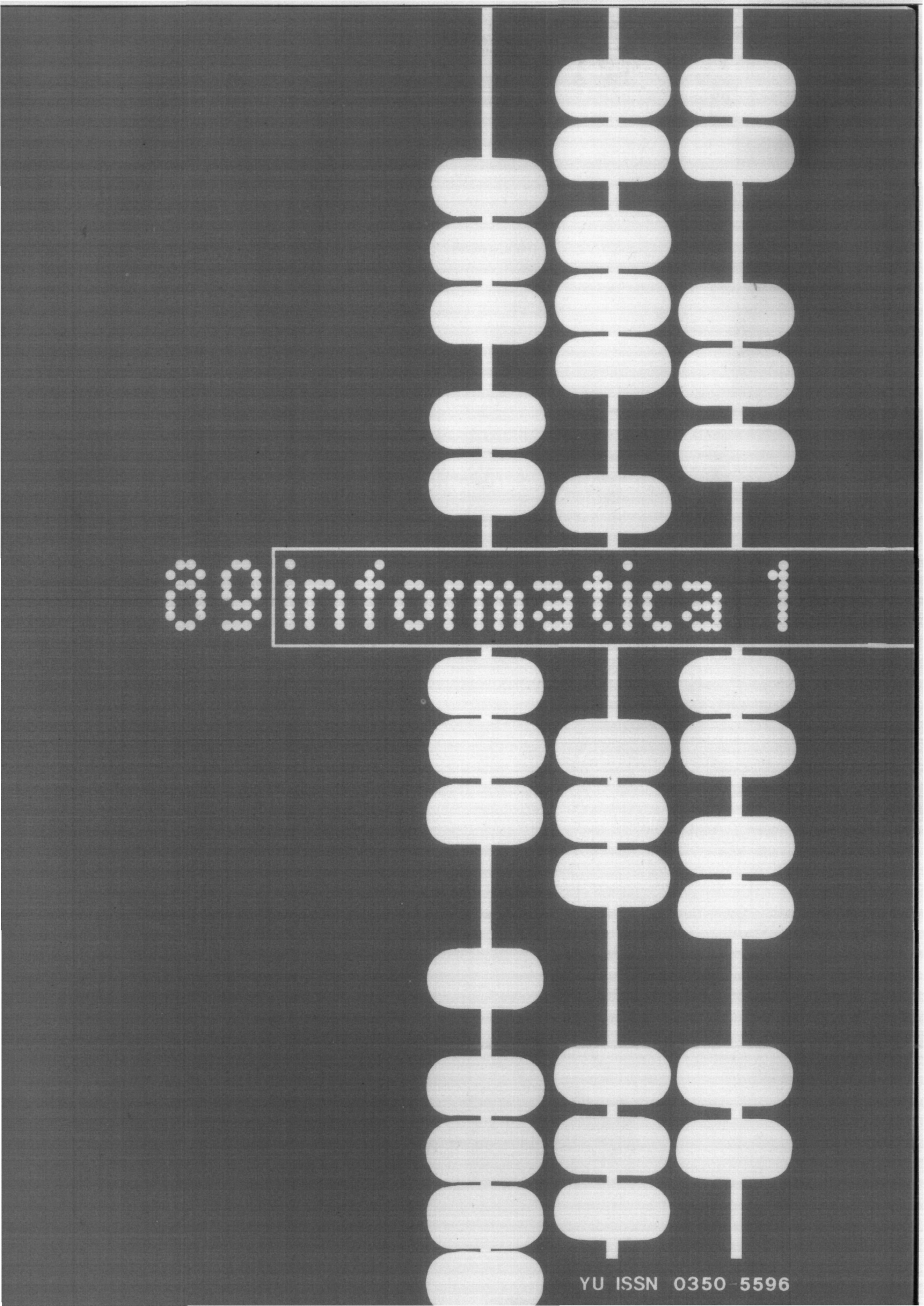


YU ISSN 0350-5596

Iskra Delta

Iskra Delta

Iskra Delta

Iskra Delta

Iskra Delta

Iskra Delta

Iskra Delta

Iskra Delta

Iskra Delta

Iskra Delta

Iskra Delta

Iskra Delta

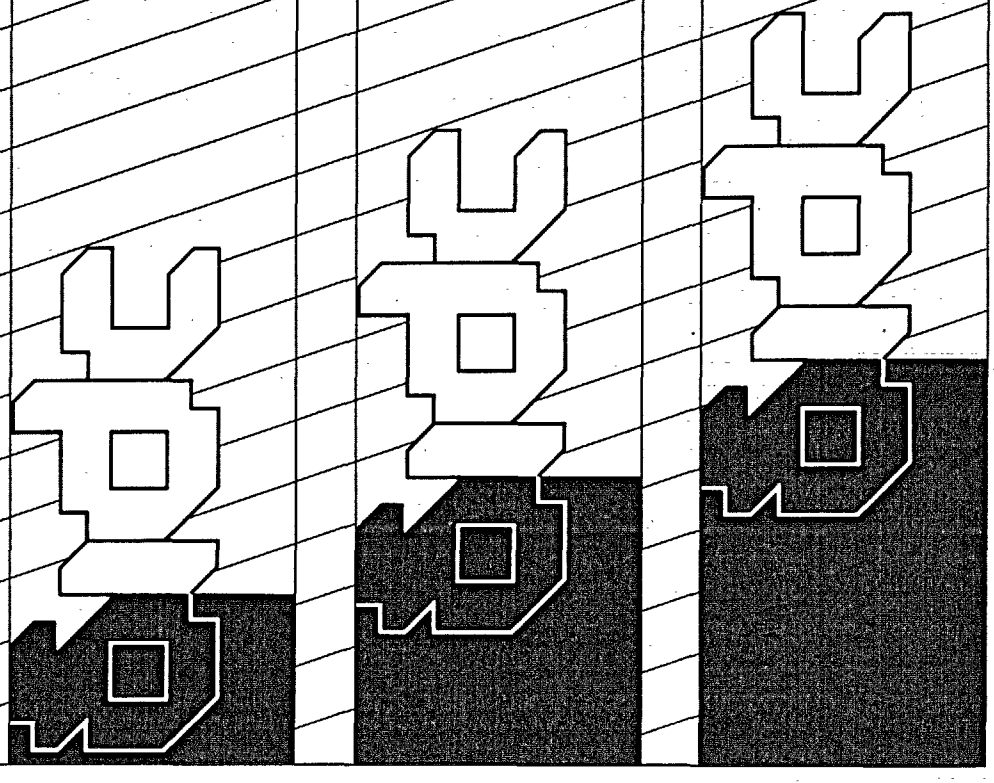
Iskra Delta

Iskra Delta

Iskra Delta

Iskra Delta

proizvodnja računalniških sistemov in inženiring, p.o.
61000 Ljubljana, Parmova 41
telefon: (061) 312-988
telex: 31366 YU DELTA



informatica

JOURNAL OF COMPUTING AND INFORMATICS

YU ISSN 0350-5596

VOLUME 13, 1989 – No. 1

Published by Informatika, Slovene Society for Informatics, Parmova 41, 61000 Ljubljana, Yugoslavia

Headquarters:

Informatica, Journal for Computing and Informatics, Iskra Delta Computers, Stegne 15C, 61000 Ljubljana, Yugoslavia
Phone: (+38 61) 57 45 54. Telex: 31366 yu delta
Fax: (+38 61) 32 88 87 and (+38 61) 55 32 61.

Annual Subscription Rate: US$ 30 for companies, and US$ 15 for individuals.

Printed by: Tiskarna Kresija, Ljubljana

CONTENTS

informatica

ČASOPIS ZA TEHNOLOGIJO RAČUNALNIŠTVA
IN PROBLEME INFORMATIKE
ČASOPIS ZA RAČUNARSKU TEHNOLOGIJU I
PROBLEME INFORMATIKE
SPISANIE ZA TEHNOLOGIJA NA SMETANJETO
I PROBLEMI OD OBLASTA NA INFORMATIKATA

YU ISSN 0350-5596

LETNIK 13, 1989 – ŠT. 1

Časopis izdaja Slovensko društvo Informatika, 61000 Ljubljana, Parmova 41, Jugoslavija

Uredništvo in uprava:

Časopis Informatika, Iskra Delta, Stegne 15C, 61000 Ljubljana, telefon (061) 574 554; teleks 31366 YU Delta; fax (061) 328 887 in (061) 553 261.

Letna naročnina za delovne organizacije znaša 48000 din, za zasebne naročnike 12000 din, za študente 4000 din; posamezna številka 16000 din

Številka žiro računa: 50101-678-51841

Pri financiranju časopisa sodeluje Raziskovalna skupnost Slovenije

Na podlagi mnenja Republiškega komiteja za informiranje št. 23-85, z dne 29. 1. 1986, je časopis oproščen temeljnega davka od prometa proizvodov.

Tisk: Tiskarna Kresija, Ljubljana

V S E B I N A

POSSIBILITIES OF PARALLEL INFORMATION PROCESING IN THE 1990s*

INFORMATICA 1/89

Descriptors: PROCESSING PARALLEL, PARSYS PROJECT, DEVELOPMENT PERSPECTIVE

Anton P. Železnikar
Iskra Delta

This essay presents a short overview of a research project in which informational logic is developed and investigated in a general and the informational parallelism in a particular manner. This investigation opens a new outlook on possibilities of parallel information processing in architectural as well as in operational philosophy and logic. In this project information is understood as an extremely parallel dynamic phenomenon, which arises in the realm of informational. Primarily the essay gives comments on the following topics: parallelism as information, informational machine, informational program, and some aspects of informational logic, its formalism, and axiomatization.

1. Introduction

The carrying out of the Parsys Project of Iskra Delta Computers has brought to light and accentuated also several questions concerning possible computer scene in the next decade. The criticism of distinguished philosophers and computer scientists in the field of artificial intelligence has been understood as a substantial change of the optimistic research initiative and as the arising of a new philosophy considering the possibilities of information processing in the future. In the leading professional journals relating computer science, new generation computing, parallel processing and the traditional field of artificial intelligence, a new philosophical and technological paradigm is coming into existence. Roughly, this paradigm tells that, for instance, the possibilities of numerous new logics of knowledge, belief, awareness, reasoning, epistemology, cognition, information, etc., will certainly influence the structure and organization of intelligent machines. Simultaneously, the appearance of new technologies exploring the concepts of biological, neural, and solid state nets will enable the implementation of extremely complexly structured and organized parallelism of information.

In this essay, our attention within the mentioned paradigm will be focused on the possibilities of parallel information processing. In this context, the necessity to redefine the notion of information is becoming essential. The common sense of the contemporary information era tells us that the notion of information is used in its narrowed meaning prevailing in the previous century, when information processing, for instance, in living organisms, was mostly a product of metaphysical imagination. Thus, together with the new philosophical paradigm in the field of artificial intelligence also the new paradigm of information has to be considered [1].

2. What is Parallelism as Information?

Nowadays we imagine information as an extremely dynamic phenomenon appearing in the cosmic, living, and artificial realm. By its belief, awareness, and commonsense reasoning, a human being can look into its own information processing by self-observation, self-investigation, self-cognition, and self-comprehension. Through such a looking into its own self, a being understands information in a much more complex way than it can be comprehended from the positions and theories of contemporary information science. For instance, the *measuring of information* within mathematically founded information theory does not concern the meaning of information or how this meaning or understanding of the meaning is coming into existence in living systems.

* This essay was read at the 1st Yugoslav-Italian Meeting on Parallel Processing, held at the International Centre for Applied Sciences, in Gradisca d'Isonzo (Italy), on September 22, 1988.

What is information as information? Information informs itself and other information and is informed by itself and by other information. This principle says that information informs actively and is informed passively. Information performs as subject and as object. Information can be viewed as operator and as operand. Information arises in its active and passive informing. In general, informing of information is a synonym for the so-called informational arising. The adverb informational is introduced to represent the described nature of information. In this context we can speak about informational form and informational process, of informational structure and informational organization, and certainly of informational spontaneity, circularity, recurrence, parallelism, sequentialness, serialness, etc., which are all informational notions and perform as information.

Informational arising means circularly spontaneous coming of information into existence, but also changing, vanishing and disappearing of information. Informational arising has to be understood as a creative component of information and its informing. Information which comes to existence is called counter-information and it has to be informationally embedded into existing information, otherwise it vanishes as informational noise. The "informationally creative" has the meaning of spontaneous, unforeseeable, autopoietic, self-structuring, and self-organizing informing of information. It becomes evident that the so-called cultural forms, for instance, philosophy, art, science, technology, ethics, etc., to mention only a few of them, can be brought into the informational framework of understanding.

It seems that informational parallelism is one of the most complex notions the mankind was ever capable to think, comprehend, and implement. Informational parallelism is a structure and organization of informationally interwoven informational forms and informational processes, which inform each other in a spontaneous, circular, recurrent, unforeseeable way. In this interweaving of informational forms and processes, complex information is coming into existence.

Currently, informational parallelism belongs to the most concealed and unrevealed phenomena and calls for philosophical and technological illumination. Parallel informational processes open the most complex spatial and temporal interweaving, dependence, and arising, as have ever been imaginable. Although, human mind in its basic structure and organization operates in a parallel-serial manner, on the higher cortical levels, in its global informational organization, for instance, in functions of belief, awareness, reasoning, world model, intention, and understanding, it appears, behaves, and experiences primarily as a serial or sequential apparatus. Since the basic parallel-serial informational structure and organization of the mind are not directly reflected in human awareness and conscious reasoning, the conscious part of human mind does not dispose of the required fundamental experience for an adequate conceptualization of informational parallelism. Thus a philosophical background of informational parallelism has to be constructed and investigated, for it cannot be discovered sufficiently in detail [2].

3. Parallel Informational Machine

The principle of parallel informational machine can be described [1] in the following sense: The parallel informational machine performs as information. Informational parallelism is inherent to informational machine. This means that its structure of forms, components, constituents, and architecture is informational, for instance, in its physical, biological, or, generally, technological constitution. If informational, the architecture of informational machine has the property of parallel informational arising, for instance, in varying of the machine connectedness, coming of structural components into existence, growing and vanishing of architecture, etc. In the same way the organization of informational machine is performing informationally in functional flexibility, controllability, programmability under changing inward and outward conditions. Architecture of an informational machine is dynamic (brain-like, for instance), is dynamically controlled, interchangeable, interweaving, arising, and depends upon the machine's environments.

The parallelism of an informational machine is a regular property. Parallelism of information is the most general form of informational interweaving. Information is always interwoven, which is only a synonym for informational parallelism. Interweaving corresponds to informational net structure and organization. Evidently, a particular strategy is needed for exploring informational nets in an optimal or economically complex parallel way. Parallel informational machine is only the synonym for various informational nets, irrespective of their specific natures, which can differ a great deal by their structure and organization.

4. Parallel Informational Program

The principle of parallel informational program can be described [1] in the following way: An informational program is simply information which spontaneously informs, embeds, arises, and counter-informs in an informationally circular manner, within an informational machine. An informational program informs, which means generates, changes itself and other informational programs during their execution and is influenced in such an informational way by other informational programs. It is used and embedded into an informational machine for production of information. This information can be, for instance, intelligence, specialized creativity, expertise, problem solving, dedicated informational functions, etc.

Evidently, there is an essential difference between a computer program and an informational program. The former is algorithmic,

mathematical, procedural and informationally static, whereas the latter is informational, intelligent and informationally dynamic. As a rule, a computer program has a stable, non-variable program structure and program organization. Its definition (declaration) cannot be changed dynamically during its execution, by the parallel execution of itself and other programs, data, etc.

An informational program performs as information. In this respect, such a program is also an informational object (operand), which can be informationally changed during its execution. A typical computer program is always performing as a subject, by which non-program objects can be changed and can arise as results of its performing. In principle, the request to informatize a program concerns essentially different programming tools from those that are in use today.

5. Parallel Informational Logic

At its beginning, parallel to the Parsys Project, the study of informational logic was initiated, with the goal to construct adequate tools concerning informationally parallel machines and programs in particular. This study of informational logic has to be understood as a free-lance undertaking within IDC, which embraces the philosophy of the redefined notion of information and the so-called axiomatization of informational logic. The logic, which in its main part deals with informational parallelism, is being developed up to the form of an informational axiomatic system. In this regard, the system is not a usual axiomatic approach of pure mathematical doctrines, but above all, incorporates the so-called informational principles. This logical theory represents a generative axiomatic system the intention of which is to preserve semantically the arising nature of the redefined notion of information.

The new formalism introduces several symbols for operators with the already known, but also a new semantics. Two types of variables appear: the operand and the operational ones. Operational variables can be particularized and universalized according to the needs, goals, and applications of a case. In operands, also the so-called functional operators can appear. In these cases operands perform operationally in an implicit manner. The most general operational variable has the meaning of informing. This operator, called informational metaoperator, can be particularized and univesalized, i.e., substituted in a logical expression in a non-uniform way. The axiomatic system of informational logic is constituted by basic informational operand and operator variables and informational constants of both types, of formation rules defining the syntax of informationally well-formed formulas, of informational axioms, and of the so-called transformation rules for transformation of axioms and formulas.

The axiomatic basis of informational logic is still in the phase of development and formal construction [2, 3]. Operators of general parallel informing are, for instance, informing in parallel, possible informing in parallel, necessary informing in parallel, non-informing in parallel, informational carrying off, informational bringing, blocking to carry off, blocking to bring, sending and informing, receiving and informing, non-sending and non-informing, non-receiving and non-informing, causing of informational appearance, non-causing of informational appearance, coming into existence, non-coming into existence, causing informational end, coming to end, non-causing informational end, non-coming to end, choosing among informational alternatives, choosing and informing, being informationally impressed or memorized, recalling, disintegrating and informing, informing similarly, interrupting, breaking down, enriching, cyclical parallel informing, etc.

6. Conclusion

The theory of parallel processing within informational logic can be the basis for conception, design, and application of future parallel computing systems. In this context, the development of informational philosophy and to it adequate axiomatization and formalization of informational concepts are becoming necessity of the future. This could be the way to really intelligent systems which would cope with living minds and living systems.

References

1. Zeleznikar, A. P., *Principles of Information,* Cybernetica 31 (1988) 99-122.

2. Zeleznikar, A. P., *Informational Logic I,* Informatica 12 (1988), No. 3, 26-38.

3. Zeleznikar, A. P., *Informational Logic II,* Informatica 12 (1988), No. 4, 3-20.

SYNTACTIC PARSING AND PLOTTING OF MATHEMATICAL EXPRESSIONS

INFORMATICA 1/89

Descriptors: SYNTAX, ANALYSIS, TEXT NATURAL, SOFTWARE, TREE GRAMMAR, MATHEMATICAL ANALYSIS, LANGUAGE ANALYSIS

Nikola Bogunović, Institut R. Bošković
Zagreb

The paper presents the parsing problem of a simple context free language. The "language" sentences are mathematical expressions with one variable. A computer program parses the expression according to the developed context free grammar rules. Upon building a parse tree, the program evaluates the expression over a given range of variable values, and plots the result on the screen. Even though the program is developed on a PC AT personal computer, it is highly portable since the C programming language is used, and graphics hardware dependent routines are removed in a separate module.

SINTAKTIČKA ANALIZA I GRAFIČKI PRIKAZ MATEMATIČKIH IZRAZA

U radu je predstavljen problem analize rečeničnog sloga u kontekstno slobodnom jeziku. "Jezičnu rečenicu" predstavlja matematički izraz s jednom varijablom. Računarski program razlaže izraz u skladu s razvijenim kontekstno slobodnim gramatičkim pravilima. Program gradi stablo sastavnih dijelova matematičkog izraza, nalazi vrijednosti izraza za dati niz vrijednosti varijable, i grafički prikazuje rezultat. Iako je program razvijen na računalu PC AT, jednostavno je prenosiv na druga računala, jer je pisan u višem jeziku (C), a grafičke, sklopovski ovisne rutine premještene su u odvojen programski modul.

INTRODUCTION

Language parsing has traditionally been one of the most intriguing research areas of artificial intelligence. The problem here is to take the information provided from the outside world and translate it into a precise internal representation. By the internal representation we mean a semantic representation, a common data structure produced or operated on by various program modules. Even though, a common data structure of internal representation may have different forms, it is assumed that the translations from one to another is easy, and all forms are variants of the same abstract representation.

The application of language parsing, in the context of this paper, is directed to engineering problems, e.g. intelligent industrial process control, rather than attempt to solve an over aspiring and not very well defined problem like automatic language translation. It is more sensible to work on the internal representation generation, since this is an intermediate point between words and actions.

The problem of language parsing can be divided into three areas, with the apparent ambiguity at each level [1]:

1. acoustic-phonetic: time domain and frequency domain analysis of the incoming sound, and translation the input into words.
2. morphological-syntactic: taking words and establishing the syntactic form of the utterance.
3. semantic-pragmatic: finding out the meaning from the syntactically analyzed utterance.

In this paper we will concentrate on the problems of second and third level only. Our goal is to develop an internal representation from the correctly received input stream of information, looking at the major data structures the computer program uses. The problems at first level, and partially at second, can be bound loosely to speech recognition, with major research advances and results given in [2] and [3].

A computer program that translates from any natural language to internal representation, must in the first step syntactically analyze, or parse, the sentence. In the process, one needs to know the rules of syntax for the language, specifying the legal syntactic structures for a sentence.

A parsed sentence is usually presented in a tree structure known as parse tree or phrase marker. Specifying the structure of a sentence in a formal knowledge structure, by a series of production rules i.e. grammar, could be far too complex for any natural language [4]. The grammar must be, in that case, context-free, indicating how to replace a nonterminal node of the parse tree with lower constituents, without any reference to the context in which the nonterminal node finds itself [5].

Since the question, whether a natural language is context-free or not, is yet to be answered and a context-free grammar for such a language devised, in this paper we will concentrate on a much easier task of parsing the arithmetic expressions, which are inherently controlled by a context-free grammar. Nevertheless, the presented parser program applies the very same methodology of natural language parsing, and develops a parse tree of an arithmetic expression of considerable complexity.

A rough outline of context-free parsing can also be found in [6]. This paper extends the presented idea with different and more efficient algorithms and data structures available in C language, introduces complex expressions based on a family of new functions, evaluates the function of one variable and develops a graphics interface for the presentation of the evaluated function over a given range of values on the terminal of a PC XT/AT or PS/2 personal computer. The computer program was written in Microsoft C, and linked with an assembly language program to run industry standard monochrome or colour graphics routines.

THE SYNTAX OF A LANGUAGE

A context-free language is one whose syntax can be specified by a set of rules, usually called productions, or simply grammar. A context- free derivation of a sentence always start with a nonterminal node of the parser tree, or a nonterminal constituent, i.e. with a node that appears only in the interior of the tree structure, and not in the final sentence. Each nonterminal node is then replaced by the right hand side of the rule, until we have only terminal nodes, or terminal constituents, i.e. nodes that appear in the final sentence.

A very simple context-free grammar might have the following rules:

```
sentence(s) --> noun-phrase(np) verb-phrase(vp)
verb-phrase(vp) --> verb(v) noun-phrase(np)
verb-phrase(vp) --> verb(v)
noun-phrase(np) --> determiner(d) noun(n)
noun-phrase(np) --> proper-noun(prn)
```

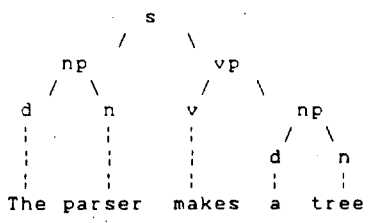
The last two rule pairs may be combined with the logical operator OR (¦):

```
vp  -->  v np  ¦  v
np  -->  d n   ¦  prn
```

The syntax of grammar rules can be also described by the recursive set of grammar rules themselves:

```
grammar_rule --> grammar_head [-->] grammar_body
grammar_head --> nonterminal_node
grammar-head --> nonterminal_node terminal_node
grammar_body --> grammar_body grammar_body
grammar_body --> grammar_item
grammar_item --> nonterminal_node
grammar_item --> terminal_node
```

The application of the context-free grammar to a simple sentence like "The parser makes a tree." is exceedingly clear from the following parse tree:



This simple example creates more questions than it really indicates the solution of the problem. It is extremely difficult to express every rule of grammar with context-free rules. The problem of number agreement (singular-plural), morphology (differences between go, goes, going), reflexivization (reflexive pronouns), imperatives, passive case, etc. are just the few most common. One can add weakly context-free structure in the variation of the above example with the sentence like "It does."

Programming languages, on the other hand, are context-free, and principal compiler task for such a language is to parse it. In that case we may even talk about the efficiency of parsing, optimal parsing, and so forth. The expansion of RISC computer architecture, as a contemporary industry trend, causes the compiler construction and language parsing problem to be of outmost importance.

Mathematical expressions are the most simple case of finite character strings whose syntax can be captured in a context-free grammar. Since our primary goal in this paper is to show the principles of a working parser, we have constrained the presented application to a "language" that can be described with only a few production rules.

Let us assume a mathematical equation with one variable, floating point constants, four arithmetic (binary) operators, and five unary operators, with left to right evaluation in the traditional fashion. An example of such an expression is:

```
f(x)=2.5+3.4-4.5sin(cos(5.8x+2.2))          (1)
```

We have decided upon these basic set of operations with the following order of precedence:

```
    -  :  unary minus
sin(x) :  sine function
cos(x) :  cosine function
log(x) :  natural log
exp(x) :  exponential
    *  :  multiplication
    \  :  division
    +  :  addition
    -  :  subtraction
```

The syntax of the expression (1) can be captured in the recursive set of context-free grammar rules. We may use the notation introduced at the beginning of this paper or instead, we may use the familiar and traditional Backus-Naur form, from the computer science literature:

```
<expr> ::= <term> | <term>+<expr> |
           <term>-<expr>
<term> ::= <factor> | <factor>*<term> |
           <factor>\<term>
<factor> ::=  <variable> | <number> |
              -<factor> | sin<expr> |
              cos<expr> | log<expr> |
              exp<expr> | (expr)
```

It is evident that the functions sin, cos, log, and exp are implemented as unary functions, like unary minus. Implied multiplication, used in the input expression, is later changed to explicit (*).

The application of these production rules to the equation (1), is presented in Fig.1. Parsing starts with the <expr>, which according to the first rule of our grammar may have three forms: a <term>, <term>+<expr> or a <term>-<expr>. Since it is obviously a <term>+<expr>, further application of grammar rules to <term> part yields a <factor>, then a <number> which is a floating point constant.

Parsing the other part needs a recursive application of the same rules. Since it is an <expr>, we apply the first rule again, which yields <term>-<expr>. The <term> part is a <factor>, a <number> and a constant. The <expr> part is a <term>, which is a <factor>*<term>. The process proceeds until the terminus of all branches is reached, yielding a <number> or a <variable>.

Once a parse tree is constructed, the expression may be evaluated starting at the top of the tree by recursively evaluating left and right branches, and then performing addition or subtraction at the top.

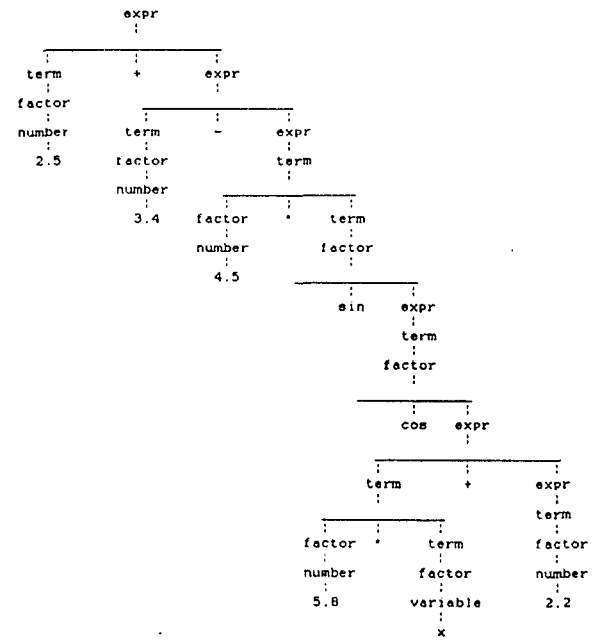


Fig.1 The parse tree of equation (1).

DATA STRUCTURES AND ALGORITHMS

The principal elements of a parse tree are nodes. Looking at Fig.1, we may deduce that there are four kinds of nodes. A node is either a number (a floating point constant), a variable (x), a unary operator node (-,sin,cos,log,exp), or a binary operator node (+,-,*,/). In our case the root node is a binary operator (+) with left and right operands, i.e. <term>+<expr>, according to the first grammar rule. Left operand will be parsed into <factor> --> <number> --> 2.5. Right operand will be parsed recursively starting with the first grammar rule again. All four kinds of nodes can be captured in a structure, in C language sense (7), with the following components:

```
struct node {
        int tag;
        char operator;
        float number;
        struct node *left_operand;
        struct node *right_operand;
             } TREENODE, *TREEPTR;
```

Integer tag identifies a node as a number (tag=0), a variable (tag=1), a binary operator (tag=2), or a unary operator (tag=3). Character operator identifies operator as +, -, *, /, sin, cos, log, or exp. If the node is a number, the floating point value is held in the "number" structure member. If the node is a binary operator, pointers left_operand and right_operand point to the left and right "child" nodes (structures). If the node is a unary operator, only left_operand pointer is used. It is absolutely important to note that the root node structure embeds the whole parse tree, because left and right operands, as the structure members, are pointers to the structures of the same type as the root node itself. This recursive declaration of a node is correct, as given in [7]. Typedef TREENODE and TREEPTR, define a node type structure and a pointer to such a node type structure.

At the beginning of the program, the string, which corresponds to the input equation, is subject to the preprocessing operation. The string is converted to lower case, and all surplus spaces are removed. Since sin, cos, log, and exp functions are implemented as unary operators, they are stripped to a single unique character operators (s,c,l,e). Finally, implicit multiplication is changed to explicit. After the preprocessing phase, our equation (1) would fill an array of characters that would look like:

```
2.5+3.4-4.5*s(c(5.8*x+2.2))          (2)
```

In the next step a parse tree is constructed. Any expression, if not constrained with parenthesis to a subexpression, must start with a term, which must start with a factor (number,variable or unary operator applied to <expr>). In the process of building a parse tree, we actually make the instances of node structures defined earlier. As already stated, the root node contain pointers to the neighbouring structures, and in essence embeds the whole tree. There is an initial function expr(), which calls the function term(), which finally calls function factor(). These functions mirror our grammar rules. The function factor() analyses the beginning of the string. In our case it will find a number 2.5 (a constant), and it will make a number node and return a pointer to its caller. The callee, function term(), will further analyze the string to find if there is a multiplication (*) or a division (/) sign according to the second grammar rule. If not, which happens in our case, term() will return a pointer of a number node to expr(). The expr() function will analyze the string further and find an addition sign, hence the root node is a binary operator node with left operator already established (previously found number node). The right node will be found by a recursive call to expr() function again.

To illustrate the creation of the number node structure, we give the function numbernode(), which is called by the factor() after it has extracted the number and its value from the beginning of the string.

```
#define new()
    (TREPTR) calloc(1,sizeof(TREENODE))

/* This is a global creation of the space
which will hold a node, and return a
pointer to that space. */

#define NULL 0

TREEPTR numbernode(value)
/* take the number value and return a
pointer to a structure */
float value;
/* the type of passed parameter */
{ TREEPTR n;
/* declaration of the pointer */
        n = new();
/* creation of an empty struct. */
        n->tag = 0;
/* it is a number node */
        n->number = value;
/* fill in the value */
        n->left_operand = NULL;
/* numbernode has no neighbours */
        n->right_operand = NULL;
return(n);
/* returning a pointer */
}
```

Since this paper describes the equation parser and plotter, we have included in List 1, an evaluation function which, for a given variable value, will traverse through the parse tree in a recursive search fashion, finding the value of the whole expression. The function eval(root_node_pointer,x) will test the tag of the root node and act accordingly. If the node is a unary or binary operator node, eval() will call itself with new pointers.

```
float eval(n,x)
TREEPTR n;              /* 1st passed parameter is a pointer
                           to the root node structure */
float x;                /* 2nd parameter is a variable value */
{       float op1,op2;
        switch (n->tag)
        { case 0:               /* it is a number node */
                return(n->number);
                break;
          case 1:               /* it is a variable node */
                return(x);
                break;
          case 2:       /* it is a binary operator node */
                op1 = eval(n->left_operand,x);
                op2 = eval(n->right_operand,x);
                      switch(n->operator)
                      {  case "+":
                                return(op1+op2);
                                break;
                         case "-":
                                return(op1-op2);
                                break;
                         case "*":
                                return(op1*op2);
                                break;
                         case "/":
                                return(op1/op2);
                                break; }
           case 3:          /* it is a unary operator node */
                switch(n->operator)
                {       case '-':
                                return(-eval(n->left_operand,x));
                                break;
                        case 's':
                                return(sin(eval(n->left_operand,x)));
                                break;
                        case 'c':
                                return(cos(eval(n->left_operand,x)));
                                break;
                        case 'e':
                                return(exp(eval(n->left_operand,x)));
                                break;
                        case 'l':
                                return(log(eval(n->left_operand,x)));
                                break;
                }
        }
}
```

List 1. Evaluation of the expression.

The presented function eval() is only a basic skeleton of the implemented function, because one has to take great care how binary and unary functions are defined (no negative values for log, divide with zero, etc.), and whether a parenthesis is encountered indicating a subexpression.

Finally, after obtaining domain and value points of the equation, we can display it graphically. The graphics functions greatly depend on the used hardware and can not be given generally. However, since industry standards like personal computers PC XT/AT and PS/2 are readily available, we will show the principles of implementation some simple graphics procedures for these computers. Even within the PC and PS family of computers, graphics standards vary from 320x200 pixels to an impressive 1024x768 pixels (with additional advanced display adapter). In this paper we have chosen to show Hercules monochrome graphics implementation, in belief to represent a popular, yet fully acceptable medium resolution (720x348) graphics standard.

Hercules graphics functions library is a set of memory resident routines, set up by INT10.COM, a program supplied and copywrited by the vendor [8]. We have chosen to implement graphics routines in the assembly language and link them with the main C program to achieve maximum portability. The assembly language program treats Hercules graphics functions as the extension of the standard display control software interrupt procedures (int 10h). All parameters are simply loaded in registers, with the function code in AH register, prior to int 10h call. Unlike the original set of functions within int 10h group, only segment registers are preserved, along with all registers used to pass parameters.

An example of assembly language function, which moves the cursor to the x,y position (move(x,y)), is given below. The caller from the C program will leave x and y coordinates, as parameters, on the stack. It was assumed that C program will run on a PC XT/AT in the small model ("Microsoft" restriction to 64K byte), hence near procedure type.

```
_text segment byte public 'code'
      assume cs:_text
; definitions as required by Microsoft C
      public _move
_move   proc    near
        push    bp
        mov     bp,sp
        push    di,
        mov     di,[bp+4]
; get x from stack
        mov     bp,[bp+6]
; get y from stack
        mov     ah,40h
; it is function "move"
        int     10h
; call function
        pop     di
        pop     bp
        ret
_move   endp
_text   ends
        end
```

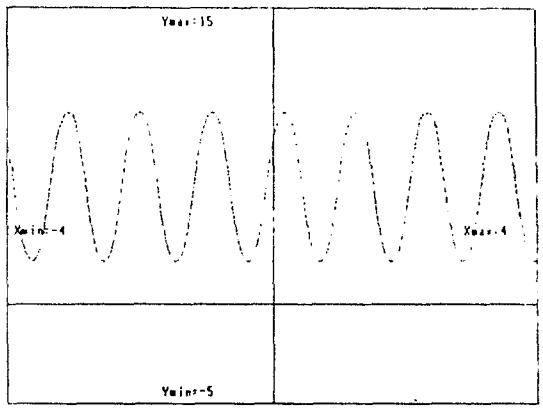


Fig.2 The graph of equation (1).

CONCLUSIONS AND REMARKS

We have studied string parsing techniques, applied to the simple mathematical equations. The syntax of these strings can be described by an elementary context free grammar. Nevertheless, the same principles apply to a broad range of languages described by context free grammars.

To illustrate the parsing, evaluating and plotting operations of the working program, we have presented the graph of the equation (1) in Fig.2. The function is plotted over a domain range (-4,+4). The scale of ordinate values is appropriately chosen to display points between -5 and +15. The program prompts for the scale before it plots the function. By changing the coordinate scale one can easily zoom, scroll and pan the graph, sustaining the same resolution.

It is worth noting that the program embeds implicit precedence rules (multiplication and division before addition and subtraction), and enforced precedence by parentheses, according to the given grammar rules.

REFERENCES:

1. E.Charniak, D.McDermott, Introduction to Artificial Intelligence, Addison-Wesley, Reading, Mass., 1985.

2. J.L.Flanagan, Speech Analysis, Synthesis, and Perceptions, Springer Ferlag, New York, 1972.

3. S.E.Levinson, L.R.Rabiner, M.M.Sondhi, An Introduction to the Application of the Theory of Probabilistic Functions of a Markov Process to Automatic Speech Recognition, Bell Syst. Tech. J., Vol. 62, No. 4, 1982.

4. N.Chomsky, Syntactic Structures, Mouton, The Hague, 1957.

5. A.V.Aho, J.D.Ullman, The Theory of Parsing, Translation and Compiling, Prentice-Hall, Englewood Cliffs, N.J., 1972.

6. J.Amsterdam, Context-free parsing of Arithmetic Expressions, Byte, Vol. 10, No. 8, August 1985.

7. B.W.Kernighan, D.M.Ritchie, The C Programming Language, Prentice-Hall, Englewood Cliffs, N.J., 1978.

8. GRAPHX V1.1 Manual, Hercules Computer Technologies, 2550 Ninth Street, Berkeley, CA 94710, USA.

RELATIONAL SCHEMA DESCRIPTION LANGUAGE

INFORMATICA 1/89

Descriptors: RERLATION BASES, PROGRAMMING LANGUAGE, SCHEMA DEFINITION, INTERPRETERS, SDL LANGUAGE

Radojko Miladinović
Dušan Velasević

ABSTRACT: In this paper a schema description language for relational databases is described. The language provides a schema description on which any query language can be defined. The implemented multiuser incremental interpreter for that language is also described.

SADRZAJ: U ovom clanku opisan je jezik za definisanje seme u relacionim bazama podataka. Taj jezik omogucava definisanje seme na kojoj bilo koji upitni jezik moze biti definisan. Za taj jezik realizovan je visekorisnicki inkrementalni interpreter.

INTRODUCTION

Since 1970., when E.F. Codd had defined the relational data model [4,5,6,7], a lot of relational database management systems (RDBMS) was developed and implemented. All these systems can be classified into two groups according to the way of the schema definition. The systems from the first group, for example Relational Database Management System [9,10], have a stand alone schema definition language. The systems from the second group do not provide such a language: the schema definition is realized as a function of data sublanguage, i.e. query language. SYSTEM R [1,3,11] belongs to this group of RDBMS.
The relation between schema description language (SDL) and other languages in RDBMS (query language, data manipulation language - DML, subschema description language and physical database description language) represents a special problem in the database design. All these languages can be implemented as stand alone languages or as extensions of standard programming languages. The majority of RDBMS does not have the independent SDL, query language and DML; in fact, the data definition, data manipulation and query facility are realized as the functions of the special language called the data sublanguage. The data sublanguage can be implemented as a stand alone programming language or as an extension of the host language. If it is implemented as a stand alone language, it is usually called the query language.
Although the most of modern RDBMS have not a separate SDL, there exists a need for such a language which should be general, simple, structured and user-friendly. This language should provide the means for a schema description over which any query language can be defined. A complete functional independence of the schema description process from the data manipulation and queries is achieved in this way. Bearing this in mind, we developed a new SDL and multiuser incremental interactive interpreter for that language.
The language design was influenced by the general principles applied to the other programming languages. We especially emphasized the language reliability, precise syntax and semantics description of the language, orthogonallity and language independence. The SDL structure was designed bearing in mind that the language should be interactive. For this reason, the commands for direct communication with the users are defined, each statement must be written in one line and the language is structured to provide better readability and documentability of SDL programs.

BASIC LANGUAGE ELEMENTS

The following notation is used in the description of SDL elements:

{ } - Braces indicate that one of the elements enclosed must be specified.
[ ] - Square brackets indicate that one of the elements enclosed may be optionally specified.
... - Ellipses indicate that the immediately preceding part of the format may be repeated.
- Upper case words are SDL reserved words.
- Lower case words indicate the information that should be supplied by the user.

Language alphabet

The complete SDL character set consists of 52 characters. All SDL characters are presented in table 1.

Table 1.

character	name
A,B,...,X,Y,Z	uppercase letters
0123456789	decimal digits
+	plus
-	minus
*	asterisk
/	slash
<	less than
>	greater than
=	equals
(	left parenthesis
)	right parenthesis
.	point
,	comma
:	colon
"	quotation marks
$	dollar sign
_	hyphen
	space

Identifiers

A SDL identifier (or word) is a character string that forms a user defined or a reserved word of not more than 20 characters. It can be any combination of letters, digits and hyphen sign, but hyphen cannot be the first or the last character of the identifier. A reserved word has specific meaning and may be used only in the manner presented in the statement format. Reserved words are chosen carefully so they enable writing of reliable and synoptic programs. The list of the reserved words is given in APPENDIX A.

The user defined words are SDL words that must be supplied by the user to satisfy the format of the statements. The user defined words are variable names, constants, function names, procedures and comments.

Constants and variables

The classification of constants and variables according to their format and type is given in Fig 1.

Variables in SDL are relations and attributes. Attributes are one-dimensional arrays with elements of the same type. Relations are considered as two-dimensional arrays in which all elements in the same column are of the same type; elements in the same row are not obligatory of the same type. The attribute type is explicitly defined by a particular statement in the SDL, and the relation type is implicitly defined through attributes which constitute that relation. The external representation of constants is very simple. It corresponds to the syntax representation of integer and real numbers. For example:

1000, -3.5 0.75E10, 0.32D11, -34E10

The internal representation of constants depends on the context in which constants appear. For example, if variable A.B in the relational expression

A.B > 10000

is declared as a real variable of the extended precision (binary floating point format), then the constant 1000 is presented in the same format.

Alphanumeric strings (literals) are externally represented as a character strings delimited by the quotation marks. The internal representation of literals is completely the same if the data compression is not applied.

Arithmetic expressions

Arithmetic expressions are used to compute new attribute values during updating. Arithmetic expressions are formed with arithmetic operands and arithmetic operators. An arithmetic operand may be a numeric constant and a variable (attribute). Arithmetic operators specify a computation to be performed using the values of arithmetic operands; they produce a numeric value as a result. The operators are: addition(+), subtraction(-), multiplication(*) and division(/). Arithmetic expressions are evaluated in an order determined by a precedence associated with each operator. The precedence of the operators is: first (* and /) and second (+ and -). Parentheses can be used to override the normal evaluation order. The attributes which appear in arithmetic expressions must be of type real or integer. The attribute names in arithmetic expressions have the format:

$$[[\{\text{OLD} \atop \text{NEW}\}] \text{ relation\_name.}] \text{ attribute\_name}$$

The relation name must be specified in front of attribute name if an attribute appears in more than one relation. If an attribute belongs to only one relation, then the relation name is optional. The reserved words OLD and NEW can appear in front of attribute name. These words denote attribute values before and after updating. If they are omitted the immediate attribute values are considered. The examples of arithmetic expressions:

ORDER.QUANTITY - PRODUCT.QUANTITY
OLD EMPLOYEE.SALARY - 1000

Relational expressions

A relational expression may be a simple relational expression, or may be a combination of simple relational expressions, functions and logical operators. A simple relational expression consists of two operands separated by a comparison operator. The comparison operators are: less than(<), greater than(>), less than or equal to(<=), greater than or equal to(=>), equal to(=) and not equal to(<>). An operand may be a constant and a variable (attribute) of any type. An attribute name has the same format as in an arithmetic expression. The logical operators are AND and OR. The precedence of the logical operators is: first AND and second OR. In a relational expression, the simple relational expressions are evaluated first to obtain their values. Evaluation of relational expressions is performed according to an order of precedence assigned to logical operators. Logical operators of equal rank are evaluated from left to right. Parentheses may be used to alter the normal sequence of evaluation, just

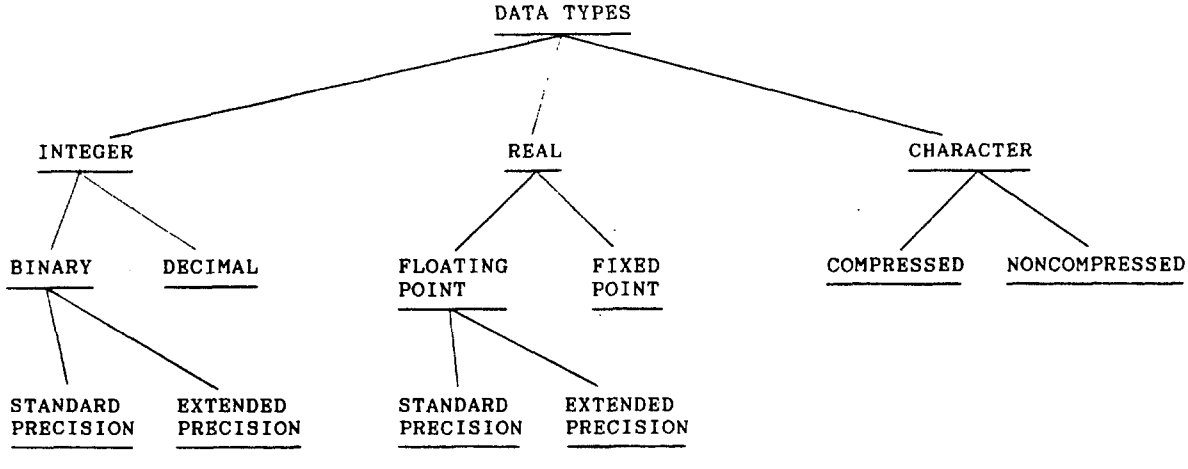


Fig 1. Data typesdefined in SDL

as in arithmetic expressions. The examples of relational expressions:
EMPLOYEE.SALARY > 2000 AND EMPLOYEE.SEX = "M"
OLD EMPLOYEE.SALARY < NEW EMPLOYEE.SALARY

Functions
The functions in SDL are used to define relational expressions that appear in more than one statements. They are defined through FUNCTION statements. The functions also enable the decomposition of long relational expressions, which cannot be written in one line, into several logical parts. Each logical part must represent the complete relational expressions defined by a separate FUNCTION statement. In that way, the long statements containing relational expressions can be spread over several lines. This must be done because a SDL statement can be written in only one line. For example:
(EMPLOYEE.JOB="PILOT" OR
EMPLOYEE.JOB="DRIVER") AND EMPLOYEE.SEX="M"
AND EMPLOYEE.AGE>60
This relational expression cannot be written in one line, so function FUN1 is defined as follows:
FUN1:=EMPLOYEE.JOB="PILOT" OR
EMPLOYEE.JOB="DRIVER"
The relational expression becomes now:
FUN1 AND EMPLOYEE.SEX="M" AND EMPLOYEE.AGE>60
The names of other functions can appear in the function definition.

LANGUAGE DESCRIPTION

Programs written in SDL have specific structure due to the fact that SDL is used only for the relational schema description. The program structure is determined by the structure of the relational database description. Each SDL program contains four type of entries: schema entry, domain entry, attribute entry and relation entry. The entries appear in the program in cited order. Each entry consists of a sequence of statements. Some statements can appear in different entries. The entry specification contains the following parts:
1. Narrative description of entry function
2. General format in which all statements of the entry are presented
3. Description of each statement
The statement specification consists of the following parts:
a. Narrative description of statement function
b. General format that defines the statement parts and their functions
c. Language rules that explain the usage of the statement

Line format
A SDL statement must be written in one line, i.e. the continuation of the statement in a few succeeding lines is not allowed. If any statement can't be written in one line, that statement is divided in several logical parts which should be defined as functions. One or more spaces can appear at the beginning of each line in other to achieve a better program structure and readability. All lexical entities in a statement may be separated by one or more spaces.

Data base example
The application of each statement described is illustrated by the examples which are composed of the following relations:

```
PRODUCT(CODE,    NAME,    PRICE,    QUANTITY)
        DOMCODE DOMNAME DOMPRICE DOMQUANTITY

EMPLOYEE(CODE,    NAME,    SEX,    SALARY)
         DOMCODE DOMNAME DOMSEX DOMSALARY

SUPPLY(CODEPRO, CODESUP, DATE,    QUANTITY)
       DOMCODE  DOMCODE  DOMDATE DOMQUANTITY

DEPT(CODE,    NAME,    ADDRESS,    EMPNO)
     DOMCODE DOMNAME DOMADDRESS DOMNUMBER
```

As we can see, an attribute can appear in more different relations. For each attribute, domain on which the attribute is defined is also represented.

SCHEMA ENTRY

1. The schema entry uniquely identifies schema by its name. Besides, the textual description of the schema contents and cripto protection method applied, if any, is given in the schema entry.
2. Entry format
   SCHEMA statement
   [DESC statement]
   [CRIPTO_PROTECTION statement]
   {domain entry}
   {attribute entry}
   {relation entry}
   ENDSCHEMA statement
3. Statements description
   The schema entry begins with SCHEMA statement and ends with ENDSCHEMA statement. The statements that define the schema are located between the SCHEMA statement and first domain entry in the schema. The order of these statements is irrelevant. In the schema entry, the entries of all domains are defined first, then the entries of all attributes, and finally the relation entries.

SCHEMA statement
a) The SCHEMA statement uniquely identifies schema by its name.
b) Format
   SCHEMA schema_name
c) Language rules
   - Schema name must be unique in the database.

DESC statement
a) The DESC statement is used to describe the content and function of schema, domain, attribute and relation. This statement can appear in any entry in the schema.
b) Format
   DESC comment
c) Language rules
   - An arbitrary number of DESC statements can appear in the schema entry
   - Comment in DESC statement can contain any character from SDL character set.
   - This statement has no meaning for interpreter.
   - If a comment can't be written in one line, several DESC statement must be used.

CRIPTO_PROTECTION statement
a) The CRIPTO_PROTECTION statement can appear in the schema and in relation entry. If it appears in schema entry, all relations in the schema are cripto protected. The cripto protection can be implemented by a special procedure defined by the user or as a standard function in RDBMS.
b) Format

$$\text{CRIPTO\_PROTECTION} \left\{ \begin{matrix} \text{procedure\_name} \\ \text{SYSTEM} \end{matrix} \right\}$$

c) Language rules
   - Only one CRIPTO_PROTECTION statement can appear in the schema or relation entry.

- If the option SYSTEM is specified, the cripto protection is implemented as one of the activities of RDBMS. If the procedure_name is specified, the cripto protection is implemented by a special procedure.

ENDSCHEMA statement
a) This statements denotes the end of the schema entry, i.e. the end of the whole schema description program.
b) Format
   ENDSCHEMA

DOMAIN ENTRY

1. The domain entry uniquely identifies the domain by its name. Besides, the domain mode, the names of all attributes defined over that domain and the domain units are specified in the domain entry.
2. Entry format
   DOMAIN statement
   DEFINES statement
   [DESC statement]
   MODE statement
   [UNIT statement]
   ENDDOMAIN statement
3. Statements description
   The domain entry begins with the DOMAIN statement and ends with the ENDDOMAIN statement. The order of other statements in the entry is irrelevant.

DOMAIN statement
a) The DOMAIN statement uniquely identifies domain by its name.
b) Format
   DOMAIN domain_name
c) Language rules
   - Domain_name has to be unique in the schema.

DEFINES statement
a) The DEFINES statement specifies the names of all attributes defined over that domain.
b) Format
   DEFINES attr_1,attr_2, ... ,attr_n
c) Language rules
   - An arbitrary number of DEFINES statements can appear in the domain entry.
   - Attributes declared in this statement must be defined in the separate attribute entries.

MODE statement
a) The MODE statement defines the data types in the domain.
b) Format

```
        ( CHARACTER int_1 [COMPRESSED proc_name]                  )
        (                                                         )
        (        ( FIXED_POINT    integer_2,int_3 )               )
MODE    ( REAL   (                                )               )
        (        ( FLOATING_POINT [EXTENDED]     )               )
        (                                                         )
        (           ( BINARY [EXTENDED] )                         )
        ( INTEGER   (                   )                         )
        (           ( DECIMAL integer_4 )                         )
```

c) Language rules
   - Only one MODE statement can appear in domain entry.
   - Integer data in the database may be represented in the binary or decimal form; the binary integer data can have the standard and extended format. If the decimal base is specified, it is necessary to give the number of decimal digits (integer_4). Default format for integer data is the binary representation in the standard format.
   - Real data can have the fixed or floating point representation in the database. Real data in floating point format is always represented in binary form with standard or extended precision. Real data in fixed point format is always represented in packed decimal form; integer_2 is the total number of decimal digits and integer_3 is the number of digits in the fractional part. The default format for real data is the binary floating point format in standard precision.
   - Character data can be represented in compressed or source form. Integer_1 specifies the number of characters. If COMPRESSED option is declared, the name of the procedure (proc_name) which performs the compression must be declared too.

UNIT statement
a) The UNIT statement defines the input unit for data in the domain. Because the domain values stored in the database can be expressed in different units, the name of the procedure which performs the conversion from input to internal units may be also specified in this statement.
b) Format
   UNIT unit [,procedure_name]
c) Language rules
   - If the procedure_name is not specified, the data in the database are measured by the same units as input data. If the procedure_name is specified, the conversion from input units to internal units must be done. For example, the input unit can be dollar but the internal unit can be million dollars, etc.
   - If this statement is not given in the domain entry, the domain values are numbers or character strings, without a specific context.

ENDDOMAIN statement
a) This statement denotes the end of the domain description.
b) Format
   ENDDOMAIN

ATTRIBUTE ENTRY

1. The attribute entry uniquely identifies the attribute by its name. Besides, the domain name over which the attribute is defined, the names of relations in which the attribute appears and the total number of different values the attribute may assume are given in the attribute entry.
2. Entry format
   ATTRIBUTE statement
   ORIGIN statement
   BELONGS statement
   [DESC statement]
   [CARDINALITY statement]
   [VALUE statement]
   ENDATTRIBUTE statement
3. Statements description
   The attribute entry begins with the ATTRIBUTE statement and ends with the ENDATTRIBUTE statement. The order of other statements is irrelevant.

ATTRIBUTE statement
a) The ATTRIBUTE statement uniquely identifies the attribute by its name.
b) Format
   ATTRIBUTE attribute_name
c) Language rules
   - Attribute_name has to be unique in the schema
   - Attribute_name must appear in DEFINES statement in the entry of the domain whose name is declared in the ORIGIN statement in this attribute entry.

- Attribute_name must appear in the CONTAINS statement in the entry of the relation whose name is declared in the BELONGS statement in this attribute entry. An attribute may appear in an arbitrary number of relations.

ORIGIN statement
a) The ORIGIN statement identifies the domain over which this attribute is defined.
b) Format
ORIGIN domain_name
c) Language rules
- Domain whose name appears in this statement must be declared in the separate domain entry.

BELONGS statement
a) This statement identifies the relations in which this
attribute appears.
b) Format
BELONGS relation_1, relation_2, ...
c) Language rules
- An arbitrary number of the BELONGS statements may appear in the attribute entry.
- Relations whose names appear in this statement must be declared in the separate relation entries.

CARDINALITY statement
a) The CARDINALITY statement specifies the total number of different attribute values. This statement specifies the total number of n-tuples in the relation, if the attribute is the primary key of the relation.
b) Format
CARDINALITY integer
c) Language rules
- Only one CARDINALITY statement may appear in the attribute entry.
- The number of different attribute values is unlimited if this statement does not appear in the attribute entry.

VALUE statement
a) This statement defines an implicit value that should be assigned to the attribute if the attribute value is not assigned during the loading of the database.
b) Format
VALUE constant
c) Language rules
- Only one VALUE statement may appear in the attribute entry.
- The constant can be of the type real, integer or character depending on the attribute type.
- If this statement is omitted, the attribute values must be assigned during the database loading.

ENDATTRIBUTE statement
a) This statement declares the end of the attribute entry.
b) Format
ENDATTRIBUTE

RELATION ENTRY

1. The relation entry uniquely identifies the relation by its name. Besides, the attributes belonging to that relation, the integrity constraints, the primary key, the cripto protection method and dependencies among this relation and other relations are specified in the relation entry.

2. Entry format
RELATION statement
CONTAINS statement
KEY statement
[DESC statement]
[CRIPTO_PROTECTION statement]
[INTEGRITY_CONSTRAINT statement]
[UNIQUE statement]
[FUNCTION statement]
[ACCESS_CONTROL statement]
[TRIGGER structure]
ENDRELATION
3. Statements description
The order of the statements in the relation entry is irrelevant.

RELATION statement
a) The RELATION statement uniquely identifies schema by its name.
b) Format
RELATION relation_name
c) Language rules
- Relation_name has to be unique in the schema.
- Relation_name must be declared in the BELONGS statement of all attributes that belongs to the relation.

CONTAINS statement
a) The CONTAINS statement specifies the names of all attributes in the relation.
b) Format
CONTAINS attribute_1, attribute_2, ...
c) Language rules
- At least one attribute name must be declared in this statement.
- An arbitrary number of the CONTAINS statements can appear in the relation entry.
- The attributes declared in this statement must be defined in the separate attributes entries.

KEY statement
a) The KEY statement defines the primary key of the relation.
b) Format
KEY attribute_1,attribute_2, ...
c) Language rules
- Only one KEY statement can appear in the relation entry.
- At least one attribute name must be declared in the KEY statement.
- The attributes declared in this statement must be defined in the separate attribute entries.
- The attributes declared in this statement must belong to this relation, i.e. they must be defined in the CONTAINS statement of this relation entry.

INTEGRITY_CONSTRAINT statement
a) This statement defines the integrity constraints in the relations. The constraints can be static and dynamic.
b) Format

$$\text{INTEGRITY\_CONSTRAINT} \left\{ \begin{matrix} \text{r\_exp\_1} \\ \text{fun\_1} \end{matrix} \right\} \; [\text{IF} \left\{ \begin{matrix} \text{r\_exp\_2} \\ \text{fun\_2} \end{matrix} \right\}]$$

c) Language rules
- An arbitrary number of the this statements can appear in the relation entry;
- Reserved words OLD and NEW can appear in the relational_exp_1 but not in the relational_exp_2;
- An arbitrary number of the this statements can be defined for one attribute because the attribute can have more than one integrity constraint;
- Static and dynamic integrity constraints for one attribute must be defined in different INTEGRITY_CONSTRAINT statements

- Integrity constraint is defined by relational_exp_1 or by function_1. Function is defined in the FUNCTION statement;
- Integrity constraint represents the comparison between old (OLD) and new (NEW) attribute values, if the integrity constraint is dynamic. In that case, attribute names must include reserved words OLD and NEW. If the integrity constraint is static, the attribute names in the relational expressions represents immediate values and reserved words OLD and NEW are not included in the attributes names.
- IF option specifies the attribute values on which the integrity constraint is applied. If this option is omitted, the integrity is valid for all attribute values.

