705

ISSN 0136-3549
0320-3409

TALLINNA TEHNIKAÜLIKOOLI

TOIMETISED

ТРУДЫ ТАЛЛИННСКОГО
ТЕХНИЧЕСКОГО УНИВЕРСИТЕТА

TRANSACTIONS OF TALLINN
TECHNICAL UNIVERSITY

DATA PROCESSING,
COMPILER WRITING,
PROBLEMS OF PROGRAMMING

TALLINN 1989

705

ALUSTATUD 1937

TALLINNA TEHNIKAÜLIKOOLI TOIMETISED

TRANSACTIONS OF TALLINN TECHNICAL UNIVERSITY

ТРУДЫ ТАЛЛИННСКОГО ТЕХНИЧЕСКОГО УНИВЕРСИТЕТА

UDK 681.3.06

DATA PROCESSING, COMPILER WRITING, PROBLEMS OF PROGRAMMING

Transactions of the Faculty of Economics LXXII

TALLINN 1989

Contents

ТАЛЛИННСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ТРУДЫ ТТУ № 705
Обработка данных. Построение трансляторов. Вопросы программирования. Экономика LXXII

На английском языке.

Vastutav toimetaja I. Amitan.

Kinnitanud TTÜ Toimetiste kolleegium 15.12.89

Trükkida antud 15.12.89. Formaat 60x90/16. Trükipg. 5,5.+0,25. Arvestuspg. 4,89. Trükiarv 350. Tellimus nr. 443/90. Hind 1 rbl
Tallinna Tehnikaülikool, Tallinn 200 108, Ehitajate tee 5,
TTÜ rotaprint, Tallinn 200 006, Koskla 2/9.

No 705

TALLINNA TEHNIKAÜLIKOOLI TOIMETISED
TRANSACTIONS OF TALLINN TECHNICAL UNIVERSITY

UDC 681.3

L. Vyhandu

FAST METHODS IN EXPLORATORY DATA ANALYSIS

Abstract

Two hard tasks of combinatorial optimization are handled using the theory of monotonic systems, including the problems of finding cliques of a graph and the best tournament ranking.

1. Introduction

This paper presents some ideas of theory and practice which have been useful for the author and his colleagues at Tallinn TU already for many years. Our concern was to create a system of effective exploratory data analysis methods. We hope in a sense we have been able to reach our goal.

We have found that to build up for a given and comparatively badly known object system a suitable inner ordering of objects two powerful tools can be used:

- frequency transformations of data,
- theory of monotonic systems developed by our group.

First we describe in a simple way the actions needed to use those tools. After that we demonstrate the effectiveness of our new methods for some well-known hard tasks.

2. Frequency transformations of nominal data

Let us have a N*M data table A with nominal data. We define a frequency transformation for A as follows. For every variable we take its histogram and change every value aij = h to its frequency fhj in the histogram. The row sums of those frequencies describe the conformity of objects in the data system.

The new matrix Z is called the frequency matrix of a

data matrix. If the numbers of categories for variables differ, we have to multiply frequencies by the number of categories aij → lj*zij. We get an equalization of sums of squares of frequencies for all columns of Z. (In practice we keep the original data naturally unchanged and use histograms of all variables directly in computations).

After that kind of simple transformation one way to get interesting results is to use the scale of conformity.

We define a measure of conformity for an object Oi as a sum

$$Si = \Sigma zij$$

The larger the sum Si the more conformity to that group's general behaviour is there for object i.

Instead of that very simple transformation one can use also more complicated transformations, e.g. the squares of frequencies, the so-called influence transformations (1), but it is adequate to explain our ideas.

3. Monotonic systems in data clustering

Classical clustering methods are fairly slow and some difficulties occur in the interpretation of clustering results. For the last 15 years our team has successfully used the monotonic systems theory for multidimensional data structuring. Here are some general ideas of this method.

Let us suppose that there is a system W with a finite number of elements. Each element has a numerical measure of its weight (influence) in the system. Further let us suppose that for every element $q \in W$ there is a feasible discrete operation which changes as well as the weight of q and the weights of any other element r of the system. If the elements in W are independent, then it is natural to suppose that there is no change in the values of other elements r.

System W is called monotonic, if the operation of weight change of any element $q \in W$ brings about changes in the weight levels of other elements only in the direction in which q itself changed.

To use the method of monotonic systems we have to meet three conditions.

1. There has to be a function P which gives a measure (weight) P(w) of influence for every element w of the monotonic system W.

2. There have to be rules f to recompute the influence of the elements of the system in case there is a change in the weight of one element.

3. The rules for influence recomputing have to be commutative.

These conditions leave a lot of freedom to the researcher to choose the influence functions and rules of influence change in the system. The only constraint we have to keep in mind is that the functions f and P have to be compatible in the sense that after eliminating all elements w of the system W the final weights of $w \in W$ must be equal to zero.

We study all $2^{|W|}$ subsets of the set W. Let $q \in H$ W and P+H(q) or P-H(q) be the value of function P on the element q. We define a kernel H+ (or H-) of a system W as a subset of W on which there is global maximum of function F of subsets H

$$F_{-}(H) = \min_{H} P_{-}H(q)$$

or global minimum of function $F_{+}(H) = \max_{H} P_{-}H(q)$.

The main theorem (4) guarantees the existence of the so-called determining sequence which defines the extremal subset of W.

We will demonstrate how to use this theory on data matrices.

Let us have a N*M nominal data matrix A. If we take the influence function for a data element aij as Pij = zij, then we can define different monotonic systems on our data matrix. We can choose the elements of the monotonic systems as follows:

- objects (rows of the data matrix),
- objects and variables (rows and columns of the data matrix),
- elements of the data matrix.

For simplicity a very brief description is given here without any programming shortcuts for the first case (object

clustering), using plus-influence.

A1. Find the sums $P(i) = \sum_{j=1}^{M} Pij$.

A2. Find $R = \max_{i} P(i)$ with index k.

A3. Copy object k as a new object into the system.

A4. Label object k as taken and calculate new influences P(i).

A5. Find $R' = \max_{i <> k} P(i)$ with index k'.

A6. If R' > = R then go to A3.

A7. All the objects from step A3 belong to the kernel.

A8. If there are more objects, eliminate the first kernel and go to A2.

Our practice has shown that for the interpretation it is best to use both objects and variables as elements of the monotonic system.

If the data are real numbers, we shall use as an influence function for a data element

$$g(aij) = aij + Ri + Cj,$$

where Ri is the sum of the i-th row and Cj - sum of the j-th column. For the i-th row we have an influence function $G(i) = \sum_{j=1}^{M} g(aij)$ and for the j-th column $G(j) = \sum_{i=1}^{N} g(aij)$.

For a multiplicative case one can take as an influence function

$$g'(aij) = aij\ (Ri-aij)\ (Cj-aij).$$

Other influence functions will be described in the following sections.

4. Graphs and monotonic systems

As a somewhat unusual example we shall demonstrate how to use the thecry of monotonic systems to study the structure of graphs. One can represent any graph as a data table, using graphs adjacency matrix, e.g.

	1	2	3	4	5	6	7	8
1:	1	0	1	0	1	0	1	1
2:	0	1	0	1	1	0	0	1
3:	1	0	1	0	1	0	0	1
4:	0	1	0	1	0	1	1	0
5:	1	1	1	0	1	1	1	0
6:	0	0	0	0	0	0	1	1
7:	1	0	0	1	1	1	1	0
8:	1	1	1	0	0	1	0	1

We have taken the adjacency matrix of an undirected symmetrical graph out of (2). As a first step we find the frequency counts for all variables:

0:	3	4	4	4	2	3	3	3
1:	5	4	4	4	6	5	5	5

As a measure of the influence of a given object (row) one can simply take the sum of frequencies of every variable (column) value (e.g. S1 = 5+4+4+4+6+3+5+5=36). After the calculation of all respective sums we get the following easy-to-use schema:

1:	36	35	32	28	24	19	15	8
2:	32	28	-	-	-	-	-	-
3:	34	34	30	27	24	20	15	-
4:	30	-	-	-	-	-	-	-
5:	36	32	30	27	22	-	-	-
6:	36	31	26	-	-	-	-	-
7:	36	31	28	22	-	-	-	-
8:	32	29	26	24	22	17	-	-

We will take the weakest object in the first column with the minimal sum and throw it out of the system (in our case row 4 with the sum equal to 30). It is easy to check

that for the given influence function the elimination of one object brings with it the diminishing of the influence values of other objects by the amount of equal variable values with the object we are just eliminating, e.g. object #1 has only one value equal to the values of object #4 therefore by conformity values diminish by 1 etc. Throwing the weakest object out on every step we get the elimination order of the rows 4,2,6,7,5,8,3,1. Using the symmetricity of the given data table we can reorder rows and columns as follows

	4	2	6	7	5	8	3	1
4:	1	1	1	1	0	0	0	0
2:	1	1	0	0	1	1	0	0
6:	1	0	1	1	1	1	0	0
7:	1	0	1	1	1	0	0	1
5:	0	1	1	1	1	0	1	1
8:	0	1	1	0	0	1	1	1
3:	0	0	0	0	1	1	1	1
1:	0	0	0	1	1	1	1	1

It is easy to see that the clique structure of the graph is nicely opened.

To find the maximal independent sets of the same graph we make only one change, namely in the adjacency matrix we take all main diagonal elements equal to zero. Taking into account the changes in 0/1 frequencies and using minusinfluence instead of plus-direction one gets the following:

1:	34	39	44	50	-	-	-	-
2:	34	40	-	-	-	-	-	-
3:	36	-	-	-	-.	-	-	-
4:	32	34	36	39	42	45	-	-
5:	30	32	32	33	36	39	45	-
6:	34	39	46	-	-	-	-	-
7:	34	39	44	48	50	-	-	-
8:	30	39	34	34	36	40	45	52.

If we have two or more equal values for a choice we take the one which has had the biggest change at the last step (e.g. rows 5 and 8 in the last schema).

After rearranging the rows and columns of the original matrix in the order given by the elimination process we get the following:

	3	2	6	1	7	4	5	8
3:	0	0	0	1	0	0	1	1
2:	0	0	0	0	0	1	1	1
6:	0	0	0	0	1	1	1	1
1:	1	0	0	0	1	0	1	1
7:	0	0	1	1	0	1	1	0
4:	0	1	1	0	1	0	0	0
5:	1	1	1	1	1	0	0	0
8:	1	1	1	1	0	0	0	0.

As we can see the structure of the maximal independence sets is nicely brought out.

5. Tournament ranking problem

Seeking the best tournament ranking is a good starting point for more general analysis of directed graphs. If you take a chess tournament table

	1	2	3	4	5	
1:	*	0	1	1	1	3
2:	1	*	0	.5	1	2.5
3:	0	1	*	0	.5	1.5
4:	0	.5	1	*	0	1.5
5:	0	0	.5	1	*	1.5,

all chessplayers do agree that the given ranking is the best one (using the so-called Berger system where all equal results are weighted by the amount of his/hers opponent's points). Closer analysis raises some doubts. First, all players lost to the next weaker player, second, the amount of wins by better and weaker players is equal (4 wins over and 4 wins under the main diagonal).

Kendall and Wei suggested another principle (5), namely to use that ranking which minimizes the number of violations where a weaker player defeats a stronger one.

Mathematical problem of renumbering the nodes of a given graph to minimize the number of the ones under the main diagonal is well-known and belongs to the class of exponentially complex tasks. We will propose a quick method to solve the problem by a combination of monotonic systems method with divide-and-conquer principle.

First we demonstrate the idea on our small tournament table. After that we represent the full algorithm.

We collect the tournament data into a simple table:

Player	To whom he lost	# of wins
1	2	3
2	3	2
3	14	1
4	15	1
5	12	1

If there would be a player who had no wins we could exclude him as the weakest. In our case there are no such players. Therefore we divide all players into two groups - weaker and stronger ones. We define a weaker player as the one having at least so many losses as wins (players 3, 4, 5). For those weaker players we put together a new schema

3	4	0
4	5	1
5	-	1

which has already a zero in it. So we can eliminate player 3, after that player 4 loses his/her only win and we get an order 3, 4, 5 (the weakest comes first) for weaker players.

For better players we get

1	2	0
2	-	1

That gives us a sequence 3, 4, 5, 1, 2 from the weakest to the strongest player.

Our new chess table would be as follows

2	1	5	4	3	
2	*	1	1	.5	0
1	0	*	1	1	1

5:	0	0	*	1	.5
4:	.5	0	0	*	1
3:	1	0	.5	0	*

In that order we have only one (irreducible) violation.

6. A minimum violation ranking algorithm for a tournament

A tournament can be described as a non-symmetrical graph with N nodes as players and an edge (i,j) representing a win of the player i over the player j. An adjacency matrix for a tournament A = (aij) is defined as aij = 1 if i defeated j, otherwise it equals 0.

Ranking of players in a tournament is defined as the next task on a graph (3).

A non-symmetrical graph is given. Renumber the nodes of that graph so that in the graph's adjacency matrix A(N,N) the number of the ones under the main diagonal will be minimal.

An algorithm for the requested renumbering works as follows:

A1. (Elimination of nonessential elements).

a) Make all A(I,I) = 0, I=1,...,N.

b) For all pairs of subscripts (I,J) where A(I,I)= =A(J,I) change A(I,J) and A(J,I) to zero (those elementary cycles cannot be eliminated and in the reordered matrix all those ones will be there).

A2. For every node I calculate the number of incoming and outgoing edges (resp. C(I) and R(I)).

A3. If there are nodes with C(I)=0 go to step A4, otherwise to step A5.

A4. Take any node with C(I)=0 and put that node into the new sequence of nodes. For every outgoing edge (I,K) calculate C(K)=C(K) - 1. Go to step A3.

A5. Divide the nodes not yet included into the new sequence into two parts. To the first active part belong all those nodes I for which Z(I)=R(I)-C(I) > =0. Put together a new adjacency matrix for that subgraph. Go to step A6.

A6. 1) If there are isolated nodes in that subgraph for which R(I)=C(I)=0 then those nodes are not yet included into the new sequence of nodes.

2) If all Z(I)=0, but R(I) and C(I) <> 0, then for that subgraph one can use any subsequence of nodes touring all its nodes as in the Hamilton cycle. The problem of including the subgraph into the new sequence of nodes is solved as follows. If that subgraph is at the very end of the graph, i.e., if after including those nodes into the sequence there are no free nodes, one can exclude at random a node from the subgraph and go on as in the main algorithm. That ends the recordering procedure.

In the case where that subgraph's cycle does not exclude all nodes of the whole graph, one puts together a new subgraph from all free nodes and goes to step A2.

3) If not all Z(I) for a subgraph equal to zero one has to act as in step A3 using it recursively up to the moment when the subgraph is eliminated. For all free nodes put together a new subgraph. Go to step A2. If all nodes of the graph are included into the new sequence end the procedure.

A7. For that list of nodes count how many violations there are in that nodes ranking. You get the number of significant ones under the main diagonal in the newly ordered adjacency matrix. (Naturally one has to include all the elements excluded in step A1 into the final list).

A8. Repeat the same procedure from step A3 on, changing everywhere the roles of R(I) and C(I). The new list of nodes is taken in reverse order.

A9. We take the order of nodes which has the minimal number of ones under the main diagonal as the solution of our task.

References

1. Vyhandu L. Express methods of data analysis // Transactions of Tallinn TU. 1979. No 464. P. 21-35 (in Russian).

2. Loukakis E. A new backtracking algorithm for generating the family of maximal independent sets of a graph // Comp. and Math. with Appls. 1983. Vol. 9. No 4. P. 583-589.

3. Ali I. a.o. On the minimum violations ranking of a tournament // Management Science. 1986. Vol. 32. No 6. P. 660-672.

4. Mullat J., Vyhandu L. Monotonic systems in scene analysis // Symposium. Mathematical Processing of Cartographic Data. Tallinn, 1979. P. 63-66.

5. Kendall M.G. a.o. On the method of paired comparisons // Biometrika. 1940. No 31. P. 324-345.

L. Võhandu

Kiirmeetod uurimuslikus andmeanalüüsis

Kokkuvõte

Artiklis käsitletakse kahte rasket probleemi kombinatoorse optimeerimise vallast - graafide avamist ja turniiride parema järjestuse leidmist.

No 705

TALLINNA TEHNIKAÜLIKOOLI TOIMETISED
TRANSACTIONS OF TALLINN TECHNICAL UNIVERSITY

UDC 681.3.06

L. Vyhandu

HOW TO GENERATE STATEMENTS FROM EXAMPLES

Abstract

The problems of inductive learning are handled using the theory of monotonic systems. An easy to use and quick schema is developed to generate statements from examples.

