these systems. Systems we have in computer and information sciences are different in nature from systems of the natural type. Knowledge about these systems is not based on experimentation. Therefore, modelling and analysis of them must be based on the analysis of our capability of knowledge representation. A new field appeared in computer science known as knowledge representation. But we must first analyze the fundamental methodology of knowledge representation and analysis, mathematical modelling.

A goal of this paper is to describe a methodology for mathematical modelling and knowledge representation which, as we believe, give us a basis for consistent and complete representation of results and basis for further development in business, technology, and science. To be efficient, the methodology should be based on and applied to analysis of real systems. We orient our methodology first toward systems in computer and information sciences which are prepared at most for application of the methodology. Systems in these sciences may serve as the touchstone for testing efficiency of the methodology. At the same time, we would like to develop techniques for the solution of a rather difficult problem in computer science, the problem of software analysis, correctness and design.

Let us consider briefly general methodologies of modelling and knowledge representation. In mathematical physics, we use mathematical constructions for mathematical modelling. Mathematics here is considered as a tool but the nature of this type of knowledge representation is not usually analyzed in the field. The analysis is left to philosophy or, at the best, to mathematical logic.

The philosophical approach to "modelling" (more precisely, to knowledge representation) is usually very broad and related to the entire world. The idea of the description of the entire world on the basis of primitive elements was originated in different ancient ideologies, and one of the constructive approaches, which has led to contemporary physics, chemistry, and biology, is known on the West as atomism.

Mathematical logic also tries to develop the methodology of knowledge representation known as the axiomatic methodology. The axiomatic methodology is usually related to different mathetical theories but not to real systems and situations. However, very broad applications of mathematics to "real world" give hope for creating a mathematical foundation for knowledge representation about this world.

There are two general approaches in the axiomatic methodology. The first approach begins with the definition of a language and then goes to formulation of axioms for a given mathematical theory. In contemporary mathematical logic, the next step for the analysis of the theory on a given set of axioms is the analysis of the so-called "models" for the set of axioms, that is, the analysis of the set of relations on a given set (sets) with some relations corresponding to the axioms. In this approach, many brilliant results were obtained which clarified the axiomatic approach.

The first axiomatic approach with model theory is however very far from axiomatization needed for practical systems and situations. The necessity for another axiomatic approach even for mathematical practice was recognized in mathematics. The second approach to axiomatization efficient in practical applications was developed by N. Bourbaki in his attempt to describe the entire mathematics on one unified principle. This unified principle is known as structural set methodology. The structural set methodology may be considered as a mathematical "atomic" approach to knowledge representation. In 1949 Bourbaki has said "... all mathematical theories may be regarded as extensions of the general theory of sets ...". These extensions are structural sets (structures). Bourbaki stated "... on these foundations I state that I can build up the whole of the mathematics of the present day ...".

What are problems we meet with in attempts to model and analyze real systems and situations, for example, such as software systems like operating systems? The first problem is the description (construction of mathematical models) of systems with complex structures. The methodology of mathematical modelling of such systems should describe not only some features of a system (as we often do in mathematical physics) but also the entire system. The second problem is the description and analysis of operations and relations on different types of complex systems and their subsystems.

For the solution of the first problem, we can try to use and to develop further ideas of atomism in the form developed in mathematics by Bourbaki which revealed to be very efficient in mathematics. An attempt for further development of Bourbaki's structural set methodology with application to systems analysis and mathematical modelling was carried out by the author (see, for example, [1] and [2]). Results of this development will be briefly discussed in this paper.

For the solution of the second problem, we have to develop a method that implements the idea of the so-called scale approach known also as an aphorism, "as above, so below". The scale approach is based on the "atomic" axiomatic approach. In the atomic axiomatic approach, we start with "primitive" elements of different types ("atoms" of different types). All complex objects and constructions (elements of the language on primitive and complex objects describing their properties and relations) are created on the basis of primitive objects. In the scale approach, complex objects of different types become primitive objects on the next level ("above"), but constructions on this level use properties of these new primitive elements as complex ones. The combination of these two approaches, the atomic axiomatic approach and the scale approach, makes our methodology rather efficient for mathematical modelling (and therefore knowledge representation) and analysis of practical systems and situations, for example, contemporary computers and their software systems. Applications of this methodology will be demonstrated in this paper.

In computer and information sciences, we usually have finite collections of primitive objects. Therefore, it seems that all constructions may be described in specific languages. In our methodology, we use languages with weights, suggested by Bourbaki (see [3]). We developed the equivalent constructions, called complexes, which are more obvious and efficient for further applications than words in a language. Complexes are trees with symbols assigned to vertexes. We consider these constructions as elements of metalanguage for representing complex objects and variable constructions of arbitrary order for describing relations. It seems that our variable constructions are much richer than in logical languages and that our language is more powerful than ordinary logical languages. We define logical constructions in our language including different types of quantifiers and define also the primitive set-theoretical relation, "belong". Therefore, we can create many constructions of set theory including structures in our language. Scale methodology is natural in this approach. Some results described above and their applications can be found in [4–6].

A goal of the methodology described in this paper is to be the main methodology for mathematical modelling in practical research and development and first of all in computer and information sciences. There are many prejudices against practical orientation of mathematics which prevent its further development for practical systems and situations. There are also many prejudices in practical research and development against mathematical applications, because mathematical models in these applications are very often much simpler than practical systems and situations. But mathematics was, is, and, we believe, will be the best way of knowledge representation and knowledge analysis. Any field of human activity based on mathematical modelling and analysis enjoys outstanding results. Examples are physics, chemistry, and, as a result, modern technology. To prevent prejudices against practical orientation of mathematics and prejudices against mathematics in practical research and development, we decided to call the methodology described in this paper as atomatics, that is, the approach of mathematics to practical research and development based on the "atomic" approach to axiomatization. We hope that the prejudice against the intrepid innovations which this term, atomatics, can stir up, is less harmful than described above prejudices. We also hope that this branch of mathematics will attract attention of many researchers for creating the proper basis (atomatic basis) for description and analysis of systems and situations in industry, business and science.

We orient this paper mainly to demonstration of applications of methodology developed. We remind first briefly main notions developed in other papers and then consider examples. These examples include: (i) modelling some practical systems; (ii) application of canonical morphisms in categories of structural sets to expansion of systems, known in some situations as "updating", or to process evaluation, known also in some situations as "scheduling"; (iii) functional analysis on spaces of states of subsystems of a system with application to analysis of software systems; this functional analysis is an extension of many sorted universal algebra; this extension incorporates structures from systems analysis into algebraic structures of many sorted universal algebra which permitted us to orient this functional analysis to practical applications; we consider, as an example of application, a simple operating system.

2. BOURBAKI'S STRUCTURAL SET METHODOLOGY AND UNIVERSAL RELATION LANGUAGE ON FINITE PRIMITIVE COLLECTIONS

In this section, we remind briefly main notions of Bourbaki's structural set methodology (see [3]) and its finite version developed by the author (see [5]).

Bourbaki starts with *primitive collections* (sets) of n different types, $E_1, E_2, \ldots, E_n$; and *auxiliary collections* of m different types, $A_1, A_2, \ldots, A_m$. Complex sets, called *echelons*, are constructed by using the operations, Cartesian product, x, and boolean, β (construction of power set). Echelons are denoted as $S(E_1, E_2, \ldots, E_n, A_1, A_2, \ldots, A_m) = S(\bar{E}, \bar{A})$. *Structural sets* (*on primitive collections* $E_1, E_2, \ldots, E_n$) is an object which consists of primitive and auxiliary sets and several, say 1, elements of echelons, $P_1 \in S_1(\bar{E}, \bar{A})$, $P_2 \in S_2(\bar{E}, \bar{A}), \ldots, P_l \in S_l(\bar{E}, \bar{A})$ with additional restrictions of them called transportable relations. We denote structural sets as

$$(E_1, E_2, \ldots, E_n; A_1, A_2, \ldots, A_m | P_1, P_2, \ldots, P_l) = (\bar{E}; \bar{A} \,|\, \bar{P}).$$

A finite version of structural set methodology allows us to describe all objects and constructions explicitly by using languages with weights or complexes. A language with weight is a set S of symbols with a function n: $S \to N$: $s \to n(s)$, where N is the set of all natural numbers. $n(s)$ is called the weight of a symbol (or a sign) s. Well-defined formulas, or correct words, in the language are strings of the form $sA_1, A_2, \dots, A_p$, where s is a symbol of the weight p and A_i are correct words for $i = 1, 2, \dots, p$. Correct words in a language with weight are called *significant* or *balanced words*. An equivalent construction to a significant word is a complex, which is a tree with vertexes being symbols (signs) and with the number of output arcs from a vertex equal to the weight of a sign associated with the vertex. Complexes permitted us to describe complex objects explicitly in the language

$$L(\,)^i\ (i \geqslant 2),\ \}^j\ (j \geqslant 1);\ E_1 \cup E_2 \cup \cdots \cup E_n)$$