Examples:

```
INTEGRITY_CONSTRAINT SUPPLY.CODEPRO =
        PRODUCT.CODE
INTEGRITY_CONSTRAINT NEW  EMPLOYEE.SAL > OLD
                     EMPLOYEE.SAL
INTEGRITY_CONSTRAINT FUN3 IF
   EMPLOYEE.DEPTCODE="B"
INTEGRITY_CONSTRAINT FUN2 IF EMPLOYEE.SEX="M"
FUNCTION FUN2:=EMPLOYEE.SAL > 5000
FUNCTION FUN3:=NEW EMPLOYEE.SAL>OLD EMP.SAL
```

The integrity constraint in the first example specifies that all values of the attribute CODEPRO in the relation SUPPLY must be equal to the values of the attribute CODE in relation PRODUCT. Second example specifies that new salaries of all employees must be greater than old salaries. The usage of the IF option is shown in the third example. This example specifies that new salary must be greater than old salary, but only for employees in department "B". The fourth example defines integrity constraint that the male employees have salary greater than 5000. In the third and fourth example functions FUN2 and FUN3 are used. These functions are defined by the FUNCTION statements.

UNIQUE statement

a) The UNIQUE statement identifies attribute or attributes group having unique values in the relation. That attribute or attributes group do not represent the primary key.
b) Format
   UNIQUE attribute_1, attribute_2, ...
c) Language rules
   - An arbitrary number of the UNIQUE statements can appear in the relation entry.
   - The attributes declared in this statement must be defined in the separate attributes entries;
   - The attributes declared must belong to this relation, i.e. they must be declared in the CONTAINS statement of the relation entry.
   - If more than one attribute is declared in the UNIQUE statement, that attribute group has unique values, not the particular attributes in that group.
   - If more than one attribute or attributes group in the relation have unique values, that should be specified by separate UNIQUE statements;
   - If UNIQUE statement doesn't appear in the relation entry, only attributes that constitute primary key have unique values.

FUNCTION statement

a) The FUNCTION statement defines a function.
b) Format
   FUNCTION function_name:=relational_exp
c) Language rules
   - Function_name must be unique in the schema;
   - The other function names can appear in the relational expression.

ACCESS_CONSTRAINT statement

a) The ACCESS_CONSTRAINT statement specifies the operations that cannot be performed on all or particular n-tuples, i.e. on all or particular attribute values.
b) Format

$$\text{ACCESS\_C FOR } \begin{Bmatrix}\text{READ}\\ \text{UPDATE}\\ \text{INSERT}\\ \text{DELETE}\end{Bmatrix} \text{ [ON attr\_1] [IF } \begin{Bmatrix}\text{r\_exp}\\ \\ \text{funct}\end{Bmatrix}\text{]}$$

c) Language rules
   - An arbitrary number of this statements can appear in the relation entry.
   - One ACCESS_CONSTRAINT statement must be specified for each forbidden operation;
   - The operations INSERT and DELETE denote the insertion and deletion of n-tuples. They must be applied to n-tuples, not to the attribute values. The READ operation denotes the reading of n-tuples or attribute values, and UPDATE operation denotes the updating of one or more attributes in the relation. The READ and UPDATE operations can be applied to n-tuples and attribute values. If ACCESS_CONSTRAINT specifies that UPDATE or READ are forbidden on some attributes, the ON option specifies these attributes. If ON option is omitted, UPDATE or READ are forbidden for all attributes of the relation. The ON option can appear in this statement only for READ and UPDATE operations.
   - IF option specifies n-tuples or attribute values for which the given operation is forbidden. If this option is omitted the requested operation is forbidden for all n-tuples or for all attribute values;
   - If the ACCESS_CONSTRAINT statement is not included in the relation entry description all operations over n-tuples or attributes of the relation are allowed

Examples:

```
ACCESS_CONSTRAINT FOR DELETE
ACCESS_CONSTRAINT FOR UPDATE ON CODE
ACCESS_CONSTRAINT FOR READ IF CODE="C"
ACCESS_CONSTRAINT FOR READ ON SAL IF CODE="C"
```

All examples are defined over the relation EMPLOYEE. First example means that the DELETE operation is forbidden for all n-tuples of relation EMPLOYEE and the second example denotes that the updating of the attribute CODE is not allowed. Third example means that n-tuples in the relation EMPLOYEE having the value of attribute CODE equal "C" cannot be read, and the fourth example means that the values of attribute SALARY of the n-tuples with attribute CODE="C" cannot be read.

TRIGGER structure

a) The TRIGGER structure defines the set of forced operations that must be performed over other relations upon the finishing of the current operation.
b) Format

$$\text{TRIGGER FOR } \begin{Bmatrix}\text{INSERT}\\ \text{UPDATE}\\ \text{DELETE}\end{Bmatrix} \text{ [ON attribute\_1]}$$

$$\begin{Bmatrix}\text{INSERT}\\ \text{UPDATE}\\ \text{DELETE}\end{Bmatrix} \text{ rel\_1 [SET atr\_2:= } \begin{Bmatrix}\text{const}\\ \\ \text{a\_exp}\end{Bmatrix} \text{ [IF } \begin{Bmatrix}\text{r\_exp}\\ \\ \text{fun}\end{Bmatrix}\text{]]}$$

ENDTRIGGER

c) Language rules
   - TRIGGER contains the head line, one or

more lines in which the set of forced operations is defined and the end line;
- An arbitrary number of triggers can appear in the relation entry;
- Relation_1 is the name of the relation to which the forced operation is applied. Attribute_2 is an attribute from that relation;
- Reserved words OLD and NEW can appear in the relational expression of the IF option.
- The FOR option in the head line specifies the operation that causes the trigger;
- If the operation declared in head line is UPDATE, then the ON option must exist. The ON option specifies the attribute in the current relation over which the UPDATE operation (causing the trigger) is performed. If the operation declared in the head line is INSERT or DELETE, the ON option must not appear in the head line because these operations are performed over the n-tuples not over the attribute values;
- The SET option exists only if the forced operation declared is UPDATE. One SET option exists in the same trigger for each attribute updated. For example, if the trigger causes the updating of two attributes (attr_2 and attr_3) in relation_1, then the trigger must contain the following lines:

UPDATE rel_1 SET attr_2:= {constant | arithmetic_exp}

UPDATE rel_1 SET attr_3:= {constant | arithmetic_exp}

The arithmetic expression can be defined in the SET option only if the attr_2 and attr_3 are REAL or INTEGER.
- The IF option specifies the n-tuples, i.e. attribute values in relation_1 over which the operations defined by the trigger are performed. If this option is omitted, the forced operations are applied to all n-tuples in relation_1, i.e. to all attribute values;
- The n-tuples, i.e. attribute values to which the forced operations must be applied are determined by relational expression or by the function whose name appears in the IF option.

Examples:

```
TRIGGER FOR UPDATE ON CODE
UPDATE SUPPLY SET CODEPRO:=PRODUCT.CODE
ENDTRIGGER
TRIGGER FOR UPDATE ON DEPTCODE
UPDATE DEPT SET EMPNO:=DEPT.EMPNO+1 IF FUN1
UPDATE DEPT SET EMPNO:=DEPT.EMPNO-1 IF FUN2
ENDTRIGGER
FUNCTION FUN1:=DEPT.CODE=NEW EMPLOYEE.DEPTCODE
FUNCTION FUN2:=DEPT.CODE=OLD EMPLOYEE.DEPTCODE
```

First trigger updates the relevant CODEPRO entries in the relation SUPPLY whenever the CODE of the PRODUCT is updated. This trigger is defined in the relation PRODUCT. Second trigger updates the relevant EMPNO entries in the relation DEPT whenever the DEPTCODE of the EMPLOYEE is updated. This trigger is defined in the relation EMPLOYEE.

ENDRELATION statement

a) This statement denotes the end of the relation entry.

b) Format

```
ENDRELATION
```

The key words of SDL are given in APPENDIX A, the formal syntax description in APPENDIX B and the schema description for the example in APPENDIX C.

THE INTERPRETER REALIZATION

SDL commands

Bearing in mind that SDL is an interactive language it contains a set of commands which enables the user to list the program, run it, modify, etc. SDL commands are: LIST, INSERT, DELETE, SUBSTITUTE, RUN, PURGE, SAVE, OLD, TERMINATE, PROMPT, TEST and EXIT. The majority of these commands are the standard commands that exist in each interactive language and they have a usual meaning. For example, RUN command runs the SDL program, the OLD command brings an existing SDL program from the file to the user area in the main memory, the PURGE command deletes all program versions but the last one etc. Only TEST and TERMINATE commands have specific role in SDL. The TEST command enables the user to control and track the interpretation process, and the TERMINATE command requests from the interpreter to finish the program interpretation after its modification is done.

Interpreter structure

The interpretation is performed in two phases. In the first phase, the source program is analyzed (statement by statement) and translated into an internal form (postfix notation). The internal form of the program is interpreted and the result is generated in the second phase. Two requests are imposed to the implementation of the SDL interpreter: a) the interpreter must be incremental, b) the multi user environment must be provided.

As a result of the interpretation process, the description file which contains the description of all elements of the database and their internal dependencies is created. Each record in that file corresponds to one element of the database (schema, domain, attribute and relation).

The interpreter is implemented as a set of six internally connected program modules, Fig. 2. The interpreter modules are: command module, lexical analyzer, parser, semantic analyzer, result generator and editor. The control communications between modules are presented in full lines, and the data flow in dashed lines.

The data structures (static and dynamic) that are used during the interpretation process are: source program, internal program, dictionaries and description file. The dictionaries contain all information about the objects of the source program (operators and operands). These information are organized so they can be accessed from any part of the interpreter. There exist two static dictionaries which are parts of the interpreter (dictionary of reserved words and dictionary of delimiters) and one dynamic dictionary (symbol table). The static dictionaries are one dimensional arrays with the elements of fixed length. The dictionary of delimiters contains all arithmetic operators, comparison operators and special characters as comma, colon etc.

The symbol table is formed during the translation of the source program. All information about constants and variables from a SDL program are stored into it. The symbol table has the fixed and variable part. The fixed part contains five fields: identifier name, object type indicator (R - relation, A - attribute, D - domain, etc.), special indicator used for semantic control (if the given object is completely defined this indicator has the value one, otherwise null) and the pointer to the variable part of the symbol table. The fields of the variable part depend on the database element to whom the

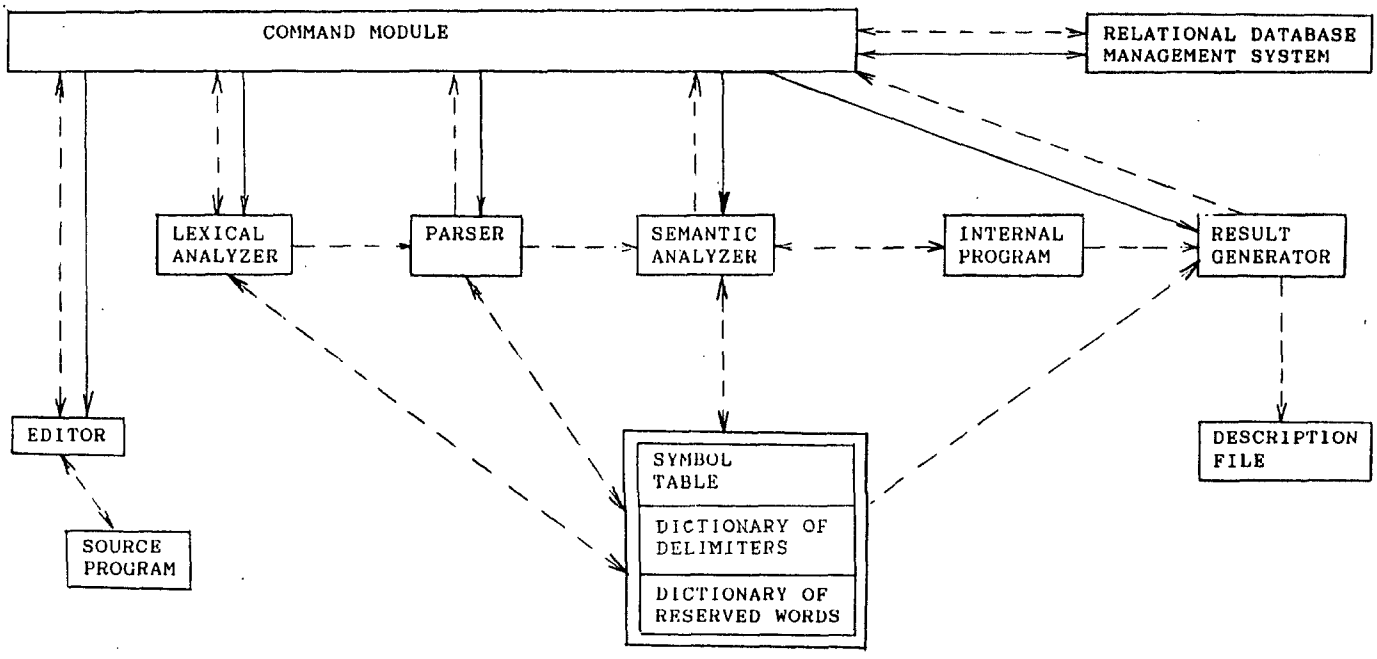


Fig. 2. The interpreter structure

given entry belongs. The number of fields in the variable part and their format for different element types (relations, domains etc.) are different, because the quantity of information that must be stored is different.
All errors detected during the program translation can be classified into five groups: lexical, syntax, semantic, editor and command errors. These errors are detected by the corresponding modules. Upon the detection of any error the interpretation process is interrupted, the command module takes over the control and sends the error message to the user. Due to the concept of the incremental interpretation, each error can be immediately corrected by the user , and the interpretation process continued.
Some diagnostic facilities are built in to enable the tracking of the interpretation process. Each phase of the interpretation process can be easily tracked independently of other phases by printing the results of that phase. For example, the lexical analysis can be tracked by printing the set of tokena which was generated during the lexical analysis. A special SDL command (TEST) specifies the interpretation phase that the user wants to track.
The communication of the SDL interpreter with other parts of the RDBMS is provided through virtual calls. In this way, the interpreter is made self-contained. In the RDBMS environment, the virtual calls must be replaced by the actual ones.

Functional analysis of interpreter parts
The lexical analyzer, parser, semantic analyzer and editor are implemented using the well known methods, because they are the standard parts of all interpreters [2,8]. The other modules are specific because their functions are dictated by the SDL characteristics.
The command module is the main interpreter module which coordinates the work of other modules and communicates with other parts of the RDBMS. This module starts and terminates the interpretation process; it accepts the statements and commands and sends messages to the user, activates other modules to do their job, performs the memory management and provides the multi-user environment.
The command module accepts the statements and commands either from terminal and or file. If the user entered a command, the command module performs the appropriate action. For example if RUN command was given, the command module activates the execution of a SDL program (in fact, activates the result generator).
If the user entered a statement, the lexical analyzer, parser and semantic analyzer are activated to perform the analysis and translation of the source statement into the internal form. If any of these modules detects an error, it informs the command module by setting the error indicator. In that case, the command module interrupts the interpretation process and sends the message to the user.
The lexical analyzer extracts tokens, forms the symbol table (fixed part) and writes into it all available information about identificators. The parser and semantic analyzer fill in the rest of the symbol table. The tokens are divided into four classes: identifiers (variable names), constants, reserved words and delimiters.
The parser performs syntax analysis of the source statements and their translation into the internal form. The internal form of the statements is postfix notation. The parser is implemented by recursive descent method [2].
The semantic analyzer performs the semantic analysis and writes corresponding information into the symbol table and internal program. The semantic analysis includes the control of the whole program and particular statements. The control of semantic correctness of the whole program includes the checking the presence of all prerequisite statements, checking the order in which statements appear and checking the uniqueness of the statements that can appear only once in the program, in the entry or in the some block structure. The consistency of the operand and operator types and the uniqueness of the identifiers (that must be unique) are also checked.
The implementation of the semantic analyzer, especially the part which controls the program form, is based on the theory of the finite

state machine. The semantic control of the single statements is done by the control routines that are called at particular places in the program structure which simulates that finite state machine [8].
The result generator, which is activated by the RUN command, creates the description file from the symbol table and internal program. The description file contains the descriptions of all database elements (schema, domains, attributes and relations). Due to the fact that SDL has no imperative statements, the result is generated by searching the symbol table and internal program and gathering all necessary information for the description file. The records in the description file have four different formats, and each record format corresponds to one data structure in the database (schema, domain, attribute and relation). The records of the same format are grouped together, so the physical description file contains four logical files. The result generator is implemented as a set of procedures and each procedure forms a description of one data type.
The editor is the module whose task is to provide the editing of the source program and support the incremental interpretation of the SDL program.
The multiuser SDL interpreter is implemented on the IBM 4331 computer using COBOL programming language. The system software for interactive work CICS [13] (which enables dynamic memory and process management, data interchange with disks and communications with the users over terminals) is used for implementation. The description file is formed as IBM VSAM KSDS data file with variable length records [12]. The interpreter volume expressed by the number of program lines is 5500.

CONCLUSION

The schema description language for relational database is a stand alone language completely independent from other languages in the DBMS (query language, data manipulation language etc.). Due to the fact that the relational database description have four different data types (schema , attribute, domain and relation), the SDL statements are grouped into four entries, so each entry describes one data type. The SDL is a structured language which provides a high readability of the SDL programs.
The interpreter developed for this language is characterized by its functional independence from the query language, the multiusesr environment and incremental interpretation.
The implementation of the interpreter provides the conditions for its easier portability to different machines. These conditions are: a) the application of standard programming language characteristics without any extensions, b) concentration of input/output activities in command module c) marking all machine dependent points in the interpreter (for example, calls of assembler routines, system service calls etc.) so they become immediately visible.

REFERENCES

1. Astrahan M. M. : "SYSTEM R: A relational database management system"; Computer, may 1979.

2. Brown P. J.: "Writing Interactive Compilers and Interpreters"; John Wiley and Sons.

3. Chamberlain D. D. etc: "A history and evolution of SYSTEM R"; Communications of ACM, no 10, october 1981.

4. Codd E.F.:"A relational model of data for large shared data banks"; Communications of the ACM, no 6, june 1970.

5. Codd E.F.:"Further Normalization of the Database Relational Model";Current Computer Science Simposia, Vol 6, Database Systems, New York city, May 1971, Prentice Hall.

6. Codd E.F.:"Normalized Database Structure: A Brief Tutorial"; Current Computer Science Simposia, Vol 6, Database Systems, New York city, May 1971, Prentice Hall.

7. Codd E.F.:"Relational Completeness of Database Sublanguages"; Current Computer Science Simposia, Vol 6, Database Systems, New York city, May 1971, Prentice Hall.

8. Gries D.: "Compiler Design for Digital Computers"; John Wiley and Sons.

9. Hutt A. T. F.:"Organizing the Description of a Relational Database";Software - Practice and Experience, vol k9, 1979.

10. Hutt A.T.F.: "A Relational Data Base Management System"; John Wiley and Sons, 1979.

11. Gray J., McJones P.:"The recovery manager of SYSTEM R database"; ACM Computing Surveys, no 2, june 1982.

12. VSE (Virtual Storage Extended) System Data Management Concepts - IBM Laboratory, Programming Publication Department, Boebling, W. Germany.

13. CICS/VS (Customer Information Control System/Virtual Storage) Introduction to Program Logic - IBM Laboratory, Technical Documentation Department, Hampshire, England.

APPENDIX A: Reserved words

ACCESS_CONSTRAINT	FUNCTION
AND	IF
ATTRIBUTE	INSERT
BELONGS	INTEGER
BINARY	INTEGRITY_CONSTRAINT
CARDINALITY	KEY
CHARACTER	MODE
COMPRESSED	NEW
CONTAINS	OLD
CRIPTO_PROTECTION	ON
DECIMAL	OR
DEFINES	ORIGIN
DELETE	READ
DESC	REAL
DOMAIN	RELATION
ENDATTRIBUTE	SCHEMA
ENDDOMAIN	SET
ENDRELATION	SYSTEM
ENDSCHEMA	TRIGGER
ENDTRIGGER	UNIT
EXTENDED	UNIQUE
FIXED_POINT	UPDATE
FLOATING_POINT	VALUE
FOR	

APPENDIX B: SDL syntax

Here we present the BNF syntax definition for SDL. In this notation square brackets [] indicate optional constructs, and braces {} indicate constructs that appear one or more times.

```
<SDL_program>::=<schema_entry>
{<domain_entry>}
                {<attribute_entry>}
                {<relation_entry>}

<schema_entry>}::=<schema_stat>
                [<description_stat>]
                [<cripto_protection_stat>]
                <endschema_stat>
<schema_stat>::=SCHEMA <schema_name>
<schema_name>::=<name>
<name>::=<letter>{<hyphen><alphanumeric>
        | <alphanumeric>}
<alphanumeric>::=<letter> | <digit>
<letter>::= A | B | C | D | ... | Z
<digit>::= 0 | 1 | 2 ... | 9
<hyphen>::= _
<description_stat>::=DESC <character_string>
<character_string>::={<character>}
<character>::=<alphanumeric>
             | <special_character>
<special_character>::= <addition_op>
                      | <multiplication_op>
                      | <comparison_op>
      | " | , | . | : | $ | ( | ) | <hyphen>
<addition_op>::= + | -
<multiplication_op>::= * | /
<comparison_op>::= < | > | = | => | <= | <>
<cripto_pr_stat>::=CRIPTO_PROTECTION <method>
<method>::=<procedure_name>
          | SYSTEM
<endschema_stat>::=ENDSCHEMA

<domain_entry>::=<domain_stat>
                 <defines_stat>
                 <mode_stat>
                 [<description_stat>]
                 [<mode_stat>]
                 <enddomain_stat>
<domain_stat>::=DOMAIN <domain_name>
<domain_name>::=<name>
<defines_stat>::=DEFINES {<attribute_name>}
<mode_stat>::=MODE <domain_type>
<domain_type>::=<character_type>
               | <integer_type>
               | <real_type>
<character_type>::=CHARACTER <integer_number>
           [COMPRESSED <procedure_name>]
<real_type>::=FIXED_POINT <integer_number>,
                          <integer_number>
             | FLOATING_POINT [EXTENDED]
<integer_type>::=INTEGER <number_type>
<number_type>::=DECIMAL <integer_number>
               | BINARY [EXTENDED]
<unit_stat>::=UNIT <unit> [,<procedure_name>]
<unit>::=<character_string>
<enddomain_stat>::=ENDDOMAIN

<attribute_entry>::=<attribute_stat>
                    [description_stat>]
                    [<value_stat>]
                    <origin_stat>
                    <belongs_stat>
                    [<cardinality_stat>]
                    <endattribute_stat>
<attribute_stat>::=ATTRIBUTE <attribute_name>
<attribute_name>::=<name>
<origin_stat>::=ORIGIN <domain_name>
<belongs_stat>::=BELONGS  {<relation_name>}
<cardinality_stat>::=CARDINALITY
                           <integer_number>
<integer_number>::={<digit>}
<value_stat>::=VALUE <constant>
<constant>::=<numeric_constant> | <literal>
<numeric_constant>::=[<sign>] <integer_number>
                    | [<sign>] <real_number>
<sign>::= + | -
<real_number>::= <integer_number>.
                 <integer_number><exp>
<exp>::=D[<sign>]<integer_number>
        |E[<sign>]<integer_number>
<literal>::="<character_string>"
<endattribute_stat>::=ENDATTRIBUTE

<relation_entry>::=<relation_stat>
                   [<description_stat>]
                   [<cripto_protection_stat>]
                   <key_stat>
                   <contains_stat>
                   [<integrity_stat>]
                   [<unique_stat>]
                   [<function_stat>]
                   [<access_constraint_stat>]
                   [<trigger_block>]
                   <endrelation_stat>
<relation_stat>::=RELATION <relation_name>
<relation_name>::=<name>
<key_stat>::=KEY {<attribute_name>}
<contains_stat>::=CONTAINS {<attribute_name>}
<integrity_stat>::=INTEGRITY_CONSTRAINT
            <constraint> [IF <condition>]
<constraint>::=<relational_expression>
             | <function_name>
<condition>::=<relational_expression>
             | <function_name>
<relational_expression>::=<simple_exp>
    | <simple_exp><log_op><function_name>
    | <simple_exp><log_op>(<relational_exp>)
<log_op>::= AND | OR
<simple_exp>::=<variable>
              <comparison_op><operand>
<operand>::=<variable> | <constant>
<variable>::=<prefix><e_variable>
<prefix>::=OLD | NEW
<e_variable>::=[<relation_name>].
              <attribute_name>
<function_name>::=<name>
<unique_stat>::=UNIQUE {<attribute_name>}
<function_stat>::=FUNCTION
<function_name>:=<logical_exp>
<access_constraint_stat>::=ACCESS_CONSTRAINT
         FOR <operation_1>
         [ON <attrname>] [IF <condition>]
<operation_1>::= READ | <operation>
<operation>::= INSERT | UPDATE | DELETE
<trigger_block>::=<head_line> <trigger_body>
                  <end_line>
<head_line>::=TRIGGER FOR <operation>
            [ON <attribute_name>]
<trigger_body>::=<operation> <relation_name>
     [SET <attribute_name>:=<expression>
     [IF <condition>]]
<expression>::=<constant> | <arithmetic_exp>
<arithmetic_exp>::=<sign><operand>
            | <sign><operand><arit_operator>
            (<arithmetic_exp>)
<sign>::= + | -
<operand>::=<variable> | <numeric_constants>
<arit_operator>::= + | - | * | /
<end_line>::=ENDTRIGGER
```

APPENDIX C: SDL program

```
SCHEMA MARKETING
 DESC There is no protection at schema level
DOMAIN DOMADDRESS
 DEFINES ADDRESS
  MODE CHARACTER COMPRESSED PROC1
   ENDDOMAIN
DOMAIN DOMCODE
 DEFINES CODE,CODEPRO,CODESUP,DEPTCODE
  MODE INTEGER BINARY
   ENDDOMAIN
DOMAIN DOMDATE
 DEFINES DATE
  MODE CHARACTER
   ENDDOMAIN
DOMAIN DOMNO
 DEFINES EMPNO
```

```
 DEFINES EMPNO
  MODE INTEGER DECIMAL
   ENDDOMAIN
DOMAIN DOMPRICE
 DEFINES PRICE
  MODE REAL FIXED_POINT
   UNIT DOLLAR
    ENDDDOMAIN
DOMAIN DOMQUANTITY
 DEFINES QUANTITY
  MODE REAL FIXED_POINT
   ENDDOMAIN
DOMAIN DOMNAME
 DEFINES NAME
  MODE CHARACTER COMPRIMED PROC2
   ENDDOMAIN
DOMAIN DOMSALARY
 DEFINES SALARY
  MODE REAL FLOATING_POINT
   UNIT DOLLAR
    ENDDOMAIN
DOMAIN DOMSEX
 DEFINES SEX
  MODE CHARACTER
   ENDDOMAIN
ATTRIBUTE ADDRESS
 ORIGIN DOMADDRESS
  BELONGS DEPT
   ENDATTRIBUTE
ATTRIBUTE CODE
 ORIGIN DOMCODE
  BELONGS SUPPLY,PRODUCT,EMPLOYEE,DEPT
   ENDATTRIBUTE
ATTRIBUTE CODEPRO
 ORIGIN DOMCODE
  BELONGS SUPPLY
   CARDINALITY 5000
    ENDATTRIBUTE
ATTRIBUTE CODESUP
 ORIGIN DOMCODE
  BELONGS SUPPLY
   CARDINALITY 100
    ENDATTRIBUTE
ATTRIBUTE DEPTCODE
 ORIGIN DOMCODE
  BELONGS EMPLOYEE
   ENDATTRIBUTE
ATTRIBUTE DATE
 BELONGS SUPPLY
  ORIGIN DOMDATE
   VALUE "01/01/1980"
    DESC Data when the product was bought
     ENDATTRIBUTE
ATTRIBUTE EMPNO
 ORIGIN DOMNO
  BELONGS DEPT
   DESC Number of employees in the department
    VALUE 0
     ENDATTRIBUTE
ATTRIBUTE QUANT
 ORIGIN DOMQUANTITY
  BELONGS PRODUCT,SUPPLY
   VALUE 0
    ENDATTRIBUTE
ATTRIBUTE NAME
 ORIGIN DOMNAME
  BELONGS PRODUCT,EMPLOYEE,DEPT
   VALUE "XXXX"
    ENDATTRIBUTE
ATTRIBUTE PRICE
 ORIGIN DOMPRICE
  BELONGS PRODUCT
   VALUE 100
    ENDATTRIBUTE
ATTRIBUTE SALARY
 ORIGIN DOMSALARY
  BELONGS EMPLOYEE
   VALUE 1000
    ENDATTRIBUTE
ATTRIBUTE SEX
 ORIGIN DOMSEX
  BELONGS EMPLOYEE
   CARDINALITY 2
    VALUE "M"
     ENDATTRIBUTE
RELATION PRODUCT
 CONTAINS CODE,NAME,PRICE,QUANT
  KEY CODE
   DESC Products in the supply
    INTEGRITY_CONSTRAINT CODE>0 AND CODE<32767
    INTEGRITY_CONSTRAINT PRICE<999.99
     TRIGGER FOR UPDATE ON QUANT
     UPDATE SUPPLY SET QUANT:=SUPPLY.QUANT+NEW
PRODUCT.QUANT
     ENDTRIGGER
    TRIGGER FOR UPDATE ON CODE
    UPDATE     SUPPLY     SET     CODEPRO:=NEW
PRODUCT.CODE
    ENDTRIGGER
ENDRELATION
RELATION EMPLOYEE
 CONTAINS CODE,NAME SEX,SALARY,DEPTCODE
  KEY CODE
   DESC Company employees in the last 5 years
   UNIQUE DEPTCODE
    TRIGGER FOR UPDATE ON DEPTCODE
    UPDATE DEPT SET EMPNO:=EMPNO+1 IF FUN1
    UPDATE DEPT SET EMPNO:=EMPNO-1 IF FUN2
    ENDTRIGGER
   FUNCTION FUN1:=DEPT.CODE=NEW DEPTCODE
   FUNCTION FUN2:=DEPT.CODE=OLD DEPTCODE
  INTEGRITY_CONSTRAINT SEX="M" OR SEX="F"
  INTEGRITY_CONSTRAINT CODE>1 AND CODE<32767
  INTEGRITY_CONSTRAINT DEPTCODE=DEPT.CODE
 ACCESS_CONSTRAINT  FOR   READ  ON  SALARY  IF
CODE<>1
ENDRELATION
RELATION SUPPLY
 CONTAINS CODEPRO,CODESUP,DATE,QUANT
  KEY CODEPRO,CODESUP
   INTEGRITY_CONSTRAINT CODEPRO=PRODUCT.CODE
  INTEGRITY_CONSTRAINT DATE>"01/01/1980"
ENDRELATION
RELATION DEPT
 CONTAINS CODE,NAME,ADDRESS,EMPNO
  KEY CODE
   INTEGRITY_CONSTRAINT CODE>1 AND CODE<10
  UNIQUE NAME
ENDRELATION
ENDSCHEMA
```

IZBOR PROGRAMSKEGA ORODJA ČETRTE GENERACIJE

INFORMATICA 1/89

Deskriptorji: REACIJSKE BAZE, IZBOR SISTEMA, PROGRAMSKA ORODJA, LUKA KOPER, ČETRTA GENERACIJA

Zvone Kribel, Boris Legac, Marjan Marušič, Andrej Novak
Luka Koper, Sektor za informacijski sistem

Podajamo kriterije, s katerimi je Luka Koper, oz. komisija za izbor nove programske opreme izbirala med različnimi ponudniki. Komisija je sestavila do podrobnosti specificirano vzorčno aplikacijo, ki jo sorazmeroma lahko preslikamo v CICS-ov program. Ob preverjanju lastnosti opreme, smo prišli do zaključka, da izbor programske opreme ni tehnično, temveč multidisciplinarno vprašanje.

SELECTING THE FOURTH GENERATION PROGRAMMING TOOLS: In this paper we describe the criterions that were used in Port of Koper for the evaluation of 4GL and DBMS. We describe in detail one case application that was tested with the PL/1 programming language and VSAM under CICS/VS environment. During the evaluation we came to a conclusion that the selection is not only a technological matter, it should be a multidiscipline process.

1. UVOD

V Luki Koper že tri leta koristimo CA-UNIVERSE, relacijsko banko podatkov, s programskim jezikom ADL in aplikacijskim generatorjem. S postavitvijo plana razvoja luškega IS in po triletnih izkušnjah dela s to relacijsko banko smo prišli do ugotovitve, da potrebujemo programsko orodje, ki nam bo omogočala doseganje boljših performans aplikacij v proizvodnji.Dne 9. marca 88 smo v ta namen ustanovili komisijo za izbor nove programske opreme. S 1. oktobrom smo pričeli prenašati produkcijsko zahtevne aplikacije iz CA-Universe v produkte ameriške družbe Applied Data Research.
Glavni zahtevi do nove opreme sta:
- omogoča izgradnjo enotne produkcijsko stabilne podatkovne banke,
- nuditi mora visoko produktivno programsko okolje.

V sestavku podajamo kriterije za izbor opreme glede na potrebe razvoja informacijskega sistema Luke Koper. Poleg tega podajamo razlike med produkti dveh družb, ki sta ostali v ožjem izboru. Opisali smo tudi vire informacij o lastnostih teh produktov. V posebnem razdelku je podrobno definirana aplikacija, ki smo jo posredovali specialistom v obeh družbah. Ob koncu sledi zaključek in naše izkušnje pri izboru nove opreme.

2. RAZLIKE

Po analizi tržišča je prišlo v širši izbor 7 produktov naslednjih proizvajalcev:

- Computer Associate: CA UNIVERSE
- IBM: SQL/DS, CSP
- ADR: DATACOM, DATADICTIONARY, IDEAL
- Software AG: ADABAS, PREDICT, NATURAL
- CINCOM: SUPRA, MANTIS
- ORACLE: ORACLE
- CULLINET: IDMS

Po prvih primerjalnih analizah sta v ožjem izboru ostala le še ADR in SAG iz naslednjih razlogov:

- CA-UNIVERSE: v produkcijskem okolju zahteva zmogljivo strojno opremo, ki je Luka Koper ne more zagotoviti.
- SQL, CSP: ni pravo produkcijsko okolje
- SUPRA,MANTIS: sistem je prekompleksen za naše razpoložljive resurse
- ORACLE: Ni v DOS/CICS okolju, večina instalacij na DEC-VAX
- IDMS: firma ima organizacijske, finančne in tehnične težave.

Po izvršitvi vrednotenja posameznih produktov smo ugotovili, da so razlike med produkti družb Software AG in ADR minimalni, tj. samo 4% možnih točk. Te razlike so v naslednjem:

- podatkovna baza (ADABAS je nekoliko enostavnejša, ima možnost dinamičnega dodajanja atributov, ponavljajoče se skupine) .
- razvojni jezik (NATURAL ima dostop do VSAM

datotek in ima možnost oken; IDEAL je enostavnejši za uporabo in ima boljšo programsko dokumentacijo)
- podatkovni slovar DATADICTIONARY ni možno izključiti pri prevodu programov, v PREDICT pa lahko vpišemo tudi nekatera integracijska pravila.

Obe družbi s svojo ponudbo zagotavljata izpolnitev kriterijev, ki smo jih postavili kot obvezne. ADABAS koristi ameriška FBI, DATACOM/DB pa ameriška vojska, ADABAS ima cca 2200 instalacij. DATACOM/DB ima 1800 instalacij. Družba Software AG ni imela v bližini instalacije s podobnim HW in v podobni gospodarski veji. Produkte družbe ADR smo si ogledali v dveh centrih v Trstu in v Interfrigo v Baslu v švici. Zanimiv je pristop tujcev, ki sploh niso vzeli v precep konkurenčnih produktov. Ob potovanju v Basel smo se pogovorili s predsednikom evropske veje družbe ADR v Klotenu pri Zurichu in istočasno smo obiskali nemško družbo Software AG v Darmstadtu.
V obe družbi smo poslali podrobno definicijo vzorčne aplikacije in zahtevali ocene CICS particij, odzivnega časa in prosili, naj nam razvijejo specificirano aplikacijo v NATURAL-u, oz. IDEAL-u.

3. APLIKACIJA

Aplikacija dodaja posamezen slog v tabelo delavcev. CICS-ovo ime transakcije je ZDDA in program naj bo ZDDOADD, ki je del projekta ZDD. Ta aplikacija mora koristiti psevdokonverzacijski način na 250 CICS terminalih (200 VTAM in 50 lokalnih 3270 ali 3170). Povprečno se doda 2.5 sloga v sekundi, največ pa 5 slogov v sekundi ob obremenitvi. Odzivni čas naj bo največ 2 sekundi, zaželjeno je povrečje ene sekunde v 8 urni izmeni.

a). Osnovno tabelo delavcev specificiramo z ANSI SQL-om z:

```
CREATE TABLE DELAVEC (
SIFDEL    INTEGER NOT NULL,
IME       VARCHAR (20),
PRIIMEK   VARCHAR (20),
DAT_ROJ   CHAR(6)
);
```

Pogled (view) ZDDVDEL vsebuje vsa polja z izjemo DAT_ROJ polja. Polje SIFDEL mora imeti masko PIC'99999' povsod, kjer se pojavi.
Prvi znak polj IME in PRIIMEK naj bo vedno črka.

b). Aplikacija koristi dve BMS mapi ZDDMCTL kontrolni zaslon in ZDDMDTL za detaljni izgled sloga. ZDDMCTL ima samo SIFDEL polje, kamor operator vnese šifro delavca, ki ga želi dodati. Zaslon ZDDMDTL ima IME, PRIIMEK in SIFDEL polja, vendar SIFDEL je v tem zaslonu samo izhodno polje. Oba zaslona imata polje sporočil na spodnjem robu zaslona. Ob vsaki napaki napačno polje osvetlimo. Obe mapi vsebujeta standardne tekste kot so ime organizacije, ime projekta, ime zaslona - mape, datum in uro ob zadnjem pritisku neke AID tipke.

c). Z vnosom ZDDA poženemo program ZDDOADD, prikaže se zaslon ZDDMCTL z LMI001 sporočilom. Vsa sporočila so v standardni tabeli. Ko uporabnik vnese vrednost polja SIFDEL (vse numerično) in ENTER, program preveri PIC'99999' masko. če je napaka, program ponovno pošlje mapo na terminal z osvetljenim poljem SIFDEL in standardnim sporočilom LME001. Program zaključimo z CLEAR AID tipko, s PF1 program prikaže zaslon za pomoč ZDDHA00 (help).
Ko operator uspešno vnese SIFDEL, program preveri obstoj sloga s tem ključem. če tabela že vsebuje tega delavca, program prikaže LME020 sporočilo. če tega delavca še ni v tabeli, program prikaže ZDDMDTL mapo s sporočilom LMI010, kamor uporabnik vnese ime in priimek delavca. Vrednost polja SIFDEL program samo prenese iz predhodnega zaslona. Obe polji program preveri ali imata prvi znak črko, sicer sporočilo LME022. če je vse v redu, program doda slog s ključem SIFDEL v ZDDVDEL. Spet se prikaže zaslon ZDDMCTL s sporočilom LMI002, ki mu sledi sporočilo LMI001 sporočilo (konkatenirano).

d). Upoštevajoč hišni standard mora vsaka aplikacija koristiti standardni modul GETMSG,ki pričakuje vhodni parameter 6 znakov dolgo šifro sporočila in vrne preko globalnega področja (npr. CICS Common Work Area) 39 znakov dolg niz sporočila iz posebne tabele. Ta aplikacija koristi sledeča sporočila:

LMI001	Prosim, vnesite ključ.
LMI002	Vrstica je uspešno dodana v tabelo
LMI010	Prosim, vnesite vrednosti stolpcev.
LME000	Ničesar niste vnesli.
LME001	Ključ mora biti številka.
LME020	Ta vrstica že obstaja v tabeli.
LME022	Prvi znak niza mora biti črka.

4. CENE

Ponujeni finančni pogoji obeh firm so podobni:

Družba	cena	letno vzdrževanje
ADR	263.688 US$	35.477 US$
Teden šolanja	600 US$	
Po petih letih znaša celotna vsota z osnovnim šolanjem (na dveh procesorjih)	461.297 US$	
SAG	201.850 US$	36.700 US$
Za dva CPU	278.438 US$	50.625 US$
Teden šolanja:		
prvo leto	5.000 US$	
drugo leto	7.500 US$	
ostala	11.000 US$	
Po petih letih znaša celotna vsota z osnovnim šolanjem	480.938 US$	

5. ZAKLJUČEK

Nakup nove programske opreme je v našem okolju dokaj zahtevna in nehvaležna naloga. Pogosto imajo glavno besedo pri izboru opreme samo vodilne osebe, ki običajno niso dovolj tehnično usmerjene. Delno se lahko izognemo napačni odločitvi z ustanovitvijo posebne komisije za izbor opreme. V Luki Koper smo izbrali slednje. Hitro se je pokazalo, da je naloga zelo zahtevna in da jo je zelo težko kvalitetno rešiti v zastavljenih šestih mesecih.
Dokaj hitro smo ugotovili utesnjenost in tehnično zaostalost našega okolja. Naštejmo nekatere najbolj pereče težave, ki smo jih srečali pri sodelovanju v komisiji:
- pol leta je nedvomno premalo časa za ocenitev programskih orodij, ki so strateškega pomena za delovno organizacijo.
- Vsak posamezen programski produkt iz razreda sistemov za upravljanje bank podatkov je plod cca 10 do 20 človek/let in tega dela ni možno spoznati v treh tednih.
- V komisiji smo močno pogrešali strokovnjake s finančnimi in pravnimi izkušnjami. Literatura za napotke pri sklepanju pogodb s tujimi softwarskimi hišami ni uporabna v

našem okolju.

6. REFERENCE

1. RELATIONAL DBMSs, Xephon Buyer's Guide, Xephon Technology Transfer Ltd, London, 1986
2. A BUYER'S GUIDE TO DATA BASE MANAGEMENT SYSTEM, Datapro Research Corp., Delran NJ, 1985
3. Diane L. HERDT: DBMS EVALUATION CRITERIA, Auerbach System Development Management portfolio 34-02-40, Auerbach Publishers Inc., Boston MA, 1987
4. Gordon C. EVEREST: DATABASE MANAGEMENT, Objectives, System Functions, and Administration, McGraw Hill 1986

PRILOGA: Kriteriji za izbor opreme z utežmi glede na pomen posameznega kriterija za izgradnjo IS Luke Koper:

1. PRODUKCIJSKO OKOLJE	UTEŽ
1.1. Operacijski sistem DOS pod VM	*
1.2. CICS okolje	*
1.3. Možnost vsaj 250 CICS terminalov (za isto transakcijo) na obstoječem HW	*
1.4. Odzivni čas 2 sekunde (5 transakcij/s)	*
1.5. Sledi spremembam releas-ov IBM prog.	*
1.6. Omogoča spremembo (strojne opreme, DOS -> MVS	*
1.7. Vključitev drugih uporabnikov v Luški IS preko SNA	*
1.8. File transfer na PC	*

2. PODATKOVNA BAZA	
2.1 Enoten podatkovni slovar (ne glede na št. DOS-ov)	10
2.2. Vsebuje orodja za poročila na DD, (crossreference)	*
2.3. Recovery (avtomatski restart)	10
2.4. Monitoring (možnost fizične namestitve podatkov)	10
2.5. Možnost sprememb podatkovnih definicij (ne prizadene obstoječih aplikacij)	5
2.6. Zaščita podatkov (permiti) je na nivoju DB/DD	*
2.7. Zaklene samo slog pri spreminjanju (locking)	10
2.8. Možen je pristop z ANSI SQL jezikom	7
2.9. Tip podatkov datum (operacije +,-, <>,=> in preverjanje pravilnosti datuma	3
2.10. Null vrednost (nedefinirana vrednost)	5
2.11. Transaction control, rollback, commit	*
2.12. Agregatne funkcije (avg,sum,group,by)	5
2.13. Možnost dostopa do banke s PL/I programom	10
2.14. Možnost definiranja VIEW-ov z dostopom tudi do datotek, ki niso v bazi (VSAM)	8
2.15. Omogoča kompresijo podatkov	3
2.16. Formiranje viewov tipa SELECT, JOIN z WHERE pogojem z možnost update na eno tabelo	5
2.17. Podpora nacionalnim znakom, latin 2	10
2.18. Integritete entitet	10
2.19. Integritete domene	5
2.20. Integritete referenčne	3
2.21. Kriptozaščita	3

3. RAZVOJ APLIKACIJ	
3.1. Programski jezik, ki deluje preko slovarja pod.	*
3.2. Generator map in reportov (map za PRINT)	*
3.3. Pomoč pri izdelavi dokumentacije	8
3.4. Interaktivni vpogledi v podatkovni slovar	10
3.5. Generator aplikacij (vodenje preko menijev, help)	7
3.6. Možnost izvajanja batch aplikacij v samostojni particiji	10
3.7. Možnost orodij za pomoč pri metodah analize in designa na PC (CASE Tools)	3
3.8. podpora poslovne grafike na PC	3
3.9. Razvoj aplikacij na PC, ki imajo dostop do centralne podatkovne baze preko file transfer	3
3.10. Popolna zamenjava za klasične programske jezike (vse kar lahko v PL/I)	10
3.11. Relacijski pristop do podatkov izključno preko VIEW-ov	10

4. MIGRACIJA	
4.1. Starih aplikacij ni potrebno prevajati DL/I (DL/I transparenca)	5
4.2. Starih aplikacij ni potrebno prevajati VSAM (VSAM transparenca)	10
4.3. Omogoča postopen prenos v novo okolje	7
4.4. Odzivni časi pri migriranih aplikacijah v DB naj ne padejo več kot za 10%	5
4.5. Utility za prenos job control	5

5. PODPORA PROIZVAJALCA	
5.1. Osnovno šolanje v Luki Koper	10
5.2. Kvalitetno šolanje specialistov (sistemci, DBA)	10
5.3. Literatura (ažurna, original 2 izvoda, številka licence, interni časopis proizvajalca)	10
5.4. Nepretrgana tehnična podpora (24 urni telefon)	10
5.5. UP-DOWN LOAD 5 G BYTOV < 8 urah	7
5.6. Menjava releasov ne sme vplivati na izdelave aplikacije in na banko podatkov	10
5.7. Zagotovljeno svetovanje pri razvoju aplikacij	4

6. NABAVNI POGOJI	
6.1. Cena enkratnega nakupa do 270.000 $ možnost obročnega odplačevanja	*
6.2. Testna inštalacija	*
6.3. Osnovno šolanje vključeno v ceno	6

Vrednost polja UTEŽ pove pomembnost te lastnosti za razvoj IS Luke Koper. Zvezdica v tem stolpcu pomeni, da je označega lastnost obvezna.

Način izračuna točk za posamezne produkte:

$$TPI = \sum_{j=1}^{N} Uj * OPij$$

TPi - točke i - tega produkta
Uj - utež j - tega produkta
N - število kriterijev izbora
OPij - ocena j - tega kriterija za i-ti produkt (v razponu od 0 do 10 točk)

INFORMATIONAL LOGIC III

INFORMATICA 1/89

Descriptors: LOGIC INFORMATIONAL, RULES FORMATIONAL, AXIOMS INFORMATIONAL, INFORMATIONAL WELL-FORMED FORMULA (IWFF)

Anton P. Železnikar
Iskra Delta, Ljubljana

In this part of the essay two main topics of the informational logic (IL) are discussed: formation rules which govern the structure of informationally well-formed formulae (iwffs) and informational axioms. In the continuation of this essay informational transformation rules of IL will be examined in a formal informational way.

Formation rules of IL have to answer the question how to construct initial informational formulae which will belong to the so-called class of iwffs. Within this question the so-called operational, operand, and parenthetic constituents and their compositions into iwffs are determined. In formatting a formula (iwff) several basic processes can be applied, for instance, beginning of formula formation, introducing of operands and operators in implicit and explicit forms into the context of an iwff, particularization and universalization of formulae, etc. Afterwards, formation rules of IL are exposed in a short and concise manner. At the end, the question can be put what could be the form of a non-informational formula.

Within the topic of informational axioms the following subjects are discussed: axiomatization of informational principles, how informational axioms can be generated, axioms of Informing, informational difference, informational circularity, informational spontaneity, informational arising, counter-information, counter-Informing, informational embedding, informational embedding of counter-information, informational differentiation, informational integration, informational particularization and universalization, informational structure and organization, informational parallelism, informational cyclicity, openness of informational axiomatization, influence of metaphysical beliefs on axiomatization, and axiomatic consequences of informational arising.

Informacijska logika III. V tem delu spisa se obravnavata dve glavni naslovni poglavji informacijske logike (IL): formacijska (oblikovalna) pravila, ki urejajo strukturo informacijsko dobro oblikovanih formul (iwff) in informacijski aksiomi. V nadaljevanju tega spisa se bodo na informacijsko formalen način preučevala še informacijska transformacijska pravila IL.

Formacijska pravila IL morajo odgovoriti na vprašanje, katere informacijske formule pripadajo t.i. razredu iwff. V okviru tega vprašanja se opredeljujejo operacijski, operandni in oklepajni konstituenti in njihove kompozicije v iwff. Pri oblikovanju informacijskih formul (iwff) se uporabljajo nekateri osnovni procesi, kot so npr. začenjanje oblikovanja formul, uvajanje operandov in operatorjev v implicitni in eksplicitni obliki v kontekst formul, stikanje operatorjev, partikulariziranje in univerzaliziranje formul itd. Oblikovalna pravila IL je mogoče opisati kratko in jedrnato. Vprašanje, ki ga je mogoče postaviti pri oblikovanju formul je, kakšna bi lahko bila oblika neinformacijske formule.

V okviru problematike informacijskih aksiomov pa se obravnavajo tale naslovna vprašanja: aksiomatizacija t.i. informacijskih principov, kako je mogoče generirati aksiome, aksiomi informiranja, informacijske diference, informacijske cirkularnosti, informacijske spontanosti, nadalje aksiomi informacijskega nastajanja, protiinformacije, protiinformiranja, informacijskega vmeščanja, informacijskega vmeščanja protiinformacije, informacijske diferenciacije, informacijske integracije, informacijske partikularizacije in univerzalizacije, informacijske strukture in organizacije, informacijskega paralelizma, informacijske cikličnosti ter še odprtost informacijske aksiomatizacije in aksiomatične posledice informacijskega nastajanja.

II. 2. FORMATION RULES OF INFORMATIONAL LOGIC

... in rejecting mental representations as the objects of belief one is not thereby rejecting the empirical hypothesis that the brain is an information processor and thus processes in a neural machine language.

Stephen Schiffer [11] 5

II.2.0. Introduction

In this section (II. 2.) we have to say clearly which kind of informational formula will belong to informational logic (IL). Thus, we shall deal with the question how to construct informational formulae which will belong to the legal form of informational formulae. The word legal will have the meaning of well-formed. We have to state precisely what is an expression composed of informational operands, informational operators, and parenthetic delimiters, in such a way that it will represent the so-called informational well-formed formula (iwff).

In the context of formation rules we must consider that an iwff has to be capable of representing any information, most abstract as well as most ordinary life information, simple as well as most complex one. In this respect, it seems senseful to put the following question: "Is it possible to state explicitly what will be the limits of formula formation or is it at all possible to set any fixed limits which would disable realization of any general principle of information?" Within this dilemmatic view of formation of an informational formula we shall develop some basic rules of formation, not saying that these rules are the only possible ones.

Already within the principles of information ([4] or [10] respectively) it was shown how informational formulae can be composed on the level of natural language. This experience tells us that informational formulae are in no way limited sequences of informational operands and informational operators in relation to spoken and written language. Relationships within information are objective (operand-characteristic) and subjective (operational) and can be changed from the operand to operational states and vice versa during an informational process. Thus, a local informational operator can be viewed as an operational variable in a wider informational observation. Due to this phenomenology of informational compositions of operands and operators, iwffs are, in general, not structurally limited in any particular way. Limitations can be determined in cases of particular informational theories, concerning, for instance, formal logical systems as traditional symbolic logic, modal logic, etc., which can be conceived as special projections (particularizations) of informational logic.

The next fact to be explicated is, that sets of objects of IL are generative, i.e. not limited (determined once and for ever) in advance. Only static theories deal with finite and strictly determined sets of objects and as such are understood to be the most informationally primitive (static) forms of IL. In this respect, the set of formation rules of IL will not be semantically limited and informational operators will be recruited depending on needs, goals, and applications, which arise during an informational process. Similar will hold for informational operands occurring in informational formulae. The principle of informational arising will govern the arising of concrete formation rules (concerning concrete informational operators). However, it will be possible to present the essential framework of the arising of formation rules within IL.

II.2.1. Informational Operands as Constituents of IWFFs

The nature of information is variable in its arising, changing, vanishing, and disappearing. An informational operand is such a sort of variable information. We have the following basic definition concerning an informational operand as iwff:

[Operands]^{DF1}:
Informational variable α, defined as informational operand, is iwff. This operand can represent various kinds of information belonging to an informational realm. Thus,

('α is informational operand') $\Rightarrow$ ('α is iwff') ■

In many cases it is reasonable to separate informational entities as variable operands. So, one can set the following definition:

[Operands]^{DF2}:
A set of informational variables $\alpha, \beta, \ldots, \gamma$, in which $\alpha, \beta, \ldots, \gamma$ are defined as informational operands, is iwff. These variables of the set can represent various kinds of information belonging to given informational domains. It is:

('$\alpha, \beta, \ldots, \gamma$ are informational operands') $\Rightarrow$ ('$\alpha, \beta, \ldots, \gamma$ is iwff') ■

In the last definition, the commas can be understood as informational operators, which connect operands into an informational set. The sequence $\alpha, \beta, \ldots, \gamma$ could as well be written in the following way:

$$\alpha \models_{comma} \beta \models_{comma} \ldots \models_{comma} \gamma$$

where $\models_{comma}$ is the set-connective informational operator representing the delimiter ",". A set of operands $\alpha, \beta, \ldots, \gamma$ can be represented by a resultant operand, say ξ, where

$$\xi = \alpha, \beta, \ldots, \gamma$$

In this case, the symbol '=' is the informational operator of representation. It has the meaning that ξ representatively informs $\alpha, \beta, \ldots, \gamma$ or

$$\xi \models_{=} \alpha, \beta, \ldots, \gamma$$

Let us set now the definition reverse to [Operands]^{DF1}:

[Operands]^{DF3}:
If α represents an iwff, then α is an informational operand:

('α is iwff') $\Rightarrow$
('α is informational operand') ■

This definition says that irrespective of how α is structured as iwff, in fact, it represents an informational operand (traditional variable). Or, in other words: irrespective of its structure, an iwff can always be used as informational operand or can be put under operation of an informational operator. Everything which is iwff can be operated or can become an operand in the structure of another, higher formula. In the iwff

$$\alpha, \beta, \ldots, \gamma$$

$\alpha, \beta, \ldots, \gamma$ are iwffs, so, each of these iwffs can have its own structure.

[Operands]^{DF4}:
If $\alpha, \beta, \ldots, \gamma$ are informational operands, then $\mathfrak{F}$ in $\mathfrak{F}(\alpha, \beta, \ldots, \gamma)$ is the so-called functional informational operand (fio) or implicit informational operator (iio). In an iwff, a fio performs as informational operand, however, it has the implicit operational property. In this respect, an informational operand can be a functional or a non-functional variable. Fios or iios will be marked by capital Gothic letters (for instance, $\mathfrak{A}$, $\mathfrak{B}$, $\mathfrak{C}$, etc. ■

In $\mathfrak{F}(\alpha, \beta, \ldots, \gamma)$, the operand variables $\alpha, \beta, \ldots, \gamma$ can be functional as well as non-functional. Some distinguished iios (or fios) are, for instance:

- $\mathfrak{A}$ general Informing, for instance, as Informing of the variable α,
- $\mathfrak{B}$ behavioral Informing or behavior of a being,
- $\mathfrak{C}$ counter-Informing of information,
- $\mathfrak{D}$ informational differentiation (which could be the synonym for counter-Informing),
- $\mathfrak{E}$ informational embedding (of counter-information or new information into source information),
- $\mathfrak{F}$ general implicit functional operator,
- $\mathfrak{I}$ Informing or informational integration,
- $\mathfrak{L}$ motor or behavioral Informing of a being,
- $\mathfrak{M}$ metaphysical Informing of a being,
- $\mathfrak{P}$ informational particularization (subscription of informational operators) and informational universalization (superscription of informational operators),
- $\mathfrak{S}$ sensory Informing of a being, etc.

Marking by Gothic letters does not mean that also capital Latin letters cannot be used (according to [Variables]^{DF1}) for the purpose of marking implicit operators within the operand expressions. In this sense, for the above list of markers of Informing, also the Latin letters A, B, C, D, E, F, I, L, M, P, S, etc. can be used, respectively. Gothic letters are introduced for better distinctness of implicit operators in the expression of operands.

II.2.2. Informational Operators as Constituents of IWFFs

In IL operators are understood to be variable. In general, informational operators, presented by the metasymbol $\models$, will belong to a generative, potentially unlimited set of informational operators. In contrast to the so-called implicit informational operators marked by capital Gothic letters, metaoperators belong to the so-called explicit informational operators, which will be marked by distinctive special symbols. The set of informational operators is generated by the two already mentioned informational procedures, called particularization and universalization of existing metaoperators or already particularized or universalized operators. The process of particularization or universalization can always begin from the most general informational operator $\models$, which as metasymbol represents the so-called general operational variable. In IL, on the level of informational operations, we regularly have to deal with operational variables rather than with operational constants. Informational operands, as well as informational operators, underlie the so-called principle of informational arising.

[Operators]^{DF42}:
The informational operator $\models$ is operational variable and is the sub-iwff. This operational variable represents various kinds of informational operators, which can be generated by particularization and universalization of $\models$, according to the needs, goals, application, and understanding of an informational formula, which is an iwff representing an informational form, process, or phenomenon. In the same way as does the operator $\models$, the particularized and universalized operators underlie the philosophy of their further (recurrent) particularization and universalization. Thus,

('$\models$ is operational variable') $\Rightarrow$
('$\models$ is sub-iwff')

('$\models_{part}$ is operational variable') $\Rightarrow$
('$\models_{part}$ is sub-iwff')

$('\models^{univ}$ is operational variable') $\Rightarrow$
$('\models^{univ}$ is sub-iwff')

where $\models_{part}$ is a particularized explicit operator and $\models^{univ}$ is a universalized explicit operator. Further, it is possible to mark

$(\models_{part}) \equiv \mathfrak{P}_{\downarrow}(\models)$ and $(\models^{univ}) \equiv \mathfrak{P}^{\uparrow}(\models)$

where $\mathfrak{P}_{\downarrow}$ is the implicit informational operator of particularization and $\mathfrak{P}^{\uparrow}$ the implicit informational operator of universalization. ■

In several cases it is reasonable to concentrate informational operators into (regular) functional compositions of operators. In such cases, the following definition can be adopted:

[Operators]^{DF43}:
A set (type) $\models$ of particularized and universalized operational variables or informational operators, marked for example as $\models_1, \models_2, \ldots, \models_m$, is the basis from which these elements can be taken to construct the so-called operational concatenations in the following way:

(1) $\models_{con} \equiv \models_i$, where $\models_i \in \models$, is operational concatenation (OC);

(2) $\models_{con} \equiv (\models_{con}\models_j)$, where $\models_j \in \models$, is a recursive definition of OC.

