Volume 15 Number 3 August 1991
YU ISSN 0350 – 5596

Informatica

A Journal of Computing and Informatics

The Slovene Society INFORMATIKA
Ljubljana

Informatica

A Journal of Computing and Informatics

Subscription Information

Informatica (YU ISSN 0350 – 5596) is published four times a year in Winter, Spring, Summer and Autumn (4 issues).

The subscription price for 1991 (Volume 15) is US$ 30 for companies and US$ 15 for individuals.

Claims for missing issues will be honoured free of charge within six months after the publication date of the issue.

Printed by Tiskarna Tipa, Gosposvetska 13, Ljubljana.

Informacija za naročnike

Informatica (YU ISSN 0350 – 5596) izide štirikrat na leto, in sicer v začetku januarja, aprila, julija in oktobra.

Letna naročnina v letu 1991 (letnik 15) se oblikuje z upoštevanjem tečaja domače valute in znaša okvirno za podjetja DEM 30, za zasebnike DEM 15, za študente DEM 8, za posamezno številko pa DM 10.

Številka žiro računa: 50101 – 678 – 51841.

Zahteva za izgubljeno številko časopisa se upošteva v roku šestih mesecev od izida in je brezplačna.

Tisk: Tiskarna Tipa, Gosposvetska 13, Ljubljana.

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

Pri financiranju časopisa **Informatica** *sodeluje Republiški komite za raziskovalno dejavnost in tehnologijo, Tržaška 42, 61000 Ljubljana*

Volume 15 Number 3 August 1991
YU ISSN 0350 – 5596

Informatica

A Journal of Computing and Informatics

The Slovene Society INFORMATIKA
Ljubljana

Letnik 15 Številka 3 Avgust 1991 YU ISSN 0350 – 5596

Informatica

Časopis za računalništvo in informatiko

V S E B I N A

CONSCIOUSNESS AS A PREREQUISITE FOR KNOWLEDGE

INFORMATICA 3/91

Keywords: intension, consciousness, knowledge representation, logic, reasoning

Oliver B. Popov
Institute of Informatics
College of Math and Natural Sciences
University of Skopje
Skopje

ABSTRACT: A certain type of granulation has been proposed in the structure of knowledge through the introduction of a level of consciousness. A system for the Logic of Consciousness is developed which is finitely axiomatizable, determined and decidable.

1. INTRODUCTION

The research in Knowledge Representation, as one of the ongoing issues in Artificial Intelligence, is mainly focussed on two principal questions (1) the identification of the primitive knowledge structures and their appropriateness as well as the level of involvement in the actuall representation, and (2) the existence and selection of the adequate methods to reason about knowledge.

In (POP90) it is positted that concepts are the primary building blocks of knowledge. Moreover, the propositions are made out of concepts. On the other hand, the idea of complex, as a primitive for analysis of epistemic and doxastic inclinations, is defined as a set of propositions which are cognitively accessible to a reasoning agent. The interplay of the notions of concept, complex and proposition gives the necessary granulation in the structure of knowledge.

The argument put forward is that before a reasoning agent knows a concept it must be conscious of the very same concept. Thus, consciousness becomes the first step in the process of knowing, i.e. acquiring knowledge. To realize this level of reasoning one must build a certain structure, named conceptual structure in which the only mode of reasoning is consciousness.

Open sentences and propositional functions are which are principal

devices for expressing propositions are not suitable for building the conceptual structure since they can not be attributed truth-values. Hence, the existance of minimal concept expressing proposition is stipulated.

For every possibly non-empty concept Δ there is a 'minimal proposition of concept expression' denoted by Π_o which has the form "The concept of . . . exists " or "There is a concept of ...". The gap is to be substituted with an arbitrary concept. The predication of existance in the minimal proposition of concept expression is to be understood to vary over the set all possible entities, where the set of actual entities is a subset of the former one. The definition of the minimal concept expressing propositions provides the primitives for establishing conceptual structures.

2. CONCEPTUAL STRUCTURES

Naturally, one proceeds with a definiton of a language. By language of consciousness L_c is understood a language of which the symbols are drawn from the following categories:

[1] Improper symbols of the system PC (propositional calculus);

[2] A denumerable set of minimal concept expressing propositions $\Pi_c = \{p, q, r, \ldots\}$;

[3] A monadic operator of consciousness 'CON' (read 'conscious of');

[4] Individual variables ranging over a denumerable set of all reasoning agents $PA = \{a_1, \ldots, a_i, \ldots\}$

The formulas of L_c are denoted by $p, \ldots, f, g, h, \ldots$ Let $\Gamma(L_c)$ be the set of all formulas of L_c which is the smallest set that contains Π_c, is closed under the logical connectives and contains $_a\mathrm{CON}(f)$ whenever $f \in \Gamma(L_c)$ and $a \in PA$. The term '$_a\mathrm{CON}(p)$' is a formal counterpart of "an agent 'a' is conscious of p" where the epistemic notion is considered as an empirical relation between an agent and a proposition.

A conceptual structure for L_c is a tuple $M = (K, \Phi, \Theta, V)$ where:

[1] K is a nonempty set of complexes

[2] Let $\rho(K)$ be the power set of K, i. e., the set of all sets of possible complexes. Then Φ is a function from the set Π_c to the set $\rho(K)$, that is $\Phi : \Pi_c \dashrightarrow \rho(K)$ The function Φ provides each minimal concept expressing proposition with an intension;

[3] Θ is a function from {a} x K to the set $\rho(\rho(K))$, viz., the function Θ assigns to an agent 'a' who is in the complex k the set of all minimal concept expressing propositions that an agent is conscious of'' in the complex k. In a multi-agent enviroment the function Θ is defined as:

Θ : PA x K --> $\rho(\rho(K))$

instead of

Θ : {a} x K --> $\rho(\rho(K))$

[4] V is a function of confirmation

$V : \Gamma(L_c) \times K \dashrightarrow 2$

such that

(a) $V(p,k) = 1$ iff $p \in \Pi_c$ and $k \in \Phi(p)$,

(b) For any wff f and any complex k, $V(\neg f,k) = 1$ iff $V(f,k) = 0$,

(c) For any two wffs f and g and any complex k, $V(f \& g) = 1$ iff $V(f,k) = 1$ and $V(g,k) = 1$,

(d) For any wff f and any complex k, $V(_a CON(f),k) = 1$ iff the $INT(f) \in \Theta(a,k)$.

Intuitively, the INT(f) of any wff $f \in \Gamma(L_c)$ determines a set of complexes which is a subset of K, where the wff f is 'inevitable'. The intension of f is interpreted as a set of complexes that confirm f, hence $INT(f) = \{k : M,k \models f\}$. Given a conceptual structure M and a complex $k \in K$, the expressions $V(f,k) = 1$ under the interpretation M and $M,k \models f$ are identical.

For each complex i, the value $\Theta(a,i)$ determines a unique set of conscousness that is assigned to a reasoning agent. By imposing certain restriction to the sets of conscoiusness one can capture different situations in the process of reasoning. For example, if $\Lambda \in \Theta(a,i)$ where Λ denotes the empty set, then an agent 'a' is conscous of the impossible; if $\Theta(a,i) \equiv \Lambda$ then an agent 'a' residing in a complex i is in a state of complete ignorance. The conceptual structure has all the necessary cri[illegible]a for conceptual satisfactic

It is interesting to examine the modal relations among concepts in terms of intensions as defined in the conceptual structures. Assume that K is a nonempty set of complexes and that each complex has at least one entity (where the term 'entity' represents things, items, objects or individuals). Given the concepts Δ_p, Δ_q, Δ_r denote with p, q, i r their concept expressing propositions respectively. Then. some of the plausible modal relations are:

- two concepts Δ_p and Δ_q are agreeable iff $INT(p) \cap INT(q) \neq \Lambda$.
- two concepts Δ_p and Δ_q are are

disagreeable iff $INT(p) \cap INT(q) \neq \Lambda$.

• two concepts Δ_p and Δ_q are universally agreeable iff $INT(p) \subseteq INT(q)$.

• a concept Δ_r is relevant iff $INT(r) \neq \Lambda$.

• a concept Δ_r is universally relevant iff $INT(r) \equiv K$.

The intension of proposition becomes the intersection of intensions of all the concept expressing propositions which constitute the proposition.

The difference between a Kripke structure KS (KRI63) and a conceptual structure M is the lack of the relation of accessibility R in the later. As a consequence the reasoning agent is deprived form his 'unterworld' intiution since the relation of accessibility provides it with a set of epistemic alternatives. However, an agent still has its 'intensional' intuition and thus a modal type of omniscience has been replaced with an intensional omniscience. For any two wffs f and g if $INT(f) \equiv INT(g)$ and CON(F) is true then $_aCON(g)$ is also true. In the intensional paradigm a reasoning agent has lost the ability to distnguish between logically equivalent formulas. It appears that the treatment of the omniscinece phenomenon with a single primitive qualifier such as a complex is not possible in a satisfactory manner (POP91).

3. LOGIC OF CONSCIOUSNESS

The fundamental system S_c for reasoning about consciousness is finitely axiomatizable. The system S_c is defined by the following axiom schemas and rules of inference:

(PC) All propositional tautologies

(RE) From $f \Leftrightarrow g$ infer $_aCON(f) \Leftrightarrow {}_aCON(g)$

The axioms (PC) are credit from propositional calculus. The rule of inference is due to the intensional approach and is actually the rule of substitutions of equivalents.

Validity in the semantics of possible complexes is a three-fold phenomenon. First, there is a local validity of wff f (a case when formula f is valid ina complex k). A wff f is structurally valid iff is valid in all complexes belonging to a structure or a model. Finally, a wff f has a global validity if it is valid in a class of conceptual structures or models $\Omega(M)$.

After establishing the finite axiomatization of the system S_c the

next step is to prove that S_c for LC (or the Logic of Conscousness) is determined, i. e., is sound and complete with respect to the class of conceptual structures $\Omega(M)$.

□ Proposition: The system L_c is determined with respect to a class of conceptual structures $\Omega(M)$.

The proof of the proposition being lenghtly and tedious, is given in the Appendix A.

Finally, the computational properties of the system S_c are being addressed. The system S_c in LC is decidable if there is an effective procedure for deciding whether or not a wff f in LC is a theorem in S_c. The the proof method is introduced in (CHE80) with respect to the classical modal systems. The proof of decidability requires the notions of axiomatizability and conceptual structure finiteness.

The system S_c in LC is axiomatizable and all its axioms are decidable. This follows from the properties of PC. The rule (RE) is a reasonable one since it always determines the relation between premises and a conclusion. The existance of a positive test for theoremhood qualifies the system as semi-decidable; it is based on its axiomatizability. When the system S_c has the property of having a conceptual structure which is finite along with the complete axiomatization then it is fully decidable. The finiteness of the structure provides the negative test for provability.

A conceptual structure is finite iff K has a finite number of complexes; otherwise the structure is infinite. Let $\Omega(M)$ be a class of conceptual structures for S_c. Imposing a standard enumeration on the elements of $\Omega(M)$ yields $M_1,\ldots,M_i,\ldots$.The class is enumerable, since each M_i is finite. The test for negative theoremhood is reduced to finding an M_i which is a conceptual structure and showing that some wff f is falsified in M_i.The fact that axiomatization and models are finite implies the finiteness of both the tasks(if the number of complexes is n, where n is finite number, then the corresponding conceptual structure has at most 2^n complexes). The tractability of the problem should be understood in principle and not in practise for it is co-NP-complete.

4. CONCLUSION

The intention of the article is to explore the possiblities for some

granulation in KR and the process of reasonin about it. The level of consciousness is considered as a prerequisite for the level of knowing and its primary objects of manipulations are concepts. A system for the Logic of Conscousness is developed which enables propositional reasoning , it is based on the intensional logic, is finitely axiomatizable, determined and decidable.

APPENDIX A

To prove that the system S_c is determined in LC, i. e., sound and complete a few definitions and two lemmas are neeeded which have become standard (KAP64,KRI68) in all proofs pertaining to determinism.

A wwf f is S_c inconsistent if $\vdash \neg f$ is a theorem in S_c. A wff f is S_c consistent if $\dashv \neg f$ in S_c. The idea of S_c consistency can be extended to a set of formulas. Let Ψ denote a set of wffs. If Ψ is finite, namely $\Psi = \{f_1, \ldots, f_k\}$ then $C(\Psi)$ stands for a conjuction of the formulas of Ψ. The set of formulas Ψ is S_c inconsistent iff if $\dashv \neg C(\Psi)$ in S_c. When Ψ is an infinite set, the consistency is defined as: the set Ψ is S_c consistent iff every finite subset of Ψ is S_c consistent. A set of formulas Ψ is called maximal iff for every formula $f \in \Psi$, either $f \in \Psi$ or $\neg f \in \Psi$. The set of formulas Ψ is called maximal S_c consistent, denoted by MAX(Ψ), iff Ψ is both maximal and S_c consistent. In other words, a set of formulas is maximal S_c consistent if and only if any other conceivable extension will make it inconsistent.

Given a formula $f \in \Gamma(L_c)$, relative to the system S_c, a proof set of $|f|$ is the set of all MAX(Ψ) sets of formulas containing f (CHE80).

The next two lemmas are also needed for the proof.

□ Lemma 1: Suppose that Ψ is MAX(Ψ) set of formulas with respect to the system S_c. Then

(a) $f \in \Psi$ iff $\neg f \notin \Psi$

(b) $f \,\&\, g \in \Psi$ iff $f \in \Psi \,\&\, g \in \Psi$

(c) $f \Leftrightarrow g \in \Psi$ iff $f \in \Psi$ iff $g \in \Psi$

(d) every theorem of S_c is in Ψ

□ Lemma 2: Every S_c consistent set of formulas Ψ has a maximal S_c consistent extension.

□ Proposition : The systenm S_c is determined with respect to a class of conceptual structures $\Omega(M)$.

Proof:

(i) soundness

The soundness of the system follows

from the fact it contains all the axioms of (PC) which are tautologies. Also, the rule of inference (RE) preserves validity. For consider a class of conceptual structures $\Omega(M)$ such that $f \Leftrightarrow g$ is valid, namely, $\models f \Leftrightarrow g$. The last assertion holds iff $INT(f) \equiv INT(g)$. Then for any complex k and any set of conscousness $\Theta(a,k)$ in an arbitrary conceptual structure M, $INT(f) \in \Theta(a,k)$ iff $INT(g) \in \Theta(a,k)$. Thus for all complexes k, $V({}_aCON(f),k) = 1$ iff $V({}_aCON(g),k) = 1$ which implies that $\models {}_aCON(f) \Leftrightarrow {}_aCON(g)$ for all $M \in \Omega(M)$.

(ii) completeness

Consider the following tuple $M = (K, \Phi, \Theta, V)$ where:

[1] K is a nonempty set of complexes and each complex is maximal S_c consistent set of formulas;

[2] $\Phi : \Pi_c \dashrightarrow \rho(K)$

[3] $\Theta : \{a\} \times K \dashrightarrow \rho(\rho(K))$

[4] V is a function of confirmation

$V : \Gamma(L_c) \times K \dashrightarrow 2$

such that

(a) $V(p,k) = 1$ iff $p \in \Pi_c$ and $k \in \Phi(p)$,

(b) For any wff f and any complex k, $V(\neg f,k) = 1$ iff $V(f,k) = 0$,

(c) For any two wffs f and g and any complex k, $V(f \& g) = 1$ iff $V(f,k) = 1$ and $V(g,k) = 1$,

(d) For any wff f and any complex k, $V({}_aCON(f),k) = 1$ iff the $INT(f) \in \Theta(a,k)$.

Since LC is based on the intensional logic, the following assumption is correct: for any complex k and any wff f, $f \in k$ iff $|f| \in \Theta(a,k)$. The intension of any wwf f consists of all those complexes k that confirm f. If each complex k is MAX it follows that $INT(f) = |f|$. Otherwise, the INT(f) will not include all the complexes that confirm f. The case when $f = {}_aCON(g)$ such that ${}_aCON(g) \in k$ iff $|g| \in \Theta(a,k)$ follows from the definition of a conceptual structure M part ([4], (d)) and the validity of Rule (RE).

The proof proceeds by induction on the structure of a wff $f \in \Gamma(L_c)$. It is mandatory to show that for every $k \in K$ and every $f \in \Gamma(L_c)$ the relation (R) holds:

(R) $V(f,k) = 1$ iff $f \in k$

Case 1: Let $f = p$. $V(p,k) = 1$ iff $k \in INT(p)$. Since $INT(p) = |p|$ it follows that $p \in k$.

Case 2: Let $f = \neg f$. Then (R) follows from Lemma 1(a).

Case 3: Let $f = g \& h$. Then (R) is a consequence of Lemma 1(b).

Case 4: Consider that $f = {}_aCON(g)$ and let (R) hold for $f = g$. By the inductive hypothesis $V(g,k) = 1$ and $INT(g) = |g|$, so for every k whenever $INT(g) \in \Theta(a,k)$ so is $|g| \in \Theta(a,k)$. From the definition of conceptual structure , part ([4], (d)), $INT(g) \in$

$\Theta(a,k)$ iff $V({}_aCON(g),k) = 1$ iff ${}_aCON(g) \in k$.

Case 5: Suppose that $f,g \in \Gamma(L_c)$ and $V(f \Leftrightarrow g, k) = 1$ for every $k \in K$. From (R) follows that $f \Leftrightarrow g \in k$ for every $k \in K$. So $f \Leftrightarrow g$ belongs to every k where k is MAX, whence $f \Leftrightarrow g$ is a theorem of S_c. By Lemma 1(c) then for every $k \in K$, $f \in k$ iff $g \in k$. Using the rule (RE) if $f \Leftrightarrow g$ is a theorem of S_c so is ${}_aCON(f) \Leftrightarrow {}_aCON(g)$ a theorem of S_c. Therefore ${}_aCON(f) \Leftrightarrow {}_aCON(g) \in k$ for all $k \in K$.

Hence M is a conceptual structure for S_c. If $f \in \Gamma(L_c)$ and f is S_c valid formula, then $V(f,k) = 1$ for all $k \in K$ From (R) follows that $f \in k$ for all $k \in K$. But each $k \in K$ is MAX set of wffs. Applying Lemma 1(d) yields that f is a theorem of S_c, i. e., provable in S_c.■

□ Corollary: A set of formulas Ψ of LC is S_c consistent iff Ψ is S_c satisfiable.

Proof:

(only if)

Let M be a conceptual structure as defined in the Proposition. Since Ψ is S_c consistent then Ψ is a subset of some $k \in K$. Therefore, $V(f,k) = 1$ for every $f \in \Psi$. Thus Ψ is satisfiable.

(if)

Let Ψ be S_c satisfiable. Then for any formula $f \in \Psi$, $V(f,k) = 1$ for some $k \in K$. The last assertion states that if f is satisfiable, $\neg f$ is certainly not. So Ψ is consistent with respect to negation in the system S_c.■

APPENDIX B

References:

(KAP64) Kaplan, S.C. "Foundation of Intensional Logic", Doctoral Dissertation, UCLA, 1964.

(KRI65) Kripke,S. "Semantical Analysis of Modal Logic II: Non-normal Propositional Calculi" in The Theory of Models, ed. Henkin,L., and Tarski,A., North-Holland, 1965,pp. 206 -220.

(KRI63) Kripke,S. "Semantical Consideration on Modal Logic", Acta Philosophica Fennica, 16, 1963, pp. 83 -94.

(CHE80) Chellas, B.F. "Modal Logic", Cambridge University Press, 1980.

(POP87) Popov, B. O. "Intensional Reasoning about Knowledge", Doctoral Dissertation, UMR, 1987.

(POP90) Popov, B. O. "The System of Knowledge", Informatika, Vol. 14, No. 4,October, 1990, pp. 87-90.

(POP91) Popov, B. O. "An Intensional

Approach towards Omniscience", International Symposium on Information Technology, Sarajevo, March, 1991, pp. 235/1-235/5.

Programski paket MSI Reality

Podjetje **Prometej,** d.o.o., p.p. 185, 64001 Kranj, ponuja programski paket *MSI REALITY,* ki deluje na operacijskih sistemih MS-DOS, VMS in UNIX in na računalnikih Hewlett Packard, Apollo, Sun, Digital, Mackintosh in IBM, pri sistemih UNIX na način X- windows in MOTIF s pomočjo uporabniško določenih objektov ICONS. Zagotovljeno je izobraževanje in podpora. Seznam programske opreme je tale:

RAČUNALNIŠKO PODPRTA RAZVOJNA ORODJA za analize elektronskih vezij in termičnih prenosov v okolico elektronskih delov - CAD;

ELEKTRONSKA IZMENJAVA PODATKOV za trenuten prenos podatkov, dokumentov in načrtov med trgujočimi partnerji - EDI;

ANALIZA NAPAKOVNIH NAČINOV IN EFEKTOV po metodi iz nižje na višjo reven sistema ali procesa z avtomatskimi preglednicami - FMEA/FMECA;

STATISTIČNA PROCESNA KONTROLA - METODE DR. TAGUCHIJA za ocenitve sposobnosti procesov, za kontrolo nad procesi in za razvojne eksperimente za odstranitev notranjih in zunanjih vzrokov variabilnosti procesov - SPC;

NADZOR DOBAVITELJEV - KONTROLA MERITEV za kvalitativno ocenitev dobaviteljev na osnovi analize partij in za kontrolo in kalibracijo strojev in meril - VENDOR/GAGING;

ANALIZE VARNOSTI IN TVEGANJA za nadzor nad procesi s pomočjo drevesa uspeha ali drevesa odpovedi po metodi naključnih procesov - TREE MASTER;

ANALIZA RAZPOLOŽLJIVOSTI po metodi ocenitve blokovnega diagrama, metodi Monte Carlo, metodi drevesa odpovedi ali metodi večdimenzionalnega naključnega sistema - BDE/AVAILABILITY/TREE/SUPER-KUB;

ANALIZA ZANESLJIVOSTI po metodi MIL-HDKK-217E in BELLCORE za elektronska vezja in sisteme v spreminjajočem se okolju za vojaško, industrijsko in komercialno kakovost - RELIABILITY;

ANALIZA VZDRŽEVALNOSTI za določitev vzdrževalnih del, njihovih časov in načrtovanje vzdrževalnih posegov - MAINTAINABILITY;

LOGISTIKA - REZERVNI DELI - STROŠKI ŽIVLJENSKE DOBE za razporeditev in pregled nad vzdrževalnimi posegi, optimizacijo rezervnih delov in stroškovno analizo vzdrževanja - LISA/CORIDA/PPCM; in

KONFIGURACIJSKO VODENJE za definiranje konfiguracij sistema, kontrolo nad verzijami, modifikacijami in statusom - BLOODHOUND.

Dr. Rihard Piskar

PRIMENA VIŠEDIMEZIONALNIH NIZOVA U PROGRAMIMA ZA MERENJE RAČUNARSKIH PERFORMANSI

INFORMATICA 3/91

Keywords: benchmark program, multiple-subscripted arrays

Jozo J. Dujmović
Elektrotehnički fakultet Beograd
P.F. 816, 11001 Beograd

U ovom radu proučavamo efekte korišćenja višedimenzionalnih nizova u programima za merenje računarskih performansi ("benchmark" programi). Najpre se identifikuju i na primerima ilustruju tipični problemi koji su uzrokovani primenom višedimenzionalnih nizova, a zatim se pokazuje da primena ovkvih nizova u programima za merenje računarskih performansi treba da bude striktno kontrolisana i najčešće ograničena. U slučaju mernih programa opšte namene koji se koriste za merenje brzine centralnog procesora može se smatrati da su višedimenzionalni nizovi opasni i treba ih izbegavati. Sa druge strane, pokazuje se da su višedimenzionalni nizovi veoma pogodni za merenje optimizacionih osobina kompajlera i efikasnosti tehnika upravljanja memorijom, naročito kada su u pitanju računari sa RISC procesorima.