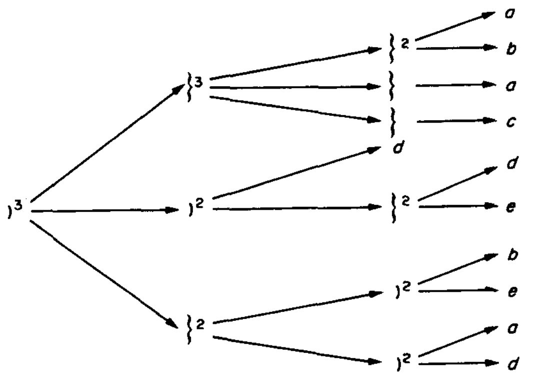
with signs $)^i$ of the weight i, $\}^j$ of the weight j, and all $e \in E_1 \cup E_2 \cup \cdots \cup E_n$ with weight 0. Let, for example, $E_1 = \{a, b, c\}$, and $E_2 = \{d, e\}$. The complex element

$$(\{\{a, b\}, \{a\}, \{c\}\}, (d, \{d, e\}), \{(b, e), (a, d)\})$$

of the echelon $\beta(\beta(E_1)) \times (E_2 \times \beta(E_2)) \times \beta(E_1 \times E_2)$ may be described as a significant word

$$)^3\}^3\}^2ab\}a\}c)^2d\}^2de\}^2)^2be)^2ad$$

or as a complex



We can classify complex objects in accordance with types. A type is defined by a significant word in the language $L(X^i\,(i \geqslant 2), P; x_1, x_2, \dots, x_n)$ with weight: $n(X) = 2$, $n(P) = 1$, $n(x_i) = 0$ for $i = 1, 2, \dots, n$. The collection of objects of a given type is an echelon. In our constructions, collections of primitive objects, $E_1, E_2, \dots, E_n$, may be any collections, for example, collections of objects of given structures. Examples below demonstrate applications of described constructions; these examples will also be used later.

Examples

1. Memory of computer

We consider a very simple structure of memory without paging. Virtual memory is considered in [4]. Memory of computer is a set of elements, say X, with assigned consecutive addresses from the interval of strings of six hexadecimal digits, [000000, FFFFFF] (a hexadecimal digit is an element from the set $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{E}, \mathrm{F}\}$). The beginning address is 000000. The address of the last element must be in the form $N_1N_2N_3\mathrm{FFF}$. Every element may be in a state from the set called byte-set, $B = \{00, 01, \dots, \mathrm{FF}\}$, that is, a state of an element is a string of two hexadecimal digits. Therefore, memory of computer is the following structure:

$$M = (X; [000000, \mathrm{FFFFFF}], B, |AD, E_X)$$

where AD is a function, AD: $X \to [000000, \mathrm{FFFFFF}]$; AD is an injection and $AD(X) = [000000, N_1N_2N_3\mathrm{FFF}]$. That is, AD assigns addresses to elements from X. Therefore, $AD \in \beta(X \times [000000, \mathrm{FFFFFF}])$.

E_X is a function which assigns the same space of states to every element, x, from X. That is, E_X: $X \to \beta(B)$: $x \to E_X(x) = E_x = B$; $E_X \in \beta(X \times \beta(B))$.

We can consider the structure called *memory elements*, $M_e = (X; B \mid E_X)$, and "primitive elements", $p_x = (\{x\}; B \mid E_x)$. Then memory of computer may be considered as

$$M = (M_e; [000000, \mathrm{FFFFFF}] \mid AD)$$

where the function, AD, is defined now on memory elements, M_e, with elements p_x for $x \in X$.

2. General registers of computer

We assume that there are 16 general registers in computer (as in IBM mainframe computers), called $R_0, R_1, \ldots, R_{15}$. States of a general register are elements from the data type, Z_F. Z_F is the set of totally ordered strings of 8 hexadecimal digits,

$$\langle 80000000, 80000001, \ldots, \mathrm{FFFFFFFF}, 00000000, 00000001, \ldots, \mathrm{7FFFFFFF} \rangle$$

with the operations: addition, $+_F$, subtraction, $-_F$, complement, C, and shift, S. That is, Z_F is the structure

$$Z_F = (\langle 80000000, 80000001, \ldots, \mathrm{7FFFFFFF} \rangle \mid +_F, -_F, C, S).$$

Elements of Z_F are strings of 8 hexadecimal digits. Therefore, a general register, R_I for $I = 0, 1, \ldots, 15$, is the structure, $R_I = (\{r_I\}; Z_F \mid E_{r_I})$ with E_{r_I}: $\{r_I\} \rightarrow \beta(Z_F)$: $r_I \rightarrow Z_F$ (here, r_I is the name of the register R_I).

3. The program status word (PSW)

We assume that PSW consists of the following elements: wait bit, w; interrupt code, ic; instruction length code, ilc; condition code, cc; program mask, pm; and address, ad. States of elements are the following sets:

$$E_w = B = \{0, 1\}, \qquad E_{ic} \subset X(4) = \{0000, 0001, \ldots, \mathrm{FFFF}\}$$
$$E_{ilc} = B(2) = \{00, 01, 10, 11\}, \qquad E_{cc} = B(2)$$
$$E_{pm} = X = \{0, 1, \ldots, \mathrm{F}\}, \qquad E_{ad} = X(6) = \{000000, 000001, \ldots, \mathrm{FFFFFF}\}.$$

Therefore, we have the following structures:

$$X = (\{x\}; B \mid E_X), \qquad Y = (\{y\}; X \mid E_Y)$$

for $x =$ w, ilc, cc and $E_X(x) = E_x$, $y =$ ic, pm, ad and $E_y(y) = E_y$. The set of elements of PSW is totally ordered, so that we have

$$\mathrm{PSW} = (\langle \mathrm{W, IC, ILC, CC, PM}, AD \rangle).$$

We represent PSW as a double word (a string of 16 hexadecimal digits) in the following way: Consider the following double words: $000X_3X_4X_5X_6X_7X_8X_9X_AX_BX_CX_DX_EX_F$ so that $X_3 \in E_w$, $X_4X_5X_6X_7 \in E_{ic}$, $X_8 \in E_{ilc} \| E_{cc}$ (here, $\|$ is the concatenation operation, that is, X_8 is a string from E_{cc} followed a string from E_{ilc}), $X_9 \in E_{pm}$, and $X_AX_BX_CX_DX_EX_F \in E_{ad}$. The space of states (the set of states) of PSW is in one-to-one correspondence with double words of this type.

4. Simplified hospital systems

We defined the structure of a simplified hospital system in [1] and [2] as follows:

$$H = (R, B, \Sigma, DS, D \mid P_B^R, F_{DS}^B, F_{DS}^\Sigma, F_{DS}^D, F_B^{P(D, DS)})$$

where R is a set of rooms, B is a set of beds, Σ is a set of specialized equipment, DS is a set of diseases, D is a set of doctors, $P(D, DS) = F_{DS}^D$. P_Y^X is a partition of the set X among the set Y, that is, a function from X into $\beta(Y)$ so that for $x_i \neq x_j$ from X we have $P_Y^X(x_i) \neq P_Y^X(x_j)$; and

$$\bigcup_{x \in X} P_Y^X(x) = Y.$$

F_Y^X is a distribution of X among Y, that is, a function from X into $\beta(Y)$. Additional axioms are:

$$P_{DS}^R = F_{DS}^B \circ P_B^R \text{ is a partition}$$

$$P_\Sigma^R = F_{DS}^{-1_\Sigma} \circ P_{DS}^R = F_{DS}^{-1_\Sigma} \circ F_{DS}^B \circ P_B^R \text{ is a partition}$$

$$pr_1 F_B^{P(D,\,DS)} = P(D, DS) = F_{DS}^D$$

$$F_B^{P(D,\,DS)}(d, dS) \subset F_{DS}^{-1_B}(dS).$$

We have for the hospital structure: $X_Y^Z \in \beta(Z \times \beta(Y))$ for $X_Y^Z = P_B^R$, or F_{DS}^B, or F_{DS}^Σ, or F_{DS}^D, or $F_B^{P(D,\,DS)}$ $(Z = D \times DS$ and $Y = B)$.

3. CANONICAL MORPHISMS, CATEGORIES OF STRUCTURAL SETS, APPLICATION OF CANONICAL MORPHISMS TO OBJECT EXPANSIONS

To define morphisms for structural sets, we have to define first extensions of mappings: For f: $E \to E'$, we have

$$\beta(f)\colon \beta(E) \to \beta(E')\colon U \to \beta(f)(U) = \{y\colon y = f(x) \text{ for some } x \in U\}.$$

For f: $E \to E'$, g: $F \to F'$, we have

$$f \times g\colon E \times F \to E' \times F'\colon (x, y) \to f \times g(x, y) = (f(x), g(y)).$$

Therefore, for every echelon, $S(E_1, E_2, \ldots, E_n)$, and f_i: $E_i \to E'$, $i = 1, 2, \ldots, n$ (or $\bar{f} = (f_1, f_2, \ldots, f_n)$: $\bar{E} \to \bar{E}'$), we have

$$S(f_1, f_2, \ldots, f_n) = S(\bar{f})\colon S(E) \to S(E').$$

THEOREM 1

If $\bar{f}$: $\bar{E} \to \bar{E}'$ and $\bar{g}$: $\bar{E}' \to \bar{E}''$, then $S(\bar{g} \circ \bar{f}) = S(\bar{g}) \circ S(\bar{f})$ $((\bar{g} \circ \bar{f}) = (g_1 \circ f_1, g_2 \circ f_2, \ldots, g_n \circ f_n)$ where $g_i \circ f_i$ is the composition of f_i and g_i). If $\bar{f}$ is an injection (surjection or bijection), then $S(\bar{f})$ is an injection (surjection or bijection) for every S.