If $\models_{con}$ is an OC, one can write instead of (1) and (2):

$(1)^*$ $(\models_i \in \models) \Rightarrow (\models_{con} \equiv \models_i)$

$(2)^*$ $((\models_j \in \models) \wedge ('\models_{con}$ marks OC')) $\Rightarrow$
$(\models_{con}\models_j)$

Expression $(2)^*$ is a kind of informational modus (a particular form of the so-called modus informationis):

$$\frac{\models_j \in \models,\ ('\models_{con} \text{ denotes OC'})}{('\models_{con}\models_j \text{ is OC'})}$$

■

The consequence of [Operators]^{DF43} is that $\models_{con}$ is a word belonging to the set

$$(\models_1, \models_2, \ldots, \models_m)^* \setminus \{\}$$

where $\{\}$ denotes the empty set. $\models_{con}$ is a composition of informational operators in an arbitrary complex (interweaving) way. In a particular case, $\models_{con}$ can be the linear (usual, mathematical) composition of operators. The complex composition means a parallel (interweaving) activity of operators constituting $\models_{con}$. For instance, we shall allow the notation $\models^{\dashv}$ instead of $\models\dashv$, where in $\models^{\dashv}$ the relation of dominance $\models \Delta \dashv$ will be assumed. It is evident that among operators, constituting $\models_{con}$, additional dependencies (relations) can be determined.

[Operators]^{DF44}:
If $\models_{con}$ represents a sub-iwff, then $\models_{con}$ is an informational operator (or operational variable):

$('\models_{con}$ is sub-iwff') $\Rightarrow$
$('\models_{con}$ is an operational variable') ■

This definition says that irrespective of how $\models_{con}$ as a sub-iwff is structured, in fact, it represents a composite (complex) operational variable in which its components (suboperators) are variables too. Formally, as a concatenation of operational variables, $\models_{con}$ functions as an operator composition. In regard to an iwff, a sub-iwff is in some way a reduced form of the iwff concept. This reduction is semantically presented as the prefix 'sub' in sub-iwff, which is a concatenation of informational operators and is marked by operator $\models_{con}$.

II.2.3. Parenthetic Delimiters and Parenthesizing of IWFFs

In fact, parenthetic delimiters can be understood as the delimiting informational operators within an iwff. Their function is to determine the so-called iwff's unities within a formula. A unity of an iwff can be used as operand of a higher operator structure. For parenthetic delimiters arbitrary symbols can be introduced, for instance, parentheses, brackets, etc. Besides parentheses, it is possible to introduce the so-called non-substantial delimiter, by which the so-called non-substantial part of an iwff will be marked. So, let us have the following definition:

[Delimiters]^{DF1}:
Irrespective of their choice, the parenthetic delimiters occurring in an iwff unite parts of the iwff or the whole iwff, with the intention to define the unit they delimit to be used for some operation over the unit. Parenthetic delimiters occur always in pairs, consisting of the beginning and the ending delimiter, and can be nested within other pairs of delimiters. Usually, parenthetic delimiters will be denoted by '(' for the beginning and by ')' for the ending delimiter. However, also other kinds of delimiters can be introduced, for instance, the pairs [,] and {, }, all of which can be understood as particularizations of the delimiter operators $\models_{beg}$ and $\models_{end}$, denoting the beginning and the ending parenthesis. ■

[Delimiters]^{DF2}:

Parenthetic delimiters can be used in pairs in such a way, that they delimit an expression within an iwff, which is either an iwff or a sub-iwff. In this case the usual nesting principle of parenthesizing is valid. ■

[Delimiters]^{DF3}:
A special, unary delimiter is in fact the symbol, marking the so-called non-substantial or self-comprehensive part of an iwff. This delimiter is composed of three consecutive dots, thus '...'. The three-dot delimiter is a legal symbol of an iwff. ■

[Delimiters]^{EX1}:
Considering the previous three definitions, the legal formulae or iwffs are, for instance:

... α, α ..., α ... β, ...(...(...))

With the last formula we can even determine the positioning of parentheses in an iwff. Evidently, this formula can be rewritten as α(β(γ)), where the entities α, β, and γ are understood as unsubstantial parts of the formula in question. ■

II.2.4. Some Basic Processes within the Formation of Formulae

So far, we have used the following basic processes in the formation of a formula:

(1) Introducing of informational operands as constituents of an arising formula, where the operand as a variable represents an iwff.

(2) Setting of informational operands into sets of variables, where distinct variables were separated by commas. Such a set of variables was declared to be the iwff.

(3) Introducing of explicit informational operators of the type ⊨ in an arising formula and concatenating them by other informational operators into a sub-iwff (OCs) within an iwff.

(4) Introducing of explicit informational operators and their operational concatenations (OCs) and concatenating them with operands and their informational sets, thus formatting an iwff.

(5) Introducing of implicit informational operators of the type ℱ (functional operands) into the operand parts of an arising formula and concatenating them (functionally) to an parenthesized informational set of operand variables.

(6) Introducing of explicit and implicit informational operators in a particularized and universalized way of their choice. Even if operational particularization and universalization are the basic formatting principles, they could be understood first of all as formula transformation principles (see subsection II.5).

(7) Formatting a complete formula (iwff) means to use rules (1)-(6) in an arising manner. In this respect an instantaneous formula can be always developed by further steps proceeding from one formational state to the other in a growing (enlarging) or a vanishing (reducing) manner.

II.2.5. Formation Rules of IL

In the previous definitions of the subsection II.2 we gave the rules for formation of an iwff in the following way: it was said what operands and operators constituting a formula are. Then it was said how operands and operators can be combined or concatenated into a formula. Also, the use of the so-called delimiters, which determine the units or subunits of a formula, was described. There were not any particular restraints for formula formation. In general, combining of operands, operators, and delimiters in the described way, leads to the formation of a formula. In such formatting processes, particularization and universalization of operators are still possible.

[Formatting of formulae]^{DF1}:
An informational well-formed formula (iwff) can be constructed by the use of the definitions [Operands]^{DF1} - [Operands]^{DF4}, [Operators]^{DF42} - [Operators]^{DF44}, and [Delimiters]^{DF1} - [Delimiters]^{DF3} by concatenating (combining) operands, operators, and delimiters according to the description in subsection II.2.5. With this procedure production of an iwff is assured. ■

[Formatting of formulae]^{DF2}:
Informational well-formed formulae, being constructed according to the previous definition, can be composed into the so-called informational system. In this system, distinct iwffs are separated by commas or/and can appear in different lines. Similarly as a single iwff can be marked by a symbol of operand, the informational system can be marked by a symbol of operand. Informational system performs as iwff. ■

Later on, discussing the transformation rules, we shall show how formulae can be decomposed into systems and how systems of formulae can be composed into formulae under certain circumstances. In this essay we have given several examples, which clarify and illustrate the principles of formula formatting.

II.2.6. What Could Be a Non-Informational Formula?

The first approach to the topical question could be in questioning what are already informational formulae. The answer to this counter-question is that all mathematical formulae certainly belong to the class of informational formulae. However, informational formulae are not necessarily mathematical, for mathematics does not deal with the creative component of information (Informing, arising, generating, functional changing, etc.). The semantics of informational operators and operands as variable subjects and objects is generative, while mathematically chosen objects are invariably determined.

The answer to the topical question is that anything we form out of informational operands, operators, and parenthetic delimiters is either iwff or sub-iwff. Certainly, there exist some unessential distinctions. For instance, if we take two informational variables, say α and β, we can form several compositions, such as

$$\alpha\beta, \quad \alpha, \beta, \quad \alpha(\beta), \quad (\alpha)\beta,$$

etc. The first case can be interpreted as informational concatenation of α and β, which is already known as operational concatenation of the type $\models_{con}$. In a similar way it is possible to determine the so-called operand concatenation and the concatenation of mixed operand and operator components. This is already performed in the case of iwff formation. The second case concerns the so-called informational set of components α and β. The third case can, in general, have two substantially different interpretations: α can be an implicit operator and β its argument, or the parentheses '(' and ')' are used simply as formula delimiters, where α is an explicit informational operator. In the fourth case we have to do with a similar case as in case three, etc. One can feel that any formula expression can have its informational sense, interpretation, understanding, and that it is not possible to say decisively what an informational formula cannot be.

The last statement is evidently only a consequence of the fact that there is not anything which could not be informational. This "all-embracing" principle governs not only the realm of the real information (metaphysics), but also its concept of self-formalization. In this respect, informational logic introduces a free, legal, and effective concept for dealing with cases concerning the generative, creative, or intelligent nature of information. It has to be said that the introduced semantics of IL is initial and has still to be elaborated and developed to the levels, where it would satisfy more properly the needs of a particular arising of information.

II. 3. INFORMATIONAL AXIOMS

... and in the end one is left with the no-theory theory of meaning, the deflationary thought that the question that now define the philosophy of meaning and intentionality all have false presuppositions.

Stephen Schiffer [11] 3

II.3.0. Axiomatization of Informational Principles

We shall now come to the point at which it will be possible to make an authentic self-experience how axiomatization of the so-called principles of information ([4] or [10]) opens the abyss of making these principles formalistic, i.e. to give them forms of iwffs. Again, the semantics of operands and operators will take its informational power. In fact, by axiomatization of informational principles, we are entering the abysmal domain of informational phenomenology. If somebody is in doubt whether this is or is not so as we have stated right now, let him simply follow the experiments and intentions of the subsequent informational axiomatization. The nature of the mentioned abyss is that it is practically inexhaustible, that axiomatization as an informational approach can generate an unlimited and unforeseeable set of axioms, where the end cannot be seen at all. It may sound surprisingly that informational axioms, although intuitively deduced from informational principles, can and will illuminate these principles in a refreshing, new, and theoretically elucidated way.

We are now coming to the point where we have to decide what can be an informational axiom. The basic association of ideas is that the so-called informational principles (discussed in [4] and also in [10]) have to be axiomatized. The first impression is that the main difficulty may lie in the principle of informational arising which concerns all informational entities, operands, as well as operators of a formula. Informational arising is the semantic property of operands and operators. Results of applying informational operators is the arisen information, for instance, counter-information. Simultaneously to the arisen information also Informing of information arises or new informational components are coming into existence. Evidently, these basic mechanisms of information have to be formally axiomatized. To enter into the discourse of informational arising a recurrent and informationally interwoven approach is necessary.

II.3.1. The Main Axiomatic Principles

II.3.1.0. How Axioms Can Be Generated

It is simply possible to follow principles of information [4] and make the list of axioms,

which will be constructed in the next paragraphs. Thus, we shall have the following axioms concerning the subsequent informational entities: Informing and mutual Informing of information, informational difference, informational circularity, informational spontaneity, arising of information, counter-information, counter-Informing of information, informational embedding, informational embedding of counter-information, informational differentiation, informational integration, informational particularization and informational universalization, informational formula, informational structure and informational organization, informational parallelism, informational cyclicity, openness of informational axiomatization, informational axioms and metaphysical beliefs, and axiomatic consequences of informational arising.

II.3.1.1. Axioms of Informing of Information

If we say that α marks an informational entity, i.e. information, then it is supposed that this entity has the property of inward (own) and outward (concerning other information) informational development or Informing. For this informational property of Informing the operator variable or general metaoperator $\models$ was introduced. According to our previous discussion we can propose the following axioms:

[Axioms]^{DF1}:

(1) ('α is information') $\Rightarrow$ $((\alpha \models) \vee (\models \alpha))$

(2) ('α is information') $\Rightarrow$ $((\models \alpha) \vee (\alpha \models))$

(3) $((\alpha \models) \vee (\models \alpha) \vee (\models \alpha) \vee (\alpha \models)) \Rightarrow$
('$\Im_\alpha(\alpha)$ is coming into existence')

(4) $((\alpha \models) \vee (\models \alpha) \vee (\models \alpha) \vee (\alpha \models)) \Rightarrow$
$((\alpha \models \alpha) \vee (\alpha \models \alpha))$ ■

The comments to these axioms are the following:

(1) If α is information, then α informs in one ($\models$) or another way ($\models$).
(2) If α is information, then α is informed in one ($\models$) or another way ($\models$).
(3) If α informs and is informed in one or another way, then Informing of α over α itself ($\Im_\alpha(\alpha)$) is coming into existence.
(4) If α informs and is informed in one or another way, then α informs itself and/or is informed by itself.

[Axioms]^{DF2}:
[of Mutual Informing of Information]

(1) ('α and β are informationally interwoven') $\Rightarrow$
$((\alpha \models \beta) \vee (\beta \models \alpha) \vee (\beta \models \alpha) \vee (\alpha \models \beta))$

(2) $(\alpha \models \beta) \Rightarrow$
$((\alpha \models \beta) \vee (\beta \models \alpha) \vee (\beta \models \alpha) \vee (\alpha \models \beta))$

(3) $(\beta \models \alpha) \Rightarrow$
$((\alpha \models \beta) \vee (\beta \models \alpha) \vee (\beta \models \alpha) \vee (\alpha \models \beta))$

(4) $(\beta \models \alpha) \Rightarrow$
$((\alpha \models \beta) \vee (\beta \models \alpha) \vee (\beta \models \alpha) \vee (\alpha \models \beta))$

(5) $(\alpha \models \beta) \Rightarrow$
$((\alpha \models \beta) \vee (\beta \models \alpha) \vee (\beta \models \alpha) \vee (\alpha \models \beta))$

(6) $((\alpha \models \beta) \vee (\beta \models \alpha)) \Rightarrow$
$((\beta \models \alpha) \vee (\alpha \models \beta))$

etc. ■

The comment to these axioms is that the informational operators $\models$ and $\models$ have metaequivalent power and that in mutual Informing of information entities all possible cases of Informing of the involved information can be considered.

II.3.1.2. Axioms of Informational Difference

Informational difference concerning two informational items is the most natural informational property. This fact can be expressed by the following axioms:

[Axioms]^{DF3}:

(1) $(('\alpha$ is information') $\wedge$ ('β is information')) $\Rightarrow$ $(\alpha \neq \beta)$

(2) $((\alpha \models \beta) \vee (\beta \models \alpha)) \Rightarrow (\alpha \neq \beta)$

(3) $(('\alpha$ is datum') $\wedge$ ('β is datum')) $\Rightarrow$ $(\models_\pi (\alpha = \beta))$

(4) $(('\alpha$ is information') $\wedge$ ('β is information')) $\Rightarrow$ $(\not\models_\pi (\alpha = \beta))$

(5) $((\alpha \models) \vee (\models \alpha)) \Rightarrow (\mathrm{L}\ \delta_\alpha(\alpha))$

(6) $((\models \alpha) \vee (\alpha \models)) \Rightarrow (\delta_\alpha(\alpha)\ \mathrm{J})$

(7) $\delta_\alpha(\alpha) \equiv_\pi \omega(\alpha)$

In (5)-(7), $\delta_\alpha(\alpha)$ denotes the difference arising as Informing of α over itself and $\equiv_\pi$ marks the possible equivalence between informational difference and arising of counter-information from α. ■

For instance, a pure logical axiomatic conclusion would be that

$$(\alpha \neq \beta) \Rightarrow (\not\models_\pi (\alpha = \beta))$$

From (3) it is understood that only data can be equal. Thus, the equality between two informational items is possible in the realm of information, which represents information as data, that is on the informationally static basis. Sooner or later informational equality remains very unnatural and lifeless informational property.

[Axioms]^{DF4}:
If β is information and if $\alpha = \beta$, then α is the marker for β. In this case α is the so-called marking information, which has the nature of information β whose marker it is. Formally,

$$(('\beta \text{ is information'}) \wedge (\alpha = \beta)) \Rightarrow ('\alpha \text{ is the marker of } \beta') \blacksquare$$

II.3.1.3. Axioms of Informational Circularity

[Axioms]^{DF5}:

(1) $(\alpha \models) \Rightarrow (\alpha \models \alpha)$

(2) $(\alpha \models) \Rightarrow (\models \alpha)$

(3) $(\models \alpha) \Rightarrow (\alpha \models \alpha)$

(4) $(\models \alpha) \Rightarrow (\alpha \models)$

(5) $((\alpha \models) \wedge (\models \alpha)) \Rightarrow (\models \alpha \models)$

(6) $(\alpha \models \alpha) \Rightarrow ((\alpha \models) \vee (\models \alpha))$

etc. If α informs, then it informs itself (1). If α informs, then it is informed (2). If α is informed, then it is informed by itself (3). If α is informed, then it informs (4). If α informs and if α is informed, then it is informed that it informs (5). If α informs itself or if α is informed by itself, then it informs or it is informed (6). Etc. Evidently, axioms of these types can be generated infinitely. ■

[Axioms]^{DF6}:

(1) $(\alpha \models \beta) \Rightarrow ((\alpha \models) \wedge (\models \beta))$

(2) $(\alpha \models \beta) \Rightarrow_{\pi} (\beta \models \alpha)$

(3) $(\alpha \models \beta) \Rightarrow ((\alpha \models \beta) \models)$

(4) $(\alpha \models \beta) \Rightarrow_{\pi} ((\alpha \models \alpha) \wedge (\beta \models \beta) \wedge ((\alpha \models \beta) \models (\alpha \models \beta)))$

etc. If α informs β or β is informed by α, then α informs and β is informed (1). If α informs β or β is informed by α, then it is possible that β informs α or α is informed by β. If α informs β or β is informed by α, then this Informing informs. If α informs β or β is informed by α, then it is possible that α informs itself, β informs itself, and that this Informing informs itself. Etc. Evidently, axioms of these types can be generated indefinitely. ■

Obviously, it can be understood how information and Informing of information perform circularly. Circularity is the basic informational phenomenology.

II.3.1.4. Axioms of Informational Spontaneity

Synonyms of the adjective "spontaneous" may be unforeseeable, generative, and arising. Informational spontaneity is the property of unforeseeable arising of information and of Informing of information. Spontaneity concerns information as informational operand and Informing as informational operator. Spontaneous means to be capable to arise, to come into existence in a possibly unpredictable way. Spontaneity belongs to the most primitive properties of information. This yields the most simple form of the axioms concerning spontaneity.

[Axioms]^{DF7}:

(1) $('\alpha \text{ is information'}) \Rightarrow ((\alpha \models) \vee (\models \alpha))$

(2) $((\alpha \models) \vee (\models \alpha)) \Rightarrow ('\Im_{\alpha}(\alpha) \text{ is spontaneous'})$

(3) $((\alpha \models) \vee (\models \alpha)) \Rightarrow ((\alpha \models \Im_{\alpha}(\alpha)) \vee (\Im_{\alpha}(\alpha) \models \alpha))$

(4) $((\alpha \models) \vee (\models \alpha)) \Rightarrow (\Im_{\alpha}(\alpha) \models \Im_{\alpha}(\alpha))$

(5) $('\Im_{\alpha}(\alpha) \text{ is Informing'}) \Rightarrow (\exists\alpha.((\alpha \models) \vee (\models \alpha)))$

etc. If α is information, then α informs or is informed (1). Spontaneity in this axiom is hidden in semantics of the metaoperator ⊨. This fact is expressed in (2). If α informs or is informed, then Informing of α, $\Im_{\alpha}$, is an implicit functional operator over α, which is informed by α or informs α (3). If α informs or is informed, then Informing of α over α, $\Im_{\alpha}$, informs itself or is informed by itself. If $\Im_{\alpha}(\alpha)$ is Informing of α over α, then there exists such an α that (.) α informs or is informed (5). Etc. Obviously, axioms of informational spontaneity can be generated infinitely. ■

II.3.1.5. Axioms of Informational Arising

The principle of informational arising is the most basic principle of the theory we call informational logic. This principle hides several other, particular informational principles, for instance, the principles of spontaneity, circularity, Informing, counter-Informing, parallelism, etc. In this respect, this principle has an integrative, originating, and conceptual role in the development of informational theories. The principle that all informational is under the protection and influence of arising, which simultaneously is the synonym for coming into existence, changing and vanishing, guarantees the most possible dynamic nature of information, as it is understood by modern common sense. Of course, the question of new semantic power of informational operators and their operands arises: How can they be determined to surpass the traditional mathematical terminology? How can they be treated to overcome the rationalistic philosophical blockade? How can they exclude, for instance, the principle of truth as the only possible logical means? Several of these efforts have been already presented in the previous text of this essay.

[Axioms]^{DF8}:

(1) $((\alpha \models) \vee (\models \alpha)) \Rightarrow ('\Im_{\alpha}(\alpha) \text{ arises'}) \vee ('\Im_{\alpha}(\alpha) \text{ is coming into existence'}) \vee ('\Im_{\alpha}(\alpha) \text{ is generated'}) \vee$

('$\Im_\alpha(\alpha)$ is changing') ∨
('$\Im_\alpha(\alpha)$ is vanishing') ∨ ...

(2) $((\Im_\alpha(\alpha) \models) \vee (\models \Im_\alpha(\alpha))) \Rightarrow$
('α arises') ∨
('α is coming into existence') ∨
('α is generated') ∨
('α is changing') ∨
('α is vanishing') ∨ ...

These two axioms have to be treated as a unique, axiomatically interwoven system in which information α and its Informing $\Im_\alpha(\alpha)$ are mutually dependent. ■

The axiomatic question for the last axiom is where do the arising, coming into existence, generating, changing, vanishing, etc. of information and its Informing originate from. The answer to this question is given by the following axiom:

[Axioms]^{DF9}:

(1) $((\alpha \models) \vee (\models \alpha)) \Rightarrow (\alpha \models \Im_\alpha(\alpha))$

(2) $((\Im_\alpha(\alpha) \models) \vee (\models \Im_\alpha(\alpha))) \Rightarrow (\Im_\alpha(\alpha) \models \alpha))$

Without changing essentially the meaning of these axioms, they can be widened in the following manner:

(1) $((\alpha \models) \vee (\models \alpha)) \Rightarrow (\alpha \models \Im_\alpha(\alpha)) \vee (\Im_\alpha(\alpha) \models \alpha))$

(2) $((\Im_\alpha(\alpha) \models) \vee (\models \Im_\alpha(\alpha))) \Rightarrow (\alpha \models \Im_\alpha(\alpha)) \vee (\Im_\alpha(\alpha) \models \alpha))$

In this respect, the consequences of information and its Informing are the same. ■

[Axioms]^{EX1}:

Interpretation of the last axioms by means of the previous ones can be given through various particularizations of the metaoperator ⊨. If we introduce particularizations of ⊨

- $\models_{ari}$ for 'arises from or causes the arising of'
- $\models_{exi}$ for 'comes into existence from or causes the coming into existence of'
- $\models_{gen}$ for 'is generated or generates'
- $\models_{cha}$ for 'is changed or changes'
- $\models_{van}$ for 'is vanished or vanishes'

then it is possible to express [Axioms]^{DF8} in the following way:

(1) $((\alpha \models) \vee (\models \alpha)) \Rightarrow$
$(\alpha \models_{ari} \Im_\alpha(\alpha) \models_{ari} \alpha) \vee$
$(\alpha \models_{exi} \Im_\alpha(\alpha) \models_{exi} \alpha) \vee$
$(\alpha \models_{gen} \Im_\alpha(\alpha) \models_{gen} \alpha) \vee$
$(\alpha \models_{cha} \Im_\alpha(\alpha) \models_{cha} \alpha) \vee$
$(\alpha \models_{van} \Im_\alpha(\alpha) \models_{van} \alpha) \vee \ldots$

(2) $((\Im_\alpha(\alpha) \models) \vee (\models \Im_\alpha(\alpha))) \Rightarrow$
$((\Im_\alpha(\alpha) \models_{ari} \alpha \models_{ari} \Im_\alpha(\alpha)) \vee$
$((\Im_\alpha(\alpha) \models_{exi} \alpha \models_{exi} \Im_\alpha(\alpha)) \vee$
$((\Im_\alpha(\alpha) \models_{gen} \alpha \models_{gen} \Im_\alpha(\alpha)) \vee$
$((\Im_\alpha(\alpha) \models_{cha} \alpha \models_{cha} \Im_\alpha(\alpha)) \vee$
$((\Im_\alpha(\alpha) \models_{van} \alpha \models_{van} \Im_\alpha(\alpha)) \vee \ldots$ ■

In general, it has to be understood that the process of axiomatization of informational arising can be continued indefinitely.

II.3.1.6. Axioms of Counter-Information

Formation, appearance, or coming into existence of the so-called counter-information is a consequence of Informing of information. Counter-information is a result of the informational phenomenology of information. Counter-information arises from information, from information as activity over itself. Appearance of other information, which is not counter-informational, may be called sensory or outward information in respect to the so-called source information or information in question.

[Axioms]^{DF10}:

Let ω denote counter-information and let ω(α) be counter-information which arises from information α. Let the meaning of operators ⌞ and ⌟ be 'causes the appearance of' or 'comes into existence from', respectively. There is possible to set several axioms, for instance:

(1) $((\alpha \models) \vee (\models \alpha)) \Rightarrow (\alpha \llcorner \omega(\alpha))$
(2) $((\dashv \alpha) \vee (\alpha \dashv)) \Rightarrow (\omega(\alpha) \lrcorner \alpha)$
(3) $((\alpha \models) \vee (\models \alpha)) \Rightarrow ((\alpha \models \omega(\alpha)) \wedge (\omega(\alpha) \models \alpha))$
(4) $((\dashv \alpha) \vee (\alpha \dashv)) \Rightarrow ((\omega(\alpha) \dashv \alpha) \wedge (\alpha \dashv \omega(\alpha)))$

etc. The axioms (1) and (2) are already particularized because counter-information ω(α) appears as the consequence of information α. In the axioms (3) and (4) the most general operators ⊨ and ⊣ are used. ■

We have already shown some axiomatic constructions concerning counter-information in [Operators]^{DF24} and [Operators]^{EX5}. Evidently, axiomatic constructions concerning counter-information can be continued indefinitely.

II.3.1.7. Axioms of Counter-Informing of Information

Counter-Informing is a component of Informing by which information is producing its counter-information. This component is interwoven in the Informing of information. It acts upon information as a subject causing the appearance of counter-information. Similarly as information is informing, counter-information is counter-informing. The acceptance of counter-information by information depends on the so-called informational embedding of counter-information into the source

information. Counter-Informing belongs to the most subtle processing components of information.

[Axioms]^{DF11}:
Let $\mathfrak{C}$ be counter-Informing and let $\mathfrak{C}(\alpha)$ denote counter-Informing which arises within the Informing of information α. It is possible, for example, to set the following axioms:

(1) $((\alpha \models) \vee (\models \alpha)) \Rightarrow ((\alpha \llcorner \mathfrak{C}(\alpha)) \wedge (\mathfrak{C}(\alpha) \models \omega(\alpha)))$

(2) $((\dashv \alpha) \vee (\alpha \dashv)) \Rightarrow ((\mathfrak{C}(\alpha) \lrcorner \alpha) \wedge (\omega(\alpha) \dashv \mathfrak{C}(\alpha)))$

(3) $((\alpha \models) \vee (\models \alpha)) \Rightarrow ((\alpha \models \mathfrak{C}(\alpha)) \wedge (\mathfrak{C}(\alpha) \models \alpha) \wedge (\mathfrak{C}(\alpha) \models \omega(\alpha)) \wedge (\omega(\alpha) \models \alpha))$

(4) $((\dashv \alpha) \vee (\alpha \dashv)) \Rightarrow ((\mathfrak{C}(\alpha) \dashv \alpha) \wedge (\alpha \dashv \mathfrak{C}(\alpha)) \wedge (\omega(\alpha) \dashv \mathfrak{C}(\alpha)) \wedge (\alpha \dashv \omega(\alpha)))$

etc. In the axioms (1) and (2), the operators $\llcorner$ and $\lrcorner$ can be understood as particularizations of the operators $\models$ and $\dashv$, respectively. ■

II.3.1.8. Axioms of Informational Embedding

Let, for instance, sensory or outward information σ arrive into informational domain of the so-called source information α. In this case, the perception of σ by α is possible only through Informing of σ, namely in the way that α is informed by σ. In this Informing, α is in no way a passive actor, because the state of being informed is in fact Informing within information in the presence of the outwardly appeared cause, i.e. information σ. The acceptance or perception of this Informing is called informational embedding, in general. The nature of informational embedding is to embed the arriving information into the source information. Evidently, embedding in this sense is a dynamic, unforeseeable Informing. Informational embedding explains and illuminates Informing of information from a particular point of understanding.

It is evident that Informing of the arriving information σ can only be informational, or that the acceptance or perception of σ by α can only be informationally particular. The only exception from this general principle can be observed within the so-called data processing, occurring in traditional, lifeless machines.

[Axioms]^{DF12}:
Let α be source information, σ arriving (sensory, outward) information, $\mathfrak{E}$ informational embedding, and ε information embedded into α by $\mathfrak{E}$. A series of axioms of informational embedding can be constructed:

(1) $(\sigma \models \alpha) \Rightarrow (('\sigma \text{ is embedded into } \alpha') \wedge ('\alpha \text{ is embedded into } \sigma'))$

(2) $(\sigma \models \alpha) \Rightarrow ((\sigma, \alpha \llcorner \mathfrak{E}) \wedge (\mathfrak{E} \llcorner \varepsilon))$

(3) $(\alpha \dashv \sigma) \Rightarrow ((\mathfrak{E} \lrcorner \alpha, \sigma) \wedge (\varepsilon \lrcorner \mathfrak{E}))$

(4) $((\sigma \models \alpha) \llcorner \mathfrak{E}, \varepsilon) \Rightarrow ((\mathfrak{E} \equiv \mathfrak{E}_{\sigma,\alpha}) \wedge (\varepsilon \equiv \mathfrak{E}_{\sigma,\alpha}(\sigma, \alpha)))$

(5) $(\varepsilon, \mathfrak{E} \lrcorner (\alpha \dashv \sigma)) \Rightarrow ((\mathfrak{E} \equiv \mathfrak{E}_{\alpha,\sigma}) \wedge (\varepsilon \equiv \mathfrak{E}_{\alpha,\sigma}(\sigma, \alpha)))$

(6) $(\sigma \models \alpha) \Rightarrow ((\varepsilon \subset \alpha) \wedge (\sigma \not\subset \alpha))$

(7) $(\sigma \models \alpha) \Rightarrow (\varepsilon \therefore \sigma, \alpha)$

etc. It is evident that the so-called axioms of informational embedding can be constructed indefinitely, for instance, by using the principle of particularization, etc. ■

Let us comment the listed axioms. In general, if σ informs α, then an informational interaction between σ and α occurs in the form that σ and α are simultaneously embedded in each other. This fact is easily understood in the case of a living being, where sensory and, for instance, perceptual (or cortical) information influence each other. If σ informs α in one (2) or another way (3), then informational embedding $\mathfrak{E}$ arises from σ and α and information of embedding ε appears as a consequence of embedding as Informing. If σ informs α in one (4) or another way (5) and this Informing causes the appearance of embedding $\mathfrak{E}$ and information of embedding ε, then in one case (4) the embedding is equivalent to $\mathfrak{E}_{\sigma,\alpha}$ and information of embedding is equivalent to $\varepsilon_{\sigma,\alpha}$, and in another case (5) the embedding is equivalent to $\mathfrak{E}_{\alpha,\sigma}$ and information of embedding is equivalent to $\varepsilon_{\alpha,\sigma}$. If σ informs α (6), then ε becomes a part ($\subset$) of α, however σ is not a part ($\not\subset$) of α. Simultaneously, ε is similar to σ as well as α (7).

It is certainly possible to introduce the explicit informational operator of embedding, for instance $\models_{\mathfrak{E}}$, which could be even more comprehensible than its implicit (functional) counterpart, denoted by $\mathfrak{E}_{\alpha,\alpha}(\sigma)$. The meaning of this implicit case is 'σ is in the process to be embedded into α by α'.

II.3.1.9. Axioms of Informational Embedding of Counter-Information

In contrary to sensory information, counter-information is a product of information itself. It appears as a kind of inward sensory information, which has to be perceived by information itself. This self-perception is performed through the process of embedding $\mathfrak{E}$.

[Axioms]^{DF13}:
In this axiom we use the following symbols: α is information which informs, however, also informs in itself. ω is counter-information which comes into existence through self-Informing of information α. Further, $\mathfrak{C}$ denotes counter-Informing within α and $\mathfrak{E}$ denotes the process of embedding performed by α. The following axioms of informational embedding of counter-information are only a few of possible ones:

(1) $(\alpha \models \alpha) \Rightarrow ((\alpha \mathrel{\mathsf{L}} \omega) \wedge (\alpha \models_{\mathfrak{E}} \omega))$

(2) $(\alpha \dashv \alpha) \Rightarrow ((\omega \mathrel{\mathsf{J}} \alpha) \wedge (\omega \dashv_{\mathfrak{E}} \alpha))$

(3) $(\alpha \models \alpha) \Rightarrow ((\alpha \models_{\mathfrak{C}} \omega) \wedge (\alpha \models_{\mathfrak{E}} \omega))$

(4) $(\alpha \dashv \alpha\nu \Rightarrow ((\omega \dashv_{\mathfrak{C}} \alpha) \wedge (\omega \dashv_{\mathfrak{E}} \alpha))$

(5) $(\alpha \mathrel{\mathsf{L}} \omega) \Rightarrow ((\alpha \models \mathfrak{C}_{\alpha}(\omega)) \wedge (\alpha \models \mathfrak{E}_{\alpha,\alpha}(\omega)))$

(6) $(\omega \mathrel{\mathsf{J}} \alpha) \Rightarrow ((\mathfrak{C}_{\alpha}(\omega) \dashv \alpha) \wedge (\mathfrak{E}_{\alpha,\alpha}(\omega) \dashv \alpha))$

etc. The axioms (3), (4), (5), and (6) bring the so-called cyclic (also circular) nature of information in the foreground, when information is understood as a cyclic process of counter-Informing and embedding of information. ■

The following comments to the last axioms are possible: If information α informs in itself in one (1) or another way (2), then, from α or by α, counter-information ω is coming into existence and this counter-information is embedded in α in one ($\models_{\mathfrak{E}}$) or another way ($\dashv_{\mathfrak{E}}$). If information α informs in one (3) or another way (4), then α counter-informs counter-information ω in one ($\models_{\mathfrak{C}}$) or another way ($\dashv_{\mathfrak{C}}$) and embeds counter-information ω in one ($\models_{\mathfrak{E}}$) or another way ($\dashv_{\mathfrak{E}}$). If α causes the appearance of counter-information ω in one (5) or another way (6), then information α informs its own counter-Informing $\mathfrak{C}_{\alpha}(\omega)$ in one or another way and informs its own informational embedding $\mathfrak{E}_{\alpha,\alpha}(\omega)$ in one or another way. The last four axioms constitute the cyclic nature of arising informational entities.

II.3.1.10. Axioms of Informational Differentiation

Differentiation of information is an inherent property of information which informs and is informed. Differentiation is not only informational arising, but arising of informational difference in comparison to the present state or processing of an informational entity. Differentiation is a component of informational arising with the intention to arise differently to existing information. The consequence of this fact is that information arises differently. To enter into the discourse concerning informational differentiation, we can introduce two basic and general differential operators which govern the so-called explicit and implicit informational differentiation.

[Operators]^{DF45}:
The explicit informational operator of differentiation can be determined in the following way:

$(\models_{\mathfrak{D}} \vee \dashv_{\mathfrak{D}}) =_{Df}$ ('differentiates') ∨ ('differentiate') ∨ ('is_differentiated(_by)') ∨ ('are_differentiated(_by)')

The implicit informational operator of differentiation can be determined as

$$\mathfrak{D}_{\alpha,\beta,\ldots,\gamma}(\xi, \eta, \ldots, \zeta)$$

with the following meaning: $\alpha, \beta, \ldots, \gamma$ differentiate $\xi, \eta, \ldots, \zeta$ or $\xi, \eta, \ldots, \zeta$ are differentiated by $\alpha, \beta, \ldots, \gamma$. ■

[Operators]^{EX14}:
To clear the meaning of the explicit informational operator of differentiation let us look at the following examples:

$\alpha \models_{\mathfrak{D}}$ information α differentiates (or α has the function of an informational differentiation);

$\models_{\mathfrak{D}} \alpha$ information α is differentiated;

$\alpha \models_{\mathfrak{D}} \beta$ information α differentiates information β or information β is differentiated by information α;

$\alpha, \beta, \ldots, \gamma \models_{\mathfrak{D}} \xi, \eta, \ldots, \zeta$

informational entities $\alpha, \beta, \ldots, \gamma$ differentiate informational entities $\xi, \eta, \ldots, \zeta$ or $\xi, \eta, \ldots, \zeta$ are differentiated by $\alpha, \beta, \ldots, \gamma$ etc.

This general case of informational differentiation ($\models_{\mathfrak{D}}$) can certainly be determined also for parallel, cyclical, and parallel-cyclical cases ($\|\models_{\mathfrak{D}}$, $\vdash_{\mathfrak{D}}$, and $\|\vdash_{\mathfrak{D}}$). ■

[Axioms]^{DF14}:

(1) $((\alpha \models) \vee (\models \alpha)) \Rightarrow ((\alpha \models_{\mathfrak{D}}) \vee (\models_{\mathfrak{D}} \alpha))$

(2) $((\dashv \alpha) \vee (\alpha \dashv)) \Rightarrow ((\dashv_{\mathfrak{D}} \alpha) \vee (\alpha \dashv_{\mathfrak{D}}))$

(3) $(\alpha \models \alpha) \Rightarrow (\alpha \models_{\mathfrak{D}} \alpha)$

(4) $(\alpha \models \beta) \Rightarrow (\alpha, \beta \models_{\mathfrak{D}} \alpha, \beta)$

etc. The last axiom can be constructed from a more general one, namely from,

$$(\alpha \models \beta) \Rightarrow (\alpha, \beta \models \alpha, \beta)$$

by the non-uniform substitution of the second operator, i.e., by its particularization. ■

Through informational differentiation of information also several differences can be determined. These differences can be marked by special symbols. For instance,

$$\delta_{\alpha,\beta,\ldots,\gamma}(\xi, \eta, \ldots, \zeta) \equiv ((\alpha, \beta, \ldots, \gamma \models \xi, \eta, \ldots, \zeta) \vee (\xi, \eta, \ldots, \zeta \dashv \alpha, \beta, \ldots, \gamma))$$

This formula shows the possibilities of conversion between implicit and explicit informational operators, where marking of a difference becomes equivalent to the result of an implicit operation. For instance,

$$\delta_{\alpha,\beta,\ldots,\gamma}(\xi, \eta, \ldots, \zeta) \equiv \mathfrak{D}_{\alpha,\beta,\ldots,\gamma}(\xi, \eta, \ldots, \zeta)$$

II.3.1.11. Axioms of Informational Integration

Integration of information is an inherent property of information which informs and is informed. By integration the incoming, arriving, and arising information is informationally integrated into source information or into information in question. Informational integration is a consequence of the appearing information which has to be integrated into an existing informational realm, otherwise it will be lost as informational noise. Similarly as in the case of differentiation, it is possible to introduce two basic and general operators of integration which govern the so-called explicit and implicit informational integration.

[Operators]^{DF46}:
One kind of the explicit informational operator of integration can be determined in the following way:

$$(\models_{\Im} \vee \dashv_{\Im}) \quad =_{Df} \quad ('integrates') \vee ('integrate') \vee ('is\_integrated(\_by)') \vee ('are\_integrated(\_by)')$$

The primitive implicit informational operator of integration can be determined as

$$\Im_{\alpha,\beta,\ldots,\gamma}(\xi, \eta, \ldots, \zeta)$$

with the following meaning: $\alpha, \beta, \ldots, \gamma$ integrate $\xi, \eta, \ldots, \zeta$ or $\xi, \eta, \ldots, \zeta$ are integrated by $\alpha, \beta, \ldots, \gamma$ into $\alpha, \beta, \ldots, \gamma$. Evidently, the operators $\models_{\Im}$, $\dashv_{\Im}$, and $\Im$ integrate the given information into the integrating information itself. At this point a clear difference between differentiation and integration comes into the foreground.

Certainly, the complete implicit informational operator of integration can be determined in the following way:

$$\Im_{[\lambda,\mu,\ldots,\nu]\alpha,\beta,\ldots,\gamma}(\xi, \eta, \ldots, \zeta)$$

The meaning of this implicit operator is as follows: informational entities $\alpha, \beta, \ldots, \gamma$ integrate informational entities $\xi, \eta, \ldots, \zeta$ into informational entities $\lambda, \mu, \ldots, \nu$. ■

[Operators]^{DF47}:
Now, we have to define an informational operator of location with the meaning "into", to enable the expression of this particular need, for instance, to be integrated "into" information. This operator has to be of explicit informational type, for difficulties of expressing the process concerning the "into" occur, for instance, by the use of the operator $\models_{\Im}$. We have:

$$\mathbb{1} \quad =_{Df} \quad ('into')$$

Further, we can introduce the left to the right and the opposite version of this operator by

$$\mathbb{1}_{\rightarrow} \quad \text{and} \quad \mathbb{1}_{\leftarrow}$$

respectively. The proposed operator is general and introduces substantial semantics into our further discourse. ■

[Operator]^{RX15}:
In the explicit case the informational operator of integration $\models_{\Im}$ the need has arisen to express into which informational entity information will be integrated. We have now the following possibility:

$$\alpha \models_{\Im} \beta \; \mathbb{1}_{\rightarrow} \; \gamma \quad \text{or} \quad \gamma \; \mathbb{1}_{\leftarrow} \; \beta \dashv_{\Im} \alpha$$

The meaning is the following: information α integrates information β in one or another way into information γ. By the operator $\mathbb{1}$ we can even capture the most subtle phenomenon of coming of information into existence. Thus, we can decompose the operator $\llcorner$ to some degree by splitting its meaning into "coming_of' ($\models_{come}$) and 'into'. We can introduce the following implication:

$$(\llcorner \alpha) \quad \Rightarrow \quad ((\models_{come} \alpha) \; \mathbb{1} \; \alpha) \qquad ■$$

At this point the question what is existence can arise. "Existence" has the meaning of information of existence or of *existing* information. Similarly, "coming" has the meaning of information which comes or of *coming* information. This kind of information is, for instance, counter-information or sensory information. Coming of information into existence is, for instance, embedding of the arisen counter-information into the existing information. Here, coming into existence concerns informational differentiation as well as informational integration. In other words, arising of information is nothing else but counter-Informing and embedding or differentiation and integration of information. These two types of Informing constitute the so-called informational cycle, which is the cycle of coming into existence: from the existing information arises the counter-information and is embedded again into the existing information, enlarging (or decreasing) its informational realm. This informational cycling is the fundamental process of any informational arising. Therefore, we can say that information informs (differentiates) and is informed (integrates) cyclically or, in a more general sense, circularly.

[Operators]^{RX16}:
Let us clear the meaning of the explicit informational operator of integration in composition with the informational operator 'into'. We can list the following examples:

$\alpha \models_{\Im}$ or $(\alpha \models_{\Im}) \; \mathbb{1} \; \alpha$

information α integrates or information α integrates into itself;

$\models_{\Im} \alpha$ or $(\models_{\Im} \alpha) \; \mathbb{1} \; \alpha$

information α is integrated or information α is integrated into itself;

$\alpha \models_{\Im} \beta$ or $(\alpha \models_{\Im} \beta) \; \mathbb{1} \; \gamma$

information α integrates information β or information α integrates information β into information γ; the last formula can also be read as information β is

integrated by information α into information γ;

$\alpha, \beta, \ldots, \gamma \models_{\mathfrak{I}} \xi, \eta, \ldots, \zeta$ or

$(\alpha, \beta, \ldots, \gamma \models_{\mathfrak{I}} \xi, \eta, \ldots, \zeta) \amalg_{\lambda, \mu, \ldots, \nu}$

informational entities $\alpha, \beta, \ldots, \gamma$ integrate informational entities $\xi, \eta, \ldots, \zeta$ or informational entities $\alpha, \beta, \ldots, \gamma$ integrate informational entities $\xi, \eta, \ldots, \zeta$ into informational entities $\lambda, \mu, \ldots, \nu$; the last formula can also be read as informational entities $\xi, \eta, \ldots, \zeta$ are integrated by informational entities $\alpha, \beta, \ldots, \gamma$ into informational entities $\lambda, \mu, \ldots, \nu$, etc.

These cases of informational integration, using the explicit operator $\models_{\mathfrak{I}}$, can certainly be determined also for parallel, cyclical, and parallel-cyclical cases ($\models_{\mathfrak{I}}$, $\vdash_{\mathfrak{I}}$, and $\dashv_{\mathfrak{I}}$). ■

[Axioms]^{DF15}:

(1) $((\alpha \models) \vee (\models \alpha)) \Rightarrow ((\alpha \models_{\mathfrak{I}}) \vee (\models_{\mathfrak{I}} \alpha))$

(2) $((\dashv \alpha) \vee (\alpha \dashv)) \Rightarrow ((\dashv_{\mathfrak{I}} \alpha) \vee (\alpha \dashv_{\mathfrak{I}}))$

(3) $(\alpha \models \alpha) \Rightarrow ((\alpha \models_{\mathfrak{I}} \alpha) \amalg \alpha))$

(4) $(\alpha \models \beta) \Rightarrow ((\alpha, \beta \models_{\mathfrak{I}} \alpha, \beta) \amalg \alpha, \beta))$

etc. The last axiom could be constructed from a more general one, namely, from

$$(\alpha \models \beta) \Rightarrow ((\alpha, \beta \models \alpha, \beta) \models \alpha, \beta))$$

by the non-uniform substitution of operators (e.g. particularization of the second operator on the left side of implication). ■

[Axioms]^{KX2}:

The axiom (4) in the last definition can be decomposed in details in the following way:

$$\begin{aligned}(\alpha \models \beta) \Rightarrow\ & (((\alpha \models_{\mathfrak{I}} \alpha) \amalg \alpha) \vee \\ & ((\alpha \models_{\mathfrak{I}} \beta) \amalg \alpha) \vee \\ & ((\beta \models_{\mathfrak{I}} \alpha) \amalg \alpha) \vee \\ & ((\beta \models_{\mathfrak{I}} \beta) \amalg \alpha) \vee \\ & ((\alpha \models_{\mathfrak{I}} \alpha) \amalg \beta) \vee \\ & ((\alpha \models_{\mathfrak{I}} \beta) \amalg \beta) \vee \\ & ((\beta \models_{\mathfrak{I}} \alpha) \amalg \beta) \vee \\ & ((\beta \models_{\mathfrak{I}} \beta) \amalg \beta))\end{aligned}$$

This is the well-known principle of the iwff decomposition. The so-called parallel decomposition of the last case would be as follows:

$$\begin{aligned}(\alpha \models \beta) \Rightarrow\ & ((\alpha \models_{\mathfrak{I}} \alpha) \amalg \alpha, (\alpha \models_{\mathfrak{I}} \beta) \amalg \alpha, \\ & (\beta \models_{\mathfrak{I}} \alpha) \amalg \alpha, (\beta \models_{\mathfrak{I}} \beta) \amalg \alpha, \\ & (\alpha \models_{\mathfrak{I}} \alpha) \amalg \beta, (\alpha \models_{\mathfrak{I}} \beta) \amalg \beta, \\ & (\beta \models_{\mathfrak{I}} \alpha) \amalg \beta, (\beta \models_{\mathfrak{I}} \beta) \amalg \beta)\end{aligned}$$

This example is in fact the axiom of the parallel decomposition of the case $\alpha \models \beta$. ■

II.3.12. Axioms of Informational Particularization and Universalization

In informational logic iwffs can be particularized and universalized. This principle permits various substitutions of explicit operators, enabling specialization (particularization) and generalization (universalization) of informational formulae. Processes of informational particularization and universalization are the basic, i.e. axiomatic properties of an iwff. These processes could be included as well into the domain of the so-called transformation rules, for through their application, formulae are transformed from original semantic domains into other special or general ones.

[Axioms]^{DF16}:

(1) ('α is iwff') $\Rightarrow$ ('$\mathfrak{P}(\alpha)$ is iwff')

(2) ('$\models_{con}$ is sub-iwff') $\Rightarrow$ ('$\mathfrak{P}(\models_{con})$ is sub-iwff')

(3) $(\alpha \models \beta) \Rightarrow (\alpha\ \mathfrak{P}(\models)\ \beta)$

(4) $(\beta \dashv \alpha) \Rightarrow (\beta\ \mathfrak{P}(\dashv)\ \alpha)$

(5) $\mathfrak{P}(\alpha \models \beta) \Rightarrow (\mathfrak{P}(\alpha)\ \mathfrak{P}(\models)\ \mathfrak{P}(\beta))$

(6) $\mathfrak{P}(\beta \dashv \alpha) \Rightarrow (\mathfrak{P}(\beta)\ \mathfrak{P}(\dashv)\ \mathfrak{P}(\alpha))$

etc. In fact, particularization is an implicit informational operation. Further, the symbol $\mathfrak{P}$ can be used for particularization ($\mathfrak{P}_{\downarrow}$) as well as for universalization ($\mathfrak{P}^{\uparrow}$) of formulae. Particularization is always a non-uniform operation in regard to substitution of operators. By particularization and universalization new semantics of operators and formulae is generated. Particularization and universalization belong to the most essential principles of informational arising. ■

II.3.13. Axioms of Informational Formula

We have already determined the so-called formation rules of iwffs. However, this rules do not ensure the constructibility of a formula which has to interpret a natural or an artificial information. We would like to know, at least hypothetically, if we do not need to take care about the nature of information which has to be formalized or put into the form of an iwff. Thus, the following questions may sound quite naturally: What kind of information can be put into the form of an informational formula? How can information be put into an adequate form of a formula? Is this form in case of information a unique or a multiplex one?

[Axioms]^{DF17}:

Let α denote an arbitrarily complex information. Let informational formula be denoted as an informationally well-formed formula (iwff). Then the following basic axiom is adopted, concerning the possibility of forming an informational formula from given information:

(∀α).(('α is information') ⇒
(∃('iwff').
('α can be put into the form of an iwff')))

This axiom says: for each α, which is information, irrespective of its complexity and informational nature, there exists at least one iwff such that (.) information can be put into the form of this iwff.

(∀α).(('α is information') ⇒
(∃('iwff').
('iwff is an adequate interpretation of α')))

This axiom says: for each α, which is information, irrespective of its complexity and informational nature, there exists at least one iwff such that this iwff is an adequate interpretation of information in question. We can understand that formal interpretation of a given information is never unique and that it depends on informational circumstances. In general, there exist (indefinitely) many interpretations of a given information.

(∀α).(('α is information') ⇒
(∃('iwff').
('iwff interprets α by an informational system of one or several sequences of informational operands and operators')))

This axiom assures the constructibility of the iwff which interprets adequately the given information α.

These three axioms can be certainly expressed in a much more symbolically compact form, for instance:

$$(\forall\alpha).(((\alpha \models) \vee (\models \alpha)) \Rightarrow ((\exists\varphi).((\alpha \models \varphi) \vee (\varphi \models_{ade} \alpha)) \vee (\exists\, \varphi_1,\varphi_2,\ldots,\varphi_n).(\varphi_1,\varphi_2,\ldots,\varphi_n \models \alpha)))$$

where φ symbolizes iwff, $\models_{ade}$ is informational operator of adequate interpretation, and each of the three conjuctive parts on the right side of implication concerns one of the three axioms. ■

II.3.14. Axioms of Informational Structure and Informational Organization

What are informational structure and informational organization and how do they reflect in informational axiomatization? To answer this question we have to consider the principle of informational structure and informational organization ([4] or [10]) as follows:

"Informational structure is a constitution of information, that is, a constitution of informational forms and informational processes that are composed as information. These forms and processes are informational components. The informational relations among informational components which determine a composite information constitute informational organization. In terms of informational epistemology, informational structure is closer to the form, whereas informational organization is closer to the process. Within information, informational forms and informational processes are informationally interwoven components. Informational components integrate information. Informational structure and informational organization are information by themselves."

What are forms and processes constituting a particular informational case? Speaking in the language of informational formulae, these forms and processes are informational operands and operators, where forms are, for instance, self-standing operands and processes are formally grouped (parenthesized) operands and operators. In this respect, informational structure appears as a more or less pure syntactic structure.

We suppose that given information always has a structure. This structure, which is observed as information concerning the structure of information in question, can always be interpreted through an iwff in a simple informational case or through an informational system of iwffs in a complex informational case. The structure of information is interpreted in iwffs by particular informational forms and processes, consisting of informational operands and operators. It is possible to list several axioms concerning the structure of information, for instance:

[Axioms]^{DF18}:

(1) (∀α).(('α is information') ⇒
((∃σ).('σ interprets the syntactic nature of information α')))

In this axiom, σ denotes information concerning the structure of information α.

(2) (∀α).(('α is structured information') ⇒
((∃φ).('α is interpreted by an adequately syntactically structured φ')))

In this and in the next axioms, φ marks the so-called iwff.

(3) (∀α).(('α is information') ⇒
((∃φ).('φ as information syntactically constitutes information α')))

(4) (∀α).(('α is information') ⇒
((∃φ).('φ interprets the syntactic structure of information α')))

etc. These axioms constitute the so-called structural hypothesis of information. This hypothesis, in fact, is the informational principle of structural constructibility of information and its adequate iwff. Here iwff is understood to be an informational system or any form of informationally connected iwffs. ■

The structure of information α is information which concerns the componential syntax of α.

This syntax is interpreted into the structure of iwff of α. Structural interpretation of α onto its iwff does not represent the sufficient condition for a completely adequate interpretation of α by its iwff. The second component, called informational organization of α, has to be considered when the iwff adequate to α is constructed.

While the structure of information predominantly concerns the so-called syntax or form of componential constitution of information, organization of information predominantly concerns the so-called semantics or componential processes, relations, and informational interweaving of informational processes. Among possible interweaving of informational forms and processes within information, the most important seems to be informational parallelism, which is the synonym for interweaving nature of informational forms and processes. It becomes evident that information, by its nature, is nothing else but extremely interwoven structure (topology, granularity) and organization (selectivity, relationship) of information.

$[\text{Axioms}]^{DF19}$:

(1) $(\forall\alpha).(('\alpha \text{ is information}') \Rightarrow ((\exists\omega).('\omega \text{ interprets the semantic nature of information } \alpha')))$

In this axiom, ω denotes information concerning the organization of information α.

(2) $(\forall\alpha).(('\alpha \text{ is organized information}') \Rightarrow ((\exists\varphi).(('\alpha \text{ is interpreted by an adequately semantically organized } \varphi') \vee ('\varphi \text{ semantically interprets } \alpha \text{ as informational organization'}))))$

In this and in the next axioms, φ marks the so-called iwff.

(3) $(\forall\alpha).(('\alpha \text{ is information}') \Rightarrow ((\exists\varphi).('\varphi \text{ as information semantically constitutes information } \alpha')))$

(4) $(\forall\alpha).(('\alpha \text{ is information}') \Rightarrow ((\exists\varphi).('\varphi \text{ interprets the semantic organization of information } \alpha')))$

etc. ■

$[\text{Axioms}]^{EX3}$:

The axioms $[\text{Axioms}]^{DF18}$ and $[\text{Axioms}]^{DF19}$ can be interpreted in a more symbolically compact and instructive manner. Let us construct the following implications:

(1) $('\alpha \text{ is information}') \Rightarrow ((\alpha \models) \vee (\models \alpha))$

(2) $('\sigma \text{ interprets the syntactic nature of information } \alpha') \Rightarrow (\sigma \models \mathfrak{S}(\alpha))$

Instead of the consequence in the last implication it could be

$\sigma \models_{int} \mathfrak{S}(\alpha)$ or, conventionally, $\sigma = \mathfrak{S}(\alpha)$

$\mathfrak{S}(\alpha)$ has the meaning of 'syntactical, i.e. structural nature of α', whereas the operator $\models_{int}$ has the meaning of Informing by interpretation.

(3) $('\alpha \text{ is structured information}') \Rightarrow ((\alpha \models \sigma) \wedge (\sigma \subset \alpha))$

If information α is structured, then it informs (gives, transmits) information σ of its structure ($\sigma \subset \alpha$).

(4) $('\alpha \text{ is interpreted by an adequately syntactically structured } \varphi') \Rightarrow ((\sigma \subset \alpha) \wedge (\sigma \subset \varphi) \wedge (\varphi \therefore_{syn} \alpha))$

The iwff φ informs structurally (syntactically) similar (analogous) to information α. The iwff φ informs structurally similar (by means of the operator $\therefore_{syn}$) to α.

(5) $('\varphi \text{ as information syntactically constitutes information } \alpha') \Rightarrow (((\sigma \subset \alpha) \Rightarrow (\sigma \subset \varphi)) \wedge (\exists(\varphi \models).(\varphi \models_{syn} \alpha)))$

The iwff φ in fact constitutes also the structure of information α. The operator of this syntactic constitution is $\models_{syn}$. There exists such Informing of φ that φ syntactically informs α.

(6) $('\varphi \text{ interprets the syntactic structure of information } \alpha') \Rightarrow ((\sigma \subset \alpha) \Rightarrow (\varphi \models_{syn} \alpha))$

(7) $('\omega \text{ interprets the semantic nature of information } \alpha') \Rightarrow (\omega \models \mathfrak{A}(\alpha))$

Instead of the consequence in the last implication it could be

$\omega \models_{int} \mathfrak{A}(\alpha)$ or, conventionally, $\omega = \mathfrak{A}(\alpha)$

$\mathfrak{A}(\alpha)$ has the meaning of "semantic, i.e. organizational nature of α".

(8) $('\alpha \text{ is organized information}') \Rightarrow ((\alpha \models \omega) \wedge (\omega \subset \alpha))$

If information α is organized, then it informs (gives, transmits) information ω of its organization ($\omega \subset \alpha$).

(9) $((('\alpha \text{ is interpreted by an adequately semantically organized } \varphi') \vee ('\varphi \text{ semantically interprets } \alpha \text{ as informational organization'})) \Rightarrow ((\omega \subset \alpha) \wedge (\omega \subset \varphi) \wedge (\varphi \therefore_{sem} \alpha))$

The iwff φ informs organizationally (semantically) similar (analogous) to information α. The iwff φ informs organizationally similar (by means of the

operator $\therefore_{sem}$) to α.

(10) ('φ as information semantically constitutes information α') ⇒
$(((\omega \subset \alpha) \Rightarrow (\omega \subset \varphi)) \wedge (\exists(\varphi \models).(\varphi \models_{sem} \alpha)))$

The iwff φ in fact constitutes also the organization of information α. The operator of this syntactic constitution is $\models_{sem}$. There exists such Informing of φ that φ semantically informs α.