THE USE OF MULTIPLE-SUBSCRIPTED ARRAYS IN BENCHMARK PROGRAMS. In this paper we study the effects of using multiple-subscripted arrays in benchmark programs. We identify and exemplify typical problems caused by multiple-subscripted arrays and show why their usage in benchmarking should be strictly controlled and frequently restricted. Multiple-subscripted arrays can be considered harmful in the case of general purpose processor-bound benchmarks. On the other hand, the multiple-subscripted arrays are shown to be suitable for measuring the optimizing features of compilers, especially for RISC machines.

1. UVOD

Indeksirane promenljive se često koriste u programiranju i stoga se redovno nalaze i u programima za merenje računarskih performansi (na primer, videti [1, 2, 3,]). Korišćenje višedimenzionalnih nizova uzrokuje dva fundamentalna tipa pristupa podacima u operativnoj memoriji:

- sekvencijalan pristup, i
- direktan pristup.

Neki procesori su tako organizovani da efikasno koriste sekvencijalan pristup memorijskim lokacijama, tj. kod njih se sekvencijalan pristup obavlja znatno brže od direktnog pristupa. Naravno, u takvim slučajevima svi raspoloživi kompajleri bi trebali biti u stanju da identifikuju sekvencijalne pristupe i da za njih generišu efikasniji mašinski kod. Nažalost, u brojnim praktičnim situacijama to nije slučaj.

Korišćenje indeksiranih promenljivih spada u one aspekte korišćenja operativne memorije koji kritično utiču na karakteristike programa za merenje računarskih performansi. Ovaj aspekt je u praksi po svemu sudeći prilično zanemaren, jer autori mernih programa po pravilu izbegavaju da eksplicitno specificiraju koje ciljeve žele da postignu svojim programima po pitanju korišćenja operativne memorije. Osnovna pitanja koja treba razmatrati u vezi sa korišćenjem operativne memorije od strane mernih programa obuhvataju sledeće:

1. Ako je obim obrađenih podataka premali, onda na vreme obrade dominantno utiče raspoloživost cache memorije.

2. Ako je obim podataka dovoljno veliki, onda na vreme obrade može dominantno da utiče mehanizam straničnih prekida.

3. Kod mnogih procesorskih organizacija sekvencijalni pristup memoriji je znatno efikasniji od direktnog pristupa. Prema tome, odnos sekvencijalnog i direktnog pristupa za merne programe treba držati pod striktnom kontrolom, tako da odnos bude neizmenjen za razne procesorske organizacije, a u nekim slučajevima i za različite jezike i njihove kompajlere.

Ovaj rad ima za cilj da pokaže da vreme izvršavanja mernih programa može biti ekstremno osetljivo na varijacije u pristupu memoriji i da je korišćenje višedimenzionalnih nizova primarni uzrok navedenih varijacija. Pored toga, korišćenje višedimenzionalnih nizova pruža mogućnost za razna neželjena podešavanja radnog opterećenja računara, bilo ručno, bilo od strane optimizacionih kompajlera. U nastavku se pokazuje da u mnogim slučajevima merenja performansi višedimenzionalni nizovi mogu biti opasni, tako da njihovu primenu u mernim programima treba striktno kontrolisati.

2. SEKVENCIJALNI I DIREKTNI PRISTUP NIZOVIMA PODATAKA

U slučaju programskog jezika FORTRAN matrice se u memoriji slažu po kolonama i da bi se matrica M(0:n-1, 0:n-1) iskopirala u vektor V(0:n*n-1) možemo primeniti dva prilaza. Ako koristimo programski segment

```
DO j = 0, n-1
   DO i = 0, n-1
      V(n*j+i) = M(i,j)
   END DO
END DO
```

onda se i V i M sekvencijalno obrađuju. Sa druge strane, ako se koristi programski segment

```
DO i = 0, n-1
   DO j = 0, n-1
      V(n*j+i) = M(i,j)
   END DO
END DO
```

onda se komponentama vektora V i matrice M pristupa direktno.

Neki procesori obavljaju sekvencijalni memorijski pristup brže od direktnog pristupa. Na primer, u slučaju klasične PDP-11 arhitekture sekvencijalni pristup nizovima podataka zasniva se na autoinkrementnom registarskom načinu adresiranja, dok se direktan pristup realizuje primenom indeksiranog adresiranja. Kao ilustraciju razmotrimo operaciju "brisanja" V[i]:=0, i=0,1,...,n-1. Operacija brisanja može se izvesti tako da se ponavlja instrukcija CLR (clear) koristeći bilo autoinkrementno adresiranje,

```
CLR (R)+
```

ili indeksirano adresiranje,

```
CLR V(R)  .
```

U prvom slučaju registar R sadrži tekuću adresu komponente vektora i dovoljan je samo jedan pristup memoriji da bi se operacija V[i]:=0 obavila. Pored toga, R se može inkrementirati simultano sa obavljanjem operacije CLR. U drugom slučaju registar R sadrži tekuću vrednost indeksa *i*, dok druga reč mašinske instrukcije sadrži adresu komponente V[0] (što se označava #V). Prema tome, najpre je potrebno dohvatiti #V iz memorije, zatim izračunati efektivnu adresu #V+R, i konačno pristupiti memoriji po drugi put da bi se obavila operacija CLR. Pored toga, R se mora zasebno (ne simultano) inkrementirati pre nego se pristupi obavljanju naredne iteracije. Očigledno, opisano indeksirano adresiranje mora biti manje efikasno od autoinkrementnog registarskog adresiranja.

Radi demonstracije razlika u sekvencijalnoj i direktnoj obradi nizova podataka koristeći razne programske jezike razmotrimo sledeće kratke programske segmente za sumiranje komponenti kvadratne matrice M po kolonama:

```
sum  = 0;
DO j = 0, n-1
     DO i = 0, n-1
        sum = sum + M(i,j)
     END DO
END DO
```

i po vrstama:

```
sum  = 0;
DO j = 0, n-1
     DO i = 0, n-1
        sum = sum + M(j,i)
     END DO
END DO
```

Neka su T_c i T_r respektivno procesorska vremena za obavljanje prikazanih programskih segmenata. Možemo li očekivati da je $T_c = T_r$?

Jedan odgovor na postavljeno pitanje za VAX 11/750 i za n=90 prikazan je u Tabeli 1 gde T_c i T_r označavaju normalizovana procesorska vremena za devet različitih programskih jezika koji se koriste pod istim operativnim sistemom (VMS) i uz korišćenje istog hardvera. Pored toga, Tabela 1 prikazuje varijacioni koeficijent

$$v := 100\ |T_r - T_c|\ /\ (T_r + T_c) \quad [\%]\ ,$$

zatim količnik T_r/T_c, kao i maksimalni količnik varijacije performansi

$$Q := \max(T_r, T_c)\ /\ \min(T_r, T_c)\ .$$

Prema očekivanju, najbrže izvršavanje programa dobija se za simbolički mašinski jezik (MACRO V03). Odgovarajući ekvivalentni programi na višim programskim jezicima mogu biti znatno (više od 6 puta) sporiji. Razlike među raznim višim programskim jezicima za ovako jednostavan programski segment su začuđujuće velike (više od 4 puta). Obrada kvadratne matrice po kolonama i po vrstama (tj. sekvencijalna i direktna obrada niza podataka) može se razlikovati za više od 2 puta (Q faktori za C i PL/1). Naš primer pokazuje da neki kompajleri razlikuju sekvencijalne i direktne memorijske pristupe i mogu to da koriste (C i PL/1 obrađuju matrice po redovima, dok FORTRAN i BLISS rade to po kolonama). Na nesreću i čuđenje, ostali kompajleri za istu

Tabela 1. Komparacija sabiranja komponenti dvoindeksne celobrojne matrice po kolonama i po vrstama za razne jezike u VAX/VMS okruženju

Language		T_r	T_c	v [%]	T_r/T_c	Q
C	V1.3	1.51	3.35	37.74	0.45	2.22
PL/1	V1.3	1.60	3.45	36.48	0.46	2.17
PASCAL	V1.3	4.64	4.81	1.75	0.97	1.03
MACRO	V03	1.00	1.00	0.00	1.00	1.00
COBOL	V2.0	6.29	6.30	0.07	1.00	1.00
CORAL	V1.0	4.12	4.11	0.11	1.00	1.00
BASIC	V1.4	4.05	4.03	0.23	1.00	1.00
FORTRAN	V3.1	3.12	1.96	22.74	1.59	1.59
BLISS	V3	2.91	1.81	23.35	1.61	1.61

mašinu (PASCAL, CORAL, COBOL, BASIC) uvek koriste iste (suboptimalne) načine adresiranja bez obzira na tip pristupa memoriji. Naravno, prikazani numerički primer ne treba generalisati u smislu da su neki jezici u svim slučajevima bolji od nekih drugih jezika (srodni primeri se mogu videti u [4]).

Prikazani Q faktor predstavlja indikator globalne varijacije performansi, i za RISC procesore je po pravilu veći nego za CISC procesore. Na primer, u slučaju gde je n=500 nalazimo Q=4.09 za IBM 6000/320 FORTRAN, i Q=3.8 za IBM 6000/320 C. Međutim, za n=1000, jedna od testiranih mašina serije 6000 dala je (za FORTRAN) Q=5.34, dok je ista mašina, ali sa manjom memorijom, dala samo Q=1.5 kao posledica režije pri upravljanju memorijom. Treba uočiti da bi vrednost Q=1 mogla indicirati i krajnje uspešnu optimizaciju; izgleda da to ponekad važi za HP 9000/835 FORTRAN i C.

Prikazani primeri su dovoljni da pokažu osetljivost vremena izvršavanja programa u odnosu na varijacije odnosa sekvencijalnih i direktnih memorijskih pristupa. Oni takođe pokazuju i to da nivo osetljivosti zavisi od odabranog kompajlera. Dopunski primeri sa izuzetno velikim nivoima osetljivosti, koji se sreću kod RISC procesora, prikazani su u narednom odeljku. Ove činjenice bi trebalo uzimati u obzir kada se specificiraju merni programi i kada se vrši izbor jezika na kome će oni biti implementirani.

3. OPTIMALNA PERMUTACIJA INDEKSA

Eksperimenti sa dvodimenzionalnim matricama se vrlo lako mogu proširiti do opšteg slučaja sa m dimenzija. Ako merni program koristi m-dimenzionalne nizove tada programom dobijeni rezultati ostaju nepromenjeni ako je izabrana permutacija indeksa konzistentno izvršena na svakom mestu u programu gde se dotični niz pojavljuje. Postoji m! različitih *permutacija m indeksa i m! programa koji obavljaju* ekvivalentne procese; oni koriste različite vrste adresiranja i mogu se razlikovati u efikasnosti. Na primer, sledeća dva potprograma za množenje matrica obavljaju ekvivalentna računanja, ali koriste različite odnose sekvencijalnih i slučajnih pristupa memoriji i obično se razlikuju u efikasnosti.

```
SUBROUTINE MM1( A, B, C, N, L, M )
DIMENSION A(30,40),B(40,50),C(30,50)
DO i = 1, N
   DO j = 1, M
      C(i,j) = 0.
      DO k = 1, L
         C(i,j)=C(i,j)+A(i,k)*B(k,j)
      END DO
   END DO
END DO
END
```

```
SUBROUTINE MM2( A, B, C, N, L, M )
DIMENSION A(40,30),B(50,40),C(50,30)
DO i = 1, N
   DO j = 1, M
      C(j,i) = 0.
      DO k = 1, L
         C(j,i)=C(j,i)+A(k,i)*B(j,k)
      END DO
   END DO
END DO
END
```

Ovaj vid permutacije indeksa u slučaju višedimenzionalnih nizova može prouzrokovati skup različitih vremena obrade za suštinski isti program. Da bi ovo prikazali, razmotrimo sledeći segment koda korišćen pri množenju matrica:

```
DO i = 1, n
   DO j = 1, n
      DO k = 1, n
         C(i,j) = C(i,j) + A(i,k) * B(k,j)
      END DO
   END DO
END DO
```

Lako je uočiti da bilo koja permutacija početne tri DO naredbe daje kao rezultat istu matricu C. Ove permutacije ćemo označiti sa ijk (za prikazani segment koda), ikj, jik, jki, kij, i kji. Naravno, vremena obrade ovih permutacija će se obično razlikovati za različite procesore i za različite kompajlere. Tabela 2 sumira odgovarajuće eksperimentalne rezultate i pokazuje da RISC mašine (sa UNIX-om) mogu biti naročito osetljive na permutacije indeksa. Tabela 2 takođe pokazuje da poboljšanja performansi mogu poticati od raznih optimizacionim mogućnosti kompajlera tipičnih za RISC mašine. Slični rezultati su nedavno objavljeni u IBM-ovim merenjima u Velikoj Britaniji [5]: u slučaju množenja matrice veličine 300*300 na računaru IBM 6000/530, uz korišćenje Pre-Release AIX XL Fortran V2.2 Pre-Processor, dobijene su sledeće Mflops vrednosti:

Tabela 2. Normalizovana vremena obrade za razne permutacije petlji u programu za množenje matrica veličine 300*300 (RISC/UNIX, jednostruka tačnost)

Permutation	IBM 6000/520 (AIX)		Hewlett-Packard 9000/835 (UX)		
	FORTRAN(opt)	C(opt)	FORTRAN(opt)	C(opt)	C(no opt)
ijk	1.24	1.21	1.23	1.23	1.03
ikj	8.82	1.00	1.06	1.00	1.00
jik	1.21	1.21	1.22	1.23	1.01
jki	1.00	8.36	1.00	1.04	1.03
kij	8.91	1.00	1.05	1.01	1.00
kji	1.02	8.44	1.01	1.04	1.04

Permutacija:	ijk	ikj	jik	jki	kij	kji
Bez pretprocesora	5.7	22.0	5.6	1.3	21.4	1.3
V2.2 pretprocesor	71	71	71	71	71	71

Ručno podešen FORTRAN postigao je najveću vrednost od 73 Mflops. Stoga navedeni rezultati pokazuju da su optimizujući kompajleri sa pretprocesorom u stanju da se približe najvišim performansama ručno podešenog programa, i da se traženje optimalne permutacije može realizovati automatski. Bez takvog optimizujućeg pretprocesora može doći do drastičnih varijacija u performansama: u zavisnosti od izabrane permutacije rezultujuće performanse mogu se promeniti i do 17 puta!

U slučaju CISC mašina u koje spadaju VAX 11/785 i MicroVAX 3600 naše merenje vremena množenja matrice dimenzije 100*100 u dvostrukoj tačnosti sumirano je u Tabeli 3 i pokazuje manje varijacije performansi negu u slučaju RISC mašina. U svim prikazanim primerima koji obuhvataju 6 permutacija uočavaju se tri različita nivoa performansi prikazana u Tabeli 3; ekvivalentni (ili veoma bliski) parovi permutacija su (ijk, jik), (ikj, kij), i (jki, kji). U nastavku je prikazan sintetski merni program MMUL za merenje optimizacionih osobina kompajlera i efikasnosti upravljanja operativnom memorijom; MMUL koristi tri karakteristične petlje pri množenju matrica i može se bez problema implementirati na bilo kom programskom jeziku u cilju ispitivanja osobina odgovarajućeg kompajlera.

Veličina matrice (N) je jedini podešljivi parametar u ovom programu. Glavni rezultati MMUL-a su tri vremena karakterističnih petlji: T_i, T_j, i T_k. Variranja ovih rezultata za razne vrednosti N mogu biti znatna, što se vidi u Tabeli 4. Ako je program MMUL preveden sa i bez optimizacije, odnos odgovarajućih vremena izvršavanja ("optimizaciono pojačanje") je približno 2 za Vax 3600, 4 za HP 9000/835, i, u zavisnosti od N, od 2 do 7 za IBM 6000/530.

ZAKLJUČAK

Programi za merenje performansi mogu biti namenjeni merenju celog spektra različitih osobina, od globalnih performansi sistema (uključujući i hardver i softver), pa do više specifičnih karakteristika, kao što su optimizujuće osobine određenog kompajlera. U slučaju merenja globalnih performansi, trebali bi pre svega biti zainteresovani za stabilnost izabranog opterećenja za različite konfiguracije hardvera i softvera. Drugim rečima, ne sme se dozvoliti da neznatno podešavanje opterećenja ili određena optimizaciona osobina kompajlera izmeni glavne izmerene indikatore performansi za red veličine. Ako program za merenje performansi koristi nizove, tada bi on trebao biti tako organizovan da odnos sekvencijalnih i slučajnih memorijskih pristupa ostane nepromenjen za različite procesorske organizacije, i za različite kompajlere. Budući da upotreba višedimenzionalnih nizova u programima za merenje performansi može biti glavni uzrok velikih varijacija vremena obrade, ima razloga da se u opštem slučaju tretiraju kao opasni i izostavljaju iz procesorski ograničenih mernih programa opšte namene. To nije teško ostvariti budući da se iste obrade mogu realizovati i korišćenjem jednodimenzionalnih nizova.

Sa druge strane, pokazali smo da je korišćenje višedimenzionalnih nizova veoma korisno u programima za merenje optimizacionih mogućnosti kompajlera i/ili efikasnosti upravljanja memorijom. To je naročito važno u slučaju upoređivanja CISC and RISC mašina. Dok se mašinski kod CISC mašina može optimizovati izborom najbržih načina adresiranja, RISC mašine koriste mali broj načina adresiranja (ne više od četri) i optimizacija njihovog mašinskog koda je bazirana na drugim tehnikama (uglavnom se koriste mogućnosti paralelizma izvršavanja na nivou instrukcija).

ZAHVALNOST

Ovaj rad je delimično finansiro Republički fond za nauku Srbije. Autor je zahvalan Ivu Dujmoviću za učešće u merenjima performansi, kao i za pomoć kod pripreme rada.

Tabela 3. Normalizovana vremena obrade za razne permutacije petlji kod programa za množenje matrica dimenzije 100*100 u dvostrukoj tačnosti (CISC procesori)

Permutation	VAX 11/785 (VMS)		MicroVax 3600 (VMS)	
	FORTRAN(opt)	C(opt)	FORTRAN(opt)	C(opt)
ijk	1.02	1.15	1.18	1.37
ikj	1.24	1.00	1.38	1.00
jik	1.04	1.16	1.18	1.43
jki	1.00	1.40	1.00	1.78
kij	1.27	1.03	1.40	1.00
kji	1.05	1.40	1.00	1.70

Tabela 4. Odabrani rezultati programa MMUL

	MicroVax 3600				Hewlett-Packard 900/835				IBM 6000/530			
N	100	200	300	400	100	200	300	400	100	200	300	400
T_i	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
T_j	1.60	1.76	1.82	1.90	1.00	1.06	1.13	1.09	2.57	4.44	9.49	13.2
T_k	1.29	1.40	1.41	1.44	1.22	1.26	1.23	1.24	1.43	1.28	1.31	2.58

```
C=====================================================
C     MMUL - MATRIX MULTIPLICATION BENCHMARK PROGRAM
C=====================================================
      PARAMETER (N=300)
      DIMENSION A(N,N), B(N,N), C(N,N)
      DO 1 I = 1,N        !   Data initialization loop
         DO 1 J = 1,N
            A(I,J) = I+J
            B(I,J) = A(I,J)
    1       C(I,J) = 0.

      S0 = SECNDS(0.0)    !   The system time in seconds

      DO 2 J = 1,N        !   The first segment yielding Ti
         DO 2 K = 1,N
            DO 2 I = 1,N
    2          C(I,J) = C(I,J) + A(I,K)*B(K,J)

      S1 = SECNDS(0.0)
      T1 = S1 - S0

      DO 3 K = 1,N        !   The second segment yielding Tj
         DO 3 I = 1,N
            DO 3 J = 1,N
    3          C(I,J) = C(I,J) + A(I,K)*B(K,J)

      S2 = SECNDS(0.0)
      T2 = S2 - S1

      DO 4 I = 1,N        !   The third segment yielding Tk
         DO 4 J = 1,N
            DO 4 K = 1,N
    4          C(I,J) = C(I,J) + A(I,K)*B(K,J)

      S3 = SECNDS(0.0)
      T3 = S3 - S2
      T  = min(T1,T2,T3)
      WRITE(*,5) N,'i',T1,T1/T,'j',T2,T2/T,'k',T3,T3/T,S3-S0,C(N,N)
    5 FORMAT(//' MATRIX MULTIPLICATION TIME FOR MATRIX SIZE =', I4//
     *       3(' T', A ,' =', F7.2, ' sec', 8X, '(', F5.2,' )'/),
     *       16('-') / ' T  =', F7.2, ' sec', 20X,'C(N,N) =',E13.6)
      END
```

REFERENCE

[1] H. J. Curnow and B. A. Wichman. A Synthetic Benchmark. Computer J., Vol. 19, No. 1 (1976), pp. 43-49.

[2] R. P. Weicker. Dhrystone: A Synthetic Systems Programming Benchmark. CACM Vol. 27, No. 10 (1984), pp. 1013-1030.

[3] F. H. McMahon. The Livermore Fortran Kernels: A Computer Test of the Numerical Performance Range. Lawrence Livermore National Laboratory, December 1986.

[4] J. J. Dujmovic. Compiler Performance Measurement and Analysis. Informatica, Vol. 6, No. 2 (1982), pp. 45-54.

[5] R. Bell. IBM FORTRAN Compiler Performance Developments and Direction. Scientific and Technical Computing - IBM International Customer Executive Seminar, La Hulpe, Belgium, May 1991.

»SLONČEK«, PROGRAM ZA GESLENJE SLOVENSKEGA TEKSTA

INFORMATICA 3/91

Keywords: Spelling checker, Slovene language, Word analysis

Jure Zupan
samostojni znanstvenik
Dragomer, 61351 Brezovica

ABSTRACT The principles of the spelling checker of the slovene language implemented in the program package 'SLONČEK' are described and discussed. Only the option called 'Analysis of words' in the plain text is described in detail. The analysis of words is based on the search for all combinations of 'roots' and 'sufices' that could give the valid words recognized in a dictionary. Besides the analysis of each word the program identifies 10 punctuation marks, hence giving a complete information for further syntax and semantic check of new words in the plain text. The dictionary of the program contains about 250 sets of sufices (approx. 20 sufices/set) and more than 32,000 'roots' of slovene words.

POVZETEK Opisan je koncept in izvedba programskega paketa SLONČEK za pregledovanje slovenskih tekstov. V prispevku je podrobneje obdelana le opcija za geslenje 'Analiza'. Geslenje je osnovano na iskanju in izpisu vseh možnih 'korenov' in pripadajočih 'pripon' za vsako besedo v tekstu. Program identificira še 10 ločil, tako da rezultirajoče geslenje daje polno izhodišče za nadaljnje sintaktično in semantično obdelavo besedil. Program vsebuje več kot 32.000 'korenov' slovenskih besed in približno 250 'nizov pripon' (v povprečju 20 pripon/niz) različnih sklanjatev in spregatev.