THEOREM 2

If $(\bar{E}; \bar{A} \,|\, \bar{P})$ for $\bar{P} = (P_1, P_2, \ldots, P_1)$ and $P_i \in \beta(S_i(\bar{E}, \bar{A}))$ or $P_i \in E_{j_i}$ $(i = 1, 2, \ldots, 1)$, and $(\bar{E}'; \bar{A} \,|\, \bar{P}')$ for $\bar{P}' = (P'_1, P'_2, \ldots, P'_1)$ and $P'_i \in S_i(\bar{E}', \bar{A}))$ or $P'_i \in E'_{j_i}$ $(i = 1, 2, \ldots, 1)$ are structural sets of the same species, then vector-functions $\bar{f} = (f_1, f_2, \ldots, f_n)$: $\bar{E} \to \bar{E}'$ such that $S_i(\bar{f}, 1_{\bar{A}})(P_i) \subset P'_i$ if $P_i \in \beta(S_i(\bar{E}, \bar{A}))$ or $f_{j_i}(P_i) = P'_i$ if $P_i \in E_{j_i}$ are morphisms.

We call morphisms defined in Theorem 2 as *canonical morphisms*. Structural sets of the same species and their canonical morphisms constitute the category of a given species.

EXAMPLE

Morphisms of hospitals. For

$$H = (R, B, \Sigma, DS, D \,|\, P_B^R, F_{DS}^B, F_{DS}^\Sigma, F_{DS}^D, F_B^{P(D,\,DS)})$$

with $X_Y^Z \in \beta(Z \times \beta(Y))$ for $X_Y^Z = P_B^R$, or F_{DS}^B, or F_{DS}^Σ, or F_{DS}^D, or $F_B^{P(D,\,DS)}$, and

$$H' = (R', B', \Sigma', DS', D' \,|\, P'^{R'}_{B'}, F'^{B'}_{DS'}, F'^{\Sigma'}_{DS'}, F'^{D'}_{DS'}, F'^{P'(D',\,DS')}_{B'})$$

with $X'^{Z'}_{Y'} \in \beta(Z' \times \beta(Y'))$, a vector-function $f_H = (f_R, f_B, f_\Sigma, f_{DS}, f_D)$, where f_Y: $Y \to Y'$ for $Y = R$, or B, or Σ, or DS, or D, is a canonical morphism, f_H: $H \to H'$, iff $f_Z \times \beta(f_Y)(X_Y^Z) \subset X'^{Z'}_{Y'}$ (for $Z = P(D, DS)$, we have $f_Z = f_D \times f_{DS}$). This relation is equivalent to:

$$\begin{tikzcd}
Z \arrow[r, "f_Z"] \arrow[d, "X_Y^Z"'] & Z' \arrow[d, "X'^{Z'}_{Y'}"] \\
\beta(Y) \arrow[r, "\beta(f_Y)"] & \beta(Y')
\end{tikzcd}$$

which commutes.

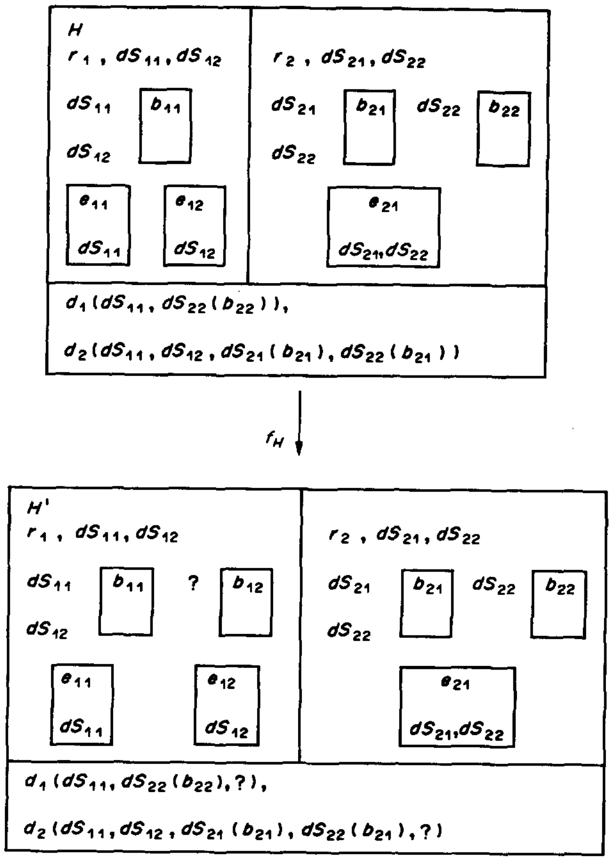


Fig. 1

We apply now canonical morphisms to expansion of structural sets. We demonstrate this idea on an expansion of a hospital. Let H be a hospital based on the following sets: $R = \{r_1, r_2\}$, $B = \{b_{11}, b_{21}, b_{22}\}$, $\Sigma = \{e_{11}, e_{12}, e_{21}\}$, $DS = \{dS_{11}, dS_{12}, dS_{21}, dS_{22}\}$, $D = \{d_1, d_2\}$ with structures described in Fig. 1 above. We would like to construct all hospitals, H', which contain one additional bed, b_{12}, in room, r_1, that is, all hospitals, H', for which the vector-function $f_H = (1_R, f_B, 1_\Sigma, 1_{DS}, 1_D)$ is a morphism (here, f_B: $B \to B'$, $B' = \{b_{11}, b_{12}, b_{21}, b_{22}\}$, and f_B is an inclusion). That is, we should have f_H: $H \to H'$ as described in Fig. 1.

We have $P'^R_{B'} = \{(r_1, \{b_{11}, b_{12}\}), (r_2, \{b_{21}, b_{22}\})\}$, $F'^\Sigma_{DS} = F^\Sigma_{DS}$, $F'^D_{DS} = F^D_{DS}$. Therefore, we have to calculate $F'^{B'}_{DS}$ and $F'^{P(D,DS)}_{B'}$. We have $F'^{B'}_{DS}(b_{ij}) = F^B_{DS}(b_{ij})$ for $ij = 11$, 12 and 22. Calculations give the following possibilities:

$$F'^{B'}_{DS}(b_{12}) = \begin{cases} \{dS_{11}\} \\ \{dS_{12}\} \\ \{dS_{11}, dS_{12}\} \end{cases}$$

and correspondingly

$$F'^{P(D,DS)}_{B'} = \left\{ \left((d_1, dS_{11}), \begin{cases} \{b_{11}, b_{12}\} \\ \{b_{11}\} \\ \{b_{11}, b_{12}\} \end{cases} \right), ((d_1, dS_{22}), \{b_{22}\}), \right.$$

$$\left. \left((d_2, dS_{11}), \begin{cases} \{b_{11}, b_{12}\} \\ \{b_{11}\} \\ \{b_{11}, b_{12}\} \end{cases} \right), \left((d_2, dS_{12}), \begin{cases} \{b_{11}\} \\ \{b_{11}, b_{12}\} \\ \{b_{11}, b_{12}\} \end{cases} \right), ((d_2, dS_{21}), \{b_{21}\}), ((d_2, dS_{22}), \{b_{21}\}) \right\}.$$

4. SYSTEMS ANALYSIS

We remind, first, notions of general systems analysis. Details and examples of classical systems can be found in [1] and [2]. A system consists of components (equipment), called *element base*, B; "physical properties" of components and relations of components are described by a *space of available states of element base*, S^B; a *space of processes*, Π, describes possible processes in a system; a space of processes is associated with a *space of time scales*, Θ, on which processes take place. Therefore, a system is the structure

$$\eta = (B, S^B, \Theta, \Pi).$$

An *element base*, B, is the structure, $B = (A, I, O, C \mid \delta_0, \delta_1)$, where A is a *set of elements*, I is a *set of inputs*, O is a *set of outputs*, C is a *set of connectors*, δ_0: $C \cup O \to A$ assigns an initial element to every connector and output, δ_1: $C \cup I \to A$ assigns a final element to every connector and input; $\delta_0 \in \beta((C \cup O) \times A)$, $\delta_1 \in \beta((C \cup I) \times A)$. A *space of available states of elements from* X is the structure, $S^X = (X, \Sigma_X \mid E_X, S_X)$, where Σ_X is a *set of possible states of all elements from* X; E_X: $X \to \beta(\Sigma_X)$: $x \to E_X(x) = E_x$ assign the *space of states*, E_x, to an element, $x \in X$;

$$S_X \subset \prod_{x \in X} E_X$$

is a *set of available states of the entire* X, that is, a set of functions of the type, s: $X \to \Sigma_X$: $x \to s(x) \in E_x$. We have, $E_X \in \beta(X \times \beta(\Sigma_X))$, $S_X \in \beta(\beta(X \times \Sigma_X))$. A *space of time scales* is the structure, $\Theta = (T, \Sigma_T \mid E_T, G)$, where T is a *set of time scales*, Σ_T is a *set of all time instants*; E_T: $T \to \beta(\Sigma_T)$: $\tau \to E_T(\tau) = E_\tau$ assigns the *set of time instants to a scale* τ; G: $T \to \beta(\Sigma_T \times \Sigma_T)$: $\tau \to G(\tau) = G_\tau$ defines total orders ("yesterdays", "todays", and "tomorrows") on all E_τ. We have, $E_T \in \beta(T \times \beta(\Sigma_T))$, $G \in \beta(T \times \beta(\Sigma_T \times \Sigma_T))$. A *space of processes in a set of states*, S, is the structure, $\Pi = (\Theta, S \mid \bar{\Pi})$, where Θ is a space of time scales; $\bar{\Pi}$ a *set of processes*, $\pi \in \bar{\Pi} \Rightarrow (\pi$: $E_\tau \to S$ for some time scale τ from $\Theta)$. We have, $\bar{\Pi} \in \beta(\beta(\Sigma_T \times S)$.