1. Introduction

This paper handles some problems of the descriptive generalization in conceptual inductive learning. As demonstrated by R.S. Michalski [3] this area includes such topics as automated theory formation, discovery of relationships in data or an automated construction of taxonomies. We describe very shortly some simplest applications of the theory of monotonic systems to statement generation.

2. Frequency transformations of nominal data

Let us have a N*M data table A with nominal data. We define a frequency transformation for A as follows. For every variable we take its histogram and change every value $a_{ij} = h$ to its frequency f_{hj} in the histogram. The row sums of those frequencies describe the conformity of objects in the data system [1].

The new matrix Z is called the frequency matrix of a data matrix. If the number of categories for variables differs, we have to multiply frequencies by the number of categories $a_{ij} \rightarrow l_j * z_{ij}$. We get an equalization of sums of squares of frequencies for all columns of Z. (In practice we keep the original data naturally unchanged and use histograms of all variables directly in computations).

After that kind of simple transformation one way to get interesting results is to use the scale of influence.

We define a measure of conformity for an object O_i as a sum

$$S_i = \sum z_{ij}.$$

The larger the sum S_i the more conformity object i shows to the general behaviour of that group.

Instead of that very simple transformation one can also use more complicated transformations, e.g. the squares of frequencies, the so-called influence transformations [1], but it is good enough for explanatory purposes.

3. Monotonic systems on data matrices

Using the frequency transformation one can build up very simply monotonic systems on data matrices and open the inner structure of those matrices [1, 2]. To use that technique for inductive learning we do not need anything special. We just take all given examples and describe them with a system of variables so that all important properties would be specified well enough. Having built up a data matrix we can use the frequency transformation with a chosen influence function. As a next step we use a computationally very simple algorithm to organize data structurally.

For simplicity we describe here very briefly but without any programming shortcuts only how to cluster the object system into kernels (clusters).

A1. Find the sums $P(i) = \sum_{j=1}^{M} P_{ij}$.

A2. Find $R = \max_i P(i)$ with index k.

A3. Copy object k as a new object into the system.

A4. Label object k as taken and calculate new influences P(i).

A5. Find $R' = \max_{i<>k} P(i)$ with index k'.

A6. If R' >= R then go to A3.

A7. All the objects from step A3 belong to the kernel.

A8. If there are more objects, eliminate the first kernel and go to A2.

All kernels have to be interpreted as structural unities we are looking for. In the case of many-object examples one has to be careful so that all interpretable kernels would include all examples.

4. Monotonic systems in learning

To demonstrate how the method works we shall take a short and well-known example by F. Hayes-Roth and J. McDermott. They introduced in [4] a method for inducing knowledge by abstraction from a sequence of training examples. Their examples are:

E1:

{{ TRIANGLE: a, SQUARE: b, CIRCLE: c } ,
{ LARGE: a, SMALL: b, SMALL: c},
{ INNER: b, OUTER: a},
{ ABOVE: a, ABOVE: b, BELOW:}c ,
{ SAMESIZE: b, SAMESIZE: c}}

E2:

{{ SQUARE: d, TRIANGLE: e, CIRCLE: f},
{ SMALL: d, LARGE: e, SMALL: f},
{ INNER: f, OUTER: e},
{ ABOVE: d, BELOW: e, BELOW: f},
{ SAMESIZE: d, SAMESIZE: f}}

E3:

{{ SQUARE: g, CIRCLE: h, CIRCLE: i},
{ SMALL: g, LARGE: h, SMALL: i},
{ INNER: i, OUTER: h},
{ ABOVE: g, BELOW: h, BELOW:}i .
{ SAMESHAPE: h, SAMESHAPE: i},
{ SAMESIZE: g, SAMESIZE: i}}

Their abstractions are [4]:

There are three objects, including a small circle and a small square. The square is above the circle. The third object is large. To use the method of monotonic systems we represent the examples as a data table:

	1	2	3	4	5	6	7
a	1	1	2	2	2	0	0
b	1	2	1	1	2	1	0
c	1	3	1	0	1	1	0
d	2	2	1	0	2	1	0
e	2	1	2	2	1	0	0
f	2	3	1	1	1	1	0
g	3	2	1	0	2	1	0
h	3	3	2	2	1	0	1
i	3	3	1	1	1	1	1

The values of the variables are as follows:

1 - the number of the example;
2 - type (1=triangle, 2 = square, 3 = circle);
3 - size (1 = small, 2 = large);
4 - in/out (1 = inner, 2 = outer, 0 = none of them);
5 - 1 = below, 2 = above;
6 - samesizeness (1 = yes, 0 = no);
7 - sameshapeness (1 = yes, 0 = no).

As a first step we create the frequency table of values for the examples

0:	0	0	0	3	0	3	7
1:	3	2	6	3	5	6	2
2:	3	3	3	3	4	0	0
3:	3	4	0	0	0	0	0

As the next step we find the weights for all objects taking the sums of the frequencies of values of objects. These sums are given in the second column of the following schema

a	25	22	17	-	-	-	-	-	-
b	32	32	31	28	25	21	17	-	-
c	34	32	30	28	24	19	-	-	-
d	32	32	30	28	26	22	18	13	-
e	26	22	-	-	-	-	-	-	-
f	34	32	29	28	23	-	-	-	-
g	32	31	30	28	25	22	18	13	7
h	23	-	-	-	-	-	-	-	-
i	29	25	24	24	-	-	-	-	-

We choose the object with the minimal weight (in our case object h with the weight 23). After that we exclude that object out of the system. It is easy to prove that the weights of all other objects change by the amount of equal values with the exluded object (e.g. object i has four equal values of variables with object h, therefore the exclusion of the object h diminishes the weight of object i from 29 to 25). Correcting all weights we choose again an object with minimal weight. In our case we have twice the value 22 (objects a and e). As a minimum we choose that object which was more alike to the excluded one (object e). After having excluded all objects we rearrange the data matrix in the order of exclusion

h	3	3	2	2	1	0	1	23
e	2	1	2	2	1	0	0	22
a	1	1	2	2	2	0	0	17
i	3	3	1	1	1	1	1	24
f	2	3	1	1	1	1	0	23
c	1	3	1	0	1	1	0	19
b	1	2	1	1	2	1	0	17
d	2	2	1	0	2	1	0	13
g	3	2	1	0	2	1	0	7

Interpretation of that rearrangement is easy to get. The data are through their determining sequence monotonic parts automatically structured into two groups {h, e, a} and {i, f, c, b, d, g}. It is important to remember that we can generalize only those statements which are true for all examples. Our first group {h,e,a} has as a first variable all 3 different example numbers. Therefore we can collect a conjunctive statement out of three first lines:
there exists a large object which is outside and does not have another object of the same size.

The second group creates a statement: there are two small objects which are of the same size; the small square is always above and the small circle is always below.

Those two statements are most important ones which do cover all data rows about examples. To analyze the bigger group more carefully one can eliminate two constant variables (#3 and #6) for that group and then two more statements are obtained.

Inner objects are always small and are either circles or squares (group {i,f,b}).

There are always objects which are not inner or outside and which are small, of the same size and do not have an object of the same shape (group {c,d,g}).

The last statement is especially interesting, because negative patterns are often more conclusive than the positive ones. However, no mention was made of that statement in [4].

References

1. Vyhandu L. Express methods of data analysis // Trans. of Tallinn TU. 1979. No 464. P. 21-35 (in Russian).

2. Mullat J., Vyhandu L. Monotonic systems in scene analysis//Symposium. Mathematical Processing of Cartographic Data. Tallinn, 1979. P. 63-66.

3. Michalski R.S. A theory and methodology of inductive learning // Artificial Intelligence. 1983. 20. P. 111-161.

4. Hayes-Roth F., McDermott J. An interference matching technique for inducing abstractions // CACM, 1978. 21. P. 401-411.

L. Võhandu

Näidetest väidete genereerimine

Kokkuvõte

Artiklis käsitletakse monotoonsete süsteemide teooria rakendamist induktiivsel õppimisel. Esitatakse lihtne ja kiire arvutiskeem näidetest väidete genereerimiseks.

No 705

TALLINNA TEHNIKAÜLIKOOLI TOIMETISED
TRANSACTIONS OF TALLINN TECHNICAL UNIVERSITY

UDC 681.3.06

J. Tepandi,
S. Trausan-Matu

TESTING OF OBJECT-ORIENTED PROGRAMS

Abstract

The main features and typical applications of object-oriented programs (OOP) are discussed. The statement and branch adequacy testing criteria for OOP are proposed. The problems associated with the specification-based testing of OOP are considered.

1. Introduction

Object-oriented programming (OOP) is widely discussed and applications with high reliability requirements [11, 19] are becoming to benefit from it. Nevertheless, testing of object-oriented programs has not received sufficient attention. At the same time, there exists a large amount of expertise in the area of conventional testing of software systems [9, 10, 18]. The goal of the paper is to apply this expertise to the testing of object-oriented programs.

To do this, we shall first consider briefly the main features and some typical applications of OOP. OOP can be characterized as a combination of abstract data types and inheritance [8]. An object may have a number of components (we shall call them slots) and can respond to a number of messages. For each message an object has associated a method, which is the code to be executed as a result of the message. The main features of OOP are [1]:

- encapsulation of specific knowledge concerning the implementation of objects' behaviour;
- inheritance of slots and methods;
- message passing;
- dynamic binding;

- dynamic storage management.

Typical application areas of OOP, determining the object-oriented program life cycle, are:

- artificial intelligence (especially expert systems) [2, 3, 14, 16, 19, 23];
- graphics [1, 19, 21];
- flexible simulation [19];
- scheduling [19];
- planning [19];
- monitoring [12, 19];
- software engineering [3, 11, 13];
- data base semantic models [15].

Historically, the first language, comprising object-oriented features, was SIMULA, with its class construct. One of the first and best known object-oriented languages is SMALLTALK [1] which intensively uses the object concept for implementing a powerful environment. In SMALLTALK everything is an object, starting from numbers and finishing with graphic concepts like windows. All processing is fulfilled by sending messages to objects.

A very important class of OOP languages was designed for artificial intelligence applications. We can mention LOOPS [5], KEE (provided by Intellicorp) [12, 13, 14, 19], Knowledge Craft (provided by Carnegie Group) [19], Flavors, XRL [4]. They are generally written in LISP and offer a powerful development environment. All of them (except Flavors) combine OOP with other knowledge representation paradigms like production rules, logic programming, active values. They allow also using LISP functions.

2. Program-based testing methods

There exist several program text based testing adequacy criteria [18]: statement adequacy (all the program statements must be executed), branch adequacy (all the edges in the program flow diagram must be traversed), path adequacy (all the possible paths through the program must be executed), etc. We may reformulate them for OOP as follows. According to the statement adequacy criterion, all statements in each method and in each

demon must be executed, all slots, defined in each object, must be accessed, each declaration of inheritance must be used at least once. Due to the code sharing by inheritance, the amount of tests corresponding to the statement adequacy criterion, may be significantly reduced with respect to the testing of the same application, implemented in a conventional programming language.

To consider the branch adequacy criterion, we accept here its analogy for the OOP, stating that all the different possibilities of code activation (i.e. branches) must be considered. Therefore, all the statement adequacy tests should be performed. In addition, the following specific tests should be included:

- for the code of methods and demons, branch adequacy testing should be performed as in conventional programs;
- all the possible different types of messages to an object should be sent;
- the same message should be sent to all the objects to which it can respond;
- all the inheritance links must be used, including implicit transitive links;
- if the language allows multiple inheritance and method combination, then all possible exits from the composed methods (obtained by applying the combination function to the conflict set of inherited methods) must be exercised;
- all the different inheritance modes must be executed (e.g. replacing or overriding a value of a slot).

In addition, there exist several new problems, associated with testing artificial intelligence applications, comprising dynamic modification of programs (e.g. development of a new system, starting from a generic application, the process of knowledge acquisition or refinement). These modifications may differ in persistency (from the modifications kept during one run of the program to the "modifications", serving as a user-defined new application), extent (from minor modifications to new systems), and regularity (from well-defined additions to making any kind of program modifications). Thus the complexity scale of the modifications starts with short-term, minor and well-defined ones (e.g. adding a new slot which inherits methods and eventually

uses a combination of methods, or adding a clause for a person into Prolog database), that can be handled as usual data structure testing. On the other side, there are long-term and complex new systems (e.g. dynamically generated, starting from a generic application) that must be tested on their own. The situation between these two extremes seems to be an interesting area of research.

3. Specification-based testing of object-oriented programs

We know that the OOP is often characterized by an evolutionary life-cycle. In such a case there may be no specifications at all; and, if there are any, they follow the same evolutionary pattern. In view of this, is there a need for specification-based testing of object-oriented programs? We will argue that there is such a need - for the following reasons.

1. The argument, given above, was not quite correct. Some form of specification (e.g. determining the application area, the allowed response times, the cost and realibility requirements, ease of operation, and many others) always exists. Besides, recent results [11] indicate that specification methods, developed for conventional programming [24], may also be useful developing object-oriented programs. In fact, modularity and encapsulation of OOP facilitate the implementation phase which starts from such a specification.

2. It is useful to have specifications for generic tasks [6, 7, 23] or generic application shells [19], thus simplifying the implementation of a whole class of applications. Combining generic specifications with specifications of peculiar features, one may tune the generic program to obtain a specific application.

3. There exist fundamental differences between what the program does and how it is accomplished. However declarative the current high-level languages (including OOP) are, the specification, written, for example, in natural language or high-level logics, allows more flexibility [20].

4. Many problems can be easily expressed in the logic languages, mentioned in the previous point, but their implementation is inefficient or even untractable. Thus, these

languages may be used for specification and testing, but not for implementation.

5. Two descriptions of one task (i.e. through a specification and a program) can improve reliability and ease testing.

6. The specification may be not complete to obtain the results (for example, the input/output specifications, given on Jackson formalism). For this reason, it cannot be identified with the program. Nevertheless, this kind of specification may be easy to extract and useful for program development and testing.

7. There exist methods peculiar to specification based testing (e.g. testing, based on equivalence classes, boundary values, etc.) This experience is useful for programs with high reliability requirements. However, it is possible only when a specification exists.

In the previous discussion we referred to some possible specification languages. There are several approaches to test generation using these languages (algebra, Prolog, predicate calculus) [22]. So now let us ask whether OOP languages can be used for specification. In [17, 20] the following criteria are stated. It must be possible to: say that something has a certain property without saying what thing has this property (existential quantification); say that everything in a certain class has a certain property without saying what everything in that class is (universal quantification); say that at least one of two statements is true without saying which is true (OR); explicitly say that something is false (NOT); either settle or leave open to doubt whether two non-identical expressions name the same object (equality). OOP allows to satisfy the first two requirements entirely and supports partially the satisfaction of the following two requirements. Also, it is essential, that OOP provides many features that make it attractive as a specification language (encapsulation as a form of abstraction objects as conceptual units, inheritance and object hierarchies as means for representing interconcept relations, etc).

The next important question is: if the specification and implementation are given in the same language, is there

a difference between specification-based and program-based testing? Namely, in such a case the specification tends to be a part of the program. The program itself is more complex and includes also procedural (e.g. bodies of methods, written in a procedural language) or declarative-procedural (e.g. rules or logic programming) components. So, the specification-based testing seems to be included into the program-based testing. This question deserves further elaboration, but the points 1-7 from the previous discussion seem to indicate that the specification may be necessary, and actually used to improve testing.

In particular (cf. point 1) there exist requirements that cannot be readily expressed in logic, OOP or any other currently used formalism. The situation seems similar to the incompleteness of the first-order theories. Also (cf. point 7), specifications enable development of knowledge-based systems that make use of the testing experience currently available.

Acknowledgement

The authors would like to thank the Basic Laboratory of AI of the Institute of Technical Cybernetics, Slovak Academy of Sciences, for support and pleasant working environment.

References

1. Anderson B. Object-oriented programming // Microprocessors and Microsystems. Oct. 1988. Vol. 12. No 8. P. 433-442.

2. Barbuceanu M. An object centered framework for expert systems in CAD // J.S. Gero (ed.). Knowledge engineering in CAD. North Holland, 1985. P. 223-253.

3. Barbuceanu M., Trausan-Matu S., Molnar B. Integrating declarative knowledge programming styles and tools in a structured object AI environment // Proceedings. Tenth International Joint Conference on Artificial Intelligence. Milan, Italy, 1987. P. 563-568.

4. Barbuceanu M., Trausan-Matu S., Molnar B. The XRL2 manual. Institute for Computer and Information Sciences. Bucharest, Romania, 1988.