1 Uvod

Pregledovalnike slovenskih besedil bi glede na namen delovanja lahko razdelili v dve skupini. V prvi so pregledovalniki, ki preverjajo pravilnost zapisane besede in do neke mere kontrolirajo pravilnost sintakse (raba velike začetnice, raba s, z, h in k itd.). V drugi vrsti so pregledovalniki, ki poleg kontrole pravilnosti zapisane besede določijo pregledovani besedi vse možne **osnovne** oblike (nedoločnik, prvi sklon moške oblike itd.) in vse možne **ustrezne** oblike (sklon, oseba, čas itd.) v kateri najdena osnovna oblika lahko nastopa. Dodatno pregledujejo še vrsto in položaj ločil, ki so bistvena za razumevanje besedila ali za določanje sintaktičnih povezav.

Da bo povsem jasno, kaj je mišljeno z omenjenimi dejavnostmi, navajam v pojasnilo nekaj primerov. Oglejmo si naslednje stavke:

Priddi h meni.

Hvala, lep dete!

Župan je rekel: "To je dobro za vas".

V prvem stavku sta dve napaki, ki ju lahko odkrije že program, ki ni namenjen geslenju. Beseda 'priddi' je napačno napisana in program je ne sme imeti v slovarju. Poleg tega je namesto predloga 'k' uporabljen predlog 'h'. Tovrstno napako lahko vedno odkrijemo s preverjanjem prve črke sledeče besede.

Drugi stavek nima z gledišča pravilnosti pisanja besed nobene napake, pač pa ima očitno napačno sintakso med pridevnikom 'lep' in samostalnikom 'dete'. Če naj bi program opozoril na tovrstne napake, mora ugotoviti, da je beseda 'lep' pridevnik **moškega** spola v prvem sklonu ednine, naslednja beseda pa samostalnik **srednjega** spola v

prvem sklonu ednine, kar ni v skladu s pravili sintakse. Pri tem stavku je treba opozoriti še na dejstvo, da mora program preveriti sintakso v skaladu z ločili. Če bi, na primer, namesto za besedo hvala stala vejiaca za besedo lep, bi moral program pridevnik 'lep' uskalditi s samostalnikom 'hvala' in ne s samostalnikom 'dete'. Za pravilno izvajanje sintakse mora torej program poleg gesel in njihovih možnih oblik podati tudi vse potrebne podatke o vrsti in položaju ločil v stavku.

Tretji stavek je sicer pravilen tako v pravopisnem kot tudi v sintaktičnem smislu, je pa seveda popolnoma nedoločljiv s stališča semantike. Za razumevanje pomena tega stavka (n.pr. za prevod v tuj jezik) je nujno potrebna obdelava **vseh** gesel za **vse** besede v kontekstu sledečih oziroma predhodnih stavkov. Pri takih primerih mora orodje, ki naj omogoči pravilno razumevanje stavka (za prevod) zajeti **vse možne pomene (gesla) vseh besed** v posameznih stavkih. Zato mora najprej ločiti stavke med seboj (obdelava ločil) in nato nuditi vse možne variante osnovnih gesel.

Šele ko imamo na voljo orodje, ki v preiskovanem tekstu poda zgoraj navedene podatke (vsa gesla in z njimi vse možne oblike spregatev, spolov, časov in sklonov ter vrsto in položaje ločil), lahko nadaljujemo s sintaktično in kasneje s semantično obdelavo besedila.

V nadaljevanju sestavka je opisana osnovna ideja in algoritem programa SLONČEK, ki vrši vse zgoraj omenjene osnovne funkcije geslenja. Program SLONČEK je (med drugim) orodje za geslenje, podajanje slovničnih oblik in identifikaciji ločil. Rezultirajoča datoteka zaporedja gesel, slovničnih oblik in identifikacij ločil je osova za nadaljnjo sintaktično in semantično obdelavo predloženega besedila. Podanih je tudi nekaj primerov (glej poglavje Primeri).

2 Koncept geslenja s programom SLONČEK

Osnovna zamisel geslenja pri programu SLONČEK je deljenje vsake besede na vrsto 'korenov' in 'pripon' in pregledovanje slovarja, če se posamezna delitev na 'koren' in 'pripono' ujema z ustreznimi podatki v bazi 'korenov' in 'standardnih pripon'. Deljenje preiskovane besede na 'koren' in 'pripono' poteka postopno - po posameznih črkah. Vsota črk 'korena' in 'pripone' je ves čas stalna: kolikor krajši je 'koren', toliko daljša je 'pripona'. Najdaljša 'pripona' je lahko dolga šest, najkrajša pa nič črk. Program vedno prične preiskovati slovar za dano besedo s korenom, ki je enak celotni besedi (pripona dolga nič črk). Če program korena v slovarju ne najde, odreže 'korenu' zadnjo črko, jo doda na začetek stare 'pripone' in išče ponovno. Rezanje zadnje črke 'korena' in iskanje v slovarju, ponavlja program tako dolgo, da 'korena' ni več ali pa je bilo odrezanih že šest črk. To pomeni, da je najdaljša 'pripona' lahko dolga le šest črk. Glej Tabelo I.

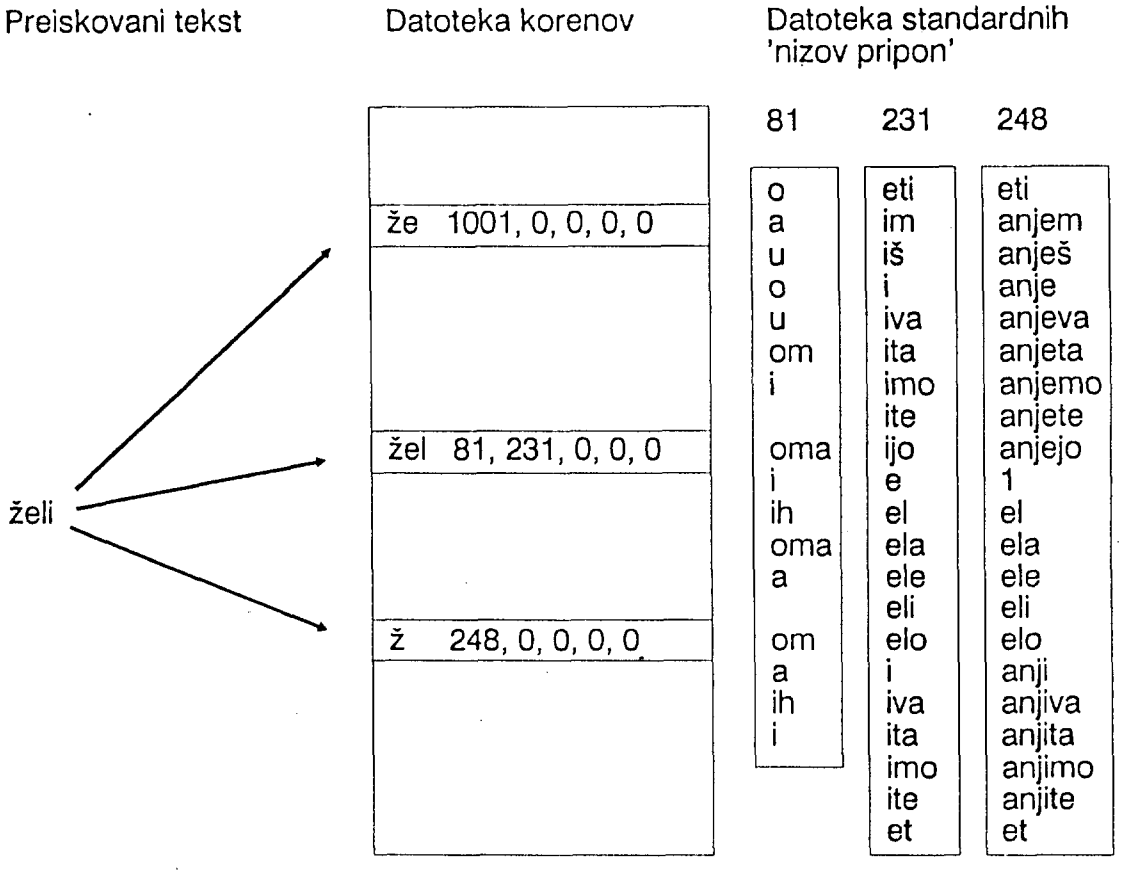
Tabela I. Rezanje preiskovane besede na 'korene' in 'pripone'. Če je beseda krajša kot šest črk (primer: mati), se algoritem razbijanja zaključi že prej.

Korak	'Koren'	'Pripona'
1	pregledovati	-
2	pregledovat	(i)
3	pregledova	(ti)
4	pregledov	(ati)
5	pregledo	(vati)
6	pregled	(ovati)
7	pregle	(dovati)
1	mati	-
2	mat	(i)
3	ma	(ti)
4	m	(ati)

Slovar sestavljajo 'koreni' ob katerih so vpisani vsi možni nizi 'pripon'. Nizi 'pripon' so identificirani z zaporednimi številkami od 1-280. Ko algoritem odkrije navzočnost 'korena' v slovarju, prične preverjati ali je 'pripona', odrezana od preiskovane besede, vsebovana v 'nizu pripon' identificiranih ob korenu v slovarju. Če 'pripone' v standardnem nizu ni, ima preiskovana beseda status 'neznana'. V primeru, ko odrezano 'pripono' v nizu najdemo, postane preiskovana beseda 'znana'. Algoritem preveri še kolikokrat in na katerih mestih se v identificiranem 'nizu standardnih pripon' odrezana 'pripona' ponovi. Mesto, kjer se odrezana

'pripona' v nizu nahaja, namreč pove sklon, spol, število in/ali čas v katerem se preiskovana beseda nahaja. Slika 1 kaže kako algortem pregleda besedico 'želi'.

Vse dobljene podatke o vsaki besedi (vsa gesla in mesta v nizu kjer se 'pripona' ujema s standardno pripono) algoritem izpiše na izhodno datoteko (glej poglavje Primeri).



Slika 1. Pregledovanje datoteke korenov in pripon za besedico 'želi'. Celotne besede 'želi' ni v datoteki. Koren 'žel' nastopa dvakrat, enkrat z nizom pripon št. 81 (samostalniki srednjega spola na 'o') in enkrat z nizom št. 231 (glagoli na 'eti'). Koren 'že' nastopa samo enkrat kot samostojna beseda (jedro) brez niza 'pripon' (oznaka 1001). Ker je koren 'že' jedro, torej po definiciji brez 'pripon', besedica 'že' v danem primeru ni potrjena kot možna. Koren 'ž' nastopa v datoteki z nizom pripon št. 248 (glagoli na 'eti').

Na sliki 1 se razločno vidi, da SLONČKOV algoritem pregleda za besedo 'želi' štiri možne 'korene: 'želi', 'žel', 'že' in 'ž'. Pri tem najde **tri** korene ('žel', 'že' in 'ž') v slovarju. Kljub temu, da imajo najdeni trije koreni skupno štiri 'nize pripon' (81,231, 248 in 1001) so možna le tri gesla: želo, želi in želeti - koren 'že' z oznako 1001 (jedro) v danem primeru ne pride v poštev. Za ostale korene pa vedno najde ustrezno 'pripono' v 'nizu'. Pri geslu želo, najde tri možne oblike (prvi in četrti sklon dvojine ter šesti sklon množine), pri glagolu 'želeti' tretjo osebo sedanjika in velelnik in končno pri glagolu 'želi' najde 'pripono' 'eli' (koren je 'ž') v skupini kjer so množinska obrazila za prihodnji in pretekli čas. Osnovno obliko gesla algoritem dobi tako, da 'korenu' doda 'pripono', ki je v pridruženem 'nizu pripon' na **prvem** mestu.

3 Organizacija slovarja korenov in pripon

Slovar korenov. Kot je bilo že omenjeno, so gesla, ki jih podaja program SLONČEK, dobljena s pomočjo slovarja 'korenov' in ustreznih nizov 'pripon'. Slika 1 kaže datoteko z neposrednim dostopom v kateri so koreni dosegljivi preko 'hash' (razpršilnega) algoritma (1) s pomočjo praštevilskih dvojčkov (2). SLONČEK ima korene razporejene v treh datotekah. Vse tri datoteke so enako dolge (vsaka ima 20023 zapisov), ločijo se le po dolžini zapisa. V prvi datoteki so shranjeni koreni dolgi do vključno šest znakov, v drugi so koreni dolgi 7, 8 in 9 znakov, v tretji pa so shranjeni vsi daljši koreni. Največja dovoljena dolžina korena je 18

znakov. Vsak polni zapis vsebuje poleg korena tudi oznako (ali več oznak) niza(ov) pripon, ki z danim korenom tvori(jo) smiselna gesla. Kot se vidi s slike 1, ima datoteka z najkrajšimi koreni prostor za identifikacijo petih nizov, srednja datoteka ima lahko tri, ob najdaljših korenih pa je prostora za identifikacijo le dveh nizov.

Dolžina zapisov je bila izbrana izkustveno, na podlagi statistike približno 15.000 besed. Med delom se je pokazalo, da bo v prihodnosti potrebno podaljšati zapise, da bi dobili možnost hranjenja večjega števila različnih nizov. Če sodi beseda v tako imenovano 'jedro' jezika (3), potem ima namesto identifikacije niza posebno oznako (1001).

Nizi pripon. Z izrazom 'pripona' označujem tisti del besede, ki skupaj s 'korenom', ki je shranjen v eni od omenjenih treh datotek, tvori pravilno slovensko besedo v poljubni obliki, sklonu, spolu, času, osebi itd. 'Niz pripon' je torej skupina 'pripon', ki skupaj s 'korenom' tvorijo vse oblike sklanjatve ali spregatve. Najbolj značilno za 'niz pripon', ki ga uvaja programa SLONČEK je, da prva črka vseh 'pripon' v nizu ni nikoli enaka pri vseh priponah. Če bi namreč bila, bi že sodila h 'korenu'.

'Pripona' v konceptu programa SLONČEK torej nima slovničnega pomena, pač pa čisto računalniškega: to je tisti del besede (samostalnika, pridevnika, glagola itd.), ki pri sklanjatvi ali spregatvi vsaj enkrat zamenja začetno črko. Z drugimi besedami povedano, 'koren' je tisti del besede, ki pri celotni sklanjatvi ali spregatvi ostane neizpremenjen. Pravilo o nelzpreminjanju korena je osnovna stalnica program SLONČEK. Velja tako pri polglasniških 'e-jih', kot pri raznih mehčanjih in podobnih posebnostih slovenščine.

Program SLONČEK trenutno razlikuje 6 vrst pripon: štiri samostalniške (moške, ženske, srednje in množinske) in po eno vrsto pridevniških (sem sodijo tudi deležniške in druge podobne) in glagolskih. Zaradi omenjene togosti 'korena', pozna SLONČEK skoro 120 različnih samostalniških, 14 pridevniških in preko 70 glagolskih 'nizov pripon'.

Vsi samostalniški nizi pripon so sestavljeni iz 18 pripon, po 6 za vsak sklon ednine, dvojine in množine. Množinski samostalniki imajo, kar je razumljivo, le 6 pripon. Pridevniški nizi imajo 13 pripon, glagolski pa 20. Tabela II kaže nekaj primerov 'nizov pripon'. Pomembno je opozoriti, da je istovrstna pripona, torej pripona istega sklona, časa, spola, ipd., v nizu vedno na istem mestu. S tem lahko program, ko pripono spozna, določi sklon, spol, čas itd., ki ga beseda v tekstu ima. V primeru, da je v nizu **več enakih** 'pripon' (to je možno samo pri samostalnikih) poda računalnik **vsa** mesta (sklone, čase) v nizu kjer je prišlo do ujemanja.

Tabela II. Šest primerov različnih vrst nizov pripon. Pod vsakim nizom je podan tudi primer. Enica v nizu pomeni, da pri danem nizu samostojni koren ni možen.

1	41	85	129	167	207
-	a	e	je	ek	ovati
a	e	a	ij	ka	ujem
u	i	u	jam	ke	uješ
a	o	e	je	ki	uje
u	i	u	jah	ko	ujeva
om	o	em	jami	kega	ujeta
a	i	i	---------	kemu	ujemo
ov				kem	ujete
oma	ama	ema		kim	ujejo
a	i	i		kih	1
ih	ah	ih		kima	oval
oma	ama	ema		kimi	ovala
i	e	a		ak	ovale
ov				---------	ovali
om	am	em			ovalo
e	e	a			uj
ih	ah	ih			ujva
i	ami	i			ujmo
----------					ujta
					ujte
					ovat
vol	žen(a)	polj(e)	škar(je)	meh(ek)	k(ovati)

Kot primer lahko služi beseda 'žerjava', ki ga lahko sklanjamo (ob 'korenu' 'žerjav') s priponami iz niza št. '1' in niza št. '5'. Pripona 'a' je v nizu št. '1' (osebe in živali moškega spola) na mestih 2, 4, 7 in 10 (2. in 4. sklon ednine ter 1 in 4 sklon dvojine), v nizu s št. '5' pa le na 2,7 in 10 mestu (žerjav

kot predmet v četrtem sklonu ednine nima obrazila 'a'). Ker sta ob korenu 'žerjav' v slovarju navedena **oba** niza (torej niz št. 1 in št. 5), bomo pri geslenju za kakršnokoli obliko besede 'žerjav' dobili podatke iz obeh nizov!

Posebej je treba poudariti, da je na **prvem mestu** v nizu vedno tista pripona s katero tvori 'koren' slovarsko **geslo**. Ker število nizov pripon še ni dokončno določeno, lahko uporabnik, v primeru da najde tako besedo, ki se ne pregiba po nobenem od obstoječih nizov, doda manjkajoči niz v opciji 'Dodajanje posameznih besed' ali pri 'Interaktivnem popravljanju besedila'.

4 Primeri

Kot primer geslenja, je na sliki 2 navedena obdelava treh stavkov s programom SLONČEK. Analizirani stavki so

Pridi	priti (228) 16
gori	gori (1001) 0
	gora (41) 3 5 7 10
	goreti (213) 4 16
k	k (1001) 0
meni	meni (1001) 0
	meniti (181) 4 10 16
	mena (41) 3 5 7 10
.	ločilo št.: 1
Hvala	hvala (41) 1
,	ločilo št.: 4
lep	lep (161) 1
dete	dete (90) 1 4
.	ločilo št.: 1
župan	župan (1) 1
je	je (1001) 0
	jesti (229) 4
rekel	reči (234) 11
:	ločilo št.: 6
"	ločilo št.: 10
Skrbi	skrb (47) 2 3 5 7 8 10 13 14 16
	skrbeti (213) 4 16
me	me (1001) 0
za	za (1001) 0
vas	vas (47) 1 4
	vas (145) 1
"	ločilo št.: 10
.	ločilo št.: 1

Slika 2. Trije stavki prostega teksta (na levi), obdelani z opcijo 'Analiza' programa SLONČEK. Gesla z ustreznimi podatki o možnih oblikah so v izhodni datoteki zapisana na desni strani dvojne črte. Glavna tri ločila (pika, klicaj in vprašaj) povzroče v izhodni datoteki poleg opisa še horizontalno delitev.

popravljeni stavki, ki sem jih navedel na začetku. Popravljene so samo tipkarske napake. Rezultat je dobljen z uporabo opcije 'ANALIZA', ki vrši geslenje, tako kot je opisano v zgornjih odstavkih. Program SLONČEK med geslenjem (sicer pa ne) razpoznava 10 ločil in jih v izhodni datoteki opremi z ustreznimi zaporednimi številkami (pika = 1, klicaj = 2, vprašaj = 3, vejica = 4, podpičje = 5, dvopičje = 6, pomišljaj = 7, predklepaj = 8, zaklepaj = 9 in narekovaj = 10). Izhodna datoteka z originalnim tekstom na levi in z gesli na desni je povsem navaden alfanumeričen zapis in ga lahko kasneje obdelujemo s poljubnim programom.

5 Zaključek

Če kompleksno obdelavo teksta razdelimo na štiri oziroma pet faz relativno lahko ločljivih procedur (črkovanje, geslenje, sintaksa, semantika, prevod v tuj jezik), potem lahko rečemo, da opravlja program SLONČEK prvi dve dejavnosti. Mimo lahko trdimo, da je geslo (geslenje besed) bistveni podatek, ki je potreben za vse nadaljnje obdelave teksta v smeri njegovega razumevanja. Z opisanim programom SLONČEK (bolj natančno z njegovo opcijo 'Geslenje') je dano osnovno orodje, ki omogoča omenjena bolj zapletene dejavnosti (sintaksa, stavčna analiza, razumevanje konteksta in prevajanje). V opisani opciji je implementiran koncept, da mora biti programski del, ki opravlja geslenje teksta, ločen od delov programa, ki opravljajo zahtevnejše operacije. Bistveni razlog je v tem, da dopušča slovenščina veliko preveč dvoumnosti in različnih interpretacij, da bi bilo tako zahtevno delo opravljeno sproti.

Poudariti je treba, da kljub temu, da število nizov pripon že presega število 250, iskanje novih pregibanj in dopolnjevanje pripon še ni končano. Posebej problematični so še števniki, vseh vrst zaimkov in oblike pomožnih glagolov, ki so trenutno dodani v slovar korenov, kot 'jedro' (oznake 1001, 1002 in 1003), oblikovno pa sploh še niso razdelani v nize pripon, razen nekaj izjem (glej sliko 2, primer besedice 'vas'). V opciji 'Dodajanje posameznih besed v slovar', si lahko uporabnik sam zgradi in doda poljubne nize pripon. Prav tako enostavno je dodajanje novih besed, saj program SLONČEK vodi uporabnika k odločitvi s prikazovanjem vseh možnih 'nizov pripon' za določeno vrsto besede.

Opisani program SLONČEK je operativen na osebnih računalnikih vrste IBM PC/ XT/AT/PS, ki imajo na voljo trdi disk. Delovanje iz gibkih diskov je zaradi velikega števila posegov na disk pri geslenju (6-12 posegov/besedo) zelo počasno. Popravljanje tekstov (zamenjevanje napačno napisanih besed) lahko opravimo interaktivno (sprotno) ali pa s t.i. označevanjem besed, v 'batch modu'. SLONČEK vsebuje tudi hitro in enostavno pregledovanje slovarja in besed ter pripon, ki so v njem shranjene.

Slovar programskega paketa SLONČEK vsebuje približno 32.000 'korenov' zbranih iz različnih virov(4-7) in nekaj več kot 250 nizov pripon, kar ob povprečni dolžini 20 pripon na niz, da okrog 5000 pripon. Ker ostale opcije, ki jih program SLONČEK nudi, niso bile opisane natančneje, jih na tem mestu samo naštejemo:

- interaktivno popravljanje ASCII besedil,
- markiranje napak v ASCII besedilu ('batch-mode'),
- geslenje teksta in identifikacija ločil,
- dodajanje novih besed v slovar in
- pregledovanje in popravljanje slovarja

Pri testiranju dveh besedil dolgih vsak po okrog 20.000 besed (izpod peres dveh različnih avtorjev) se je pokazalo, da SLONČEK prepozna cca 97-98 % besed. Testiranje je bilo narejeno z opcijo 'Markiranje' neznanih besedi.