(11) ('φ interprets the semantic organization of information α') ⇒
$((\omega \subset \alpha) \Rightarrow (\varphi \models_{sem} \alpha))$

Considering implications (1)-(11), $[Axioms]^{DF18}$ and $[Axioms]^{DF19}$ can be rewritten in the following manner:

(1) $(\forall\alpha).(((\alpha \models) \vee (\models \alpha)) \Rightarrow ((\exists\sigma).(\sigma \models \mathfrak{S}(\alpha))))$

(2) $(\forall\alpha).(((\alpha \models \sigma) \wedge (\sigma \subset \alpha)) \Rightarrow ((\exists\varphi).((\sigma \subset \alpha) \wedge (\sigma \subset \varphi) \wedge (\varphi \therefore_{syn} \alpha))))$

(3) $(\forall\alpha).(((\alpha \models) \vee (\models \alpha)) \Rightarrow ((\exists\varphi).(((\sigma \subset \alpha) \Rightarrow (\sigma \subset \varphi)) \wedge (\exists(\varphi \models).(\varphi \models_{syn} \alpha))))$

(4) $(\forall\alpha).(((\alpha \models) \vee (\models \alpha)) \Rightarrow ((\exists\varphi).((\sigma \subset \alpha) \Rightarrow (\varphi \models_{syn} \alpha))))$

etc. Further, for $[Axioms]^{DF19}$ there is:

(1) $(\forall\alpha).(((\alpha \models) \vee (\models \alpha)) \Rightarrow ((\exists\omega).(\omega \models \mathfrak{A}(\alpha))))$

(2) $(\forall\alpha).(((\alpha \models \omega) \wedge (\omega \subset \alpha)) \Rightarrow ((\exists\varphi).((\omega \subset \alpha) \wedge (\omega \subset \varphi) \wedge (\varphi \therefore_{sem} \alpha)))))$

(3) $(\forall\alpha).(((\alpha \models) \vee (\models \alpha)) \Rightarrow ((\exists\varphi).(((\omega \subset \alpha) \Rightarrow (\omega \subset \varphi)) \wedge (\exists(\varphi \models).(\varphi \models_{sem} \alpha)))))$

(4) $(\forall\alpha).(((\alpha \models) \vee (\models \alpha)) \Rightarrow ((\exists\varphi).(((\omega \subset \alpha) \Rightarrow (\omega \subset \varphi)) \wedge (\exists(\varphi \models).(\varphi \models_{sem} \alpha)))))$

etc. ■

The axioms $[Axioms]^{DF18}$ and $[Axioms]^{DF19}$ assure the existence of an adequate (informationally complete) interpretation of any information α onto its iwff φ in the sense of informational structure σ and informational organization ω. This leads to the fundamentally important axioms of constructibility of iwffs for arbitrarily occurring informational cases.

$[Axioms]^{DF20}$:
The axioms which follow govern the interpretation of information α onto the structure σ and organization ω of an iwff (or of a system of iwffs) φ, which models α in the informationally complete way. The process of constructing iwff from given information can be expressed in the following manner:

$$\alpha \models \sigma, \quad \alpha \models \omega, \quad \sigma, \omega \models \varphi$$

The consequence of this system is $\alpha \models \varphi$. The construction of iwff φ from α is a parallel informational system which assures the so-called formalization of information α onto iwff φ. Thus, the last system can be particularized in the form

$$\alpha \models \sigma, \quad \alpha \models \omega, \quad \sigma, \omega \models \varphi$$

The consequence of this system is $\alpha \models \varphi$. Further particularization is possible:

$$\alpha \models_{syn} \sigma, \quad \alpha \models_{sem} \omega, \quad \sigma, \omega \models_{form} \varphi$$

The consequence of this system is $\alpha \models_{form} \varphi$. ■

II.3.15. Axioms of Informational Parallelism

Information is a parallel informational phenomenon in itself as well as in its interaction with other or outside information. It means that its forms and its processes appear, inform, change, vanish, etc. in a parallel manner. In this phenomenology, parallelism can be understood not only topologically and temporally, but also symbolically, abstractly, expressively. The basic question might be how information is performing parallel in itself. Why is informational interaction in fact always a parallel Informing? Thus, these conclusions (or beliefs) can be axiomatically framed in the following axioms:

$[Axioms]^{DF21}$:

(1) ('α is information') ⇒
$(((\alpha \models) \vee (\models \alpha)) \vee ((\nmid \alpha) \vee (\alpha \nmid)))$

If α is information, then it informs and is informed in parallel in one or another way. This fact can be expressed also in the form of parallel informational system, i.e.,

('α is information') ⇒
$(\alpha \models, \models \alpha, \nmid \alpha, \alpha \nmid)$

(2) $(\forall\alpha).(('\alpha \text{ is information}') \Rightarrow (\exists(\mathfrak{C}_\alpha, \mathfrak{E}_\alpha).(\mathfrak{C}_\alpha \models, \models \mathfrak{C}_\alpha, \mathfrak{E}_\alpha \models, \models \mathfrak{E}_\alpha)))$

If α is information, then there exist counter-Informing caused by α, $\mathfrak{C}_\alpha$, and informational embedding of α, $\mathfrak{E}_\alpha$, which inform in parallel. This Informing of $\mathfrak{C}_\alpha$ and $\mathfrak{E}_\alpha$ is an immanent property of parallelism of information. As we have already recognized, counter-Informing and embedding of counter-information constitute the so-called basic informational cycle (informational cyclicity). It also follows from the last axiom that counter-Informing and informational embedding within information α perform as information.

(3) $(\forall\alpha).(('\alpha \text{ is information}') \Rightarrow (\exists(\mathfrak{C}_\alpha, \mathfrak{E}_\alpha).(\alpha \models \mathfrak{C}_\alpha(\alpha), \alpha, \mathfrak{C}_\alpha(\alpha) \models \mathfrak{E}_\alpha(\mathfrak{C}_\alpha(\alpha)))))$

where $\mathfrak{C}_\alpha(\alpha)$ is in fact counter-information ω produced by counter-Informing $\mathfrak{C}_\alpha$ and $\mathfrak{E}_\alpha(\mathfrak{C}_\alpha(\alpha))$ is the embedding of the produced counter-information ω into α.

(4) The most complex informational system of inward informational parallelism can be axiomatized by the following iwff:

$$(\forall\alpha).((\text{'}\alpha \text{ is information'}) \Rightarrow ((\alpha, \Im_\alpha, \mathfrak{C}_\alpha, \omega, \mathfrak{E}_\alpha \models \alpha, \Im_\alpha, \mathfrak{C}_\alpha, \omega, \mathfrak{E}_\alpha) \vee (\alpha, \Im_\alpha, \mathfrak{C}_\alpha, \omega, \mathfrak{E}_\alpha \nmid \alpha, \Im_\alpha, \mathfrak{C}_\alpha, \omega, \mathfrak{E}_\alpha)))$$

The parallel decomposition of the first disjunctive iwff part on the right side of implication is

$$\begin{array}{lllll} \alpha \models \alpha, & \alpha \models \Im_\alpha, & \alpha \models \mathfrak{C}_\alpha, & \alpha \models \omega, & \alpha \models \mathfrak{E}_\alpha, \\ \Im_\alpha \models \alpha, & \Im_\alpha \models \Im_\alpha, & \Im_\alpha \models \mathfrak{C}_\alpha, & \Im_\alpha \models \omega, & \Im_\alpha \models \mathfrak{E}_\alpha, \\ \mathfrak{C}_\alpha \models \alpha, & \mathfrak{C}_\alpha \models \Im_\alpha, & \mathfrak{C}_\alpha \models \mathfrak{C}_\alpha, & \mathfrak{C}_\alpha \models \omega, & \mathfrak{C}_\alpha \models \mathfrak{E}_\alpha, \\ \omega \models \alpha, & \omega \models \Im_\alpha, & \omega \models \mathfrak{C}_\alpha, & \omega \models \omega, & \omega \models \mathfrak{E}_\alpha, \\ \mathfrak{E}_\alpha \models \alpha, & \mathfrak{E}_\alpha \models \Im_\alpha, & \mathfrak{E}_\alpha \models \mathfrak{C}_\alpha, & \mathfrak{E}_\alpha \models \omega, & \mathfrak{E}_\alpha \models \mathfrak{E}_\alpha \end{array}$$

(5) $(\alpha \models \beta) \Rightarrow (\alpha, \beta \models \alpha, \beta)$

etc. ■

II.3.16. Axioms of Informational Cyclicity

Information is a cyclic informational phenomenon in itself as well as in its interaction with other or outward information. It means that informational forms and informational processes appear, inform, change, vanish, etc. in a cyclic manner. Cyclicity of information can be viewed to be purely serial, parallel, or serial-parallel phenomenon. The last case seems to be the most obvious one. Within this phenomenology, cyclicity can be understood not only topologically and temporally, but also symbolically, abstractly, expressively. The basic question is how information performs cyclically in itself. Why informational interaction is in fact always a cyclic Informing? Let us frame these observations axiomatically in the following manner:

[Axioms]^{DF22}:

(1) $(\text{'}\alpha \text{ is information'}) \Rightarrow (((\alpha \vdash) \vee (\vdash \alpha)) \vee ((\alpha \models) \vee (\models \alpha)) \vee ((\dashv \alpha) \vee (\alpha \dashv)) \vee ((\nmid \alpha) \vee (\alpha \nmid)))$

If α is information, then it informs and is informed cyclically and parallel-cyclically in such or another way. This fact can be expressed also in the form of parallel-cyclical informational system, i.e., in a particular form:

$$(\text{'}\alpha \text{ is information'}) \Rightarrow (\dashv \alpha, \alpha \dashv, \nmid \alpha, \alpha \nmid)$$

(2) $(\forall\alpha).((\text{'}\alpha \text{ is information'}) \Rightarrow ((\alpha \models \mathfrak{C}_\alpha) \wedge (\alpha, \mathfrak{C}_\alpha \models \omega) \wedge (\alpha, \mathfrak{C}_\alpha, \omega \models \mathfrak{E}_\alpha) \wedge (\alpha, \mathfrak{C}_\alpha, \omega, \mathfrak{E}_\alpha \models \alpha)))$

This axiom determines the so-called inward informational cycle. The formula can be universalized in the following manner:

(3) $(\forall\alpha).((\text{'}\alpha \text{ is information'}) \Rightarrow ((((\alpha \vdash \mathfrak{C}_\alpha) \vdash (\alpha, \mathfrak{C}_\alpha \vdash \omega)) \vdash (\alpha, \mathfrak{C}_\alpha, \omega \vdash \mathfrak{E}_\alpha)) \vdash (\alpha, \mathfrak{C}_\alpha, \omega, \mathfrak{E}_\alpha \vdash \alpha)))$

In this formula it is possible to observe distinct cycles, i.e., also cycles within cycles, where for the right side of implication there is

$$(((\text{cycle}_1 \vdash \text{cycle}_2) \vdash \text{cycle}_3) \vdash \text{cycle}_4)$$

In this expression there are three more cycles, namely cycle_5 between cycle_1 and cycle_2, cycle_6 between cycle_5 and cycle_3, and cycle_7 between cycle_6 and cycle_4. All these cycles can be understood as cyclically parallel, thus:

(4) $(\forall\alpha).((\text{'}\alpha \text{ is information'}) \Rightarrow (\alpha \models \mathfrak{C}_\alpha, (\alpha, \mathfrak{C}_\alpha \models \omega), (\alpha, \mathfrak{C}_\alpha, \omega \models \mathfrak{E}_\alpha), (\alpha, \mathfrak{C}_\alpha, \omega, \mathfrak{E}_\alpha \models \alpha))$

This cyclic parallelism can be captured in the most complex form by the iwff

(5) $(\forall\alpha).((\text{'}\alpha \text{ is information'}) \Rightarrow (\alpha, \mathfrak{C}_\alpha, \omega, \mathfrak{E}_\alpha \models \alpha, \mathfrak{C}_\alpha, \omega, \mathfrak{E}_\alpha))$

(6) To explain the nature of informational cycle, the following auxiliary axioms can be adopted:

$$(\mathfrak{C}_\alpha \models \alpha) \nmid \omega \quad \text{and} \quad (\mathfrak{E}_\alpha \models \omega) \nmid \alpha$$

with the meaning $\omega = \mathfrak{C}_\alpha(\alpha)$ and $\mathfrak{E}_\alpha(\omega) \subset \alpha$, respectively.

Obviously, axiomatization of informational cyclicity can be continued indefinitely. ■

II.3.17. Openness of Informational Axiomatization

But taking the methodologies as an end in themselves is ultimately limiting in the same sense as the analytic tendency to take the arguments as an end in themselves.

Terry Winograd [12] 255

The axioms determined show the possibilities of their indefinite axiomatic continuation. Beside the already existing axiomatic cases new axiomatic interpretations are possible which concern an axiomatic type. In a similar way it is possible to add new axiomatic types to the existing ones. The consequence of these possibilities is that an axiomatic system remains open for new axiomatic determinations. Finally, it is possible to conclude that informational axiomatization irrespective of the informational system involved remains open in the described sense. To clear this informational phenomenon to some extent, we can put several principled questions concerning the

structure, organization, parallelism, etc. of information.

The axiomatic basis of informational logic remains open. Principles of informational particularization and universalization contribute to an additional and constructively senseful component of keeping the axiomatic basis open. In fact, informational logic in its axiomatic nature performs as regular information. Thus, the exposed axiomatization in this essay is informational.

II.3.18. Informational Axioms and Metaphysical Beliefs

We want to expand our ability as observers, within a context in which we are not detached but are engaged in the practices we ourselves observe.

Terry Winograd [12] 255

It cannot be disputable that the listed informational axioms arise from a particular metaphysical disposition from which they are thrown into a broader professional, scientific, and certainly also philosophical discourse. Whichever theory comes into existence, it begins its march as a scientific or philosophical literature and in fact represents nothing more than an authorial telling of a story. This storytelling, which concerns informational axioms and processes of axiomatization of diverse informational principles, grounds in epoch-making beliefs, i.e. in the metaphysical background constituting the philosophy of the so-called information era. Again, metaphysics has to be understood as a totality of information spontaneously arising in a living being and within its population.

The awareness that axiomatization of informational principles grounds in metaphysical beliefs lets the processes of axiomatization be generative, indefinitely predictable, and open for further development. Such kind of axiomatization certainly does not fit properly into the hardly predestined realms of traditional and emphasized rationalistic science. Does the time come when new, non-traditional, and also non-rationalistic approach in exact sciences is becoming an evident advantage in the research of unrevealed possibilities?

II.3.19. Some Axiomatic Consequences of Informational Arising

At the end of section II.3, in which we have discussed informational axioms, it seems necessary to stress again the arising (or at least variable) nature of informational operands, operators, and formulae. Such as they are, all of the listed axioms in this essay concern the arising principle of occurring informational entities. Thus, this informational nature is found not only in the semantics of explicit informational operators but also in implicit informational operators and informational operands. By themselves, axioms are arising structures of informational formulae. In this respect the axiomatic consequences of informational arising can find their continuation in any construction of iwff.

References of IL I, II, and III

[1] H. L. Dreyfus and S. E. Dreyfus: *Mind over Machine.* The Free Press, Macmillan, New York 1986.

[2] A. P. Zeleznikar: *On the Way to Information.* Informatica 11 (1987), No. 1, 4-11.

[3] A. P. Zeleznikar: *Information Determinations I.* Informatica 11 (1987), No. 2, 3-17.

[4] A. P. Zeleznikar: *Principles of Information.* Informatica 11 (1987), No. 3, 9-17. [Published also in Cybernetica 31 (1988), 99-122.]

[5] A. P. Zeleznikar: *Information Determinations II.* Informatica 11 (1987), No. 4, 8-25.

[6] A. P. Zeleznikar: *Problems of Rational Understanding of Information.* Informatica 12 (1988), No. 2, 31-46.

[7] L. A. Steen (Editor): *Mathematics Today.* Twelve Informal Essays. Springer-Verlag. New York (Third Printing, 1984).

[8] E. de Bono: *The Mechanisms of Mind.* Penguin Books, Harmondsworth, Middlesex, England (Reprinted 1977).

[9] G.W.F. Hegel: *The Science of Logic.* In Hegel's Logic, being Part One of the Philosophical Sciences (1830), translated by W. Wallace. Oxford, At the Clarendon Press, reprinted 1985.

[10] A.P. Zeleznikar: *Principles of Information.* Cybernetica 31 (1988) 99-122.

[11] S. Schiffer: *Symposium on Remnants of Meaning: 1. Overview of the Book.* Mind and Language 3 (1988), No. 1, 1-8 (Basil Blackwell).

[12] T. Winograd: *On Understanding Computers and Cognition: A New Foundation for Design (A response to the reviews).* Artificial Intelligence 31 (1987) 250-261.

Borut Jereb,* Ljubo Pipan,** Aleš Klofutar
Institut J. Stefan
* Gorenje raziskave in razvoj, T. Velenje
** Fakulteta za elektrotehniko, Ljubljna

Descriptors: TRANSPUTERS, PARALLEL PROCESSING

Za doseganje večje zmogljivosti računalnikov, je znanih veliko teoretičnih pristopov. Nekatere od njih je možno realisirati, nekatere delno, nekateri pristopi pa so s sedaj poznanimi tehnologijami nerealni. Pri transputerjih je načrtovalcem uspelo realisirati nekaj zanimivih zamisli. Članek je pregleden in obravnava predvsem najnovejši transputer T800, ki je relativno cenen in zmogljiv gradnik v eno ali večprocesorskem sistemu. Pri večprocesorskem sistemu so lahko transputerji močno ali šibko povezani. Model T800 ima obenem veliko numerično moč. Ta moč je posledica nekaterih izvirnih rešitev, ki so opisane v članku.

TRANSPUTERS

Several theoretical principles for the achievement of greater computer capability already exist. Some of these can be carried out, some are only partially aplicable, while many theories can not yet be realised, given today's technology. Since experts have succesed in discovering several good and applicable ideas and possibilities in transputer designing, this paper is meant to give the reader a lucid survey of transputers, with special emphasis placed on the newest model: T800. This is a relatively cheap and capable unit, designed for use in both single or multiprocessor systems. In the latter, the transputers can be either tightly or loosely linked and the T800 model has, at the same time, great numerical power - a result of the several original solutions presented further on in this paper.

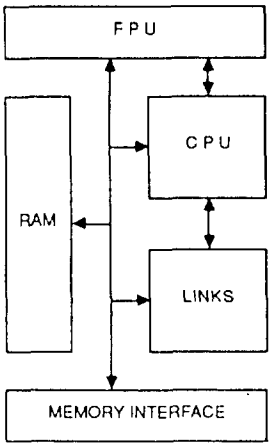
0 UVOD

Članek opisuje družino transputerjev. To so izdelki podjetja Inmos, ki se od klasičnih mikroprocesorjev razlikujejo po prilagojenosti paralelnemu procesiranju.

Prva štiri poglavja podajajo splošen opis družine transputerjev. Peto poglavje na kratko opisuje nekatere zmožnosti transputerja IMS T800 (predvsem enoto za delo s števili v plavajoči vejici). Sledi še zaključek.

1 KONCEPT IN ARHITEKTURA TRANSPUTERJA

VLSI tehnologija omogoča ceneno izdelavo velikega števila enakih integriranih vezij velike zmogljivosti. Z uporabo več enakih integriranih vezij lahko realiziramo sistem z neko mero paralelizma oz. sočasnosti.



slika 1
Bločni diagram transputerja IMS T800

Transputer je VLSI integrirano vezje, ki vsebuje: procesor (CPU), pomnilnik (RAM), komunikacijske kanale (LINKS) za direktno povezavo z ostalimi transputerji v tako imenovani mreži transputerjev in vmesnik za zunanji pomnilnik (MEMORY INTERFACE); nekateri, odvisno od tipa, tudi enoto za delo s števili v plavajoči vejici (FPU). (slika 1).

Načrtovalcem tega vezja je uspelo napraviti vezje, ki je dobro prilagojeno paralelnemu procesiranju. Dodatek k paralelizmu realiziranem z večimi transputerji je velika mera notranjega paralelizma v samem transputerju. Samostojen sistem lahko predstavlja že en sam transputer ali pa več v mrežo povezanih transputerjev. Kot primer slika 1 podaja bločni diagram transputerja IMS T800.

1.1 PROCESOR IN POMNILNIK NA SKUPNEM VEZJU

V sistemih sestavljenih iz več VLSI vezij (procesor, pomnilnik itd), je hitrost prenosa podatkov med vezji, glede na hitrost delovanja samih vezij, zelo majhna. Obenem pa vsaka operacija, ki jo izvede procesor, zahteva uporabo pomnilnika. Zaradi slednjega sta pri transputerjih procesor in pomnilnik sestavna dela enega samega vezja. Tipična velikost pomnilnika na vezju je pri sedanjih transputerjih nekaj Kslogov.

1.2 KOMUNIKACIJE MED TRANSPUTERJI

Povezave med vezji in dodatna vezja za upravljanje povezav pomenijo glavno omejitev pri zmanševanju velikosti celotnega sistema sestavljenega iz integriranih vezij. Iz omenjenega sledi, da želimo zmanjšati število povezav med integriranimi vezji in obenem to povezavo čim bolj poenostaviti oz. uporabiti čim manj dodatnih vezij za upravljanje povezav. Transputer lahko povežemo z ostalimi transputerji s serijskimi enosmernimi "point-to-point" povezavami. Pri tem ne potrebujemo nikakršnih dodatnih vezij.

S tem zmanjšamo število povezav med integriranimi vezji in zelo poenostavimo upravljanje komunikacij.

1.3 ENOSTAVEN PROCESOR Z MIKROKODIRANIM RAZPOREJEVALCEM OPRAVIL

Transputer vsebuje enostaven sekvenčni procesor z majhnim številom instrukcij. Dodani sta specializirani skupini instrukcij za aritmetične operacije v plavajoči vejici in za razporejanje opravil.

Proces, ki se izvaja v transputerju lahko, vsebuje večje število sočasnih procesov, katerih sočasnost transputer podpira interno. To je omogočeno z mikrokodiranim razporejevalcem opravil, ki odmerja procesorjev čas sočasnim procesom. Razporejevalec omogoča dva prioritetna nivoja.

1.4 TIPI TRANSPUTERJEV

IMS T414 je 32 bitni procesor s pomnilnikom dveh Kilogov na vezju. Ima 32 bitno povezavo s zunanjim pomnilnikom in štiri kanale za povezavo z ostalimi transputerji. Hitrejša verzija tega izdelka IMS T414-20 omogoča tipično 10 MIPS.

IMS T212 je zelo podoben pravkar opisanemu tipu. Razlika je v širini podatkovnih poti. Povezava z zunanjim pomnilnikom je 16 bitna.

Obstajajo še procesorji serije T212 za kontrolo diskovnih in disketnih pogonov in procesorji, ki omogočajo priključevanje transputerskih sistemov na običajne sisteme preko vodila.

IMS T800 je najnovejši izdelek Inmosa. To je 32 bitni procesor. Na vezju so štirje Kilogi pomnilnika in enota za računanje s števili, katerih zapis je po ANSI/IEEE Standardu [6]. Izvedba IMS T800-30 omogoča 2.5 MFLOPS. Pri tem vezju je ohranjena kompatibilnost z vezjem T414-20 na nivoju nožic vezja[1].

2 PROGRAMIRANJE

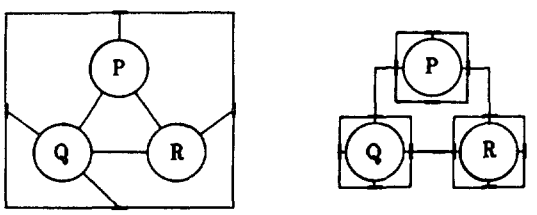
Če je osnovni gradnik paralelnega sistema transputer, potem nam proces predstavlja osnovni programski gradnik. Sistem transputerjev lahko načrtujemo in programiramo z jezikom, ki ga ponuja proizvajalec transputerjev - to je z jezikom Occam. Zaenkrat je poleg Occama na voljo še nekaj visokonivojskih jezikov. To so C, Fortran, Pascal in Fifth (podoben programskemu jeziku Forth) [1,6,8,9,10].

Program pisan v enem od visokonivojskih jezikov in nato preveden predstavlja modul. Z occam modulom nato povezujemo in konfiguriramo module neodvisno od originalnega jezika, v katerem je bil nek modul napisan, pred prevajanjem v kodo.

Pri izvajanju programa lahko pride do nekaterih usodnih napak (aritmetične prekoračitve, prekoračitve pri indeksiranju polj, deljenje z nič...), ki ustavijo procesor. Ko se pojavi takšna napaka, se postavi zastavica za napako in napaka se lahko obdela interno z programsko opremo ali pa se obdela zunaj z primerno strojno opremo (npr sosednji transputer v mreži transputerjev). V slednjem primeru zunanja logika zazna posebno stanje, v katerem se nahaja procesor na posebni nožici integriranega vezja.

2.1 OSNOVNI KONCEPT PROGRAMSKEGA JEZIKA OCCAM

Programski jezik Occam so razvili za programiranje sočasnih distribuiranih sistemov. Poudarek je na besedi distribuirani, saj dosedanji jeziki praviloma niso podpirali distribuiranih sistemov in temu ekvivalentnega razmišljanja.



Program na enem transputerju Program na treh transputerjih

slika 2
Realizacija procesov pri transputerjih

Razvoj jezika Occam je potekal sočasno z razvojem transputerja in ga nekateri jemljejo kot zbirni jezik za transputerje. Ima zelo malo (32 besed) rezerviranih besed, kar daje slutiti veliko prilagojenost jezika sami arhitekturi transputerja.

Izvajanje procesa v Occamu, je formalno ekvivalentno izvajanju programa na transputerju. Tako postane proces materialni in programski gradnik večprocesorskega sistema. Sočasni procesi so realizirani na mreži transputerjev; komunikacija med njimi in z zunanjimi napravami pa je realizirana s kanali. Konfiguracijo programa na mreži omogoča konstrukt PLACE PAR (pojem konstrukta je razložen kasneje). Occamov program je možno izvajati tudi na enem procesorju, ki deli čas med 'sočasne' procese. Glej sliko 2. Pri tem konfiguracija in število transputerjev nimata nikakršnega vpliva na logično obnašanje programa.

Sintaksa Occama uporablja zamikanje od levega roba za nazačevanje programske strukture. Vsak proces in vsak konstrukt je predstavljen z vrstico v programu.

2.1.1 PROCESI

Po zagonu procesa le ta izvaja akcije in se nato ustavi ali pa je ustavljen. Program pisan v Occamu je sestavljen iz treh osnovnih procesov. Ti so: prireditveni, vhodni in izhodni proces. Njihov zapis je naslednji:

v := e	prireditev izraza e spremenljivki v
c ! e	izhod izraza e v kanal c
c ? v	vhod spremenljivke v iz kanala c

Vsak Occamov kanal omogoča komunikacijsko pot med dvema sočasnima procesoma. Komunikacija je sinhronizirana in se izvede, ko sta oba - to je oddajajoči in sprejemajoči proces - pripravljena. Po končanem prenosu podatkov se oba procesa nadaljujeta.

2.1.2 KONSTRUKTI

Osnovne procese kombiniramo v konstrukte. Konstrukt je sam zase proces in ga lahko uporabimo kot del naslednjega konstrukta. Osnovna značilnost konstruktov je, da se začnejo s karakteristično oznaćbo, ki ji v naslednji vrstici - z zamikom glede na zgornjo - sledi lista osnovnih procesov ali/in konstruktov. Osnovni konstrukti so:

SEQuential	procesi se izvajajo en za drugim
PARallel	procesi se izvajajo sočasno
IF	proces se izvede ob izpolnjenosti pogoja
ALTernative	prvi pripravljen proces se izvede

Pri sekvenčnem izvajanju, se komponente procesov izvajajo ena za drugo. Sekvenčni konstrukt se konča s koncem izvajanja zadnje komponente konstrukta.

Komponente paralelnega konstrukta se izvajajo sočasno. Vsaka komponenta procesa operira na svojih spremenljivkah in komunicira z ostalimi sočasno delujočimi procesi preko kanalov. Paralelni konstrukt se konča le, če so se končale vse komponente konstrukta.

Tako je naslednji program sestavljen iz dveh paralelnih procesov. Prvi proces bo sprejel spremenljivko next.problem iz kanala source. Drugi proces pa je sestavljen iz dveh procesov, ki se bosta izvedla zaporedno: prvi računa computr.next.solution, drugi pa po zaključku prvega pošlje solution v kanal result.

```
PAR
    source ? next.problem
    SEQ
        computr.next.solution (this.problem, solution)
        result ! solution
```

Klasične sekvenčne programe lahko v Occamu napišemo tako, da uporabimo spremenljivke in prireditve v sekvenčnih konstruktih.

Pri pogojnem konstruktu se testirajo komponente zaporedno. če je komponenta pravilna se izvrši odgovarjujoči proces. Vedno se izvrši le en proces. Naslednji primer prikazuje uporabo konstrukta IF pri primerjavi števil a in b.

```
IF
    a > b
        order := gt
    a < b
        order := lt
    TRUE
        order := eq
```

Alternativni konstrukt omogoča večim procesom hkratno pripravljenost za sprejem podatkov iz kateregakoli od možnih kanalov. Sprejem se bo izvršil najprej iz tistega kanala, ki ga prvega uporabi nek drug proces za izhod. Pri tem imamo možnost dela z zunanjimi in notranjimi dogodki. V klasičnih mikroprocesorjih se takšne stvari rešujejo na nivoju sbirnika s prekinitvami. Za primer si oglejmo primer, kjer čakamo na signal na kanalu count in total. Če pride signal na kanal count povečamo spremenljivko counter za ena, v drugem primeru pa pošljemo skozi kanal out vrednost counter, ki jo nato še postavimo na nič.

```
ALT
    count ? signal
        counter := counter + 1
    total ? signal
        SEQ
            out ! counter
            counter := 0
```

Tudi ponavljanje je uporabljeno kot konstrukt. V spodnjem primeru se proces P izvršuje dokler ni pogoj condition napačen (false). Primer:

```
WHILE condition
    proces
```

Uporaba konstrukta kopiranja je razvidna iz naslednjega primera:

```
SEQ i = base FOR count
    a[i] := i
```

Slednje je ekvivalentno zapisu:

```
SEQ
    a[base]              := base
    a[base + 1]          := base + 1
    .....
    a[base + count - 1]  := base + count - 1
```

V običajnih jezikih uporabljamo za takšno nalogo ukaz FOR.

2.1.3 TIPI

Sedanja verzija Occama podpira več podatkovnih tipov, kakor tudi večdimensionalna polja. (Tipi:CHAN OF type (kanal tipa ...), TIMER, BOLL, BYTE, INT, INT16, INT32, INT64, REAL32, REAL64, [n,m,...] type). Polja se lahko prirejajo, prenašajo med procesi in uporabljajo kot parametri v procedurah. Occam obenem omogoča, da del polja obravnavamo kot polje. Za primer si oglejmo naslednji program, ki deklarira celoštevilčno polje desetih elementov s imenom a. Vrednosti elementov zajema paralelno iz kanalov c in d (prvih pet elementov iz kanala c, drugih pet pa iz kanala d).

```
[10] INT a
PAR
    c ? [a FROM 0 FOR 5]
    d ? [a FROM 5 FOR 5]
```

2.1.4 PROCEDURE, IZRAZI, ČASOVNIK IN ZUNANJE ENOTE

Procedura je proces, ki mu lahko damo ime. Npr:

```
PROC square (INT n)
    sqrt := n*n
```

Izrazi so sestavljeni iz operatorjev, ki jih najdemo v tabeli 2, iz spremenljivk, števil, logičnih izrazov ter predklepaja in zaklepaja.

+, -, *, /, REM	integer, real
PLUS, MINUS, TIMES, AFTER	integer
=, <>	enostaven iz.
>, <, ≤, ≥	integer, real
AND, OR, NOT	boolean
∧ (bitwise and), ∨ (bitwise or)	
>< (bitwise xor), ~ (bitwise not)	integers
<<, >> (premikanje)	integer

Tabela 1
Operatorji v Occamu

Vsak proces lahko ima svoj neodvisen časovnik, ki ga uporabi za svoje meritve ali za razdeljevanje dela v realnem času. Časovnik prebere vrednost v spremenljivko tipa INT. Npr:

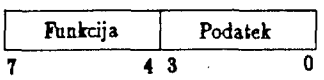
```
tim ? v
```

postavi spremenljivko v na trenutno vrednost prostotekoče ure, ki je deklarirana kot časovnik tim.

Dostop do zunanjih enot je v Occamu omogočena s mehanizmom vhodno/izhodnih vrat. Vrata se uporabljajo podobno kot kanali. Podobno lahko le en proces bere iz v/i vrat in le eden daje na v/i vrata.

3 KODIRANJE INSTRUKCIJ

Vsi transputerji imajo enak osnovni nabor maloštevilčnih instrukcij. Vsaka instrukcija vsebuje osem bitov, ki so razdeljeni v dve skupini po štiri bite. (Glej sliko 3.)



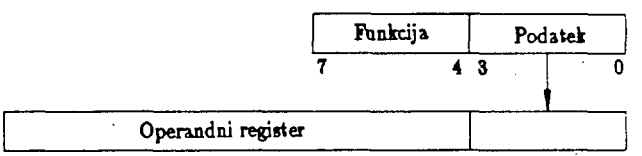
slika 3
Format transputerjeve instrukcije.

Pomembnejši štirje biti tvorijo funkcijsko kodo, preostali štirje so podatki. Dobimo 16 "direktnih" funkcij (nalaganje, shranjevanje, skoke, klice...).

Vse instrukcije se izvedejo tako, da se spodnji štirje biti prepišejo v spodnje štiri bite operandskega registra, katerega vsebina se kasneje uporabi kot operand instrukcije (glej sliko 4). Pri tem se vse instrukcije, razen Prefix instrukcij (njihova funkcija bo razložena kasneje), končajo s brisanjem operandskega registra. Tako je le ta pripravljen za naslednjo instrukcijo.

Ker pa takšno kodiranje dopušča samo štiri bitne operande, imamo med zgornjimi 16. funkcijami dve, ki omogočata razširjavo velikosti operandov. To sta Prefix in Negative Prefix. Prefix instrukcija napolni spodnje štiri bite operandskega registra s svojim podatkovnim poljem in potem premakne vsebino operandskega registra za štiri mesta v levo (šiftanje). Negative Prefix instrukcija je podobna pravkar opisani Prefix instrukciji, le da komplementira operandski register pred premaknitvijo vsebine za štiri mesta. Tako lahko operand s uporabo Prefix instrukcij povečujemo do velikosti operandskega registra.

Naslednja od zgornjih 16. funkcij je Operate, ki tretira svoj operandski del - osiroma vrednost operandskega registra - kot operacijo nad vrednostmi v registrih procesorja. Ta funkcija omogoča kodiranje še dodatnih 16. operacij v enem slogu.



slika 4
Polnjenje operandnega registra pri IMS T800.

S Prefix funkcijo lahko razširimo tudi operande funkcije Operate, kar je ekvivalentno povečanju nabora instrukcij. Seveda so instrukcije kodirane tako, da so najpogosteje uporabljane instrukcije predstavljene brez Prefix instrukcij. Merjenja so pokazala, da je okoli 70% instrukcij, ki se izvajajo, kodiranih v enem slogu. [3]

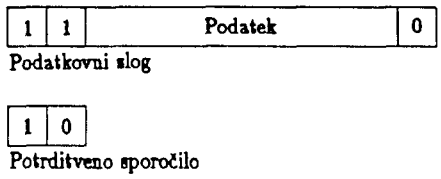
T800 ima dodatne instrukcije za delo s FPU. Pravtako vsebuje instrukcije za barvno grafiko, raspoznavanje vzorcev in implementacijo kod za odpravljanje napak. To je realizirano z zgoraj opisano možnostjo, ki omogoča razširitev nabora instrukcij. [5,7]

4 KOMUNIKACIJSKE POVEZAVE

Štiri identične dvosmerne povezave omogočajo sinhronizirano komunikacijo med procesorji in komunikacijo z zunanjim svetom. Vsaka povezava vsebuje vhodni in izhodni kanal. Povezava med dvema transputerjema je realizirana s povezovanjem vmesnika povezave enega transputerja na vmesnik povezave drugega transputerja. Vsak poslan podatkovni slog mora biti potrjen preko vhodnega kanala iste povezave. To je sinhronizacija, ki poteka na vseh štirih povezavah transputerja avtomatično in ne zahteva dodatnega programiranja.

Ko linija ni aktivna je izhodni kanal na nizkem nivoju. Vsak podatkovni slog se prenese kot zaporedje visokega start bita, enega visokega bita, tema bitoma sledi osem podatkovnih bitov in nizek stop bit. Potrditev, ki jo čaka oddajnik vsebuje visok in nizek bit (glej sliko 5) in je indikator za dvoje: proces je sprejel podatek in vmesnik je pripravljen za sprejem naslednjega sloga.

Pošiljanje potrditvenih paketov, preden se podatkovni paket popolnoma sprejme, poveča zmožnost povezav. IMS T414 nima implementiranega pravkar opisanega pošiljanja in dosega le 0.8 Mslogov na sekundo. Z implementacijo prekrivanja in zadostnih izravnalnikov za povezavo, je IMS T800 omogočena več kot dvakrat višja hitrost prenosa. [1,2,3,6,7]



Podatkovni slog

Potrditveno sporočilo

slika 5
Elementi komunikacijskega protokola

5 IMS T800

IMS T800 je naslednik T414, ki je bil prvi širše uporabljen predstavnik iz družine transputerjev. T800 vsebuje, za razliko s T414, integrirano enoto za delo s števili v plavajoči vejici (FPU). To predstavlja, glede na običajne rešitve s koprocesorji, le malo površino dodatnega silicija. Običajno zahteva zadovoljivo povečanje numerične zmogljivosti dodatno površino silicija, ki se giblje v razredu velikosti površine silicija, porabljene za izvedbo mikroprocesorja. Seveda to avtomatično zahteva eno ali več integriranih vezij z vso logiko, ki je potrebna za delovanje samega vezja (Weitek WTL1167 koprocesor za mikroprocesor Intel 80386 zahteva tri integrirana vezja). FPU deluje sočasno s centralno procesno enoto (CPU) in pod kontrolo CPU. (Glej sliko 1.)

5.1 ARHITEKTURA

Procesor transputerja T800 ima malo registrov, kar je kompenzirano s zelo hitrim pomnilnikom na vezju. Šest registrov in enostaven instrukcijski nabor omogoča enostavno kontrolno logiko in enostavne ter hitre podatkovne poti. Registri procesorja so: kazalec na delovno področje, kjer so shranjene lokalne spremenljivke; kazalec instrukcij, ki kaže na naslednjo instrukcijo, ki se naj izvrši; operandni register, kjer se nahaja operand instrukcije; registri A, B in C, ki predstavljajo strojno izveden sklad. Slednji trije registri se uporabljajo za aritmetiko pri naslavljanju, za celoštevilčno aritmetiko ter za logične operacije.

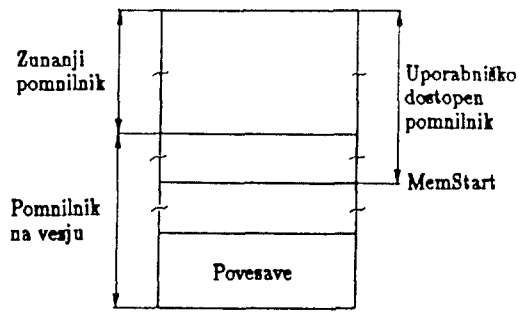
Tudi FPU ima tri registre (AF, BF in CF) v obliki sklada za računanje.

Naslovi za podatke, ki so zapisani v plavajoči vejici se formirajo na skladu CPU. Obenem je pod kontrolo CPU prenos vrednosti med naslovljenimi pomnilniškimi lokacijami in FPU. Ker je CPU sklad uporabljan le za naslavljanje vrednosti v plavajoči vejici, je dolžina besede CPU neodvisna od dolžine besede FPU. Tako dosežemo, da isti FPU na vezju T800 lahko uporabljata T212 (16 bitna beseda) in T414 (32 bitna beseda).

Registrski sklad FPU je podvojen. Pomembnost tega se vidi ob preklopu T800 v delovanje z visoko prioriteto, ne da bi bilo potrebno prepisovanje vsebine sklada v pomnilnik. Rezultat slednjega je zelo ugoden časovni odziv na prekinitve [3,6,7].

5.2 ORGANIZACIJA NASLOVNEGA POLJA

Celoten pomnilniški prostor je naslovljiv po slogih. Naslovi med #80000000 in #80000FFF naslavljajo pomnilnik na vezju (to je 4 Ksloge). Uporabniški pomnilnik se začenja na naslovu #80000070. Lokacija s tem naslovom je označena kot MemStart (Memory Start) [3]. Glej sliko 6. Naslovi povezav so v spodnjem delu pomnilnika na vezju.



slika 6
Organizacija naslovnega polja

5.3 INSTRUKCIJE ZA DELO S ŠTEVILI V PLAVAJOČI VEJICI

Jedro množice instrukcij za delo v plavajoči vejici so določili v fazi pred načrtovanjem IMS T800. To jedro vsebuje enostavne operacije vpisovanja in branja FP operandov ter osnovne aritmetične operacije. Po drugi strani pa je statistika, ki je bila izdelana na osnovi fortranskih programov pokazala, da bi z dodatkom nekaterih bolj kompleksnih instrukcij povečali učinkovitost in kompaktnost kode. Odločiti so se morali za najustreznejši nabor instrukcij. Zato so opravljali raziskave učinkovitosti predlaganih razširjav naborov instrukcij. Tak nabor so potem testirali v numerično orientiranih programih. Pri tem so za vsak predlagan nabor instrukcij skonstruirali prevajalnik, program prevedli z njim in tako dobljeno kodo testirali na simulatorju. V nadaljevanju sledi opis resultirajočega nabora instrukcij.

IMS T800 prenaša operande med pomnilnikom transputerja in skladom FPU z uporabo instrukcij za shranjevanje in nalaganje števil v plavajoči vejici. Obstajata dve skupini takšnih instrukcij: ena za števila enojne dolžine in ena za števila dvojne dolžine.

Naslov operandov v plavajoči vejici se izračuna na CPU skladu, nakar se operand naloži iz naslovljene pomnilniške lokacije na FPU sklad. Omogočena sta dva načina naslavljanja FP operandov: direktni in indirektni. Slednji način naslavljanja olajša delo s polji.

Operandi na FPU skladu imajo oznake, ki predstavljajo njihovo dolžino. Oznaka operanda se nastavi, ko operand naložimo oz. izračunamo. Te oznake zmanjšajo število instrukcij, ki jih rabimo pri aritmetiki v plavajoči vejici. Npr. ne rabimo instrukcije za seštevanje dolgih besed in za seštevanje kratkih besed, temveč preprosto le instrukcijo za seštevanje.

Operaciji sa branje resultatov iz FPU shranita prebrano vrednost iz FPU sklada v transputerjev pomnilnik. Za branje ne obstajajo indeksne instrukcije. To izgleda na prvi pogled presenetljivo, vendar izvira iz dejstva, da je v programih manj operacij branja iz FPU, kot pa vpisovanja v FPU. Zato indeksno naslavljanje pri branju ni realizirano.

Enojne instrukcije omogočajo najpogostejše operacije v FP: seštevanje, odštevanje, množenje, deljenje in primerjavo. Rezultat primerjave se prenese v register CPU.

Zaradi pogostega seštevanja in množenja v programih so v smislu čim večje kompaktnosti kode in hitrosti izvajanja nekatere instrukcije sestavili iz več osnovnih instrukcij. Npr instrukcija prištevanja operanda operandu na skladu je enakovredna dvema operacijama: vpisu operanda na sklad in seštevanju operandov na skladu. [3,6,7]

5.4 SOČASNE OPERACIJE FPU IN CPU

Pri IMS T800 dela FPU sočasno s CPU. Ta sočasnost zelo izboljša značilnosti v realnih problemih, kjer so elementi polja relativno težko dostopni. To je razvidno iz "Livermore Loops" testa, ki bo opisan v nadaljevanju. Ta test je množica majhnih jeder napravljenih tako, da predstavljajo možne tipe izračunov. Posebno vsebuje dostope do dvo in trodimensionalnih polj, to je tam kjer transputerjeva sočasnost pokaže zelo dobre resultate. Prevajalnik izbere najugodnejši vrstni red računanja naslovov in s tem poveča časovno prekrivanje.

Pri testu "Livermore Loops" je IMS T800-30 dosegel 2.25 MFLOPS, IMS T800-20 1.5 MFLOPS, T414-20 0.09 MFLOPS in VAX 11/780 (s PF pospeševalnikom) 0.54 MFLOPS. Program napisan v Occamu za ta test ima obliko [7]:

```
- LIVERMORE LOOP 7
SEQ k = 0 FOR n
    x[k] := u[k] + ((( r*(z[k] + (r*y[k]))) +
                    (t*((u[k+3] + (r*(u[k+2] + (r*u[k+1])))))))) +
                    (t*((u[k+6] + (r*(u[k+5] + (r*u[k+4])))))))
```

5.5 ZMOŽNOST IMS T800 ZA FP OPERACIJE

Časi izvajanj FP operacij niso zanesljivo merilo hitrosti izvajanja pravih numerično orientiranih programov. Zaradi tega primerjamo numerično učinkovitost procesorjev s Whetstonovim preiskusom. To je program, ki je dobra imitacija znanstveno-tehničnega programa. Vsebuje ustrezno število in strukturo operacij v plavajoči vejici, klicov procedur, indeksiranja polj in računanja transcendentnih funkcij.

Tabela 2 podaja zmožnosti IMS T414 in IMS T800 v primerjavi s ostalimi procesorji glede na Whetstonov test. IMS T414 je trikrat počasnejši kot koprocesor MC68881, vendar ima kombinacija MC68000/MC68881 le 25% večje zmožnosti kot T414. To je zato ker je hitrost izračuna FP izrasa odvisna od dveh stvari: prvič od hitrosti prenosa operandov v in iz koprocesorja in od hitrosti same FP enote. S skrbnim uravnotežen jem teh faktorjev, postane eno samo integrirano vezja IMS T800-20 več kot petkrat hitrejše od kombinacije MC68000/MC68881 [7].

Procesor	Tip	Količina/s
Intel 80286/80287	8 MHz	300K
IMS T414-20	20 Mhz	663K
NS 32332-32081	15 Mhz	728K
MC68000/MC68881	16/12 Mhz Sun3	860K
VAX 11/780 FPA	Unix 4.3 BSD	1083K
IMS T800-20	20 Mhz	4000K
IMS T800-30	30 Mhz	6000K

Tabela 2
Primerjava procesorjev na osnovi Whetstonovega testa

Drugi pomembni kriteriji so še učinkovitost glede na površino silicija, zasedanje prostora na tiskanem vezju in potrebno št. dodatnih vezij. Poleg tega IMS T800 v mnogih aplikacijah ne potrebuje zunanjega pomnilnika, saj ga je na čipu že 4 Ksloge. Štirje IMS T800-30 zavzemajo enako površino na tiskanem vezju kot 80386 s WTL1167, obenem pa omogočajo šestkratno učinkovitost v vsaki sočasni aplikaciji.

Procesor	Količina/s
IMS T414-20	33K
Intel 80286/80287	37.5K
NS 32332-32081	48.5K
MC68000/MC68881	54K
IMS T800-20	200K
IMS T800-30	200K

Tabela 3
Normirana primerjava procesorjev na osnovi Whetstonovega testa pri taktu 1MHz

Tudi iz tabele 3 je razvidna moč T800 v primerjavi s predhodnikom T414. Ta moč izvira iz naslednjih dejstev:

- V testih T800 uporablja 4 Ksloge velik pomnilnik, ki je integriran v vezju.
- FPU je prav tako integrirana v vezje.
- FPU in CPU delujeta sočasno, pri čemer CPU računa naslove operandov v FP operacijah.

Rezultati bi bili za T800 manj ugodni, če bi naslavljal pomnilnik, ki ni na vezju: še slabši bi bili, če bi uporabljal serijske povezave za doseganje operandov.

6 ZAKLJUČEK

IMS T800 transputer je zelo zmogljiv gradnik za paralelne sisteme. Obenem dokazuje, da ni potrebno uporabljati koprocesorja, če želimo veliko numerično računalniško moč. Pri transputerju T800 najdemo na enem integriranem vezju centralno procesno enoto, numerično enoto za delo s števili v plavajoči vejici, pomnilnik in komunikacijski sistem. Skratka cel računalnik na enem vezju. Na primer: štiri Ksloge pomnilnika je v aplikacijah, kjer obdelujemo signale, ponavadi dovolj. Pri tem ne potrebujemo zunanjega pomnilnika. Zanimivo je tudi dejstvo, da je zmogljivost T800 pri računanja v plavajoči vejici večja od zmogljivosti mnogih drugih procesorjev pri računanju v številskih sistemih s fiksno piko.

To vezje bo osnova najmočnejšega evropskega superračunalnika, ki ga načrtujejo na univerzi v Edinburgu. Vsebovalo bo 1000 transputerjev in 1 Gslog pomnilnika. S takšno rešitvijo bo superračunalnik velik le slab kubičnih meter. Današnja tehnologija s katero je izdelan T800-20 ali T800-30 nudi 1.5 GFLOPS na 0.028 kubičnega metra strojne opreme.

Literatura

[1] Inmos reference manual for transputers, 1986
[2] Engineering data - IMS T414 transputer, 1986
[3] Engineering data - IMS T800 transputer, 1987
[4] Inmos product overwiew (transputer development system), 1986
[5] Inmos compilers writer's guide, 1987
[6] IMS T800 Architecture - technical note 6, 1988
[7] M Homewood, D May, D Shepherd, R Shepherd: The IMS T800 transputer: IEEE Micro, October 1987
[8] B Mihovilovič, S Mavrič, P Kolbezen: Transputer - osnovni gradnik večprocesorskih sistemov: Informatica 4, 1986
[9] F Mayer-Lindenberg: FIFTH on the transputer: Microprocessing and microprogramming, December 1987

OCENA INFORMACIJSKE ŠKODE STROŠKOVNO IN DOHODKOVNO TRANSFORMIRANIH PROIZVODNIH SISTEMOV

INFORMATICA 1/89

Deskriptorji: PROIZVODNJA, ORGANIZACIJA, VODENJE, MODELIRANJE, STROŠEK, DOHODEK

Viljem Rupnik
Reboljeva 16, Ljubljana

Modeliranje proizvodnih struktur je še posebej občutljivo na kriterij modeliranja. Tu obravnavamo stroškovni in dohodkovni kriterij za meritve uspešnosti delovanja proizvodnega sistema in njun vpliv na izgubo informacij, pomembnih za upravljanje.

1. Uvod

Na področju organizacije in vodenja proizvodnje poznamo zelo pestro množico modelov, ki so (vsaj v praksi) pretežno statične narave in ki kljub morebitni visoki parametrizaciji zaradi svoje arhitekture spravljajo v zadrego, ko želimo doseči največji možni sinergetski efekt tako pri planiranju, kakor tudi pri upravljanju proizvodnih procesov. Ta primanjkljaj sinergetske mase - kot se da pokazati - je še najmočnejši pri operativnih oblikah upravljanja, manj pa pri taktičnih verzijah modelov proizvodnih sistemov, kjer se ta defekt izgubi oziroma preglasi z razponom možne disperzije, ki jo povzroča slučajnostni značaj pogojev delovanja proizvodnega sistema predvsem v njegovi okolici. Naravna se zdi zahteva po ločeni "odgovornosti" ekonomista od "odgovornosti" tehnika, zato je smiselna takšna gradnja modelov proizvodnih sistemov, ki so zlahka dostopni za različne vrednostne transformacije, kot so npr. stroškovne in dohodkovne transformacije. V dosedanji literaturi, še bolj pa v praksi, prevladujejo modeli, ki so mešanega tipa, torej vsebujejo tako tehnične in tehnološke kot tudi ekonomske kategorije kot konstituante v istem modelu, pri tem pa je redkokdaj ločljiva ena množica konstituant od druge na način, ki bi omogočal preproste razumljive preslikave.

Da bi ocenili informacijske izgube, ki nastanejo pri prehodu modela proizvodnega sistema v "naturalni" obliki v vrednostno obliko modela, predpostavimo poljubni proizvodni sistem, ki naj bo po svoji naravi dinamični sistem ter sintetizirane oblike glede na posamične proizvodne podsisteme (npr. stroškovna mesta). Za tako oblikovani model proizvodnega sistema lahko uporabimo rezultate o njegovi multikriterialni upravljivosti (/1/). Znano je namreč, da praksa zahteva že v tehnološki sferi optimizacijo proizvodnih sistemov po več kriterijih hkrati. Eksistenčni izreki multikriterialne upravljivosti dinamičnega modela splošnega proizvodnega sistema

$$\hat{S}_\Sigma = (S_o, S_1, \ldots, S_s; R, \Phi_\Sigma) \qquad (1)$$

slonijo na razmeroma ostrih lastnostih posameznih informacijskih razredov, vendar ne tako težko izpolnjivih, da se ne bi potrudili zanje v zameno za optimalnost, ki daleč presega koncepcijo neparetovske optimalnosti. Zgoraj pomenijo $S_1, \ldots, S_s$ proizvodne podsisteme, od katerih je S_o okolica proizvodnega sistema, R pomeni množico binarnih, triadnih, itd. relacij med podsistemi $S_1, \ldots, S_s$, Φ_Σ pa je operator upravljanja dinamičnega sistema v multikriterialnem prostoru upravljalskih kriterijev. Glede na /1/ natanko vemo, kdaj je tak proizvodni sistem popolno in hkrati polno upravljiv na danem upravljalskem horizontu T; podali smo tudi pogoje absolutne in enakomerne upravljivosti kot tisto zaostritev polne upravljivosti, ki jo terjajo proizvodni sistemi, ki jih moramo upravljati v realnem času (procesna kontrola). Dovolj bo, da matematičnemu sistemu (1) dopustimo kalmanovsko naravo, saj ta zelo visoko prekriva večino proizvodnih sistemov, ki so hibridi končne množice elementarnih tehnoloških procesov in operacij. Opozoriti pa je treba, da takrat, ko proizvodni sistem (1) oklenemo z organizacijskim sistemom, predpostavka o kalmanovski naravi sistema (1) ne zadošča več. To velja toliko bolj za oba sistema, tj. proizvodni in organizacijski sistem (za katere se da ne tako enostavno dokazati, da organizacijski sistem vsebuje proizvodni sistem in ne narobe). Toda oblikovanje poslovnega sistema terja še dve inkluziji: objem pravkar zgrajenega tandema s strani informacijskega sistema in objem vseh treh v upravljalski sistem. Pri določenih sistemskih lastnostih lahko tako dobljeno četverico obravnavamo kot poslovni sistem. Zanj se da uvideti, da je njegova matematična struktura mnogo bolj zahtevna (/2/) kot pa je matematična struktura, ki smo jo privzeli za (1).

Oznaka proizvodnega sistema (1) kot "naturalnega" modela je opravičena z lastnostjo, da so vsi inputi kot tudi outputi nevrednostne dimenzije, medtem ko operator upravljanja Φ_Σ lahko vsebuje tudi kakšno vrednostno kategorijo kot kriterij upravljanja. Vrednostna transformacija, kot sta npr. stroškovna, dohodkovna, prihodkovna in druge, je definirana kot tista transformacija, ki obravnava vrednostne inpute in outpute, pri čemer z izrazom "vrednost" označujemo poljubno vrednost v najširšem pomenu besede (korist, "benefit"). Preden se lotimo ocene informacijskih izgub, ki jih s seboj prinašata dve najbolj pogosti vrednostni transformaciji, tj. stroškovna in dohodkovna transformacija proizvodnega sistema (1), povzemimo peterico izrekov v (/1/) s tem, da navedemo osnovni izrek o zadostnem pogoju terminalne multikriterialne upravljivosti:

Proizvodni sistem $\hat{S}_\Sigma$ z vektorskim kriterijem

$(\hat{\phi}_\Sigma)$ in omejenim normiranim vektorskim prostorom upravljanja U_Σ kot komutativnima grupama, z monoidnim prostorom običajnih vhodov $\overline{X}_\Sigma$, operatorjem $\hat{\phi}_\Sigma$, ki je aditivno separabilen glede na $\overline{X}_\Sigma$ in U_Σ v smislu

$$\forall\, x,u:\ \hat{\phi}_\Sigma = \hat{\phi}_{\Sigma(1)}(t,\ x_\Sigma(t)) + \hat{\phi}_{\Sigma(2)}(t,\ u_\Sigma(t))$$

in ki je aditivno homomorfen z ozirom na $u_\Sigma(t)$ v smislu

$$\hat{\phi}_\Sigma(.,u_{\Sigma 1}(t) + u_{\Sigma 2}(t)) = \hat{\phi}_{\Sigma(2)}(.,u_{\Sigma 1}(t)) +$$
$$+ \hat{\phi}_{\Sigma(2)}(.,u_{\Sigma 2}(t))$$

je na Δ - popolnoma upravljiv in na T polno upravljiv, če je

1) $\hat{S}_\Sigma$ popolnoma upravljiv na

$$\Delta\,\hat{\mathfrak{R}}\Big|_{u'_\Sigma} = \hat{\phi}_\Sigma(T,\ \overline{x}_\Sigma,\ U'_\Sigma),\quad \exists U'_\Sigma \subset U_\Sigma,\ U'_\Sigma \neq \emptyset$$

2) če velja

$$\Delta\,\hat{\mathfrak{R}} = \bigcap_{u_\Sigma \in U_\Sigma} \mathfrak{R}\,\{\hat{\phi}_\Sigma(T,\ \overline{x}_\Sigma,\{u_\Sigma\})$$

Pri tem naj spomnimo, da popolna upravljivost pomeni upravljivost glede na vse izbrane kriterije istočasno in da ta upravljivost ne dopušča paretovskega kompromisa. Opozarjamo tudi, da je naloga, oceniti informacijsko škodo zaradi transformacije (1) na stroškovno in dohodkovno izražene konstituante proizvodnega sistema, precej ožja od naloge, oceniti upravljalsko škodo, ki utegne nastati pri kakšni drugi transformaciji. Naša naloga torej pomeni napor, ugotoviti, ali stroškovna in dohodkovna transformacija vplivata na definicijske prostore proizvodnega sistema (1). Pri tem se bomo zaradi obsežnosti problematike informacijskih izgub, ki nastanejo zaradi dimenzijske redukcije proizvodnega sistema (1), popolnoma izognili vprašanjem reducibilnosti tega sistema in jih obravnavali kdaj drugič.