5. Bobrow D.G., Stefik M. The LOOPS manual, T.R. KB-VLSI-81-13, Xerok Palo Alto Research Center, 1981.

6. Chandrasekaran B. Generic tasks in knowledge based reasoning: high level building blocks for expert systems design, IEEE Expert, Fall, 1986. P. 23-30.

7. Chandrasekaran B. Towards a functional architecture for intelligence based on generic information processing tasks // Proceedings. Tenth International Joint Conference on Artificial Intelligence. Milan, Italy, 1987. P. 1183-1193.

8. Danforth S., Tomlinson C. Type theories and object-oriented programming // ACM Computing Surveys. March 1988. Vol. 20. No 1. P. 29-72.

9. EWICS, Guideline for verification and validation of safety related software - a report of a TC7 systems reliability, safety and security committee. Computers & Standards 3. 1984. P. 91-99.

10. EWICS. Techniques for verification and validation of software-related software. Computers & Standards 4. 1985. P. 101-112.

11. Falk H. CASE tools emerge to handle real-time systems // Computer Design. Jan. 1988. P. 53-74.

12. Fikes R., Kehler T. The role of frame-based representation in reasoning // Communications of the ACM. Sept. 1985. Vol. 28. No 9. P. 904-920.

13. Filman R. Retrofitting objects//Proceedings OOPSLA'87.

14. Filman R. Reasoning with worlds and truth maintenance in a knowledge-based programming environment // Communications of the ACM. Apr. 1988. Vol. 31. No 4. P. 382-401.

15. Hull R., King R. Semantic database modeling: survey, applications and research issues // ACM Computing Surveys. Sept. 87. Vol. 19. No. 3. P. 201-260.

16. Motta E., Eisenstadt M., Pitman K., West M. Support for knowledge acquisition in the Knowledge Engineer's Assistant (KEATS) // Expert Systems. Feb. 1988. Vol. 5. No 1. P. 6-27.

17. Moore R. Reasoning About Knowledge and Action // Technical Note 191. Artificial Intelligence Center. SRI International. Menlo Park, CA, 1980.

18. Myers G.J. The Art of Software Testing. New York: Wiley, 1979.

19. Richer M. An evaluation of expert system development tools, report KSL 85-19, Stanford University, June 1986.

20. Reiter R. Foundations for knowledge-based systems// Information Processing 86, H.-J. Kugler (ed.). Elsevier Science Publishers B. V., North-Holland, 1986. P. 663-668.

21. Stefik M., Bobrow D. Object-oriented programming: themes and variations // AI Magazine. Winter, 1986, 6 (4). P. 40-62.

22. Tepandi J. J. A knowledge-based approach to the specification-based program testing // Computers and Artificial Intelligence. Jan. 1988. Vol. 7. No 1. P. 39-48.

23. Trausan-Matu S., Barbuceanu M. Generic knowledge processing architectures for CAD in civil engineering // Proceedings of INFOTEC'88. Bucharest, Romania. Sept. 1988 (in Romanian).

24. Yau S.S., Tsai J. J.-P. A survey of software design techniques // IEEE Trans. Software Eng. June 1986. Vol. SE-12. No 6. P. 713-721.

J. Tepandi, S. Trausan-Matu

Objektorienteeritud programmide testimine

Kokkuvõte

Artiklis esitatakse objektorienteeritud programmide testimise põhimõtted. Käsitletakse testimist programmi teksti alusel (kõikide lausete katmise meetod, kõikide lahenduskäikude katmise meetod) ning testimist programmi spetsifikatsiooni alusel.

No 705

TALLINNA TEHNIKAÜLIKOOLI TOIMETISED
TRANSACTIONS OF TALLINN TECHNICAL UNIVERSITY

UDC 681.3.06

J. Tepandi, P. Parmakson

CHOOSING AN EXPERT SYSTEM SHELL

Abstract

The principles for choosing an expert system shell for a specific application are proposed. A description of an expert system for such a selection is given.

At present many expert system shells (ESS) are accessible. Some of them are described in [1-4]. Often it is not so easy to choose the most suitable one. Meanwhile, the correct choice of ESS can substantially simplify the construction of expert system (ES), while an erroneous one would prolong ES development considerably or produce an unsuitable system. Below we will present the principles for choosing ESS as well as describe ES for choosing ESS, developed consistent with those principles.

It should be noticed that the use of special expert-system development tools, although very time-saving, is not crucial. Highly usable expert systems for small-sized problems can be developed in any high-level language. These languages may be capable of performing arithmetic calculations and drawing graphics, not always available even in quite costly ESS. Artificial intelligence languages such as Prolog or Lisp are widely regarded as particularly appropriate for expert-system development. To perform a given task, a suitable high-level language tends to be easier to use than a low-level language; however, low-level languages are more adaptable and powerful when mastered.

Principles of choosing ESS. When working out the principles of choosing ESS we were guided by previous experience in the field of ES and program system construction. We also took into account some results in [5] concerning the evalu-

ation of the usefulness of various ES development tools and in [6] concerning the classification and the selection of a database query language.

To choose ESS, it is necessary to take into consideration several groups of factors. We will discuss them one at a time. The order of discussion is due to practical reasons - we believe that such an order allows to reduce the expenses for ESS selection.

First of all, nature of the task is to be considered in order to find out if the task is altogether solvable by means of ESS. The following criteria can be suggested here:

- if there is an exact algorithm for solving the task then ES will not be needed as ordinary programming language is sufficient;

- if the human expert solves the task in less than half an hour then the task is probably too simple to be solved by means of ES (although for educational purposes it can also be useful);

- if a man spends more than an hour solving the task, then it is probably too complex for modern computers;

- ES can only be created in case there is no expert for the given task.

Analysis of the task can prove the use of ESS unnecessary already at the start, thus avoiding further waste of time. If the task obviously suits ES or if the object of interest is not so much the product itself than the principles of ES development, then the next significant factor in the real situation is the accessibility of ESS and the information concerning it. It might turn out, that only one system is accessible - in that case the problem is almost solved and one will only have to check the consistency of the ESS available with the remaining requirements. Other significant criteria of this type are: (1) the kind of strategy the firm is carrying out in the field of computer hard- and software purchasing, (2) the price of the system or knowledge, (3) the existence of system documentation, (4) the availability of experts who have been working with the system, (5) any possible relations to some ESS development team.

The next important factor is the preparation level of

the person working out the ES and of its future user. For example, it is important to have information about the language the developer knows and about the interface languages acceptable to the user. If the developer has no experience working with ESS he may begin with more simple ESS of prefer ESS with powerful knowledge base editors.

It is also essential if the system will be operated by a regular staff or will be used in the mode of ad hoc queries asked by casual users.

In addition it is necessary to deal with the problem that is to be solved in more detail from the standpoint of implementation. The choice of ESS can be due to the following considerations: necessity to perform arithmetic calculations (not all ESS are offering such an opportunity), necessity to have a link with external files, necessity to use external (sub)programs (e.g. when there are program libraries and it is expedient to manage the processing using ESS), necessity to work with fuzzy knowledge, necessity of computer graphics in the interface, necessity to work in real-time, necessity to work with distributed data bases, necessity of high work efficiency, supposed dimension of knowledge base, model of the computer used, etc.

Very important is the assumed future use of the ES to be created, including its purpose, i.e either for research or for company use, the countries where it is to be used, the assumed frequency of updating knowledge base, etc.

Expert system ESVES for choosing an ESS. In the Basic Laboratory of Artificial Intelligence, Institute of Technical Cybernetics, Slovak Academy of Sciences in Bratislava several different ESS are accessible. They differ in possibilities, types of tasks to be solved and the knowledge level required by the expert system developer. The majority of research workers in ITC are insufficiently informed of types of tasks where these ESS can be used. The system called ESVES, developed on the above-stated principles, is intended to make up for that deficiency. It gives answers to the following questions.

- What ESS can be found in Basic Laboratory?
- Which ESS can one use to become acquainted with the problems of expert systems?

- Is my task suitable to be solved with the help of an expert system?

- Which ESS serve best for solving my tasks?

If there are any suitable ESS, then the conclusion from system ESVES is given in the form of a list of recommended ESS, in the order of confidence values of recommendations. But if the task is not suitable for accessible ESS, then the relevant information is given. In that way a research worker at ITC who is interested in building expert systems, can choose the ESS suitable for one's own task.

System ESVES can be used in various projects of Basic Laboratory as well as in other development activities connected with expert systems.

One of the authors, J. Tepandi, would like to express his gratitude to the leadership of ITC and Basic Laboratory for offering computing resources and favourable working conditions.

References

1. Koov M.J., Haav H.-M. H. Expert system and data bases for MicroPRIZ system// Personal Computers and their Applications. Tallinn: Valgus, 1986. P. 35-36.

2. GESMI - an expert system generator. User's manual. Sofia, 1987.

3. Ho Tu Bao. On the design and implementation of an expert system using the inference engine COTO // Computers and Artificial Intelligence. 1987. Vol. 6. No 4. P. 297-310.

4. Tepandi J. A knowledge-based approach to the specification-based program testing // Computers and Artificial Intelligence. 1988. Vol. 7. No 1. P. 39-48.

5. Grafton C., Permaloff A. Microcomputer expert systems // Social Science Microcomputer Review. 1987. Vol. 5. No 4. P. 547-557.

6. Jarke M., Vaasiliou Y. A framework for choosing a database query language // ACM Computing Surveys. 1985. Vol. 17. No 3. P. 313-340.

J. Tepandi, P. Parmakson

Instrumentaalse ekspertsüsteemi valik

Kokkuvõte

Artiklis esitatakse instrumentaalsete ekspertsüsteemide valiku põhimõtted, lähtudes rakenduste iseloomust. Arvessevõetavate tegurite hulka kuuluvad lahendatava ülesande tunnusjooned, organisatsiooni riist- ja tarkvarastrateegia, süsteemi väljaarendaja ning kasutaja ettevalmistuse tase jm. Instrumentaalse ekspertsüsteemi valikuks on koostatud ekspertsüsteem.

No 705

TALLINNA TEHNIKAÜLIKOOLI TOIMETISED
TRANSACTIONS OF TALLINN TECHNICAL UNIVERSITY

UDC 681.3
J. Henno

ALGEBRAIC FOUNDATIONS OF BELIEF SYSTEMS

Abstract

The main problem of logical programming is inference - characterizing logical consequences of formulas using the properties (and structure) of those of formulas only. A new, algebraic inference procedure is introduced here, which does not depend on the structure of formulas. The new principle is based on properties of logical operations on formulas only and is therefore very suitable for generalizations. The soundness and completeness of the new procedure are proved.

1. Introduction

A. Informal

The first step in formalization of our knowledge about some universe of discourse is conceptualization - definition of objects and their interrelationships. We represent our knowledge in a special language, called predicate calculus.

The most simple, unstructured (known) objects are represented as constants of our language, unknown objects - as variables. Relationships between objects are represented by relations. Since our knowledge about relations between objects is always somewhat uncertain, we will connect with every instance of a relation the degree of belief - a value which expresses how strongly we believe this particular instance of the relation to hold. Therefore we have to use predicates instead of relations, which differ from relations just in having this belief value.

Constants can also be viewed as values of attributes, represented by variables. Since attributes express prop-

erties, predicates express now properties of properties, i.e. a new kind of objects. This interpretation is often used in connection with databases and expert systems.

As a special kind of interrelationships we have to consider also functions, which map tuples of objects into other objects. The result of applying a function is represented in our language by a term and can be viewed as generating a new, structured object.

Predicate instances form most simple, atomic sentences of our word representation. Often we want to connect various facts using logical operations, which in our everyday language appear in the expressions like "... and ...","... or ...", "I strongly believe that ...", "I don't think that ...", "maybe ...". Logical operations act on the set of belief values. They are not independent - "I think that neither A nor B holds ..." is usually interpreted as "I don't think that either A or B holds". Thus instead of (possibly infinite) set of all possible logical operations it is sufficient to consider only some small subset of them. However, this subset should be complete, i.e. it should be possible to express every logical operation (from the class of operations, we are interested in) using operations from this class only. Here we will consider operations, which correspond to our use of "and" (denoted by $\wedge$), "or" (denoted by $\vee$) and unary operation of negation (inversion). This set of operations is complete in ordinary (two-valued) and in multiple-valued logic, thus these basic logical operations could be used to express other ones, such as "... implies" (denoted by $\rightarrow$).

In our everyday use of logical operations we also always assume them to satisfy certain conditions. For instance, "either A or B " is always the same (has the same belief value) as "either B or A " (commutativity of or-operation), "either A or then, either B or C " is the same as "either A or B , or C" (associativity of or-operation), " A or A " is the same as "A" (idempotency of or). The same properties hold for and-operation.

The basic problem of logic is logical inference - how to determine, which sentences follow from some given set of sentences, which we assume to be believable (up to

a certain extent). Various inference procedures have been introduced in classical logic, e.g. resolution procedure [2]. Usually these procedures use the syntactical structure of formulas and therefore could be used for formulas of a certain class only (e.g. resolution principle can be used only for formulas in the clause form). This makes them also difficult to use in generalizations, such as belief systems (essentially an infinite-valued logic). However, a very natural inference procedure can be introduced, based on algebraic property of and (conjunction) operation:

- if we believe the sentence "A and B" to be true, than we believe also both A and B to be true.

B. Formal

Let us denote by

$F = \{f, g, h, \ldots\}$ - an alphabet of function symbols,

$P = \{p, q, r, \ldots\}$ - an alphabet of predicate symbols.

For every $f \in F$ and $p \in P$ is uniquely defined a positive integer $n(f)$ ($n(p)$) - its arity, which indicates the number of arguments it is expected to take. Arity of predicate symbols is always greater than zero. Nullary function symbols are constants of our language. The set of all constants is denoted by C.

Terms are inductively defined as follows:

- variables and constants are terms;
- if $t_1, t_2, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term for every $f \in F$.

Let $T(X)$ be the set of all terms, $T = \langle T(X), F \rangle$ - the free algebra, generated by the set $X \cup C$.

A term is called ground if it does not contain variables, i.e. is composed of constant and function symbols only; the set of all ground terms is denoted by T.

Let $T = [0,1]$ be the unit interval of belief values. On this set we assume that two binary operations - disjunction ($\vee$) and conjunction ($\wedge$) exist. It is assumed that there exists also inversion (negation) operation on T. The result of applying negation operation to $x \in T$ is denoted by $\bar{x}$. We do not determine any formulas to calculate their value (usually $\vee = \max$, $\wedge = \min$, $\bar{x} = 1 - x$, but there

exist many other possibilities to determine these operations).

The disjunction and the conjunction operations are supposed to be commutative, associative, idempotent and satisfy the following properties:

$$\tau \vee p \geqslant \tau,\ \tau \vee p \geqslant p \qquad (1)$$

$$\tau \wedge p \leqslant \tau,\ \tau \wedge p \leqslant p \qquad (2)$$

and be mutually distributative:

$$(\tau_1 \vee \tau_2) \wedge p = (\tau_1 \wedge p) \vee (\tau_2 \wedge p),$$

$$(\tau_1 \wedge \tau_2) \vee p = (\tau_1 \vee p) \wedge (\tau_2 \vee p) \qquad (3)$$

The inversion operation is supposed to enjoy the following properties:

$$(\bar{\tau})^{-} = \tau,\ (\tau \vee p)^{-} = \bar{\tau} \wedge \bar{p},\ (\tau \wedge p)^{-} = \bar{\tau} \vee \bar{p} \qquad (4)$$

for every τ, $p \in T$. Notice that it is not asked if identities $\bar{\tau} \vee \tau = 1, \bar{\tau} \wedge \tau = 0$ hold.

Formulas are defined inductively by

- 0,1 are constant formulas;

- $p^{\tau}(t_1 \ldots t_n)$ is an atomic formula for any terms $t_1, \ldots, t_n$, predicate symbol p and $\tau \in \{0,1\}$; here $p^0(t_1, \ldots, t_n)$ denotes $\bar{p}(t_1 \ldots t_n)$, $p^1(t_1 \ldots t_n) - p(t_1 \ldots t_n)$; call them the negative and positive atoms, respectively;

- if $\Phi_1, \ldots, \Phi_n$ are formulas, then $\Phi_1 \vee \Phi_2 \vee \ldots \vee \Phi_n$, $\Phi_1 \wedge \Phi_2 \wedge \ldots \wedge \Phi_n$ are also formulas.

A formula is called ground if it does not contain variables.

Operations $\vee, \wedge, ^{-}$ are supposed to satisfy the identities, derived from (1)-(4), e.g. disjunction and conjunction are commutative, associative, idempotent, etc., also $\Phi \vee 0 = \Phi$, $\Phi \wedge 0 = 0$, $\Phi \vee 1 = 1$, $\Phi \wedge 1 = \Phi$ for every formula Φ.