Delovanje programa si vsak zainteresirani lahko ogleda pri avtorju na Kemijskem Inštitutu "Boris Kidrič". Uporaba programa za raziskovalne namene, je možna po dogovoru z avtorjem.

ZAHVALA

Zahvaljujem se Kemijskemu Inštitutu "Boris Kidrič" za finančno in organizacijsko pomoč pri izvedbi okrogle mize "Besedišče slovenskega jezika", dne 16. junija 1991. Glede na skrb, ki jo slovenski novinarji posvečajo slovenščini, je

seveda popolnoma jasno zakaj se, kljub obvestilom, ki so bila poslana vsem slovenskim časopisnim hišam, okrogle mize ni udeležil noben njihov sodelavec.

Pričujoča raziskava **ni bila** financirana niti s strani bivše RSS niti s strani RS ZRTD.

REFERENCE

1. D.E. Knuth, The Art of Computer Programming, Vol. 2., Seminumerical Algorithms, Addison-Wesley, Reading, Mass. 1975, Chapter 3.4

2. J. Zupan, Algortihms for Chemists, J. Wiley, Chichester, 1989, str. 34-42,

3. Okrogla miza 'Računalniška besedišča' na SAZU, 14. junij 1991, Ljubljana,

4. D. Debenjak, Slovensko-nemški slovar, CZ, Ljubljana, 1974 (cca. 15.000 besed)

5. F. Prešeren, Poezije, Ljubljana, 1848, (cca.25.000 besed)

6. S. Gregorčič, Poezije, DSM, Celovec, 1908 in Naša beseda, MK, Ljubljana, 1960, (cca. 26.000 besed),

7. J. Menart, Parizina, DZS, Ljubljana, 1960; Semaforl mladosti, DZS, Ljubljana, 1963; Srednjeveške pridige in balade, CZ, 1990 (cca 26.500 besed)

PRIMERJALNA ANALIZA TREH ORODIJ ZA IZGRADNJO IN UPORABO BAZE ZNANJA EKSPERTNEGA SISTEMA

INFORMATICA 3/91

Keywords: expert system, knowledge base, automatic learning, credit worthiness, knowledge acquisition, expert system maintenance

V. Rajković[1,2], E. Delidžakova-Drenik[3], B. Urh[2]
[1] Fakulteta za organizacijske vede, Prešernova 11, Kranj
[2] Institut »Jožef Stefan«, Jamova 39, Ljubljana
[3] Ljubljanska banka d.d., Šmartinska 132, Ljubljana

Abstract: A COMPARATIVE ANALYSIS OF THREE TOOLS FOR THE CONSTRUCTION AND USAGE OF KNOWLEDGE BASES IN EXPERT SYSTEMS. In the paper a comparative analysis of three expert systems shells: Dex, Optrans and Assistant Professional is given. Dex is a specialised expert system shell for preference knowledge modelling in multiattribute decision making. Optrans is a general purpose expert system shell that offers some additional modules, which make it particularly convenient for financial modelling. Assistant Professional is an expert system shell based on automatic learning from given examples. These tools were used for the estimation of credit worthiness in a banking environment. The comparison deals with knowledge acquisition, representation and usage for credit worthiness evaluation.

The goal of the paper is to illustrate and discuss advantages and disadvantages of the tools. On this basis, a complementary use of the tools can be employed in order to improve the effectiveness of the process of knowledge base construction, usage and maintenance.

Povzetek: V referatu je podana primerjava uporabe treh lupin ekspertnih sistemov: Dex, Optrans in Assistant Professional. Dex je specializirana lupina za modeliranje preferenčnega znanja v okviru večparametrskega odločanja. Optrans je splošna lupina ekspertnega sistema, ki je skupaj z dodatki še posebej primerna za finančno modeliranje. Assistant Professional pa je lupina ekspertnega sistema, ki temelji na zajemanju znanja z učnimi primeri ob uporabi metod avtomatskega učenja. Vsa tri orodja so bila uporabljena za ugotavljanje bonitete poslovnega partnerja banke. Primerjava se nanaša na zajemanje znanja in njegovo predstavitev v bazi posameznega sistema in na uporabo tega znanja pri ugotavljanju bonitete partnerja.

Namen tega prispevka je prikazati dobre in slabe strani posameznih orodij. Na tej osnovi lahko načrtujemo njihovo komplementarno uporabo, ki omogoča učinkovitejši proces izgradnje, uporabe in vzdrževanja ekspertnega sistema.

1 Uvod

Na trgu so dosegljiva različna orodja za izgradnjo in uporabo baz znanja ekspertnih sistemov. Poleg vprašanja, katero orodje uporabiti v dani situaciji, se nam zastavlja tudi vprašanje o morebitni smiselnosti uporabe več orodij. To lahko razumemo tudi kot iskanje rešitev za učinkovitejše zajemanje znanja v pogledu razreševanja Feigenbaumovega ozkega grla (Feigenbaum 1977). V splošnem nas zanima možnost doseganja sinergije z uporabo komplementarnih pristopov, ki jih nudijo različna orodja. Na primer, gradnja baz znanja po več različnih poteh nas lahko pripelje do boljših baz znanja v pogledu količine znanja, pa tudi njegove razumljivosti in preverljivosti. Splet uporabe več orodij odpira tudi nekatere možnosti dograjevanja, preverjanja in različnih artikulacij znanja, kar je še posebej dobrodošlo v procesu vzdrževanja baz znanja ekspertnih sistemov.

Da bi v diskusiji in zaključku tega prispevka lahko odgovorili na nekatera izmed teh vprašanj, bomo v nadaljevanju predstavili problem izgradnje in uporabe baze znanja za bančni ekspertni sistem "Boniteta", ki je bil razvit v Ljubljanski banki (Delidžakova-Drenik et al. 1989). Pri tem so uporabljene tri različne lupine ekspertnih sistemov: Optrans, Dex in Assistant Professional. Vsa ta orodja tečejo na osebnih računalnikih tipa IBM-PC in so komercialno dosegljiva.

2 Primer izgradnje in uporabe baze znanja

Primer, ki ga obravnavamo z navedenimi orodji, se nanaša na boniteto v bančnem smislu. Z boniteto je mišljena ocena poslovnega partnerja, da v roku vrne odobreni kredit in plača obresti. Čim večja je boniteta poslovnega partnerja, tem manjši je riziko poslovanja banke. Zato je ocenjevanje bonitete partnerja, ki mu dajemo kredit, pomembna mera za zmanjšanje rizičnosti poslovanja in zagotavljanje likvidnosti banke (Delidžakova-Drenik, Mavec 1989).

Pri delu z bazo znanja so bila uporabljena vsa tri orodja. V posameznih fazah uporabe so se pokazale različne prednosti in slabosti, ki jih ilustriramo v tem razdelku.

2.1 Dex

Dex je specializirana lupina ekspertnega sistema za večparametrsko odločanje, ki je bila razvita na Institutu "Jožef Stefan" v Ljubljani (Bohanec, Rajkovič 1991, DEX 1989). Odločitveni prostor znanja predstavljajo parametri, kazalci, atributi oz. kriteriji, ki so drevesno organizirani. Povezava podrednih parametrov v nadredne je osnovana na preferenčnem znanju, ki je izraženo s preprostimi produkcijskimi pravili (Bohanec et al. 1988, Rajkovič, Bohanec 1990).

2.1.1 Baza znanja

Bazo znanja predstavlja drevo parametrov in pravila združevanja ocen posameznih parametrov v ocene višje ležečih parametrov v drevesu. Za primer bonitete je drevo parametrov predstavljeno na sliki 2. Poleg parametrov so tudi njihovi opisi in zaloge vrednosti, ki jih lahko posamezni parametri zavzamejo. Slika 1 prikazuje tabelo pravil povezovanja treh parametrov: REV_DOH, PROF_ST in ST_RENT v agregirano oceno DOHODEK. Kako čitamo tako tabelo pravil? Prva tri pravila povedo, da je dohodek ocenjen **slabo**, če je vsaj eden izmed izhodiščnih treh parametrov na spodji meji, tj. zavzema vrednost **slabo**. Zvezdica (*) pomeni katerokoli vrednost. Četrto pravilo pove, da je dohodek *srednji*, če je revalorizacijski dohodek ocenjen z dobro, profitna stopnja s *srednje* in stopnja rentabilnosti z oceno, ki je večja ali enaka oceni *srednje*. V našem primeru imamo 7 takih tabel, za vsak nadredni parameter po eno.

	REV_DOH	PROF_ST	ST_RENT	DOHODEK
1.	slabo	*	*	slabo
2.	*	slabo	*	slabo
3.	*	*	slabo	slabo
4.	dobro	srednje	>=srednje	srednje
5.	dobro	>=srednje	srednje	srednje
6.	dobro	dobro	dobro	dobro

Sl.1: Tabela pravil za ugotavljanje parametra DOHODEK v sistemu DEX

Parameter	Opis parametra	Vrednosti parametra
BONITETA	ocena bonitete poslovnega partnerja	slabo, srednje, dobro
├─RIZIČNOST	rizičnost naložb	visoka, srednja, nizka
│ ├─ST_LIK_ZAD	prejeti LK / sredstva	slabo, dobro
│ ├─KAZ_14	dvomljive terjatve / rizične naložbe	slabo, dobro
│ ├─KAZ_13	sklad solid.odg. / rizične naložbe	slabo, dobro
│ ├─IZ_KKP	izkoriščenost kratk.kred.potenciala	slabo, dobro
│ └─STR_PLAS	struktura plasmajev	slabo, dobro
├─LIKVIDNOST	likvidnost poslovnega partnerja	slabo, srednje, dobro
│ ├─LIK_A		slabo, srednje, dobro
│ │ ├─ST_LIK_OŽ	stopnja ožje likvidnosti	slabo, dobro
│ │ ├─ST_LIK_ŠI	stopnja širše likvidnosti	slabo, dobro
│ │ └─KAZ_11	medbančno kreditiranje	slabo, dobro
│ ├─LIK_B		slabo, srednje, dobro
│ │ ├─OR_DNI	št.dni uporabe obvezne rezerve	slabo, srednje, dobro
│ │ ├─RS_DNI	št.dni uporabe rezervnega sklada	slabo, dobro
│ │ └─UP_SR_NB	uporaba sredstev pri NB	slabo, dobro
│ └─LIK_C		slabo, srednje, dobro
│ ├─DOS_PL	doseganje posebne likvidnosti	slabo, dobro
│ ├─DOS_ML	doseganje minimalne likvidnosti	slabo, srednje, dobro
│ └─ST_LIK_NB	stopnja likvidnosti pri NB	slabo, dobro
└─DOHODEK	dohodkovna uspešnost posl.partn.	slabo, srednje, dobro
├─REV_DOH	revalorizacijski dohodek	slabo, dobro
├─PROF_ST	profitna stopnja	slabo, srednje, dobro
└─ST_RENT	stopnja rentabilnosti	slabo, srednje, dobro

Sl.2: Drevo parametrov ocenjevanja bonitete v sistemu DEX

Z Dexom smo sorazmerno enostavno kreirali bazo znanja. Čeprav mora drevo parametrov zgraditi ekspert oz. ekspertna skupina, kar je vsekakor zahteven miselni proces, pa je s tem podan celovit pogled na domeno znanja. V drugem koraku eksperti v dialogu z računalnikom artikulirajo pravila, ki so relativno enostavna (podproblemi). Njihovo konsistentnost in pokritost problemskega prostora pa zahvaljujoč strukturi parametrov lahko avtomatsko nadziramo in vodimo v dialogu, ki ga omogoča DEX.

2.1.2 Ocenjevanje bonitete

Za vsakega poslovnega partnerja moramo zbrati podatke po osnovnih parametrih. Primer za enega partnerja, ki ga želimo oceniti, kaže slika 3. V Dexu smo namesto numeričnih kvantitativnih kazalcev uvedli opisne vrednosti, ki jih dobimo s pravili pretvarjanja, kot na primer za stopnjo rentabilnosti (ST_RENT):

numerični podinterval	*opisna vrednost*
1. slabo	≤ 0 %
2. srednje	> 0 % in < 2 %
3. dobro	≥ 2 %

Parameter	številska vrednost	opisna vrednost
ST_LIK_ZAD	11.81	dobro
KAZ_14	0	dobro
KAZ_13	2.1	slabo
IZ_KKP	33.08	dobro
STR_PLAS	110	dobro
ST_LIK_OŽ	726.26	dobro
ST_LIK_ŠI	289.37	dobro
KAZ_11	166851	dobro
OR_DNI	3	srednje
RS_DNI	5	dobro
UP_SR_NB	0	dobro
DOS_PL	3072.62	dobro
DOS_ML	382.47	dobro
ST_LIK_NB	5.9	dobro
REV_DOH	8507	dobro
PROF_ST	10.84	dobro
ST_RENT	0.27	srednje

Sl.3: Primer podatkov poslovnega partnerja, ki ga želimo oceniti

Ko podatke o varianti vstavimo v računalnik, dobimo s pomočjo pravil ocene po vseh parametrih v drevesu, vključno s končno oceno. Tak izpis podaja slika 4. Možna pa je tudi agregirana razlaga, kjer so izpisane le vrednosti tistih parametrov, ki so bistveno vplivali na končno oceno.

```
BONITETA          dobro
├─RIZIČNOST       srednja
│  ├─ST_LIK_ZAD   dobro
│  ├─KAZ_14       dobro
│  ├─KAZ_13       slabo
│  ├─IZ_KKP       dobro
│  └─STR_PLAS     dobro
├─LIKVIDNOST      dobro
│  ├─LIK_A        dobro
│  │  ├─ST_LIK_OŽ dobro
│  │  ├─ST_LIK_ŠI dobro
│  │  └─KAZ_11    dobro
│  ├─LIK_B        dobro
│  │  ├─OR_DNI    srednje
│  │  ├─RS_DNI    dobro
│  │  └─UP_SR_NB  dobro
│  └─LIK_C        dobro
│     ├─DOS_PL    dobro
│     ├─DOS_ML    dobro
│     └─ST_LIK_NB dobro
└─DOHODEK         srednje
   ├─REV_DOH      dobro
   ├─PROF_ST      dobro
   └─ST_RENT      srednje
```

Sl.4: Ocena bonitete partnerja v sistemu DEX

2.2 Optrans

To je integrirano programsko orodje za razvoj ekspertnih sistemov, ki ga je razvila firma "Systems informatiques de gestion" iz Francije (PC-OPTRANS EXPERT 1988). Poleg same lupine ekspertnega sistema Optrans vsebuje tudi module za delo s podatki in modeli. V osnovi je namenjen reševanju poslovnih problemov s posebnim poudarkom na finančnem modeliranju (Klein, Methlie 1990).

2.2.1 Baza znanja

Znanje je predstavljeno s produkcijskimi pravili tipa ČE-POTEM. V bazi ekspertnega sistema "Boniteta" za potrebe ugotavljanja bonitete bančnega partnerja je 117 pravil. Primer teh pravil kaže slika 5. To so pravila za ugotavljanje dohodkovne uspešnosti na osnovi revalorizacijskega dohodka, profitne stopnje in stopnje rentabilnosti. Pravila smo dobili v razgovoru z bančnimi eksperti, tako da baza znanja predstavlja artikulirano znanje ekspertov. Težave so bile pri zagotavljanju celovitosti znanja in konsistentnosti pravil.

```
IF revalor_dohodek ≤ 0 AOR profitna_stopnja ≤ 0
AOR stopnja_rentab ≤ 0
THEN FACTS_DEDUCED dohodek IS slabo
CRITERIA_TO_EXAMINE stop
MESSAGE Dohodek banke je slab.
FINISH_RULE

IF revalor_dohodek > 0 AND profitna_stopnja ≥ 0.4
AND stopnja_rentab ≥ 2
THEN FACTS_DEDUCED dohodek IS dobro
CRITERIA_TO_EXAMINE stop
MESSAGE Dohodek je dober.
FINISH_RULE

IF revalor_dohodek > 0 AND profitna_stopnja ≥ 0.4
AND stopnja_rentab < 2 AND stopnja_rentab > 0
THEN FACTS_DEDUCED dohodek IS srednje
CRITERIA_TO_EXAMINE stop
MESSAGE Dohodek je srednji.
FINISH_RULE
```

Sl.5: Nekaj pravil za oceno dohodkovne uspešnosti v sistemu Optrans

```
** * Ocena likvidnosti ** *
Odstotek doseganja posebne likvidnosti presega predpisano
mejo 100 %. Odstotek doseganja minimalne likvidnosti je
nad 200 %. C_likv je dobra. A_likv je dobra.
Likvidnost je dobra.

** * Ocena rizičnosti ** *
Sklad solidarne odgovornosti ne dosega 3 % rizičnih naložb.
Kaz_13 znaša 2.10 %. Izkoriščenost KKP znaša 33.08 %
in je v mejah predpisane vrednosti 40.00 %. Struktura
plasmajev (110.00 %) zadošča predpisani vrednosti 85 %.
Dvomljive terjatve znašajo manj kot 5 % rizičnih naložb.
Kaz_14 znaša 0.00 %.
Riziènost je srednja.

** * Ocena dohodkovne uspešnosti ** *
Banka dosega revalorizacijski dohodek. Profitna stopnja
v višini 10.84 % je odlična. Stopnja rentabilnosti v višini
0.27 % je srednja.
Dohodek je srednji.
```

Sl.6: Ocena bonitete z Optransom

2.2.2 Ocenjevanje bonitete

Za poslovnega partnerja, ki ga želimo oceniti, moramo imeti podatke kot je primer na sliki 3. Ekspertni sistem pa nam postreže z oceno bonitete, ki jo pripiše temu partnerju. Izpis je na sliki 6. Kot vidimo, je boniteta ocenjena z oceno *dobro* izmed treh možnih ocen (slaba, srednja, dobra), ki izvira iz delnih ocen likvidnosti, rizičnosti in dohodkovne uspešnosti partnerja. Poleg tega je možno dobiti tudi izpis vseh pravil, ki so bila uporabljena v tem postopku ocenjevanja.

2.3 Assistant Professional

Ta lupina ekspertnega sistema omogoča avtomatsko generiranje baze znanja na osnovi učnih primerov. Razvita je bila v sodelovanju med Fakulteto za elektrotehniko in računalništvo v Ljubljani in Institutom "Jožef Stefan". Posebej primerna je povsod tam, kjer *ekspertizo* posedujemo v obliki ustrezno rešenih primerov kake ekspertne domene (Cestnik et al. 1987). Osnovni algoritem je privzet iz sistema ID3 (Quinlan 1987) in dopolnjen z mnogimi izboljšavami, predvsem v smislu obvladovanja nezanesljivih in nepopolnih podatkov.

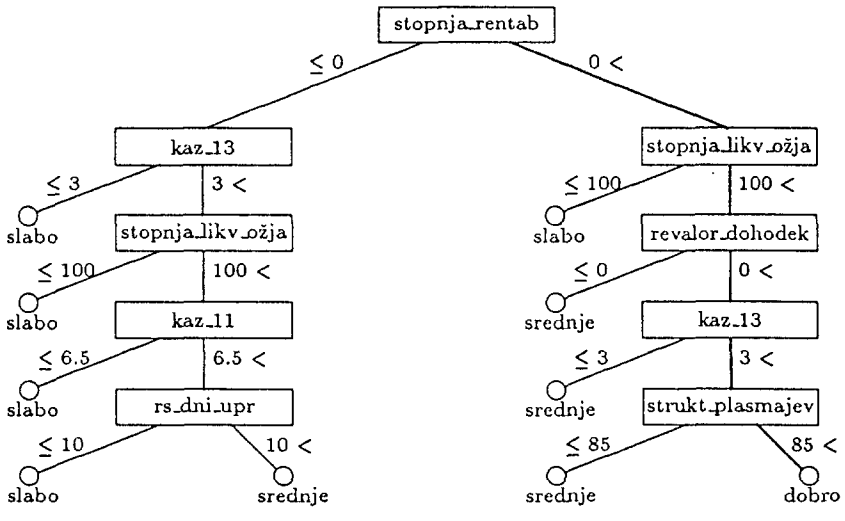
2.3.1 Baza znanja

Kot smo že omenili, potrebujemo za izgradnjo baze znanja učne primere, to je ocene konkretnih bonitet partnerjev, ki smo jih zbrali pri svojem delu v preteklosti. Primere opišemo s standardnimi kazalci in pripišemo boniteto, ki se je v praksi potrdila. Assistant Professional generira odločitveno drevo, ki ga v obravnavanem primeru kaže slika 7.

Čeprav je nabor kazalcev enak kot pri prejšnjih dveh lupinah, je struktura zapisa znanja v odločitvenem drevesu drugačna. Eden izmed ciljev gradnje drevesa v sistemu Assistant Professional je namreč kar najbolj kompaktno predstaviti znanje, ki je kodirano v učnih primerih.

2.3.2 Uporaba baze znanja

Ko želimo s tem sistemom oceniti boniteto kakega partnerja, vnesemo njegove podatke in sistem v skladu z odločitvenim drevesom oceni boniteto. Tako ocenjevanje je še posebej primerno, če nam vrednosti nekaterih parametrov manjkajo. Na sliki 8 je prikaz ekrana po vnosu le štirih (vendar bistvenih) vrednosti kazalcev. Na osnovi teh vrednosti sistem ugotavlja, da je z verjetnostjo blizu 94% boniteta partnerja *dobra* in le s približno 6% *srednja*.



Sl. 7: Odločitveno drevo za ocenjevanje bonitete sistema Assistant Professional

Tree only

ATTRIBUTE	VALUE	Select value for izkoriščenost KKP	CLASS	PROB.
odstot doseg PL	3072.620	33.080	dobro	93.878
izkoriščenost KKP	33.080	4	srednje	6.122
stopnja rentab	0.270		slabo	0.000
revalor dohodek	8507.000			
opr sred NB	unknown			
stopnja likv NB	unknown			
odstot doseg ML	unknown			
:	:			
:	:			

Sl.8: Izračun verjetnostne porazdelitve bonitetnega razreda ob le štirih znanih podatkih v sistemu Assistant Professional

3 Diskusija

Prikazani sistemi se pomembno razlikujejo tako v zajemanju znanja, kot tudi v načinu uporabe. Dex in Optrans zahtevata eksplicitno artikulacijo znanja. V Optransu le-ta poteka podobno kot pri običajnih lupinah ekspertnih sistemov, znanje formuliramo z množico produkcijskih pravil. V Dexu poteka artikulacija znanja v dveh korakih. Najprej identificiramo drevo parametrov, zatem pa še pravila povezovanja, tj: medsebojne vplivnosti parametrov na višje ležeče (agregirane) parametre v drevesu. Kot smo že omenili, ta način zajemanja znanja olajša zagotavljanje celovitosti problemskega prostora. Postopnost in artikulacija vsebinsko zaokroženih podmnožic pravil (v našem primeru sedmih tabel pravil) pa razbremeni človeka, saj se le-ta ukvarja s praviloma enostavnejšimi podproblemi. Oboje ima za posledico enostavnejši in hitrejši proces zajemanja znanja. Assistant namesto eksplicitno artikuliranega znanja potrebuje učne primere in nato na njihovi osnovi zgradi bazo znanja v obliki odločitvenega drevesa. Njegova uporaba je posebej priporočljiva takrat, ko ustrezni učni primeri obstajajo, na primer kot dokumentacija dodeljevanja kreditov v preteklosti.