2. Stroškovne in dohodkovne transformacije proizvodnega sistema

Med najrazličnejšimi vrednostnimi transformacijami proizvodnega sistema (1) si oglejmo dohodkovno in stroškovno transformacijo posameznih ali celo vseh naturalnih konstituant modela proizvodnega sistema (1). Predpostavimo, da je $\hat{S}_\Sigma$ dimenzijsko nereduciran in si najprej oglejmo prostor običajnih vhodov $X_\Sigma = (X_{\Sigma\rho},\ X_{\Sigma\chi},\ X_{\Sigma\omega})$, kjer pomeni $X_{\Sigma\rho}$ podprostor neinformacijskih vhodov (proizvodni materiali, energija itd.), $X_{\Sigma\chi}$ podprostor karakteristik nematerialnega vhoda in $X_{\Sigma\omega}$ podprostor informacijskih vhodov v informacijski sistem (1). Na te podprostore uporabimo naslednje stroškovne transformacije

$$\begin{aligned} X_{\Sigma\rho} &\xrightarrow{C_\rho^{(x)}} \tilde{X}_{\Sigma\rho}^{(c)} \\ X_{\Sigma\chi} &\xrightarrow{C_\chi^{(x)}} \tilde{X}_{\Sigma\chi}^{(c)} \\ X_{\Sigma\omega} &\xrightarrow{C_\omega^{(x)}} \tilde{X}_{\Sigma\omega}^{(c)} \end{aligned} \qquad (1)$$

kjer npr. operator $C_\rho^{(x)}$ zavisi o cenovnih parametrih na nabavnem tržišču (npr. materialov). Pri tem v splošnem pričakujemo $C_\rho^{(x)} = C_\rho^{(x)}\left[C_\omega^{(x)},\ X_{\hat{S}_\Sigma}^{(x)}\right]$ in ta transformacija vedno eksistira, če $\hat{S}_\Sigma$ deluje kot samostojna proizvodna celota, ki komunicira z okolico. Ker se lastnosti X_χ posredno izražajo preko nabavnih cen za X_ρ, imamo običajno $C_\chi^{(x)} \equiv 0$; vendar pa je v splošnem $C_\omega^{(x)} \neq 0$. Naslonimo se na rezultate iz (/2/) o načinu sinteze posameznih proizvodnih podsistemov S_i, $\forall i$. Proizvodnemu sistemu tako vedno lahko najdemo totalni primarni običajni vhod $X_\Sigma = \prod_k^s X_k/Y^{(x)}$; temu glede na (1) pripada stroškovno transformirani prostor

$$X_\Sigma = \prod_k^s X_k/Y^{(x)} \qquad (2)$$

ki je <u>stroškovni totalni primarni običajni vhod</u> za S_Σ s strukturo $X_\Sigma = (X_{\Sigma\rho},\ 0,\ X_{\Sigma\omega})'$. Tako smemo predpostaviti $C_\Sigma^{(x)} = (C_\rho^{(x)},\ 0,\ C_\omega^{(x)})$ in imamo spet

$$X_\Sigma \xrightarrow{C_\Sigma^{(x)}} \tilde{X}_\Sigma \qquad (3)$$

kot osnovno stroškovno transformacijo proizvodnega sistema S_Σ.

Za stroškovni transformat (rezultat transformacije, ki pripada totalnemu navadnemu sekundarnemu izhodu $Y^{(x)}$, moramo dovoliti obstoj operatorja kot kompozitum operatorja $C^{(u)}$ z lastnostjo

$$U_k \xrightarrow{C_k^{(u)}} \tilde{U}_k \qquad (4)$$

in imamo s tem

$$\overline{X}_{kj} \xrightarrow{C_{kj}^{(\overline{x})}} \tilde{\overline{X}}_{kj} \qquad (5)$$

ter je $C_{kj}^{(\overline{x})} = (C_{kj}^{(u)},\ C_{kj}^{(x)})$

stroškovna transformacija navadnega in upravljalskega vhoda v proizvodni sistem. V poslovnem svetu velja načelo, da se vrednost ne sme izgubljati, zato to načelo - imenovano načelo kompenzacije - določa identiteto $Z_\chi \equiv 0$, $Y_{jk} \equiv \overline{X}_{kj}$. Po definiciji celotnega izhoda iz S_Σ pa imamo sedaj

$$\prod_k^{s-1} Y_{jk} = \prod_k^{s-1} (Y_{jk}^{(x)} \times X_{jk}^{(u)}) \subset Y_j \qquad (6)$$

kjer so $Y_{jk}^{(x)}$, $Y_{jk}^{(u)}$ in Y_j stroškovni transformati prostorov $Y_{jk}^{(x)}$, $Y_{jk}^{(u)}$ in Y_j. Na tem mestu seveda še nismo ugotovili, ali ustrezni stroškovni operatorji sploh obstajajo. Iz (6) sledi, da je $Y_{jk}^{(x)}$ <u>stroškovni navadni sekundarni vhod</u>, ki ga generira podsistem S_j; za celotni proizvodni sistem pa imamo

$$\prod_j^s \prod_k^{s-1} Y_{jk}^{(x)} = Y^{(x)} \qquad (7)$$

<u>kot totalni stroškovni navadni sekundarni vhod</u>

za S_Σ. Glede na rezultate v /2/ pa lahko sklepamo, da stroškovni operatorji, ki se nanašajo na sekundarne vhode, ne zavisijo samo od nabavnih cen, temveč tudi od operatorjev tipa G (operatorji outputov) in relacije R med podsistemi S_j. Ker operatorji G_j zavisijo od prostora stanj Z_j v vsakem podsistemu, si najprej oglejmo stroškovne transformacije

$$\begin{aligned} z_{\Sigma\rho} &\xrightarrow{C_\rho^{(z)}} \tilde{z}_{\Sigma\rho} \\ z_{\Sigma\chi} &\xrightarrow{C_\chi^{(z)}} \tilde{z}_{\Sigma\chi} \\ z_{\Sigma s} &\xrightarrow{C_s^{(z)}} \tilde{z}_{\Sigma s} \end{aligned} \qquad (8)$$

in ki torej stroškovno izražajo stanje proizvodnega sistema (npr. revalorizirana neodpisana vrednost osnovnih sredstev, ki še "čaka" za vkalkulacijo v stroške bodoče proizvodnje). Pri tem $C_\rho^{(z)}$ pomeni kompozitum za posamične S_j, j=1,...,s, tako kot prej. Za proizvodni podsistem S_j npr. $Z_{j\rho}$ vsebuje vsa možna stanja v pogledu cepitve materialnih tokov tako znotraj S_j kot tudi med njimi. V prvem primeru $Z_{j\rho}$ c $Z_{\Sigma\rho}$ učinkuje preko G_j na y_j pa je zato ustrezna stroškovna transformacija sposobna vključevati posledice takšnih dogajanj za y_j kot stroškovni transformat vektorskega outputa y_j proizvodnega podsistema S_j. V drugem primeru $Z_{\Sigma\rho}$ sodeluje pri oblikovanju medfaznih transformatov, tj. stroškovnih izrazov notranje navadne emisije Y_{jk}, kar pomeni, da zaradi $Z_{\Sigma\rho} \neq 0$ načelo kompenzacije dobi naslednjo obliko

$$\begin{aligned} \tilde{\tilde{Y}}_{kj} = \tilde{\tilde{X}}_{kj} &+ C_{\rho,kj}^{(z)}(Z_{\rho,kj}) + \\ &+ C_{\chi,kj}^{(z)}(Z_{\chi,kj}) + C_{s,kj}^{(z)}(Z_{s,kj}) \end{aligned} \qquad (9)$$

Tako lahko sedaj <u>stroškovni princip kompenzacije</u> izrazimo krajše

$$\tilde{\tilde{Y}}_{kj} = \tilde{\tilde{X}}_{kj} + \left[C_{\rho,kj}^{(z)}, C_{\chi,kj}^{(z)}, C_{s,kj}^{(z)}\right] \begin{bmatrix} R_{\rho,kj}(\bar{X}_{kj}) \\ R_{\chi,kj}(\bar{X}_{kj}) \\ R_{s,kj}(\bar{X}_{kj}) \end{bmatrix} \qquad (9')$$

Diskusija trojice stroškovnih operatorjev izhaja iz same definicije prostorov $Z_{\Sigma\rho}$, $Z_{\Sigma\chi}$ in $Z_{\Sigma s}$. Tako se $C_{\rho,kj}^{(z)}$ definira kot enotna projekcija stroškov transporta izhoda $\bar{X}_{kj}$ od S_k k S_j, kar pomeni, da je ta projekcija stroškovni parameter povsem internega značaja. Podobno se pri $C_{\chi,kj}^{(z)}$ definirajo enotne projekcije stroškov, ki povzročajo procesne lastnosti proizvodnega sistema na relaciji od S_k k S_j. Kar zadeva transformacijo $C_{s,kj}^{(z)} : Z_{s,kj} \to Z_{s,kj}$, velja opozoriti na naslednji problem. V prostor Z_s spadajo fizikalne lastnosti osnovnih sredstev, naprav itd., izražene z njihovimi projekcijami na čas. Zato se $C_s^{(z)}$ definira s projekcijo <u>kontingenčne cene</u> na življenjski horizont materialnega substrata takšnega elementa; nabavna vrednost npr. se projecira na življenjsko dobo neke naprave. Tako torej $C_s^{(z)}$ skupek "cen" odnosno stroškov vzdrževanja eksistenčnih lastnosti materialnih tokov (vhodov, izhodov in vmesnih produktov) med proizvodnimi podsistemi.

Stroškovne transformacije izhodov Y_{kj} v $\tilde{Y}_{kj}$ so s tem v celoti opravljene s pomočjo operatorja $C_{kj}^{(z)} = (C_{\rho,kj}^{(z)}, \cdot, \cdot)$. Z uporabo preslikav $Y_{kj}^{(x)} \to \tilde{Y}_{kj}^{(x)}$, $\forall k$ in uporabo zaporedja operatorjev tipa $C_\Sigma^{(x)}$, $C_\Sigma^{(z)}$ in $C_\Sigma^{(u)}$ smo našli

$$\begin{aligned} Y_{kj}^{(x)} &\longrightarrow \tilde{Y}_{kj}^{(x)} \qquad k=1,\ldots,s \\ Y_{kj}^{(u)} &\longrightarrow \tilde{Y}_{kj}^{(u)} \qquad k=0,\ldots,s \qquad j = 1, \ldots, s \\ Y_j &\longrightarrow \tilde{Y}_j \end{aligned}$$

kar vodi k stroškovnim transformacijam

$$Y_{j_o}^{(x)} \longrightarrow \tilde{Y}_{j_o}^{(x)} \quad \text{oziroma} \quad Y_{j_o}^{(u)} \longrightarrow \tilde{Y}_{j_o}^{(u)} \qquad (10)$$

kar pomeni, da smo našli <u>stroškovne transformate totalnega navadnega izhoda</u> in totalnega upravljalskega izhoda za vsak proizvodni podsistem S_j. Sedaj je možna tudi stroškovna transformacija

$$U_\Sigma + \tilde{U}_\Sigma^{(c)} = \pi_k^s \tilde{U}_\Sigma^{(c)} / \tilde{Y}^{(u)} \qquad (11)$$

kar pomeni, da poznamo tudi <u>totalni stroškovni primarni upravljalski vhod</u>, medtem ko je <u>totalni stroškovni sekundarni upravljalski vhod</u> vključen v $\bar{X}_\Sigma$.

Da bi dobili popoln stroškovni obračun procesov v proizvodnem sistemu, moramo podvreči stroškovni transformaciji tudi vse izhode. V ta namen moramo osvetliti obstoj $\tilde{Y}_j$, $\forall j$. Predvsem pričakujemo naslednje implikacije $Y_{j\rho} \neq 0 \Rightarrow \tilde{Y}_{j\rho}^{(c)} \neq 0$, $\tilde{Y}_{j\omega} \neq 0 \Rightarrow Y_{j\omega} \neq 0$. V splošnem lahko rezultat "ostroškovanja" izrazimo kot $\tilde{Y}_j = \tilde{Y}_{j\rho} + \tilde{Y}_{j\chi}$, kajti stroški, ki imajo svoj "izvor" v lastnostih materialnih vhodov, dodatno obremenjujejo stroške "proizvodnje" totalnega izhoda. Vendar pa redkeje lahko pričakujemo eksplicitno poznavanje $Y_{j\chi}$, ker stroške izhodov proizvodnega sistema ne zajemamo istočasno kot za materialni substrat izhoda in pa njegove lastnosti kot <u>posledice sistemskih izhodov</u>. Zaradi preglednosti lahko v nadaljnjem povzamemo kar $\tilde{Y}_{j\chi} = 0$ oziroma $\tilde{Y}_{j\rho} + \tilde{Y}_{j\chi} = \tilde{Y}_{j\rho}$, odkoder sledi

$$\begin{aligned} \tilde{y}_j &= G_j\left[t, t_o, C_j^{(z)} z(t_o), (t, C_j^{(x)} x(t), C_j^{(u)} u(t)\right]_{[t_o,t]} \\ &= (C_j^{(z)}, C_j^{(x)}, C_j^{(u)})\, G_j\left[t, t_o, z(t_o), (t, x, u)\right]_{[t_o,t]} = \\ &= \tilde{G}_j\,[\cdot] \end{aligned} \qquad (12)$$

če le poznamo operator izhoda G_j in stroškovne transformacije na desni strani v (12). Tako smo ugotovili, da so za poznavanje stroškovnega transformata y_j potrebne vse stroškovne

transformacije upravljalskega vhoda, običajno vhoda in stanja, kar lahko zapišemo takole

$$G_j \xrightarrow{c_j^{(x)},\ c_j^{(u)},\ c_j^{(z)}} \tilde{G}_j$$

Podobno se lahko prepričamo tudi o transformaciji $H_j \longrightarrow \tilde{H}_j$ preko istih stroškovnih transformacij.

V celoti smo našli stroškovne transformacije in sicer: za totalni stroškovni navadni primarni in sekundarni vhod, za totalni stroškovni upravljalski primarni in sekundarni vhod in za totalni stroškovni upravljalski in navadni izhod. Na kratko zapišemo $C_\Sigma = (C_\Sigma^{(x)}, C_\Sigma^{(z)}, C_\Sigma^{(u)})$, s tem pa stroškovno izražena operatorja prehoda stanj v sistemu in izhoda iz sistema $\tilde{G}_j = C_\Sigma G_j$ in $\tilde{H}_j = C_\Sigma H_j$ omogočata, da je definicijsko področje stroškovno transformiranega proizvodnega sistema

$$\mathcal{D}(\hat{S}_{\Sigma,c_\Sigma}) = \{(X_k, Z_k, Y_k) \in S_k,\ k=0,\dots,s,(R, C_\Sigma)\} =$$
$$= \{(\bar{X}_{kj}, Y_{kj}, Z_{kj}), (Z_k, Z_j), Y_{j_o}^{(u)}, Y_{j_o}^{(x)};$$
$$k,j = 0,\dots,s;\ X_\Sigma, U_\Sigma\}$$

oziroma da smo prišli na stroškovno transformirani proizvodni sistem (stroškovno-proizvodni transformat)

$$\hat{S}_{\Sigma,c_\Sigma} = \{S_o, S_1, \dots, S_s; R, C_\Sigma, \hat{\phi}_\Sigma\} \qquad (13)$$

Tako smo torej spoznali, da je stroškovna transformacija definirana v načelu nad celotno množico definicijskih prostorov proizvodnega sistema (1), da pa so oblike takšne transformacije le prostori, katerih število je manjše od števila definicijskih prostorov prvotnega sistema. Sistem (13) ima torej manjše informacijsko bogastvo, čeprav se na tem omejenem prostoru ni mogoče lotiti obsežnega razmišljanja o posledicah zmanjšanega informacijskega ozadja na upravljivost sistema $\hat{S}_{\Sigma,c_\Sigma}$, vendarle že vidimo, da je zaradi obstoja $\{\tilde{H}_k\}$ in $\{\tilde{G}_k\}$ stroškovno transformirani sistem $\hat{S}_{\Sigma,c_\Sigma}$ upravljiv na isti način kot so upravljivi vsak S_j posebej. Dalje, stroškovni operator C_Σ v $\hat{S}_{\Sigma,c_\Sigma}$ definira množico $\phi_1,\dots,\phi_s$ funkcionalov s stroškovno dimenzijo; takšno naravo pričakujemo tudi od $\phi_\Sigma(\phi_1,\dots,\phi_s)$. Odtod pa vidimo, da ostane definicija upravljivosti stroškovno-proizvodnega sistema (13) ista kot v primeru (1). Vendar pa nas specifikacija definicijskih področij za $\phi_1,\dots,\phi_s$ vodi do vprašanja upravljivosti $\hat{S}_{\Sigma,c_\Sigma}$ glede na

a) X_Σ, Z_Σ, Y_Σ, U_Σ pri C_Σ = const ;

b) vsaj en stroškovni transformat izmed X_Σ, Z_Σ, Y_Σ, U_Σ ;

c) C_Σ kot prostor kontrolnih parametrov.

S tem pa smo pripravili tla za študij upravljivosti poljubnega "stroškovno-proizvodnega" sistema, ki se tako rad ponuja v obdelavo v okviru ekonomije. Odtod je tudi očitno, da upravljivost proizvodnega sistema, ki ga pričakujemo v sferi tehnologije, še ne zagotavlja upravljivosti iste vrste tudi za sistem (13).

Oglejmo si sedaj dohodkovno transformirane proizvodne sisteme. Da ne bi preveč izgubili na širini veljavnosti razmišljanja, naj opozorimo, da z manjšimi spremembami tako razmišljanje lahko ponovimo tudi na primeru poljubne dohodkovne transformacije. Najprej se spomnimo, da po definiciji interna emisija Y_j podsistema S_j v smeri k podsistemu S_k ni predmet "trženja", zato ne more biti nosilec niti prihodka niti dohodka; ostaja samo nosilec stroškov. Proizvodni sistem izhodno komunicira samo preko $Y_{j_o}^{(x)}$ in $Y_{j_o}^{(k)}$. Pri tej izbrani stroškovni transformaciji velja

$$Y_{j_o}^{(x)} = (Y_{j_o,\rho}^{(x)}, Y_{j_o,\chi}^{(x)}, Y_{j_o,\omega}^{(x)}) \to (\bar{Y}_{j_o,\rho}^{(x),(c)}, 0, Y_{j_o,\omega}^{(x),(c)})$$

$$Y_{j_o}^{(u)} = (Y_{j_o,\rho}^{(u)}, Y_{j_o,\chi}^{(u)}, Y_{j_o,\omega}^{(u)}) \to (Y_{j_o,\rho}^{(u),(c)}, 0, Y_{j_o,\omega}^{(u),(c)})$$

in je za $t \in T$ znan sistem prodajnih cen

$$P_{j_o} = \{P_{j_o,\rho}^{(x)} = (p_{j_o,1}^{(\rho)},\dots,p_{j_o n_{j_o}}^{(\rho)}),\ P_{j_o,\chi}^{(x)} = (0,\dots,0),$$
$$P_{j_o,\omega}^{(x)} = (p_{j_o,1}^{(\omega)},\dots,p_{j_o n_{j_o}}^{(\omega)}),\ P_{j_o,\rho}^{(u)} =$$
$$= (p_{j_o,1}^{(u)},\dots,p_{j_o n_{j_o}}^{(u)}),$$
$$P_{j_o,\chi}^{(u)} = (0,\dots,0),\ P_{j_o,1}^{(u)},\dots,P_{j_o,n_{j_o}}^{(u)})\}$$
$$\forall j \in \{1,\dots,s\}$$

ki omogoča prehod

$$(Y_{\Sigma,o}^{(x)}, Y_{\Sigma,o}^{(u)}) \xrightarrow{\Phi_\Sigma} (Y_{\Sigma,o}^{(x),(p)}, Y_{\Sigma,o}^{(u),(p)}) \qquad (14)$$

imamo tedaj z operatorjem določeno transformacijo finalnih izhodov za njihove tržne vrednosti. Zaradi preglednosti razmišljanja predpostavimo, da se ta tržna vrednost tudi realizira, torej je operator π definiran takole

$$(Y_{\Sigma,o}^{(x)}, Y_{\Sigma,o}^{(u)}) \xrightarrow{\pi_\Sigma} (Y_{\Sigma,o}^{(x),(p)} - Y_{\Sigma,o}^{(x),(c)}, Y_{\Sigma,o}^{(u),(p)} - Y_{\Sigma,o}^{(u),(c)}) \qquad (15)$$

oziroma krajše $\pi_\Sigma = \Phi_\Sigma - C_\Sigma$ je operator, ki finalnim izhodom prireja njihove dohodke. Odtod vidimo, da π_Σ deluje na precej manjšem številu prostorov kot pa delujejo operatorji Φ_Σ in C_Σ, namreč samo na prostorih $Y_{\Sigma,o}^{(x)}$ in $Y_{\Sigma,o}^{(u)}$. V (2) smo pokazali, kako ta dva prostora zavisita od $\pi_{k=1}^{s}(Y_{kj}^{(x)} \times \pi_{k=0}^{s} Y_{kj}^{(u)})$, Y_j in $\pi_{k=0}^{s} Y_{kj}^{(x)} \times \pi_{k=1}^{s} Y_{kj}^{(u)}$, ki so preko operatorja tipa G povezani s prostori U_Σ, Z_Σ in X_Σ. Analogno kot prej, imamo torej dohodkovno-transformirani proizvodni sistem podan kot

$$\hat{S}_{\tau,\bar{\tau}_\Sigma} = (S_0, \dots S_s, R, \bar{\tau}_\Sigma) \qquad (16)$$

ki je definiran na podmnožici $(Y^{(x)}_{\Sigma,o}, Y^{(u)}_{\Sigma,o})$ celotne množice definicijskih prostorov proizvodnega sistema, če informacijsko ozadje pripravljamo za analizo upravljivosti proizvodnegi sistema samo z ozirom na $Y^{(x)}_{\Sigma,o}$, $Y^{(u)}_{\Sigma,o}$ in π_Σ. Ta vrsta upravljivosti je najbolj pogosta v praktičnih primerih za naše gospodarstvo; kadar pa gre za močnejše vzvode optimizacije, kakršne terja sanacija gospodarjenja, procesi prestrukturiranja itd., pa je treba definirati celotno definicijsko področje proizvodnega sistema. Ta dva primera v nadaljnjem ne bomo formalno ločevali in bomo torej splošneje predstavljali dohodkovni transformat proizvodnega sistema v obliki

$$\hat{S}_{\xi,\, \Phi_\Sigma - C_\Sigma} = (S_0, \dots S_s; R, \Phi_\Sigma, C_\Sigma, \hat{\Phi}_\Sigma) \qquad (16')$$

Tako smo torej prišli do naslednjega spoznanja: dohodkovna transformacija deluje nad manjšim prostorom iz množice konstituant proizvodnega sisitema ter je zato tudi operator Φ_Σ enostavnejši. Informacijsko ozadje dohodkovno transformiranih proizvodnih sistemov je torej še skromnejše kot pa informacijsko ozadje stroškovno transformiranih proizvodnih sistemov. To pa nadalje pomeni, da v primeru (16') pričakujemo vse slabše efekte upravljanja, kot so v primeru (13). Vprašanje upravljivosti sistema (16') glede na funkcional Φ_Σ je analogno prejšnjemu vprašanju, in je treba prav tako upoštevati primere a), b) in c), vendar dodatno še primer d): Φ_Σ je lahko prostor upravljalskih parametrov in to celo v kombinaciji z ostalimi prostori, pri katerih je $\hat{S}_\Sigma$ definiran. To pa je osnova za najbolj splošen primer upravljivosti dohodkovno transformiranega proizvodnega sistema.

Viri:

1) Viljem Rupnik, Eksistenčni izreki multikriterialne upravljivosti dinamičnih sistemov; Ekonomska revija, 1981, št. 3-4.

2) Viljem Rupnik, Matematična teorija sistemov; RCEF, Ljubljana, 1977.

Deskriptorji: SISTEM PORAZDELJENI, VODENJE, UPRAVLJANJE, SISTEM OPERACIJSKI

Jože Rugelj
IJS Ljubljana

POVZETEK Upravljanje omogoča ohranjanje konsistentnosti sistema in pomaga uporabnikom pri delu s sistemom. Pomen upravljalskih aktivnosti se povečuje z naraščanjem zmogljivosti in kompleksnosti sistemov in z njihovo porazdeljenostjo. Najpomembnejše upravljalske funkcije so upravljanje konfiguracije, spremljanje delovanja sistema, ravnanje ob odpovedih, varnostne in zaščitne aktivnosti, obračunavanje in upravljanje z imeni.

ABSTRACT Management enables keeping the consistency of a system and facilitates the usage of a system. The importance of management activities increases with the growing of performances and complexity of the systems and with their distribution. The most important management functions are configuration management, monitoring, fault handling, protection and security activities, accounting and name management.

Upravljanje porazdeljenih sistemov vključuje mnogo vidikov porazdeljenega procesiranja in je v enakem odnosu do porazdeljenih aplikacij kot klasičen operacijski sistem do lokalnih aplikacij [COST83]. Zato pogosto imenujemo nabor upravljalskih funkcij za porazdeljen sistem porazdeljen operacijski sistem. Najbolj splošno bi lahko kot cilj upravljanja porazdeljenega sistema opredelili ohranjanje sistema v stanju, ki ga zahtevajo uporabniki, in to s čimmanjšimi stroški. To pomeni, da sistem omogoča uporabnikom in upravljalcem sistema načrtovanje, organiziranje, nadzorovanje in vodenje uporabe porazdeljenih virov informacijskega procesiranja.

1 Cilji upravljanja

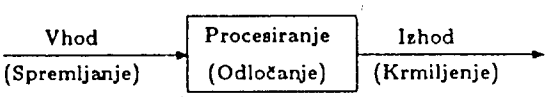
Upravljanje sistema se izvaja v skladu z upravljalsko politiko, ki določa osnovna pravila v zvezi z upravljalskimi odločitvami [Slom87]. Problemi, s katerimi se ukvarja upravljalska politika, so komercialne in tehnične narave. V podjetjih je njen cilj maksimizacija prihodkov, v izobraževalnih ustanovah minimizacija stroškov in pri vojaških projektih zagotovljena visoka zanesljivost. Upravljalsko politiko uporabljamo za določitev strategije in taktike pri upravljanju. Strategija je dolgoročen načrt, kaj je treba storiti za uresničenje ciljev, taktika pa nam pove, kako bomo to storili.

Upravljalske aktivnosti bi lahko razdelili na

- dolgoročne
- operativne.

Dolgoročne odločitve se nanašajo predvsem na načrtovanje razvoja in širjenja porazdeljenega sistema, pa tudi na strategije optimizacije uporabe virov, poimenovanja virov, varnosti in zanesljivosti sistema. Take dolgoročne odločitve sprejemajo *ljudje, ki vodijo in upravljajo porazdeljen sistem*. V primeru tesno sklopljenih sistemov, ki jih obravnavamo v tem delu, pri tem ni posebnih težav, saj je lastnik porazdeljenega sistema ena organizacija. Zato ne prihaja do konfliktov pri reševanju teh problemov tako kot pri šibko sklopljenih sistemih.

Operativno upravljanje porazdeljenega sistema se izvaja v času delovanja sistema, torej sočasno z izvajanjem uporabniških programov. Operativno upravljanje skrbi za optimizacijo izrabe virov v sistemu in možnost spreminjanja konfiguracije sistema, za reševanje iz napak, za varnost sistema in za obračunavanje stroškov za uporabo virov. Večina teh funkcij je soodvisna med seboj. Odločitve morajo biti zelo hitre, v nekaj desetinkah ali tisočinkah sekunde in zato je skoraj nemogoče, da bi pri njih sodeloval človek. Upravljalske aktivnosti so zato izvedene kot strežniki znotraj sistema. Pogosto upravljalske aktivnosti niso eksplicitno ločene in delujejo kot vključene komponente v standardnih strežnikih. Spremljajo delovanje posameznih komponent in sproti ocenjujejo njihovo delovanje. V primeru, ko delovanje komponente ne ustreza pričakovanjem, strežniki upravljalci izvedejo predvidene operacije [COST83]. Slika .1 prikazuje splošno shemo upravljalske operacije.



Slika .1: Shema upravljalske operacije

Samo v skrajnih primerih ali na zahtevo upravljalske funkcije javljajo rezultate svoje aktivnosti človeku, upravljalcu in uporabniku sistema.

Oblika in obseg informacije, ki jo pripravi upravljalska aktivnost, je odvisna od tega, komu je namenjena. Uporabnika predvsem zanimajo osnovne informacije o stanju sistema, ki naj bodo kratke in pregledne. Za upravljalce, ki skrbijo tudi za vzdrževanje sistema in dolgoročno upravljanje, pa so pomembne vse podrobnosti.

2 Zgodovinski pregled upravljalskih pristopov v računalniških sistemih

Glavna cilja upravljanja računalniških sistemov sta čimboljša izraba računalniške opreme in čimbolj enostaven vmesnik proti

3.2 Spremljanje delovanja sistema

Spremljanje delovanja sistema je povezano z delovanjem vseh upravljalskih funkcij, saj služi kot vhodni podatek za proces odločanja in izvajanja upravljalskih aktivnosti. Ta funkcija upravljanja torej nudi servis ostalim upravljalskim funkcijam, redkeje pa neposredno uporabniku. Poseben pomen imajo rezultati spremljanja delovanja sistema za ravnanje ob odpovedih in reševanje iz napak. Funkcija spremljanja delovanja sistema je lahko realizirana centralizirano ali porazdeljeno. Tudi pri tej funkciji imata ena in druga rešitev dobre in slabe lastnosti. Centralizirana rešitev je enostavnejša, a manj zanesljiva in prilagodljiva, porazdeljena pa ima komplementarne lastnosti. Pri obeh se pojavlja problem konsistentnosti stanja sistema, ki je posledica nedeterminizma pri prenosu sporočil po komunikacijski mreži in sinhronizacijskih problemov, ki sledijo iz tega. Pri spremljanju delovanja se zbere velike količine podatkov in strežnik, ki opravlja funkcijo spremljanja, mora te podatke zbrati in urediti tako, da nudijo ustrezno informacijo o delovanju sistema.

3.3 Ravnanje pri odpovedih

Ravnanje pri odpovedih je upravljalska funkcija, ki nudi servis strežnikom in uporabnikom porazdeljenega sistema, ko pride do odpovedi ali nepravilnega delovanja ene ali več komponent v sistemu. Posledica nenormalnega stanja sistema ali kateregakoli njegovega dela se pokaže v obliki napak pri izvajanju operacij pri nekaterih strežnikih v sistemu. Ravnanje pri odpovedih vključuje zaznavo napak in njihovo analizo, iskanje okvarjenih komponent, odpravo okvare ali ustrezno zamenjavo komponente, odpravljane posledic napak in obvestila o napaki upravljalcem sistema in po potrebi uporabnikom. V porazdeljenem sistemu so problemi ravnanja z odpovedmi še večji, saj se napake zaradi sodelovanja med porazdeljenimi komponentami hitro širijo, nadzor in odpravljanje posledic pa sta zaradi porazdeljenosti težja [LeLa81].

Pri pregledu aktivnosti se bomo posvetili tistim servisom, ki jih v primeru odpovedi nudijo strežniki v sistemu.

Odkrivanje napak

Če hočemo odkrivati napake v sistemu med delovanjem, moramo vnaprej poznati vsaj približno obnašanje sistema. Tako lahko v vsakem trenutku primerjamo dejansko stanje sistema s predvidenim in če se pojavi odstopanje, to pomeni napako. Ker pa v splošnem stanj sistema ne poznamo vnaprej, dodamo posebne komponente strojne in programske opreme, ki izvajajo različne teste ob začetku delovanja sistema in med samim delovanjem, periodično ali ob posebnih pogojih. Rezultate teh testov pri pravilno delujočem sistemu poznamo in zato lahko odkrijemo odpovedi in nepravilnosti, če se pojavijo. Taki dodatni testi sicer zmanjšujejo zmogljivost sistema, vendar je to cena, ki jo moramo plačati za povečano zanesljivost delovanja. Pogostost in natančnost testiranja je odvisna od zahtev po zanesljivosti. Tudi zmanjšanje zmogljivosti sistema je lahko znak za napake v sistemu, vendar je take značilnosti razmeroma težko formalno opisati in zato tudi niso primerne za uporabo pri sprotni diagnostiki.

Analiza napak in odkrivanje okvarjenih komponent

Analiza rezultatov testov pomaga pri odkrivanju napak in vzrokov, zaradi katerih je prišlo do napak. V porazdeljenih sistemih je težko določiti izvor napake, saj se rezultati prenašajo med posameznimi strežniki, ki so na različnih lokacijah. Napako lahko zaradi premalo pogostega testiranja spregledamo in jo zaznamo šele, ko se je že razširila po sistemu. Zato kljub očitni napaki težko odkrijemo njen izvor. Sodelovanje strežnikov, ki se ukvarjajo z analizo napak in so porazdeljeni po sistemu, omogoča analizo dogajanj v celotnem sistemu.

Popravilo ali zamenjava okvarjene komponente

Ko odkrijemo vzrok za napako in s tem tudi komponento, ki ne deluje pravilno, najprej tako komponento izločimo, da ne bi povzročala dodatnih težav. To storimo tako, da o napaki obvestimo upravljalca konfiguracije. Potem poskušamo tako komponento delno ali v celoti popraviti s servisi, ki so na razpolago v sistemu. Če tako popravilo uspe, komponento vrnemo v sistem. Pri komponentah, ki so bolj kompleksne, je možno tudi delno popravilo. V takih primerih komponenta izvaja samo omejeno število funkcij. Kadar okvare ne moremo odpraviti s servisi, ki so na razpolago v sistemu, ali jo lahko le delno odpravimo, o tem obvestimo upravljalca sistema; le-ta o okvari obvesti vzdrževalce programske ali strojne opreme in po potrebi tudi uporabnika. To je potrebno predvsem v primerih, ko servis zaradi odpovedi komponent ne more nuditi predvidenega servisa, saj določeni viri niso replicirani. Če se zaradi odpovedi posamezne komponente delo porazdeli med druge in to pomeni samo zmanjšanje zmogljivosti sistema, to običajno ni usodno za uporabnika in ga zato ni treba vznemirjati s poročili o odpovedih.

Odprava posledic odpovedi

Odpravljanje posledic odpovedi po odstranitvi ali popravilu okvarjene komponente je osredotočeno predvsem na razveljavitev rezultatov, ki vsebujejo napake, posledice odpovedi. Kompleksnost problema lahko zmanjšamo z uvedbo principa nedeljivih akcij. To so segmenti programske opreme, ki se izvedejo skupaj. Poznamo vse povezave, ki jih ima komponenta, ki vsebuje tak segment, znotraj segmenta z drugimi komponentami. Takoj po izvedbi tega segmenta izvedemo potrebne teste in v primeru napake razveljavimo dobljene rezultate. Ker poznamo vse povezave z drugimi komponentami in je le-teh omejeno število, je to dokaj preprosto. V končni fazi postavimo vse komponente v stanje, v kateri so bile, tik preden se je pojavila napaka. S tem smo odpravili posledice odpovedi.

3.4 Varnost in zaščita

V vsakem računalniškem sistemu morajo obstajati metode in mehanizmi za zaščito vseh objektov v sistemu. S tem je vsakemu uporabniku zagotovljena varnost. To pomeni, da so podatki in servisi dostopni samo pooblaščenim strežnikom in uporabnikom, da so servisi, ki jih nudijo strežniki, ustrezni, ter da vedno poznamo identiteto uporabnikov servisov. To potrebujemo zaradi nadzora delovanja sistema in analize ob zlorabi in za obračunavanje.

Zaščita objektov pomeni nadzor dosega do le-teh. Objekti v sistemu morajo biti zaščiteni pred nepooblaščenimi uporabniki in pred nenadzorovano uporabo zaradi napak in odpovedi v programski ali strojni opremi. Pri problemih zaščite v računalniških sistemih je treba ugotoviti, komu lahko zaupamo. V centraliziranih sistemih so bili mehanizmi za zaščito zbrani v jedru operacijskega sistema. Jedro je bilo namreč ustrezno zaščiteno pred ostalimi deli sistema in so mu lahko vsi zaupali.

Pri porazdeljenih sistemih sta dva problema, ki narekujeta drugačno reševanje. Ker so jedra operacijskega sistema porazdeljena v sistemu, zelo lahko pride do poskusov spreminjanja le-teh in do nedovoljenih operacij v sistemu. Drugi razlog za

uporabniku.

V teku razvoja sistemov so se načini in možnosti za zagotavljanje teh ciljev spreminjali. Pri prvih računalnikih so bile procesne in pomnilniške zmogljivosti zelo drage. Zato si ni bilo mogoče predstavljati, da bi računalnik sam izvajal kakršnekoli upravljalske funkcije. Uporabnikov je bilo zelo malo in vsak je bil hkrati programer, operater in upravljalec sistema. Sam je poskrbel za nalaganje programov in podatkov, za izvajanje programa in za izpis rezultatov. Ko so se zmogljivosti računalnikov povečevale in se je tudi cena procesiranja zniževala, je bilo uporabnikov vedno več. Računalniki so že imeli enostavne upravljalske mehanizme, ki so se imenovali paketni sistemi. Le-ti so skrbeli za zaporedno nalaganje programov in podatkov, za izvajanje programov in za shranjevanje rezultatov. Vsak program se je izvedel do konca in šele potem je sistem naložil nov program. Tak sistem je bistveno izboljšal uporabo virov v sistemu, saj je bilo mnogo manj izgube časa med posameznimi posli kot prej pri ročnem upravljanju. Naslednja težava, ki je onemogočala boljšo izrabo zmogljivosti, je bila razlika v hitrosti delovanja procesnih enot in vhodno-izhodnih enot. Zato so uvedli tako imenovano istočasno računanje in vhod-izhod, dobro poznano po začetnicah angleškega opisa te dejavnosti kot SPOOLer (simultaneous peripheral operation on line). To je bila prva oblika kvazi-paralelnega izvajanja več dejavnosti, ki je zahtevala določene zmožnosti v strojni opremi. Tu mislimo predvsem na prekinitvene mehanizme. Na ta način je računalnik prevzel skrb za upravljanje mnogih virov in v večji meri omogočil doseganje zgoraj opredeljenih ciljev.

Naslednji korak, ki je računalnik še bolj približal uporabniku, je bil uporaba operacijskih sistemov z dodeljevanjem časa (time sharing). Tak operacijski sistem hkrati več uporabnikom omogoča interaktivno delo. Za razliko od paketnega sistema se tu program praviloma ne izvrši naenkrat. Uporabnik dobi pravico do uporabe virov v sistemu samo za kratke intervale, vendar ima zaradi velike hitrosti delovanja občutek, kot da uporablja računalnik sam. To seveda velja, če sistem ni preobremenjen. Viri v sistemu so dobro izkoriščeni, vendar zahteva režijsko delo pri menjanju dejavnosti precej časa.

Čim boljši so bili računalniki in čim večje so bile zmogljivosti, več uporabnikov jih je uporabljalo in pojavljale so se nove zahteve. Z optimizacijo algoritmov in programov in z novimi tehnološkimi rešitvami so nekaj časa uspeli zadovoljevati zahteve. Bolj dolgoročna rešitev pa je porazdelitev procesiranja in paralelnost.

Prvi problem, ki so se ga lotili strokovnjaki, je bilo upravljanje komunikacij. Prenos podatkov med porazdeljenimi procesnimi mesti mora biti zanesljiv in brez napak. Vsako procesno mesto je imelo svoj lokalni operacijski sistem. Lokalni operacijski sistem uporablja servise komunikacijskega sistema in nudi servise uporabniku. Razen zanesljive komunikacije v teh zgodnjih oblikah porazdeljenih sistemov ni bilo posebnih pripomočkov, ki bi uporabniku pomagali pri delu v porazdeljenem sistemu [Fort86].

Uporabniki pa so potrebovali mehanizme, ki bi jim omogočili bolj celovit pogled na porazdeljen sistem in čimvečjo transparentnost porazdeljenosti sistema. *Mrežni operacijski sistem* je bil nov nivo, ki so ga dodali med komunikacijski sistem in lokalni operacijski sistem na vsakem procesnem mestu. Značilnost mrežnega operacijskega sistema je, da vsak uporabnik še vedno v glavnem dela ne enem procesnem mestu, da se zaveda porazdeljenosti objektov v sistemu in da sistem vsebuje zelo malo mehanizmov za reševanje iz napak [Trip87]. Glavna področja, kjer uporabik čuti porazdeljenost so datotečni sistem, zaščita in izvajanje programov [Tane85]. To pomeni, da mora uporabnik še vedno sam reševati množico problemov, povezanih z upravljanjem sistema.

Porazdeljeni operacijski sistem omogoča uporabniku pogled na porazdeljen sistem kot celoto, brez mej med posameznimi procesnimi mesti. Upravljanje se nanaša na vse objekte v sistemu in optimira rabo vseh virov, za razliko od mrežnega operacijskega sistema, kjer gre za optimizacijo delovanja vsakega posameznega procesnega mesta. Porazdeljen operacijski sistem je v enakem odnosu do porazdeljenega računalniškega sistema, kot lokalen operacijski sistem do centraliziranega računalniškega sistema. Porazdeljen operacijski sistem deluje enotno, vendar ima svoje kopije na vseh procesnih mestih, njegove funkcije pa se izvajajo paralelno in med seboj sodelujejo.

3 Upravljalske funkcije

Pri pregledu upravljalskih funkcij se bomo omejili predvsem na tiste, ki so povezane z operativnim upravljanjem porazdeljenih sistemov in so realizirane v porazdeljenih operacijskih sistemih.

3.1 Upravljanje konfiguracije

Porazdeljen računalniški sistem lahko predstavimo kot množico komponent programske in strojne opreme, ki so lahko prostorsko porazdeljene. Vendar pa opisana množica tvori porazdeljen sistem šele, ko so komponente povezane in sodelujejo pri reševanju problemov. Funkcija upravljanja konfiguracije sistema poskrbi za začetno povezavo komponent in vzpostavitev delovanja sistema in tudi za dinamično prilagajanje sistema, ki je potrebno zaradi napak ali spremenjenih potreb uporabnikov.

Določene operacije pri konfiguraciji so dolgoročnega značaja in jih mora opraviti operater. Mislimo predvsem na instalacijo osnovnih komponent programske in strojne opreme in fizično povezavo posameznih komponent sistema. Ko je najbolj osnovna konfiguracija sistema vzpostavljena, lahko nadzor nad konfiguracijo sistema prevzame ustrezen strežnik. Večina ostalih aktivnosti, ki jih vključuje upravljanje konfiguracije, se izvaja dinamično. Tako se razporejajo aktivnosti sistema glede na trenutno zasedenost posameznih virov v sistemu in njihovih sposobnosti za opravljanje posameznih operacij. To je tudi edini način za optimizacijo zmogljivosti, ki se lahko izvaja dinamično. Ostali načini so dolgoročnega značaja in zato ne sodijo v okvir naše obravnave. Tudi v primeru odpovedi komponent se mora konfiguracija sistema spremeniti tako, da uporabniku ostanejo na razpolago vsi servisi, čeprav se zmogljivost sistema zmanjša. Komponente, ki so bistvene za zagotavljanje določenih servisov, ni pa jih mogoče nadomestiti z drugimi komponentami, morajo biti zato replicirane. Poleg skrbi za povezave med posameznimi komponentami sistema mora upravljalska aktivnost v vsakem trenutku posredovati informacije o stanju sistema, če uporabnik to zahteva.

V splošnem pa je zaželjena transparentnost lokacije virov, torej se uporabnik ne ukvarja z iskanjem lokacije vira. Transparentnost lokacije vira je tesno povezana s poimenovanjem virov in semantično konsistentnostjo strežnikov in pomeni, da je njihov servis enak, ne glede na to, kje v sistemu se strežnik nahaja. Večje težave se pri tem pojavijo v porazdeljenih sistemih, ki ga sestavljajo zelo različne komponente strojne ali programske opreme. Transparentnost lokacije virov lahko dosežemo na različnih nivojih, vendar je najbolje, če je realizirana v jedru operacijskega sistema, saj je tako dostopna najširšemu krogu uporabnikov.

realizacijo mehanizmov zaščite izven jedra pa je želja po čim manjšem jedru.

Za zaščito sta v porazdeljenih sistemih uporabljena dva mehanizma: seznam za nadzor dostopa in mehanizem prenašanja pravic.

Seznam za nadzor dostopa do vira vsebuje vse uporabnike, ki imajo pravico dostopa do tega vira. Seznam je pridružen strežniku, ki vsebuje vir. Zahtevek, s katerim uporabnik zahteva servis, vsebuje tudi enolično oznako uporabnika. Ta del sporočila je tako zaščiten, da ga ni mogoče ponarediti. Dobra lastnost mehanizma s seznamom je tudi to, da je mogoče enostavno razveljaviti pravice v zvezi z dostopom, če je to potrebno. To je potrebno ob odpovedih in pojavih napak, pri spremembah konfiguracije sistema ali zaradi optimizacije zmogljivosti sistema.

Osnovni element drugega mehanizma je posebna vstopnica ali žeton, ki ga imenujemo pravica [Tane84]. Le-ta omogoča svojemu lastniku dostop do virov ali izvajanje določenih operacij, ki so vključene v dani strežnik. Uporabnik, ki želi generirati nov objekt, pošlje zahtevek za to ustreznemu strežniku. Zahtevek vsebuje tudi neko naključno število iz določenega intervala. Strežnik generira nov objekt, hkrati pa zgenerira tudi pravico za dostop do tega objekta. S pomočjo naključnega števila, ki ga je poslal uporabnik, šifrira pravico in jo vrne uporabniku. Le-ta jo lahko potem razmnoži in jo preda tudi drugim strežnikom in uporabnikom. Strežnik shrani naključno število v posebno tabelo, povezano z novim objektom. Ko pride zahtevek za doseg novega objekta, strežnik preveri pristnost pravice s številom, shranjenim v tabeli. Pravica je zaščitena tako, da je ni mogoče ponarediti. Pravice je težko razveljaviti in to je slaba lastnost tega mehanizma. Ko strežnik, ki je odgovoren za podani objekt, generira pravico, jo šifrira z naključnim številom. To pomeni, da uporabnik, ki je zahteval kreiranje novega objekta, sam ne more spremeniti vsebine pravice in omejiti pravice uporabe posameznih operacij nad objektom uporabniku, ki bi mu želel dati kopijo pravice. To stori lahko le tako, da pošlje zahtevek za kreiranje take pravice strežniku.

Opisane lastnosti mehanizma s pravicami omogočajo porazdeljeno realizacijo zaščite. Zaradi porazdeljenosti sistema potrebujemo metode za šifriranje sporočil, ki onemogočajo prisluškovanje na komunikacijskih medijih, in overjanje sporočil s tako imenovanim digitalnim podpisom. Najbolj je razširjeno šifriranje po tako imenovanem DES (Data Encryption Standard) standardu. Oba osebka, ki komunicirata, morata poznati ključ, to je neko dovolj veliko število, ki služi kot osnova za šifriranje in dešifriranje sporočil. Metoda šifriranja je poznana, to pomeni, da je vsa skrivnost v ključu. Funkcije za šifriranje sporočil so realizirane v siliciju in zato je šifriranje v realnem času mogoče tudi za velike hitrosti prenosa podatkov, ki so potrebne v porazdeljenih sistemih. Problem predstavlja samo generacija in distribucija ključev. Zato sta Diffie in Hellman razvila nov sistem, imenovan sistem šifriranja z javnim ključem. Poznana sta sistem šifriranja in dešifriranja in tudi ključ za šifriranje. Ključa za šifriranje in dešifriranje sta pridobljena iz istega osnovnega ključa. Vsa skrivnost sistema je v posebnosti funkcije, s katero iz osnovnega ključa dobimo ključ za šifriranje. Ta funkcija je zelo enostavno izračunljiva, njena inverzna funkcija pa ni izračunljiva v realnem času. Zato ni mogoče učinkovito izračunati osnovnega ključa in iz tega ključa za dešifriranje.

3.5 Obračunavanje

Obračunavanje je sistemska funkcija, ki spremlja in beleži uporabo virov in servisov v sistemu. Poleg sestavljanja računov na osnovi pogodb in tarif za posamezne servise, ta funkcija omogoča tudi omejevanje uporabe virov in servisov in zagotavlja poštenost pri uporabi le-teh. V tesno sklopljenih sistemih, ki so običajno v lasti ene same organizacije in kjer se stroški pogosto ne delijo med uporabnike, prevladujejo predvsem ti drugi motivi.

Tarife za uporabo posameznih servisov se določijo na osnovi stalnih stroškov, spremenljivih stroškov in posebnih stroškov. Stalni stroški se plačujejo periodično in so neodvisni od uporabe sistema. Posebni stroški so odvisni od časa uporabe ali količini obdelanih in prenešenih podatkov, kvalitete servisa in oddaljenosti strežnika. Posebni stroški so povezani s servisi, ki niso vključeni v osnovni nabor [Slom87].

Pravice za uporabo posameznih virov lahko izrazimo z virtualnim denarjem, ki ga ima uporabnik [COST84]. Z virtualnim denarjem si kupi pravice za uporabo virov oz. servisov. Virtualni denar lahko uporabnik dobi na osnovi dogovora med uporabniki ali kot protivrednost denarja, ki ga plača za uporabo sistema. Virtualni denar je v različnih valutah, za vsak vir ali servis druga valuta.

Vse operacije z denarjem so tesno povezane z uporabo mehanizmov za varnost in zaščito.

3.6 Upravljanje z imeni

Imena imajo v računalniških sistemih zelo pomembno vlogo, saj z njihovo pomočjo identificiramo izbrane objekte v sistemu na vseh nivojih abstrakcije. Imena uporabljamo na različnih področjih upravljanja. Še posebno pomembna je vloga imenskega sistema v tesno sklopljenih računalniških sistemih zaradi dovoljene raznolikosti gradnikov sistema in želje, da bi uporabnik videl sistem kot enovito celoto.

Zaradi čimboljše prilagodljivosti sistema želimo, da ime vsebuje čim manj informacij o objektih, saj se lahko le-te dinamično spreminjajo [Terr86]. Te informacije morajo zato biti shranjene neodvisno in organizirane tako, da je spreminjanje in dodajanje podatkov o objektih enostavno, hkrati pa je enostavna tudi povezava z imeni. To namreč omogoča učinkovito delovanje imenskega sistema.

4 Literatura

[COST83] T. Kalin (edt.),
Management issues in Local Area Networks,
COST 11 bis LAN Management Group Report,
Proc. EUTECO '83, North Holland, 1983

[COST84] A. Langsford (edt.),
Distributed Systems Management in Wide Area Networks, report by COST 11 bis DSM Group,
February 1984

[Fort86] P.J. Fortier,
Design of Distributed Operating Systems
McGraw-Hill, New York, 1986

[LeLa81] G. LeLann,
Error recovery, in Distributed systems,
LCNS 105, Springer-Verlag, Berlin 1981

[Slom87] M. Sloman, J. Kramer,
Distributed Systems and Computer Networks,
Prentice-Hall International, 1987

[Tane84] A.S. Tanenbaum et.al.,
Capability Based Protection in Distributed
Operating Systems, Proc. Symp. Certificering
van Software, Utrecht, November 1984

[Tane85] A.S. Tanenbaum, S.J. Mullender,
Distributed Operating Systems,
ACM Computing Surveys, vol.17/4,
December 1985

[Tane87] A.S. Tanenbaum, R. van Renesee,
Reliability Issues in Distributed Operating
Systems, Proc. 6th Symp. Reliability
of Distr. Softw. & Datab. Systems,
Williamsburg, Virginia, March 1987

[Terr86] D.B. Terry,
Structure-free Name Management for
Evolving Distributed Enviroments
IEEE Proc. 6th ICDCS, May 1986

[Trip87] A. Tripathi,
Distributed Operating Systems,
Tutorial no.4, 7.ICDCS,
W.Berlin, September 1987

VERJETNOSTNI MODEL RAČUNANJA

INFORMATICA 1/89

Deskriptorji: RAČUNANJE, VERJETNOSTNI MODEL, ALGORITMI HEVRISTIČNI

Janez Žerovnik
Inštitut za matematiko, fiziko in mehaniko, Ljubljana

POVZETEK: V preglednem članku predstavimo verjetnostni model računanja in podamo definicijo nekaterih razredov časovne zahtevnosti verjetnostnih algoritmov. V začetnih razdelkih vpeljemo klasični (deterministični) model računanja in ponovimo znane definicije iz teorije časovne zahtevnosti algoritmov.

ABSTRACT: A Probabilistic Model of Computation

In article a survey of a probabilistic model of computation is given. Some classes of probabilistic algorithms are defined. Preliminary sections give definition of classical (deterministic) model of computation and introduction to the theory of time complexity of computation.

1. UVOD

V članku bomo vpeljali nekatere pojme in definicije iz teorije časovne zahtevnosti algoritmov. V slovenščini s tega področja poznamo dve deli [Pisa83, PePi82] in prevod [AhUl86], tuja literatura pa je precej bolj bogata. Samo problemu trgovskega potnika na primer je posvečena cela knjiga [LLRS85]. Že klasična je knjiga Gareya in Johnsona [GaJo79], mi pa bomo zaradi novejših definicij nekaterih razredov slučajnih algoritmov sledili knjigi [Melh84], oprli pa se bomo še na [HoUl79] in [Gill77].

V zadnjem desetletju in pol so razred NP–polnih odločitvenih problemov veliko raziskovali. Vprašanje, ali je razred P enak razredu NP je verjetno eden najpomembnejših odprtih problemov teoretičnega računalništva. Teoretično računalništvo tu imenujemo vedo, ki jo nekateri imenujejo tudi informatika (Informatique), kibernetika uporabna matematika (Prikladnaja matematika) ali 'algoritmika' (algorithmics) [Knut81]. Če bi za katerikoli NP–poln problem uspeli pokazati, da je v P, bi sledilo P=NP.

Ker je veliko praktičnih problemov NP–polnih, si v praksi pomagamo do delnih rešitev na različne načine. Aproksimativni algoritmi na primer so taki algoritmi, ki v polinomskem času najdejo rešitev, ki je 'dovolj blizu' optimalne rešitve. Deterministični hevristični * algoritmi imajo pogosto lepo lastnost, da najdejo dobre rešitve na neki podmnožici problemske domene. Običajno imajo taki algoritmi 'Ahilove pete': obstajajo primeri, na katerih se algoritem obnaša zelo slabo. Avtorji algoritmov običajno že sami navajajo primere, ki so neugodni za algoritem [WePo67,DuBr81]. Pri slučajnih hevurističnih algoritmih nekatere korake naredimo slučajno. S tem se včasih izognemo 'Ahilovim petam' na račun nekaj večje časovne zahtevnosti.

Eden od intuitivno najlažje razumljivih NP-polnih problemov je problem barvanja točk grafa. Ker je sestavek nekoliko predelano poglavje avtorjeve magistrske naloge 'Verjetnostni algoritem za barvanje točk grafa', je tudi tu najpogosteje obravnavani primer ravno problem barvanja točk grafa. *Graf* je kombinatorični objekt, ki ga sestavlja ta množica točk V in množica E, v kateri so pari točk. Elementom množice E pravimo *povezave*. Točki, ki sta povezani s povezavo, sta *sosednji*. *Barvanje* (točk grafa) je prireditev barv točkam grafa. Za barve običajno vzamemo kar naravna števila. Barvanje je *dobro*, če je vsak par sosednjih točk pobarvan z različnima barvama. Formalno to zapišemo na primer takole: $b : V \to N$ je dobro barvanje $\iff (u, v) \in E \implies b(u) \neq b(v)$. Odločitveni problem barvanja grafa lahko zdaj zastavimo takole: Ali za dani graf G in množico barv $\{1, \ldots, k\}$ obstaja dobro barvanje? Če tako barvanje obstaja, rečemo, da je G *k-pobarvljiv*. Pogosto je zanimiva optimizacijska varianta problema: Poišči minimalni k, da je G še k-pobarvljiv! Ta minimalni k imenujemo *kromatično število* grafa G. V nadaljevanju bomo predpostavljali, da bralec pozna osnovne pojme teorije grafov, kot so vpeljani na primer v [BaPi85] ali [CvMi77].

Problem barvanja grafa je zanimiv tudi iz zgodovinskih razlogov. Problem štirih barv je bil skoraj sto let velika uganka, ob kateri se je razvijala teorija grafov. Problem si je prvi zastavil leta 1852 londonski študent Francis Guthrie, ko je barval zemljevid angleških grofij. Domneval je, da je mogoče vsak zemljevid pobarvati s štirimi barvami, seveda tako, da ozemlja, ki mejijo, niso pobarvana z isto barvo. Pri tem ozemlja, ki se dotikajo samo v končno mnogo točkah ne štejemo za sosednja. Problem običajno zastavimo v nekoliko drugačni obliki. Namesto ozemelj barvamo točke (lahko si mislimo, da so to glavna mesta), povezani pa sta točki, ki sta v sosednjih državah. Problem se potem glasi: ali je mogoče točke vsakega planarnega grafa pobarvati s štirimi barvami (tako da pobarvamo poljubni povezani točki z različnima barvama)? Francisov

* Splošno privzete definicije hevurističnega algoritma ni [Tros83]. Po Verbinčevem Slovarju tujk je hevristika: 'nauk o metodah raziskovanja (z uporabljanjem še nedokazanih metod, hipotez, itd.)'. Za našo rabo naj hevristični algoritem pomeni postopek, ki ga uporabljamo, čeprav nimamo dokaza, da so njegove rešitve vedno točne (optimalne). Vemo pa, da na primer velja, da je točen na neki podmnožici problemske domene ali da točno rešuje nek 'podoben' problem ali kakšen podoben delni rezultat.

brat Frederic je problem predstavil profesorju Augustu de Morganu. Širše znan je problem postal leta 1878, ko je Arthur Cayley na srečanju Londonske matematične družbe (London Math. Society) vprašal, ali je problem že rešen.

Prvi se je reševanja problema resno lotil A.B.Kempe, ki je leta 1879 objavil dokaz, da štiri barve zadoščajo. Deset let kasneje je P.J.Heawood odkril napako v Kempejevem dokazu, ki pa so jo mnogi imeli bolj za tehnično pomanjkljivost kot pa za resno napako. Ko pa so leta minevala in nikomur ni uspelo popraviti dokaza, je postalo jasno, da problem ni od muh. Leta 1976 sta Kenneth Appel in Wolfgang Haken objavila, da sta dokazala izrek o štirih barvah. Ideja dokaza je v bistvu enaka Kempejevi, le število posebnih primerov, ki jih je bilo potrebno pregledati, je precej večje (okoli 1500 namesto 5!). Obsežnost je tudi glavna hiba dokaza, saj je za pregled (okoli 600 strani) dolgega dokaza potrebno ogromno časa, tudi če za nekatera rutinska opravila uporabimo računalnik, kot sta to storila avtorja dokaza [WoWi78,ApHa86].