Let $F(X,C)$ be the set of all formulas, $F(C)$ - the subset of all ground formulas. Let $\mathfrak{F} = \langle F(X,C), \{\vee, \wedge, 0, 1\} \rangle$ be the factor-algebra of the free algebra of formulas, satisfying the above laws (commutativity, associativity and idempotency for the disjunction and conjunction, derived from (1)-(4) laws, etc). The term and the formula algebras together with the set of truth values form a heterogeneous

two-carrier algebra, which we will denote $\langle T, F \rangle$. The operations of this algebra are:

- operations from the set $F: T \to T$;

- logical operations $\vee, \wedge, \neg: F \to F$.

In the following we will deal with representations and congruences of that algebra.

An interpretation I determines for every formula its belief value. Since the belief value of a result of applying a logical operation should follow from the belief values of component formulas, it is sufficient to determine the belief values for positive (or negative) atoms only. Algebraically this means that an interpretation I is a representation of the heterogeneous algebra $\langle T, F \rangle$. Thus it consists of two components:

- an algebra (D, F) of the same type as our term algebra; usually the denotations of constants and operations of this algebra will be the same as in the term algebra, but when it is necessary to distinguish between them we will use notations c^I, f^I ;

- a mapping $I: \{p(t_1 \dots t_n)\} \to T$, which determines for every positive atom $p(t_1 \dots t_n)$ its belief value $I(p(t_1 \dots t_n))$ (or, equivalently, a fuzzy relation on the set D, D is also called the domain of the interpretation I).

This mapping is in a natural way continued into a homomorphism of the corresponding algebras of ground formulas:

$$I(p(t_1 \dots t_n)) = p^I(I(t_1) \dots I(t_n)); \quad (4)$$

$$I(p^\tau(t_1 \dots t_n)) = (p^I)^\tau(I(t_1) \dots I(t_n)); \quad (5)$$

$$I(\phi_1 \vee \psi) = I(\phi_1) \vee I(\psi), \quad I(\phi \wedge \psi) = I(\phi) \wedge I(\psi). \quad (6)$$

Thus we can speak about truth value $I(\phi)$ of any ground formula ϕ under the interpretation I.

A variable assignment (with respect to the interpretation I) is an assignment to each variable in X of an element in the domain D . A variable assignment $A: X \to D$ can be in a natural way continued into a homomorphism

$A: \Phi \to T$ by
$A(c) = I(c)$ for every $c \in C$;

$$A(f(t_1 \dots t_n)) = f^I(t_1^I \dots t_n^I) ;$$

$$A(p^\tau(t_1 \dots t_n)) = (p^I)^\tau(A(t_1) \dots A(t_n)),$$

$$A(\phi \vee \psi) = A(\phi) \vee A(\psi), A(\phi \wedge \psi) = A(\phi \wedge \psi),$$

that makes it possible to speak about truth value $A(\phi)$ of an arbitrary formula ϕ under the assignment A and interpretation I.

In the following we assume all the variables in all formulas to be universally quantified. The truth value of such a closed formula does not depend any more on assignment, since we can take

$$I(\forall x \phi(x)) = \bigwedge_A A(\phi(x)) \quad (= \bigwedge_{c \in C} I(\phi(c))).$$

Thus we can speak about the truth value $I(\phi)$ of an arbitrary formula ϕ under the interpretation I.

A subset $\Gamma \subseteq T$ is called directed if $\tau \in \Gamma$, $\tau \leq \pi$ always implies $\pi \in \Gamma$.

A set of formulas Δ is called directed if for every formula ϕ, δ from $\delta \leq \phi$, $\delta \in \Delta$ follows $\phi \in \Delta$.

Let Γ be a directed subset of the set of truth values T. Interpretation I is called Γ-model for a formula ϕ if $I(\phi) \in \Gamma$. Interpretation I is a Γ-model for a set S of formulas if it is a Γ-model for every formula $\phi \in S$.

A formula ϕ is a logical Γ-consequence of a set of formulas Δ (denote $\Delta \vdash_\Gamma \phi$) if every interpretation I, which is a Γ-model for Δ, is also a Γ-model for ϕ, i.e. from $I(\Delta) \subseteq \Gamma$ follows $I(\phi) \in \Gamma$ for any interpretation I. Denote the set of all Γ-consequences of a set Δ of formulas by $\Gamma(\Delta)$. Clearly

$$\Gamma(\Delta) = \bigcap_{I(S) \subseteq \Gamma} I^{-1}(I(\Delta))$$

A substitution μ is a mapping

$$\mu : X \to T(X).$$

If $\mu(x) = t$, then the term t is also called a binding of the variable x under the substitution μ. We are interested only in non-identical bindings, therefore usually even under the term binding only a non-identical one is assumed. Again, a substitution μ can be in a natural way continued into an endomorphism $\mu : T \to T$ by

$$\mu(c) = c,$$

$$\mu(f(t_1 \dots t_n) = f(\mu(t_1) \dots \mu(t_n)).$$

Every endomorphism of terms also induces an endomorphism of formulas, thus we can consider μ to be an endomorphism of the whole two-carrier algebra $\langle T, F \rangle$.

On the set $\mathrm{Sub}(F)$ of all substitutions (actually, induced endomorphisms) a semigroup structure is induced by multiplication (superposition) of substitutions (i.e. mappings):

$$\eta\mu(t) = \eta(\mu(t)).$$

Since the operation of multiplication is idempotent ($\mu \cdot \mu = \mu$ for any substitution μ), this set of all substitutions (endomorphisms) also has a natural partial order:

$\mu \leq \eta$ if there exists a substitution ν such that $\mu = \nu\eta$.

It is convenient to consider also the empty substitution o, which maps all variables and terms into the empty set Φ. Clearly the empty substitution is the minimal element of the semigroup $\mathrm{Sub}(F)$.

A formula ϕ is called Γ-tautology if $I(\phi) \in \Gamma$ for every interpretation I.

The main problems of logical programming are:

1) for a set of formulas Δ and directed subset Γ, characterize all the formulas from the set $\Gamma(\Delta)$;

2) for a formula ϕ, directed subset Γ of truth values and a set of formulas (logical program) Δ find the maximal substitution μ such that $\Delta \vdash_\Gamma \mu(\phi)$.

The following answers are proposed to those questions:

- $\Gamma(\Delta)$ is the minimal directed subset of the set $F(X)$ of all formulas, containing all Γ-tautologies and the set $U(\Delta)$ of units of Δ;

- μ is the substitution, which generates the maximal endomorphism of the algebra $\langle T, F \rangle$, such that the formula ϕ is a unit of the set Δ.

A formula ψ is called unit for formulas $\phi_1, \dots \phi_n$ if

$$\phi_1 \wedge \dots \wedge \phi_n \wedge \psi = \phi_1 \wedge \dots \wedge \phi_n.$$

For instance, the resolvent [2] $a \vee b$ is a unit for clauses $a \vee \bar{x}$, $b \vee x$, since from (1)-(4) it follows, that

$(a \vee \bar{x}) \wedge (b \vee x) \wedge (a \vee b) = (a \vee \bar{x}) \wedge (b \vee x)$.

The set of units $U(\Delta)$ of a set Δ of formulas consists of all units for some finite subset of formulas from Δ.

2. Some properties of Γ-consequences

In the following let Δ be any set of formulas, Γ - a directed subset of truth values. Denote by $\Gamma(\Delta)$ the set of all formulas, which are Γ-consequences of the set Δ.

Proposition 1. For any formula $\phi \in \Delta$, $\phi \subseteq \Gamma(\Delta)$.

Proof. Let I be any interpretation, which is a model for Δ, i.e. $I(\psi) \in \Gamma$ for any $\psi \in \Delta$. But then also $I(\phi) \in \Gamma$, thus $\Delta \vdash_{\Gamma} \phi$.

Proposition 2. For any Δ, $\Gamma(\Gamma(\Delta)) = \Gamma(\Delta)$.

Proof. Suppose $\psi \in \Gamma(\Gamma(\Delta))$, i.e. if $I(\Gamma(\Delta)) \subseteq \Gamma$, then also $I(\psi) \in \Gamma$ for any interpretation I. By the previous proposition $\Delta \subseteq \Gamma(\Delta)$, thus also $I(\Delta) \subseteq \Gamma$. On the other hand, for any interpretation I such that $I(\Delta) \subseteq \Gamma$ by the definitions also $I(\Gamma(\Delta)) \subseteq \Gamma$. This means that $\Gamma(\Gamma(\Delta)) \subseteq \Gamma(\Delta)$, but from the previous proposition $\Gamma(\Delta) \subseteq \Gamma(\Gamma(\Delta))$.

Proposition 3. If $\Delta_1 \subseteq \Delta_2$, then $\Gamma(\Delta_1) \subseteq \Gamma(\Delta_2)$.

Proof. Suppose $\phi \in \Gamma(\Delta_1)$, i.e. $I(\phi) \in \Gamma$ whenever $I(\Delta_1) \subseteq \Gamma$. If I is a model for Δ_2, i.e. $I(\Delta_2) \subseteq \Gamma$, then also $I(\Delta_1) \subseteq \Gamma$, thus also $I(\phi) \subseteq \Gamma$ and so $\phi \in \Gamma(\Delta_2)$.

I set of formulas is called contradictory, if it does not have any models. Non-contradictory sets of formulas are called satisfiable.

Proposition 4. For any satisfiable set Δ of formulas and for any Γ-tautology ϕ, $\Gamma(\Delta \cup \{\phi\}) = \Gamma(\Delta)$.

From the definition of directed subset it follows that $\in \Gamma$ for any directed subset Γ, thus all the models for the set of formulas $\Delta \cup \{\phi\}$ are also models for the set Δ. From here the claim follows.

Proposition 5. For any satisfiable set Δ and for any Γ-tautology ϕ, $\phi \in \Gamma(\Delta)$.

Proof. By propositions 1 and 4, $\phi \in \Gamma(\Delta \cup \{\phi\}) = \Gamma(\Delta)$.

Proposition 6. The set $\Gamma(\Delta)$ is closed under the logical operations of disjunction and conjunction.

Proof. Suppose $\phi, \psi \in \Gamma(\Delta)$ and interpretation I is a

model for the set Δ. Then $I(\phi), I(\psi) \in \Gamma$. From (1) it follows, that $I(\phi \vee \psi) \geq I(\phi)$, thus by the definition of the directed subset $I(\phi \vee \psi) \in \Gamma$, i.e. $\phi \vee \psi \in \Gamma(\Delta)$. Similarly, $I(\phi \wedge \psi) = I(\phi) \wedge I(\psi) \in \Gamma$, thus also $\phi \wedge \psi \in \Gamma$.

Corollary. $\Gamma(\Delta)$ is closed under the operations $\vee$, $\wedge$ subset of the set of all formulas F, which contains the set Δ and all the Γ-tautologies.

Denote by $[\Delta]_\Gamma$ the set of minimal consequences from Δ, i.e. the set of all formulas $\phi \in \Gamma(\Delta)$, which could not be represented as $\phi = \psi \vee \tau$, $\psi \in \Gamma(\Delta)$.

Theorem 1 (Soundness) $U(\Delta) \subseteq \Gamma(\Delta)$.

Proof. Let $\phi \in U(S)$, i.e. $\psi_1 \wedge \ldots \wedge \psi_n \wedge \phi = \psi_1 \wedge \ldots \wedge \psi_n$ for some $\psi_1, \ldots, \psi_n \in \Delta$. Let I be any model for Δ, then from the properties of the conjunction operation it follows, that

$$I(\psi_1) \wedge \ldots \wedge I(\psi_n) \wedge I(\phi) = I(\psi_1 \wedge \ldots \wedge \psi_n \wedge \phi) = I(\psi_1) \wedge \ldots \wedge I(\psi_n) \in \Gamma,$$

thus by (2) also $I(\phi) \in \Gamma$, since Γ was directed. Thus every model for Δ is a model also for ϕ, i.e. ϕ is a Γ-consequence of Δ.

It is easy to see that even stronger claim holds, namely $U(\Delta) \subseteq [\Delta]_\Gamma$.

Theorem 2 (Ground completeness). Suppose Δ consists of ground formulas only, then $[\Delta]_\Gamma \subseteq U(\Delta)$.

Proof. The proof is by induction on the number of excess literals in Δ. The number of excess literals is the number of occurrences of atomic formulas in all formulas from Δ minus the number of formulas in Δ.

If the number of excess literals is zero, then all the formulas from Δ are atomic and all the minimal consequences of Δ are atomic formulas from Δ. Since every formula is a unit for itself (the idempotency of conjunction operation), the claim follows.

Suppose the number of excess literals in Δ is greater than zero and the claim holds for all sets of formulas with smaller number of excess literals. Suppose, for instance, $\psi_1 \vee \psi_2 \in [\Delta]_\Gamma$, where ψ_2 is an atomic formula (the proof is quite similar for formulas of a different structure, e.g. for $\phi_1 \wedge \phi_2 \in \Gamma(\Delta)$). For the sake of simplicity let

Δ be finite; the proof can be easily reformulated for an infinite Δ.

Let $\Delta' = \Delta(\psi_2=0)$, i.e. Δ' is obtained from Δ taking all the atoms ψ_2 equal to 0, all the atoms $\bar{\phi}_2$ equal to one and simplifying the results.

From $\psi_1 \vee \psi_2 \in [\Delta]_\Gamma$ it follows, that $\psi_1 \in [\Delta']_\Gamma$, thus by induction $\Delta' \wedge \psi_2 = \Delta'$. In the same way we obtain, that $\Delta'' \wedge \psi_1 = \Delta''$, where $\Delta'' = \Delta(\phi_1=0)$. Thus

$$\Delta \wedge (\psi_1 \vee \psi_2) = (\Delta \wedge \psi_1) \vee (\Delta \wedge \psi_2) = (\Delta'' \wedge \psi_1) \vee (\Delta' \wedge \psi_2) = \Delta'' \vee \Delta' = \Delta,$$

i.e. $\psi_1 \vee \psi_2$ is a unit for Δ.

For the general (non-ground) case we need a variant of the well-known

Lifting theorem. If Δ' is a set of ground instances of clauses in Δ and $\phi' \in \Gamma(\Delta')$, then there is a clause $\psi \in \Gamma(\Delta)$ such that ψ' is a ground variant of ψ.

This theorem can be proved along the similar lines as in the classical (two-valued logic) case, see e.g. [1].

From these theorems it follows:

Theorem 2. For an arbitrary set of formulas $U(\Delta) = [\Delta]_\Gamma$.

Corollary. $\Gamma(\Delta)$ is the minimal directed set of formulas, containing the set $U(\Delta)$ and the set of all tautologies.

References

1. Genesereth M.R., Nilsson N.J. Logical foundations of artificial intelligence. Morgan Kaufman Publishers, 1988.

2. Robinson J.A. A machine oriented logic based on the resolution principle// Journal of the ACM. 1965. 12(I). P. 23-41.

J. Henno

Uskumussüsteemide algebralised alused

Kokkuvõte

Loogilise programmeerimise põhiprobleemiks on loogiline inferents - loogilistest valemitest järelduste tuletamine, kasutades ainult valemite süntaktilisi omadusi ja struktuuri. Esitatakse uus algebraline inferentsiprintsiip, mis põhineb ainult loogiliste operatsioonide omadustel ja on seetõttu väga sobiv üldistusteks. Tõestatakse printsiibi mittevasturääkivus ja täielikkus.

No 705

TALLINNA TEHNIKAÜLIKOOLI TOIMETISED
TRANSACTIONS OF TALLINN TECHNICAL UNIVERSITY

UDC 681.3

E. Õunapuu

PRINCIPLES FOR SOFTWARE DEVELOPMENT

Abstract

Software systems can be divided into two categories. The life-cycle of the first category usually ends in the programming stage while other systems are inspiring user's trust. A successful software system needs continuous modification and development. This paper discusses some principles of software development and describes the practical experience gained. System GENSI is one of the most successful software systems, developed at the Department of Data Processing, Tallinn Technical University.

1. Development principles

The following major principles serve as quidelines for creating software systems.

1. Portability, possibility to use a software system on various computers, under different operation systems. Usually portability implies software portability, less often infoware portability. In fact, both are important, but not these alone. In software system development the user cannot be neglected. Every user has his/her own habits, portability of working style and working habits is very welcome from the user's point of view. This holds only when the user is satisfied with the software system. A new version can include new possibilities and usually does, but it is seldom necessary to change user interface entirely. Consistency is more important for the user than minor improvements in a software system at the cost of overall redesign of user interface.