Example

Hospital as a system. A hospital as a system consists of an input, I, describing patients to be cured, a hospital, H, as an element (a state of the element H is a set of patients in the hospital), and an output, O, describing patients leaving the hospital. Therefore, the element base, B, is

$$\xrightarrow[I]{} H \xrightarrow[O]{}$$

A set of patients is described by the structure, $P^S = (P, DS \mid T^P_{DS})$, where P is a *set of patients*, DS is a *set of diseases*, and T^P_{DS} is a distribution, T^P_{DS}: $P \to \beta(DS)$: $p \to T^P_{DS}(p) = DS_p$; DS_p is the set of diseases of a patient p. A state of the input, I, is a patient structure, $P^S_I = (P_I, DS \mid T^P_{DS,I})$.

To define states of a hospital, H, and an output, O, we define an assigning array. An *assigning array*, A^P, for a given patient structure, P^S, and a hospital, H, is an array, $A^P = (f^P_B, f^P_{DS}, f^P_\Sigma, f^P_D)$, of the following mappings:

f^P_B: $P \to B$ is an injection, assigns a unique bed, $f^P_B(p)$, to every patient $p \in P$;

f^P_{DS}: $P \to \beta(DS)$ $(f^P_{DS}(p) \subset T^P_{DS}(p))$ assigns diseases, $f^P_{DS}(p)$, to a patient $p \in P$, which will be treated in the hospital, $f^P_{DS}(p) \subset F^B_{DS}(f^P_B(p))$;

f^P_Σ: $P \to \beta(\Sigma)$ assigns a set of equipment, $f^P_\Sigma(p) \subset \Sigma$, to a patient $p \in P$ that may cure diseases $f^P_{DS}(p)$, $f^P_{DS}(p) \subset F^\Sigma_{DS}(f^P_\Sigma(p))$;

f^P_D: $P \to \beta(D)$ assigns a set of doctors, $f^P_D(p) \subset D$, to a patient $p \in P$ that may cure diseases $f^P_{DS}(p)$, $f^P_{DS}(p) \subset F^D_{DS}(f^P_D(p))$.

A *state of the hospital* is a pair, (P^S_H, A^{P_H}), of a patient structure, P^S_H, and an assigning array, A^{P_H}, for these patients. A *state of the output* is a pair, (P^S_O, A^{P_O}), of a patient structure, P^S_O, and a corresponding array, A^{P_O}.

The definition of an available state is based on the assumption that there is no queue for an available state and on a composition operation, "+" on pairs (P^S, A^P). If P_1 and P_2 are two sets of different patients, $P_1 \cap P_2 = \varnothing$, then

$$(P_1^S, A^{P_1}) + (P_2^S, A^{P_2}) = (P_1^S + P_2^S, A^{P_1+P_2})$$

where

$$P_1^S + P_2^S = (P_1 \cup P_2, DS \mid T_{DS}^{P_1+P_2}),$$

$$T_{DS}^{P_1+P_2}\colon P_1 \cup P_2 \to \beta(DS)\colon p \to T_{DS}^{P_1+P_2}(p) = \begin{cases} T_{DS}^{P_1}(p) \text{ for } p \in P_1 \\ T_{DS}^{P_2}(p) \text{ for } p \in P_2 \end{cases}$$

$$A^{P_1+P_2} = (f_B^{P_1+P_2}, f_{DS}^{P_1+P_2}, f_\Sigma^{P_1+P_2}, f_D^{P_1+P_2})$$

$$f_B^{P_1+P_2}\colon P_1 \cup P_2 \to B;\ f_B^{P_1+P_2}|_{P_i} = f_B^{P_i} \quad (i = 1, 2)$$

$$f_X^{P_1+P_2}\colon P_1 \cup P_2 \to \beta(X);\ f_X^{P_1+P_2}|_{P_i} = f_X^{P_i} \quad (i = 1, 2)$$

for $X = DS$, Σ, and D, and $F_{DS}^{P_1+P_2}(p) \subset F_{DS}^Y(f_Y^{P_1+P_2}(p))$ for $Y = B$, Σ, D.

A *state*, $(P_I^S, (P_H^S, A^{P_H}), (P_O^S, A^{P_O}))$ is *available* iff there exists an assigning array, A^{P_I}, so that the sums $(P_I^S, A^{P_I}) + (P_H^S, A^{P_H})$ and $(P_H^S, A^{P_H}) + (P_O^S, A^{P_O})$ exist. Time scales of the hospital are $E_\tau = \{t_1^\tau, t_2^\tau, \ldots, t_{n_\tau}^\tau\}$ with $t_1^\tau < t_2^\tau < \cdots < t_{n_\tau}^\tau$. A process, π_τ, is $\pi_\tau\colon E_\tau \to S^H$, where S^H is the space of available states.

5. APPLICATION OF CANONICAL MORPHISMS TO PROCESS EVALUATION

A space of available states, S^B, of a system is usually a category of structural sets. For a discrete process, π, of a system on a time scale, $\tau = \{t_1, t_2, \ldots, t_n\}$, we have, $\pi\colon \tau \to S_B\colon t_i \to \pi(t_i)$. The state $s_{i+1} = \pi(t_{i+1})$ is a morphism from $s_i = \pi(t_i)$. We will use these morphisms for process evaluation.

Example

Processes in a hospital. An available state of a hospital as a system is $s = (P_I^S, (P_H^S, A^{P_H}), (P_O^S, A^{P_O}))$, where $A^P = (f_B^P, f_{DS}^P, f_\Sigma^P, f_D^P)$ is an assigning array for a given patient structure $P^S = (P, DS \mid T_{DS}^P)$. A state is an object in the category with primitive sets, B, R, Σ, DS, D, P_I, P_H, P_O. A morphism of a state s into a state s' is a vector-function $\varphi = (f_B, f_R, f_\Sigma, f_{DS}, f_D, f_{P_I}, f_{P_H}, f_{P_O})$, where $f_X\colon X \to X'$ for $X = B$, R, Σ, DS, D, P_I, P_H, and P_O, such that

(i) the vector-function, $\varphi = (f_R, f_B, f_\Sigma, f_{DS}, f_D)$, is a morphism of the hospital, H, into the hospital, H';

(ii) the pair, (f_{P_I}, f_{DS}), is a morphism of the patient structure, P_I^S, into $P_I'^S$;

(iii) the pair, (f_{P_H}, f_{DS}), is a morphism of the patient structure, P_H^S, into $P'^S_{H'}$, and the vector-function, $\varphi_{A^H} = (f_{P_H}, f_B, f_{DS}, f_\Sigma, f_D)$, is a morphism of the assigning array, A^{P_H}, into $A'^{P'_{H'}}$;

(iv) the pair, (f_{P_O}, f_{DS}), is a morphism of the patient structure, P_O^S, into P'^S_O, and the vector-function, $\varphi_{A^O} = (f_{P_O}, f_B, f_{DS}, f_\Sigma, f_D)$, is a morphism of the assigning array, A^{P_O}, into $A'^{P'_O}$.

For process evaluation, we have, $f_B = 1_B\colon B \to B$, $f_R = 1_R\colon R \to R$, $f_\Sigma = 1_\Sigma\colon \Sigma \to \Sigma$, $f_{DS} = 1_{DS}\colon DS \to DS$, $f_D = 1_D\colon D \to D$.