V pogledu uporabe baze znanja in razlage ocen bonitete nam Optrans omogoča zelo prijazno komunikacijo po zaslugi razlage (komentarjev) v domačem jeziku. Dex ima praviloma bolj zgoščen izpis rezultatov, ki je primernejši za medsebojno primerjavo več ocen.

Pomembna prednost Assistanta je enostavna in učinkovita uporaba sistema ob nepopolnih ali nezanesljivih podatkih.

Če na koncu povzamemo, lahko rečemo, da je smiselno pričeti z artikulacijo znanja z Dexom, nato pa bazo znanja preliti v Optrans in ga operativno uporabljati. Poleg razkošnih opisov rezultatov in razlag je pomembna tudi uporaba Optransovih modulov za pripravo in izračun konkretnih numeričnih podatkov. Če pa imamo na razpolago primere konkretnih ocen, npr. v kakem preteklem obdobju poslovanja banke s svojimi partnerji, uporabimo Assistanta. S tem po drugi poti pridemo do baze znanja, kar nam omogoča primerjalno analizo z artikuliranimi bazami znanja, npr. v Dexu ali Optransu.

4 Zaključki

Pokazali smo pomembne prednosti posameznih orodij v različnih fazah izgradnje in delovanja ekspertnega sistema za določanje bonitete. S kombinirano rabo opisanih treh lupin ekspertnega sistema hitreje in enostavneje pridemo do baze znanja, jo preverjamo in dopolnjujemo. Prilagajamo pa se tudi različnim zahtevam uporabe od poglobljene analize posamezne ocene do ocenjevanja v primeru, ko nekateri podatki manjkajo. Povedano nas navaja na misel o izgradnji novih prilagodljivih integriranih orodij kot tudi na uporabo obstoječih orodij, ki jih mora sam uporabnik kombinirati in prilagajati svojim potrebam.

Zahvala

Za nasvete in pripombe pri našem delu se želimo zahvaliti avtorjem uporabljenih lupin ekspertnih sistemov dr. Marku Bohancu, mag. Bojanu Cestniku in prof.dr. Michelu Kleinu. Zahvalo smo dolžni tudi diplomantkama Fakultete za organizacijske vede v Kranju Darji Habicht in Jani Seražin, ki sta v okviru svojih diplomskih del izvedli nekatere eksperimente, ki so prispevali k primerjalni analizi uporabljenih orodij.

Literatura

Bohanec, M., Gyergyek, L., Rajkovič, V., Večparametrsko odločanje, podprto z lupino ekspertnega sistema, Elektrotehniški vestnik, vol. 55., št. 3–4 (Ljubljana, 1988), 189–198.

Bohanec, M., Rajkovič, V., DEX : An Expert System Shell for Decision Support, Sistemica, Vol.1, Nr.1, 1990, pp. 145–158.

Cestnik, B., Kononenko, I., Bratko, I., "ASSISTANT 86": A knowledge elicitation tool for sophisticated users, Sigma Press, 1987.

Delidžakova-Drenik E., Mavec, A., Analysis of a knowledge-based system in use, Eurobanking, Heemskerk, The Netherlands, 1989.

Delidžakova-Drenik E., Stariha, M., Mavec, A., Razvoj i uvodjenje ekspertnog sistema za odredjivanje boniteta banaka, Zbornik radova SYM-SP-IS, 1989.

DEX - An expert system shell for multi-attribute decision-making, User's Manual, "J.Stefan" Institute, Ljubljana, 1989.

Feigenbaum, E.A., The art of artificial intelligence: Themes and case studies of knowledge engineering, Pub.no. STAN-CS-77-621, Stanford University, Dept. of Computer Science, 1977.

Klein, M., Methlie, B.L., Expert systems: A decision support approach - With applications in management and finance, Addison-Wesley, 1990.

Kononenko, I., Cestnik, B., Bratko, I., ASSISTANT PROFESSIONAL User's Guide, Institut "Jožef Stefan", Ljubljana, 1988.

PC-OPTRANS EXPERT, Manual de l'utilisateur, Systemes informatiques de gestions, Jouy-en-Josas, 1988.

Quinlan, J.R., Generating rules from decision trees, IJCAI, 1987.

Rajkovič, V., Bohanec, M., Decision Support by Knowledge Explanation, in Environments for supporting decision processes (Sol, G.H., Vecsenyi, J., - eds.), North-Holland, 1991, pp. 47–57.

Keywords: formalization, formation, formula as a thing, information, metaphysics, phenomenon, thing for others, thing in itself, thing itself.

Anton P. Železnikar
Volaričeva ulica 8
61111 Ljubljana

Ta spis se ukvarja s vprašanji formalizacije pojava, ki ga imenujemo informiranje stvari. Kako stvar informira in kako je mogoče načine informiranja formalno razložiti? Formula (7) opisuje informiranje stvari same, formula (9) in njeno razgrajevanje informiranje stvari v sebi (metafiziko stvari) in formula (23) informiranje stvari za druge. Pri tem je mogoče zaznati tudi formalno razliko med informiranjem stvari kot informacije in informiranjem stvari kot podatka, in sicer v obliki informiranja podatka samega s formulo (41), podatkovne metafizike (42) in informiranju podatka za druge (43). Okvirno je opisan primer informiranja matematične formule. S tem se uvaja informacijski jezik, ki opisuje fenomenalnost stvari, tj. njeno pojavnost kot realnost, metafiziko in informacijo za druge.

Informing of Things II

This essay deals with the formalization of phenomenon called informing of things. How does a thing inform and how it is possible to explain formally several modes of informing? Formula (7) describes the informing of thing itself, formula (9) and its decomposition the informing of thing in itself (metaphysics of thing), and formula (23) the informing of thing for others. Within this philosophy it is also possible to distinguish the formal difference occurring between the informing of thing as information and informing of thing as data, so, it is possible to determine the informing of data itself by formula (41), data metaphysics (42), and the informing of data for others (43). Informing of a mathematical formula is described in a general way. This essay introduces an informational language for description of phenomenality of things, that is, for a thing's phenomenality as reality, metaphysics, and information for others.

Stvar in informacija

Osnovno vprašanje v okviru informacijske filozofije ostaja prej ko slej, kaj je informacija in kako informira. V okviru formalne teorije informacije, kjer je cilj konstruiranje formul in sistemov formul za opisovanje pojavov, dogodij, scenarijev, procesov, oblik stvari in njihovih stvarnosti, se nujno pojavi vprašanje posebnega jezika, ki je logika stvari v območju stvarnega informiranja. To disciplino bi lahko poimenovali ustrezno s posebnim imenom, ki bi bilo simbol za tovrstno informacijsko logiko, npr. kot *informologija*. Informologija bi bila naravi informacije značilen in dovolj kompleksen jezik za opisovanje informacijskih entitet, ki so lahko fenomenalne (fenomenologija) tudi v eidetičnem, kognitivnem, teleološkem smislu.

V prvem poskusu premisleka o informaciji se ponuja pojem »formacija«, ki bi lahko bil splošnejši od pojma »informacija«. Če je vsaka stvar neka formacija, potem bi bila formacija stvari izvor in mesto informacije. Pri tem bi formacijo kot stvar oziroma stvar kot formacijo pojmovali dovolj široko, kot npr. poljubno entiteto, pri kateri ugotavljamo njeno strukturo in organizacijo, njeno pozicijo in atitudo v odnosu do okolja in same sebe in s tem njeno »pojavljanje« (javljanje, nagovarjanje oziroma pojmovne izpeljanke iz *grškega* glagola φαι νω). Formacija, ki je kot formacija opazljiva, zaznavna, ki postaja ali je predmet razumevanja, ni le oblikovna stvar v ožjem pomenu, temveč je prav tako procesna, posamična, sistemska oziroma fenomenalna. Z besedo »fenomenalna« mislimo formacijo pojavljanja, in sicer tako, kot se formacija pojavlja kot čista realnost (formacija sama, formacija na sebi), kot formacija v sami sebi (navzven zakrita ali nikoli do konca odkrita formacija, tj. kot metafizika formacije) in nazadnje kot formacija za druge (od zunaj, z drugimi formacijami opazovana, zaznana formacija).

Kakšne so razlike med materialno in duhovno, oblikovno in procesno, strukturno in organizacijsko formacijo? Pri materialni formaciji se lahko postavimo na fizikalno (fundamentalistično naravno, naturalistično, naravoslovno) stališče, da imamo opravka s fizikalno (molekularno, atomsko, kvantno, sevalno, valovno itd.) formacijo, v okviru katere je mogoča vsa znana in neznana fizikalna fenomenalnost (nastajanje raznovrstnih, možnih materialnih oblik in procesov v formaciji, zunanjih vplivov na formacijo, skratka vseh mogočih fizikalnih fenomenov, povezanih s formacijo). Materialna formacija skupaj s procesi v njej je podlaga duhovni formaciji; tu mislimo npr. možgane ali splošneje osrednji živčni sistem kot materialno formacijo, v kateri nastaja fenomenalnost (pojavnost, procesnost, dogodje) duhovne formacije (uma, razuma, zavesti, nezavesti, njihovih dogodij). Primer duhovne formacije, ki jo lahko opazujemo, je npr. zavest ali zavedanje določene informacije in v okviru zavesti (spontano, cirkularno) pojavljajoče opredmetene zavestne in nezavestne formacije.

Duhovna formacija živega človeka seveda ni zgolj »izmišljotina« (npr. čista fikcija, imaginacija). Pri današnji stopnji znanstvene zavesti je mentacija živega bitja sistemski fenomen nevronske mreže, tj. fiziološko zapletene in dinamično (procesno in strukturno) nastajajoče formacije nevronov in njihovih fizičnih in procesnih povezav in pogojenosti. Duhovnost je procesnost, pojavnost, materialno gibanje, spontana in cirkularna povezava miselnih formacij v okviru nevronske formacije. Duhovnost postaja skozi zavestno pojavnost razvidna, kot formacija zavestne formacije razločljiva, predmet razumevanja s posebnim pomenom. Nastajajoča duhovnost v nevronskih mrežah cerebralnih korteksov (v možganih) je oblikovno in procesno nastajajoča materialna (fizikalna, naravna) formacija formacij, formacijskih fenomenalitet (pojavnosti, fenomenalnosti). S tem nismo o duhovnosti povedali ničesar presenetljivega (novega), le osvetlili smo mentacijo z določenega, dovolj splošnega, fenomenološko formativnega vidika, ki ima svoj temelj v materialističnem in idealističnem razumevanju duhovnosti, tj. v posebni miselni formaciji razumevanja kot fenomena.

Oblikovna in procesna fenomenalnost sta le navidezno povsem razločljivi, dejansko pa gre za njuno fenomenalno povezanost. Oblika stvari je

njena zunanjost, razpoznavna notranja zgradba, vendar tudi druga procesna pojavnost v okviru strukture in organizacije stvari. Za t.i. obliko se vselej skriva fizikalna fenomenalnost stvari, ki je lahko razpoznana do določene strukturno-procesne stopnje. Tradicionalno umevanje strukture je le pogoj za nastajanje razumevanja določenih fizikalnih procesov v tej strukturi. Pri formaciji oblike kot stvari mislimo še vso njeno predstavljivo pojavnost, prepleteno in nastajajočo procesnost v znanem in neznanem pomenu.

Strukturna in organizacijska formacija stvari sta sicer lahko razločljivi, vendar sta v fenomenu stvari (krožno in paralelno) soodvisno povezani, ko ena drugo pogojujeta, vzajemno nastajata in se odvisnostno spreminjata. Organizacija je procesni videz strukture, struktura v spreminjanju in nastajanju, gibanje in utripanje strukture kot oblikovnega in procesno pojavljajočega fizikalnega sistema stvari. Organizacija je akcija in potencial strukture za strukturne spremembe, je opazljivo dogodje stvari, njena strukturna relacijskost, prepletenost, pogojnost in posledičnost.

Če vse zapisano velja za stvar kot oblikovno-procesno formacijo, potem je vprašanje, kako stvar kot formacija informira in kako je informirana. Nahajamo se pred miselnim obratom iz pojmovanja formacije, stvari kot formacije, v pojmovanje formacije kot »in—formacije«, ki zapira pojmovanje formacije v formacijo. Stvar lahko informira le kot formacija, saj ne razumemo (se ne zavedamo, priznavamo) predformacijskega (še ne formacijskega) ali postformacijskega (nadformacijskega, onkrajformacijskega) stanja stvari. Vsakokrat, ko izrečemo besedo »informacija«, imamo v mislih pojavljanje konkretne stvari ali, kot rečemo, entitete (filozofsko tako ali drugače bivajočega, tubitnostnega). Informacija tedaj pojavno zastopa (označuje, predstavlja) določeno stvar. In če se postavlja izvorno vprašanje, kaj je informacija, potem je lahko splošen odgovor le eden: »Informacija je sinonim za pojavljajočo stvar, za njeno formacijo.« Zaradi takega stališča lahko v okviru jezikovnega razumevanja namesto o »informiranju informacije« govorimo o »informiranju stvari« ali filozofsko še naprej o bistvu, bivanju, biti, naravi stvari. Informacija je tedaj preprosto stvar v njeni pojavni oblikovnosti in procesnosti, v fenomenalnosti njene stvarnosti. *Latinska* beseda res (rei, f.) ima pomene stvar, zadeva, dogodek, v množini pa še svet, vesolje, narava, pa še pogoj, položaj, okoliščine, odnos itd. Stvarnost je pojavljanje stvari oziroma tudi realnost pojavljajočega.

Formalna zasnova informiranja stvari

Razprava z vidika informiranja stvari nudi nekatere spoznavne (razumevne) olajšave v primerjavi z razpravo o informiranju informacije. Kdorkoli se namreč lahko kritično vprašuje, kaj sploh pomeni informacija kot stvar in kako se kot posebna stvar lahko razumeva. Informacija se namreč vobče (običajno) ne pojmuje kot predmet, stvar, entiteta temveč veliko bolj kot apriorni, megleno predočljiv pojav sporočanja. Stvar kot stvar pa je lahko predmet oziroma objekt, o katerem je vsaj načelno možna tudi filozofsko-formalna ali celo matematično-teorijska kontemplacija (npr. stvar kot formaliziran matematični objekt, ki pripada množici stvari-objektov).

V tem poglavju želimo pojasniti tri različne vidike informiranja stvari, ki niso pomembni le za informatiko in njeno tehnologijo ampak tudi za filozofijo in matematiko. Kolikor mogoče enostavno in bistveno želimo raziskati informiranje stvari, in sicer kot **informiranje stvari same,** kot **njeno metafiziko** in kot **informiranje stvari za druge.** Bralec sam naj razsodi, kako formalni koncepti lahko povratno-bistveno vplivajo na razumevanje fenomena stvari v okviru filozofije same.

Pristop k razločevanju (razpiranju) narave stvari same ali k razloki (dosežku razločevanja) te narave (*franc.* différance s črko a in ne s črko e) bo kot vselej na začetkih določenih filozofij in teorij aksiomatičen oziroma definitoren. Za avtorjev okus bi bila kot definitorna informacija primernejša mehkejša implikacija, kar pa bi lahko povzročilo tradicionalne formalne zaplete. Implikacija vobče ne bo primordialno aksiomatsko izhodišče konstruiranja formalnih začetkov (aksiomov), temveč bo ta (v določenem pomenu

definitorna) vloga podeljena t.i. definicijski ekvivalenci (znak oziroma informacijski operator $\Leftrightarrow_{Df}$), ki je hkrati dovoljenje za neomejeno (spontano) formalno substitucijo v okviru konkretne ekvivalence. Definicijska ekvivalenca bo partikularna oblika informacijske ekvivalence $\Leftrightarrow$, katero bo mogoče podobno kot splošni informacijski operator $\models$ ekvivalenčno partikularizirati. To seveda ne pomeni, da ne bomo informacijske implikacije uporabljali pri oblikovanju (konstruiranju) smisla veljavnih definicijskih ekvivalenc.

V formulah bomo stvari označevali z malimi grškimi črkami α, β, ... , ω. Ti simboli bodo v formulah označevalci, ki kažejo na konkretne stvari, tj. na označence kot poljubno oblikovno in procesno kompleksne formacijske entitete. To kratkomalo pomeni, da je mogoče označevalce dekonstruirati (dekomponirati, razstavljati) v formule z več označevalci, ki kažejo na svoje označence. Sam označevalec bo formalna entiteta, tj. formula dejanskega in potencialnega označevanja označenca v procesu njegovega nastajanja. Ta koncept bo, kot bomo videli, izražen z osnovnim aksiomom informiranja stvari same.

Kaj bo formula (informacijska formula)? Formula bo entiteta, ki bo na določen način označevala neko stvar kot formacijo, in sicer kot informiranje te (vobče sestavljene ali nikoli dokončno razstavljive) formacije. V formuli se bodo sicer pojavljali še znaki operatorjev in ločilni znaki (oklepaji, vejice, podpičja), vendar bodo to le pomožni znaki (pomožni, ločevalni operatorji) oziroma natančneje deli označevalcev samostojnih (razločenih) formacij v formuli. Kot bomo videli, operator (informacijski operator $\models$) ne bo samostojna entiteta, temveč bo pripadal v operaciji udeleženim operandom (označevalcem oziroma njihovim označencem). Pokazali bomo, kako bo tudi posamezen operand (označevalec stvari same) formula v pravkar opisanem smislu.

Informiranje stvari same

Stvar kot formacija v prostoru in času je vobče gibljiva, nastajajoča in spreminjajoča entiteta. Zamišljamo si jo kot predmet (tudi abstraktni, duhovni predmet), ki se pojavlja. Pojavlja se tako, da ima svojo lastno (fizikalno, materialno, energijsko, valovno itd.) fenomenalnost, ki pa je pod vplivom fenomenalnosti drugih stvari in s svojo fenomenalnostjo vpliva tudi na fenomenalnosti drugih stvari. Naštete »lastnosti« stvari moramo strniti v kar se da kompaktno formulo. Pri tem pomislimo, da je stvar v prostoru in času (ta fizikalni vidik je izbran v skladu s spoznavno tradicijo) na razpolago svoji in tuji pojavnosti, skratka da biva v prostoru in času dogajanja drugih stvari. Beseda fenomenalnost skriva t.i. energijski princip, ki je določena aktivnost in pasivnost, procesnost in oblikovnost stvari. Do tu se še vedno gibljemo v okviru fizikalno možnega (fizikalno discipliniranega) razumevanja stvari.

Pri graditvi začetnega (izhodiščnega, aksiomatskega) sistema stvari kot formacije, ki informira, uvedimo najprej dve izhodiščni formuli, in sicer z utemeljitvijo (predpostavko, hipotezo), da stvar informira in je informirana. »Stvar informira« je povsem odprta formula, ki ne opredeljuje objektov, tj. stvari, ki so s stvarjo informirane (vplivane z dogodjem stvari same). Naj bo z α označena neka stvar. Vendar formula »stvar informira« govori, da stvar informira. Za glagol »informirati« uvedimo operatorski znak $\models$, ki označuje (vse možne) lastnosti informiranja stvari α v najsplošnejši obliki. Vsakokrat, ko izrečemo besedo »stvar«, predpostavljamo (impliciramo, hipotetiziramo, predvidevamo), da »stvar informira«. Pri tem ne izrekamo ničesar o tem, katere stvari in kako informira; torej ohranjamo popolno odprtost formule. Za »stvar α informira« zapišemo simbolično

(1) $\alpha \models$

Desna stran operatorja $\models$ je prazna, kar simbolizira odprtost zapisane formule (možnost, da se na njeni desni strani pojavi kateri koli operandni, tj. stvarni označevalec oziroma formula). Pri tem je α lahko vobče formula v formuli $\alpha \models$. S formulo (1) je opisan (omogočen) aktualni in potencialni vpliv stvari na druge stvari. Hkrati s formulo (1) ne pozabljamo, da vsakokrat, ko zapišemo α, predpostavljamo že $\alpha \models$. Torej imamo opravka z

informacijsko implikacijo (operatorski znak $\Rightarrow$) oziroma s formulo nekega osnovnega modusa informacijskega sklepanja

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

Formula (2) poudarja, da se ob pojavu stvari α zavedamo stvarnega informiranja stvari in da smemo ali moramo to zavest upoštevati pri graditvi (kompoziciji, induciranju, sestavljanju), izpeljevanju (deduciranju), dekonstruiranju (dekompoziciji, destrukciji) formul skupaj z njihovo univerzalizacijo in partikularizacijo.

V trenutku, ko smo izrekli predpostavko, da stvar informira, tj. informira druge stvari in samo sebe, nastane zavest, da je stvar tudi informirana, tj. vplivana s stvarmi. Tudi v tem primeru ne izrečemo določno, s katerimi stvarmi je stvar informirana, odprta ostaja torej vsa aktualnost in potencialnost informiranja (vplivanja na stvar). Odprta formula »stvar α je informirana« ima simbolični zapis

(3) $\models \alpha$

Leva stran operatorja $\models$ je prazna, kar označuje odprtost zapisane formule. Pri tem je α lahko vobče formula v formuli $\models \alpha$. S formulo (3) je opisan (omogočen) aktualni in potencialni vpliv na stvar α z drugimi stvarmi. Hkrati s formulo (3) ne pozabljamo, da vsakokrat, ko zapišemo α, že predpostavljamo $\models \alpha$. Torej je v ozadju razumevanja stvari α prisotna informacijska implikacija (operatorski znak $\Rightarrow$) oziroma formula informacijskega sklepanja

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

Formula (4) poudarja, da se ob pojavu stvari α zavedamo njene informiranosti (vplivanosti) s stvarmi in da smemo ali moramo to zavest upoštevati pri informacijski graditvi, izpeljevanju, dekonstruiranju (dekompoziciji) formul itd.

V formulah (2) in (4) je na površje informacijskega razumevanja stvari stopila konsekvenca, da je t.i. operator informiranja $\models$ vselej sestavni (konstitucionalni) del stvari α same. Kadar imamo interakcijo med dvema stvarema, je operator $\models$ med njima (npr. v formuli $\alpha \models \beta$) izraz njune informacijske povezanosti (operatorske kompozicije obeh, v interakciji udeleženih stvari). Narava operatorja $\models$ v konkretnem primeru je tedaj odvisna izključno od narave udeleženih stvari samih. Npr. aritmetična formula a + b govori (implicitno), da sta a in b seštevljivi števili, da npr. a informira seštevljivo (v obliki formule a +) in da je b informirano seštevljivo (formula + b). Če sta a in b števili, velja namreč $a \Rightarrow (a +)$ in $b \Rightarrow (+ b)$. O informacijski naravi matematičnih formul bomo govorili izčrpneje v posebnem poglavju.

Posledica dosedanjih izpeljav je, da se stvar α sama kaže kot informacijski sistem formul

(5) $\alpha \models; \models \alpha$

Ko govorimo o stvari α, predpostavljamo

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

Formula (6) poudarja navzven in navznoter pojavljajoče informiranje stvari α, možnosti njenega »izhodnega« $(\alpha \models)$ in »vhodnega« $(\models \alpha)$ odprtega informiranja. Na osnovi premislekov v okviru formul (1) do (6) postavimo končno naš izhodiščni aksiom stvari na sebi, ki nam bo omogočal spontano in cirkularno substitucijo v informacijskih formulah, in sicer s formulo

(7) $\alpha \Leftrightarrow_{Df} (\alpha \models; \models \alpha)$

Znak $\Leftrightarrow_{Df}$ je definicijska (aksiomatska) ekvivalenca in formula (7) je osnovno pravilo za substitucijo formule α s sistemom formul $(\alpha \models; \models \alpha)$ in za substitucijo tega sistema s formulo α v informacijskih formulah.