2. Modularity means using functionally independent components when building a software system. Two levels of modularity are distinguished - that of the software (program modules) and that of the whole system. Software modularity means the following modular programming principles during software development; system modularity gives freedom to choose independent parts of a software system and to use them separately or jointly with other software systems. A combined modular software system allows more flexibility.

3. Interfaces with other software systems (this principle is closely connected with modularity). It is quite widely accepted that a software system must ensure a direct data exchange with the most popular software systems. The more direct the interfaces are, the better. However, a basic interface applicable to most software systems and programming languages must be available. A text file, containing minimal data description in the standard form as well as data, is a good example of a basic interface.

4. Multi-level user interface. In other words, a set of instruments must be provided for standard, easy and typical activities, whereas another must be saved for complicated ones. The end-user has an easy-to-learn, easy-to-use interface with the system, all his/her work is accomplished with this one. Creating an end-user interface is the function of an administration of a software system or a maintenance programmer. He/she uses instruments of lower level. When the universality of the software system is designer's concern, he/she will foresee changeable parts in software. Using the modular approach, program modules can be changed or added into the existing software. The lowest level interface is the programmer's one.

2. A software generation system for information system

The software generation system GENSI serves as an example of using the software development principles in real life.

The first version of GENSI was realized on PDP-11 computers using the operation system RSX. Presently a version of GENSI is realized for the IBM pc-compatible computers using MS DOS.

To generate the information systems software by means of GENSI high level description languages are used by the application programmers.

To create an information system the application programmer can use

- data processing procedures of GENSI;
- the procedures of the operation system of a host machine;
- the program realized by the application programmer.

In GENSI the following list of data processing procedures is available:

- creating, updating and printing of database;
- sorting of records in database;
- union or merging of files;
- data transfer from one file to another;
- verifying of data;
- report generation;
- menu generation.

To describe a data processing function a user must describe many aspects. For example, the description of a report includes the descriptions of overall logic, of layout, of arithemetic operations, etc.

GENSI is used as a basic system for creating information system software by five organizations in Estonia and by three in Perm (Russian Federation).

E. Õunapuu

Tarkvara loomise põhimõtted

Kokkuvõte

Artiklis tuuakse ära TTÜ infotöötluse kateedris väljatöötatud tarkvara loomise põhimõtted. Nende kasutamise näitena esitatakse generaatorsüsteem GENSI.

No 705

TALLINNA TEHNIKAÜLIKOOLI TOIMETISED
TRANSACTIONS OF TALLINN TECHNICAL UNIVERSITY

UDC 681.3

R. Kuusik

APPLICATION OF THEORY OF MONOTONIC SYSTEMS FOR DECISION TREES GENERATION

Abstract

In this paper a new way for generation of the decision trees immediately from initial data matrix X(N,M) is proposed. The use of the features extracted in conjunction with the theory of monotonic systems is described. The corresponding algorithms are estimated.

1. Formulation of the problem

Formation and use of the so-called decision trees (DT) deserves deep attention. The problem involves the following.

Let us assume that X(H,M) is a final data matrix, where H is the number of objects and M is the number of quantities Fj in X, in which each element Xij can have value from interval Cj = 1,...,Ej.

The decision tree <T,Fx> is defined as a rooted tree T with a root Fx so that all subtrees Ti of the tree T have exactly the same successors as they had in the tree <T,Fx>.

The best decision tree is called a tree <T,Fx>, to which the greatest value corresponds according to the previously determined criterion of "goodness".

2. The present situation

To solve the above-mentioned problem the algorithms for formation of a decision tree, the criterion of "goodness" and an algorithm of extraction the best DT are proposed in [1].

It is supposed that from data matrix X(H,M) by the GUHA method [2] elementary conjunctions Kl, l = 1,...,L,

were previously generated which are the nodes of a decision tree, and the corresponding output results Dg, g=1,...,G, i.e. elementary implications Kl --> Dg are determined. For each Dg its own best decision tree is generated. Value V= =A/B, where A is the number of objects in X(H,M), for which Kl--> Dg is true on X, B is the number of objects in X, containing Kl, serves as a criterion of "goodness" of a decision tree.

It is assumed that the best decision tree is the one to which the greatest value of criterion V corresponds.

General algorithm for generation of the best DT, according to [1], is the following:

1) form a tree for Fx according to Dt from {Kl};

2) determine the existence of one whole branch from a root (Cj)Fx to a leaf (i.e. that it contains all quantities j=1,...,M in the branch as nodes) on $\langle T,Fx \rangle$;

3) calculate the measure of a DT goodness. If Vx > > Vmax, then Vmax:= Vx and elementary conjunctions, corresponding to nodes $j \in \langle T,Fx \rangle$, j=1,...,R, $R \leq M$, being excluded from {Kl};

4) if there are no more trees, then the best DT is found, or else determine the next root quantity and pass to 1.

For Dt formation a table is constructed which measurements

$$P = Ex * M1 * \sum_{j=1}^{M1} Ej,$$ where M1 - the number of input quantities, $M \leq M1$.

If, for example, M1=10 and Ex=Ej=3, then P=900 cells.

E x a m p l e 1

Let {Kl} be a set of elementary conjunctions, generated by the GUHA method from X, l=1,...,11. Let F1, F2, F3 be input quantities and F4 an output quantity. For each Kl it is determined, what result Dg=F4 it comes to (see example in [1]).

Let us introduce elementary implications Kl--> Dg by table X(11,4):

Fj / Kl	F1	F2	F3	F4
1	1			1
2	3			2
3		2		1
4	2	1		1
5	2	2		2
6	2	3		2
7	2		2	2
8		2	1	1
9	2	1	1	2
10	2	1	2	1
11	3	2	1	2

For example, to the eleventh row of X corresponds the elementary implication (3)F1 & (2)F2 & (1)F1 --> (2)F4.

If you neglect the resulting quantity F4, then at the existence of elementary conjunction K12= (2)F1 from the root F1 by the data of the table X, a DT is generated

```
F1 -->  (1)F1
   -->  (2)F1   -->  (2)F2
                -->  (1)F2   -->  (1)F3
                                  (2)F3
                -->  (3)F2
   -->  (3)F1
```

The node (3)F1 is a list because an elementary conjunction in {Kl} equal to (3)F1 & (2)F2 does not exist.

3. A new solution

Solution of the problem suggested in [1] has some faults.

First, the set of elementary conjunction {Kl}, generated from X by the GUHA method and the corresponding elementary implications are the initial for DT generation algorithm, not the initial data matrix X(H,M).

The application of the GUAH method is a polynominal process. GUHA does not guarantee coverage of elements of the

initial data matrix X with generated by it elementary implications K1--> Dg because of existence of

a) subjective restrictions. The researcher determines previously the maximum length of the elementary conjunctions. It is assumed that the short conjunctions are "more interesting" than long ones,

b) algorithmical restrictions. They do not have quick algorithms for determination of all elementary conjunctions in X.

Secondly, the algorithm suggested for generation of the best DT is quite work-consuming:

1) the usage of the GUHA method,

2) constructing a table for each Fx,

3) determination of existence of the whole branch from a root Fj to a leaf (do there exist all quantities j=1,... ...,M as nodes). If it is so, we have to determine whether this branch is the best one.

Further we suggest a new approach for realization of the described problem using the theory of monotonic systems. The algorithms suggested can be used for generation of all DT on the initial data matrix X(H,M), or the best DT on X, or the corresponding DT also on the matrix of elementary implications, generated by the GUHA method or by its analogues (see example 1).

3.1. Description of the algorithm

A decision tree (DT) is defined in [1] as follows:

1) in one tree with a root Fx all values Fx: 1,...,Ex are the nodes of the root;

2) the sequence of quantities in DT for all branches with a root Fx is the same.

Using the criterion given in [1] , the best Dt is the one which has the greatest number of leaves.

In [3] the combinatorial algorithm for generating the best DT by the definition above directly from the initial data matrix is suggested. In [3] it is shown that the best DT is the tree which has the greatest coverage of elements of the initial data matrix, i.e. the best DT is the tree with the greatest number of leaves.

Below we will introduce a new solution of the problem, specify a DT, describe the algorithms and criteria of "goodness" for generating the best DT.

In [1] it is suggested that from the examined data matrix by the GUHA method Dtree nodes are previously generated. From these nodes various DT are formed and on the basis of the criterion, suggested by the authors, the best one is defined. From this point of view the suggested criterion satisfies the goals.

In reality it is more complicated. For example, during the expert interrogations a certain part of the experts may give similar answers. It means that the choice of criteria of the best DT may be put some other way too, taking as a principle also frequency of suggestions of certain answers or its combinations. For example, if we have two decision branches

1) K9= (2)F1 & (2)F2 & (1)F3 given by different experts 10 times and 2) K10= (2)F1 & (2)F2 & (2)F3 accordingly 6 times, then at the equal criterion value by [1] they can be equal, but K9 is preferable by the expert evaluation.

Concluding the above described discussion, we will suggest a new criterion and a new algorithm generation of the best DT direct from the initial data matrix X(H,M). According to this, to generate from X one node of the tree less than 2HM operations are needed.

Further, under the best DT we shall assume a whole branch of DT (from root to leaf), to which the greatest criterion value T (will be determined further) corresponds.

The algorithm for DT formation suggested below is based upon the theory of monotonic systems [4,5]. To create a monotonic system on X the so-called frequency conversion described in [6] is used.

Let X(H,M) be a final matrix of nominal data, where each element Xij may have a value from the interval Cj=1,2,. ...,Ej.

In algorithm A we use the following denotations:

X^l - the subset of X, $X^l \subset X$, $X^{l+1} \subset X^l$, $X^0 = X(N,M)$, max $l \leq M$;

$^l A^l$ - the table of frequencies of the subset X^l, $A^{l+1} \subset A^l$.

ALGORITHM A

A1. Find frequencies for every quantity $j=1,\ldots,M$ values $Cj=1,\ldots Ej$ from X^0 (they form table A^0) and find the greatest of them: $MAX:=\max|Cj|$. $F:=j$; $Y:=Cj$; $l:=0$; $t:=0$; $Kt:=(Y)F$.

A2. Find from X objects which contain a Kt. They form data matrix X^{l+1}.

A3. Find frequencies for every quantity $j=1,\ldots,M$ values $Cj=1,\ldots,Ej$ from X^{l+1} (we form table A^{l+1}). $t:=t+1$. The values of quantities, whose frequencies are equal to the value of MAX, form an elementary conjunction $Kt=\&_u(Yu)Fu$ with a frequency MAX.

A4. $l:=l+1$. If all quantities are described in Kt, then DO $A^{l-1}:=A^{l-1}-A^l$, $l:=l-1$. If $l=0$ then go to A5. END. Go to A2.

A5. The end of the algorithm

The frequency of the elementary conjunction is a monotonic function of weight. Elementary conjunctions Kt, separated by the algorithm A, are kernels according to the theory of monotonic systems [4]. The corresponding theorems are proved in [3].

Let us represent the elements of the set {Kl} (see example 1) in the form of a data matrix and assuming that zero corresponds to "is not determined", we obtain the data matrix X(11,4):

Fj / i	1	2	3	4
1	1	0	0	1
2	3	0	0	2
3	0	2	0	1
4	2	1	0	1
5	2	2	0	2
6	2	3	0	2
7	2	0	2	2
8	0	2	1	1
9	2	1	1	2
10	2	1	2	1
11	3	2	1	2

Table of frequencies for quantities in X is as follows:

Fj / Cj	F1	F2	F3	F4
0	2	3	6	0
1	1	3	3	6
2	6	4	2	5
3	2	1	0	0

E x a m p l e 2

Using X as the initial for the algorithm A and assuming that

a) zero value is not considered in the analysis,

b) the quantity F4 is not considered in formation of DT,

c) sequence of quantities in the DT is not defined previously, then in answer to the algorithm A usage, we acquire the following DT:

```
        6        3          1
1) --> (2)F1 --> (1)F2 --> (1)F3    (=10)
                            1
                       --> (2)F3    (=10)
                   2        1
                  --> (2)F3 --> (1)F2    (=9)
                   1
                  --> (1)F3 &  (1)F2     (=8)
                   1
                  --> (2)F2              (=7)
                   1
                  --> (3)F2              (=7)
         4         2          1
2) --> (2)F2 --> (1)F3 --> (3)F1    (=7)
                   1
                  --> (2)F1              (=5)
                   1
                  --> (3)F1 &  (1)F3     (=6)
```

```
                3                    1
3)  --> (1)F2  &  (2)F1  -->  (1)F3          (=7)
                                 1
                          -->  (2)F3         (=7)
                3         2      1
4)  --> (1)F3  --> (2)F2  -->  (3)F1         (=6)
                      1
                  -->  (1)F2  &  (2)F1       (=5)
                2                    1
5)  -->  (2)F3  &  (2)F1  -->  (1)F2         (=5)
                2         1
6)  -->  (3)F1  --> (2)F2  &   (1)F3         (=4)
                1
7)  -->  (3)F2  &  (2)F1                     (=2)
              1
8)  -->  (1)F1                               (=1)
```

In the represented trees the numbers above the pointers indicate frequency of a conjunction corresponding to the node j (it is an elementary conjunction which contains all antecedent to the node j elements ()Fx of the tree). For example, in the DT No. 1 |(2)F1| =6, |(2)F1&(1)F2|=3, etc.

We suggested the measure $T = \sum_{j=1}^{M} Sj$, where Sj is weight of the node j, as a criterion of the DT branch "goodness".

The combination of values of quantities (see example 2, the fourth branch of DT No. 1) may serve as a node. According to [1] it means that there is no branch from the root node ()Fx which contains all quantities as the nodes of DT. On the basis of our suggestions and the criterion of the DT "goodness", quantities, which belong to the node j, are not separated in relations to the node j-1, i.e. we do not prefer any of them.

If in the node j there are several elements (as a combination of values several quantities), the weight of the node is calculated as Sj= Qj * Zj, where Qj is the number of elements ()Fx in the node j and Zj is an elementary conjunction frequency in X corresponding to the node j.

The corresponding values of the criterion T are given in the brackets through the label "=" at the end of each branch of the DT. The measurement T of each node of a DT is calculated immediately in generation of DT.

Let us assume that the initial value of the measurement T=0 and after generation of the first branch Tmax:=T1. Then for each following branch b we may immediately say whether it is better or not: if Tb > Tmax then Tmax=Tb.

Forming a DT with the help of the algorithm A, formation of a new branch does not begin from the root Fx, but from the antecedent node b-1. If there are no more paths from this branch, then it is from b-2, etc. For each node the corresponding measurements equal to a sum of weights of the previous nodes are recorded. If in the formation of a new branch from a node b a value U (so-called forecast) $U=T_{b-1} + (M-b) * Sb < Tmax$ (where T_{b-1} is the measurement of the previous node to b, M-b is the number of this branch quantities not yet described), then we may finish formation of all branches from this node, because the measure T will always be less than Tmax. This is determined by monotonous of the weight function.

If for the succedent root of DT its forecast U < Tmax, then we finish work of the algorithm A, because we cannot find a new branch on X whose measure T > Tmax.

Therefore in our example we have to form only three completed branches:

```
   6          3          1
--> (2)F1 --> (1)F2 --> (1)F3      (=10)
                      1
                  --> (2)F3      (=10)
           2          1
           --> (2)F3  --> (1)F2   (= 9)
```

U=6+2*2=10 < =Tmax=10

For

```
  6          1
--> (2)F1 --> (1)F2 &  (1)F3 --> ... and we go to
U=3*6=18>Tmax  U=6+2*1 < Tmax      form a new DT
                                   from root (2)F2.
```

For the next branch

4 2

2) --> (2)F2 --> (1)F3... we finish the formation of branches from the node (2)F2 and go to a new root.

U=3*4 > Tmax U=4+2*2 < Tmax

3

3) --> (1)F2 & (2)F1 --> ...

U=3*3=9 < Tmax

Since already for the root of the tree 3) the forecast U < Tmax, the best DT is found and there is no reason to proceed.

Now let us concentrate our attention upon a peculiarity of measure T suggested by us. You may have noticed that its value can change if the sequence of quantities in the branch changes. It means that the algorithm A determines also the sequence of steps F1,...,Fm in making decisions, i.e. what is the best sequence of leading to the required result. The best sequence is the one to which the greatest value of a function T corresponds.