As an example, we consider, now, the hospital, H, described in Fig. 1. We assume that the space of available states of the hospital is:

$$P_I^S = (\{p_{I1}\}, DS \mid \{(p_{I1}, \{dS_{11}, dS_{22}\})\})$$

$$P_H^S = (\{p_{H1}, p_{H2}\}, DS \mid \{(p_{H1}, \{dS_{11}, dS_{12}, dS_{22}\}), (p_{H2}, \{dS_{22}\})\})$$

$$P_O^S = (\{p_{O1}\}, DS \mid \{(p_{O1}, \{dS_{22}\})\})$$

$$f_B^{P_H} = \{(p_{H1}, b_{11}), (p_{H2}, b_{21})\}$$

$$f_{DS}^{P_H} = \{(p_{H1}, \{dS_{11}, dS_{12}\}), (p_{H2}, \{dS_{22}\})\}$$

$$f_\Sigma^{P_H} = \{(p_{H1}, \{e_{11}, e_{12}\}), (p_{H2}, \{e_{21}\})\}$$

$$f_D^{P_H} = \{(p_{H1}, \{d_1, d_2\}), (p_{H2}, \{d_2\})\}$$

$$f_B^{P_O}(p_{O1}) = b_{22}$$
$$f_{DS}^{P_O}(p_{O1}) = \{dS_{22}\}$$
$$f_\Sigma^{P_O}(p_{O1}) = \{e_{21}\}$$
$$f_D^{P_O}(p_{O1}) = \{d_1\}.$$

We would like to find out all next available states of the hospital if the patient on input is hospitalized and the patient on output left the hospital. Therefore, for the next state we have, $P'_I = \varnothing$; $P'_O = \varnothing$; $P'_H = P'_H \cup \{p_{I1}\}$; $f_{P_I} = \varnothing$; $f_{P_O} = \varnothing$; f_{P_H}: $\{p_{H1}, p_{H2}\} \to \{p_{H1}, p_{H2}, p_{I1}\}$ is an inclusion. Calculations give only one next available state:

$$P'^S_I = (\varnothing, DS \mid \varnothing)$$
$$P'^S_O = (\varnothing, DS, \mid \varnothing)$$
$$f'^{P'_O}_B = f'^{P'_O}_{DS} = f'^{P'_O}_\Sigma = f'^{P'_O}_D = \varnothing$$
$$P'^S_H = (\{p_{H1}, p_{H2}, p_{I1}\}, DS \mid \{(p_{H1}, \{dS_{11}, dS_{12}, dS_{22}\}), (p_{H2}, \{dS_{22}\}), (p_{I1}, \{dS_{11}, dS_{22}\})\})$$
$$f'^{P'_H}_B = \{(p_{H1}, b_{11}), (p_{H2}, b_{21}), (p_{I1}, b_{22})\}$$
$$f'^{P'_H}_{DS} = \{(p_{H1}, \{dS_{11}, dS_{12}\}), (p_{H2}, \{dS_{22}\}), (p_{I1}, \{dS_{22}\})\}$$
$$f'^{P'_H}_\Sigma = \{(p_{H1}, \{e_{11}, e_{12}\}), (p_{H2}, \{e_{21}\}), (p_{I1}, \{e_{21}\})\}$$
$$f'^{P'_H}_D = \{(p_{H1}, \{d_1, d_2\}), (p_{H2}, \{d_2\}), (p_{I1}, \{d_1\})\}.$$

The vector-function, $(1_B, 1_R, 1_\Sigma, 1_{DS}, 1_D, \varnothing, f_{P_H}, \varnothing)$, is a morphism from the given state into the next one.

6. INFORMATION BASES OF SYSTEMS

For a system, $\eta = (B, S^B, \Theta, \Pi)$, a pair, (B, S^B), is called an *information base*. A simple version of an information base is a *space of states*, that is the structure, $(X, \Sigma_X \mid E_X)$, where X is a set of elements (components) of a system, Σ_X is a set of all possible states, and E_x: $X \to \beta(\Sigma_X)$: $x \to E_x(x)$ is a function assigning spaces of states to elements of the system (that is, defining "physical sense" of elements). For finite set $X = \{X_1, X_2, \ldots, X_n\}$, we can represent a space of states in the form of PL/1-like structure:

```
1 INF_BASE,
    2 X1         EX1,
    2 X2         EX2,
    ...................
    2 Xn         EXn;
```

Here, INF_BASE is a name of an information base and $EX_i = E_X(X_i)$ is a space or states of an element X_i.

For a space of available states, $(X, \Sigma_X \mid E_X, S_X)$, where

$$S_X \subset \prod_{x \in X} E_X(x)$$

describes relations between states of elements, we assume that this additional structure S_X may be described by an axiom, that is, by some statement (relation) in the Universal Relation Language on primitive collections X and Σ_X (or may be additional primitive collections, if the system uses them; it seems that this assumption is a theorem, because the Universal Relation Language is rather rich; in applications we will not have difficulties to formulate these axioms). Therefore, the space of available states may be represented as

```
1 INF_BASE                AXIOM(R),
    2 X1                       EX1,
    2 X2                       EX2,
..............................
    2 Xn                       EXn;
AXIOM R:.......................;
```

We use the word, AXIOM, as a key word, therefore we underline it.

We can consider now situations when our primitive elements may be spaces of available states. For such elements, we can use spaces of available states as spaces of states. If we would like to specify explicitly the structure of an element, we can use additional levels.

For example,

```
1 INF_BASE                AXIOM(R),
    2 X1                       EX1,
    2 X2                AXIOM(RX2),
            3 Y1               EY1,
            3 Y2               EY2,
    AXIOM RX2:...................,
    2 X3                       EX3;
AXIOM R:.......................;
```

The general construction is the following. Let X and Σ be different primitive collections of a Universal Relation Language, $\mathrm{URL}(E_1, E_2, \ldots, E_n)$. A *relational complex on X in URL* is a tree, Γ, with two functions, f_Γ, and f_A. The function f_Γ, assigns primitive objects from X to elements of Γ, and the function f_A, assigns relations (axioms) to some internal vertexes of Γ. Let L_Γ, IN_Γ, and A_Γ be correspondingly leaves of Γ, internal vertexes of Γ, and a subset of IN_Γ. We have, f_Γ: $\Gamma \to X$, $f_\Gamma(\Gamma) = X$, and f_A: $A_\Gamma \to \mathrm{REL(URL)}$, where REL is the set of relations of URL. Therefore, a relational complex is the structure, $C(\Gamma, A_\Gamma, X) = (\Gamma, X; \mathrm{URL} | f_\Gamma, A_\Gamma, f_A)$, with the additional axiom: if γ_1 and γ_2 are two vertexes of Γ, so that $f_\Gamma(\gamma_1) = f_\Gamma(\gamma_2)$, then two relational complexes, $C(\Gamma, A_\Gamma, X)(\gamma_1)$ and $C(\Gamma, A_\Gamma, X)(\gamma_2)$ with roots at γ_1 and γ_2 are the same.

Proposition 1

Let R be the following equivalence relation, $\gamma_1 \equiv \gamma_2 \Leftrightarrow f_\Gamma(\gamma_1) = f_\Gamma(\gamma_2)$, and Γ/R, L_Γ/R, and A_Γ/R be corresponding classes. Then $L_\Gamma/R \subset \Gamma/R$, $IN_\Gamma/R \subset \Gamma/R$, $A_\Gamma/R \subset \Gamma/R$ and there are one-to-one correspondences between $f_\Gamma(\Gamma)$ and Γ/R, and between $f_A(A_\Gamma)$ and A_Γ/R.

Therefore, every class of Γ/R is uniquely represented by an element x from X. The space of available states is the following structure:

$$S^X = (X, \Sigma \mid C(\Gamma, A_\Gamma, X), E_X, S_X)$$

where E_X: $L_\Gamma \to \beta(\Sigma)$: $x \to E_x$. The space of available states,

$$S_X \subset \prod_{x \in X} E_x$$

is defined as follows.

Because Γ is a tree, it can be represented in canonical way by its layers. Let the number of layers be m, and let $L_\Gamma(i)$, $IN_\Gamma(i)$, $A_\Gamma(i)$ be vertexes from L_Γ, IN_Γ, A_Γ correspondingly on the layer i, for $i = 1, 2, \ldots, m$. We have, $IN_\Gamma(m) = A_\Gamma(m) = \varnothing$. We construct now the family $(S_x)_{x \in \Gamma/R}$. If $x \in L_\Gamma/R$, then $S_x = E_x$. To define other elements of the family, we will use the operation, OUT. OUT: $\Gamma \to \beta(\Gamma)$: $\gamma \to \mathrm{OUT}(\gamma)$, where $\mathrm{OUT}(\gamma)$ are output vertexes from γ. If $x \in IN_\Gamma(i)/R - A_\Gamma(i)/R$, then

$$S_x = \prod_{y \in \mathrm{OUT}(x)} S_y$$

if $x \in A_\Gamma(i)/R$, then

$$S_x = \left\{ s\colon s \in \prod_{y \in \mathrm{OUT}(x)} S_y \quad \text{and} \quad f_A(x) \right\}.$$

If z is a root of Γ, then $S_X = S_z$.

For example, let $X = \{a, b, c, d, e, f, g, h\}$ and Σ be primitive collections. Let $A1$ and $A2$ be axioms, that is, elements from REL(URL(X, Σ)). For the relational complex

$$\begin{tikzcd}
{} & d & a & \\
{} & (e,A2) \arrow[ur] \arrow[r] & b & \\
{} & & b & \\
(h,A1) \arrow[uuur] \arrow[uur] \arrow[r] \arrow[ddr] \arrow[ddddr] & f \arrow[ur] \arrow[r] & c & \\
{} & & a & \\
{} & (e,A2) \arrow[ur] \arrow[r] & b & \\
{} & & & a \\
{} & g \arrow[r] & (e,A2) \arrow[ur] \arrow[r] & b
\end{tikzcd}$$

we have, $S_a = E_a$, $S_b = E_b$, $S_c = E_c$, $S_d = E_d$; $S_e = \{s\colon\ s \in E_a \Pi E_b$ and A2$\}$, $S_f = E_b \Pi E_c$, $E_g = E_a \Pi S_e$, $S_h = \{s\colon s \in E_d \Pi S_e \Pi S_f \Pi S_e \Pi S_g$ and A1$\}$. The space of available states can be represented as PL/1-like structure:

```
1 h                           AXIOM(A1),
    2 d                       Ed,
    2 e                       AXIOM(A2),
        3 a                   Ea,
        3 b                   Eb,
    AXIOM A2:...............,
    2 f,
        3 b                   Eb,
        3 c                   Ec,
    2 e                       AXIOM(A2),
        3 a                   Ea,
        3 b                   Eb,
    AXIOM A2:...............,
    2 g,
        3 e                   AXIOM(A2),
                4 a           Ea,
                4 b           Eb,
        AXIOM A2;...........,
AXIOM A1:.......................;
```

It is obvious that every space of available state can be represented as PL/1-like structure.