2. Primer NP–polnega problema

Za motivacijo si v tem razdelku še pred formalnimi definicijami oglejmo problem klike (skupka). Zastavimo ga takole: Dan je (neusmerjen) graf $G = (V, E)$ in naravno število k. Ali v G obstaja podgraf, ki je izomorfen polnemu grafu K_k? Obstaja enostaven, toda neučinkovit algoritem za reševanje tega problema. Pregledamo vse podgrafe v G, ki so inducirani z množicami točk kardinalnosti k. Za vsak tak podgraf pa preverimo, če je poln. Med n točkami lahko izberemo k točk na $\binom{n}{k}$ načinov. V primeru $k = n/2$ torej naš algoritem preveri $\binom{n}{n/2} \geq 2^n/(n+1)$ podmnožic. Preverjanje, ali je graf poln, je enostavno in hitro: za vsako od $n(n-1)/2$ povezav pogledamo, ali je v induciranem grafu. Naš naivni algoritem torej zahteva čas, ki je eksponentna funkcija števila točk danega grafa. Preden trdimo, da je to neučinkovito, povejmo, kaj mislimo z velikostjo problema. Predpostavljali bomo, da so kombinatorični objekti (grafi, množice, cela števila,...) podani v 'normalni obliki' s končno abecedo. Tako bo na primer graf z n točkami zakodiran v obliki zaporedja n^2 bitov, v katerem so zložene vrstice matrike sosednosti. Cela števila zapišemo v binarnem zapisu in množice z zaporedjem elementov v nekakšnem redu. Primer problema klike na grafu z n točkami je potem zaporedje bitov dolžine $m := n^2 + \log(n/2) + 1$, če spet zaradi enostavnosti vzamemo primer $k = n/2$. Naš naivni algoritem torej razpoznava jezik CLIQUE $= \{wv; w, v \in \{0,1\}^* \text{ in } |w| = n^2$ za neki n in graf G z matriko sosednosti w ima kliko velikosti k, kjer je v binarni zapis števila $k\}$ v času $O(2\sqrt{m})$, kjer je m velikost problema (dolžina vhoda). Algoritma, ki bi bil bistveno boljši od opisanega, ne poznamo. Ker je problem klike NP–poln, je zelo malo verjetno, da tak algoritem sploh obstaja. To bi namreč pomenilo, da obstajajo učinkoviti (polinomski, kot bomo kasneje definirali) algoritmi za vse NP–polne probleme, na primer za problem trgovskega potnika in problem izpolnjivosti. Splošno je privzeta hipoteza 'P≠NP', da takih algoritmov ni. Dokaz ali protidokaz te hipoteze je verjetno najbolj znan odprt problem teoretičnega računalništva.

Še malo se zadržimo pri problemu klike. Podmnožic kardinalnosti k med n točkami je $\binom{n}{k}$, torej obstaja zelo veliko kandidatov za kliko. Po drugi strani je zelo lahko preveriti, ali je dani kandidat klika ali ne. Edina znana pot pa je pregledati vse kandidate.

Če bi imeli na razpolago nedeterministični ukaz CHOICE, bi lahko problem rešili učinkoviteje. Ukaz bi bil na primer oblike

CHOICE $Naslov_1$, $Naslov_2$

Z nedeterminizmom je mišljeno, da izbira nadaljevanja nima nobene vnaprej določene verjetnosti. Ukaz CHOICE pač pomeni, da se algoritem lahko nadaljuje na naslovu $Naslov_1$ ali pa na naslovu $Naslov_2$. Algoritme z ukazom CHOICE imenujemo nedeterministične. Tak algoritem razpozna vhodno zaporedje, če obstaja vsaj ena izvedba algoritma, ki razpozna vhod. Časovna zahtevnost nedeterminističnega algoritma je čas najkrajše izvedbe, ki je razpoznala vhod.

Za primer poglejmo, kako bi problem klike rešili z nedeterminističnim algoritmom. Nedeterminizem bomo uporabili pri generiranju kandidatov za kliko. Algoritem je sestavljen iz treh faz. Najprej pregledamo vhodno zaporedje bitov in preverimo, ali je vprašanje dobro postavljeno. Če vhod ni oblike wv in $|w|$ ni kvadrat naravnega števila, ga zavrnemo, sicer pa izračunamo $n = \sqrt{|w|}$ in k. Z $|w|$ smo označili dolžino zaporedja znakov (v tem primeru znakov iz abecede $\{0,1\}$) w. Časovno zahtevnost prve faze lahko ocenimo z $O(n^4)$. V drugi fazi nedeterministično izberemo podmnožico V kardinalnosti k, tako da zgeneriramo zaporedje n bitov z natanko k enicami. To naredimo takole:

```
for i := 1 to n
do CHOICE m1, m2
   m1 : A[i] := 0;
   m2 : A[i] := 1; k := k − 1;
od
if k ≠ 0 then zavrni;
```

Uspešna izvedba gornjega dela algoritma generira eno od $\binom{n}{k}$ podmnožic V. Potrebni čas je očitno reda $O(n)$. V tretji fazi preverimo, ali je izbrana podmnožica klika. Tudi to lahko storimo hitro, vsaj $O(n^4)$. Opisani algoritem potrebuje čas $O(n^4) = O(m^2)$, kjer je m dolžina vhodnega zaporedja. Podani algoritem res razpoznava jezik CLIQUE: Če G ne vsebuje klike velikosti k, potem v drugi fazi zgenerirana podmnožica gotovo ni klika, torej izvedbe, ki bi sprejela vhod, ni. Če pa graf G ima kliko velikosti k, potem v eni od mogočih izvedb algoritem zgenerira prav to kliko. Ta izvedba seveda sprejme vhod.

Videli smo, da lahko problem klike rešimo v polinomskem času z nedeterminističnim algoritmom. Razred problemov, ki so rešljivi v polinomskem času z nedeterminističnim algoritmom označimo z NP. S P pa označimo razred problemov, rešljivih v polinomskem času z determinističnim algoritmom. Med NP problemi posebej obstajajo problemi, na katere je mogoče prevesti (v polinomskem času) vsak drug NP problem. To so NP–polni problemi. Cook je v članku [Cook71] je pokazal, da je problem izpolnjivosti (SAT) NP–poln. S polinomskim prevajanjem pa je bilo od tedaj za veliko problemov dokazano, da so tudi NP–polni, med drugimi tudi problem klike in problem barvanja točk grafa [Karp72]. Obsežni seznami NP–polnih problemov so v knjigah, na primer v [GaJo79].

Za strogo obravnavo opisanih problemov potrebujemo formalno definiran model računanja, ki ga podajamo v naslednjem razdelku.

3. Turingov stroj

V tem razdelku bomo definirali model računanja s pomočjo katerega bomo lahko strogo definirali časovno zahtevnost algoritmov in razdelili probleme glede na težavnost. Model računanja, ki ga običajno izberemo, je Turingov stroj (TS), saj kljub enostavnosti ni nič šibkejši od katerega koli doslej zgrajenega 'superračunalnika'. Vsi znani modeli so namreč polinomsko prevedljivi na model TS, kar pomeni, da so v smislu teorije, ki nas tu zanima, ekvivalentni. Če drži Churcheva teza, pa to velja za vse mogoče modele računanja! Naša definicija se rahlo razlikuje od tistih v [Pisa83] in [PePi82], vendar nas to ne moti zaradi znane ekvivalentnosti modelov računanja [PePi82, GaJo79, Melh84, AhHU76, HoUl86].

Turingov stroj ima dve enoti: kontrolno in spominsko. Spomin je v desno stran neomejen trak, ki je razdeljen na kvadratke, v katerega je mogoče zapisati en znak končne abecede, ki jo uporablja konkretni Turingov stroj. Kontrolna enota ima končno mnogo stanj in lahko naenkrat dosega en znak (kvadratek) na traku. Turingov stroj programiramo s tabelo, v kateri za

vsako stanje in vsak znak abecede (v trenutno vidnem kvadratku) določimo množico možnih operacij. Če je množica prazna, se TS ustavi. Če je Turingov stroj deterministično ima množica možnih operacij v vsakem primeru največ en element, obnašanje deterministočnega Turingovega stroja je s tabelo in vhodnim podatkom torej natanko določeno. TS v eni operaciji lahko naredi troje: lahko zapiše znak na trenutno vidni kvadratek traku, lahko premakne 'oko' v levo ali desno in lahko preide v novo stanje.

TS poženemo tako, da zapišemo vhodno zaporedje znakov na prvih n kvadratkov, postavimo 'oko' na prvi (skrajnje levi) kvadratek traku in postavimo TS v (vedno isto) začetno stanje. Vsi drugi kvadratki na traku so v začetku prazni, vsebujejo torej posebni znak abecede B. TS izvaja operacije, dokler ne naleti na par (znak, stanje) za katerega v tabeli nima nobene operacije in se ustavi. Rezultat računanja je neprazni del traku. Na ta način lahko TS uporabimo za računanje funkcij. Pogosteje pa rabimo TS za razpoznavanje jezikov. TS razpoznava jezik, če izračuna njegovo karakteristično funkcijo. Da se izognemo (nepotrebnemu) brisanju traku pa se dogovorimo, da odlikujemo nekaj stanj in rečemo: če se Turingov stroj ustavi v katerem od teh stanj, je razpoznal vhod za element jezika. Obe definiciji sta ekvivalentni, druga pa je primernejša za obravnavo nedeterminizma.

1.DEFINICIJA: Turingov stroj (TS) je sedmerka $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, kjer je

Q končna množica stanj

Γ tračna abeceda

$B \in \Gamma$ prazen simbol

$\Sigma, \Sigma \subseteq \Gamma \setminus \{B\}$ množica vhodnih simbolov

δ prehodna funkcija $Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$, ki ni nujno definirana na vsej množici $Q \times \Gamma$

$q_0 \in Q$ začetno stanje

$F \subseteq Q$ množica končnih stanj

Zaradi nedvoumnosti privzamemo, da sta Γ in Q disjunktni množici simbolov. Trenutno stanje Turingovega stroja opisuje *konfiguracija*, to je niz oblike $\alpha_1 q \alpha_2 \in \Gamma^* \times Q \times \Gamma^*$, kjer je q trenutno stanje TS, $\alpha_1\alpha_2$ je zapis na traku, TS pa vidi prvi znak niza α_2. Če je $\alpha_2 = \varepsilon$ prazen niz, potem TS po dogovoru vidi prazen znak B.

Definirajmo *korak* TS. Bodi $X_1X_2 \ldots X_{i-1}qX_i \ldots X_n$ konfiguracija. Naj bo $\delta(q, X_i) = (p, Y, L)$, kjer v primeru $i - 1 = n$ vzamemo $X_i = B$. Če je $i = 1$, potem naslednja konfiguracija ni definirana, saj stroj ne sme 'pasti' čez levi rob traku. Če je $i > 1$ potem zapišemo

$$X_1X_2 \ldots X_{i-1}qX_i \ldots X_n \vdash X_1X_2 \ldots X_{i-2}pX_{i-1}YX_{i+1} \ldots X_n$$

Če se nam na koncu zapisa konfiguracije pojavijo prazni simboli, jih po dogovoru zbrišemo.

Podobno definiramo korak, če je $\delta(q, X_i) = (p, Y, R)$.

$$X_1X_2 \ldots X_{i-1}qX_i \ldots X_n \vdash X_1X_2 \ldots X_{i-1}YpX_{i+1} \ldots X_n$$

V primeru $i - 1 = n$ je niz $X_1 \ldots X_n$ je zapis konfiguracije po koraku daljši od zapisa prejšnje konfiguracije.

Konfiguracija $C = X_1X_2 \ldots X_{i-1}qX_i \ldots X_n$ je *ustavitvena*, če velja $\delta(q, X_i) = \emptyset$. Konfiguracija je *sprejemajoča*, če je $q \in F$. Konfiguracija je *začetna* pri podatku $x = X_1, X_2, ..., X_n$, če je $q = q_0$, in $i = 1$.

Izračun pri podatku $x \in \Gamma^*$ je zaporedje konfiguracij $C_0, C_1, ..., C_k$, za katere velja $C_i \vdash C_{i+1}$ za $0 \leq i < k$. Izračun *se ustavi*, če je končna konfiguracija C_k ustavitvena. Izračun je *sprejemajoč*, če je njegova končna konfiguracija C_k sprejemajoča. Brez škode za splošnost lahko privzamemo, da se vsak sprejemajoč račun ustavi.

2.DEFINICIJA: *Nedeterministični Turingov stroj* (NTS) definiramo enako kot TS, le prehodna funkcija δ ima več mogočih vrednosti. Točneje $\delta: Q \times \Gamma \rightarrow 2^{Q \times \Gamma \{R,L\}}$.

V posebnem primeru, ko število elementov v sliki δ ni nikoli večje od 1, dobimo prej definirani (deterministični) TS.

3.DEFINICIJA: Bodi $M = (Z, \Gamma, \delta, F)$ TS:

a) $L(M) = \{x \in \Gamma^*;$ obstaja sprejemajoč izračun *TS M z vhodom x*$\}$ je jezik, ki ga razpoznava M

b) Bodi M deterministično in *totalen*, torej naj se M ustavi pri poljubnem vhodnem nizu $x \in \Gamma^*$. Potem M računa funkcijo $f_M : \Gamma^* \rightarrow \Gamma^*$, če je za vsak $x \in \Gamma^*$ vsebina traku ob ustavitvi enaka $f_M(x)$.

c)(Časovna zahtevnost T_M TS M)

$T_M(n) = \max_{x \in \Gamma^*, |x|=n} \{k;$ *k je dolžina izračuna, ki se ustavi pri vhodu* $x\}$

4.DEFINICIJA: (Razreda P in NP)

P = $\{L; L \subseteq \Gamma^*$ *za neko končno abecedo* Γ *in obstaja deterministični TS M in polinom p, da je* $L = L(M)$ *in* $T_M(n) \leq p(n)$ *za vsak* $n\}$

NP = $\{L; L \subseteq \Gamma^*$ *za neko končno abecedo* Γ *in obstaja nedeterministični TS M in polinom p, da je* $L = L(M)$ *in* $T_M(n) \leq p(n)$ *za vse* $n\}$

V nekaterih virih [GaJo79,AhHU76,Melh84] je časovna zahtevnost NTS definirana nekoliko drugače:

5.DEFINICIJA:

$T_M(n) = \max_{x \in L(M), |x|=n} \min\{k;$ *k je dolžina sprejemajočega izračuna pri vhodu* $x\}$

če je M nedeterministično, in

$T_M(n) = \max_{x \in \Gamma^*, |x|=n} \{k;$ *k je dolžina izračuna, ki se ustavi pri vhodu* $x\}$

če je M deterministično.

$T_M(n) = \infty$, če obstaja $x \in \Gamma^*$, $|x| = n$ tak, da se M ne ustavi pri vhodu x.

Ker je stara definicija $T_M(N)$ v primeru NTS strožja od nove, bi se lahko zgodilo, da bi se razred NP zdaj zajemal manj jezikov. Z NP_1 za hip označimo razred jezikov, ki je definiran tako, da v definiciji 4. uporabimo novo definicijo $T_M(N)$. Očitno je

$$NP_1 \supseteq NP$$

Na srečo pa velja tudi obratno, torej:

6.IZREK: $NP_1 = NP$

DOKAZ: Dokazati moramo $NP_1 \subseteq NP$.

Naj bo $L \in NP_1$. Torej obstaja NTS M s polinomsko časovno zahtevnostjo v smislu definicije 5, ki razpoznava L. Konstruirali bomo NTS N, ki bo razpoznaval isti jezik in bo imel polinomsko časovno zahtevnost v smislu definicije 3. Bodi $p(n) = T_M(n)$ časovna zahtevnost NTS M, kjer je p neki polinom.

TS N naj deluje takole:

- simulira delovanje NTS M in šteje korake
- ko doseže število korakov $p(n)$, se N ustavi
- če je v tem času M razpoznal vhod, ga razpozna tudi N, v vseh drugih primerih pa N odgovori NE.

N očitno razpoznava natanko L, saj po predpostavki za vsak element iz L obstaja sprejemajoč izračun z ne več kot $p(n)$ koraki.

Ker simulacija gre v polinomskem času, je časovna zahtevnost NTS N

očitno polinomska v smislu definicije 5.

Q.E.D.

Ker je vsak deterministični TS hkrati nedeterministični TS je očitno

$$P \subseteq NP$$

Prišli smo do znanega odprtega problema: ali morda velja P=NP? Prevladuje splošno prepričanje (ki seveda ne dokazuje ničesar), da P=NP ne velja. Tukaj navajamo dva 'zdravorazumska' argumenta:

a) Obstajajo problemi (takoimenovani NP–polni, definicija pride kasneje), na primer problem izpolnjivosti (SAT), za katerega velja

$$P = NP \iff SAT \in P$$

Problemov s to lastnostjo je še veliko, prihajajo pa z različnih področij. Iz teorije grafov omenimo problem klike in problem barvanja točk grafa. Razvpiti problem trgovskega potnika je samo eden izmed težkih (beri NP–polnih) transportnih problemov v operacijskih raziskavah. Mnogo dela mnogih raziskovalcev je bilo brez uspeha, ko so iskali učinkovite (polinomske) algoritme za te probleme. Še več,

b) za mnoge algoritme, ki so jih predlagali, se je izkazalo, da imajo več kot polinomsko časovno zahtevnost.

Za konec razdelka navedimo še izrek, ki primerja deterministično in nedeterministično časovno zahtevnost. Enostavna simulacija da za eksponentni red večjo časovno zahtevnost pri simulaciji nedeterminističnega TS z determinističnim. Boljše simulacije ne poznamo. Če velja $P \neq NP$ potem je to tudi najboljše, kar lahko dosežemo.

7.DEFINICIJA: Funkcija $T : N \to N$ je *časovna funkcija*, če obstaja deterministični TS, ki se ustavi po natanko $T(n)$ korakih pri vsakem vhodu dolžine n.

Nekatere običajne funkcije so tudi časovne funkcije, na primer $T(n) = n$, $T(N) = n\lfloor \log n \rfloor$, $T(n) = n^2$, $T(n) = 2^n$.

8.IZREK: (Simulacija nedeterminističnega TS z determinističnim) Bodi $T(n)$ časovna funkcija, $L \subseteq \Gamma^*$ jezik in N nedeterministični TS, ki sprejema L v času $T_N(n) = O(T(n))$. Potem obstaja deterministični TS M z $L(M) = L$ in $T_M(n) = O(c^{T(n)})$ za neko konstanto c.

DOKAZ: Za vsak $x \in L$ obstaja izračun dolžine $\leq T_N(|x|)$, ki sprejme x. Bodi $k = \max\{card(\delta(z,a)); z \text{ stanje TS } N \text{ in } a \in \Gamma\}$. Potem ima TS N na vsakem koraku največ k različnih mogočih operacij. Torej je največ $k^{T_N(|x|)}$ različnih izračunov, ki se ustavijo, dolžine $\leq T_N(|x|)$ pri vhodu x. Spomnimo se, da je $x \in L$ natanko tedaj, ko je vsaj eden od izračunov, ki se ustavijo, sprejemajoč. Uporabimo to pri simulaciji! Izberimo tak m, da bo $T_N(n) \leq mT(n)$ za vsak n. Poglejmo cela števila od 0 do $k^{mT(n)}$ pri osnovi k. Vsaka od mogočih izvedb TS N je zakodirana kot eno od teh števil na naslednji način: i-ta cifra v k-adičnem zapisu števila določa operacijo na i-tem koraku izvedbe. (Nekatera števila ne predstavljajo možne izvedbe, vendar nas to ne moti.) Enostavno je videti, da je vsako konkretno izvedbo TS N potrebno največ $p(mT(|x|))$ korakov, za neki polinom p. (Res, dodatno moramo na vsakem koraku poiskati naslednjo cifro zakodirane izvedbe, za to se moramo največ dvakrat zapeljati čez trak.) Torej celotna simulacija zahteva čas $p(mT(|x|))k^{mT(|x|)} = O(c^{T(|x|)})$ za neko konstanto c.

Q.E.D.

4. Problemi, jeziki in optimizacijski problemi

Razreda P in NP smo definirali za jezike. Praktični problemi pa običajno niso formulirani v obliki problema razpoznavanja nekega jezika. V tem razdelku bomo pokazali zvezo med optimizacijskimi problemi in njihovimi odločitvenimi inačicami.

Za primer si poglejmo odločitveni problem barvanja točk grafa in ustrezni optimizacijski problem.

Problem: Optimizacijski problem barvanja točk grafa

Vhod: Graf G

Izhod: Dobro barvanje grafa G z $\chi(G)$ barvami

Optimizacijski problem barvanja točk grafa očitno ni jezik, pač pa zahteva izračun neke transformacije iz matrike $n \times n$ bitov v zaporedje n števil $\in \{1, 2, ..., \chi(G)\}$ (ki označujejo barve). Pripadajoči odločitveni problem je

Problem: Odločitveni problem pobarvljivosti grafa (k-COL)

Vhod: Graf G in celo število k

Vprašanje: Ali obstaja dobro barvanje grafa G z največ k barvami?

Odločitvenemu problemu je mogoče na naraven način prirediti jezik. To je množica vseh primerov problema (G, k) ali točneje zapisano wv, kjer je w koda grafa G in v koda števila k, za katere je odgovor na dano vprašanje pozitiven.

$L_{k-COL} = \{(G,k); G \text{ je } k\text{–pobarvljiv }\}$

Oziroma

$L_{k-COL} = \{\ wv;\ w$ je matrika povezav nekega grafa G in v je binarni zapis nekega celega števila k in graf G je k–pobarvljiv $\}$

Mimogrede smo privzeli, da je kodiranje *naših problemov* 'naravno'. Kot smo omenili že v uvodu poglavja, se dogovorimo, kako bomo kodirali kombinatorične objekte: naravna števila bomo zapisali binarno, splošne grafe pa z matriko sosednosti. Bistvena je zahteva, da kodirna shema ne sme umetno znižati zahtevnosti problema. Če bi na primer naravna števila zapisali unarno ($n = \underbrace{111\ldots1}_{n}$), bi namesto vhoda dolžine $O(\log n)$ imeli vhod dolžine $O(n)$. Tako bi eksponentni algoritem navidez postal polinomski! Podrobneje so zahteve 'naravnega' kodiranje opisane na primer v [PePl82].

Odločitvenemu problemu smo priredili jezik, torej se lahko vprašamo, ali je problem v P. V nadaljevanju bomo pokazali, da bi v tem primeru optimizacijski problem lahko rešili v polinomskem času. Torej lahko tudi za optimizacijski problem rečemo, da je polinomski, če je njegov pripadajoči odločitveni problem v razredu P.

Za začetek poglejmo karakteristično lastnost razreda NP.

9.IZREK:

NP $= \{L; L \subseteq \Sigma^*$ *za neko končno abecedo* Σ *in obstaja polinom* p *in polinomsko izračunljiv predikat* $Q(x,y) \subseteq \Sigma^* \times \Sigma^*$ *, da za vsak* $x \in \Sigma^*$ *velja:*

$$x \in L \iff \exists y \in \Sigma^* : |y| \leq p(|x|) \text{ in } Q(x,y)\}$$

Niz y imenujemo *certifikat*.

DOKAZ: a) Bodi p polinom in Q polinomsko izračunljiv predikat. Torej obstaja deterministični TS M, ki izračuna $Q(x,y)$ v času $q(x,y)$ za neki polinom q. Naslednji nedeterministični algoritem sprejme L.

(1) Ob vhodu x nedeterministično zgeneriraj zaporedje $y \in \Sigma^*, |y| \leq p(|x|)$

(2) Razpoznaj x, če je res $Q(x,y)$ (izračunano z M)

Nedeterministični algoritem v delu (1) je podoben algoritmu, s katerim smo v prvem razdelku reševali problem klike. Kot smo se prepričali že tam, je algoritem polinomski na nedeterminističnem TS, število korakov pa je omejeno s $p(|x|)$. Časovna zahtevnost drugem delu algoritma je omejena s polinomom q, ki je odvisen od dolžine vhoda $|x|$ in dolžine $|y|$, ki pa je spet omejena s polinomom $p(|x|)$. Torej je $L \in$ NP.

b) Bodi $L \in$ NP $L \subseteq \Sigma^*$. N naj bo nedeterministični TS, ki razpoznava L v času, omejenem s polinomom p. Kot v dokazu prevedljivosti nedeterminističnega TS na deterministični TS (Izrek 8) naj bo k največje število različnih operacij, ki jih na enem koraku lahko izbere N. V dokazu omenjenega

izreka smo ugotovili, da lahko z zaporedjem cifer v k-adičnem številskem zapisu natanko določimo (zakodiramo) vsako od možnih izvedb NTS pri vhodu x. Brez škode za splošnost vzemimo, da je $|\Sigma| \geq k$. Definirajmo

$Q(x,y)$ = 'y je koda sprejemajočega izračuna NTS N pri vhodu x'

Q je (po definiciji razreda NP) izračunljiv v polinomskem času in $L = \{x; \exists y |y| \leq p(|x|)\ in\ Q(x,y)\}$

Q.E.D.

Izrek daje možnost alternativne definicije razreda NP, ki ima vsaj eno prednost. S tako definicijo se izognemo razlaganju fenomena nedeterminizma, tu vse delo opravi en eksistenčni kvantifikator!

10.DEFINICIJA: *Minimizacijski* problem je podan s polinomsko izračunljivim predikatom $Q \subseteq \Sigma^* \times \Sigma^*$ in polinomsko izračunljivo *namensko* funkcijo $c : \Sigma^* \times \Sigma^* \longrightarrow N$. Če velja $Q(x,y)$, potem je y *dopustna* rešitev problema pri vhodu x s 'stroškom' $c(x,y)$. Če velja $Q(x,y)$ in $c(x,y) \leq c(x,y')$ za vsak y' za katerega velja $Q(x,y')$, potem je y *optimalna* rešitev za x. Minimizacijski problem je *polinomsko omejen*, če obstaja polinom p, tako da iz $Q(x,y)$ sledi $|y| \leq p(|x|)$.

V tem delu bomo obravnavali samo polinomsko omejene probleme. Podobno kot minimizacijske bi lahko definirali maksimizacijske optimizacijske probleme. V primeru optimizacijskega problema barvanja točk grafa je $Q(x,y)$ izpolnjen, če je x koda grafa, y pa koda dobrega barvanja. $c(x,y)$ je število barv, uporabljeno v barvanju, ki ga kodira y.

11.DEFINICIJA: Bodi (Q,c) polinomsko omejen minimizacijski problem. Definirajmo štiri verzije problema:

a) **Problem:** (Q,c)-odločitveni problem
Vhod: $x \in \Sigma^*$, celo število C
Vprašanje: Ali obstaja y, da velja $Q(x,y)$ in $c(x,y) \leq C$?

b) **Problem:** (Q,c)-optimizacijski problem
Vhod: $x \in \Sigma^*$
Izhod: Optimalna rešitev $optsol(x) \in \Sigma^*$ za x

c) **Problem:** (Q,c)-problem optimalne vrednosti
Vhod: $x \in \Sigma^*$
Izhod: $optval(x) = c(x, optsol(x))$

d) **Problem:** (Q,c)-problem C-rešitve
Vhod: $x \in \Sigma^*$, celo število C
Izhod: $y = witness(C)$, da velja $Q(x,y)$ in $c(x,y) \leq C$, če tak y obstaja.

Za primer barvanja točk grafa smo prvi dve verziji problema že videli. Problem iskanja optimalne vrednosti v primeru barvanja točk grafa pomeni izračun kromatičnega števila $\chi(G)$, problem C-rešitve pa formuliramo takole: pri danem C poišči dobro C-barvanje, če (seveda) obstaja.

Primerjajmo časovno zahtevnost štirih verzij problema. Očitno problem razpoznavanja ni težji od problema C-rešitve in problema optimalne vrednosti, ta dva pa nista težja od optimizacijskega problema. Ali bolj nazorno:

problem razpoznavanja $\propto$ problem C-rešitve $\propto$ optimizacijski problem
problem razpoznavanja $\propto$ problem optimalne vrednosti $\propto$ optimizacijski problem

Natančneje formuliramo pravkar povedano z naslednjimi trditvami:

12.LEMA: Bodi (Q,c) polinomsko omejen optimizacijski problem. Potem velja:

a) Če je funkcija $optsol : \Sigma^* \longrightarrow \Sigma^*$ izračunljiva v polinomskem času, potem isto velja za funkciji $optval : \Sigma^* \longrightarrow \Sigma^*$ in $witness : \Sigma^* \longrightarrow \Sigma^*$.

b) Če je vsaj ena od funkcij *witness* in *optval* izračunljiva v polinomskem času, potem je

$$L_{(Q,C)} := \{(x,C); x \in \Sigma^*, C \in N in\ optval \leq C\} \in P$$

DOKAZ: Očitno iz prej povedanega.

Manj očitno je, da velja tudi obrat zadnje leme. V splošnem znamo dokazati nekoliko šibkejšo obratno trditev. Primer, ki ga v splošnem ne znamo dokazati, pa bomo dokazali za poseben primer barvanja točk grafa.

13.LEMA: Bodi (Q,c) polinomsko omejen optimizacijski problem.

a) Če sta funkciji *witness* in *optval* izračunljivi v polinomskem času, potem isto velja za *optsol*.

b) Če je problem razpoznavanja v razredu P in za neki polinom q velja $optval(x) \leq 2^{q(|x|)}$ za vse x, potem je funkcija *optval* izračunljiva v polinomskem času.

DOKAZ:

a) Trivialno, saj za vsak x velja $optsol(x) = witness(x, optval(x))$.

b) Za izračun optimalne rešitve optval bomo uporabili binarno iskanje med vrednostmi $\{1, ..., 2^{q(|x|)}\}$. Poglejmo naslednji algoritem:

```
low := 0;
high := 1;
while x nima rešitve ≤ high do high := 2 * high;
while high − low ≥ 1
    do middle := trunc((high + low)/2)
    if x ima rešitev ≤ middle
      then high := middle
      else low := middle;
    od
optval(x) := low;
```

Ker je problem po predpostavki polinomsko omejen, vemo, da je rešitev omejena z $2^{q(|x|)}$, kjer je q neki polinom. Prva **while** zanka torej v polinomskem številu korakov določi zgornjo mejo rešitve.

Hitro vidimo, da je $low \leq optval(x) \leq high$ invarianta do zanke. Torej gornji algoritem pravilno izračuna vrednost $optval(x)$ v $log_2(2^{q(|x|)}) = q(|x|)$ ponovitvah zanke. Če v drugi **while** zanki uporabimo polinomski algoritem za razpoznavanje, ki po predpostavki obstaja, smo torej res našli polinomski algoritem za izračun optimalne vrednosti $optval(x)$.

Q.E.D.

Primerjavo med problemom C-vrednosti in problemom razpoznavanja naredimo za konkretni primer barvanja. V knjigi [Melh84] najdemo dokaza za problem trgovskega potnika in za izpolnjivost, pa tudi trditev, da podobni dokazi, v katerih uporabimo tehniko redukcije problema na manjši problem iste vrste, obstajajo tudi za mnoge druge NP-polne probleme.

14.LEMA: Če je odločitveni problem C-barvanja v P potem obstaja algoritem, ki v polinomskem času poišče C-barvanje.

DOKAZ: Idejo iz [PePi82] zapišimo v obliki algoritma:

```
if not C − COL(G) then (*barvanja ni*)
else
for c := C downto 2 do begin
    izberi poljubno točko u ∈ V(G');
    pobarvaj u z barvo c;
    for all točke v ∈ V(G'), ki niso sosede u-ja do
        if not c − COL(V(G'), E(G') ∪ (u, v)) then begin
          (*točki sta neodvisni!*)
          pobarvaj v z barvo c;
          identificiraj u in v v G';
        end;
    G' := G' \{u}
end
preostale točke v G' pobarvaj z barvo 1
```

Identifikacijo dveh točk grafa očitno lahko naredimo v linearnem času (potrebno je namesto dveh vrstic (stolpcev) enega pobrisati, v drugega pa vpisati konjunkcijo vrstic (stolpcev)), zato je gornji algoritem polinomski, seveda pri predpostavki, da je algoritem, ki odloči, ali je graf c-pobarvljiv, polinomski.

Q.E.D

5. Polinomsko prevajanje in NP–polni problemi

Prevajanje problemov je uporabna tehnika za ugotavljanje 'težavnosti' problemov. (Primer smo videli v prejšnjem razdelku.)

15.DEFINICIJA: a) Naj bosta Σ in Γ končni abecedi. Preslikava $f : \Sigma^* \to \Gamma^*$ je ***polinomska transformacija***, če lahko f izračunamo v polinomskem času na determinističnem TS.

b) Bodita $L_1 \subseteq \Sigma^*$ in $L_2 \subseteq \Gamma^*$ jezika. L_1 je ***polinomsko prevedljiv*** na L_2, oznaka $L_1 \propto L_2$, če obstaja polinomska transformacija f, tako da je

$$x \in L_1 \iff f(x) \in L_2$$

za vse $x \in \Sigma^*$.

c) Jezik L je NP–poln, če velja

1) $L \in$ NP

2) $L' \propto L_1$ za vse $L' \in$ NP

Pomembnost definicije pojasni naslednji

16.IZREK: Bodi L_0 NP–poln. Potem

a) $L_0 \in$ P $\iff$ P = NP

b) če je $L_0 \propto L_1$ in $L_1 \in$ NP, potem je L_1 NP–poln

DOKAZ: a) Če je P=NP potem je gotovo $L_0 \in$ P. Dokazati moramo še drugo smer ekvivalence. Bodi $L_0 \in$ P in $L \in$ NP poljuben. Ker je $L_0 \in$ P, obstaja deterministični TS M, ki sprejme L_0 v času omejenem z nekim polinomom p. Ker je L_0 NP–poln in $L \in$ P, velja $L \propto L_0$. Torej obstaja polinomsko izračunljiva preslikava f, tako da je $L = f^{-1}(L_0)$. Bodi N deterministični TS, ki izračunava f v času, omejenem s polinomom q. Iz TS M in N konstruirajmo nov TS A, ki bo razpoznaval L takole:

(1) Izračunaj $f(x)$ v času $q(|x|)$ s TS N.

(2) Premakni glavo ('oko') na prvi simbol $f(x)$ v času $|f(x)|$

(3) Razpoznaj $f(x)$ uporabljajoč M v času $p(|f(x)|)$. Če je $f(x) \in L_0$, potem sprejmi x, sicer ga zavrni.

Gornji algoritem očitno razpoznava L. Potreben čas je $q(|x|) + |f(x)| + p(|f(x)|) \leq r(|x|)$ za neki polinom r. Zadnja neenakost je posledica dejstva, da je $|f(x)| \leq |x| + q(|x|)$, saj lahko TS v enem koraku porabi največ en nov kvadratek traku.

b) Pokazati moramo, da je $L \propto L_1$ za vsak $L \in$ NP. Vzemimo poljuben $L \in$ NP. Ker je L_0 NP–poln, obstaja polinomska transformacija f, da je $L = f^{-1}(L_0)$. Ker je $L_0 \propto L_1$ obstaja tudi polinomska transformacija g, tako da je $L_0 = g^{-1}(L_1)$. Označimo $h = g \circ f$. Potem je $L = f^{-1}(L_0) = f^{-1}(g^{-1}(L_1)) = (g \circ f)^{-1}(L_1) = h^{-1}(L_1)$. S podobnimi argumenti kot v točki a) pokažemo, da je h polinomska transformacija.

Q.E.D.

NP–polni problemi so torej najtežji med problemi razreda NP. Če bi bil eden izmed njih rešljiv v polinomskem času, potem bi bili vsi rešljivi v polinomskem času in veljalo bi P=NP. Po drugi strani pa velja: iz N≠NP sledi, da noben NP–poln problem ni rešljiv v polinomskem času.

Točka b) nam da enostavno tehniko za dokaz NP–polnosti danega problema. Če želimo pokazati, da je L NP–poln, moramo dokazati $L \in$ NP in $L_0 \propto L$ za nek problem L_0, za katerega že vemo, da je NP–poln.

Prvi dokaz NP–polnosti nekega problema je naredil Cook v svojem klasičnem članku [Cook71]. Dokazal je, da je problem izpolnjivosti NP–poln. Dokaz tu opuščamo, bralec ga lahko najde v knjigah, na primer v [GaJo79] ali [Mejh84], v slovenščini pa v [PePi82]. Ko tak dokaz imamo, lahko relativno enostavno pokažemo NP–polnost drugih problemov, kot je prvi storil Karp [Karp72]. Seznam NP–polnih problemov se od tedaj širi, zajetne sezname lahko bralec najde v omenjenih knjigah.

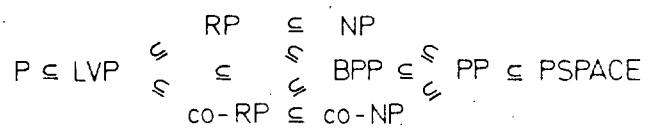
Za problem *k-COL* (odločitveni problem barvanja točk grafa) je dokazal NP–polnost že Karp [Karp72]. Dokaz je posreden, vmes pokaže NP–polnost problema 3-izpolnjivosti (*3-SAT*). Dokaz tu opuščamo, bralec ga lahko v slovenščini najde v [Gode83]. Pravkar povedano zapišimo v obliki izrekov:

17.IZREK: Problem izpolnjivosti je NP–poln.

18.IZREK: *SAT* $\propto$ *3-SAT* $\propto$ *k-COL*

6. Verjetnostni model računanja

V tem razdelku bomo definirali še nekaj novih razredov časovne zahtevnosti algoritmov. Namesto načrta si poglejmo naslednji diagram:



Slika 1. Nekateri razredi jezikov

kjer je co-C:={$\Gamma^* - L; L \in$ C *in* Γ *končna abeceda*} za vsak razred jezikov C.

Nove razrede problemov bomo definirali s pomočjo verjetnostnega Turingovega stroja. Verjetnostni Turingov stroj je TS z možnostjo slučajnih odločitev. Tukaj samo omenimo, da s tem računska moč stroja ni nič večja. Mogoče pa je, da se lahko kak problem reši hitreje ali na manjšem prostoru, kot z determinističnim Turingovim strojem.

19.DEFINICIJA: *Verjetnostni Turingov stroj* (VTS) je DTS s posebnimi stanji, v katerih sta mogoči dve nadaljevanji, ki sta enako verjetni.

Taka definicija VTS je ugodna zaradi enostavnosti. Ker so vse veje enako verjetne, dobimo enostavno porazdelitev možnih rezultatov. Model, v katerem bi dovolili neuravnotežene verjetnosti nadaljevanj, ni nič močnejši od tukaj definiranega [Schø86].

V splošnem VTS računa slučajno funkcijo: za vsak vhod x izračuna VTS rezultat y z verjetnostjo $Pr\{M(x) = y\}$.

20.DEFINICIJA: Delna funkcija, ki jo izračunava VTS M je definirana z

$$\phi_M(x) = \begin{cases} y & ,\text{če } Pr\{M(x) = y\} > \frac{1}{2} \\ nedefinirano & ,\text{če takega } y \text{ ni} \end{cases}$$

21.DEFINICIJA: *Verjetnost napake* VTS M je

$$e_M(x) = \begin{cases} Pr\{M(x) \neq \phi_M(x)\} & ,\text{če je } \phi_M(x) \text{ definirano} \\ nedefinirano & ,\text{sicer} \end{cases}$$

22.DEFINICIJA: VTS M izračunava ϕ_M z omejeno verjetnostjo napake, če obstaja konstanta $\varepsilon < \frac{1}{2}$, tako da je $e_M(x) \leq \varepsilon$ za vsak x iz domene ϕ_M.

23.DEFINICIJA: *(Blumova) časovna zahtevnost* VTS M je

$$T_M(x) = \begin{cases} najmanjši\ n,\ da\ je \\ Pr\{M(x) = \phi_M(x)\} > \frac{1}{2} \\ v\ manj\ kot\ n\ korakih & ,\text{če je } \phi_M(x) \text{ definirano} \\ \infty & ,\text{sicer} \end{cases}$$

in

$$T_M(n) = \max_{|x| \leq n} T_M(n)$$

Zadnja definicija je analogna definiciji časovne zahtevnosti NTS kot dolžine najkrajšega sprejemajočega izračuna (definicija 5.).

24.DEFINICIJA: VTS je *polinomsko omejen*, če obstaja polinom p, tako da se vsak možen izračun vhodov dolžine n ustavi po največ $p(n)$ korakih.

25.DEFINICIJA: VTS razpoznava jezik, če izračuna njegovo karakteristično funkcijo.

Ali, drugače zapisano: $x \in L \Leftrightarrow Pr\{M(x) = 1\} > \frac{1}{2}$ in $x \notin L \Leftrightarrow Pr\{M(x) = 0\} > \frac{1}{2}$

26.DEFINICIJA:

PP = 'razred jezikov, ki jih razpoznavajo polinomsko omejeni VTS'

BPP = 'razred jezikov, ki jih razpoznavajo polinomsko omejeni VTS z omejeno verjetnostjo napake'

ZPP (ali LVP) = 'razred jezikov, ki jih razpoznavajo VTS s polinomsko povprečno časovno zahtevnostjo in z verjetnostjo napake 0'

27.IZREK: LVP $\subseteq$ BPP $\subseteq$ PP $\subseteq$ PSPACE

S PSPACE smo označili razred jezikov, ki jih razpoznavajo TS s polinomsko omejeno dolžino uporabljenega traku. Izkaže se, da je vseeno, če vzamemo v definiciji deterministični ali nedeterministični TS [Savi74].

DOKAZ: Relacija BPP $\subseteq$ PP je očitna iz definicije. PP $\subseteq$ PSPACE je posledica izreka o deterministični simulaciji verjetnostnega TS, ki ga tu nismo navedli. Dokaz bi šel podobno kot dokaz izreka 8.

Pokažimo še LVP $\subseteq$BPP. Naj bo L jezik, ki ga razpoznava VTS M z verjetnostjo napake 0 in povprečnim časom omejenim s polinomom $p(n)$. Bodi $c > 2$ poljubna konstanta. Z M' označimo VTS, ki simulira delovanje VTS M do največ $cp(n)$ korakov. Če se M do tedaj ne bi ustavil, M' odgovori karkoli. Ker M naredi več kot $cp(n)$ korakov z verjetnostjo manj kot $1/c$, je verjetnost napake polinomsko omejenega stroja M' največ $1/c < 1/2$.

Q.E.D.

28.IZREK: P $\subseteq$ LVP $\subseteq$ NP $\subseteq$ PP

DOKAZ: Ker vsak polinomsko omejeni TS računa z verjetnostjo napake 0, je očitno P$\subseteq$ LVP.

Bodi L $\in$LVP in M VTS, ki razpoznava L z verjetnostjo napake 0 in polinomsko omejenim povprečnim časom. Če na M gledamo kot na nedeterministični TS vidimo, da M razpozna L v polinomskem času, saj mora biti za vsak vhod vsaj en izračun s polinomsko omejenim številom korakov. Torej $L \in$ NP in LVP $\subseteq$ NP.

Pokažimo še NP $\subseteq$ PP. Brez škode za splošnost lahko privzamemo, da L razpoznava polinomsko omejen NTS, ki ima v vsakem stanju največ dva mogoča koraka. Če na M gledamo kot na verjetnostni stroj, potem je L množica nizov, za katere obstaja sprejemajoč izračun. Torej $x \in L \Leftrightarrow \Pr\{M(x)\ sprejme\} > 0$. Za dokaz, da je L v razredu PP konstruiramo stroj M' takole: M' najprej vrže kovanec; če je rezultat grb, potem sprejme vhod, sicer pa simulira delovanje stroja M, tako da vsakič, ko ima na voljo dve nadaljevanji, izbere eno ali drugo z enako verjetnostjo ($\frac{1}{2}$).

Vendar verjetnostni stroj M' še ni povsem dober. Če $x \notin L$ je mogoče, da velja $Pr\{M'(x)\ sprejme\} = 1/2$. Torej je mogoče, da M' ne izračuna karakteristične funkcije L. Definirajmo VTS M'', za katerega bo veljalo $Pr\{M''(x)\ sprejme\} < 1/2$ za vse x-e, ki niso v L.

Bodi $p(n)$ polinom, ki omejuje število korakov izvajanja stroja M. Vsak $x \in L$ dolžine n je sprejet z verjetnostjo vsaj $2^{-p(n)}$, saj gotovo obstaja vsaj en sprejemajoč izračun in je verjetnost vsakega od izračunov dolžine n vsaj $2^{-p(n)}$. Verjetnostni TS M'' deluje takole: Najprej M'' vrže $p(n)+1$ kovanec in sprejme vhod brez nadaljevanja z verjetnostjo $\frac{1}{2} - 2^{-p(n)}$. Potem M'' simulira delovanje M in sprejme vhod, če ga sprejme M. M'' torej zavrne vsak $x \notin L$ z verjetnostjo vsaj $\frac{1}{2} + 2^{-p(n)-1}$ in sprejme vsak $x \in L$ z verjetnostjo vsaj $\frac{1}{2} + 2^{-p(n)-1}$. Torej VTS M'' razpoznava L v polinomskem času, zato NP$\subseteq$PP.

Q.E.D.

29.DEFINICIJA: VPP (ali RP) je razred jezikov, ki jih razpoznajo polinomsko omejeni VTS, ki imajo za vhode, ki niso v jeziku, verjetnost napake 0.

Ali, drugače zapisano:

$$L \in RP \Leftrightarrow L\ razpoznava\ polinomsko\ omejeni\ VTS\ M\ in$$
$$Pr\{M(x)\ sprejme\} = 0\ za\ \forall x \notin L.$$

Opomba: Tu bi dobili isti razred, če bi za $x \in L$ zahtevali $Pr\{M(x) sprejme\} \geq q$ za katerokoli konstanto $0 < q < 1$. Po n ponovitvah algoritma je namreč verjetnost napake enaka $1 - \binom{n}{0}(1-q)^n$.

30.IZREK:

1.) RP $\subseteq$ NP $\cap$ BPP

2.) LVP = RP $\cap$ co-RP

DOKAZ: 1.) Inkluziji RP $\subseteq$ BPP in RP $\subseteq$ NP sta očitni.

2.) Pokazali bomo LVP $\subseteq$ RP, LVP $\subseteq$ co $-$ RP in RP $\cap$ co $-$ RP $\subseteq$ LVP.

Najprej pokažimo LVP $\subseteq$ RP. Če je $L \in$ LVP, lahko L razpoznamo TS M, katerega pričakovani čas izvajanja je omejen z nekim polinomom, imenujmo ga q, rezultati pa so popolnoma zanesljivi. Konstruirajmo TS M' takole: - pri vhodu x simulira delovanje TS M največ $q(|x|)$ korakov. Če bi se M ustavil, preden se ustavi M' (to se zgodi z verjetnostjo vsaj 1/2), potem M' odgovori isto kot M. Sicer M' odgovori NE. Vidimo, da so pritrdilni odgovori M' povsem zanesljivi, negativni odgovori pa so pravilni z verjetnostjo 1/2. Torej je $L \in$ RP in LVP $\subseteq$ RP.

Podoben argument pokaže LVP $\subseteq$ co − RP.

Ostala nam je še inkluzija RP $\cap$ co − RP $\subseteq$ LVP. Bodi $L \in$ RP $\cap$ co − RP. Potem obstajata VTS M_1 in M_2, za katera velja:

- čas izvajanja obeh VTS je omejen s polinomom
- pozitivni odgovori M_1 in negativni odgovori M_2 so povsem zanesljivi
- negativni odgovori M_1 in pozitivni odgovori M_2 so pravilni z verjetnostjo, ki je večja od neke konstante, na primer 1/2.

Pokažimo, kako lahko konstruiramo TS z verjetnostjo napake 0 in s polinomskim povprečnim časom izvajanja. Bodi x poljuben. Simulirajmo delovanje M_1 in delovanje M_2, kar lahko naredimo v polinomskem času. Če je vsaj en odgovor zanesljiv, končamo. Zanimiv je primer, ko oba algoritma dasta odgovor, ki ni povsem zanesljiv. (verjetnost tega dogodka je manjša od 1/4). V tem primeru poskus ponovimo, dokler se ne zgodi prvi primer. Pričakovano število ponovitev poskusa je omejeno $(\sum_{i=1}^{\infty}(1/4^i))$, zato je povprečni potrebni čas omejen s polinomom.

Q.E.D.

Za konec navedimo še nekaj primerov. Začnimo s problemom določitve, ali je neko število praštevilo.

Problem: Praštevila ($PRIMES$)

Vhod: Naravno število n v binarnem zapisu.

Odgovor: Da, če je n praštevilo in ne, če ni.

31.IZREK:

a) $PRIMES \in$ co − NP

b) $PRIMES \in$ co − RP

DOKAZ: a) trivialno

b) Naslednji algoritem (Solovay-Strassen) dokazuje, da je problem $PRIMES$ v razredu co-RP.

izberi $a \in \{1, 2, \ldots, n-1\}$

če $(a, n) \neq 1$ **potem** n ni praštevilo

če $(a/n) \not\equiv a^{\frac{n-1}{2}} (mod\ n)$ **potem** n ni praštevilo

sicer n je (z verjetnostjo $> 1/2$) praštevilo

kjer je (a/n) Legendrov simbol (prim. [Gras75]).

$$(a/n) = \begin{cases} 1 & \exists x : a \equiv x^2 (mod\ n) \\ -1 & a \not\equiv x^2 (mod\ n) \forall x \end{cases}$$

Legendrov simbol znamo učinkovito (v polinomskem času) izračunati z algoritmom, ki je nekoliko podoben Evklidovemu. Temelji na recipročnem zakonu, ki pravi, da je $(q/p) = -(p/q)$, če je $p = q = 3$ in $(q/p) = (p/q)$ sicer. Poleg tega za $q > p$, torej $q = mp + r$, velja $(q/p) = (r/p)$.

Če je p praštevilo, je kongruenca

$$(a/p) \equiv a^{\frac{p-1}{2}} (mod\ p)$$

izpolnjena za vsa števila a, $1 \leq a \leq p-1$. Če pa p ni praštevilo, pa kongruenca velja za največ polovico a-jev [SoSt77].

Q.E.D.

Neodvisno je podoben dokaz našel Rabin [Rabi76].

Kot drugi primer si poglejmo naslednja problema:

Problem: MAJ

Vhod: (KNO) formula stavčnega računa $F(x_1, \ldots, x_n)$

Vprašanje: Ali F velja za večino (za več kot 2^{n-1}) naborov vrednosti za $x_1, \ldots, x_n$?

Problem: $\#SAT$

Vhod: (KNO) formula stavčnega računa $F(x_1, \ldots, x_n)$ in naravno število i

Vprašanje: Ali obstaja vsaj i različnih naborov vrednosti za spremenljivke $x_1, \ldots, x_n$, ki zadoščajo F.

KNO je kratica za konjuktivno normalno obliko.

Brez dokaza navedimo (glej npr. [Gill77])

32.IZREK: MAJ in $\#SAT$ sta PP−polna problema.

7. Reševanje NP−polnih problemov

Doslej najboljša ocena za čas, potreben za reševanje NP- polnega problema je eksponentna funkcija dolžine vhoda. Pri velikih n je stvar videti brezupna. Po drugi strani so mnogi praktični problemi dokazano NP−polni. Kaj storiti? Poglejmo nekaj splošnih pristopov:

a) Posebni primeri: Pogosto se zgodi, da v praksi ne potrebujemo rešiti NP−polnega problema v vsej splošnosti. Podproblemi imajo zaradi dodatnih omejitev včasih polinomske rešitve.

b) Dinamično programiranje in razveji−omeji sta tehniki, ki ju je mogoče uporabiti pri reševanju nekaterih NP−polnih problemov. V obeh primerih s pametno izbiro nadaljevanja precej zmanjšamo potrebno računanje na neperspektivnih rešitvah. Čas obeh metod pa je še vedno eksponenten. Več o omenjenih metodah najdemo na primer v knjigi [Koza86].

c) Z verjetnostno analizo lahko včasih pokažemo, da so resnično 'težki' primeri NP−polnega problema dokaj redki. V takem primeru lahko najdemo algoritem z dobrim pričakovanim časom računanja, čeprav zgornje meje ne moremo omejiti s polinomom. Seveda je potrebno paziti pri izbiri porazdelitve primerov NP−polnega problema. Dokaz, da je izbrana porazdelitev prava, pa utegne biti resen problem [Karp76].

Za primer omenimo metodo simpleksov za reševanje problema linearnega programiranja, ki ima eksponentno časovno zahtevnost. Metoda simpleksov se v praksi dobro obnese, verjetno zaradi izredne redkosti 'težkih' primerkov problema. Dokazano je namreč, da je metoda v povprečju polinomska [Smal83]. Kljub temu, da je problem linearnega programiranja v razredu P [Khac79], pa ne poznamo polinomskega algoritma, ki bi bil v praksi boljši od metode simpleksov.

d) Aproksimacijski algoritmi lahko včasih dajo zadovoljive rešitve v kratkem času. Seveda ni vseeno, kaj nam pomeni v konkretnem primeru zadovoljiva rešitev. V primeru barvanja točk grafa na primer velja, da bi bil tudi algoritem, ki bi poiskal skoraj optimalno rešitev NP−poln. Garey in Johnson sta namreč pokazala [GaJo76], da velja: če za kakšni konstanti $r < 2$ in d velja, da algoritem A poišče barvanje z $A(G) \leq r\chi(G) + d$ barvami v polinomskem času, potem obstaja algoritem, ki poišče $\chi(G)$-barvanje v polinomskem času.

e) Hevristični algoritmi dajejo dobre rezultate na kakšni podmnožici problemske domene. Pri deterministčnih hevrističnih algoritmih se običajno zgodi, da imajo Ahilove pete: na nekaterih primerih se obnašajo zelo slabo. Če nekatere korake naredimo slučajno, se običajno izognemo Ahilovim petam na račun nekaj večje (pričakovane) časovne zahtevnosti. Omenimo nekaj hevrističnih algoritmov za problem barvanja točk grafa:

- Corneil-Grahamov hevristični algoritem temelji na algoritmu Zykova [CoGr73,Gode83].

- Veliko hevrističnih algoritmov spada v skupino, imenovano postopki zaporednega barvanja [Bata80]. Opišemo jih z naslednjo algoritemsko shemo:

pobarvaj točke grafa s 'prazno' barvo

dokler $\exists$ *točka, pobarvana s prazno barvo* **ponavljaj**

izberi barvo za točko v

Če določimo vrstni red barvanja točk in strategijo izbire barve, dobimo algoritem. Če na primer točke izbiramo po padajočih stopnjah in izbrano

točko vsakič pobarvamo z najnižjo barvo, ki je še prosta (noben sosed še ni pobarvan s to barvo), dobimo Welsh–Powellov postopek. Za ta postopek je mogoče pokazati, da porabi največ $d+1$ barv, kjer je d največja stopnja točke v grafu. Vendar (kar pa ni presenetljivo) obstaja družina grafov, na kateri je razmerje med številom barv, ki jih porabi Welsh- Powellov postopek in med dejansko potrebnim številom barv $\chi(G)$ poljubno veliko [WePo67]. V [Žero88a] je podana konstrukcija take družine grafov.

algoritem (Welsh–Petford), ki ga obravnavamo v [Žero87,Žero88,Žero88a], temelji na protivolilnem modelu delcev z interakcijo iz statistične mehanike (percolation theory). Je primer verjetnostnega hevrističnega algoritma. Gre za iterativno (slučajno) lokalno 'popravljanje' danega barvanja, ki z verjetnostjo 1 v končno mnogo korakih doseže neko dobro barvanje, če smo na začetku izbrali vsaj $\chi(G)$ barv. Če algoritem po vnaprej določenem številu korakov nasilno prekinemo, se nam lahko zgodi, da ne dobimo dobrega barvanja, čeprav to obstaja. Vemo, da je pričakovani čas absorbcije na regularnih grafih reda $O(n^2)$. Poleg tega se algoritem na testiranih grafih obnaša zelo dobro. Zato algoritem z vgrajeno nasilno prekinitvijo (s polinomsko mejo dovoljenega števila korakov) apliciramo na 'vse' grafe in upamo, da bodo rezultati še vedno dobri.

8. Literatura

AhHU76] A.V.Aho, V.E.Hopcroft, J.D.Ullman: The Design and Analysis of Computer Algorithms, Addison-Wesley 1976

ApHa86] K.Appel, W.Haken: The Four color Proof Suffices, The Mathematical Intelligencer, Vol. 8, 1986, No. 1, p.10-20

[Bata80] V.Batagelj: Barvanja točk grafov, Seminar za računalniško in numerično matematiko 201, Ljubljana 1980

[BaPi85] D.Bajc, T.Pisanski: Najnujnejše o grafih, Presekova knjižnica 22, DMFA SRS, Ljubljana 1985

*[Cook71] S.A.Cook: The Complexity of Theorem-proving Procedures, Proc. of 3^{rd} Annual ACM Symposium on Theory of Computing, Shaker Heights, Ohio, 151-158, (1971)

CoGr73] D.G.Corneil, B.Graham: An algorithm for determining the chromatic number of a graph, SIAM J. Comp. 2 (1973) 4, 311-318

CvMi77] R.Cvetković , M.Milić : Teorija grafova i njene primjene, Naučna knjiga Beograd, 1977

DoWe83] P.Donnelly, D.Welsh: Finite particle systems and infection models, Math. Proc. Camb. Phil. Soc (1983), 94, 167-182

DuBr81] R.D.Dutton, R.C.Brigham: A new graph coloring algorithm, The Computer Journal 24 (1981) 1, 85-86

GaJo76] M.R.Garey, D.S.Johnson: The Complexity of Near-Optimal Graph Coloring, JACM 23 (1976) 1, 43-49

GaJo79] M.R.Garey, D.S.Johnson: Computers and Intractability, W.H.Freeman and Co., (San Francisco) 1979

[Gill77] J.Gill: Computational complexity of probabilistic Turing machines, Siam. J. Comp. 6 (1977) 4, p.675-695

[Gras75] J.Grasselli: Osnove teorije števil, zbirka Sigma, DZS 1975

Gode83] H.Godec: Algoritmi za barvanje grafov, diplomsko delo, FNT Ljubljana 1983

[HoUl79] V.E.Hopcroft, J.D.Ullman: Introduction to automata theory, languages and computation, Addison-Wesley 1979

[HoUl86] V.E.Hopcroft, J.D.Ullman: Uvod v teorijo avtomatov,jezikov in izračunov, (prevod B.Vilfan), Fakulteta za elektrotehniko, Ljubljana 1986

[Karp72] R.M.Karp: Reducibility among combinatorial problems, v Complexity of Computer Computations (ur. Miller, Thatcher), Plenum Press, New York, 85-103, (1972)

[Karp76] R.M.Karp: The Probabilistic Analysis of some Combinatorial Search Algorithms, v Algorithms and complexity (ur. Traub), 1-20, Academic Press 1976

[Khac79] L.G.Khatchjan: LP in polynomial time, Doklady Akad. Nauk SSSR 244 (1979) p.1093-1096

[Knut81] D.E.Knuth: Algorithms in modern mathematics and computer science, Lecture Notes in Comp. Sci. 122, Springer-Verlag, Berlin 1981

[Koza86] J.Kozak: Podatkovne strukture in algoritmi, DMFA SRS, Ljubljana 1986

LLRS85] E.L.Lawler, J.K.Lenstra, A.H.G.Rinnooy Kan, D.B.Shmoys: The Traveling Salesman Problem, Wiley 1985

[Melh84] K.Melhorn: Graph Algorithms and NP-Completeness, Springer-Verlag, 1984

[PePi82] M.Petkovšek, T.Pisanski: Izbrana poglavja iz računalništva, I. del, DMFA SRS, Ljubljana 1982

[Pisa83] T.Pisanski: Izračunljivost in rešljivost, DMFA SRS, Ljubljana 1983

[Rabi76] M.O.Rabin: Probabilistic Algorithms, v Algorithms and Complexity (ur. Traub), 21-39, Academic Press 1976

*[Savi74] W.J.Savitch: Relationship between nondeterministic and deterministic tape complexities, Journal for Computer and System Sciences (1974), p.177-192

*[Smal83] Smale: On the average number of steps of the simplex method of linear programming, Math. Prog. 27 (1983) p.241-262

[SoSt77] R.Solovay, V.Strassen: A Fast Monte-Carlo Test for Primality, SIAM J. Comp., Vol. 6, No. 1, March 1977

[Schø86] U.Schøning: Complexity and Structure, Lecture Notes in Comp. Sci. 211, Springer-Verlag 1986

[Tros83] V.N.Trostnikov: Što su konstruktivni procesi u matematici, Moderna matematika, Školska knjiga Zagreb 1983

WePe83] D.J.A.Welsh, D.M.Petford: A Randomised Attack to an NP-Complete Problem, Univ. of Oxford (preprint), 1983

WePo67] D.J.A.Welsh, M.B.Powell: An upper bound for the chromatic number of a graph and its application to timetabling problems, The Computer Journal 10 (1967), 85-86

WoWi78] D.R.Woodall, R.J.Wilson: The Appel-Haken Proof of the Four-Colour Theorem, v Selected Topics in Graph Theory I, (ur. L.W.Beineke, R.J.Wilson), Academic Press 1978

[Žero87] J.Žerovnik: Poskusi s slučajnim hevrističnim algoritmom, Zbornik XI. simpozija iz informatike Jahorina 1987, 294-(1-10)

[Žero88] J.Žerovnik: Randomised Heuristical Algorithm for Graph Colouring, Proceedings of Eighth Yugoslav Seminar on Graph Theory, Novi Sad 1987, (sprejeto)

Žero88a] J.Žerovnik: Verjetnostni algoritem za barvanje grafa, magistersko delo, FNT Ljubljana 1988

Z * smo označili posredne reference.