Taking into account also the resulting quantity F4 in choosing the best DT, the shape of the generated DT strongly changes, because depending on the value F4, we obtain two best DT, one for the data matrix X1 and the other for X2:

1) X1(5,3) and 2) X2(6,3):

	1	2	3	4
1:	1	0	0	1
2:	0	2	0	1
3:	2	1	0	1
4:	0	2	1	1
5:	2	1	2	1

	1	2	3	4
1:	3	0	0	2
2:	2	2	0	2
3:	2	3	0	2
4:	2	0	2	2
5:	2	1	1	2
6:	3	2	1	2

4. Conclusion

In conclusion, we will present the final algorithm of extracting the best DT.

ALGORITHM B

B1. T=0, j=1, Y=1 (contents of Y is No. of a branch =j).

B2. By the algorithm A we generate one node j of Dt. Let us compute the forecast U=Tj-1 + (M-j)*Sj.

If U < Tmax, then if j=1 (the node is a root), then go to B3, otherwise j=j-1 and go to B2.

If the node j is a leaf, then Tmax=Tj, Y=j, j=j+1 and go to B2.

B3. The end of algorithm.

This algorithm ensures a fairly rapid process of generation of the best DT from the initial data matrix X(H,M), because there is no need for preliminary formation and tracing of all DT. By the suggested criterion T we can preliminary to value "goodness" of DT, i.e. if it can serve as the best DT, or not. If the value of the forecast for a root node is not greater than Tmax, then according to the suggested algorithm, the best DT is found.

If we take into account that the search for the combination, corresponding to the node of DT, differs from the combinatorial algorithms and needs no more than 2HM operations, then the generation process of the best tree is very fast.

The algorithm suggested enables us to generate also K of the best DT, or even the worst one. Adding to the algorithm B the requirement that in a generated tree the sequence of quantities related to a root should be equal for all branches of the DT, then it is possible to generate also DT according to definition [1].

The algorithms described above are realized on PC IBM/AT.

References

1. Renc Z., Setikova L. Decision trees: a contribution to automatic interpretation of GUHA results // Int. J. Man-Machine Studies. 1985. No 22. P. 193-207.

2. Hajek P., Havranek T. Mechanizing hypothesis formation. Berlin-Hagelberg: Springer-Verlag, 1978.

3. Kuusik R. Generator hypotheses for qualitative data // Trans. of Tallinn Tech. Univ. 1987. No. 645. P. 141-148.

4. Mullat I. Extremal monotonic systems // Automation and Remote Control. 1976. No 5. P. 130-139; No 8. P. 169--178.

5. Vyhandu L., Müllat I. Monotonic system in scene analysis. Symposium "Mathemathical processing methods for data analysis and processing of cartographical data". Tallinn, 1979. P. 63-66.

6. Vyhandu L. Fast methods for data analysis and processing// Trans. of Tallinn Tech. Univ. 1986. No 614. P. 15-23.

R. Kuusik

Monotoonsete süsteemide teooria rakendusest otsusepuude formeerimisel

Kokkuvõte

Käesolevas artiklis esitatakse uus lähenemine nn. otsusepuude formeerimiseks, neist parima leidmiseks vahetult nominaalsete algandmete maatriksist X(N,M) ilma kõiki otsusepuid läbimata. Kirjeldatud lähenemine põhineb monotoonsete süsteemide teoorial. Esitatakse vastavad algoritmid.

No 705

TALLINNA TEHNIKAÜLIKOOLI TOIMETISED
TRANSACTIONS OF TALLINN TECHNICAL UNIVERSITY

UDC 681.-

P. Vyhandu, K. Regi

A GENERATING CAI SYSTEM

Abstract

The paper presents a CAI system for teaching list processing. The system takes an algorithm, written in a pseudocode for input and with the help of graphical output, step by step, demonstrates all the actions of the algorithm.
The purpose of such a system is twofold: on one hand, it is a tool for representing a new material, and, on the other hand, the students can check the correctness of their own algorithms.

Introduction

Recently computer aided instruction (CAI) has become very popular, for its instructiveness and illustrativeness. It involves the computer use for direct contact with the learner. Many subjects, that are difficult to explain to the students, become quite clear, when properly demonstrated on a computer.

At the Department of Data Processing of Tallinn Technical University list-processing and sorting and searching are taught to the students within one term. It is done on the logical level based on the approach of D. Knuth [1]. The purpose of such a course is twofold. First, the students should be acquainted with classics. Second, acquiring the technique of list-processing on this level helps them in their studies of other disciplines such as programming in C, data base design, etc.

Data structures in an information system must reflect some part of the reality. As this reality varies in the real time, these changes must also be revealed in the information system. On the level of list-processing it means updating the lists. So the main objective of teaching is to train students to understand processes occurring in the real time and to express them in the formality of list-processing algorithms.

As the experience has shown, the students are not used to such kind of thinking. Furthermore, the usage of linked lists instead of simple arrays is also quite unknown to them. Resulting from this they get a little confused and tend to mix up elements of lists, fields of elements and values written into fields. To overcome this confusion a lot of explaining has to be done. Explaining processes on the lists is quite a laborious and complicated work with the help of chalk and blackboard only, or occasionally with overhead. So there is a certain need for some kind of a CAI system.

This paper introduces a prototype CAI system for teaching list-processing with the help of algorithm animation. Algorithm animation, which means that the execution of the algorithm is illustrated with pictures, has proved to be a very useful technique in studying the operation of algorithms. Animations have turned out to be especially valuable in the analysis of algorithms and teaching algorithms. The present system can be used for preparing materials for a lecturer as well as for students to test their own algorithms. The system works on YAMAHA and the program is written in MSX-BASIC.

What should the system do?

The CAI system for teaching list-processing must illustrate graphically all the operations performed on lists. The system will take an algorithm for input and then it will show on each line of the algorithm the change on the list caused by this line. Furthermore, the system must do it for any algorithm for two reasons.

1. It requires too much work to make deterministic programs for each lecture. According to unpublished information obtained from Th. Ottmann, a professor at the University

of Freiburg, it takes about 30 hours for a professor to design a lesson and about 150 hours for a student to write the realisation programs. There is always a possibility that the example will fail and will not work for students. Then the work has been done all in vain.

2. A general system can also be used by students in order to watch the work of their own algorithms and discover their mistakes.

To meet these requirements the system must provide the user with the following:

- input, maintenance and updating of user's algorithm;
- representing the work of an algorithm with the help of animated illustrations.

The first requirement is quite obvious, but the contents of the second one should be cleared.

For presenting animated illustrations, the system must generate a program to do this job. For this task the mere text of input algorithm is not enough because the algorithm analysis does not determine its contents. It means that in addition to the text of algorithm the user must describe some parameters, i.e. he must assign names to the fields and variables used. Naturally, the algorithms must be written in a language with strict syntax rules. The system reads the algorithm and checks its syntax. If the algorithm contains no syntax errors the system, using both the text of the algorithm and the description of list-structure, generates a program. The generated program performs the work of the algorithm with animated illustrations.

Using the CAI system

Before the user can run the algorithm, the text of the algorithm is to be entered and the list-structure described. While running the algorithm mistakes can occur. To get rid of the mistakes the user can correct the description of the list-structure or the text of the algorithm. So the usage of the CAI system may be divided into three parts:

(1) describing the list-structure;
(2) input and correcting the text of the algorithm;
(3) running the algorithm.

These actions make up the main menu and the user can choose between the parts of the system.

Describing the list-structure

The user describes the list-structure with the help of a dialogue. The dialogue consists of a menu and a series of questions. In the menu the user can choose between different types of lists (list with one link, list with two links, table, hashtable). Questions of a dialogue depend on a chosen list type and they determine the number of fields being used, the names and the values of the fields, pointers and variables. Additional questions concern the state and the properties of the list, for example "Is the list empty?", "Is the list sorted?", etc.

After describing the list the picture of the list is being displayed on the screen. The user can now check the list description and make corrections if necessary. If he is satisfied, he can save and/or exit to the main menu.

The algorithm-writing language

The language in which the algorithm is written must enable the use of the main features of programming languages and special constructions for list-processing. The language resembles the pseudo-code of D. Knuth [1]. Two types of variables can be used: simple variables and addresses. With the help of the language both four arithmetic operations and the operation MOD can be performed. The language allows to use such logical operations as =,<,>,<>,=<,=>. The main three programming constructions – sequence, selection and loop are used for writing the algorithm. The language also recognizes some words which describe situations arising in executing the algorithm. Such names are OVERFLOW, UNDERFLOW, SUCCESSFUL END, UNSUCCESSFUL END. In addition, special names OUTPUT and AVAIL are used.

Running the algorithm

The user can run his/her algorithm only after describing the list-structure and input of the algorithm text. First, the syntax of the algorithm is checked. When there are syntax errors, the user can choose whether to correct the mistakes or run the algorithm in spite of them (without wrong instructions).

When running the algorithm, the described picture of the list and the first line of the algorithm appear on the screen. To activate the command any key can be pressed. After the keystroke the changes caused by the executed command are shown on the screen and the next line appears. In such a way the user can step by step go through the entire algorithm. If the user wishes the algorithm can be terminated and it is possible to enter other parts of the system or to repeat the algorithm animation from the beginning.

The CAI system has many fine points

By Taylor [2] the computer acts in education as a tutor, as a tool, and as a tutee. The present CAI system for list processing can be used in all these roles. The system interacts as a tutor, when it shows to the student mistakes made by him, for example syntax errors. The system discovers not only typing errors but also conflicts between list description and algorithm, where the student may have mixed up different types of variables. The system can be used as a tool for checking an algorithm for another kind of list-processing application. For a teacher the system is a very useful tool for preparing lessons. It takes little time to describe some lists and write algorithms for the following lesson. To use the system as a tutee is to tutor the computer how to perform the task in the way the user wants. The user must give an exact description of his algorithm, specifying what he wants, otherwise the result would not satisfy him. It is very difficult to teach somebody without knowing the subject himself. So the student himself has to learn and then teach the computer. The computer shows with the help of animation what he just did not "understand" and the student has to improve the instructions. This interactive process would have high teaching effect.

Most of the CAI lessons are made using either direct programming or authoring systems. Both ways have their drawbacks - programming is too tiresome, authoring systems do not provide much flexibility. The presented CAI

system, which generates programs for algorithm animation, avoids these drawbacks. Describing the list-structure through the dialogue and writing the algorithm in the simple code takes little time. These tools also give the system a great deal of flexibility, the only limitation being that the given algorithm has to consider lists.

The user-friendliness of the system allows to use it not only for preparing lessons; students themselves can check their own algorithms. Instructiveness is guaranteed with illustrations, which clearly show the correctness or incorrectness of the result, while animation makes it possible to watch the process caused by the algorithm. Students can check the outcome at each stage of the work and choose whether to proceed or redo something. And what is also important, they can proceed with the tempo which is most suitable for them.

References

1. Knuth D. The art of computer programming. Vol. 1. General algorithms. Stanford University, 1970.

2. Taylor R. The computer on the school: Tutor, tool, tutee. New York: Teacher's College Press, 1980.

P. Võhandu, K. Regi

Genereeriv raalõpetussüsteem

Kokkuvõte

Artiklis kirjeldatakse genereerivat raalõpetussüsteemi nimistute töötlemise õpetamiseks. Süsteemi sisendiks on pseudokoodis kirjutatud algoritm, mille töö seejärel sammsammult graafilise väljundi abil demonstreeritakse.

Süsteemi võib kasutada uue materjali esitamiseks, kuid ka õppurid saavad selle abil kontrollida isekoostatud algoritmide töö õigsust.

No 705

TALLINNA TEHNIKAÜLIKOOLI TOIMETISED
TRANSACTIONS OF TALLINN TECHNICAL UNIVERSITY

UDC 681.3.016

T. Lumberg

ON ORGANIZATIONAL "STATUS QUO" IN INFORMATION SYSTEM DESIGN

Any automation without a good understanding of the human work may cause a disaster.

Hiroyuki Yoshikawa
University of Tokyo, Japan

Abstract

In our paper we describe some problems of modelling of the data processing on an enterprise. The contradictions between the traditional infological approach and actual needs of the enterprise for functional improvements lead to the creation of a method for criterial analysis and design on the informational/functional basis.

To offer practical suggestions a functional approach to information production and dissemination is discussed.

Practical informatics and widely spread consultations for enterprises have resulted in the following. For instance, an enterprise orders a strategic design of an information system, or consultations for computer-aided implementations and gets a science-based qualified result reflecting the enterprise from the informational aspect. But then it occurs that the result does not correspond on a large scale to the specific needs of this enterprise. To be exact, the enterprise is willing to make some organizational changes based on these results. The changes may involve different management activities, including work, handling

of resources, etc. The enterprise has strong intentions to get here a sound foundation for these changes.

And the first contradiction emerges.
To have knowledge of the enterprise as a system, a design based on a wider and optimum foundation rather than being simply an aspect of the system is required, though, in fact, the aspect serves as a source.

In some respect this contradiction appears due to the specifics of the infological analysis (see the infological approach [1,2]), because it embraces organizational, informational and activity levels of the mentioned enterprise. A whole, manyleveled analysis brings forward functional disharmony, lack of information and organizational disorder in the investigated system. To construct the automated infosystem on such a basis, according to all regulations, means fixing and enlarging these faults.

The problem is that, by dealing with the development of information system, the enterprise actually undertakes substantial improvements in its work. At the same time, the structure of computing is just one aspect of the structure of an organization [3].

Strict specific informational analysis represents an informational dimension. In some respect it is an objective dimension, because it exhaustively reflects the reality. But the question is, how to change this reality. Owing to its specifics, the informational analysis may serve as a means for the estimation of an alienation [4], but not for overcoming it. For example, when we investigate informational needs of an enterprise worker, we do not know, whether his subjective informational needs rest on the objective grounds, in terms of the enterprise.

The second contradiction comes in here.
The informational analysis cannot give an objective motivation to overcome alienation, until it stays on all three levels within the existing framework of the enterprise.

The experience [5,6,7], related to our objective, indicates that we need a broad approach, including many aspects and many parts of the industrial organization, in order to

fully utilize modern technology. We must verify such an experience in our economic environment, which is not similar to other ones.

Since we are concerned with quite different environments from the experience mentioned, the functioning mode of the enterprise structure is also different. Let us name conditionally the enterprise in our environment as No. 2 and the enterprise in the environment of the authors' experience, mentioned above, as No. 1.

The enterprise No. 1 is bound for efficient work. The structure and the functioning mode of the enterprise are formed accordingly. The analysis of the existing information system on that enterprise and the design of automated data processing guarantee the future results. The information system is described as a whole unit in its current status and the set of suggestions is restricted by technological suggestions rather than by inner ones.

The analysis of the informational aspect analysis enables to illuminate redundancy and faults of information processing, though even after the sorting, the structure of information processing remains mainly consistent with the structure of the enterprise activities, as it had been before. Since the enterprise No. 1 is an efficient system (i.e. had been optimized before), then the information system naturally develops into it and strengthens its efficiency.

The enterprise No. 2 functions in the framework of alienation [4] and is not an optimized system. That is why, the mechanical use of the methodology that suits the enterprise No. 1, may prevent in the case of the enterprise No. 2 compatibility of the system with reality, and the alienation may strengthen. Infological analysis reveals a lot of uncertainty the sorting of which, on the basis of the traditional methodology, will still be optional. Even after the sorting, as was mentioned above, the consistency with the current structure of the enterprise and with its functioning model involving all its faults is followed.

Let us approach the problem from another angle. Up to now, we did it assuming that information is an objective dimension. There is a need for a more objective dimension to call on the level of changes for projecting it on an enterprise.

Circulating information on the enterprise reflects its material processes, the principal function of which is production. For a functioning enterprise it is necessary to support:

1) technological processes in the production;
2) production as a whole.

All the activities of the enterprise should be oriented on the above-mentioned tasks. That makes it possible to show the activities and functional units which are necessary for an operative enterprise. So we intend to answer the question: HOW DOES IT WORK? It has a physical context, a data processing context and also an organizational context. It is not solely the concern of data processing. The problem belongs to the area of enterprise analysis.

Thus is shaping the second dimension - the functional. Application of that dimension in the enterprise analysis permits us to model the enterprise operating system. The term 'operating system' means that the model involves related activities which are oriented on performing or supporting the major function of the enterprise. In the process are the physical and organisational contexts (for example, it is obvious that certain resources are needed for production, consequently, resource handling too, etc.) are also involved.

Informational aspect of the operating system can be involved to show, proceeding from the defined activities, based on the functional grounds, what kind of information is needed on a certain level to carry out a certain activity (what and about what), what information flows are outlining in the operating enterprise. Actually it means the design of informational environments for the objective activities. In other words, it is possible to describe the information base which is determined by the major function of the enterprise. This relation determines the informational base [8] objectivity. To describe this informational base we need methods, which provide a more in-depth description by focusing on the structure of information in support of what is being performed. By that we attempt to overcome the fact that data processing activities are determined by the outlined ac-

tivities of the enterprise, which are not necessarily objective, especially outside of production.