Kaj je formula (7) kot definicija, kakšna je njena narava? Ali je ta narava definicije dopustna? Formula (7) je rekurzivna formula, saj se v njej stvar α definira z α in z operatorjem $\models$. Kakšen je smisel te tavtološke definicije in kakšne so njene posledice? Definicija (7) ni izrazito »vsebinska«,

je predvsem »gramatična«. Ker je rekurzivna, definira α kot α-jevsko formalno razširitev v sistem $\alpha \models; \models \alpha$ ali redukcijo tega sistema v α. Ker pa je α zaenkrat poljubna formacija (stvar, formacija, formula, informacijska entiteta), velja (7) za poljubno formacijo oziroma tudi formulo, npr. tudi za formuli sistema $\alpha \models; \models \alpha$. Posledici rekurzivne definicije α s formulo (7) sta tedaj sistematično tile:

(8.1) $(\alpha \models) \Leftrightarrow_{Df} ((\alpha \models) \models; \models (\alpha \models))$;
(8.2) $(\models \alpha) \Leftrightarrow_{Df} ((\models \alpha) \models; \models (\models \alpha))$

Definiciji (8.1) in (8.2) govorita o odprtem informiranju formacije α, tj. o informiranju entitete $\alpha \models$ in entitete $\models \alpha$. Podobne definicije lahko nedaljujemo v nedogled. Kaj pravita formuli (8.1) in (8.2)? Da entiteta $\alpha \models$, ki je prvi modus informiranja α, informira in je informirana in da entiteta $\models \alpha$, ki je drugi modus informiranja α, informira in je informirana. Itd., itd.

Vprašanje, ki ostane, je, ali je definicija (7) zadostna, da se z njo opredeli celostna narava stvari α, tj. narava formacije same oziroma stvari same? Če je temu tako, potem izven informiranja stvari same ni ničesar več in je vse o stvari zajeto z definicijo (7), in sicer tako informiranje stvari v sebi kot informiranje stvari za druge stvari. Formula (7) je zaenkrat (hipotetično) povsem zadovoljiva, saj ohranja skrajno možno odprtost (spontanost, cirkularnost) stvari same kot razpoznavne formacije.

Informiranje stvari v sebi (metafizika stvari)

Stvar oziroma formacija stvari je vselej odprta, izpostavljena vplivom drugih stvari in same sebe. Kako vpliva stvar sama nase in kako je vplivana s samo seboj kot v okolje odprt sistem? Na to specifično vprašanje bomo odgovarjali s pojmom metafizike stvari, ki je odprto informiranje stvari v sami sebi. Kaj je metafizika stvari same? To očitno ni stvar sama, kot smo jo opredelili v prejšnjem podpoglavju, je pa informacijski del stvari same. Kateri del stvari same je metafizika kot informacija stvari v sebi sami?

Ker nam gre za formalno izpeljavo pojma metafizika, se vprašajmo, kaj vse lahko implicira aksiom (7). Entiteta α je tako kot vsaka formacija odprta tudi za vpliv entitete na sebe. Aksiom (7) omogoča, da delno zapremo njegov sistem z entiteto samo. To pa pomeni

(9) $(\alpha \models; \models \alpha) \Rightarrow (\alpha \models \alpha)$

Desno stran implikacije (9), tj. izraz $\alpha \models \alpha$, imenujemo metafizika stvari α. Metafizika $\alpha \models \alpha$ je kot vsaka formacija tudi odprt proces, za katerega velja podobno kot v primerih (2) in (4)

(10) $(\alpha \models \alpha) \Rightarrow ((\alpha \models \alpha) \models)$;
(11) $(\alpha \models \alpha) \Rightarrow (\models (\alpha \models \alpha))$

Tako imamo skladno z aksiomom (7) za metafiziko stvari kot poseben primer

(12) $(\alpha \models \alpha) \Leftrightarrow_{Df} (((\alpha \models \alpha) \models); (\models (\alpha \models \alpha)))$

Stvar sama $(\alpha \models; \models \alpha)$ implicira metafiziko stvari oziroma stvar v sebi $(\alpha \models \alpha)$, tj.

(13) $\alpha \Rightarrow (\alpha \models \alpha)$

in skladno z aksiomom (7) tudi

(14) $\alpha \models (\alpha \models \alpha)$

saj je metafizika $\alpha \models \alpha$ odprta informacija stvari same α oziroma njenega odprtega sistema $(\alpha \models; \models \alpha)$. Metafizika stvari je regularna (odprta) formacija (informacija) v stvari sami, to pa je vsebovanost $(\alpha \models \alpha) \subset \alpha$.

Metafizika $\alpha \models \alpha$ stvari α pa je v svojem formalnem okviru (formuli $\alpha \models \alpha$) zapopadena le kot začetni, izhodiščni scenarij pri dekompoziciji, dekonstrukciji, pomenskemu razčlenjevanju in sestavljanju razumevanja metafizike $\alpha \models \alpha$ kot kompleksne, formacijske entitete. V primeru zapisa (osnutka) $\alpha \models \alpha$ torej ne gre za primer trivialne tavtologije, kot bi se to lahko domnevalo

na prvi pogled. Formula $\alpha \models \alpha$ bi bila trivialna izražava le v primeru, če bi ne bila mogoča nikakršna njena notranja in zunanja (odprtnostna) dekonstrukcija, če ne bi bilo nobene namere ali vidne možnosti za njeno dekompozicijo kot notranjo in zunanjo fenomenologijo, oziroma če v njej sami ne bi bilo ničesar drugega kot prav in samo njena trivialnost $\alpha \models \alpha$, npr. v obliki njene podatkovne obstojnosti, tj. $\alpha = \alpha$.

Predpostavimo neko razumevanje $\mathfrak{A}$ v okviru metafizike $\alpha \models \alpha$, torej pojav formacije $\mathfrak{A} \subset (\alpha \models \alpha)$, ki je situacija nekega delnega razumevanja v okviru metafizike stvari α. To razumevanje naj informira (generira, povzroča) nastajanje pomena $\pi(\alpha)$, tj. nastajanje α-metafizičnega pomena π. Naravna dekonstrukcija začetnega scenarija $\alpha \models \alpha$ je tedaj lahko

(15) $((\alpha \models \mathfrak{A}) \models \pi(\alpha)) \models \alpha$

Izpeljava te formule iz prvotne formule $\alpha \models \alpha$ (dekonstrukcija) je

(16) $(\alpha \models \alpha) \models (((\alpha \models \mathfrak{A}) \models \pi(\alpha)) \models \alpha)$

Primerjava formule (15) s formulo $\alpha \models \alpha$ kaže, kako smo operator $\models$ v formuli $\alpha \models \alpha$ na določen način partikularizirali, in sicer kot

(17) $\models \mathfrak{A}) \models \pi(\alpha)) \models$

Operator (17) kaže način povezave med levim in desnim označevalcem α v formuli $\alpha \models \alpha$. Z levim operandom α se pojavita tudi »manjkajoča« oklepaja »(«, torej

(18) $((\alpha \models \mathfrak{A}) \models \pi(\alpha)) \models$

Odprta stran te formule pa se poveže z α.

Scenarij $\alpha \models \alpha$ je mogoče še dalje dekonstruirati, npr. z obliko paralelnega sistema razumevanja, v katerem se pojavljajo različne oblike razumevanja in z njimi generirani pomeni. Tako dobimo (paralelni) sistem formul, ki se skozi osnovno entiteto α povezujejo kot

(19) $((\alpha \models \mathfrak{A}_1) \models \pi_1(\alpha)) \models \alpha;$
$((\alpha \models \mathfrak{A}_2) \models \pi_2(\alpha)) \models \alpha;$
. . . . . .
$((\alpha \models \mathfrak{A}_m) \models \pi_m(\alpha)) \models \alpha$

in seveda še drugače in na več načinov, npr. kot

(20) $((\alpha \models \mathfrak{A}) \models (\pi_1(\alpha), \pi_2(\alpha), \ldots, \pi_m(\alpha))) \models \alpha$

kjer so $\pi_1(\alpha)$, $\pi_2(\alpha)$, ... , $\pi_m(\alpha)$ medsebojno prepleteni pomeni s skupno spremenljivko α, ki je formacija stvari same, njene metafizike in vplivni del formacije za druge.

Kakšna bi lahko bila standardna dekompozicija metafizike stvari $\alpha \models \alpha$, ki bi zadoščala poljubnim partikularnim konceptom (zahtevam, scenarijem, modelom, pojavnostim stvari)? Vprašanje o metafiziki stvari je mogoče postaviti v okviru odprtega sistema nastajanja stvari v sami sebi. Nastajanje stvari je tu sinonim za njeno pojavnost v najširšem pomenu besede, torej tudi za njeno spreminjanje, postajanje, izginjanje, ohranjanje itd. Kako bi bilo mogoče nastajanje stvari $\alpha \models \alpha$ dekomponirati (destrukturirati, dekonstruirati) na razumen in standarden način, iz katerega bi bilo mogoče izhajati v vsakem primeru?

Stvar sama α informira in je informirana (kot običajno rečemo), tj. $\alpha \Leftrightarrow_{Df} (\alpha \models; \models \alpha)$. V okviru tega informiranja stvari same se pojavlja njena metafizika $\alpha \models \alpha$. Principsko je metafizika stvari neko njeno odprto cirkularno informiranje, ko stvar odprto informira sama sebe in je informirana s svojim okoljem. Formula $\alpha \models \alpha$ je odprta, torej $(\alpha \models \alpha) \models$ in $\models (\alpha \models \alpha)$.

Osnovna (izhodiščna) metafizična shema je $\alpha \models \alpha$. To shemo je mogoče dekomponirati na različne načine. Toda upoštevaje principe informacije, s katerimi so informiranje, protiinformiranje in umeščanje informacije osnovni informacijski pojavi, je mogoče šteti te oblike informiranja za standardne konstituente tudi in zlasti v okviru metafizičnega informiranja $\alpha \models \alpha$.

Tako je mogoče uvideti, kako postane formula $\alpha \models \alpha$ netrivialna oziroma netavtološka (v običajnem pomenu) zaradi možnosti dekompozicije te formule. Uvedimo najprej tri vrste

entitet, ki se pojavljajo v okviru informiranja stvari α. Te entitete so informacija α, protiinformacija γ in umeščevalna informacija ε. Prva standardna dekompozicija formule $\alpha \models \alpha$ je tako lahko tale:

(9a) $(\alpha \models \alpha) \Rightarrow ((\alpha \models \gamma) \models \varepsilon) \models \alpha$

Formula (9a) kaže, kako je bil operator $\models$ v formuli $\alpha \models \alpha$ zamenjan (dekomponiran) z nečim, kar se formalno pojavlja kot

(9b) $\models \gamma) \models \varepsilon) \models$

tj. kot operatorska entiteta, ki s svojo oklepajno nezaprto obliko kaže, da je operator (enostaven ali dekomponiran, tj. razčlenjen) vselej vezan na svoje operande.

Seveda pa lahko uvedemo še pojavljajoče (spremljajoče) fenomene informiranja za α, γ in ε. Te fenomene označimo vobče z $\mathfrak{Z}$, $\mathfrak{C}$ in $\mathfrak{E}$ ter jih poimenujmo informiranje, protiinformiranje in umeščanje informacije. Dokompozicija začetne sheme $\alpha \models \alpha$ je potem lahko tale:

(9c) $(\alpha \models \alpha) \Rightarrow$
$(((((\alpha \models \mathfrak{Z}) \models \gamma) \models \mathfrak{C}) \models \varepsilon) \models \mathfrak{E}) \models \alpha$

Ta shema predstavlja eno od smiselnih konceptualizacij metafizičnega informiranja, ki opisuje metafizično nastajanje stvari α. Zaradi cirkularne narave zadnje formule, tj. njenega desnega dela v implikaciji, so vse njene operandne komponente α, $\mathfrak{Z}$, γ, $\mathfrak{C}$, ε in $\mathfrak{E}$ cirkularno povezane in zaradi te cirkularnosti medsebojno odvisne. Seveda pa lahko ima ta odvisnost še druge oblike, npr. kot so paralelne formule, posledice nadaljnih dekompozicij konkretnih metafizičnih situacij itd. Tako je npr. mogoče vpeljati formule

(9d) $\mathfrak{Z} \models \alpha$; $((\mathfrak{Z} \models \mathfrak{C}) \models \mathfrak{E}) \models \mathfrak{Z}$; ...

paralelno k temeljnim formulam ali pa jih postaviti kot samostojne formule nekega metafizičnega primera. Tako formula $\alpha \models \alpha$ informira spontano in cirkularno in prav na tak način informirajo tudi njene metafizično vpletene komponente.

Informiranje stvari za druge (opazovanje stvari)

Kakšna je fenomenologija opazovanja stvari v stvari sami? Opazovalec (označevalec ω) opazuje (sprejema neko informacijo) opaženca (označevalec α). Pri tem še ni povedano, kaj in kako opazovalec opazuje; vendar lahko opazuje le to, kar opaženec informira. Ta ugotovitev je bistvena za razumevanje fenomenalnosti opaženca kot odprte entitete (formacije stvari), ki je ali bo lahko opazovana z različnimi aktualnimi in potencialnimi opazovalci.

Stvar α informira kot stvar za druge v odprti obliki

(21) $\alpha \models$

Opazovalec ω pri tem predpostavlja (sklepa, je prepričan, pričakuje), da stvar, ki je, informira; torej

(22) $\alpha \Rightarrow (\alpha \models)$

Informacijska implikacija je tako (kot vobče vsaka implikacija) konstrukt opazovalne stvari, ki je lahko stvar sama (npr. metafizika stvari $\alpha \models \alpha$ kot odprta informacija) ali druga (opazovalna, vplivana) stvar (ω). V konkretnem primeru, ko opazovalec ω opazuje formacijo α, se formula (21) delno zapre, in sicer v obliki

(23) $\alpha \models \omega$

Ta formula, ki je stvar opaženca α in njegovega opazovalca ω, je seveda še naprej odprta, saj za njo kot formacijo velja formula (7), torej

(24) $(\alpha \models \omega) \Leftrightarrow_{Df} (((\alpha \models \omega) \models); (\models (\alpha \models \omega)))$

Opazovalni proces (23) je lahko vobče informacijsko vplivan z drugimi zunanjimi (tudi nezavednimi, neopaženimi) entitetami.

Znanstveno (ali filozofsko) opazovanje pred-

postavlja, da opazovana formacija α ni motena oziroma vplivana z opazovalcem ω. Opazovalec tako predpostavlja formulo

(25) $(\alpha \models \omega) \Rightarrow ((\alpha \models \omega) \not\models \alpha)$

ki govori, da opazovalni proces $\alpha \models \omega$ ne vpliva na (ne informira) opazovano formacijo α. Operator $\not\models$ je informacijski operator (določenega, partikularnega) neinformiranja, tj. opazovalnega nevplivanja na opazovano stvar α.

Kaj je vobče informiranje stvari same v metafiziki drugih stvari samih? V okviru opazovanja stvari α opazuje opazovalec ω tudi sam proces opazovanja $\alpha \models \omega$, npr. kot informiranje specifične opazovalne metodologije, ki je proces nastajanja opazovalnega pomena (rezultata, razumevanja pomena) v okviru opazovalčeve metafizike $\omega \models \omega$. Opazovalec ω lahko tako predpostavlja več različnih informacijskih modusov v okviru opazovalnega procesa $\alpha \models \omega$. Ti modusi so npr. tile (od preprostih, trivialnih do čedalje bolj zapletenih):

(26) $(\alpha \models \omega) \Rightarrow (\alpha \models (\omega \models \omega))$;
(27) $(\alpha \models \omega) \Rightarrow ((\alpha \models \omega) \models (\omega \models \omega))$;
(28) $(\alpha \models \omega) \Rightarrow ((\alpha, (\alpha \models \omega)) \models (\omega \models \omega))$

itd. ad infinitum. V okviru opazovalne formacije ω se pojavlja neko razumevanje kot informiranje stvari α in to razumevanje β stvari α (zaradi α), označeno z β_α, povzroča nastajanje (generiranje) pomena $\pi_\omega(\alpha)$ entitete α. Opazovalna formacija se tako vobče sooča s svojim razumevanjem opazovane formacije, s katerim se generira pomen (smisel) opazovanja opazovane formacije α. Tako dobimo naposled informacijski modus opazovalnega razumevanja fenomena α v obliki

(29) $(((\alpha, (\alpha \models \omega)) \models (\omega \models \omega)) \models \beta_\alpha) \models \pi_\omega(\alpha)$

Formulo (29) smo zapisali v neimplikativni obliki kot proces opazovanja, ki ga izvaja entiteta ω. Ta proces je implicitni cikel opazovanja, ko se opazovalni pomen $\pi_\omega(\alpha)$ povratno informacijsko navezuje na sestavljeno entiteto $(\alpha, (\alpha \models \omega))$ na začetku formule.

Kako se vobče razumevanje stvari α in skozi to razumevanje nastajanje pomena o stvari α pri opazovalni entiteti ω pojavlja v osnovnem procesu $\alpha \models \omega$? Kaj označuje osnovni proces $\alpha \models \omega$ oziroma skupni operator $\models$ operandov α in ω tega procesa? Operator $\models$ je namreč skupni, z obema operandoma pogojeni operator, torej nekakšna kompozicija informacijskega stika med entitetama α in ω, in sicer tako, da entiteta α vpliva (informacijsko) na entiteto ω. Operator informiranja je vselej lastnost informirajoče entitete, del njene »biti«, njene eksistence, kot smo opredelili s formulama (1) in (3). Prav pri opazovanju stvari s stvarjo imamo priložnost, da to lastnost pojasnimo tudi s stališča operatorske kompozicije.

Oglejmo si najprej, kaj bo operatorska kompozicija v osnovnem opazovalnem primeru $\alpha \models \omega$. V tem primeru gre za spojitev dveh samostojnih procesov, in sicer

(30) $\alpha \models ; \models \omega$

V prvem primeru, tj. $\alpha \models$, informira stvar α v prostor in čas, in sicer na sebi svojstven način. Torej lahko združimo svojstvenost informiranja stvari α in njeno odprtost v prostor in čas s formulo

(31) $\alpha \models_\alpha \circ \models$

Operator $\models_\alpha \circ \models$ je operatorska kompozicija (znak $\circ$) operatorjev $\models_\alpha$ in $\models$, kjer je zadnji operator zopet operator splošnega tipa, ki se lahko veže kamor koli, na katero koli (vplivano) stvar.

V drugem primeru, tj. $\models \omega$, je stvar ω informirana v prostoru in času, in sicer na sebi svojstven način. Zopet lahko združimo svojstvenost informiranosti (vplivanosti) stvari ω in njeno odprtost v prostor in čas s formulo

(32) $\models \circ \models_\omega \omega$

Operator $\models \circ \models_\omega$ je operatorska kompozicija (znak $\circ$) operatorjev $\models$ in $\models \circ \models_\omega$, kjer je prvi operator kompozicije zopet operator splošnega tipa, ki je lahko vezan kamor koli, na katero koli

(vplivajočo) stvar.

V primeru procesa $\alpha \models \omega$ gre torej za stik pojavov (31) in (32) brez posredovanja neke tretje stvari. Sistem

(33) $\alpha \models_{\alpha} \circ \models ; \models \circ \models_{\omega} \omega$

je tedaj spojljiv v kompozicijo parcialnih sistemov v »točki« $\circ$, tj. v obliko

(34) $\alpha \models_{\alpha} \circ \models_{\omega} \omega$

Tudi osnovna opazovalna (α-vplivna) formula $\alpha \models \omega$ je nastala s spojitvijo sistema $\alpha \models ; \models \omega$. Tako je operator $\models$ v bistvu spojni (in razločevalni) operator. Formula (34) je tedaj posledica premisleka o informiranju procesa $\alpha \models \omega$. Torej imamo

(35) $(\alpha \models \omega) \Rightarrow (\alpha \models_{\alpha} \circ \models_{\omega} \omega)$

Formula (35) govori o naravi informiranja stvari, o tem, kako stvar informira drugo stvar, kakšen je možen vpliv informirajoče stvari na informirano stvar, kako je medsebojno informiranje stvari vselej pogojeno z nagovorom stvari, ki informira, in s poslušanjem stvari, ki je informirana. Operacijski znak $\circ$ v operatorski kompoziciji $\models_{\alpha} \circ \models_{\omega}$ je v bistvu znak razločevanja med informiranjem entitete $\alpha \models_{\alpha}$ in entitete $\models_{\omega} \omega$. Tu gre torej za opaženo razloko (*franc.* différance) med procesoma α in ω v sestavljenem informacijskem procesu $\alpha \models \omega$.

Kaj je tedaj informacija kot stvar sama?

Na to vprašanje lahko odgovorimo šele potem, ko smo delno odgovarjali na vprašanja, kaj je stvar sama, kaj je njena metafizika in kaj je stvar za druge. Poudarili smo, da je stvar sama vsa njena aktualna in potencialna informacijska fenomenalnost, da je torej v pojmu stvari same zapopadena tudi njena metafizika in to, kar je stvar za druge, tj. kako se stvar sama pojavlja v metafiziki drugih. Formula (7), ki za α implicira sistem $(\alpha \models ; \models \alpha)$, je tedaj vseobsegajoča (vseupoštevajoča) formula glede na α, v njej je vsebovano (zaobjeto) vse, kar na stvar lahko vpliva zunaj nje same, kar se kot stvar sama pojavlja znotraj stvari same in kar povzroča stvar zunaj sebe glede na poljubno in nepredvidljivo pojavljanje drugih stvari (opazovalcev). Pri vsem tem so bistvene tudi formalne razlike, ki jih je sedaj mogoče opaziti (razločevati) med stvarjo samo, njeno metafiziko in stvarjo za druge. Prav formalna predstavitev problema je povzročila, da je navedena razloka prišla do svojega lastnega nagovora tisti opazovalni entiteti, ki je za ta nagovor stvari same ostala odprta.

Vprašanje je, ali smo z razloko treh entitet v okviru vprašanja o stvari sami povedali nekaj dokončnega, bitno konstruktivnega. Seveda ne, saj smo le grobo razprli okvir možnega razpoznavnaja stvari same, ki prav za prav ni le stvar sama, temveč je tudi stvar drugih stvari, v posebnih primerih pa celo pretežno stvar drugih stvari in to tako, kot druge stvari stvar samo anticipirajo. Vprašanje je sedaj, kako omenjena anticipacija stvari same pri (v) drugih stvareh vstopa v pojavljanje stvari same. Trdimo, da navedene simbolne formule informiranja stvari odgovarjajo tudi na to vprašanje, dokler ostajamo odprti za nagovor postavljenih simbolnih formul informiranja stvari.