MODELIRANJE PODATAKA: ER JEZIK I NORMALNE FORME

INFORMATICA 1/89

Descriptors: PROGRAMMING LANGUAGE, DATA MODELING, DATA BASES RERLATIONAL

Mario Radovan
Sveučilište Rijeka, SET Pula
Univerza v Ljubljani, IJS Ljubljana

Modeliranje podataka temelji na dva osnovna sredstva: ER jeziku i normalizaciji. Dok je prvo relativno jednostavno i široko poznato, prema drugom vlada izvjestan "animozitet", i to zbog njegove "sofisticirane" i strogo formalne prirode. U ovom članku analiziramo neke tipične primjere "nenormalnosti" formi relacija, te pokazujemo da te "nenormalnosti" slijede iz grešaka u ER modelu podataka. Drugim riječima, ovdje pokazujemo da ako je ER model podataka korektno izrađen, sheme relacija koje iz njega slijede redovito već jesu u "optimalnoj (normalnoj) formi".

DATA MODELING: ER LANGUAGE AND NORMAL FORMS Data modeling is based on two principal means: *ER language* and *normalization*. While the first one is relatively simple and wide known, the second one undergoes certain "animosity" due to its "sophisticated" and strictly formal nature. In this paper we analyse typical examples of "non-normal" forms of relations, and show that such "non-normalities" derive from errors in the ER data model. In other words, it is shown that if the ER data model is correctly drawn out, the relational schemes deriving from it *are already* in "an optimal (normal) form".

1. Uvod

ER jezik

Model objekti-veze-svojstva, kao *grafički jezik* za predstavljanje "strukture svijeta" (ili: *forme znanja* o njemu!) predložen je u <CHE 76>. Taj jezik obično se naziva *ER modelom/jezikom* (*E*ntity-*R*elationship), pa ćemo ga ovdje tako i nazivati. Polaznu osnovu ER modela/jezika možemo izreći slijedećim riječima:

> Podaci su znanja o *objektima*, *vezama* (među tim objektima) i *svojstvima* (objekata odnosno veza). Stoga, je cilj jezika za modeliranje podataka da omogući precizno i jednostavno predstavljanje *forme* tih triju temeljnih kategorija znanja.

Na Chenov prijedlog ER modela podataka uslijedilo je više terminoloških i notacijskih nadopuna, kao i proširenja samoga modela. Zbog ograničenosti prostora, ovdje ne iznosimo eksplicitno sam ER model podataka. Dobar prikaz toga modela/jezika te načina njegova prevođenja na relacijski jezik dat je npr. u <TOR 86>.

Relacijski jezik

Relacija, kao temeljni element relacijskog jezika/modela, ima dva aspekta: *značenje* i *sadržaj*. Značenje relacije naziva se *intenzijom*, a formalno se iskazuje *shemom relacije*. Sadržaj relacije naziva se *ekstenzijom*, a iskazuje se *naslovljenom tabelom podataka*. Tabelu tvore *n-torke atomarnih vrijednosti*.

Pored "tabelarnog izgleda", relacijski model karakterizira i skup *operatora* definiranih na skupu tabela-relacija. Ti operatori omogućavaju da se, pored znanja (podataka) *eksplicitno datih* pojedinim relacijama, *deduciraju* (izračunaju) i ona znanja koja iz toga *skupa* relacija

logički slijede. U kontekstu modeliranja podataka bitni su operatori *projekcije* i *spajanja*. Prikaz relacijskog modela dat je npr. u <MAI 83>, <DAT 86>, <KOR 86>, <TKA 88>, ...

Normalizacija

Neformalno rečeno, (standardnim) procesom normalizacije nastoji se razviti dobar model podataka na taj način da se iz *datog* modela podataka otklanjaju slabosti. Stoga, prikaz problematike normalizacije otpočnimo analizom slabosti koje mogu karakterizirati neki dati model podataka.

Na slici 1 dat je ilustrativni primjer skupa relacija KUPAC, ARTIKAL i NARUDŽBA.

KUPAC

Š-KUP	IME	SJEDIŠTE
K1	Galeb	Zagreb
K2	Arena	Pula
K3	Mirna	Rovinj
K4	Badel	Zagreb

ARTIKAL

Š-ART	NAZIV	BOJA	CIJENA
A1	olovka	crvena	3
A2	gumica	bijela	7
A3	penkala	plava	8
A4	penkala	crvena	9

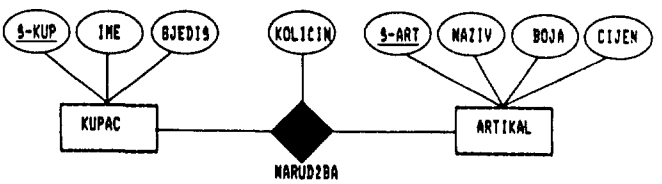
NARUDŽBA

Š-KUP	Š-ART	KOLIČINA
K1	A1	100
K1	A2	200
K2	A1	200
K2	A2	200
K3	A3	100

Slika 1.

Postavlja se pitanje zašto je dati *skup podataka* (o fragmentu realnog svijeta) *razdijeljen* upravo u tri zasebne relacije. Eventualni odgovor da to "očito treba biti tako" nije neumjesan. Međutim, kod opsežnijih (kompleksnijih) modela stvari obično nisu toliko očite.

S druge strane, mogli bismo isto tako reći da data podjela podataka slijedi iz ER modela sa slike 2.



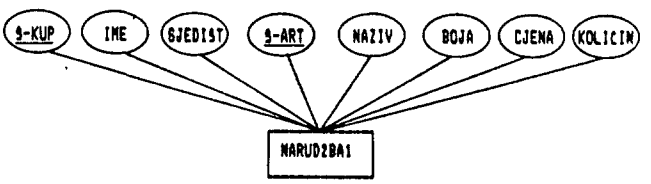
Slika 2.

Naime, prijevod tog ER modela podataka na relacijski jezik daje sheme relacija:

KUPAC(Š-KUP, IME, SJEDIŠTE),
ARTIKAL(Š-ART, NAZIV, BOJA, CIJENA),
NARUDŽBA(Š-KUP, Š-ART, KOLIČINA),

tj. točno sheme relacija sa slike 1.

Međutim, to je prije odgovor na pitanje *odakle* nego na pitanje *zašto* tri relacije. Da bi odgovorili na pitanje *zašto*, pokušajmo isti fragment realnog svijeta opisati *nekorektnim* ER modelom podataka datim na slici 3.



Slika 3.

Relacijski zapis ER modela podataka sa slike 3 glasi:

NARUDŽBA1(Š-KUP, IME, SJEDIŠTE, Š-ART,
NAZIV, CIJENA, KOLIČINA)

Primjer ekstenzija relacije NARUDŽBA1 dat je na slici 4.

Prva slabost relacije NARUDŽBA1, jeste *prisustvo redundance*. Na primjer, ime i sjedište kupca javljaju se toliko puta koliko artikala je naručio pojedini kupac. Zbog prisustva redundance javljaju se problemi i kod mijenjanja ekstenzije (tj. sadržaja) relacije. Te probleme zajedničkim imenom nazivamo *anomalijama održavanja*.

NARUDŽBA1

Š-KUP	IME	SJEDIŠTE	Š-ART	NAZIV	BOJA	CIJENA	KOLIČINA
K1	Saleb	Zagreb	A1	olovka	crvena	3	100
K1	Saleb	Zagreb	A2	gumica	bijela	7	200
K2	Arena	Pula	A1	olovka	crvena	3	200
K2	Arena	Pula	A2	gumica	bijela	7	200
K3	Mirna	Rovinj	A3	penkala	plava	5	100

Slika 4.

Upis

Podatke o pojedinom kupcu *nije moguće upisati* u relaciju NARUDŽBA1 sve dok taj kupac nešto ne naruči. Analogno vrijedi i za artikle.

Brisanje

Brisanjem pojedine narudžbe mogu biti izgubljeni i svi podaci o kupcu odnosno artiklu.

Mijenjanje

Ako kupac promjeni ime i/ili sjedište, onda se ta promjena mora provesti na toliko mjesta koliko narudžbi ima za toga kupca. Analogno vrijedi i za artikle.

Primjetimo da se nijedna od iznad navedenih slabosti relacije NARUDŽBA1 ne javlja u skupu relacija sa slike 1, gdje su podaci razdjeljeni u tri zasebne relacije. Stoga, iznijeti primjer navodi na zaključak da je relaciju sa "mnogo atributa" poželjno dekomponirati (razdijeliti) na "više relacija sa manje atributa". Cilj teorije normalizacije jeste da definira kriterije *kada* i proces *kako* se dekompozicija date sheme relacije treba izvesti.

Kažemo da se normalizacija date sheme relacije izvodi na temelju *dodatnih znanja* o odnosima među entitetima realnog svijeta, čiji model podataka oblikujemo. Ta znanja nazivamo *zavisnostima*. Govorimo o tri vrste zavisnosti, i to: *funkcijskoj zavisnost* (FZ), *višeznačnoj zavisnost* (VZ) i *zavisnosti spajanja* (ZS).

2. Funkcijska zavisnost

> Neka bude data shema relacije R sa skupom atributa A(R). Nadalje, neka V i W budu podskupovi od A(R). Na shemi relacije R vrijedi funkcijska zavisnost
>
> V --> W
>
> akô i samo ako svakoj instanci od V može biti pridružena točno jedna instanca od W. Tada kažemo da V *funkcijski determinira* W, odnosno da W *funkcijski zavisi* od V.
>
> Za FZ oblika X --> Y kažemo da je *trivijalna* ako je Y podskup od X.

Zavisnosti ne iskazuju odnose unutar neke date *ekstenzije* relacije (tj. nekog *trenutnog* sadržaja relacije), već odnose koji - u datom fragmentu realnog svijeta - *uvijek* vrijede među promatranim entitetima.

Na temelju zavisnosti, definiran je i pojam *legalne ekstenzije* relacije.

> Neka bude data shema relacije R, sa pripadnim skupom funkcijskih zavisnosti F. Ekstenzija relacije R legalna je ako i samo ako su na njoj zadovoljeni svi uvjeti iskazani sa FZ iz skupa F.

Legalnost ekstenzije relacije *ne* garantira točnost podataka. Međutim *ne*legalnost garantira da su netočne barem neke od tvrdnji iz date ekstenzije relacije.

Iz datog skupa funkcijskih zavisnosti F mogu *logički slijediti* i FZ koje u F nisu eksplicitno sadržane. Na primjer, iz skupa FZ koji sadrži zavisnosti A --> B i B --> C logički slijedi zavisnost A --> C. Skup svih FZ koje logički slijede iz F naziva se *zatvorenjem od F*, a označava se sa F+. Niz *pravila* koja omogućavaju da se iz datog skupa F izvedu *sve i samo* FZ koje spadaju u F+ naziva se *Armstrongovim aksiomima*.

Formalna definicija postupka normalizacije temelji na skupu zavisnosti F+. No, radi pojednostavljenja prikaza, ovdje proces normalizacije temeljimo na skupu *eksplicitno* datih zavisnosti, tj. na skupu F. U praktičkim terminima, to ne umanjuje valjanost iznijetih postupaka. Naime, projektant je taj koji - promatranjem odnosa u realnom svijetu - utvrđuje skup eksplicitnih zavisnosti F. Stoga, nema razloga da u taj skup ne uključi *sve relevantne* FZ, čime skup F+ postaje praktički beznačajan.

3. Dekompozicija bez gubitka informacija

> Neka bude data shema relacije R. Relacijske sheme R1 i R2 su *dekompozicija od R* ako i samo ako vrijedi:
> A(R1) unija A(R2) = A(R).

Dakle, sheme relacija R1 i R2 tvore dekompoziciju sheme relacije R ako i samo ako se svaki atribut iz A(R) javlja u barem jednoj od shema relacija R1 odnosno R2.

> Dekompozicija (R1, R2) sheme relacije R je *bez gubitaka informacija* ako i samo ako se svaku legalnu ekstenziju relacije R dade rekonstruirati spajanjem njenih projekcija na skupove atributa A(R1) i A(R2).

Ta definicija ne daje prikladan kriterij za utvrđivanje da li dekompozicija jeste ili nije bez gubitaka informacija. Pogledajmo, stoga, slijedeću definiciju.

> Neka bude data shema relacije R i pripadni skup funkcijskih zavisnosti F. Dekompozicija (R1, R2) sheme relacije R je *bez gubitaka informacija* ako i samo ako na R vrijedi barem jedna od slijedećih FZ:
> A(R1) presjek A(R2) --> A(R1)
> A(R1) presjek A(R2) --> A(R2).

Drugim riječima, presjek skupova atributa shema relacija R1 i R2 mora bit *kandidat ključa* u barem jednoj od tih relacija.

> Shema relacije R na kojoj vrijedi FZ oblika X --> Y dekomponira se na sheme relacija R1 i R2 tako da vrijedi:
> A(R1) = X unija Y
> A(R2) = A(R) minus Y.

Takva dekompozicija očito jeste bez gubitaka informacija jer je
A(R1) presjek A(R2) = X,
a zbog X --> Y, skup atributa X je kandidat ključa u R1, tako da vrijedi X --> A(R1).

Datu shemu relacije R općenito se dekomponira na proizvoljan broj shema relacija R1, ..., Rn. No, radi pojednostavljenja, ovdje se ograničavamo na dekompoziciju na dvije sheme relacija. Proces dekomponiranja može se dalje sukcesivno izvoditi na shemama relacija generiranim u prvom koraku dekompozicije.

4. Prva, druga i treća normalna forma

Prva normalna forma

> Shema relacije je u *prvoj normalnoj formi* (1NF) ako i samo ako je domena svakog od njenih atributa skup *atomarnih* vrijednosti.

Shema relacije NARUDŽBA1 jeste u 1NF. Međutim, javljanje redundance i anomalija održavanja u relaciji NARUDŽBA1 pokazuje da 1NF sheme relacije nije dovoljan uvjet za dobar model podataka.

Druga normalna forma

Definirajmo najprije neke osnovne pojmove.

> *Nekljućnim atributom* nazivamo atribut koji nije sadržan u kandidatu ključa.

> FZ oblika X --> Y nazivamo *potpunom FZ* ako ne postoji skup V, V pravi podskup od X, za koji vrijedi V --> Y. Tada kažemo da Y *potpuno zavisi* od X.

> Funkcijsku zavisnost (od X) koja nije potpuna nazivamo *parcijalnom FZ* (od X).

> Shema relacije R nalazi se u 2NF ako je svaki nekljućni atribut od R *potpuno zavisan* od kandidata ključa.

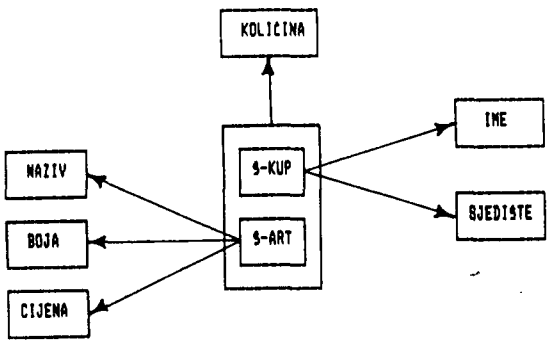
2NF nema većeg praktičkog značaja jer se prevođenjem sheme relacije *samo* u 2NF slabosti modela (tj. redundanca i anomalije) općenito ne otklanjaju. Stoga 2NF možemo smatrati samo "prirodnim prethodnikom" treće normalne forme.

Treća normalna forma

> Naka bude data relacija R, i neka X, Y i Z budu podskupovi od A(R). Funkcijska zavisnost X --> Y je tranzitivna FZ na R ako na R vrijede zavisnosti X --> Z i Z --> Y.

> Shema relacije R nalazi se u *trećoj normalnoj formi* (3NF) ako nekljućni atributi nisu tranzitivno zavisni od kandidata ključa.

Na slici 5 dat je grafički prikaz FZ koje vrijede na shemi relacije NARUDŽBA1. Ta shema relacije nije u 2NF - a time ni u 3NF - jer je parcijalna zavisnost oblik tranzitivne zavisnosti.



Slika 5.

Polazeći od FZ

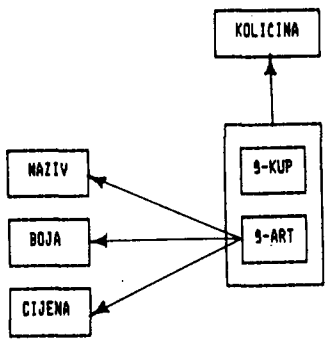
Š-KUP --> IME, SJEDIŠTE,

shemu relacije NARUDŽBA1 dekomponirajmo na sheme relacija:

R1(Š-KUP, IME, SJEDIŠTE)

R2(Š-KUP, Š-ART, NAZIV, BOJA, CIJENA, KOLIČINA).

Shema relacije R1 jeste u 3NF, jer IME ne determinira SJEDIŠTE tako da nema tranzitivnih zavisnosti. Međutim, dijagram FZ za shemu relacije R2, dat na slici 6 pokazuje da ta shema relacije nije u 2NF (a time ni u 3NF).



Slika 6.

Polazeći od FZ

Š-ART --> NAZIV, BOJA, CIJENA,

shemu relacije R2 dekomponirajmo - prema iznad datom principu - na sheme relacija R3 i R4:

R3(Š-ART, NAZIV, BOJA, CIJENA)

R4(Š-KUP, Š-ART, KOLIČINA).

Sheme relacija R3 i R4 jesu u 3NF, jer ne sadrže tranzitivnih zavisnosti.

Dekompozicijom sheme relacije NARUDŽBA1 na relacije R1, R3 i R4 generirane su shema relacija sa slike 1 (ovdje sa drukčijim imenima, naravno). To ujedno pokazuje da *proces normalizacije* sheme relacije NARUDŽBA1 (koja slijedi iz *nekorektnog* ER modela podataka sa slike 2) daje one sheme relacija koje daje i *sam korektan ER mode podataka* sa slike 1! Dakle, normalizacija se ovdje svela na otklanjanje nekorektnosti ER modela podataka. A to sugerira da korektna izrada ER modela podataka dovodi do shema relacija koje *već jesu* "u optimalnoj (normalnoj) formi", čime se ukida i sama potreba po normalizaciji.

5. Dekompozicija bez gubitka zavisnosti

Funkcijske zavisnosti iskazuju odnose koji vrijede u realnom svijetu. Stoga, ako želimo da ti odnosi vrijede i u modelu podataka (a to je globalni kriterij dobrog modeliranja!), onda prilikom dekomponiranja sheme relacije nijedna FZ ne smije biti ukinuta.

> Dekompozicija (R1, R2) sheme relacije R je *bez gubitaka funkcijskih zavisnosti* ako sve FZ definirane na R *logički slijede* iz unije skupova FZ definiranih na R1 odnosno R2.

Slijedeća definicija daje operativno pravilo za dekomponiranje bez gubitaka FZ:

> Shema relacije R dekomponira se (bez gubitka FZ) ako se dekomponira prema FZ koja *nije od kandidata ključa*.

Dosadašnji prikaz procesa modeliranja podataka zaključimo slijedećom tvrdnjom:

> Svaka shema relacije R koja nije u 3NF dade se - sukcesivnom primjenom ovdje opisanog postupka - dekomponirati na skup shema relacija R1, ..., Rn, tako da vrijedi:
>
> - Svaka Ri, 1 =< i =< n, *jeste u 3NF*;
> - Dekompozicija je *bez gubitka informacija*;
> - Dekompozicija je *bez gubitka funkcijskih zavisnosti*.

Možemo reći da je 3NF (za praksu) *najvažnija normalna forma*. Naime, shema relacije koje nije u 3NF redovito dovodi do ranije iznijetih slabosti (redundanca, anomalije održavanja). S druge strane, shemu relacije koja jaste u 3NF ponekad *nije moguće* dekomponirati bez gubitka FZ.

Iznijetim primjerom ilustrirano je da sam korektan ER model podataka daje sheme relacija koje jesu (barem) u 3NF. Utoliko se i normalne forme mogu ovdje smatrati formalnim kriterijima za *kontrolu ispravnosti ER modela podataka*.

6. Boyce/Coddova normalna forma

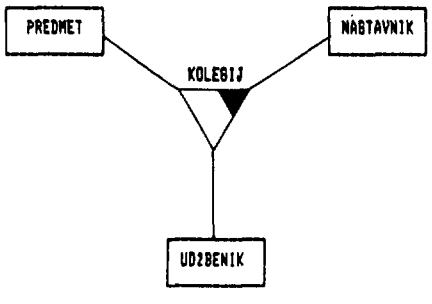
Za Boyce/Coddovu normalnu formu (BCNF) možemo reći da predstavlja stroži oblik 3NF. BCNF je relevantna za one sheme relacija koje imaju više *sastavljenih* kandidata ključa koji se međusobno djelomično *prekrivaju*.

> Shema relacije je u BCNF ako i samo ako su sve njene netrivijalne FZ ujedno FZ od kandidata ključa.

Problematiku vezanu za BCNF iznijeti ćemo na primjeru trojne veze. Najprije je data trojna veza koja - prevedena na relacijski jezik - daje shemu relacije koja je u 3NF, ali i u BCNF. Zatim je taj primjer modificiran tako da shema relacije jeste u 3NF, ali ne i u BCNF.

Na slici 7 dat je ER model podataka koji predstavlja slijedeće tvrdnje:

1. Jedan nastavnik za jedan predmet koristi jedan udžbenik.
2. Jedan predmet prema jednom udžbeniku predaje više nastavnika.
3. Jedan udžbenik jedan nastavnik koristi za jedan predmet.



Slika 7.

Vezu KOLEGIJ zapisujemo na relacijskom jeziku slijedećom shemom relacije:

KOLEGIJ(<u>Š-PRED, Š-NAST</u>, Š-UDŽ)

Da su <Š-PRED, Š-NAST> i <Š-NAST Š-UDŽ> kandida- ti ključa slijedi iz tvrdnje (1) odnosno (3). Odatle slijede i FZ:

FZ1: Š-PRED, Š-NAST --> Š-UDŽ,

FZ2: Š-NAST, Š-UDŽ --> Š-PRED.

Obzirom da shema relacije KOLEGIJ nema neključnih atributa, ne može imati ni tranzitivnih zavisnosti takvih atributa te se stoga nalazi u 3NF. Nadalje, obzirom da su obje iznad date netrivijalne FZ od kandidata ključa, shema relacije KOLEGIJ nalazi se i u BCNF.

Izmijenimo sada treću od iznad datih tvrdnji, koja neka glasi:

3'. Jedan udžbenik koristi se samo za jedan predmet.

Tvrdnja (3') *implicira* tvrdnju (3). Naime, ako se jedan udžbenik koristi samo za jedan predmet (tvrdnja 3'), onda to mora poštivati svaki nastavnik (tvrdnja 3).

Tvrdnju (3') - samu za sebe! - predstavili bi u ER modelu podataka *binarnom* vezom. Međutim, u kontekstu tvrdnji (1) i (2) predstavljamo ju u okviru trojne veze, točno kako je to učinjeno ER modelom podataka na slici 7. Odatle slijedi da je i shema relacije - nazovimo ju KOLEGIJ1 - za vezu datu tvrdnjama (1), (2) i (3') jednaka predašnjoj shemi relacije KOLEGIJ. Dakle,

KOLEGIJ1(<u>Š-PRED, Š-NAST</u>, Š-UDŽ)

Međutim, na tim shemama ne vrijede isti skupovi FZ. Obzirom da je tvrdnja (1) ostala neizmjenjena, očito ovdje vrijedi FZ1. Nadalje, obzirom da (3') implicira (3) vrijedi i FZ2. No, iz tvrdnje (3') slijedi i

FZ3: Š-UDŽ --> Š-PRED

koja *ne vrijedi* na shemi relacije KOLEGIJ.

Na slici 8 dat je primjer legalne ekstenzije relacije KOLEGIJ1.

Ekstenzija je legalna jer su njome zadovoljene sve tri iznad date FZ. Nadalje, obzirom da vrijednost atributa Š-UDŽ ne "adresira" samo jednu n-torku iz relacije KOLEGIJ1, taj atribut očito nije kandidat ključa. A to, nadalje, znači da FZ3 nije funkcijska zavisnost od kandidata ključa, tako da ni shema relacije KOLEGIJ1 nije u BCNF.

KOLEGIJ1

Š-PRED	Š-NAST	Š-UDŽ
P1	N1	U1
P1	N2	U1
P2	N1	U2
P2	N2	U2

Slika 8.

S druge strane, ta shema relacije jeste u 3NF, jer nema nekljućnih atributa. Međutim, primjetimo da se u relaciji KOLEGIJ1 - uprkos 3NF - javlja redundanca (a sa njom i anomalije održavanja). Ta redundanca je posljedica od FZ3. Naime, ako sam udžbenik determinira predmet, onda se podatak o udžbeniku ne bi trebao zapisivati toliko puta koliko nastavnika predaje taj predmet (kako je to slučaj u relaciji KOLEGIJ1).

Postavlja se, stoga, pitanje što učiniti sa shemom relacije KOLEGIJ1. U datom primjeru, preporučljiv odgovor glasi *ništa*. Dakle, zadržati ju u modelu podataka takvu kakva jeste. Naime, shemu relacije KOLEGIJ1 nije moguće dekomponirati bez gubitka FZ. Pogledajmo to.

Polazeći od "kritične" FZ3, shemu relacije KOLEGIJ1 možemo - bez gubitka informacija! - dekomponirati na:

R1(Š-UDŽ, Š-PRED)
R2(Š-NAST, Š-UDŽ).

Međutim, tom dekompozicijom izgubljene sz FZ1 i FZ2. Naime, od FZ sa sheme relacije KOLEGIJ1, ovdje vrijedi samo FZ3 (na R1), iz koje zacjelo ne slijede FZ1 i FZ2. Dakle, takvu dekompoziciju ne valja izvoditi, tako da prikaz veze KOLEGIJ1 ER modelom sa slike 7 *jeste korektan*.

Postoje sheme relacija koje jesu u 3NF a nisu u BCNF, i koje se dadu prevesti u BCNF *bez gubitaka FZ*. Pogledajmo primjer. Neka bude data shema relacije

NARUDŽBA1(Š-KUP, IME-KUP, Š-ART, KOLIČINA)

i neka skup F sadrži slijedeće FZ:

Š-KUP, Š-ART --> KOLIČINA,
IME-KUP, Š-ART --> KOLIČINA,
IME-KUP --> Š-KUP,
Š-KUP --> IME-KUP

Shema relacije NARUDŽBA1 jeste u 3NF jer (jedini) nekljućni atribut KOLIČINA ne zavisi tranzitivno. Međutim, ta shema relacije nije u BCNF jer sadrži FZ koje nisu od kandidata ključa.

BCNF modela podataka možemo doseći dekompozicijom sheme relacije NARUDŽBA1 prema

Š-KUP --> IME-KUP

koja nije od kandidata ključa) na:

R1(Š-KUP, IME-KUP)

R2(Š-KUP, Š-ART, KOLIČINA).

Ta dekompozicija je bez gubitaka informacija i bez gubitaka funkcijskih zavisnosti. Naime:

- treća i četvrta FZ iz F definirane su na R1,
- prva FZ iz F definirana je na R2,
- druga FZ iz F logički slijedi iz treće i prve FZ.

Međutim, shema relacije NARUDŽBA1 slijedi iz *grube greške u ER modelu podataka*. Naime, svojstvo IME-KUP očito pripada entitetu KUPAC a ne entitetu NARUDŽBA. Dakle, sam korektan ER model podataka doveo bi do shema relacija R1 i R2, a ne do sheme relacije NARUDŽBA1. Dakle, sheme relacija R1 i R2 - formirane iz sheme relacije NARUDŽBA1 - pokazuju da iz korektno izrađenog ER modela podataka slijede sheme relacija koje nisu samo u 3NF već i u BCNF.

Ako pak ER model podataka ne daje shemu relacije u BCNF (slučaj sheme KOLEGIJ1), onda valja provjeriti da li se takva shema *uopće dade* dekompnirati (bez gubitaka!) na sheme relacija koje jesu u BCNF. Ovdje nemamo formalnog dokaza da takva dekompozicija nije moguća. No, nije nam pošlo za rukom naći primjer u kojem bi takva dekompozicija bila moguća.

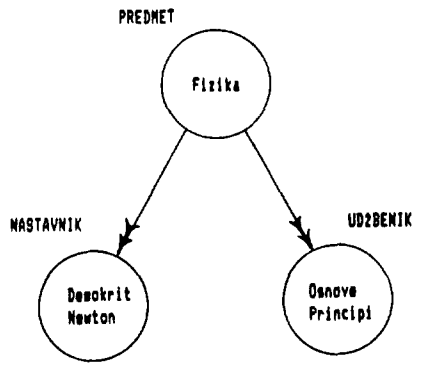
7. Višeznačna zavisnost

Neka bude data shema relacije R i neka X, Y i Z budu podskupovi od A(R). Na shemi relacije R vrijedi VZ

X -->> Y

ako i samo ako *skup* vrijednosti atributa iz Y za *dati* par vrijednosti atributa iz X i Z zavisi isključivo od vrijednosti atributa iz X, a nezavisan je od vrijednosti atributa iz Z. Kažemo da X *višeznačno determinira* Y, odnosno da Y *višeznačno zavisi* od X.

Grafički prikaz VZ dat na slici 9, čini gornju definiciju razumljivijom.



Slika 9.

Na slici 9 date su dvije VZ, i to:

PREDMET -->> NASTAVNIK
PREDMET -->> UDŽBENIK.

VZ se javljaju u "parovima". Vrijedi slijedeće pravilo:

Ako na shemi relacije R vrijedi VZ
X -->> Y
onda vrijedi i VZ
X -->> R minus (X unija Y).

Dakle, ako skup atributa višeznačno determinira skup atributa Y, onda X višeznačno determinira i skup preostalih atributa sheme relacije R. To obično zapisujemo na slijedeći način:

X -->> Y | R minus (X unija Y)

VZ koje su zadovoljene u svakoj ekstenziji relacije nazivamo trivijalnima.

Na primjer, u svakoj ekstenzij binarne relacije R(X, Y) zadovoljena su trivijalne VZ X -->> Y i Y -->> X. Naime, svakoj vrijednosti atributa X pripada skup (od jedne ili više) vrijednosti atributa Y, i obrnuto.

VZ predstavljaju generalizaciju FZ. Pod time podrazumijevamo da svaka FZ - po definiciji! - jeste ujedno i VZ. Naime, FZ oblika X --> Y je ujedno VZ kod koje *skup* vrijednosti atributa Y za *jednu* vrijednost atributa X sadrži *samo jedan element*.

8. Četvrta normalna forma

Shema relacije koja se nalazi u BCNF i/ili 3NF može svejedno dovesti do javljanja redundance (u bazi), a time i do anomalija održavanja. Pogledajmo primjer. Na slici 10 dat je primjer ekstenzije relacije KOLEGIJ2, čija shema glasi

KOLEGIJ2(Š-PRED, Š-NAST, Š-UDŽ)

KOLEGIJ2

Š-PRED	Š-NAST	Š-UDŽ
Fizika	Demokrit	Osnove
Fizika	Demokrit	Principi
Fizika	Newton	Osnove
Fizika	Newton	Principi

Slika 10.

Shema relacije KOLEGIJ2 jeste u BCNF jer na njoj nije definirana nikakva netrivijalna FZ. Međutim, relacija KOLEGIJ2 sa slike 10 očito sadrži redundancu: zapis o tome da pojedini nastavnik predaje pojedini predmet javlja se toliko puta koliko udžbenika ima za taj predmet. Analogno vrijedi i za udžbenike.

Kao i FZ, VZ daju osnovu za definiranje (nove) normalne forme, odnosno dekomponiranje shema relacija, u cilju otklanjanja redundance i anomalija održavanja.

Shema relacije je u *četvrtoj normalnoj formi* (4NF) ako i samo ako su sve njene netrivijalne VZ ujedno FZ od kandidata ključa.

Jednostavnije rečeno, shema relacije je u 4NF ako je u BCNF i ako *nema* "pravih VZ". (Dakle, ako nema VZ koje nisu FZ.) Naime, ako shema relacije nema "pravih VZ", onda se zahtjev iskazan samom *definicijom* 4NF svodi na zahtjev iskazan definicijom BCNF. (Dakle, da sve FZ budu od kandidata ključa.)

Shema relacije KOLEGIJ2 nije u 4NF. Naime, ona očito sadrži VZ

Š-PRED -->> Š-NAST | Š-UDŽ,

koje, nisu FZ (pa ni FZ od kandidata ključa).

U cilju otklanjanja redundance i anomalija održavanja, shemu relacije koja sadrži VZ valja dekomponirati.

Shema relacije na kojoj vrijedi netrivijalna VZ oblika
X -->> Y
dekomponira se bez gubitaka informacija

na sheme relacija R1 i R2, pri čemu vrijedi:

A(R1) = X unija Y

A(R2) = A(R) minus Y.

Polazeći od datih VZ, shema relacije KOLEGIJ2 dekomponira se na sheme relacija

R1(Š-PRED, Š-NAST)

R2(Š-PRED, Š-UDŽ).

Na slici 11 date su ekstenzije relacija R1 i R2, dobivene projekcijom ekstenzije relacije KOLEGIJ2 na skupove atributa A(R1) odnosno A(R2).

R1

Š-PRED	Š-NAST
Fizika	Demokrit
Fizika	Newton

R2

Š-PRED	Š-UDŽ
Fizika	Osnove
Fizika	Principi

Slika 11.

Sheme relacija R1 i R2 jesu u 4NF. Naime, na tim shemama vrijede samo trivijalne VZ.

Ranije isticana nužnost *očuvanja zavisnosti* kod dekompozicije, odnosi se i na višeznačne zavisnosti. Formalni prikaz toga problema prelazi potrebe ovoga prikaza. Stoga, navodimo samo slijedeće (praktičko) načelo:

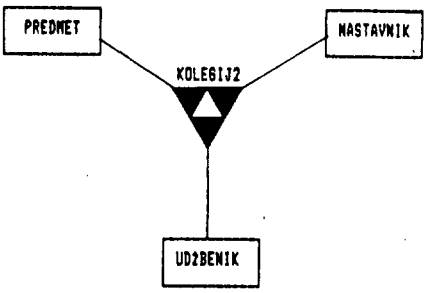
> Ako se dekompozicija izvodi polazeći od netrivijalne VZ definirane na polaznoj shemi relacije (kako je iznad učinjeno), onda dekompozicija neće dovesti do gubitka VZ.

Činjenica da neka shema relacije jeste u BCNF a nije u 4NF ukazuje na *evidentnu grešku u ER modelu podataka*, iz kojeg takva shema relacije slijedi. Na primjer, shema relacije KOLEGIJ2 očito slijedi iz ER modela podataka datog na slici 12. Napomenimo da tip povezanosti *mnogo* svih triju tipova objekata koji tvore vezu slijedi iz toga što je ključ sheme relacije KOLEGIJ2 sastavljen iz identifikatora *svih triju* tipova objekata.

"Otkriće" VZ

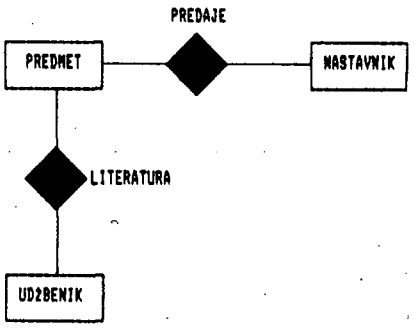
Š-PRED -->> Š-NAST | Š-UDŽ

na shemi relacije KOLEGIJ2 ukazuje na to da su tipovi objekata NASTAVNIK i UDŽBENIK međusobno *potpuno nezavisni*. A to onda znači da *trojna* veza KOLEGIJ2, kojom su ti tipovi objekata po-



Slika 12.

vezani u jednu vezu, *ne* opisuje stvarno stanje stvari u fragmentu realnog svijeta čiju strukturu modeliramo. Drugim riječima, javljanje VZ na shemi relacije je posljedica *nekorektnosti ER modela podataka sa slike 12*. Naime, ako vrijede iznad date VZ, onda stvarno stanje stvari ne opisuje taj model, već ER model podataka sa slike 13.



Slika 13.

Iz potonjeg ER modela očito direktno slijede sheme relacija R1 i R2 (tj. PREDAJE i LITERATURA), koje su ranije bile generirane dekompozicijom sheme relacije KOLEGIJ2 dobivene iz nekorektnog ER modela podataka. Dakle, normalizacija ima i u ovom slučaju samo ulogu otkrivanja (i/ili otklanjanja) grešaka učinjenih u procesu izrade ER modela podataka.

Naravno, veza KOLEGIJ2 *mogla bi biti i trojna*. Međutim, u tom slučaju na pripadnoj shemi relacije KOLEGIJ2 ne bi vrijedile iznad date VZ. Dakle, tada bi shema relacije KOLEGIJ2 bila ne samo u BCNF već i u 4NF. A to znači da bi i ER model podataka sa slike 12 tada dao shemu relacije koja jeste "u optimalnoj normalnoj formi".

9. Zavisnost spajanja

Postoje relacije koje se ne mogu dekomponirati bez gubitka informacija na dvije relacije, ali se mogu bez gubitka informacija dekomponirati na tri ili više relacija. Pogledajmo primjer. Neka bude data shema relacije

KOLEGIJ3(Š-PRED, Š-NAST, Š-UDŽ)

na kojoj nisu definirane nikakve netrivijalne VZ. Dakle, ta shema relacije je u 4NF. Međutim, ekstenzija relacije KOLEGIJ3 data na slici 14 sadrži redundancu.

KOLEGIJ3

Š-PRED	Š-NAST	Š-UDŽ
P1	N1	U1
P1	N1	U2
P1	N2	U1
P2	N1	U1

Slika 14.

Na primjer, podatak da nastavnik N1 predaje predmet P1 zapisan je toliko puta koliko udžbenika taj nastavnik koristi za taj predmet. Međutim, obzirom da nastavnik N2 za *isti* predmet *ne* koristi isti skup udžbenika kao i nastavnik N1, na shemi relacije KOLEGIJ3 *ne* vrijedi VZ

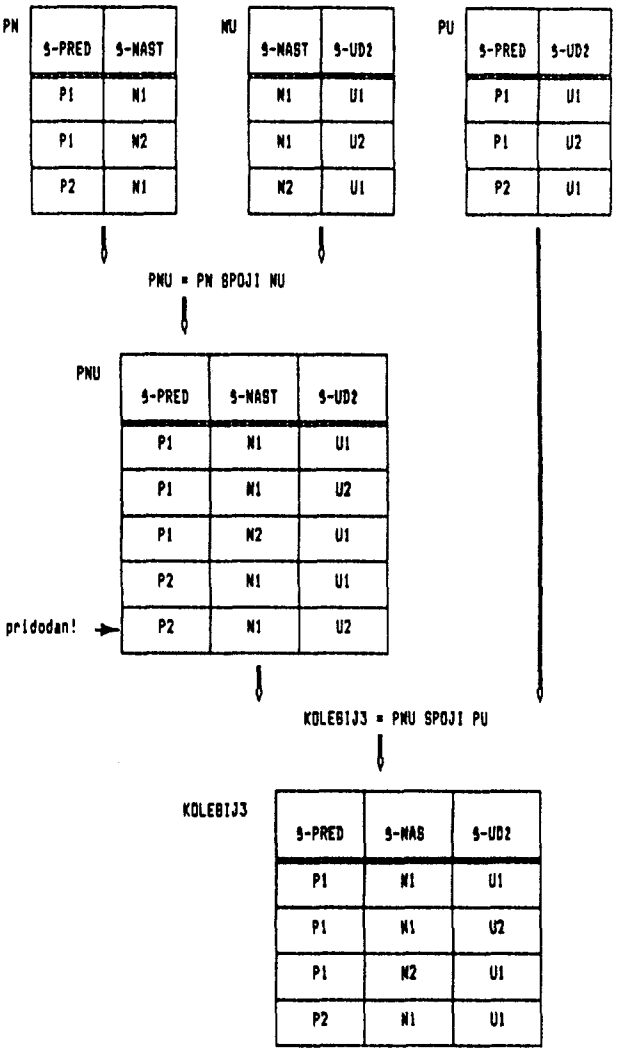
Š-PRED -->> Š-NAST | Š-UDŽ.

Odatle slijedi da tu shemu relacije nije moguće dekomponirati na dvije sheme relacija (na temelju VZ), kako je to učinjeno sa shemom relacije KOLEGIJ2.

Međutim, na slici 15 pokazano je da se ta relacija dade dekomponirati na *tri* relacije, i to bez gubitka informacija.

Relacija PNU, dobivena spajanjem relacija PN i NU sadrži jednu suvišnu n-torku. Drugim riječima, rezultat tog spajanja sadrži i tvrdnju da za predmet P2 nastavnik N1 koristi i udžbenik U2, što - prema relaciji KOLEGIJ3 - nije istina. Dakle, dekompozicija relacije KOLEGIJ3 *samo* na relacije PN i NU dovela bi do gubitka informa- cijskog sadržaja relacije KOLEGIJ3. Međutim, spajanje relacije PNU sa relacijom PU daje točno polaznu relaciju KOLEGIJ3.

Obzirom da na shemi relacije KOLEGIJ3 nisu definirane nikakve netrivijalne VZ, tro-



Slika 15.

dekomponibilnost te relacije ne može biti formalno utemeljena (obrazložena) na osnovu takvih zavisnosti. Stoga je uvedena (definirana) nova vrsta zavisnosti: *zavisnost spajanja*.

> Na shemi relacije R vrijedi *zavisnost spajanja* (ZS)
>
> *(R1, ..., Rn)
>
> ako i samo ako je R1, ..., Rn dekompozicija od R bez gubitka informacija.
>
> Zavisnost spajanja *(R1, ..., Rn) nazivamo *trivijalnom* ako je R jednaka nekoj od Ri, i =< i =< n.

Zavisnost spajanja omogućava da se iznad dato svojstvo tro-dekomponibilnosti (odnosno, općenito n-dekomponibilnosti) definira *na razini sheme relacije*. Za promatrani primjer to činimo pomoću slijedeće zavisnosti spajanja:

*(PN, NU, PU).

Naravno, time smo ujedno definirali *klasu legalnih ekstenzija* relacije KOLEGIJ3. Stoga, takvu ZS ne valja definirati samo na temelju jedne ekstenzije relacije, već to smijemo učiniti samo ako ta ZS iskazuje stvarne odnose u realnom svijetu. O tom (problemu!) biti će više riječi u slijedećem odjeljku.

Iz definicije ZS slijedi da su FZ i VZ samo *posebni slučajevi* ZS. Već je ranije pokazano da je FZ samo poseban slučaj VZ. Nadalje, pokazano je da se shema relacije oblika R(X, Y, Z) može dekomponirati (bez gubitka informacija) na sheme relacija R1(X, Y) i R2(X, Z) ako na shemi relacije R vrijedi VZ

X -->> Y | Z.

Prema definiciji ZS, to ujedno znači da na shemi relacije R vrijedi ZS

*(XY, XZ).

Dakle, gornja VZ dade se na ekvivalentan način izraziti kao ZS.

S druge strane, očito postoje ZS koje nisu VZ. Takva je, na primjer, ZS *(PN, NU, PU), definirana na shemi relacije KOLEGIJ3, na kojoj nije definirana nikakva VZ.

Iz definicije ZS slijedi da je to *najopćenitiji mogući* oblik zavisnosti, sve dok se sheme relacija dekomponiraju primjenom operacije projekcije i regeneriraju primjenom operacije spajanja.

Kao i prethodne zavisnosti, ZS daje osnovu za definiranje nove (a u kontekstu projekcije-spajanja i *najviše moguće*) normalne forme.

10. Peta normalna forma

Shema relacije na kojoj su definirane netrivijalne ZS može se dalje dekomponirati, i to upravo na one sheme relacija koje su date samim zapisom ZS. Cilj takve dekompozicije jeste doseći *petu normalnu formu* shema relacija, kao najvišu moguću normalnu formu, koja definitivno ne dovodi do nikakvih redundanci ni anomalija održavanja.

> Shema relacije R nalazi se u *petoj normalnoj formi* (5NF) ako i samo ako za svaku netrivijalnu ZS oblika
>
> *(R1, ..., Rn)
>
> definiranu na R vrijedi da je svaki A(Ri), 1 =< i =< n, ***superključ od R.***

Jednostavnije rečeno, shema relacije je u 5NF ako se *ne da* "suvislo dekomponirati". Pokušajmo to ilustrirati na primjeru sheme relacije

OSOBA(MAT-BROJ, IME, GOD-ROĐ).

Takvom shemom relacije može u relacijskom jeziku biti predstavljen objekt OSOBA iz ER modela podataka. Obzirom da je MAT-BROJ ključ sheme relacije OSOBA, na toj shemi relacije zacijelo vrijedi FZ

MAT-BROJ --> IME.

Polazeći od te FZ, shemu relacije OSOBA može se dekomponirati (prema ranije iznijetom principu) na sheme relacija

OSOBA1(MAT-BROJ, IME),
OSOBA2(MAT-BROJ, DAT-ROĐ).

Prema definiciji ZS, to ujedno znači da na shemi relacije OSOBA vrijedi ZS

*((MAT-BROJ, IME), (MAT-BROJ, DAT-ROĐ)).

Međutim, primjetimo da je ključ sheme relacije OSOBA (tj. MAT-BROJ) sadržan u obje sheme relacija generirane dekompozicijom. Dakle, A(OSOBA1) i A(OSOBA2) *jesu superključevi* sheme relacije OSOBA. A, prema definiciji 5NF, to znači da relacija OSOBA *jeste* u 5NF. Dakle, kao što je *objekt* OSOBA *jedan entitet* ER modela podataka, tako je i *shema relacije* OSOBA u 5NF, te ne postoje *formalni* razlozi za njeno dalnje dekomponiranje.

S druge strane, shema relacije KOLEGIJ3, uz pretpostavku da na njoj vrijedi ZS

*(PN, NU, PU),

nije u 5NF. Naime, ključ sheme relacije KOLEGIJ3 je trojka atributa

<Š-PRED, Š-NAST, Š-UD2>,

tako da A(PN), A(NU) i A(PU) nisu superključevi sheme relacije KOLEGIJ3. Stoga je ta shema relacije mogla biti "suvislo dekomponirana" na sheme relacija PN, NU i PU.

Sheme relacija PN, NU i PU jesu u 5NF jer se binarne relacije ne da netrivijalno dekomponirati, tako da na njima nema netrivijalnih ZS. Stoga, svaka binarna relacija jeste u 5NF.

Postavlja se pitanje da li bi (i na temelju čega) i iz ER modela podataka slijedile sheme relacija PN NU i PU, a ne shema relacije KOLEGIJ3. Prije odgovora, iznesimo slijedeća činjenice o 5NF.

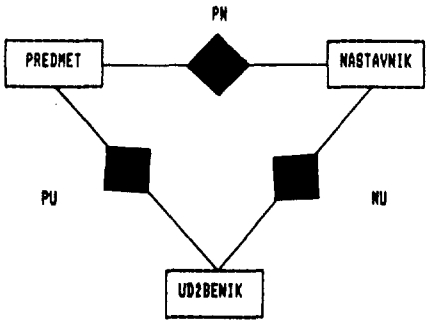
Obzirom da je ZS najopćenitiji oblik zavisnosti, i 5NF najviša normalna forma, mogli bismo zaključiti da ZS i 5NF imaju dominantnu ulogu u

oblikovanju modela podataka. No, tome nije baš tako. Na primjer, u <BRA 87> nalazimo tvrdnju da se 5NF (a time ni ZS) u praksi "općenito ne koriste". Razlog za to iznosi Date:

> "... dok je *otkrivanje* FZ i VZ relativno jednostavno, ..., isto se ne bi moglo reći za ZS (koje nisu VZ), jer je *intuitivno značenje ZS daleko od jednostavnog*. Stoga je i *postupak* utvrđivanja *kada* data relacija jeste u 4NF ali nije u 5NF (te bi mogla biti dalje dekomponirana) još uvijek *nejasan*."
> (Podcrtao M.R.)

Napomenimo da je u <DAT 86> gornja tvrdnja (iz prethodnog izdanja iste knjige) nešto ublažena: umjesto "daleko od jednostavnog" nalazimo "može ne biti očito". Međutim, zaključak o *nejasnoći postupka otkrivanja* ZS ostaje neizmjenjen. Date, nadalje, sugerira mogućnost da su relacije koje jesu u 4NF a nisu u 5NF "patološki slučajevi" koji se, izgleda, rijetko javljaju u praksi. Taj stav, zajedno sa ranije rečenim o BCNF i 4NF, potkrepljuje ocjenu da je sa praktičkog aspekta 3NF najvažnija. Naime, ako je ER model podataka korektno izrađen (i ako relacija nije "patološka"), onda će sheme relacija generirane iz tog ER modela koje jesu u 3NF biti ujedno i u BCNF, 4NF i 5NF. Stoga, ne izgleda primjerenim reći da se 5NF "u praksi općenito ne koristi", već da se (u praksi) ne pokušava *eksplicitno* doseći. Međutim, to ne znači da sheme relacije generirane iz dobro oblikovanih ER modela podataka nisu *i* u 5NF.

Iznad rečeno ujedno ukazuje na odnos ER modela i 5NF. Naime, ako se *uspije otkriti* zavisnost spajanja poput *(PN, NU, PU) onda ista može biti iskazana tako da se umjesto trojne veze KOLEGIJ3 (koja je po formi jednaka vezi KOLEGIJ2 sa slike 12), ER modelom iskažu *tri binarne veze*, kako je to učinjeno na slici 16.



Slika 16.

Odatle direktno slijede i sheme relacija PN, NU i PU. Međutim, problem 5NF je u njenom *otkrivanju*. Točnije: u *otkrivanju ZS koje nisu VZ*. A taj problem je jednako prisutan u formalizmu normalnih formi kao i u (bitno) jednostavnijem ER jeziku. Dakle, i s tog aspekta, ER jezik nudi mogućnost izrade jednako kvalitetnog modela podataka kao i "sofisticirani" proces normalizacije.

REFERENCE

<BRA 87> Brackett, H.M.:
Developing Data Structured Databases, Prentice-Hall, 1987.

<CHE 76> Chen, P.P.:
The Entity-Relationship Model - Toward a unified View of Data, *ACM Trans. on Database S.*, No.1, 1976.

<DAT 86> Date, C.J.:
Introduction to Database Systems, Vol I, Addison-Wesley, 1986.

<KOR 86> Korth, F.H., Silberschatz, A.:
Database System Concepts, McGraw-Hill, 1986.

<MAI 83> Maier, D.:
The Theory of Relational Databases, Comp. Sci. Press, 1983.

<NAV 86> Navathe, S., Elmasri, R., Larson, J.:
Integrating Users Views in Database Design, *IEEE Cmputer*, No. 1, 1986.

<TKA 88> Tkalac, S.:
Relacijski model podataka, Informator (u tisku).

<TOR 86> Torey, J.T., Yang, D., Fry, P.J.:
A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model, *ACM Computimg Surveys*, No. 2, 1986.

Deskriptorji: ELEKTRONSKI IMENIK, KOMUNIKACIJE, IMENIK PORAZDELJENI

Helena Tvrdy
IJS Ljubljana

POVZETEK Opisani so osnovni koncepti elektronskega imenika: funkcionalni, organizacijski in informacijski model. Prikazana so načela, po katerih je fizično in logično porazdeljena informacija v imeniku. Podan je pregled servisov, ki jih nudi imenik. Opisano je delovanje imenika kot porazdeljene aplikacije, specificirano je obnašanje posameznih sistemskih posrednikov imenika, ki pri tem sodelujejo. Našteti so koncepti, ki še niso standardizirani.

ABSTRACT The basic concepts of the Directory are described. The functional, organizational and informational models are introduced. The principles of the logical and physical distribution of the directory information base are given. The services provided by the Directory to its users are presented. The operation of the Directory as a distributed application is described; the behaviour of the directory system agent, that participates in this application is specified. The concepts which are not standardized yet are presented.

1 Uvod

Uporabo komunikacijskih storitev, ki jih omogočajo računalniške mreže (elektronska pošta; prenos in obdelava poslov; prenos, dostop in upravljanje datotek; virtualni terminal; ...) v veliki meri omejujeta čas in napor, potrebna za pridobitev informacij o možnih storitvah, o možnih komunikacijskih partnerjih, o načinu vzpostavitve in uporabi storitev, itd.

Tudi izvajalci storitev se soočajo s številnimi težavami, povezanimi z izvajanjem in upravljanjem le-teh. Pomoč v obliki ustreznih informacij potrebujejo za vključevanje in brisanje naročnikov (uporabnikov); za dogovarjanje z naročniki (na primer za pooblastila); za dogovarjanje z drugimi izvajalci; za merjenje porabe in obračunavanje storitev; ...

Sodobno prepričanje je ([But86], [Ver86], [Ami88]), naj bi se vse informacije, potrebne za različne komunikacijske storitve, *zajele enotno* in *hranile* v nekem univerzalnem *porazdeljenem elektronskem imeniku*. Ta porazdeljeni elektronski imenik (ali na kratko: *imenik*) naj bi združil:

- informacije o osebkih, ki komunicirajo, o izvajalcih storitev in o uporabnikih storitev;
- informacije glede na različna organizacijska področja;
- informacije, ki so potrebne za vse različne komunikacijske storitve.

Ta združitev podatkov naj bi ohranila avtonomnost administrativnih in organizacijskih območij ter sistemov nad vzdrževanjem dela podatkov, za katere so zadolženi.

Imenik bi uporabljali na dva načina:

1. kot informacijsko storitev (neposredni uporabniki so ljudje);
2. v povezavi z ostalimi storitvami – kot pomoč tem storitvam (neposredni uporabniki so aplikacije, na primer elektronska pošta ([Bon88])).

Z obravnavanjem ter izvedbo imenskih strežnikov in elektronskih imenikov se je in se še ukvarja več raziskovalnih projektov. Taka sta bila na primer projekta: Grapevine pri XEROX-PARC in V-System, Spice-project ([Lan86]). DFN (Deutsches Forschungsnetz) načrtuje porazdeljeni imenik ([Ver86], [Ver88]) predvsem kot pomoč pri uporabi njihovega sistema za izmenjavo sporočil (MHS). Projekt THORN (THe Obviously Required Nameserver) ([Kil88], [Sir87]), ki je eden od projektov v ESPRIT (European Strategic Programme for Research into Information Technology), se ukvarja z razvojem in izvedbo storitev imenika, v skladu z OSI.

S standardizacijo na področju imenika se je prva začela ukvarjati ECMA (European Computer Manufacturing Association) ([ECMA85]). Od leta 1986 sta na tem področju intenzivneje začela delovati tudi ISO/TC97/SC21/WG4 in CCITT Study Group VII (Question 35), ki probleme rešujeta skupno in izdajata skupne dokumente ([ISO87]).

V članku so v poglavjih 2–8 opisani koncepti in načela, ki naj bi jih izvajalci upoštevali pri izvedbi porazdeljenega imenika. V poglavju 9 pa je navedeno, katera od obravnavanih področij še niso standardizirana. Podatke sem črpala pretežno iz ([COS88], [ISO87], [VER88]); v poglavju 10 pa so navedeni še ostali viri, na katere sem naletela pri študiju imenikov.

2 Funkcionalni model

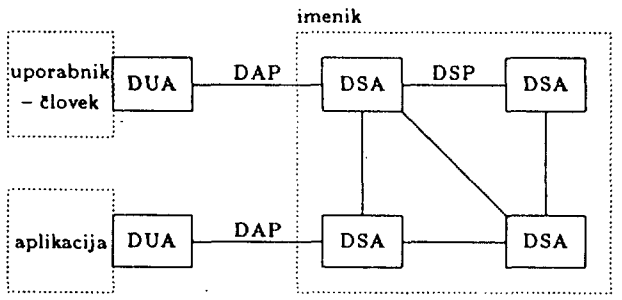
S stališča uporabnika je imenik objekt, ki:

- vsebuje podatkovno bazo, imenovano *informacijska baza za imenik* (DIB – Directory Information Base) in ki
- izvaja dostop do te baze.

DIB vsebuje vhode; vsak vhod predstavlja neki *objekt*. Objekt je karkoli iz realnega sveta (človek, organizacija, aplikativni proces, materialna oprema, ...). Edino, kar zahtevamo od objekta je, da se da identificirati (ima ime).

Imenik tvori množica enega ali več aplikativnih procesov, imenovanih *sistemski posrednik imenika* (DSA – Directory System Agent). Če je imenik sestavljen iz več kot enega DSA, pravimo, da je *porazdeljen*. Uporabniki imenika (ljudje ali aplikacije) dostopajo do imenika preko *uporabniškega posrednika imenika* (DUA – Directory User Agent)(glej sliko 1). DUA je aplikativni proces, ki s stališča imenika predstavlja enega uporabnika.