After the infological analysis, as an ideal version, the structures to be (re)designed, should be torn apart' from the analyzed structures, i.e. they ought to be optimized. On the grounds of the model proposed it is possible to give some specific advice and suggestions on how to introduce internal changes into the enterprise, i.e. to overcome alienation.

Thus, the question is, what can serve as a foundation for optimizing.

The model of the operating enterprise (including information processing) should approach an objective basis. Here appears the need to analyse functions and activities of the enterprise, on the account that production is essential, and design on this basis (i.e. on operating system) the information work, taking into account, that information and communication on each level should form an information environment of objective activities. We do not design an information system with its activities as a separate unit within the existing framework of management, office routines and documents. We intend to create an overall design from the related environmental modules on the functional ground.

It leads us to create a method for criterial analysis and design, in order to describe and to formalize the current systems as well as to optimize it. The method is usable only if it does not reduce the complexity of the enterprise system, but enables to understand the complexity, in order to use it better. Its application should be performed, taking into account iteratively the various levels of abstraction and by increasing progressively the level of detail. So, from a global viewpoint through a top-down approach (because it is an organizational system with several aspects) and in details through a bottom-up approach (in order to obtain an accurate running).

Considering the above mentioned, it is possible to assume that such a method should have the following steps.

I Enterprise analysis: the study of the current structures and processes, identification of the characteristics and criteria.

II Development of the "optimized" enterprise model: data processing on several levels is determined by and linked with functions in the operating system of the enterprise.

III Verification and validation of the optimized model: determination of the model consistency with the characteristics and criteria, detection and characterization of the conformities/unconformities with the real system.

In creation of the methodology according to specific needs of our economic environment (as mentioned above) we can use "ICAM Definition" Method (IDEF)[6], the idea of which is to work out a description of manufacturing from a completely functional viewpoint, irrespective of organizational lines and interfaces. Also M*[7] methodology, which tends to integrate enterprise analysis methods with logical database design methods.

Conclusion Remark

In this manner we shall redesign the operating system of the enterprise. (It is assumed that the operating system is consistent with the lowest level of decisions.) We get a global view of the current status of the enterprise, on the functional as well as on the informational dimensions, so that problems can be identified, and the effects of their solutions on the existing system can be evaluated. Thus, we can carry out the enterprise alteration program on the functional/informational basis.

References

1. Laast-Laas J. Data base infological design in practice // Trans. of Tallinn Tech. Univ. 1986. No 614. P. 107--112.

2. Vapper T., Laast-Laas J., Urvak A., Elmik L. Practice aspects of information system design // Trans. of Tallinn Tech. Univ. 1987. No 645. P. 118-125.

3. Gray J. An approach to decentralized computer systems // IEEE Transactions on Software Engineering. 1986. Vol. SE-12. No 6. P. 684-692.

4. Lumberg T. Information and alienation // In: Seminar on Social Synergetics and Informatics. Tallinn, 1989 (in Estonian).

5. Burbidge J.L., Falster P.P., Riis J.O., Svendsen O.M. Integration in manufacturing // Computers in Industry. 1987. No 9. P. 297-305.

6. Bravoco R.R., Surya B.Y. Requirement definition architecture - an overview // Computers in Industry. 1985. No 6. P. 237-277.

7. DiLeva Al, Vernadat F., Bizier D. Information system analysis and conceptual database design in production environments with M* // Computers in Industry. 1987. No 9. P. 183-217.

8. Vapper T. On strategy of strategic design // Computer Technics & Data Processing. 1988. No. 5. P. 1-8 (in Estonian).

T. Lumberg

Ettevõtte optimeeritud infosüsteemi väljatöötamine

Kokkuvõte

Artiklis käsitletakse ettevõtte infosüsteemi projekteerimise küsimusi. Lähtudes funktsionaalsel alusel kirjeldatavast ettevõtte nn. operatsioonisüsteemist, kirjeldatakse täiendavaid kriteeriume projekteerimismetoodika täiustamiseks.

No 705

TALLINNA TEHNIKAÜLIKOOLI TOIMETISED
TRANSACTIONS OF TALLINN TECHNICAL UNIVERSITY

UDC 681.3

L. Elmik

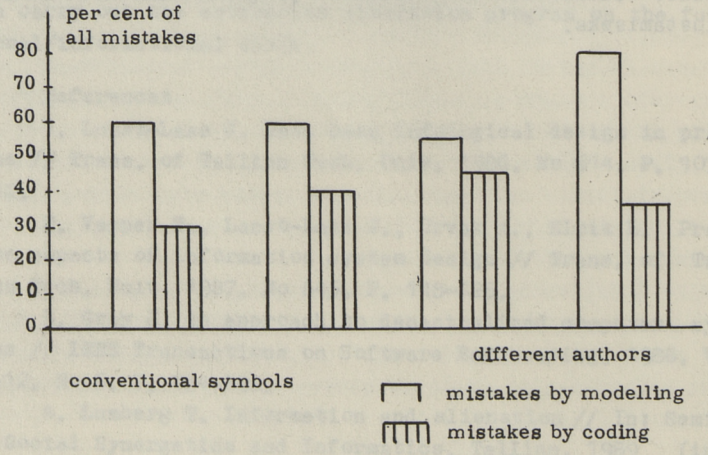
OFFICE WORKSTATION MODELLING IN THE WHOLE INFORMATION SYSTEM

Abstract

An approach is given representing a model of an office workstation as a main functional block of information system in an office. The main idea of the model creation is to describe the data conversion in regular communicative situation on three levels.

Creation of a large software system is a complex process, which appears to be an endless source of making mistakes. In practice there are no faultless systems.

In [1] M. Leppart and B. Stork summarized the parameters of mistakes made by different authors which can be represented by the following scheme:



As seen from the scheme, all the authors reached the same conclusion - mistakes of modelling are dominating over the coding mistakes. Apparently, methods for modelling an information system on the whole are insufficient.

The majority of modelling approaches provide a limited means, being designed only for a part of the information system [2], and these can model only some processes within the whole of management activities [3,4]. Here we shall propose an approach for representing a model of an office workstation, which acts as a main functional block of an office information system. The modelling process is based on the office works brought out by M. Mizuno [3], which include the following.

1. Creative human activities (thinking, decision making, planning, negotiating, directing).

2. Information input (reading, hearing).

3. Document creation (writing, making clean and neat copies).

4. Data processing (calculating, sorting, classifying, comparing, editing).

5. Information storage and retrieval (filing, searching).

6. Communication and information distribution (talking. inter- and intraoffice mailing).

7. Disposal of unnecessary documents (scrapping).

8. Conferences and meetings (communicating, recording).

Those activities can be classified into the following groups:

- management works (1);
- data processing and transmission works for management (2, 3, 4, 5, 6, 7);
- communication works (2, 6, 8).

It follows from this, that an office has three kinds of functions: management, data transmission and processing, communication. But it is unclear what the object is for the management and between which objects the communication is held. Therefore the object of management and the relations between the objects in a communication process should be specified. As a result, we should know the final role of the office.

To bring out the role of an office, we shall not discuss how data processing is designed but rather why it is needed.

To answer the latter question, we must answer the following questions.

- Which processes/activities are data sources?
- Why is the data needed?
- Which process/activity will need this data?
- What is the meaning of this data?

It means that we are interested in the part of communication process which is covered by the data processed in the office activities.

In our previous article [5] the model of information streams was presented:

- data from a production process;
- data for planning a new product;
- marketing.

In this model inside and outside streams are distinguished and the structure of the inside streams is given. Anyway this inner structure is not sufficient for understanding the role of the office. Therefore we must investigate the real data streams in more detail. It appears that these data streams contain:

- data about the real production process;
- data about the real marketing process;
- data about planning resources/production.

In general, we may say that this data covers all processes in the life-cycle of the product. From this it follows that an office environment, which involves communicative and management processes, influences all processes in the life-cycle of the product. Consequently it follows that the object of management is to influence processes of the life-cycle of the product. The communications between those processes are imperative to get information for influencing them.

Now let us look at the role of the workstation in an office. Is it just a data processing tool or beyond that? Do we need data processing as such at any cost or do we need exact and operative information on the processes of the life-cycle of the product?

If the latter case is our goal, then an office workstation must be a tool, which creates an active communicative context (united knowledge, united information, united data) for influencing the life-cycle of the product and also supports the office work with the software tools for updating and using that context.

Now let us proceed how to model the office workstation.

The main idea of the model creation is to describe the data conversion in every regular communicative situation [6]. The description of this conversion is given on three different levels:

- communicative level;
- information system level;
- programming level.

Level 1:

- specifying the activity of a management enterprise - the activity which needs the data (the receiver activity);
- specifying the activity of a management enterprise - the activity which creates the data (the sender activity);
- specifying the essence of the data - the message - why the receiver activity needs the data from the sender activity.

Level 2:

- specifying the activating communication process in the computer information system;
- specifying data representation for the user;
- specifying data storage structures.

Level 3:

- specifying the program.

Distinguishing those levels appears to be the basics of office workstation modelling. It ensures the independence of results from the office clerks' manners and documentation circulation which are commonplace up to now. Instead of this the model will depend only on communicative needs.

In conclusion, we achieved the model which allows to integrate the work of several office clerks and results in a better circulation of data streams. Due to that a great deal of unnecessary work and mistakes, caused by them, can be avoided.

References

1. Leppert M., Stork B. Entwicklungsumgebungen - Status quo und Perspektiven // Informatik - Fachberichte. GI. Programmierumgebungen: Entwicklungswerkzeuge und Programmiersprachen. Berlin-Heidelberg-New York-Tokyo: Springer-Verlag, 1984. S. 205-234.

2. Software engineering: methods and techniques. N.Y.-Chichester-Brisbane-Toronto-Singapore: John Wiley & Sons, 1983.

3. Masayoshi Mizuno. Office automation // Journal of Asia Electronics Union. AEU. October 1986. No. 129. P. 89-94.

4. Weston Ch.D., Stewart G.A. Workstations // Byte. February 1987. P. 85-97.

5. Лааст-Лаас Ю.Г., Эльмик Л.Н., Диго С.М. Проектирование информационной системы на промышленном предприятии. Вопросы проектирования СМОД. Москва: МЭСМ, 1988. С. 53-59.

6. Якобсон Р. Лингвистика и поэтика. Структурализм "за" и "против". Москва, 1975. С. 198-202.

L. Elmik

Kontoritöökoha modelleerimise põhimõtteist

Kokkuvõte

Artiklis käsitletakse kontori infosüsteemi modelleerimise protsessi ja kriteeriume. Vastavalt modelleerimisprotsessile töötatakse välja kommunikatsiooninõudeid kajastavad põhimõtted.

No 705

TALLINNA TEHNIKAÜLIKOOLI TOIMETISED
TRANSACTIONS OF TALLINN TECHNICAL UNIVERSITY

UDC 681.3

T. Vapper

INFORMATION FLOW MATRICES

Abstract

The design process of information systems consists of several phases. Every phase is based on the results of the previous one. So the initial stages of information system's life-cycle build the foundation for the realization of the system. The representation and results of specification of a company's information work, its structure and information requirements are especially important for the later phases. The success of transition from one phase to another depends on the success of fusing the different methods into an integral model. Information flow matrix is one of the possible complex models for different stages of the design process.

Introduction

The aim of this paper is to introduce the information flow matrix (IFM) as a relational model of an information system design methodology called Complex Methodology of Information Systems Design & Strategic Planning (CM), as well as several applications of IFM.

CM is a data-oriented methodology, where methods follow the corresponding concept. So, information flow matrices principally differ from information flow diagrams, widely used as a technique of process-oriented methodologies. However, they are both primarily used at the same phase of information system design process. The life cycle model for the information system development represented by P.C. Lockemann and H.C. Mayr [1] will be discussed further.

As S. Ceri distinguishes "social" and "socio-technical" approach to the initial phase of the life cycle of an information system ("requirements collection and analysis") [2]. IFM belongs to the first one. Optimization of information work, organizational development and managerial improvements have been considered essential in the circumstances of an unaccomplished socio-economic system.

The term "real system" is used in the sense of precisely determined subject of design activities (the whole or a part of a company, organization, office, etc.) with its defined functions and complicated structure. "Object system" is an integrated component of the real system embracing events, occurrences, acts and other material or non-material related objects of which information is required.

Figure 1 represents a generalized model of a real system's information work (a "cube"), acting as a source-vision of IFM application [3, 4].

The components of IFM

Information flow matrix is a method primarily used for the first phase of information systems design - requirements analysis and specification. However, it can be used for the later phases as well.

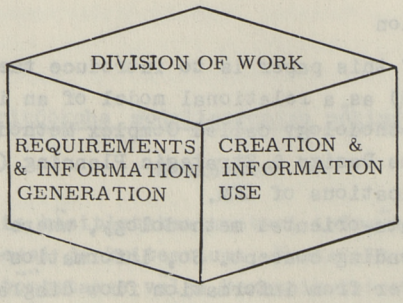


Figure 1. Generalized model of information work

Principally, IFM is an evaluated model of the "cube" (Fig. 1) and represents information work on the actual real system.

IFM consists of the following components (Fig. 2):

- information;
- organizational structure (actors);
- actions.

Information can be formal, informal or unformal. No matter what the mode of transmission is anyone is described by IFM. Description of information contains the name, representing the semantics, time periods and the form of transmission (a document, phone-call, etc.).

Organizational structure is represented on the main diagonal of IFM by departments (divisions), persons or organizations. Every department is characterized by its name, location, telephone numbers and the number of workplaces. The definition of the department depends on the function and information flows, e.g. the chief engineer can be a member of administration of one real system and a separate department of another.

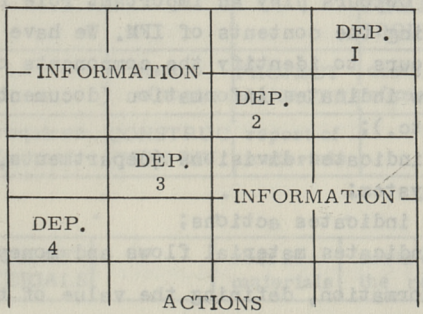


Figure 2. The structure of IFM

Actions of information work are described corresponding to the departments (actors) at the lower part of IFM.

Actions and information are connected with the corresponding indices.

Figure 3 represents the direction of information flows on IFM.

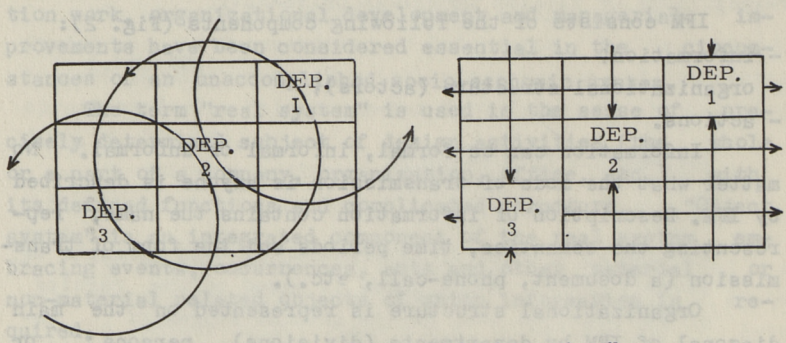


Figure 3. The direction of information flows

The mode of representation

To cooperate with end-users IFM is represented according to the principles of the wall technique [5]. So, large surfaces (walls) and papers as well as different colours are required. Colours play an important role for end-users in understanding the contents of IFM. We have used the following colours to identify the components of IFM:

- yellow indicates information (documents, reports, phonecalls, etc.);
- blue indicates divisions (departments, persons, etc.) of the real system;
- green indicates actions;
- red indicates material flows and money (if necessary).

Any information, defining the value of the matrix element, is written on a coloured paper-label. They are glued to the greater form of paper. So, it is easy to make changes.

IFM as a description technique

One of the best features of IFM is its complexity. A real system can be described with one or several IFM. It is often useful to compose an IFM of the whole organization as well as an IFM of the divisions. Figure 4 represents a sample extract of a building company's IFM (without actions).

Instead of departments it is possible to describe any other kind of divisions. For instance if we use the main diagonal for the specification of technological devices

(divisions), we could analyse the informational interrelations between CAM and information systems.

IFM as a testing tool

Due to the complexity and simplicity IFM can be checked by the end-users of the future information system. IFM can be used as an interface between designers and end-users. It gives a perfectness guarantee to the results of the first design phase.