In practical applications, we need more complex construction than the space of available states, namely, the space of available states with variable structure. On intuitive level, this structure is a set of different versions of spaces of available states. The *space of available states with variable structure* is the following structure:

$$SV^X = (X, \Sigma_X | E_X, J, C_X, S_X)$$

where X and Σ_X are primitive collections in a URL, $E_X\colon X \to \beta(\Sigma_X)\colon x \to E_x$, as before. J describes different versions, that is, a set of subsets of elements from X; $J \in \beta(\beta(X))$. C_X is a function from the set of versions, J, into the category of relational complexes in X, that is, C_X: $J \to \mathrm{COM}(X, \mathrm{URL})\colon j \to C_X(j) = C_X(\Gamma_j, A_{\Gamma,j}, j)$. S_X is a function from the set of versions, J, into the category of spaces of available states, that is, $S_X\colon J \to \mathrm{SAS}(X, \Sigma_X, \mathrm{URL})\colon j \to S_X(j)$ and $S_X(j)$ is defined for j and $C_X(j)$ as described above.

In the simple situation, when

$$X = \{X_1, X_2, \ldots, X_N\}$$

$$J = \{\{X_{11}, X_{12}, \ldots, X_{1N_1}\}, \{X_{21}, X_{22}, \ldots, X_{2N_2}\}, \ldots, \{X_{L1}, X_{L2}, \ldots, X_{LN_L}\}\}$$

(here X_{IJ} is from X) and there is no complex elements, we can represent a space of available states with variable structure in the following form:

```
SPACE_AV_VER:
     VERSION(1):  1 SUB1            AXIOM1(R1),
                    2 X11           EX11,
                    2 X12           EX12,
                    .....................
                    2 X1N_1         EX1N_1,
                  AXIOM R1:..............;
     VERSION(2):  1 SUB2            AXIOM(R2),
                    2 X21           EX21,
                    2 X22           EX22,
                    .....................
                    2 X2N_2         EX2N_2,
                  AXIOM R2:...............;
     ......................................
     VERSION(L):  1 SUBL            AXIOM(RL),
                    2 XL1           EXL1,
                    2 XL2           EXL2,
                    .....................
                    2 XLN_L         EXLN_L,
                  AXIOM RL:..............;
```

Here, an axiom, R_I, describes the space of available states of the version I for $I = 1, 2, \ldots, L$.

7. PROGRAMMING IN ASSEMBLER-LIKE LANGUAGES: ANALYSIS OF SOFTWARE

We remind, first, some notions and ideas of programming in a very simple situation, which we call as *one-sorted functional programming language* (see, for example, [7] and [8]).

Let $\mathscr{F}(X, X)$ be the set of all functions on X into X and F be a subset closed under composition operation. Let P be a set of predicates on X, that is, $P \subset \mathscr{F}(X, \{0, 1\})$. We consider the set,

$$p \in P \circ F \Leftrightarrow \bigcup_{f \in F} \bigcup_{q \in P} p = q \circ f.$$

Let $\mathscr{P}$ be the closure of $P \circ F$ under the operations, $\wedge$ (AND) and $\neg$ (NOT). A *primitive pre-program* (PPP) is a subset of $F \times \mathscr{P}$, that is, $\text{PPP} \in \beta(F \times \mathscr{P})$. For the definition of primitive programs, we will use directed graphs without cycles. Let G be a directed graph without cycles with one root and one end vertex, that is, there is only one vertex with only output arcs and only one vertex with input arcs. We assume also that every vertex (except the end) has only one or two outputs. We associate an element from F with every vertex with one output, and an element from P with every vertex with two outputs. We call vertexes of the first type *functional vertexes*, but of the second type as *predicate vertexes*; we assign the value 0 or 1 to output arcs of a predicate vertex. We call a graph of this type as FP-*complex*. We associate a PPP with every FP-complex in the following way. An FP-complex has a set, say I, of paths from the root to the end. For $i \in I$, let $f^i_1, f^i_2, \ldots, f^i_{n_i}$ be the set of functional vertexes on the path i; $p^i_1, p^i_2, \ldots, p^i_{m_i}$ be the set of predicate vertexes on i; s^i_j for $j = 1, 2, \ldots, m_i$ be values of output arcs from p^i_j on i, and $f^i_1, f^i_2, \ldots, f^i_{k_j}$ be functional vertexes before p^i_j on i. We associate the function, $f^i_{n_i} \circ f^i_{n_i-1} \circ \cdots \circ f^i_2 \circ f^i_1$ and the predicate

$$\bar{s}^i_1 \cdot p^i_1 \circ f^i_{k_1} \circ f^i_{k_1-1} \circ \cdots \circ f^i_1 \wedge \bar{s}^i_2 \cdot p^i_2 \circ f^i_{k_2} \circ f^i_{k_2-1} \circ \cdots \circ f^i_1 \wedge \ldots \wedge \bar{s}^i_{m_i} \cdot p^i_{m_i} \circ f^i_{k_{m_i}} \circ f^i_{k_{m_i}-1} \circ \cdots \circ f^i_1$$

with a path i. Here $\bar{s}^i_1$ is 1 if s^i_1 is one and $\bar{s}^i_1$ is $\neg$ if s^i_1 is 0.

Therefore, we have a PPP associated with an FP-complex. We call the PPP associated with an FP-complex as a *primitive program* (PP).

FP-complexes represent flowcharts for PPs, and PPs are used for creating general programs by using different types of loops (see [7, 8]). We are not going to consider these constructions here. We saw that a one-sorted functional programming language is characterized by the set of functions, the set of predicates, and widely uses the composition operation. We consider here corresponding characteristics of languages, called *assembler-like languages*, in which real software systems are usually written. We will see that functions in languages of this type are defined and take values on spaces of available states with variable structures. We consider here only examples.

Consider, for example, the addition operation, AR, of the IBM/370 assembler. This operation is defined on the space of available states of two general registers, R_1 and R_2, current PSW, CR_PSW, and new program PSW, NP_PSW, and uses different parts of these subsystems depending on the result of the operation. The range of the operation uses general register R_1, different fields of current and old PSW, CR_PSW and OP_PSW, depending of the result of operation. We consider only the simplest memory organization of computer with blocked I/O, external, and machine-check interrupts. In this case, the operation works in the following way. The contents of R_2 are added to the contents of R_1, the result is stored in R_1. If the result is correct, then condition code and the new address are set in CC and IA fields of CR_PSW. If the result is incorrect, then condition code is set to 3, and the program interrupt takes place if it is enabled. We call the domain of the operation as DOM(AR). We have

```
        DOM(AR):                        VERSIONS(AR),
        VERSION(1):                     AXIOM(IN),
            1 ARC_DOM,
                    2 R1                ZF,
                    2 R2                ZF,
                    2 CR_PSW.IA         X(6),
            AXIOM IN: CR_PSW.IA IS EVEN;
        VERSION(2):                     AXIOM(IN),
            1 ARO_DOM,
                    2 R1                ZF,
                    2 R2                ZF,
                    2 CP_PSW            SPSW,
                    2 NP_PSW            SPSW,
            AXIOM IN: CR_PSW.IA IS EVEN;
    VERSIONS AR: IF pr2(R1 +F R2)=COR OR CR_PSW.PM(1)=0
            THEN VERSION(1) ELSE VERSION(2);
```

In the above space of available states, we used the notation, CR_PSW.IA IS EVEN instead of s(CR_PSW.IA) IS EVEN, where $s(x)$ denotes the state of the element x. In the same way, we used the expression $R_1 +_F R_2$. We used also S_{PSW} as the space of available states of PSW and additional field VERSIONS(AR) to describe conditions for versions. The range of the operation is (we call it RAN(AR))

```
        RAN(AR):                        VERSIONS(AR),
        VERSION(1):
            1 ARC_RAN,
                    2 R1                ZF,
                    2 CR_PSW.RAN,
                            3 CC        B(2),
                            3 IA        X(6);
        VERSION(2):
            1 ARO_RAN,
                    2 R1                ZF,
                    2 CR_PSW            SPSW,
                    2 OP_PSW            SPSW;
    VERSIONS AR: IF pr2(R1 +F R2)=COR OR CR_PSW.PM(1)=0
            THEN VERSION(1) ELSE VERSION(2);
```