Imenik nudi svojim uporabnikom množico dostopnih možnosti, imenovano *abstraktni servis imenika*. Oskrbovanje in uporabljanje servisov imenika zahtevata od uporabnika (oziroma dejansko od DUA) in od ostalih funkcionalnih komponent imenika (DSA), da znajo med seboj sodelovati in komunicirati. Če se DUA in DSA nahajata v različnih odprtih sistemih, uporabita *dostopni protokol imenika* (DAP – Directory Access Protocol): DUA pri posredovanju zahteve DSA ter DSA pri vrnitvi odgovora DUA. Vsak DSA ima direktni dostop do informacije, ki jo vsebuje sam;



Slika 1: Funkcionalni model imenika

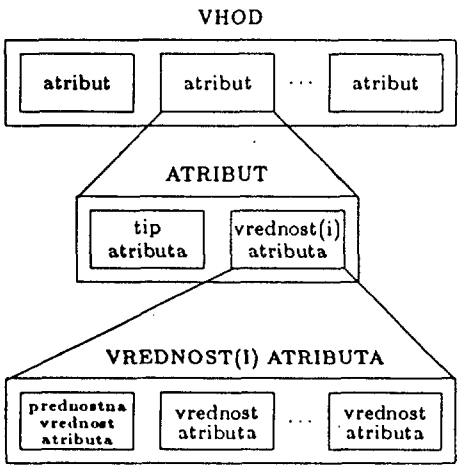
do preostale DIB pa pride tako, da komunicira z drugimi DSA z uporabo *sistemskega protokola imenika* (DSP – Directory System Protocol).

3 Organizacijski model

Množico enega ali več DSA ter nobenega, enega ali več DUA, ki jo upravlja ena sama organizacija, imenujemo *področje upravljanja imenika* (DMD – Directory Management Domain). Področje upravljanja lahko vodi *pošta* (ADDMD – Administration DMD) ali pa kaka druga *organizacija* (PRDMD – Private DMD). Področje upravljanja skrbi za specifikacijo komunikacij med funkcionalnimi komponentami znotraj DMD in za nekatere vidike obnašanja DSA. Vsak uporabniški posrednik mora biti registriran pri enem področju upravljanja. Mednarodne povezave so možne le preko področij upravljanja, ki jih vodi pošta.

4 Informacijski model

Informacija o objektu zanimanja je shranjena v *vhodu*. Vhode, ki jih hrani in vzdržuje imenik, lahko obravnavamo – vsaj konceptualno – kot enotno skupino informacij, imenovano *informacijska baza za imenik* (DIB). Uporabnik vidi imenik kot eno samo podatkovno bazo, čeprav so realni sistemi za imenik fizično porazdeljeni.



Slika 2: Struktura vhoda

Vhod vsebuje enega ali več *atributov*, s katerimi so opisane lastnosti objekta. Atributi, ki opisujejo objekt *oseba*, so na primer: ime in priimek; domači naslov; domača telefonska številka; delovna organizacija, v kateri je oseba zaposlena; službena telefonska številka; funkcija, ki jo ima oseba v delovni organizaciji;...

Vsak atribut sestavljajo *tip atributa*, ki določa vrsto informacije, ki jo opisuje ta atribut in ena ali več *vrednost(i)* tega tipa atributa. Tip atributa je predstavljen z *identifikatorjem objekta*.

Objekti, ki vsebujejo iste tipe atributov, pripadajo istemu *objektnemu razredu*. Vsi objektni razredi so bodisi neposredni, bodisi posredni *podrazredi najvišjega* razreda. Podrazredi podedujejo definicije svojega(-ih) *nad*razreda(-ov). Če na primer v najvišjemu razredu definiramo objekt s tipom atributa "objektni razred", pomeni, da vsebujejo ta tip atributa tudi vsi podrazredi, torej vsak vhod v imeniku.

4.1 Koncept imenovanja

Za uspešno upravljanje vhodov v imeniku je potrebno ustrezno rešiti problem upravljanja imenskega prostora. (Imenski prostor sestavljajo vsa imena, ki jih na osnovi pravil lahko tvorimo nad dano abecedo).

Imenik uporablja hierarhično drevesno strukturo imenskega prostora, ki jo imenujemo *informacijsko drevo za imenik* (DIT – Directory Information Tree). Zaradi hierarhičnega pristopa, se da problem dodeljevanja enoličnih imen vhodom rešiti brez centralne administracije.

V hierarhičnem imenskem prostoru imenika se pojavljata dve konceptualni obliki imen: *del prednostnega imena* – DPI (relative distinguished name) in *prednostno ime* (distinguished name). Vsak vhod – natančneje: upravljalec, odgovoren za ta vhod – je pooblaščen za dodeljevanje enoličnih imen (delov prednostnih imen) naslednikom tega vhoda. Za del prednostnega imena vhoda zadostuje, da je enoličen znotraj omejenega področja – le med vhodi, ki imajo istega predhodnika v DIT. Prednostno ime vhoda je zaporedje, sestavljeno iz delov prednostih imen vseh predhodnih vhodov (začenši pri korenini) in DPI tega vhoda. Ker so vsi DPI enolični znotraj posameznih področij, je prednostno ime enolično znotraj celega imenika. Odnos med deli prednostnih imen in prednostnimi imeni prikazuje slika 3.

DPI so oblikovani iz atributov, zato moramo, če želimo doseči neki vhod v imeniku, le-tega podati v obliki zaporedja, v katerem je vsak člen sestavljen iz tipa atributa in vrednosti atributa.

4.2 Alternativna imena

Prednostno ime mora biti enolično, ni pa nujno, da je to edino ime objekta. Vsak objekt ima lahko poleg prednostnega imena še eno ali več *alternativnih imen*. *Alternativni vhod* ne vsebuje nobene informacije o objektu, ampak kaže na vhod, ki vsebuje prednostno ime danega objekta. Z alternativnimi imeni lahko tvorimo uporabniško prijaznejša imena objektov.

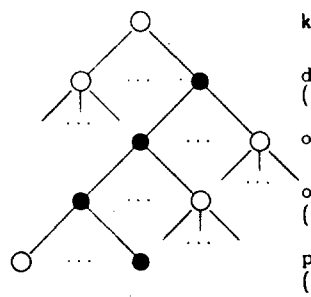
Obratni kazalci – to je kazalci, ki bi kazali iz vhoda (na katerega kaže neki alternativni vhod) na alternativne vhode tega vhoda – ne obstajajo. Zato alternativni vhodi danega vhoda ostanejo v imeniku tudi, če dani vhod odstranimo iz imenika.

4.3 Shema imenika

Shema imenika določa, katera informacija je lahko shranjena v DIB in kakšne so logične relacije med vhodi. Struktura vhodov, tipi informacij, ki jih vhodi vsebujejo ter predstavitev teh informacij so definirani z objektnimi razredi, tipi atributov in sintaksami atributov (slika 4).

Shema obsega množico definicij:

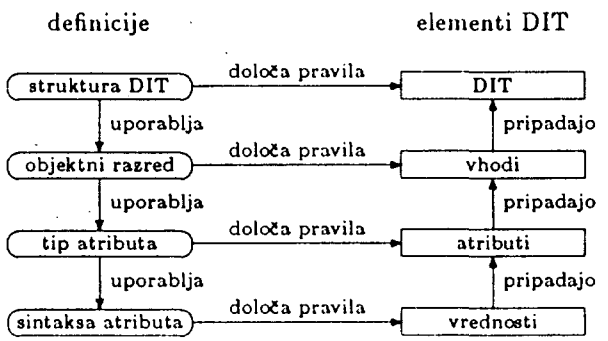
- o *strukturi DIT*,
  ki določajo pravila, po katerih so oblikovana prednostna imena vhodov, ter odnose med posameznimi vhodi v DIT;



	DPI	prednostno ime
korenina	{}	{}
države (C-country)	{C=YU}	{C=YU}
organizacije	{O=IJS}	{C=YU,O=IJS}
organizacijske enote (OU-organisational unit)	{OU=E6}	{C=YU,O=IJS,OU=E6}
posamezni objekti (CN-common name)	{CN=Tvrdy}	{C=YU,O=IJS,OU=E6, CN=Tvrdy}

Slika 3: Hierarhični imenski prostor

- *objektnih razredov*,
  ki določajo obvezne in neobvezne atribute, ki morajo oziroma smejo biti prisotni v nekem vhodu iz danega objektnega razreda;
- *tipov atributov*,
  ki za posamezni atribut določajo njegov objektni identifikator, njegovo sintakso in ali sme imeti več vrednosti;
- *sintaks atributov*,
  ki za vsak atribut določajo ustrezni podatkovni tip in pravila primerjanja.



Slika 4: Shema imenika

Shema imenika je (tako kot DIB) porazdeljena in je lahko v vsakem DSA različna. Upravljalec, ki je pooblaščen za dani DSA, sam določi shemo, ki bo veljavna v delu DIB, za katerega je zadolžen.

5 Servisi imenika

Imenik izvaja servise na zahteve uporabnikov, ki jih zastopajo njihovi DUA. Nekateri servisi omogočajo preiskovanje imenika, drugi pa njegovo spreminjanje. Za vsako zahtevo imenik vedno vrne odgovor, v katerem sporoči rezultat izvedene zahteve, ki pa je odvisen od parametrov podanih v zahtevi. Ti parametri so:

- *servisna krmila*, ki omogočajo uporabnikom, da omejijo uporabo imenika; omejitve se nanašajo na čas dovoljen za izvedbo zahteve, na dimenzijo rezultatov, na način medsebojnega sodelovanja DSA, na prioriteto zahteve,...;
- informacija, potrebna za izvajanje *mehanizmov zaščite*;
- *filtri*, ki se uporabljajo za izbor ustreznih vhodov pri operacijah, ki se nanašajo na več vhodov; filter sestavlja množica pravil za primerjanje, ki so med seboj logično povezana z disjunkcijo, konjunkcijo in negacijo.

Servise (oziroma operacije), ki jih izvaja imenik, razdelimo v tri kategorije:

1. *eno-objektne operacije*:
   so operacije, ki učinkujejo le na en objekt in se izvedejo v DSA, ki vsebuje vhod, ki predstavlja ta objekt:
   - *beri*:
     imenik prebere določeni vhod in vrne nekatere ali vse atribute tega vhoda;
   - *primerjaj*:
     imenik primerja podano vrednost z vrednostjo določenega atributa v vhodu;
   - *dodaj_vhod*:
     imenik doda nov list (končni vhod) v DIT;
     odstrani_vhod:
     imenik odstrani list iz DIT;
   - *spremeni_vhod*:
     imenik izvede na podanem vhodu eno od naslednjih sprememb: doda, odstrani ali zamenja atribut oziroma njegovo vrednost;
   - *spremeni_DPI*:
     imenik spremeni DPI podanega lista;
2. *več-objektni operaciji*:
   sta operaciji, ki delujeta na več vhodih, ki so, ali pa niso vsi v istem DSA:
   - *naštej*:
     imenik vrne seznam neposrednih naslednikov podanega vhoda v DIT;
   - *išči*:
     imenik vrne vse vhode iz določenega dela DIT, ki zadoščajo pogojem navedenim v filtru;
3. *operacija sproščanja*:
   - *prekini*:
     imenik konča z izvajanjem trenutne zahteve, ker povpraševalca rezultati ne zanimajo več.

6 Varnost in zaščita

6.1 Overovljanje

Overovljanje (preverjanje identitete uporabnika) je pomembni razlog za uporabo imenika v aplikacijah. Imenik uporabljamo izvajanje funkcij overovljanja, ker je zanje potrebna globalna podatkovna baza, v kateri so shranjeni javni ključi (public keys) in z njimi povezane informacije.

Za *enostavno overovljanje* s strani imenika zadostuje, da povpraševalec dokaže svojo identiteto (specificirano v imeniku) tako, da navede svoje ime in geslo. Če pa so v imeniku shranjeni zaupni podatki, enostavno overovljanje ne zadošča. Za preprečitev zlorab je potrebno uvesti servis overovljanja, ki nudi večjo varnost (*strogo overovljanje*).

6.2 Nadzor dostopa

Zaščito podatkov pred nedovoljeno uporabo izvedemo z nadzorom dostopa. Za vsak atribut definiramo, kateri uporabnik in pri kateri operaciji lahko uporablja dani atribut.

Nadzor dostopa realiziramo z dostopnimi pravicami za:

- branje zaščitenih atributov;
- zaznavanje zaščite atributov;
- spreminjanje zaščitenih atributov;
- oblikovanje / brisanje atributov oziroma vhodov.

Dostopne pravice se nanašajo na operacije v imeniku. Med izvajanjem servisov mora imenik najprej preveriti dostopne pravice. Če le-te ustrezajo, lahko izvede ustrezno predajo podatkov. Operacije na informaciji, ki zahtevajo posebno overovljanje, lahko izvede le po vnaprejšnji identifikaciji povpraševalca.

7 Porazdeljenost

Porazdelitev DIB (ki navidezno hrani vse vhode imenika na enem mestu) med različne DSA (ki fizično vsebujejo vhode) temelji na principu, da lahko vsak DSA vsebuje enega ali več poddreves v DIT.

Vsakega od vhodov, ki so neposredni nasledniki korenine v DIT, lahko dodelimo nekemu DSA, ki odslej vsebuje popolno informacijo o tem vhodu. Imenujemo ga *izvorni DSA* za ta vhod. Upravljalec tega DSA je zadolžen za vzdrževanje in upravljanje vhoda; pooblastilo za upravljanje vseh ali nekaterih izmed naslednikov tega vhoda, lahko preda drugemu(-im) DSA. Nasledniki tega vhoda postanejo korenine novih poddreves, ki jih na isti način lahko porazdelimo naprej.

Ta postopek omogoča zelo prilagodljivo porazdelitev vhodov: vsak DSA lahko vsebuje poljubno število popolnih ali delnih poddreves. Poddrevesa, ki jih vsebuje posamezni DSA, so si lahko celo disjunktna (korenine med seboj disjunktnih poddreves nimajo istega predhodnika).

7.1 Kopiranje

Kopiranje informacij je v imeniku potrebno za:

1. *zagotovitev visoke uporabnosti imenika*:
   informacija je v imeniku porazdeljena in shranjena na različnih sistemih; če se kateri od teh sistemov pokvari, ali če (iz kakršnihkoli razlogov) ni dosegljiv, potem so objekti imenika, ki se nahajajo v sistemu, nedosegljivi; s stališča uporabnosti imenika je torej nujno, da se nahajajo kopije posameznih informacij na več mestih;
2. *izboljšanje sposobnosti imenika*:
   večina dostopov do imenika se nanaša na iskanje in ne na ažuriranje informacije; več kopij iste informacije omogoča lokalno iskanje – namesto sicer potrebnega vstopanja v oddaljene mreže in s tem povezanih zakasnitev.

Vsak vhod ima natanko eno *izvorno kopijo* (master copy) in morebitne *neizvorne kopije* (non-master copy). Izvorna kopija vhoda leži le v enem DSA – izvornem DSA ostali DSA pa lahko vsebujejo neizvorne kopije tega vhoda. Vzdrževanje in upravljanje nekega vhoda se vedno izvaja v izvornem DSA. Izvorna kopija je vedno popolna kopija vhoda (vsebuje del prednostnega imena in vse atribute).

Neizvorna kopija je bodisi:

- popolna (vsebuje vso informacijo iz vhoda) in se uporablja za optimalnejši (hitrejši in direktni) dostop do informacije; bodisi je
- nepopolna (ne vsebuje vseh atributov; vsebuje na primer le atribute, potrebne za nadaljnje usmerjanje zahtev, ne vsebuje pa atributov, ki opisujejo dani vhod).

V splošnem lahko neizvorno kopijo dobimo na dva načina:

- z dogovorom med dvema DSA, da bo tisti, ki vsebuje izvorno kopijo nekega vhoda pošiljal vse spremembe, ki jih "doživlja" informacija tega vhoda, drugemu DSA. Ta način imenujemo *sledenje* informacije (shadowing);
- če se DSA med izvajanjem operacije dokoplje do informacije, ki je sam ne vsebuje (nima izvorne kopije), se lahko odloči, da to informacijo obdrži. Ta način pridobivanja informacij imenujemo *skladiščenje* informacije (caching).

Uporabnik imenika mora biti obveščen, ali je informacija, ki jo dobi kot odgovor na zahtevo, dobljena iz kopije ali iz originala.

7.2 Znanje

Zaradi porazdeljenosti DIB mora vsak DSA vsebovati neko *znanje*, ki mu omogoča določiti, kateri DSA vsebuje zahtevano informacijo. DSA vsebuje toliko znanja, da je iz vsakega DUA možen dostop do vsakega vhoda. Del znanja uporablja za določitev, ali vsebuje zahtevani vhod v katerem od poddreves; drugi del pa za usmerjanje zahtev na druge DSA.

Poddrevesa, ki jih vsebuje DSA, opišemo s pojmom *konteksta imenovanja*. Kontekst imenovanja je sestavljen iz *kontekstne predpone* (prednostnega imena korenine danega poddrevesa) in iz *referenc*, ki kažejo, kje (v katerem DSA) se nahajajo vhodi danega poddrevesa (*notranje reference* in *reference na naslednike*) in, kje se nahaja predhodnik korenine poddrevesa (*referenca na predhodnika*).

Slika 5 prikazuje DIT, ki je logično razdeljena med pet kontekstov imenovanja (KI-A, KI-B, KI-C, KI-D in KI-E) in ki je fizično porazdeljena med tri sistemske posrednike (DSA-a, DSA-b in DSA-c).

Kontekste imenovanja, ki jih vsebujejo posamezni DSA, v splošnem oblikujemo tako, da lahko zadostijo čimvečjemu številu zahtev (po najrazličnejših operacijah):

- nekatere DSA oblikujemo (na primer) tako, da vsebujejo le vhode, ki predstavljajo področja imenovanja na višjih nivojih (znotraj nekega logičnega dela DIB), pri čemer ni nujno, da vsebujejo tudi vse podrejene vhode;
- druge DSA pa oblikujemo tako, da vsebujejo kontekste imenovanja, ki predstavljajo predvsem končne vhode.

Minimalno znanje

Ker mora imenik zagotoviti dostop do vsakega vhoda, obstaja za vsak DSA tako imenovana *referenčna pot* (reference path), to je zvezno zaporedje referenc, ki vodijo iz danega DSA do vseh kontekstov imenovanja v imeniku. Množico vseh teh referenc in vseh DSA si lahko predstavljamo kot povezan graf; pri čemer minimalno množico referenc, ki še zagotavljajo pravilno obnašanje imenika, obravnavamo kot ogrodje tega grafa. Zmogljivost imenika povečamo, če osnovnemu ogrodju dodamo dodatne prvine (nove reference).

DSA mora vsebovati in vzdrževati:

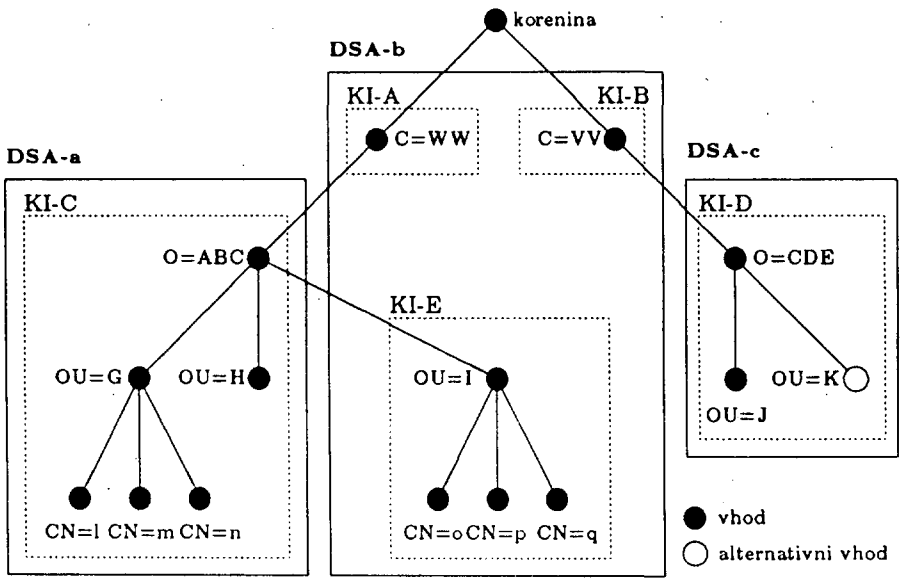
- *znanje o predhodnikih* in
- *znanje o naslednikih* (če seveda obstajajo).

Kontekst korenine

Funkcijo "koreninskega DSA" izvajajo vsi DSA, ki so neposredni nasledniki korenine. Te DSA imenujemo *DSA prvega nivoja*. Vsak DSA prvega nivoja zna simulirati koreninski DSA. To pomeni, da ima popolno znanje o kontekstu korenine. Ker je kontekst korenine kopiran (repliciran) v vsak DSA prvega nivoja, so zanj skupno pristojni vsi (sicer med seboj avtonomno delujoči) upravitelji vseh DSA prvega nivoja, ki postopke za upravljanje konteksta korenine določajo na osnovi medsebojnih sporazumov.

Povzemimo:

- vsak DSA prvega nivoja vsebuje kontekst korenine in zato tudi referenčno pot do vsakega drugega DSA prvega nivoja in



Slika 5: Primer DIT

- vsak DSA, ki ni prvega nivoja, vsebujejo referenco na predhodnika, kar pomeni, da poznamo referenčno pot do poljubnega DSA prvega nivoja.

Dodatno znanje

Optimalnejši dostop do drugih DSA je omogočen z uporabo *navzkrižnih referenc* in *nespecifičnih referenc na naslednike*. Navzkrižne reference vsebujejo "tuje" kontekstne predpone, ki omogočajo direktno interakcijo z DSA, ki vsebuje zahtevani vhod. DSA lahko vsebuje poljubno število navzkrižnih referenc; pridobiva pa jih med običajnim izvajanjem operacij v imeniku. Nespecifične reference na naslednike pa uporabljamo, kadar za neki DSA vemo, da vsebuje naslednike danega vhoda, ne poznamo pa DPI teh vhodov.

Opis posameznih tipov referenc

- *Notranja referenca* vsebuje:
  - del prednostnega imena, ki ustreza vhodu v DIB in
  - kazalec, ki kaže, kje je vhod lokalno shranjen.
- *Referenca na naslednika* vsebuje:
  - del prednostnega imena, ki ustreza neposrednemu nasledniku danega vhoda v DIB,
  - prednostno ime DSA, ki je pristojen za upravljanje naslednika in
  - predstavitveni naslov tega DSA.

  Vse znanje o vhodih, ki jih je dani DSA predal v pristojnost drugim DSA, je v danem DSA predstavljeno v obliki referenc na naslednike.
- *Referenca na predhodnika* vsebuje:
  - prednostno ime DSA, ki vsebuje neposrednega predhodnika korenine danega konteksta imenovanja in
  - predstavitveni naslov tega DSA.

  Vsak DSA, ki ni prvega nivoja, vsebuje natanko eno referenco na predhodnika. Ob oblikovanju novega DSA, ki ni prvega nivoja, moramo le-temu z referenco na predhodnika določiti vsaj minimalno začetno znanje. Dodatno znanje lahko uvedemo kasneje v obliki referenc na naslednike ali/in v obliki navzkrižnih referenc. Ob vključevanju novega DSA prvega nivoja pa moramo le-tega opremiti s kontekstom korenine, ostale (že obstoječe) DSA prvega nivoja pa obvestiti o njegovi vključitvi.
- *Navzkrižna referenca* vsebuje:
  - kontekstno predpono,
  - prednostno ime DSA, ki je pristojen za ta kontekst imenovanja in
  - predstavitveni naslov tega DSA.

  Ta tip reference je neobvezen in služi za optimalnejše razreševanje porazdeljenega imena. DSA lahko vsebuje poljubno število (tudi nič) navzkrižnih referenc.
- *Nespecifična referenca na naslednike* vsebuje:
  - prednostno ime DSA, ki vsebuje enega ali več neposredno sledečih kontekstov imenovanja in
  - predstavitveni naslov tega DSA.

  Tudi ta tip reference je neobvezen. V vsakem kontekstu imenovanja, ki ga vsebuje, ima DSA lahko poljubno število (tudi nič) nespecifičnih referenc na naslednike, ki jih med delovanjem (na primer med postopkom razrešitve imena) upošteva šele, ko neuspešno "obdela" že vse specifične reference (notranje in na naslednike).

7.3 Načini medsebojnega sodelovanja DSA

DSA lahko medseboj sodelujejo na enega od naslednjih treh načinov.

1. *Veriženje* (chaining)
   je način, pri katerem DSA, ki sam ne more razrešiti zahteve, preda zahtevo drugemu (le enemu!) DSA. To pomeni, da DSA preide na izvajanje oddaljene operacije na tistem DSA, za katerega pozna njegove kontekste imenovanja, podane bodisi z navzkrižno referenco, bodisi z referenco na naslednike, bodisi z referenco na predhodnika.
2. *Dodeljevanje večim* (multi-casting)
   je način, pri katerem DSA, ki sam ne more razrešiti zahteve, preda identično zahtevo enemu ali več drugim DSA hkrati ali zaporedno; to je preide na hkratno ali zaporedno izvajanje identične oddaljene operacije na tistih DSA, za katere vsebuje nespecifična referenca na naslednike. Običajno je sposoben nadaljevati izvajanje oddaljene operacije le en izmed sodelujočih DSA, vsi ostali pa vrnejo sporočilo, da izvedba ni možna.
3. *Namigovanje* (hinting)
   je način, pri katerem DSA, ki sam ne more razrešiti zahteve, vrne povprašujočemu (DUA ali DSA) namig, kateri DSA je bližji rešitvi. Namig vsebuje referenco (možna je vsaka referenca) na enega ali več DSA.

7.4 Upravljanje znanja

Če spremembe v enem DSA povzroče, da postanejo reference v drugih DSA napačne, je treba vse reference (na naslednike, na predhodnika in nespecifične na naslednike) ažurirati na osnovi medsebojnih dogovorov upraviteljev ustreznih DSA.

7.5 Protislovnost znanja

Vrsta protislovnosti (znanja) in zaznava te protislovnosti je različna za različne tipe referenc:

- *navzkrižne reference in reference na naslednike*:
  ta vrsta referenc je neveljavna, če referencirani DSA lokalno ne vsebuje konteksta imenovanja s kontekstno predpono, navedeno v referenci; protislovje zazna proces za razrešitev imena med iskanjem konteksta imenovanja (glej razdelek 8.2).
- *reference na predhodnika*:
  napačna referenca na predhodnika kaže na DSA, ki ne vsebuje konteksta imenovanja, ki bi bil nadrejen vsem kontekstom imenovanja, ki jih lokalno vsebuje dani DSA.

Pri veriženju se DSA, ki vsebuje napačno referenco, zave protislovnosti znanja potem, ko kot odgovor na verižen zahtevo dobi sporočilo o napačni referenci. Enak odgovor dobi pri namigovanju povpraševalec, ko hoče uporabiti (od drugega DSA dobljeni) napačni namig. Pri tem pa se DSA, ki je posredoval napačni namig, ne zaveda protislovnosti svojega znanja.

Potem, ko DSA zazna prisotnost napačnih referenc, skuša napake odpraviti in ponovno vzpostaviti neprotislovno stanje. To doseže tako, da (na primer) protislovne navzkrižne reference bodisi:

- pobriše, bodisi
- jih zamenja s pravilnimi, ki jih dobi s (ponovno) uporabo ustreznega mehanizma za pridobivanje navzkrižnih referenc.

7.6 Primer oblikovanja znanja

Slika 6 kaže, kakšno informacijo o znanju vsebuje in vzdržuje DSA-b s slike 5. Uporabljene so naslednje oznake:

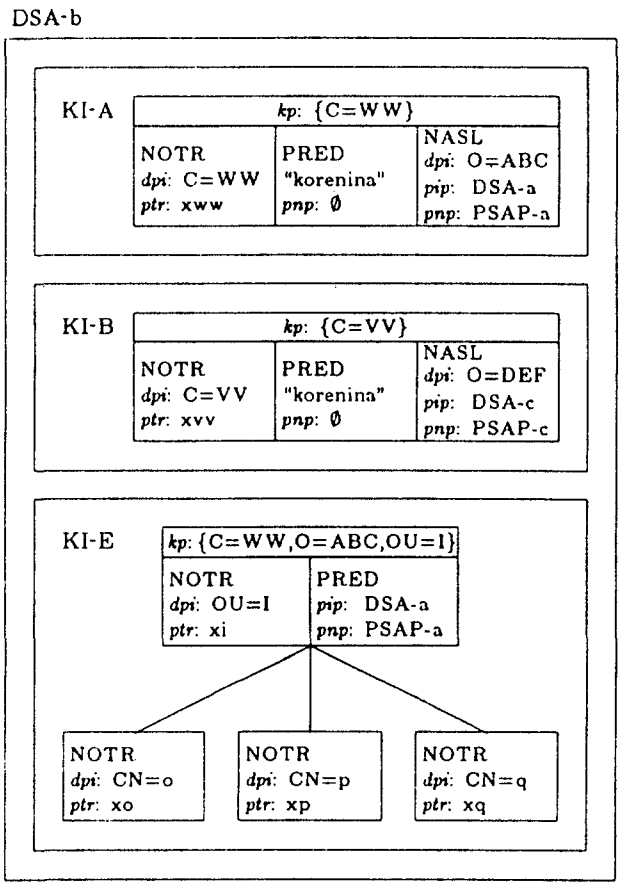
NOTR:	notranja referenca
PRED:	referenca na predhodnika
NASL:	referenca na naslednike
kp:	kontekstna predpona
dpi:	del prednostnega imena
ptr:	kazalec na lokalno pozicijo vhoda
pip:	prednostno ime posrednika
pnp:	predstavitveni naslov posrednika
DSA-n:	prednostno ime posrednika "DSA-n"
PSAP-n:	predstavitveni naslov posrednika "DSA-n"

Znanje, ki ga vsebuje DSA-b, sestavljajo:

kontekstne predpone:	{C=WW} za kontekst KI-A; {C=VV} za kontekst KI-B in {C=WW,O=ABC,OU=I} za kontekst KI-E.
navzkrižne reference:	∅
reference na predhodnika:	
(za kontekst KI-E):	na DSA-a s PSAP-a;
notranje reference:	
(za kontekst KI-A):	na vhod {C=WW}, na poziciji xww;
(za kontekst KI-B):	na vhod {C=VV}, na poziciji xvv;

(za kontekst KI-E):	na vhod {C=WW, O=ABC, OU=I}, na poziciji xi;
	na vhod {CN=o}, na poziciji xo;
	na vhod {CN=p}, na poziciji xp in
	na vhod {CN=q}, na poziciji xq.

reference na naslednike:	
(za kontekst KI-A):	na vhod {C=WW, O=ABC} v DSA-a s PSAP-a in
(za kontekst KI-B):	na vhod {C=VV, O=DEF} v DSA-c s PSAP-c.
nespecifične reference na naslednike:	∅



Slika 6: Znanje v DSA-b

8 Delovanje porazdeljenega imenika

Delovanje imenika lahko opazujemo iz dveh zornih kotov, ki nam združena nudita popolno sliko o delovanju. Prvi zorni kot je osredotočen na:

a) *DSA*,
   ki podaja natančno sliko, kako se izvajajo postopki znotraj vsakega posameznega DSA; drugi pa na
b) *operacijo*,
   ki prikazuje strukturo posameznih operacij, ki se v splošnem izvajajo v več DSA.

Vsak DSA mora izvajati:

1. akcije, ki so potrebne za realizacijo vsake posamezne operacije in
2. dodatne akcije, ki so potrebne za porazdelitev dobljenih rezultatov med druge DSA.

8.1 Obnašanje porazdeljenega imenika

Obnašanje porazdeljenega imenika kot celote je vsota obnašanj posameznih DSA, ki je opisano v razdelku 8.2.

Za porazdeljeno okolje je tipično, da:

- DSA obravnava vsako operacijo kot *prehodni dogodek*: ko dobi (od DUA ali nekega drugega DSA) zahtevo po operaciji, obdela dano zahtevo (kolikor jo pač zmore), nakar jo usmeri k drugemu DSA v nadaljnjo obdelavo in da
- se celotna obdelava neke zahteve izvede v več fazah, ki so skupne vsem operacijam; pri izvajanju vsake od teh faz pa sodeluje več DSA.

Faze izvajanja operacije

Vsaka operacija se v imeniku izvede v treh fazah:

a) *faza razreševanja imena*,
v kateri se na osnovi imena objekta, na katerem naj se izvede operacija, določi, kateri DSA vsebuje vhod, za ta objekt;

b) *faza vrednotenja*,
v kateri se zahtevana operacija (na primer *beri*) tudi dejansko izvede;

c) *faza združitve rezultatov*,
v kateri se povprašujočemu DUA vrnejo rezultati operacije; v primeru veriženja so v to fazo vključeni vsi DSA, ki so posredovali zahtevo drugemu DSA med eno od obeh ali med obema predhodnima fazama.

Vsaka operacija vsebuje argument *napredek operacije*, to je informacijo o fazi, do katere se je operacija, ki jo izvaja več DSA, že izvedla. Vsak DSA potem, ko izvede ustrezni del operacije in preden odda operacijo v nadaljnjo obdelavo drugim DSA, označi, do kje je operacija že izvedena.

Veljavnost zahteve

Po prejemu zahteve za izvedbo neke operacije se DSA najprej prepriča, ali je *zahteva veljavna*, to je, ali se del operacije, ki naj bi jo izvedel, da izvesti. Okoliščine, v katerih se pojavijo zanke v DIT (ob napačni uporabi alternativnih imen, ali ob uporabi napačnih referenc), namreč lahko povzroče, da dobi DSA zahtevo, ki je ne more izvesti.

Stanje operacije

Izvajanje operacije in pogoje, pri katerih pride do zančenja, nadzoruje DSA s *stanjem operacije*, ki je določeno:

- z imenom DSA, ki trenutno izvaja operacijo,
- z imenom objekta, na katerega se nanaša operacija ter
- s parametrom *napredek operacije*.

Poleg trenutnega stanja operacije mora DSA poznati tudi vsa predhodna stanja dane operacije. Predhodna stanja operacije hrani v tako imenovani *beleženi informaciji*, ki jo kot argument operacije preda (drugim DSA) hkrati z operacijo.

Zančenje

Do zančenja (glede na posamezno operacijo) pride vedno, ko je trenutno stanje operacije enako nekemu od predhodnih stanj operacije. Pri obravnavanju zank ločimo dve strategiji:

- *zaznava zanke*:
  DSA preveri dohodno operacijo in javi napako, če najde zanko v njej;
- *izogib zanki*:
  DSA preveri, ali bo operacija, ki naj bi jo poslal naprej (drugemu DSA), povzročila zanko.

Servisna krmila

Omejimo se na obravnavo servisnih krmil, ki so bistvena za pravilno izvajanje operacij v porazdeljenem okolju.

- *prepovedano veriženje*
  DSA upošteva to servisno krmilo pri določanju načina posredovanja operacij drugim DSA. Če je postavljeno, uporabi namigovanje, sicer se odloči med veriženjem in namigovanjem (na osnovi lastne opremljenosti).
- *časovna omejitev*
  DSA skrbi, da ne preseže navedene vrednosti. Ko dobi od DUA zahtevo za operacijo, dobi hkrati tudi časovno omejitev, za izvedbo celotne operacije. V primeru nadaljnjega veriženja operacije preda naslednjemu DSA tudi časovno omejitev, ki je enaka času, ki je še ostal glede na prvotno (v DUA) specificirano omejitev.
- *prostorska omejitev*
  DSA skrbi, da seznam rezultatov ne preseže navedene velikosti. Ob veriženju zahteve drugim DSA, jim posreduje nespremenjeno začetno vrednost omejitve, ki ostane nespremenjena tudi v primeru, ko je potrebno zahtevo razstaviti v več podzahtev: vsaka podzahteva vsebuje celotno začetno vrednost omejitve. Ko DSA dobi rezultate, jih pregleda, upošteva omejitev in vrne skupni odgovor, ki ne presega te omejitve. V odgovoru označi tudi, če je bila v posameznih rezultatih omejitev presežena.
- *prioriteta*
  Pri vseh načinih širjenja operacij DSA skrbi, da se operacije izvajajo v vrstnem redu, ki je naveden v tem servisnem krmilu.
- *lokalna izvedba*
  Operacija je omejena na izvajanje v danem DSA; ni je mogoče širiti na druge DSA.

Dovršitev operacij

Vsak DSA, ki začne ali, ki povzroči širjenje kakšne operacije, mora hraniti podatek o obstoju *nedovršene operacije* vse dokler, od drugih DSA ne dobi rezultatov ali sporočil o napaki; oziroma dokler se ne izteče maksimalni dovoljeni čas za izvedbo operacije. Ta zahteva se nanaša na vse operacije, na vse načine širjenja operacij in na vse faze operacij.

8.2 Obnašanje posameznega DSA

Notranje obnašanje DSA prikazuje slika 7, ki kaže, v kakšnem odnosu je glavni nadzorni postopek (razdelilnik operacij) z ostalimi postopki (razrešitev imena, iskanje konteksta imenovanja, lokalna razrešitev imena, ovrednotenje, ovrednotenje enega objekta, ovrednotenje več objektov, združitev rezultatov).

V nadaljevanju na kratko opišimo vsakega od postopkov, ki skupno tvorijo proces, ki predstavlja obnašanje DSA.

Razdelilnik operacij

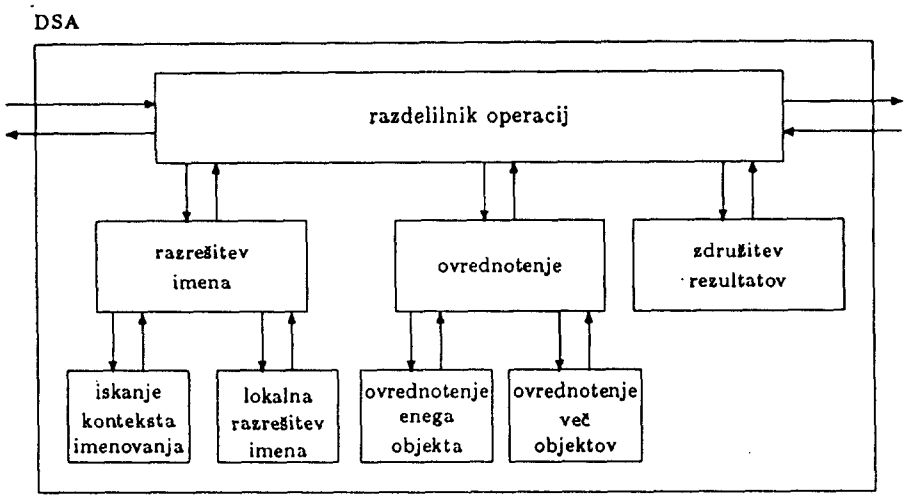
Po prejemu zahteve za izvedbo neke operacije razdelilnik operacij najprej preveri veljavnost zahteve. Če ne najde niti zanke niti overovitvene napake, sproži postopek *razrešitev imena*, ki vrne enega od naslednjih odgovorov:

- sporočilo *ime najdeno*
  v tem primeru razdelilnik operacij pokliče postopek *ovrednotenje*, nakar čaka na rezultate;
- *referenco*
  v tem primeru razdelilnik operacij (odvisno od načina širjenja operacij) bodisi:
  - izvede veriženje ali dodeljevanja večim, s katerim preda operacijo drugemu(-im) DSA, nakar čaka na rezultate; bodisi
  - vrne povprašujočemu (DUA ali DSA) namig, ki je vsebovan v referenci;
- sporočilo *napaka*
  v tem primeru razdelilnik operacij vrne povprašujočemu (DUA ali DSA) enako sporočilo o napaki.

Ko razdelilnik operacij dobi vse rezultate: notranje (od postopka za ovrednotenje) in zunanje (od drugih DSA), jih s postopkom *združitev rezultatov* uredi in vrne povprašujočemu (DUA ali DSA).

Razrešitev imena

Postopek kliče najprej podpostopek *iskanje konteksta imenovanja* in v primeru, da najde ustrezni lokalni kontekst imenovanja, pokliče še podpostopek *lokalna razrešitev imena*. Dobljene rezultate (sporočilo *ime najdeno*, referenco ali sporočilo o napaki) vrne razdelilniku operacij, razen v primeru, ko postopek za lokalno razrešitev imena sporoči, da je izvedel prevedbo alternativnega imena; tedaj postopek za razrešitev imena ponovno izvede celotno analizo (aktivira postopek za iskanje konteksta imenovanja, ...).



Slika 7: Notranje obnašanje DSA

Iskanje konteksta imenovanja

Postopek primerja *predlagano ime* (to je ime objekta, na katerem naj bi se izvedla operacija) s kontekstnimi predponami – išče kontekstno predpono, ki se vsaj delno ujema s predlaganim imenom, če jih je več izbere tisto, ki se najbolj ujema. Če ne najde nobenega ujemanja, primerja predlagano ime s kontekstnimi predponami v navzkrižnih referencah (seveda, če le te obstajajo). Če tudi to ne uspe, poišče referenco na predhodnika. V odgovoru vrne bodisi lokalni kontekst imenovanja, bodisi navzkrižno referenco, bodisi referenco na predhodnika; v vsakem primeru pa nastavi parameter *napredek operacije* tako, da kaže, da se izvaja faza razreševanja imena.

Lokalna razrešitev imena

Postopek primerja dele prednostnega imena (DPI) predlaganega imena z lokalnimi vhodi in vrne sporočilo *ime najdeno*, če se vsi DPI predlaganega imena ujemajo. Če kateri od DPI ne ustreza nobenemu od lokalnih vhodov, potem poskuša najti ustrezno specifično referenco oziroma nespecifično referenco na naslednike, ki jo nato posreduje postopku za razrešitev imena. Če pri primerjanju naleti na alternativno ime, vrne postopku za razrešitev imena sporočilo *prevedba alternativnega imena*; sporočilo *ime najdeno* pa le, če so v trenutku odkritja alternativnega imena, že vsi DPI uspešno obdelani (primerjani).

Ovrednotenje

Postopek izvede zahtevano operacijo na objektu. Odvisno od vrste operacije se deli na dva podpostopka:

- **ovrednotenje enega objekta**:
  deluje nad eno-objektnimi operacijami: *beri, primerjaj, dodaj_vhod, odstrani_vhod, spremeni_vhod* in *spremeni_DPI*; postopek dejansko najde, primerja ali spreminja atribute.
- **ovrednotenje več objektov**:
  deluje nad operacijama *išči* in *naštej*; pri tem uporablja filtre in (če je potrebno) razbije zahtevo na podzahteve, ki jih posreduje razdelilniku operacij, ki jih odda ustreznim DSA.

Združitev rezultatov

Razdelilnik operacij aktivira postopek za združitev rezultatov, kadar obstajajo zunanji rezultati. Seveda lahko hkrati obstaja tudi notranji rezultat. DSA hrani vse rezultate (in vse napake) dokler se postopek za združitev rezultatov v celoti ne zaključi.

Če so zunanji rezultati posledica:

- *veriženja*,
  je postopek za združitev rezultatov trivialen.
- *dodeljevanja večim*,
  se lahko zgodi, da rezultati bodisi ne obstajajo, bodisi obstaja en rezultat, bodisi obstaja več identičnih rezultatov; obstajajo pa lahko tudi napake. Če kakšen od rezultatov obstaja, postopek vrne enega izmed njih, sicer javi napako.
- *razstavitve zahteve*,
  postopek združi rezultate tako, da oblikuje unijo posameznih rezultatov. V primeru, ko so prisotni tako rezultati kot tudi napake, lahko bodisi javi napako, bodisi vrne nepopolni rezultat.

8.3 Izvajanje operacij

Vse *eno-objektne operacije* DSA izvede na enak način z naslednjim zaporedjem dogodkov:

1. aktivira razdelilnik operacij;
2. izvede postopek za razrešitev imena (da najde lokacijo objekta, na katerega se operacija nanaša);
3. izvede postopek za ovrednotenje enega objekta; pri tem
4. upošteva servisna krmila, ki določajo omejitve specificirane s strani uporabnika;
5. vrne rezultate tistemu DUA ali DSA, ki mu je posredoval zahtevo.

Obe *več-objektni operaciji* izvede DSA z naslednjim zaporedjem dogodkov:

1. aktivira razdelilnik operacij;
2. izvede postopek za razrešitev imena;
3. ko najde lokacijo objekta, na katerega se operacija nanaša, izvede postopek za ovrednotenje več objektov;
4. če le-ta izvede razstavitev zahteve na podzahteve, razdelilnik operacij začasno "shrani" lokalne rezultate in počaka, da dobi verižene odgovore, nakar aktivira postopek za združitev rezultatov;
5. med obravnavo operacije preveri in upošteva servisna krmila, da ostane v dovoljenih mejah;
6. vrne rezultate ali sporočila o napakah tistemu DUA ali DSA, ki mu je posredoval zahtevo.

9 Zaključek

V ([COS88]) uvrščajo pri klasifikaciji storitev na *začetne in bodoče* imenik med bodoče storitve; to pa ne zato, ker ni nujno potreben, ampak zato, ker standardi zanj še niso dovolj dodelani. Trenutni standard:

- ne določa, kateri atributi morajo biti uporabljeni pri oblikovanju DPI, predlaga le, kakšna naj bo oblika imen; ne standardizira pa sheme (pravil o strukturi DIT);
- ne omogoča izvedbe operacij *dodaj_vhod* in *odstani_vhod* v porazdeljenem okolju: operaciji sta omejeni na primer, ko sta hkrati vhod, ki naj bi bil dodan oziroma odstranjen in neposredni predhodnik tega vhoda oba v istem DSA;
- ne podpira kopiranja (oziroma natančneje povedano: upravljanja kopij);
- prepušča nadzor dostopa vsakemu DSA kot predmet lokalne izvedbe;
- ne vsebuje globalne zasnove za pridobivanje in upravljanje znanja; operacije, ki vplivajo na konsistentnost znanja izven meja DSA, so dovoljene le na osnovi medsebojnega sporazuma med upravljalci ustreznih DSA.

Kljub vsemu pa so vsaj posamezni deli standarda dovolj stabilni, da je na njihovi osnovi upravičeno začeti z izvedbo pilotskih projektov za imenik (tak je na primer projekt THORN). Pilotski projekti naj bi služili kot demonstacijski projekti za potrditev in izboljšanje starih ter za razvijanje in kasnejšo standardizacijo novih konceptov.

10 Viri

[Ami88] *The Amigo MHS+ Report*, Cost 11ter, Editors: Nottingham University, March 1988

[Ben87] Benford S., *Mapping the DIB to a Relational Schema: Name Representation and Verification*, Internal report, Department of Computer Science, University of Nottingham, 1987

[Ben88] Benford S., *Navigation and Knowledge Management within a Distributed Directory System*, Internal report, Department of Computer Science, University of Nottingham, 1988

[Bon88] Bonač M., *Grupna komunikacija z elektronsko pošto*, Magistrska naloga, Univerza Edvarda Kardelja, Fakulteta za elektrotehniko, Ljubljana 1988

[But86] Butscher B., Tschichholz M., *Directory System, Requirements and the CCITT-Model*, HMI, Berlin, 1986

[Cho86] Chong K. N., Chew E. K., McDonell K. J., *Implementation Considerations of a Name Validation Function for Distributed Directory Services*, Proc. of the IFIP TC 6 International Symposium on Computer Message Systems – 85, North-Holland, 1986

[Com86] Comer D. E., Peterson L. L., *A Model of Name Resolution in Distributed Systems*, IEEE Proc. 6th ICDCS, May 1986

[COS88] COSINE Specification Phase, *Study on Directories*, RARE WG3 Report 7.1, Biber A., Forster J., Softlab, Gmbh, Munich, July 1988

[ECM85] ECMA TC 23, *OSI Directory Access Service and Protocol*, ECMA TR/32

[Hui88] Huitema C., *A Proposal for a naming, data structures, and name date distribution in RARE*, RARE WG3, Internal Report, May 1988

[ISO87] ISO/CCITT, *Directory Convergence Document*, Part 1–8, Gloucester, November 1987

[Kil88] Kille S., *The THORN Large Scale Pilot Exercise*, RARE WG3 Report, May 1988

[Lam86] Lampson B. W., *Designing a Global Name Service*, Proc. 5th ACM Symposium on Principles of Distributed Computing, 1986

[Lan86] Lantz K. A., Eighoffer J. L., Hitson B. L., *Towards a Universal Directory Service*, ACM Operating Systems Review, Vol. 20, No. 2, April 1986

[Lat86] Latella D., *A Model for a Distributed Directory System: Specification of a Distributed Algorithm for Name Resolution*, CNUCE, Internal Report C86-08, July 1986

[Lat88] Latella D., Lenzini L., *The OSIRIDE Directory System: Status and Perspectives*, Computer Standards & Interfaces, Vol.7, No.1/2, 1988

[OSN87] *Directory Standardisation Rethink Results in a more Limited Service*, The Open Systems Newsletter, Vol.1, Issue 2, April 1987

[Par86] Pareta J. S., Huitema C. *THORN – Directory Data Structure and Name Assessment Algorithms*, INRIA, October 1986

[Sir87] Sirovich F., *The ESPRIT Directory System*, ESPRIT 87 – Achievements and Impact, CEC, ed. #2, North Holland, September 1987

[Ter86] Terry D. B., *Stucture-free Name Management for Evolving Distributed Environment*, IEEE Proc. 6th ICDCS, May 1986

[Tvr88] Tvrdy H., *Mehanizmi kopiranja informacij v porazdeljenem elektronskem imeniku*, IJS-DP 5116, Ljubljana 1988

[VER86] *VERDI – A Distributed Directory System for German Research Network (DFN)*, Report, Editors: Santo H., Tschichholz M., July 1986

[VER88] *VERDI II, Konzeption für den Einsatz des Standardisierten Directory Systems im Deutschen Forschungsnetz*, Bonacker K., Schüth H., Thoma G., Tschichholz M., Wenzel O., Meyer A., Tadje R., Zwischenbericht, Juni 1988

[Zat88] Zatti S., Janson P. *Interconnecting OSI and Non-OSI Networks using an Integrated Directory Service*, Computer Networks and ISDN Systems, Vol.15, No.4, 1988

AVTORSKO STVARNO KAZALO ČASOPISA INFORMATICA, LETNIK 12 (1988)

Članki

Sifrar, Marija, Data Resource Sharing in Yugoslavia. Informatica 12 (1988), No. 3, pp. 47-48.

Vouk, V., J. Ferbežar in A. Brodnik, Večopravilno okolje za delo v realnem času na računalniku IBM-PC. Informatica 12 (1988), No. 2, pp. 94-101.

Vuga, L., Matematično sodilo o možnosti strojne inteligence (ali o Churchevem teoremu). Informatica 12 (1988), No. 2, pp. 114-116.

Welzer, Tatjana, I. Rozman in J. Gyorkos, The Normalization of relational Database. Informatica 12 (1988), No. 3, pp. 12-15.

Zimic, N., J. Virant, M. Bradeško in I. Pepelnjak, Možnosti zaščite programske opreme na mikroračunalnikih. Informatica 12 (1988), No. 1, pp. 44-47.

Zorman, A., M. Gerkeš, V. Zumer and Krista Rizman, A Simulation Approach before Using the Industrial Microcomputer Controller. Informatica 12 (1988), No. 3, pp. 16-25.

Zeleznikar, A.P., Uvodno predavanje. Informatica 12 (1988), No. 1, pp. 3-5.

Zeleznikar, A.P., Lastnosti informacije: mala enciklopedija (A). Informatica 12 (1988), No. 1, pp. 65-76.

Zeleznikar, A.P., O razvojni pobudi računalniške industrije. Informatica 12 (1988), No. 2, pp. 3-11.

Zeleznikar, A.P., Problems of the Rational Understanding of Information. Informatica 12 (1988), No. 2, pp. 31-46.

Zeleznikar, A.P., Informational Logic I. Informatica 12 (1988), No. 3, pp. 26-38.

Zeleznikar, A.P., Informational Logic II. Informatica 12 (1988), No. 4, pp. 3-20.

Zeleznikar, A.P., Informacija in informacijski sistem. Informatica 12 (1988), No. 4, pp. 63-68.

Report of a Journey

Zeleznikar, A.P., Parsys Expedition to New Worlds II. Informatica 12 (1988), No. 1, pp. 77-91.

FORUM INFORMATIONIS

Pisanski, T., V. Batagelj, M. Petkovšek, M. Spegel, B. Vilfan in I. Tvrdy, Odprto pismo uredniškemu odboru časopisa Informatica. Informatica 12 (1988), No. 3, pp. 78-79.

Piskar, R., Odgovor na odprto pismo. Informatica 12 (1988), No. 4, p. 72.

Prešern, S., Odprto pismo - napoved odprtega strokovnega dialoga? Informatica 12 (1988), No. 4, pp. 70-72.

R., I., O recenzijah. Informatica 12 (1988)., No. 4, pp. 78-79.

Zeleznikar, A.P., Kratko poročilo s prvega zasedanja Forum Informationis. Informatica 12 (1988), No. 3, pp. 78.

Zeleznikar, A.P., Se o soncih v praznem prostoru. Informatica 12 (1988), No. 3, pp. 81-83.

Zeleznikar, A.P., Kratko poročilo z drugega zasedanja Forum Informationis. Informatica 12 (1988), No. 4, pp. 69-70.

Zeleznikar, A.P., Kratek komentar k drugemu zasedanju Forum Informationis. Informatica 12 (1988), No. 4, p. 70.

Zeleznikar, A.P., A Joined Discourse of Science Fiction in Computer Science (I). Informatica 12 (1988), No. 4, pp. 72-78.

Zeleznikar, A.P., Se o recenzijah. Informatica 12 (1988), No. 4, pp. 79-80.

MEDNARODNA POLETNA ŠOLA
"CONSTRUCTIVE METHODS IN COMPUTING SCIENCE",
MARKTOBERDORF, ZVEZNA REPUBLIKA NEMČIJA
24.7. - 5.8.1988

Ta dvotedenska šola iz računalništva je del programa ASI (Advanced Study Institutes) komiteja za znanost NATO, ki jo tudi v največji meri financira. Njen organizator je Tehniška univerza iz Münchna. Pripravi jo vsaki dve leti, vedno v Marktoberdorfu. Letošnja je bila že deveta, tako lahko rečemo, da je tradicionalna. Je zelo cenjena, saj na njej predavajo priznani znanstveniki s področja teoretičnega računalništva, predavanja pa so vedno posvečena najaktualnejšim problemom, raziskavam in dosežkom. V splošnem so na postdoktorski ravni.

Letos so bile tema predavanj konstruktivne metode v računalništvu. Minevajo namreč časi snovanja kar tako, pa naj gre za programsko ali materialno opremo, v celotnem postopku se uveljavljajo matematične metode.

E.W. Dijkstra, ki mu pripada tudi čast uvodnega in zaključnega govornika šole, je prikazal primere lepega dokazovanja. Dokazi nas v računalništvu zanimajo, ker jih lahko vzporejamo z algoritmi. Dokaz sestoji iz dela, v katerem postavimo približno idejo dokaza (v računalništvu ideja algoritma) in iz dela, v katerem na osnovi opisa problema in ideje dokaz (algoritem) izpeljemo bolj ali manj mehanično.

Velika pozornost je bila namenjena konkurentnim in distribuiranim sistemom. M. Broy je predstavil funkcionalen pristop k načrtovanju distribuiranih sistemov, ki sledi klasičnemu vzorcu razvoja sekvenčnih sistemov in vodi od specifikacije zahtev do distribuiranega sistema, sestavljenega iz programov, ki med seboj komunicirajo in sodelujejo. Podal je formalen okvir za vse faze načrtovanja. Za vsako fazo je uporabil poseben formalizem, za prehod med njimi pa je oblikoval pravila. Komponente sistema je predstavil z množicami funkcij.

B.W. Lampson je na različnih organizacijah pomnilniškega prostora prikazal metodo specificiranja konkurentnih in distribuiranih sistemov. Za nekatere specifikacije pa je podal več različnih implementacij, ker se je v predavanjih ukvarjal tudi z dokazovanjem, da implementacija zadošča specifikaciji.

J. Misra je predstavil programski zapis in dokazovalno teorijo, skupaj imenovana UNITY, ki jo je razvil skupaj s K.M. Chandyjem. Teorija je uporabna za obravnavo različnih arhitektur (sistolična polja, skupni pomnilnik, prenašanje sporočil...) in aplikacij (operacijski sistemi, kombinatorični problemi, komunikacijski protokoli...). Zanimivo pa je, da opušča skoraj vse tradicionalne pojme iz paralelnega programiranja, tudi osnovni pojem - proces. V UNITY ne združujemo procesov, temveč programe. Za dokazovanje so uvedeni logični operatorji, ki spominjajo na operatorje časovne logike.

J.L.A. van de Snepscheut pa se je v predavanjih ukvarjal z oblikovanjem konkurentnih programov, sestavljenih iz sekvenčnih procesov, ki med seboj komunicirajo preko skupnih spremenljivk. Metodo za sekvenčne procese je razširil še na sistolične algoritme. Za sinhronizacijo med procesi je uporabil semaforje. Podal je definicijo semaforjev, dokazal njihove lastnosti in prikazal uporabo binarnega razcepljenega semaforja.

Namen predavanj C.A.R. Hoarea je bil, pokazati, zakaj nas bi v teoretičnem računalništvu morala zanimati teorija kategorij. Ta teorija nudi podobne osnove za logiko, teorijo množic, teorijo modelov, dokazovalno teorijo in za osnove matematike. Obeta pa veliko pomoč pri raziskovanju odnosov med specifikacijami, izvedbami, dokazi pravilnosti, implementacijami, programi in jeziki, v katerih so vsi ti napisani.

Pri programiranju velikokrat opazimo, da je struktura programa pogojena z uporabljenimi podatkovnimi strukturami ali tipi. To lastnost izkorišča Martin-Löfova teorija tipov. R. Backhouse je pokazal, kako nam pomaga konstruktivno sklepanje o podatkovnih tipih pri izpeljevanju algoritmov in dokazovanju pravilnosti. V predavanjih R.L. Constablea smo spoznali pojem očitnosti. Običajno govorimo v predikatni logiki o logični pravilnosti izjav. Izjava pa je pravilna natanko tedaj, kadar obstaja množica elementov, ki pričajo o njeni očitnosti. Predavatelj je pokazal, da so dokazi pravzaprav zapisi očitnosti izjav.

R.S. Bird je predstavil svoj način zapisa za funkcionalno programiranje in račun za izpeljavo funkcionalnih programov iz njihovih specifikacij. Njegovo uporabo je prikazal na različnih podatkovnih strukturah (seznami, drevesa, polja).

J.S. Moore nas je najprej seznanil z logiko in sposobnostmi t.i. Boyer-Mooreovega dokazovalnika izrekov. Podrobneje pa je pokazal razširitev uporabljene logike z omejenimi kvantifikatorji (npr. vsota po določenih vrednostih indeksa) in delnimi funkcijami. Tako razširjeni dokazovalnik lahko med drugim dokaže binomski izrek in izreke o funkcijah, ki zadoščajo nenavadnim rekurzivnim enačbam.

Vsa predavanja bodo izšla v knjigi zbirke NATO ASI.

Tatjana Kapus
Tehniška fakulteta Maribor
Smetanova 17
62000 Maribor