Vpeljimo informacijski operator $\subset$, ki označuje operacijo (lastnost) vsebovanosti (biti-v). Naj $\xi \subset \eta$ pomeni »proces ξ je del procesa η«. Združimo formule o informiranju stvari same (7), o metafiziki stvari (9) in o informiranju stvari za druge (22) v sistem

(36) $\alpha \Leftrightarrow_{Df} (\alpha \models ; \models \alpha);$
$(\alpha \models ; \models \alpha) \Rightarrow (\alpha \models \alpha);$
$\alpha \Rightarrow (\alpha \models)$

Ta sistem je izveden (deduciran) v samem sebi, in sicer tako, da imamo

(37) $(\alpha \models ; \models \alpha) \subset \alpha;$
$(\alpha \models \alpha) \subset (\alpha \models ; \models \alpha);$
$(\alpha \models) \subset \alpha$

Pri nagovoru tega sistema je priporočljiva posebna občutljivost. Sistem je večkrat cirkularno strukturiran, odprt za nagovarjanje in za nagovor drugih

stvari na sebi in metafizično. Stvar sama, njena metafizika in stvar za druge so povezani konstituenti stvari, torej

(38) $((\alpha \models; \models \alpha); (\alpha \models \alpha); (\alpha \models)) \subset \alpha$

Ta formula se tedaj vendarle razlikuje od razčlenjene oblike te formule, ki bi bila sistem

(39) $(\alpha \models; \models \alpha) \subset \alpha;$
$(\alpha \models \alpha) \subset \alpha;$
$(\alpha \models) \subset \alpha$

s katerim je partikularna razčlenjenost stvari same, njene metafizike in stvari za druge pokazana eksplicitno glede na α. Ta razčlenjenost pa nikakor ni definitivna, saj velja do nadaljnega (in kar naprej in tako dalje)

(40) $\subset \alpha$

kot odprta formula.

Pojavnost informiranja stvari

Kako informirajo stvari oziroma njihove formacije kot najbolj enostavne stvari in kot strukturno in organizacijsko medseboj prepletene entitete? Kaj sploh je najenostavnejša, še informacijsko sprejemljiva, do skrajnosti poenostavljena formacija stvari? Kako stvari lahko medsebojno vplivajo na svoje formacije, kaj je lahko formacija delnih formacij stvari samih? Na ta vprašanja bomo odgovarjali v naslednjih dveh podpoglavjih.

Podatek kot posebna oblika formacije stvari

Kako informira podatek kot stvar sama, kaj je njegova metafizika in kaj je podatek za druge? V čem je podatek različen — če je sploh različen — od splošnega pojma informacije v formuli (7) in kakšna je njegova posebna, glede na formulo (7) partikularna opredelitev? Katere »enostavnejše« (reducirane) oblike opredelitve informacije so sploh mogoče?

Asociacija, da je podatek enostavnejši od informacije, je predvsem zgodovinsko-razvojna. Podatek naj bi bilo nekaj, kar še ni informacija v popolnem informacijskem smislu. Podatek je predvsem zanesljiva informacija, tj. stabilna, nespremenljiva informacija, ki »ostaja« informacijski fakt, tj. bistvo podatka. Vendar preglejmo analitično, ali je to povsem res in ali ne gre pri podatku že za specifično partikularizacijo (konkretnost) splošnega pojma informacije, izraženega s formulo (7).

Podobno kot za pojem informacije je tudi za podatek kot specifičen informacijski pojav mogoče opredeliti, kaj je podatek sam, kaj njegova metafizika in kaj je podatek za druge. Ta primer je zanimiv, ker gre za partikularizacijo formule (7), ki je definicija oblike $\alpha \Leftrightarrow_{Df} (\alpha \models; \models \alpha)$. Za podatek δ sam imamo podobno kot za vsako drugo informacijsko entiteto najprej v splošni obliki

(41) $\delta \Leftrightarrow_{Df} (\delta \models; = \delta)$

Kakšna je tedaj bistvena razlika med to formulo in formulo (7)? Razlika je očitno v informiranju podatka, ki naj pojavno oziroma informacijsko učinkuje kot nespremenljivo dejstvo. Vsako dejstvo pa mora predvsem ohranjati svojo pojavnost (podatkovnost) kot danost, ki naj obvelja tako ali drugače enkrat za vselej. Iz formule (41) seveda lahko izpeljemo vse nadaljne posebne oblike (metafizika podatka in podatek za druge).

Metafizika podatka je tedaj njegova nespremenljiva identiteta, ko se splošna formula informacijske metafizike $\alpha \models \alpha$ partikularizira (prevesi) v podatkovno metafizično identiteto

(42) $\delta \Leftrightarrow_{Df} (\delta = \delta)$

To formulo izpeljemo iz formule (41) tako, da upoštevamo operatorsko kompozicijo oblike $\models \circ =$, ki se pojavi v okviru operandne kompozicije $(\delta \models) \circ (= \delta)$. Operatorska kompozicija $\models \circ =$ se vselej reducira v operator $=$. Medtem ko ostaja informiranje podatka δ neomejeno odprto pa ostaja informacijski vpliv na podatek ničen zaradi aksioma (42). Ta aksiom onemogoča (preprečuje) kakršno koli nastajanje že nastalega podatka, ki se

je ob nekem dogodku, tj. pri svojem nastanku, pojavil kot dejstvo. Torej velja za podatek eksplicitna predpostavka

(43) $\delta \Rightarrow (\delta \models; \delta = \delta)$

ali tudi implicitno

(44) $\delta \Rightarrow ((\delta = \delta); (\delta = \delta) \models; \not\models (\delta = \delta))$

Značilnost podatka v primerjavi s splošno definicijo informacije je v njegovi neinformiranosti (enakosti), tj. v implikaciji

(45) $(\delta = \delta) \Rightarrow (\not\models \delta)$

Odprtost podatka za njegovo nastajanje, ko je podatek že nastal, naj bi bila tako nična. Seveda pa tudi podatek enkrat nastane, ima svoj začetek, svoje bivanje kot nespremenljiva identiteta in svoj konec (pozabo, zbris, izginotje). Pokazali bomo, kako so različne formalne entitete, za katere si domišljamo njihovo spremenljivost, v bistvu le podatkovne stvari, ki so bile (tudi nezavedno) podatkovno zasnovane že na samem začetku.

Čeprav velja za podatek formula $\delta \Leftrightarrow_{Df} (\delta \models; = \delta)$ pa to vobče ni res pri opazovanju podatka δ z opazovalcem ω. Za proces opazovanja $\delta \models \omega$ spet velja formula (7), tj.

$$(\delta \models \omega) \Leftrightarrow_{Df} ((\delta \models \omega) \models; \models (\delta \models \omega))$$

Pri $= \delta$ imamo vendarle $\models (\delta \models \omega)$. Čeprav je podatek (npr. zapis, fotografija) δ vselej enak pa je njegovo opazovanje $\delta \models \omega$ regularna informacijska entiteta, odprta za informiranje opazovalca in drugih opazovalcev, to pa je lastnost $\models (\delta \models \omega)$. Zaradi tega je podatek δ v okviru opazovanja $\delta \models \omega$ tako stvar opazovalne informacijske logike in še posebej opazovalne entitete same.

Medsebojni vpliv formacij stvari

Medsebojni vpliv formacij stvari zadeva v svojem bistvu informiranje stvari za druge stvari. Odprtost informiranja stvari α v obliki $\alpha \models$ lahko povzroči informiranost (vplivanost) zaradi odprtosti sprejemnikov (vplivancev) β, γ itd. v oblikah $\models \beta$, $\models \gamma$ itd. Torej informiranje stvari implicira obstoj stvari, ki bodo informirane. Imamo

(46) $(\alpha \models) \Rightarrow ((\exists \beta, \gamma, \ldots) \cdot (\alpha \models \beta; \alpha \models \gamma; \ldots))$

kjer je $\exists$ eksistencialni informacijski operator in vobče $\exists \xi$ odprta eksistencialna formula za ξ ($\xi = \beta, \gamma, \ldots$). Entiteta ali entitete, ki informira ali informirajo eksistenco drugih stvari β, γ, ..., ostaja odprta (nedoločena) ali ostajajo odprte. Informacijski operator $\cdot$ povezuje eksistencialno formulo s pripadajočo namensko formulo ($\alpha \models \beta; \alpha \models \gamma; \ldots$).

Kot zanimivost navedimo še primer medsebojnega učinkovanja dveh stvari α in β, ko imamo npr.

(47) $(\alpha \models \beta; \beta \models \alpha) \Rightarrow (((\alpha \models \beta) \models \alpha); ((\beta \models \alpha) \models \beta))$

kjer prihaja do izraza cirkularna narava informacije (cikla za α in β) in je mogoče konstruirati raznovrstno informacijsko aksiomatiko pomnjenja, hermenevtike in drugih semantično cirkularnih pojavov.

Naslednje vprašanje v okviru medsebojnega vpliva stvari zadeva nekatera bistvena razločevanja alternativnega informiranja stvari. V tem okviru je mogoče konstruirati določene aksiome, ki kažejo posebnosti narave t.i. informacijske logike. Vzemimo najprej tale aksiom:

(48) $(\alpha \models \beta) \Rightarrow (\beta \rightleftharpoons \alpha)$

Entiteta »α informira (vpliva na) β« informacijsko implicira entiteto »β je informirano (vplivano) z α«. Ti entiteti sta lahko medseboj povsem različni,

saj v prvem (implikacijso levem) primeru α vpliva na β, v drugem (implikacijsko desnem) primeru pa je β vplivano z α. Operatorja $\models$ in $=\!|$ sta tedaj alternativno različna. Drugi smiselni aksiom je tudi

(49) $(\beta =\!| \alpha) \Rightarrow (\alpha \models \beta)$

Entiteta »β je informirano (vplivano) z α« informacijsko implicira entiteto »α informira (vpliva na) β«. Podobno kot v prvem primeru gre tudi tu za alternativno informiranje levega in desnega dela informacijske implikacije, kjer sta $\alpha \models \beta$ in $\beta \models \alpha$ različna (lahko alternativna, možna) procesa. Posledica tega je formula

(50) $\neg ((\alpha \models \beta) \Leftrightarrow (\beta =\!| \alpha))$

ki je informacijska negacija (operator $\neg$) ekvivalence (operator $\Leftrightarrow$) procesov $\alpha \models \beta$ in $\beta =\!| \alpha$.

Naslednje vprašanje, ki ga je smiselno postaviti v okviru medsebojnega vpliva formacij stvari, je vprašanje zavedanja in nezavedanja vpliva stvari samih. Zavedanje je povezano z lastnostjo opazovanja vpliva ene stvari na drugo oziroma z vplivanostjo stvari z drugo stvarjo. Če npr. α opazuje proces svojega vpliva na β, tj. proces $\alpha \models \beta$, imamo

(51) $(\alpha \models \beta) \models \alpha;$
$\alpha =\!| (\beta =\!| \alpha)$

Tu vidimo, kako prav α kot opazovalec opazuje distinkcijo med procesoma $\alpha \models \beta$ in $\beta =\!| \alpha$, se —kot rečemo — zaveda procesiranja (vplivanja) na β in distinkcije enega in drugega primera informiranja. Nezavedanje tega, pravkar opisanega pojava bi lahko eksplicitno izrazili z uvedbo posebnih operatorjev neinformiranja, npr. kot

(52) $(\alpha \models \beta) \not\models \alpha;$
$\alpha \not=\!| (\beta =\!| \alpha)$

ali pa tudi še drugače. S tem primerom bi eksplicitno poudarili nezavedanje takega ali drugačnega informacijskega vpliva α na β s strani α.

Informiranje matematične formule

Kaj je matematična formula (kratko MF) sama, njena metafizika in kaj je kot formula za druge? Na primeru MF je mogoče pokazati, kaj je formula kot predpis, kako ta predpis kot algoritem uresničuje svoje predpisano in kaj lahko pomeni formula za njene opazovalce, npr. za matematike oziroma tiste, ki jo »razumejo« in za tiste, ki zanjo nimajo le »čistega« matematičnega razumevanja.

Matematična formula je vobče informacijski (literarni, pisni, algoritmični, pedagoški) pomenski zapis (konstrukt, izraz), ki je na dobro oblikovan (*angl.* well-formed, dogovorjen, matematično legalen oziroma razumljen) način sestavljen iz formalnih (operacijskih, operandnih, relacijskih, ločilnih itd.) simbolov z dodatnim (še drugim, največkrat implicitnim, zavednim in nezavednim) pomenskim oziroma razumevnim ozadjem. Zaenkrat naj bo širši pomen MF, označen s $\varphi(\xi)$, informacijska entiteta, ki pojasnjuje matematični predpis (formulo, algoritem, funkcijo, funkcional, enačbo, spremenljivčno odvisen matematični sistem formul itd.), matematično-označevalno formuliran kot f(x). Medtem ko naj bi bil f(x) čista matematična forma (abstraktni zapis z matematično določenim pomenom), razumevanje katere je v okviru disciplinarne matematike standarizirano vobče z F(X), pa je $\varphi(\xi)$ informacijsko (naravno, metafizično in drugo) razumevanje tega prečiščenega (disciplinarnega) pomena, torej tista entiteta, ki pomeni f(x) hkrati kot stvar samo, njeno metafiziko in stvar za druge. Kaj je tedaj formula f(x) v perspektivi njenega aktualnega in potencialnega razumevanja $\mathfrak{F}(\mathfrak{X})$, s katerim nastaja pomen $\varphi(\xi)$?

Ker je f(x) standariziran (matematično disciplinaren) pomen, obstaja zanj bolj ali manj predpisano (pravzaprav natančno, enoumno) matematično razumevanje, ki ga označimo z F(X). V okviru matematike se formula (matematični simbolni zapis) f(x) razumeva kot sistem cirkularnih procesov

(53) $(f(x) \models F(X)) \models f(x);$
$(x \models X) \models x$

kjer je X še matematično razumevanje zadevne množice spremenljivk x. To razumevanje pravzaprav intencialno (nezavedno) poudarja enopomenskost formule f(x), tj. njeno podatkovno metafizičnost (tavtološkost), torej

(54) $((f(x) = f(x)) \models F(X)) \models (f(x) = f(x));$
$((x = x) \models X) \models (x = x)$

Formula f(x) = f(x) izraža pomensko ireduktibilnost, ki je determinirana podatkovnost zapisa f(x). Kaj se skriva za simbolnim zapisom f(x), ki ga matematiki razumevajo skozi F(X)? Kakšna je metafizika formule f(x)?

V konkretnem primeru f(x), ko so za ta zapis določeni vsi podatki (argumenti, postopki za izračun ali določanje vrednosti pri izbranih argumentih), se lahko izračuna ali določi vrednost formule (npr. funkcije) f(x) v okviru matematičnega razumevanja F(X) formule f(x) in razumevanja X njenih argumentov x. Ta proces razumevanja je za dano formulo vselej ponovljiv, kar pomeni, da daje ponovljive (predvidljive) rezultate za formulo f(x). Razumevanje F(X) formule f(x) in razumevanje X njenih argumentov je dobro (natanko) opredeljena receptura (postopkovnost, procedura) in vsa ta informacija oblikuje metafiziko (matematično legalnost) formule f(x). Ta legalnost lahko doseže tako stopnjo, da je izračun formule f(x) mogoče uresničiti tudi s strojem (tehnološkim orodjem). V tem primeru orodje samo razpolaga z ustreznim (legalnim) razumevanjem F(X) in X.

Metafizičnost matematične formule f(x) je tako zajeta v scenariju, ki ga opisuje formula (53) in skozi optiko narave njene podatkovnosti še formula (54). Pri tem velja za formulo f(x) in njene argumente x tudi podatkovni ekvivalenčni princip, in sicer

(55) $x \Leftrightarrow_{Df} (x \models; = x);$
$f(x) \Leftrightarrow_{Df} (f(x) \models; = f(x))$

V teh formulah sta prav = x in = f(x) definitorni formuli, s katerima neka, tu neopredeljena teorija, opredeljuje (definira, enači s pripadajočima pojmoma) zapisa (znaka) x in f(x). Tako je npr. y = f(x) definicijsko informirano z neko teorijo $\mathfrak{T}$ v obliki $\mathfrak{T} \models (y = f(x))$.

Kompleksen cikel razumevanja in nastajanja pomena argumentov x in formule f(x) je mogoče opisati z upoštevanjem širšega razumevanja $\mathfrak{X}$, $\mathfrak{Y}(\mathfrak{X})$, ki temelji v naravi naravnega in ne le matematičnega jezika oziroma v mišljenju, in sicer v obliki formule

(56) $((((x, f(x) \models X, F(X)) \models \mathfrak{X}, \mathfrak{Y}(\mathfrak{X})) \models$
$\xi, \varphi(\xi)) \models x, f(x)$

in iz nje izpeljivih nadaljnjih scenarijev. V tem okviru je pomen ξ, $\varphi(\xi)$ utemeljen s koncepti naravnega jezika, ki so vplivali in še pomensko, tj. razumevno vplivajo na osnovno matematično formulacijo x, f(x).

Informiranje računalniških programov

Računalniški program je kot informacijska entiteta lahko razumljen na več načinov: kot zapis v posebnem jeziku (ki tako ali drugače predstavlja neko rekurzivno funkcijo), kot program v pomnilniku računalnika in naposled kot izvajajoča (v računalniku, z njegovo pomočjo izvršujoča) podatkovna procedura (proces, operacija nad podatki).

Zaradi podatkovnosti razumevanja matematičnih formul je mogoče v vsakem posebnem primeru matematične formule prenesti to razumevanje tudi v samo računajoče oziroma procesirajoče orodje. Računalnik s programom je idealno matematično orodje, še posebej, če je pomnilno kompleksen in dovolj hiter.

Za računalniški program veljajo podobne ugotovitve kot za matematično formulo. Program, ki je zapisan v visokem programirnem jeziku, je le v posebnem jeziku izražena formula z natanko določenimi operandi in operatorji. V okviru matematičnega razumevanja je program rekurzivna funkcija (lahko tudi trivialna) in namesto f(x)

imamo pač p(x), kjer je p(x) izraženo v programirnem jeziku. Analogno k formuli (56) lahko zapišemo

(57) $((((x, p(x) \models X, P(X)) \models$
$\mathfrak{X}, \mathfrak{P}(\mathfrak{X})) \models$
$\xi, \pi(\xi)) \models x, p(x)$

Tu so x deklaracije spremenljivk in druge proste spremenljivke, nastopajoče operacije so legalni operatorji jezika in programske procedure itd. Razumevanje X, P(X) pripada računalniškemu prevajalnemu programu (prevajalniku, kompilatorju, interpreterju), razumevanje $\mathfrak{X}$, $\mathfrak{P}(\mathfrak{X})$ pa programerju, ki razumeva še kaj več pa tudi drugače kot prevajalnik in si tako generira nek pomen ξ, $\pi(\xi)$, ki je utemeljen z njegovim profesionalnim in drugim jezikovnim znanjem. Ta konfiguracija lahko vpliva individualno na izhodiščno formulacijo x, p(x), ki postane tako bistveno odvisna od posameznega programerja, izbire programirnega jezika, trenutne osebne atitude itd.

Sklep

V tem spisu smo pokazali nekatera izhodišča za formalizacijo informiranja stvari, ko smo vpeljali informacijski subjekt/objekt kot stvar samo, stvar v sebi in stvar za druge. Stvar sama α je kot odprt sistem $\alpha \models$; $\models \alpha$ opisovala vso aktualnost in potencialnost entitete α. Stvar v sebi (metafizika) in stvar za druge sta bili obliki te odprte aktualnosti in potencialnosti, ki sta seveda ostali tudi sami odprti. Stvar v sebi $\alpha \models \alpha$ je bila tako odprt sistem $(\alpha \models \alpha) \models$; $\models (\alpha \models \alpha)$, stvar α za druge stvari β, γ, ... pa odprt sistem $(\alpha \models \beta, \gamma, \ldots) \models$; $\models (\alpha \models \beta, \gamma, \ldots)$. Nakazana odprtost informacijskih formul je bila rekurzivno neomejena.

Pri tem velja poudariti, da je bila prikazana simbolna izpeljava formul (pojavov, scenarijev, procesov informiranja stvari) prevzeta iz avtorjevih spisov [1] in [2]. Osrednje izhodišče in hkrati vprašanje ostaja ustreznost oziroma logična sprejemljivost formule (7), ki opredeljuje formalizem (formalni opis) stvari same. Od priznanja smiselnosti takih opredelitev je odvisno, kako bo informacijski jezik lahko prodiral v zapopadenje subjektno-objektnega odnosa pri povsem praktičnih problemih, ki zadevajo — ne nazadnje — tudi oblikovanje informacijskih orodij. Proti vpeljavi informacijskega simbolizma govori prav gotovo matematična tradicija, ki zaenkrat ne sprejema tega, kar se incidentno od samega začetka uporabe računalniških sistemov dogaja v njih pri obdelavi podatkov — npr. pri procesiranju programov samih kot podatkov — in seveda pri programiranju kot individualnem, spontanem in značilno intencionalnem postopanju. Prav v okviru informacijskega principa spontanosti, ki je informacijska nastajalnost, se pri oblikovanju formul (programov) uporabljajo tudi polzavedni (ali nezavedni, instiktivni) načini kompozicije in dekompozicije, partikularizacije in univerzalizacije, serialnega in paralelnega širjenja in zoževanja (redukcije) konceptov, formul, programov. Do izraza prihaja to, kar v okviru informacije vobče pojmujemo kot fenomensko, eidosno, epistemsko, gnozno, telosno in poezno. Vse to pa je stvar informiranja stvari.

Slovstvo

[1] Železnikar, A.P., An Introduction to Informational Algebra, Informatica **14** (1990) *1*, 7-28.

[2] Železnikar, A.P., Understanding as Information II, Informatica **14** (1990) *4*, 5-30.

ER JEZIK: PRIJEDLOG PROŠIRENJA

INFORMATICA 3/91

Keywords: entity-relatinship data model, data modeling, logical database design

Mario Radovan
Fakultet Ekonomije i turizma, Pula;
Institut »Jožef Stefan«, Ljubljana

ER LANGUAGE: A PROPOSAL FOR EXTENSION The paper gives an analysis of the expressive power and of (some) limitatins of ER language as a means for designing data models. Starting from a different "origin", "nature" and function of different entities - and from the practical need that such a features be emphasized in the model - two proposals for extending the "standard" ER language are given: symbols for *process* and *disjunction* are introduced.

ER JEZIK: PRIJEDLOG PROŠIRENJA U članku je data analiza izražajnih mogućnosti i (nekih) ograničenja" ER jezika kao sredstva za oblikovanje modela podataka. Polazeći od različitog "porijekla", "prirode" i funkcije pojedinih entiteta - i praktikče potrebe da se ta svojstva naglase u modelu - iznijeta su dva prijedloga proširenja "standardnog" ER jezika: uvedeni su simboli *procesa* i *disjunkcije*.

1. Uvod

Model (baze) podataka tvori esencijalni dio informacijskog sistema, i *presudno* utječe na njegov kvalitet. Stoga je tokom posljednjih decenija razvijen niz prijedloga jezika (pretežno grafičkih) za oblikovanje modela (baze) podataka. Dominantno mjesto i najširu primjenu među njima ima *ER jezik* (ili: *ER model podataka*, kako se češće naziva u literaturi), čija je osnova predložena u <Che 76>.