The instruction, AR, transforms the first version of DOM (AR) into the first version of RAN (AR) if the result of addition is correct (COR) or fixed-point interrupt is disabled in the following way:

$$pr_{ARC\_RAN.R_1}(AR(DOM(AR)) = pr_1(ARC\_DOM.R_1 +_F ARC\_DOM.R_2);$$

$pr_{ARC\_RAN.CR\_PSW.RAN.CC}(AR(DOM(AR)) = 0, 1,$ or, 2

if $pr_1(ARC\_DOM.R_1 +_F ARC\_DOM.R_2)$ is correspondingly zero, negative, or positive;

$$pr_{ARC\_RAN.CR\_PSW.RAN.IA}(AR(DOM(AR)) = ARC\_DOM.CR\_PSW.IA + 2.$$

When the result gives an overflow and the interrupt is enabled, then the second version of DOM (AR) is transformed into the second version of RAN (AR) in the following way (we simplify our notations):

$$ARO\_RAN.R_1 = pr_1(ARO\_DOM.R_1 +_F ARO\_DOM.R_2)$$
$$ARO\_RAN.CR\_PSW = ARO\_DOM.NP\_PSW$$
$$ARO\_RAN.OP\_PSW.W = ARO\_DOM.CR\_PSW.W$$
$$ARO\_RAN.OP\_PSW.IC = 0008$$
$$ARO\_RAN.OP\_PSW.ILC = 10$$
$$ARO\_RAN.OP\_PSW.CC = 11$$
$$ARO\_RAN.OP\_PSW.IA = ARO\_DOM.CR\_PSW.IA + 2.$$

We can represent this operation in the form

$$AR\colon DOM(AR) \xrightarrow{(1,1),(2,2)} RAN(AR).$$

We would like to show now that functional constructions described above should be used for analysis of software systems. We consider now an example of a rather simple operating system realizing services on computer system described above. In general, an operating system is an organizational system of a computer system with an internal organization of resources and management providing services for users. We consider the simplest case of an operating system providing only services for users. Every user is represented (this is a model for a user) by a call, called supervisor call (SVC), for a service. Every service is a program which is characterized by an input information base and an output information base. We assume that the input information base contains a subbase describing parameters which must be passed to the service. Therefore, the domain for a service can be described as

```
DOM(SERVICE):
   1 SV_DOM(SERVICE),
        2 PARM              SUB,
        2 DOM1              SUB;
```

Here, SV_DOM (SERVICE), PARM and DOM1 are the names of the service, parameter base, and remaining information base correspondingly; we use also the keyword, SUB, indicating an information base, that is, a space of available states with variable structure. The range of a service can be described as

```
RAN(SERVICE):
   SV_RAN(SERVICE)          SUB;
```

We assume that the set of all services is described by the set, SVC $\subset X(2)$ (supervisor calls), and realized by SVC interrupts for every element from SVC. Therefore, the set of all services is described by the following information bases:

```
DOM(SVCS):              DO (I ∈ SVC),
      1 SV_DOM(I),
            2 PARM         SUB,
            2 DOM1         SUB;
                        END;
RAN(SVCS):              DO (I ∈ SVC),
      SV_RAN(I)            SUB;
                        END;
```

We used here the primitive relation of set theory, ∈, which can be described in the URL as a definition for a series of "OR" ($\vee$) operations, that is, $i \in X \Leftrightarrow (i = x_1 \vee i = x_2 \vee \cdots \vee i = x_n)$ if $X = \{x_1, x_2, \ldots, x_n\}$. We used also DO ... END notation to describe repetition. A service, I ∈ SVC, is a transformation, SVC(I): SV_DOM(I) → SV_RAN(I).

The input information base to our operating system consists of 16 general registers, old supervisor-call PSW, and specifically its interruption code field, and the set of domains for services, that is,

```
DOM(OS):
      1 OS_DOM,
            2 GREGS(0,15)          X(8),
            2 OSVC_PSW,         AXIOM(PSW),
                  3 (B)IC          X(4),
                  3 IC             '00' || SVC,
                  3 (A)IC          X(8),
            AXIOM PSW: OSVC_PSW ∈ $S_{PSW}$,
            2 DOM(SVCS)            SUB;
```

Here, (B)IC means before the fields, IC; (A)IC means after IC; '00' || SVC means the concatenation of the string '00' and elements from SVC; DOM(SVCS) ... SUB means subsystem (described above).

To describe the output information base of the operating system, we assume that the set of parameters of a service consists of two portions, COR(SERVICE) and ERR(SERVICE). If a parameter is correct (belongs to COR(SERVICE)), then the operating system creates the output for the corresponding service, restores the general registers and old SVC PSW (the old SVC PSW becomes the current PSW). If a parameter is incorrect, then the current PSW becomes the "BAD" PSW, that is, its value is X'0002000000000BAD'; this PSW puts computer into "wait" state. Therefore, the range of OS is

```
RAN(OS):                      VERSIONS(OS),
VERSION(1):
      1 OS_RAN,
            2 GREGS(0:15)        X(8),
            2 CR_PSW             $S_{PSW}$,
            2 RAN(SVCS)          SUB;
VERSION(2):
      CR_PSW                     $S_{PSW}$;
VERSIONS OS: IF SV_DOM.PARM(I) COR(I) FOR I ∈ SVC
                  THEN VERSION(1) ELSE VERSION(2);
```

We have operating system as a transformation from DOM(OS) into RAN(OS), that is,

$$\text{OS: DOM(OS)} \xrightarrow{(1,1),(1,2)} \text{RAN(OS)}.$$

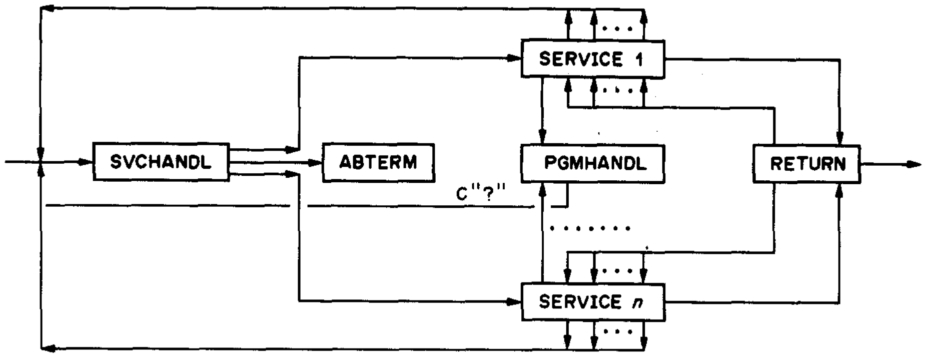


Fig. 2

In this transformation,

$$\text{OS\_RAN.GREGS}(0{:}\,15) = \text{OS\_DOM.GREGS}(0{:}\,15)$$
$$\text{CR\_PSW} = \text{OSVC\_PSW}$$

if parameters are correct and PSW(BAD) if parameters are incorrect;

$$\text{RAN}(\text{I}) = \text{SV\_RAN}(\text{I}) = \text{SVC}(\text{I})(\text{SV\_DOM}(\text{I})).$$

In the operating system, we permit a service to call another service.

We would like to realize the operating system as the composition of subprograms shown in Fig. 2. We described above services of this OS. We assume that a service may call another service (see discussion below). We assume that incorrect data for the service results in a program interrupt. Program interrupts are handled by PGMHANDL which contains only the call for the service, ABTERM. ABTERM simply loads the "BAD" PSW and may be realized in the IBM/370 assembler as

```
        LSPW    BAD
        DS      0D
BAD     DC      X'0002000000000BAD'
```

and PGMHANDL is

```
        SVC     C'?',
```

where C'?' is the code for ABTERM.

The RETURN subprogram works with the set, SA, of save areas, SAVE(I), from which it restores registers and PSW. It is convenient to assign codes to save areas, for example to create a table of their addresses and to use a relative address in the table as a code (see, for example, [9]). The table is called as SVCSAVE. We have

```
SVCSAVE        DO (I ∈ SA),
  SA_ADDR(I)       A;
               END;
```

Here A is the set of hexadecimal strings of the form 00XXXXXX representing an address of memory. Because SVC $\subset \mathcal{X}(2)$, and the number of elements in SA $\leq$ number of elements in SVC, we can assign the following codes to elements of SA: 0000 to I1, 0004 to I2, ..., $0M_1M_2M_3$ to IN where N is the number of save areas; here the decimal number corresponding to $0M_1M_2M_3$ is N * 4. We denote the set of codes as SACD. We have one-to-one correspondences, f_{SA}: SA $\rightarrow$ SACD and f_{ADDR}: SA $\rightarrow$ SA_ADDR: I $\rightarrow$ SA_ADDR(I).

The RETURN subprogram restores the contents of all registers which they have just before a service was called and restores also old SVC PSW at the SVC interrupt. We assume that register 14 contains the address of the corresponding save area. Therefore,

```
DOM(RETURN):                                  RAN(RETURN):
  1 RETURN_DOM,                                 1 RETURN_RAN,
                              (RETURN)
    2 R14            X(8),    ——————→             2 GREGS(0:15)  X(8),
  2 SAVES DO (I∈SA),                              2 CR_PSW       S_PSW;
    3 SAVE(I),
      4 OSVC_PSW     S_PSW,
      4 GREGS(0:15)  X(8);
        END;
```

where

$$\text{RETURN\_RAN.GREGS}(0{:}15) = \text{SAVE}(f^{-1}_{\text{ADDR}}(\text{R14})).\text{GREGS}(0{:}15);$$

$$\text{CR\_PSW} = \text{SAVE}(f^{-1}_{\text{ADDR}}(\text{R14})).\text{OSVC\_PSW}.$$

In the IBM/370 assembler language, this subprogram can be implemented as: Let SA be a dummy section describing the structure of save area, that is,

```
SA        DSECT
SAPSW     DS       CL8
SAREGS    DS       CL64
```

We assume also that SA was associated with register 14, that is, we issued

```
USING     SA, R14
```

then we have for RETURN

```
MVC     SVCOLD, SAPSW     RESTORE PSW
MVC     0, 15, SAREGS     RESTORE REGISTERS
LPSW    SVCOLD            RETURN CONTROL
```

Here, SVCOLD is OS_DOM.OSVC_PSW; at the end of RETURN we returned control to the caller.