IFM as a mechanism of reorganization

IFM gives the possibility to identify the anomalies of information work, including informational "bottlenecks", duplication of data and actions, disproportions of division of work, etc. As IFM is an integral model it is not difficult to find out the reasons for disconnections of information flows (inadequate specification or organizational defect).

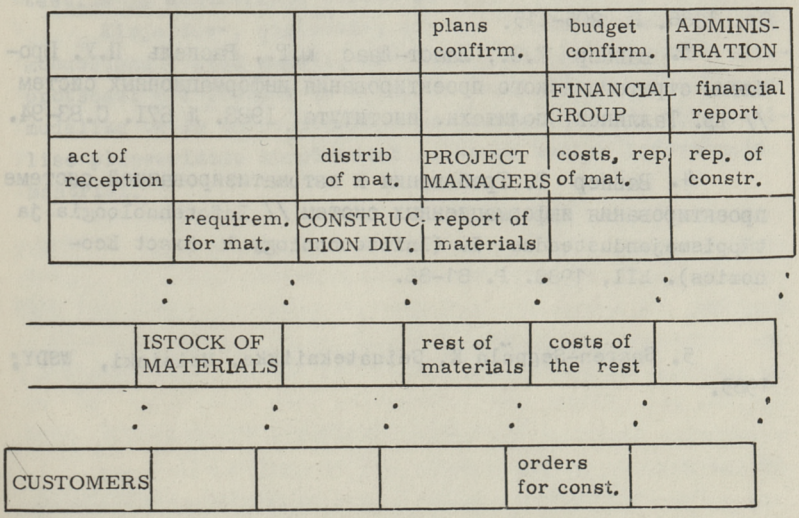


Figure 4. An extract of a company's IFM (example)

Any changes of the IFM values affect other components. If we are abolishing a document, sent from one division to another, the corresponding action cannot be executed. Abolish-

ing the actions, the corresponding information will not be obtained. Any modification affects the whole organization. The future information work structure will be modelled before the future information system is realized.

IFM is a tool for optimizing information work processes and information flows.

IFM plays an important role at all design phases, since every state of the design process will be modelled in advance. But, of cource, it is possible only with the help of special automated tools [4].

References

1. Lockemann P.C., Mayr H.C. Information systems design: techniques and software support// IFIP 86. North-Holland Publ. Co., 1986. P. 617-631.

2. Ceri S. Requirements collection and analyses in information systems design// IFIP 86. North-Holland Publ. Co. 1986. P. 205-215.

3. Ваппер Т.Э., Лааст-Лаас Ю.Г., Распель П.У. Проблемы стратегического проектирования информационных систем // Тр. Таллинск. политехн. института. 1988. № 671. С.83-94.

4. Ваппер Т. Требования к автоматизированной системе проектирования информационных систем // Infotehnologia ja täppismajandusteadus I. (Infotechnology & Exact Economics). EII, 1988. P. 81-86.

5. Saaren-Seppälä K. Seinatekniikka. Helsinki, WSDY; 1983.

T. Vapper

Infovoogude maatriksid

Kokkuvõte

Infosüsteemi projekteerimine on mitmejärguline protsess. Igal järgneval etapil kasutatakse eelmisel saavutatud tulemusi. Projekteerimise algfaasis rajatakse tulevsüsteemi "vundament", mistõttu kasutatavad meetodid peavad tagama tulemuste ülima korrektsuse ja täpsuse.

Projekteerimise edukus sõltub nii meetodite sujuvast üleminekust ühelt elutsükli etapilt teisele kui ka sellest, kui efektiivselt suudetakse luua kontaktpinda lõpptarbijatega. Esitusviis ei ole tähtsusetum analüüsimeetoditest, kuivõrd ainult süsteemi lõppkasutajaga koostöös on võimalik testida ja kontrollida tulemusi.

Kirjeldus-, analüüsi-, esitus- ja meetodid peavad moodustama teatud mudeli, mis tagaks seosed projekteerimisprotsessi eri faaside ja etappide vahel. Üheks võimalikuks mudeliks on ka infosüsteemide projekteerimise ja strateegilise planeerimise kompleksmetoodikasse kuuluv infovoogude maatriks.

No 705

TALLINNA TEHNIKAÜLIKOOLI TOIMETISED
TRANSACTIONS OF TALLINN TECHNICAL UNIVERSITY

UDC 681.3

M. Roost

ORGANIZATION OF DISTRIBUTED DATA RESOURCES IN JUST-IN-TIME INFORMATION SYSTEMS

Abstract

The problems of data organization in an enterprise information system on the level of automated workstations of information work are examined.
To raise the efficiency of information work in an enterprise correspondent interactive informational environments on workstations in the local network are to be created. With this aim in view an approach to data organization on the basis of dynamic and distributed data models has been elaborated.

The creation and redesign of information systems for enterprise are a current concern in the context of economic restructuring. It is ensured by the appearance of cheap personal computers and local area networks (LAN) to our markets. The local area networks of personal computers with the corresponding user-oriented software enable to create an information processing system [4] which immediately follows and affects the information work technology of an enterprise (current information system). We call this kind of information system a just-in-time system [4].

The information system is a complicated distributed system [1] which requires multilevelled approach from different aspects on several levels. That is why a great variety of information system definitions exist. In the main these treatments split into two approaches, one specifying an information system as a tool for information work or as a data processing system (data approach to infosystem) [1,3]. The information system there is separated

from information work (as it is the case in the real world). Main emphasis is placed on the data organization (processing) in a computer system.

The other approach [2, 4] regards the information system as the environment of information work (without any direct reference to that). Further, we assume it to be an environmental approach to information systems. The information work and its subjects (performers) are viewed here involved in the information system, that must guarantee favourable conditions to achieve their goals. Thus the information system must create an informational environment.

The different approaches, mentioned above, are not contradicting, however, they disagree. There is a necessity to integrate the approaches. Such integrity should be guaranteed by modeling the information system/information work using a new way. The model should integrate at a sufficiently high level the human and technical aspects of information work, the informational environment and its technological basis. The present paper attempts to formulate this approach in a corresponding model and thus to fill in the gap between the two approaches. We will try to propose the basics for data treatment in the information system which are compatible with the environmental approach and result from the latter.

Therefore we are dealing with the components of information work and focus on the relation of every component to data organization. Thus we are treating a distributed data organization, which is oriented to support an informational environment, to be more exact, the environment of information work.

2. Information work and data models

Any work or action at an enterprise is related to the information generation and use, and is thus assumed as information work. The enterprise would respectively be viewed as the information system which generates the environment for information work (for its components, finally for workers). The components of information work (and also its environment) are: subjects, objects and actions. Informational requirements call out the information work, which is performed by

units of an enterprise/information system (which have the active role of the subject/agent of information work), satisfying immediately their own actual needs for information and realizing their information work abilities. During the information work the information of these units is generated (produced and consumed).

We observe these units from the informational aspect in the given relation (role) as passive infoobjects which form the object of information work, i.e. information. An object is characterized by certain utility (or information-containability) to meet the requirements of a subject and realize its abilities. Through the common objects the common informational requirements and a common information resource will form. The above mentioned units (the objects and subjects of information work) as the components of information work are related to each other and formed by the activities of information work (with the means, realizing the latter activities). We treat the latter activities and means separately from the subjects - activity carriers, as the independent components of information work.

From the observed components of information work fundamental levels in the treatment of enterprise infosystem will also result. The organizational level of information work must be the startpoint where the goals of information work and the appropriate conditions for their realization will be formed.

The organizational units perform certain functions in information work, which are treated independently on the functional level of information work - separately from organizational units, supporting it. The separation of that level enables to optimize the information system of a subject according to the goals of information work and main functions, determined with these goals [5].

The functions of information work (actually the subject, performing these) need appropriate informational environment [6] for functioning. Such mutually related, intertwined informational environments generate the information level of a system, where the information units are viewed in the framework of their mutual relations and dynamics. The separation of this level is a necessary condition to improve

information work on both functional and organizational levels by forming the respective informational environments.

Information work proceeds in fact through the data medium. The data have their own organization (the structure and relations with information work organization), dynamics and content, that must be compatible with the information work organization, functions and objects. So, the data level as a 'physical' level on information work will be differentiated as a foundation for the infosystem (see Fig. 1) .Its characteristics should arise from the organizational, functional and informational levels of information work. This foundation must also guarantee the harmony between the levels, mentioned last. It gives a basis to view the data as an aspect of informational environment, that must in a sense embrace (model and realize) this environment.

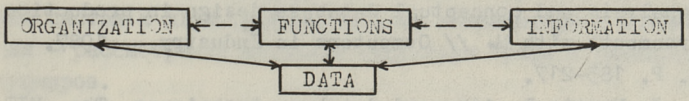


Fig. 1. Role of data models in the information work environment

So, the informational environment design/generation is the common goal for both the information work and the data organization. Thus the concept of informational environment can be concerned as an interface between above mentioned two approaches (Fig. 2). Data organization by environmental approach must be oriented to generation of informational environment for subjects of information work.

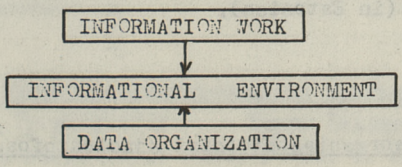


Fig. 2. The relation between information work and data organization in information system

Data as an aspect of such environment form the data environment [4]. Data environment is organized on the basis

of data models. We can look at the data model as an interface between the computer as a formalized information processing means and the nonformalized information processing that proceed in the external environment. The importance of that function arises immediately whenever information environment is needed.

Conclusion. The key concepts and features of the distributed data model are to be discussed. We hope to continue the discussion in the nearest issue of transactions.

References

1. Gray J. An approach to decentralized computer systems // IEEE Transactions on Software Engineering. 1986. Vol. SE-12. No 6. P. 684–692.

2. DiLeva A., Vernadat F., Bizier D. Information system analysis and conceptual database design in production environments with M* // Computers in Industry. 1987. No 9. P. 183–217.

3. Martin D. Advanced database techniques. The MIT Press. London, England. 1986. P. 370.

4. Roost M. Data environment of user's interface for just-in-time systems // Trans. of Tallinn Technical University. 1988. P. 116–129 (in Russian).

5. Lumberg T. On organizational "status quo" in information system design // Trans. of Tallinn Technical University. In current issue P. 65–71.

6. Roost M., Lumberg T. Information marketing and an economic unit environment // Information and Marketing. Proceedings of the Republican Scientific Seminar. 1989. P. 21–26 (in Estonian).

M. Roost

Andmeorganisatsioonist ajastusinfosüsteemides

Kokkuvõte

Artiklis käsitletakse andmeorganisatsiooni modelleerimist töökoha tasemel. Esitatakse lokaalse arvutivõrgu loomiseks sobiv lähenemine infokeskkonnale.

№ 705

TALLINNA TEHNIKAÜLIKOOLI TOIMETISED

ТРУДЫ ТАЛЛИННСКОГО ТЕХНИЧЕСКОГО УНИВЕРСИТЕТА

ОБРАБОТКА ДАННЫХ, ПОСТРОЕНИЕ ТРАНСЛЯТОРОВ. ВОПРОСЫ ПРОГРАММИРОВАНИЯ

Труды по экономике LXXII

УДК 681.3

Быстрые методы разведывательного анализа данных.
Выханду Л.К. - Труды Таллиннского технического университета. 1989. № 705. С. 3-13.

В статье демонстрируется методика применения монотонных систем к двум трудным задачам комбинаторной оптимизации - рассмотрение кликовой структуры графа и ранжирование турниров.

Библ. наименований - 5.

УДК 681.3.06.

Генерирование предложений из совокупности примеров.
Выханду Л.К. - Труды Таллиннского технического университета. 1989. № 705. С. 14-19.

В статье демонстрируется методика применения теории монотонных систем к индуктивному обучению. Представляется быстрая и простая схема генерирования предложений из совокупности примеров.

Библ. наименований - 4

УДК 681.3.06

Тестирование объектно-ориентированных программ.
Тепанди Я.Я., Троушан-Мату С. - Труды Таллиннского технического университета. 1989. № 705. С. 20-27.

В статье предлагаются принципы тестирования объектно-ориентированных программ. Рассматривается тестирование на основе текста программы (на основе критериев покрытия операторов и покрытия решений) и на основе спецификации.

Библ. наименований - 24.

УДК 681.3.06

Выбор инструментальной экспертной системы. Тепанди Я.Я., Пармаксон П.М. - Труды Таллиннского технического университета. 1989. № 705. С. 28-32.

В статье предлагаются принципы выбора инструментальных экспертных систем (ИЭС), а также описывается экспертная система для выбора ИЭС, разработанная с учетом этих принципов.

Библ. наименований - 6.

УДК 681.3

Алгебраические основы систем доверия. Хенно Я.А. - Труды Таллиннского технического университета. 1989. № 705. С. 33-43.

Основной проблемой логического программирования является вывод, характеризующий логические следствия формул только на основе свойств и структур этих формул. В статье приводится новая алгебраическая процедура вывода, которая не зависит от структуры формул. Эти новые принципы основываются на свойствах логических операций и следовательно, удобны для обобщения. Доказывается корректность и полнота предложенной процедуры.

Библ. наименований - 2.

УДК 681.3

Принципы создания математического обеспечения. Цунапуу Э. - Труды Таллиннского технического университета. 1989. № 705. С. 44-46.

В статье представлены принципы создания программного обеспечения, разработанные на кафедре обработки информации, на основе многолетнего опыта создания информационных систем. В качестве примера использования названных принципов приводится система генерации программного обеспечения информационных систем ГЕНСИ.

УДК 681.3

Применение теории монотонных систем при генерации деревьев решений. Куузик Р.Э. - Труды Таллиннского технического университета. 1989. № 705. С. 47-58.

В данной статье описан новый подход генерации деревьев решений прямо из исходной матрицы номинальных данных X(N,M) и выделения лучшего из них по определенному кри-

терию без формирования всех деревьев решений. Работа предложенных алгоритмов основывается на теории монотонных систем.

Библ. наименований - 6.

УДК 681

Генерирующая автоматизированная учебная система. Выханду П., Реги К. - Труды Таллиннского технического университета. 1989. № 705. С. 59-64.

В статье дается описание генерирующей автоматизированной учебной системы для обучения обработки списков. Для ввода системы существует написанный псевдокодом алгоритм, работу которого затем показывают шаг за шагом с помощью графического вывода.

Систему можно использовать как при представлении нового материала, так и при проверке правильности работы алгоритма, составленного самими обучающимися.

Библ. наименований - 2.

УДК 681.3.016

Проблемы проектирования оптимизационной информационной системы на предприятии. Лумберг Т.А.- Труды Таллиннского технического университета. 1989. № 705. С. 65-71.

В статье рассматриваются проблемы проектирования информационной системы на предприятии.

Затрагиваются некоторые вопросы соответствия существующей информационной системы с объективными нуждами производственной системы.

Обсуждается проблема функциональной обоснованности процессов обработки информации.

Исходя из этого предлагается идея дополнить методики проектирования информационной системы средствами информационного анализа на основе функционального критерия.

Библ. наименований - 8.

УДК 681.3

Принципы моделирования АРМ конторского типа. Эльмик Л. - Труды Таллиннского технического университета. 1989. № 705. С. 72-76.

В данной статье представлен основной принцип подхо-

да к моделированию автоматизированного рабочего места, исходящего из потребностей коммуникационного процесса.

Фигур - I, библ. наименований - 6.

УДК 681.3

Матрицы информационных потоков. Ваппер Т.Э. - Труды Таллиннского технического университета. 1989. № 705. С. 77-83.

Проектирование информационных систем является процессом, где на каждом последующем этапе используют результаты предыдущих. На первоначальных этапах проектирования строят фундамент будущей системы. Успех зависит от того, как разные методы, использованные на соответствующих этапах жизненного цикла проектирования информационной системы, образуют единую модель. Например, кроме эффективных средств проектирования нужны и способы представления результатов и тестирования, т.е. нужно иметь возможность контакта с конечными пользователями.

Одной такой моделью является и матрица информационных потоков, как часть созданной комплексной методики проектирования и стратегического планирования информационных систем.

Фигур - 4, библ. наименований - 5.

УДК 681.3

Организация данных в информационных системах. Роост М.Х. - Труды Таллиннского технического университета. 1989. № 705. С. 84-88.

В статье рассматриваются проблемы организации данных в инфосистеме предприятия на уровне рабочих мест информационной работы.

Для повышения эффективности информационной работы и всей деятельности предприятия необходимо создать соответствующие взаимосвязанные информационные среды на рабочих местах в локальной сети предприятия. Для этого в статье разработан подход к организации данных на основе динамичных и децентрализованных моделей данных.

Библ. наименований - 6.

Hind 1 rbl.