We have to describe the SVCHANDL subprogram (routine). We assume that one and only one save area is assigned to every service. The set of save areas is described by the subsystem, SAVES. The consistent way of assigning a save area to a service will be considered below. Therefore, we have a function, FSA: SVC → SA, which assigns the save area, FSA(I), to a service I ∈ SVC. The SVCHANDL routine passes control to a service (to an SVC routine) by loading its PSW from the so-called SVCRTN table. Relative addresses of these PSWs are associated with their codes by using the SVCTABLE table:

```
SVCTABLE    DS      256AL2
        DO (n∈[1, N]),
            ORG     SVCTABLE + 2 * X'svc_n'
            DC      AL2((n − 1) * 8)
        END
```

Here we assumed $\text{SVC} = \{\text{svc}_1, \text{svc}_2, \ldots, \text{svc}_N\}$. The set,

$$\{\text{AL2}(0), \text{AL2}(8), \ldots, \text{AL2}((N-1) * 8)\}$$

of relative addresses of PSWs will be denoted by SVCCD.

The table, SVCRTN, of PSWs for all services is

```
1 SVCRTN                 DO (I∈SVCCD),
    2 PSW_SERV(I),        AXIOM(PSW),
        3 (B)SACODE       X(4),
        3 SACODE          SACD,
        3 ADDR            A,
    AXIOM PSW: PSW_SERV∈S_PSW;
                          END;
```

Here PSW_SERV.ADDR(I) is the address of the service $\text{I} \in \text{SVCCD}$.

The SVCHANDL routine stores the contents of all general registers and an old SVC PSW in the save area assigned to a called service. It loads also the address of a called service into register 1, the address of the corresponding save area into register 14, and current PSW becomes the address of the service, that is, it passes control to the service.

Therefore,

$$\begin{array}{l}
\text{DOM(SVCHANDL):}\\
\quad 1\ \text{SVCHANDL\_DOM,}\\
\qquad 2\ \text{GREGS(0:15)} \quad \text{X(8),}\\
\qquad 2\ \text{OSVC\_PSW} \quad \underline{\text{AXIOM}}\text{(PSW),}\\
\qquad\quad 3\ \text{(B)IC} \quad \text{X(6),}\\
\qquad\quad 3\ \text{IC} \quad \text{SVC,}\\
\qquad\quad 3\ \text{(A)IC} \quad \text{X(8),}\\
\qquad \underline{\text{AXIOM}}\text{: OSVC\_PSW} \in S_{\text{PSW}},\\
\qquad 2\ \text{SVCTABLE} \quad \underline{\text{SUB}},\\
\qquad 2\ \text{SVCSAVE} \quad \underline{\text{SUB}},\\
\qquad 2\ \text{SVCRTN} \quad \underline{\text{SUB}};
\end{array}
\xrightarrow{\text{SVCHANDL}}
\begin{array}{l}
\text{RAN(SVCHANDL):}\\
\quad 1\ \text{SVCHANDL\_RAN,}\\
\qquad 2\ \text{SAVES} \quad \underline{\text{SUB}},\\
\qquad 2\ \text{CR\_PSW} \quad S_{\text{PSW}},\\
\qquad 2\ \text{GREGS(1)} \quad \text{X(8),}\\
\qquad 2\ \text{GREGS(14)} \quad \text{X(8);}
\end{array}$$

Here:

$$\text{pr}_{\text{OSVC\_PSW}}(\text{SAVES(IC)}) = \text{SVCHANDL\_DOM.OSVC\_PSW}$$

$$\text{pr}_{\text{GREGS}}(\text{SAVES(IC)}) = \text{SVCHANDL\_DOM.GREGS}$$

$$\text{CR\_PSW} = \text{SVCRTN(SVCTABLE(IC))}$$

$$\text{GREGS(1)} = \text{pr}_{\text{ADDR}}(\text{SVCRTN(SVCTABLE(IC))})$$

$$\text{GREGS(14)} = \text{SA\_ADDR}(f_{\text{SA}}^{-1}(\text{pr}_{\text{SACODE}}(\text{SVCRTN(SVCTABLE(IC))}))).$$

The SVCHANDL routine can be realized in the IBM/370 assembler as (see also [9]):

```
STM    0, 15, TRAPSAVE
. . .
LM     10, 14, =5F'0'        INITIALIZE REGISTERS
IC     10, SVCOLD+3          LOAD SVC CODE IN R10
LH     10, SVCTABLE(10)      LOAD REL ADDRESS OF PSW IN R10
LA     10, SVCRTN(10)        LOAD ADDRESS OF PSW IN R10
LH     11, 2(10)             LOAD SACODE IN R11
A      14, SVCSAVE(11)       LOAD ADDRESS OF SAVE(I) IN R14
MVC    SAPSW, SVCOLD         SAVE OLD PSW
MVC    SAREGS, TRAPSAVE      SAVE REGISTERS
L      1, 4(10)              LOAD ADDRESS OF SERVICE IN R1
LPSW   0(10)                 PASS CONTROL TO SERVICE
```

We used here TRAPSAVE as working area.

We consider now the problem of assigning save areas to services so that the operating system to be a correct transformation. It is possible to assign different save areas to different services. But this solution is trivial. We would like to find out the minimal number of save areas and their assignment to services. First, we must have services which do not call other services; otherwise the operating system is incorrect, or we must use another procedure of assigning save areas to services (not one save area for a service). Second, if we consider sequences of services starting with a given one, then we must not have any service twice in this sequence. That is, the oriented graph with vertexes being services and arcs representing calls of services from services must not have cycles. Let N be the number of services. We have the following partition:

$$[1, N] = \bigcup_{i=1}^{k} \bigcup_{j=0}^{m_i} I_j^i$$

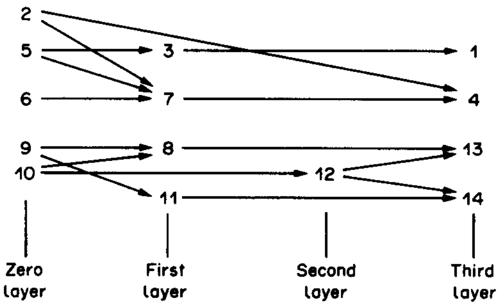
where

$$\bigcup_{j=0}^{m_i} I_j^i$$

for $i = 1, 2, \ldots, k$ are vertexes of a connected component of the graph; $I_0^i, I_1^i, \ldots, I_{m_i}^i$ are vertexes of the 0-layer, 1-layer, ..., m_i-layer of the i-connected component in canonical representation of the component. The number of save areas is $\max(m_1, m_2, \ldots, m_k)$. The assigning of save areas may be the following: for all vertexes on a given layer assign the same save area.

EXAMPLE

For 14 services with the following sequences of call 1; 2, 4; 2, 7, 4; 3, 1; 4; 5, 3, 1; 5, 7, 4; 6, 7, 4; 7, 4; 8, 13; 9, 8, 13; 9, 11, 14; 10, 8, 13; 10, 12, 13; 10, 12, 14; 11, 14; 12, 13; 12, 14; 13; 14; we have the following canonical representation of a graph:



We assign SAVE1 to calls 2, 5, 6, 9, 10; SAVE2 to calls 3, 7, 8, 11; SAVE3 to calls 12; SAVE4 to calls 1, 4, 13, 14.

REFERENCES

1. A. A. Tuzhilin, Application of category theory of structural sets to modelling of information bases of systems. *Math. Infor. Processing, Proc. Symp. appl. Math.* **34,** 119–233 (1986).
2. A. A. Tuzhilin, Category theory of structural sets with application to mathematical modelling and systems analysis. *Mathl Modelling* **7,** 27–48 (1986).
3. N. Bourbaki, *Theory of Sets, Elements of Mathematics.* Addison-Wesley, Reading, MA (1968).
4. A. A. Tuzhilin, Universal algebra of instructions for virtual memory management of the IBM/4341 computer. *Proc. IASTED Int. Symp. applied Simulation and Modelling ASM '85,* pp. 204–206 (1985).
5. A. A. Tuzhilin, Universal query language on finite primitive collections for mathematical modelling in information and computer sciences. *5th Int. Conf. Mathematical Modelling,* p. 172 (1985).
6. A. A. Tuzhilin and P. Zmigrodsky, Category of spaces of available states with variable structures, Specifications, implementation in IBM/370 assembler. *5th Int. Conf. Mathematical Modelling,* p. 173 (1985).
7. Z. Manna, *Mathematical Theory of Computation.* McGraw-Hill, New York (1974).
8. R. C. Linger, H. D. Mills and B. I. Witt, *Structured Programming: Theory and Practice.* Addison-Wesley, New York (1979).
9. S. E. Madnick and J. J. Donovan, *Operating Systems.* McGraw-Hill, New York (1974).