"Nešto lakonskiji" pristup oblikovanju baze podataka možemo naći, naprimjer, u <Per 89>. Tamo čitamo: "Relacijska teorija zasnovana je na zdravom razumu. ... Ako 'zdravo razmišljate' (use sound thinking) kod oblikovanja svojih tabela, onda nećete imati problema kod kasnijeg 'prikazivanja' svojih podataka". Autorima ove "teorije oblikovanja baze podataka" oprostiti ćemo na "nepodnošljivoj lakoći" (i kratkoći) njihove teorije, jer je knjiga "Understanding Oracle" ipak prije svega priručnik za sistem Oracle. (No, takva "teorija" zacijelo može imati (a po svemu sudeći i ima!) nepovoljan utjecaj na modele podataka koje oblikuju njeni čitatelji!)

U cilju izbjegavanja mogućih (a nepotrebnih) "terminoloških prijepora", pogledajmo najprije (uobičajeno) značenje samog pojma "model podataka".

Prema Ullmanu, <Ull 88>, modelom podataka nazivamo "matematički formalizam" sačinjen od dva dijela:

1. Notacije (i metode) za opis *forme* podataka.
2. Skupa operacija za rukovanje *podacima*.

Obzirom da u ER jeziku nisu definirane "operacije" (niti se taj jezik eksplicitno bavi samim podacima, već samo njihovim formama) Ullman zaključuje da možemo tvrditi kako "ER model" zapravo i nije "model podataka".

Formalno, Ullmanova primjedba stoji, jer ER jezik ne ispunjava uvjet 2. Stoga ovdje i preferiramo izraz "ER jezik" (kojim se opisuju *forme* entiteta iz "fragmenta realnog svijeta" o kojem želimo "bilježiti neke podatke"). Utoliko bi i zapise u ER jeziku valjalo zvati "modelima *forme* podataka" (što ER model, zapravo, i jeste). No, radi jednostavnosti izražavanja, u nastavku koristimo izraz "ER model podataka" (kako je to, uostalom, u literaturi i uobičajeno).

Analiza i prijedlozi koje iznosimo u nastavku, uvelike temelje na iskustvima primjene ER jezika u okviru projektiranja baza podataka za potrebe turističke privrede (te su, stoga, iz tog problemskog područja uzimani i primjeri).

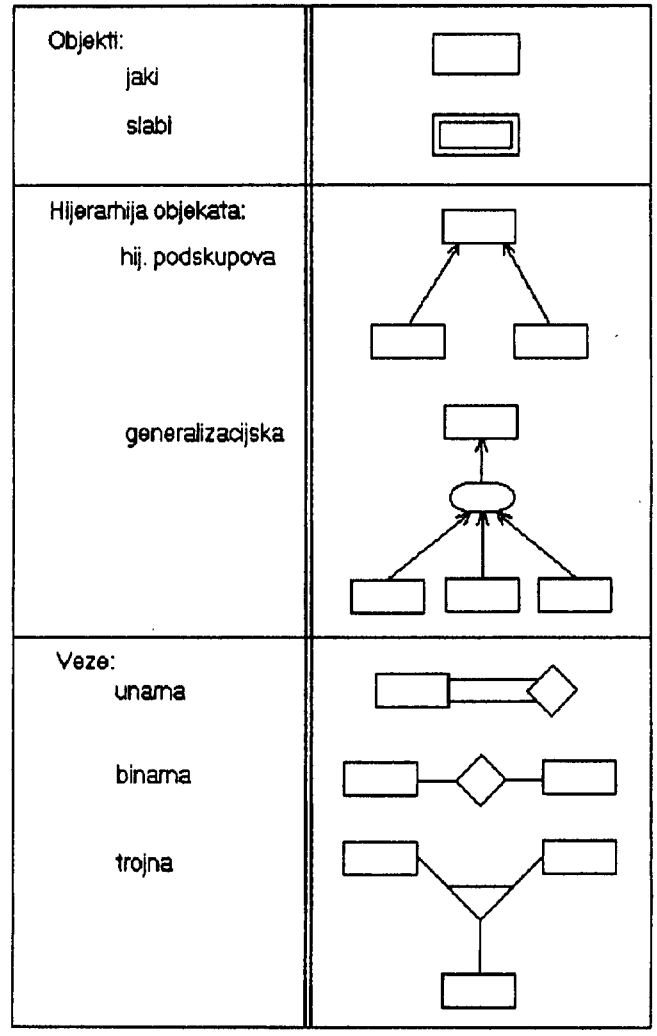
Polazeći od tih iskustava - te od "standardnog" ER jezika (datog u npr. <Teo 86>, <Dat 90> ili <Rad 91>) - ovdje predlažemo ono što držimo *stvarnom potrebom* a ne (samo) hipotetičkom *mogućnošću* proširenja ER jezika.

2. O ER jeziku

Polaznu osnovu ER jezika možemo izreći otprilike slijedećim riječima: Podaci su znanja o *objektima*, *vezama* (među tim objektima) i *svojstvima* (objekata odnosno veza); stoga, cilj jezika za predstavljanje strukture podataka jeste da omogući precizan i jednostavan zapis znanja tih triju temeljnih kategorija. (Napomenimo, da pojmom "entitet" ovdje ne označavamo samo "objekte" - kako se to često čini - već je to "generativni pojam" koji obuhvaća sve tri iznad navedene kategorije.)

Na Chenov prijedlog uslijedile su brojne dopune (proširenja) jezika, posebno uvođenjem n-arnih veza za n > 2, i hijerarhije entiteta. Obzirom na terminološki i notacijski nesklad pojedinih prikaza jezika, na slici 1 data je osnovna notacija ER jezika kakvu koristimo u ovom tekstu.

Pritom, imena objekata i veza upisujemo u sam simbol, a njihova svojstva zapisujemo pored simbola (kako to ilustriraju slijedeće slike). Isto tako, u simbol upisujemo i "svojstvo generalizacije" (kod generalizacijske hijerarhije). Identifikatore objekata podcrtavamo.



Slika 1

Prema Thompsonu, <Tho, 89>, ER jezik je "izuzetno koristan za *razumijevanje i oblikovanje* baze podataka ... Njegova temeljna odlika

jeste u tome što "prekida" sa razmišljanjem na razini "zapisa" (records)", i okreće se ka *entitetima* sistema. Naime, " ... zapisi su sasma beskorisni (na razini oblikovanja baze), dok su entiteti i veze (među njima) "obećavajući početak". Stoga je od suštinskog značaja "zahvatiti" upravo njih". U svom vrlo elokventnom stilu (više efektnom nego efikasnom), Thompson zaključuje: "Model podataka živi ili umire 'od lakoće' kojom se dade shvatiti". I koristiti! - dodali bismo. A dosadašnjih 15 godina široke primjene ER jezi- ka pokazuje da ER jezik - "živi". Čini se da bismo iz toga onda smjeli izvesti i zaključak o "lakoći njegova shvaćanja i primjene".

ER jeziku pripisuju se odlike i kao komunikacijskom jeziku. "ER jezik pokazao se najuspješnijim kao *sredstvo komuniciranja* između projektanta i korisnika u toku analize (sistema) i oblikovanja baze podataka", nalazimo u <Teo 86>. Ta odlika, prema Teorey et al, proizlazi iz njegove "okrenutosti entitetima" (koju ističe i Thompson), te jednostavnosti samog jezika.

Ostajući uvelike suglasni sa iznijetim pozitivnim ocjenama ER jezika, u ovom članku želimo istaći njegove odlike kao sredstva za *zorno dokumentiranje* strukture baze podataka, ali i neka ograničenja koja "dolaze" iz nemogućnosti prikaza različitosti "porijekla" i "procesnih uloga" pojedinih entiteta modela.

Iznijeti prijedlozi proširenja standardnog ER jezika imaju za cilj otklanjanje tih ograničenja (nedostataka).

3. Izvorni i izvedeni entiteti

Prvi "raskorak" između ER jezika "na razini načela" i "na razini primjene" dolazi odtuda što, u praksi, među entitetima sistema postoji znatna razlika u prirodi nastanka (formiranja) i ulozi koju imaju u modelu (baze) podataka (odno- sno u procesu obrade podataka). S tog aspekta promatranja, držimo primjerenim (i potrebnim) entitete (tj., u samoj bazi: "tabele" i "kolone") podijeliti na slijedeći način:

Svojstva: *izvorna* i *izvedena*
Objekti i veze: *izvorni*, *izvedeni* i *kombinirani*.

Izvedenim (izračunatim) svojstvima nazvali smo ona svojstva čije se vrijednosti izračunavaju (izvode) iz vrijednosti drugih svojstava (istog ili pak nekih drugih objekata).

Izvornim svojstvima nazvali smo ona svojstva objekata koja nisu izvedena.

Izvedenim objektima (vezama) nazvali smo one objekte (veze) čija su sva svojstva izvedena.

Izvornim objektima (vezama) nazvali smo one objekte (veze) kod kojih nijedno svojstvo nije izvedeno. Takve objekte često nazivamo i matičnima.

Kombiniranim objektima (vezama) nazvali smo one objekte (veze) koji sadrže bar jedno (ali ne sve!) izvedeno svojstvo.

3.1. Izvedena svojstva

Jednostavan primjer izvedenog svojstva imali bismo kod (slabog) objekta STAVKA, sa svojstvima: ... KOLIČINA, CIJENA, IZNOS.

Očito, svojstvo IZNOS izvedeno je svojstvo jer se njegova vrijednost izračunava iz vrijednosti svojstava CIJENA i KOLIČINA. A to bi onda ujedno značilo da na relaciji dobivenoj "prijevodom" tako definiranog objekta STAVKA, postoji (i) funkcijska zavisnost

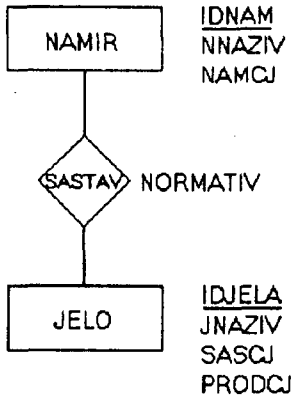
CIJENA,KOLIČINA --> IZNOS

Teorijski gledano, entiteti ne bi trebali (a ni smjeli!) sadržavati takova (izračunata) svojstva. Naime, takva svojstva, zapravo (u buduću tabelu) uvode redundantna polja, a time - u pravilu - i funkcijske zavisnosti koje nisu od ključa.

Prihvatimo da je u promatranom primjeru (i praktički gledano) odista nepotrebno eksplicitno "čuvati" vrijednosti svojstva IZNOS jer je CPU vrijeme za njegovo izračunavanje zanemarivo

malo u odnosu na vrijeme dostupa do samog zapisa u kojem su sadržane vrijednosti za njegovo izračunavanje. Drugim riječima, dostup do "iznosa" neznatno varira u zavisnosti da li ga samo čitamo ili ga i izračunavamo.

Međutim, sa aspekta ER modela, zanimljivijim izgleda problem prikaza izvedenosti onih svojstava čije se vrijednosti izračunavaju na temelju vrijednosti svojstava *drugih* objekata. Takav primjer dat je na slici 2.



Slika 2

Relacijski zapis tog modela glasi:

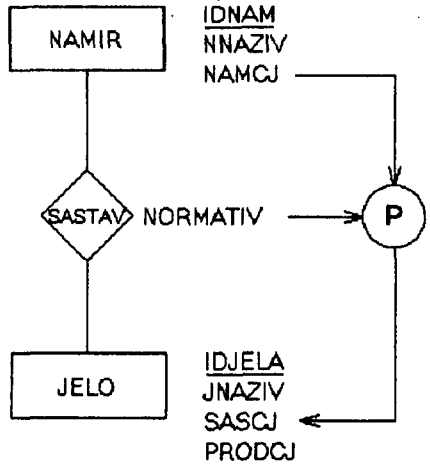
NAMIR(IDNAM,NNAZIV,NAMCJ)
SASTAV(IDNAM,IDJELA,NORMATIV)
JELO(IDJELA,JNAZIV,SASCJ,PRODCJ).

U ovom modelu, izvedeno svojstvo jeste SASCJ (cijena sastojaka (namirnica) u datom jelu). Vrijednost tog svojstva izračunava se iz vrijednosti (izvornog) svojstva NAMCJ (cijene namirnica) i izvornog svojstva NORMATIV veze SASTAV. Naime, normativ pokazuje količinu pojedine namirnice u pojedinom jelu.

Napomenimo da svojstvo PRODCJ (prodajna cijena jela) jeste izvorno svojstvo, jer se njegova vrijednost ne izračunava (algoritamski) iz cijene sastojaka (SASCJ). Naime, iako se vrijednost SASCJ svakako uzima kao osnova, ipak se vrijednost svojstva PRODCJ utvrđuje aproksimativno, i to u zavisnosti od niza faktora, poput energije i rada potrebnih za pripremu (i serviranje) jela, pouzdanosti ponude namirnica, potražnje jela, itd.

Vratimo se izvedenom svojstvu SASCJ. U razmatranom slučaju, pokazalo se potrebnim ("procesno opravdanim") uvođenje tog (redundantnog) svojstva u model podataka. Naime, to se svojstvo *često koristi*, i to ne samo za aproksimativno utvrđivanje prodajne cijene jela, već i kod raznih drugih kalkulacija. Bilo bi stoga neopravdano - zbog samog načelnog zahtjeva po neredundantnosti modela - to svojstvo izostaviti iz njega.

Međutim, ostavljajući po strani probleme redundance na razini relacijskog modela (tj. na razini "tabela podataka"), ovdje se postavlja pitanje da li i kako - u samom ER modelu - *eksplicitno prikazati* izvedenost svojstva i/ili način izračunavanja njegove vrijednosti. Drugim riječima, valja odlučiti o eventualnoj ekstenziji ER jezika uvođenjem (simbola) *procesa*. Na slici 3 dat je jedan prijedlog načina na koji se to može učiniti.



Slika 3

Čini se da bi takva ekstenzija jezika učinila model informativnijim (a da pritom ipak ne zađemo u DeMarcov dijagram toka podataka (DTP, <Rad 91>)). Međutim, kod opsežnijih modela, eksplicitnim prikazom procesne prirode svojstava (pose- bno ako izvedenih svojstava ima mnogo!) model bi mogao brzo postati nepreglednim. A upravo smo jasnoću i preglednost modela podataka isticali kao temeljnu odliku ER jezika

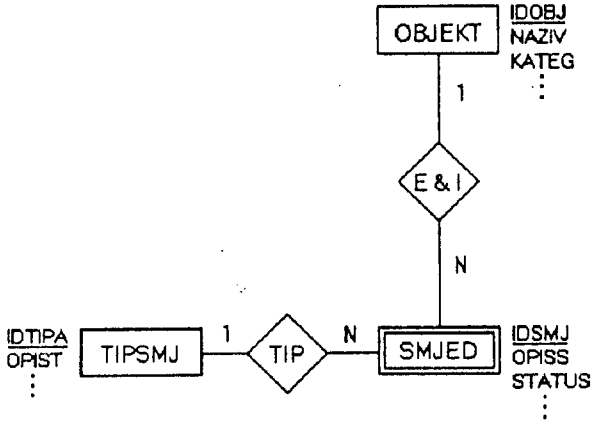
u ulozi sredstva za dokumentiranje. Stoga, prijedlog sa slike 3 - na razini izvedenih svojstava - navodimo samo kao *mogućnost*, ali ne i kao obavezan element ER jezika.

U nastavku, razmatramo potrebu (opravdanost) uvođenja simbola procesa na razini prikaza izvedenih objekata (veza).

3.2. Izvedeni objekti (veze)

Prema danim definicijama, objekt NAMIRnice i veza SASTAV jesu izvorni. Objekt JELO je kombiniran jer sadrži jedno izvedeno svojstvo (ali i dva izvorna svojstva).

No, čini se da posebnu pažnju zaslužuju izvedeni objekti (veze). Potrebu po uvođenju takvih entiteta analiziramo na primjeru situacije koju opisuje model podataka sa slike 4.

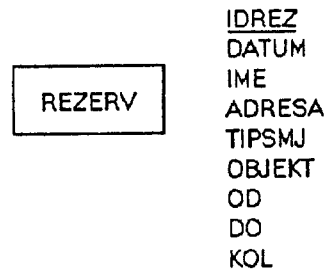


Slika 4

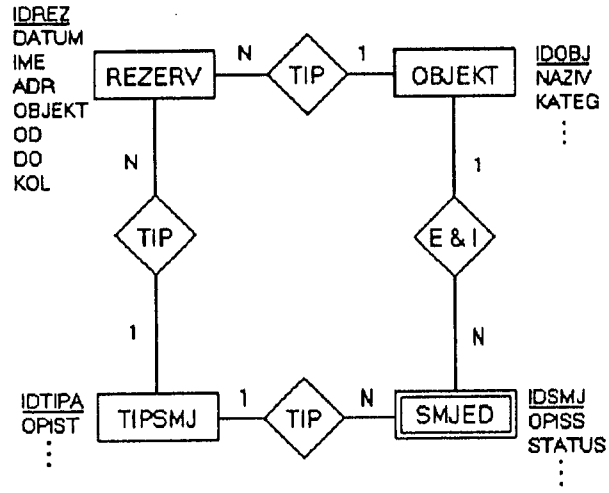
Model sadrži objekte: OBJEKT (hotel, turističko naselje, ...) SMJED (smještajna jedinica u okviru objekta) i TIPSMJ (tip smještajne jedinice, određen prema kapacitetu, komforu, ... pojedine smještajne jedinice).

Postavlja se pitanje da li takav model omogućava da se izvrši rezervaciju npr. 5 smještajnih jedinica datoga tipa. Uzmimo pritom da bi rezervacija mogla (morala) sadržavati barem svojstva data na slici 5.

Model sa slike 6 pokazuje na koji način je to *moguće* učiniti.



Slika 5

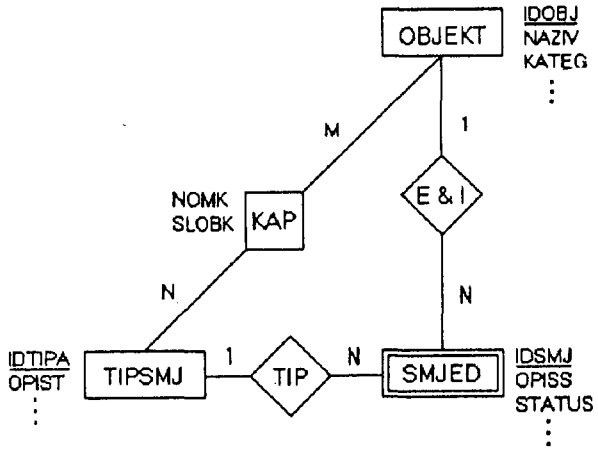


Slika 6

Međutim, već pokušaj provjere ispunjenosti prvog preduvjeta za *prihvaćanje* rezervacije - naime, postojanja slobodnih smještajnih jedinica! - ukazuje na izrazite slabosti takova rješenja. Naime, prema modelu podataka sa slike 6, uvid u "slobodne kapacitete" bio bi procesno vrlo zahtjevan. Jer postojanje slobodnih smještajnih jedinica (u datom objektu, datog tipa, u datom periodu) bilo bi moguće utvrditi jedino tako da se za dati objekt najprije izračuna broj smještajnih jedinica (datog tipa); da se zatim - iz tabele REZERVacije - utvrdi rezerviranost (zauzetost) jedinica toga tipa (za dati period), te da se "iz razlike" utvrdi postojanje i broj slobodnih smještajnih jedinica - i tek nakon toga izvrši rezervaciju (ako slobodni kapacitet to dopušta). Naravno, iako je takav postupka - u kontekstu datog

modela podataka - *moguć*, operativno ("procesno") promatrano, takav bi postupak izračunavanja *bio krajnje suboptimalan.*

Proces provjere prihvatljivosti (a i izvršenja) rezervacije dade se *bitno* pojednostaviti uvođenjem redundantnog entiteta (veze) KAPacitet. (Dakle, odstupanjem od jednog od temeljnih načela oblikovanja modela podataka, koje u <Teo 86> glasi: *"Redundantne veze trebaju biti eliminirane!"*) Pritom, kapacitetom nazivamo količinu smještajnih jedinica po tipovima i objektima. Dakle, KAPacitet može biti prikazan vezom (mnogo-mnogo), kako je to učinjeno na slici 7.



Slika 7

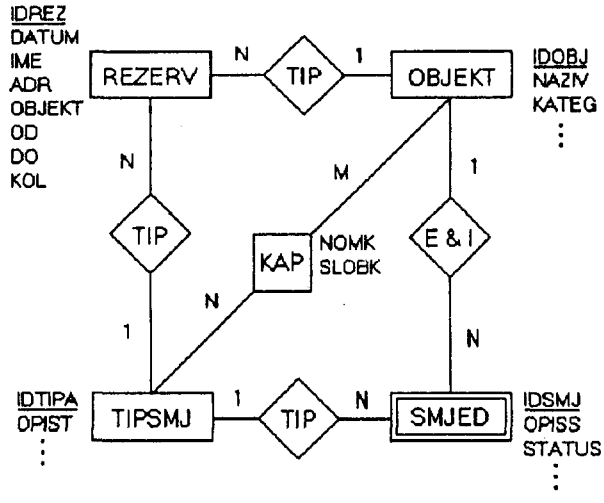
Napomenimo da je svojstvo NOMK (nominalni kapacitet) skalar, a da je svojstvo SLOBK (slobodni kapaciteti) vektor (sa 366 polja), pri *čemu svako sadrži broj slobodnih smještajnih* jedinica (datog tipa u datom objektu) za pripadni dan u godini. Rezervacija je sada, naravno, moguća ako - u traženom periodu - broj slobodnih smještajnih jedinica nije manji od broja traženih. A uz postojanje izvedene (redundantne) veze KAPaciteti, to je trivijalno utvrditi.

Naravno, prihvaćanje rezervacije uvjetuje i on-line ažuriranje vrijednosti svojstva SLOBK za dati period. (Napomenimo, da se i na razini smještajne jedinice vodi analogna evidencija (svojstvo STATUS), i to zbog omogućavanja rezerviranja točno određene smještajne jedinice. Nadalje, status zauzetosti sobe odnosno kapaciteta ne mijenja se samo na temelju rezervacije, *ali u te pojedinosti ovdje nema potrebe* zalaziti.)

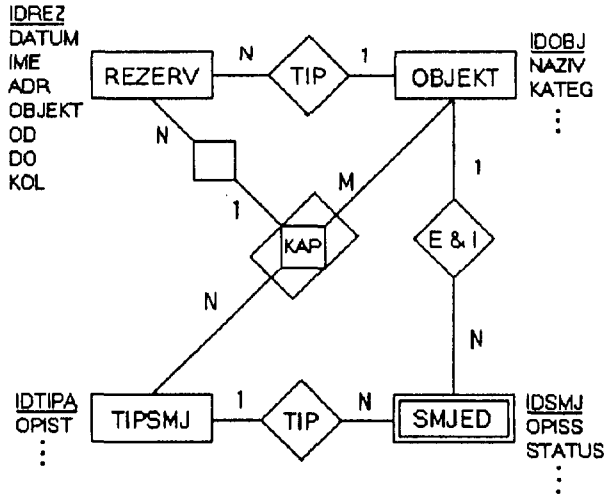
Na razini "zornog prikaza i dokumentacije", model podataka sa slike 7 pokazuje dvije slabosti.

1. Objekt REZERVacija ne možemo "vezati" na *vezu* KAPacitet. (Naime, veza može "stajati" samo između objekata.) Stoga smo prinuđeni postupiti na jedan od dva načina data na slici 8.

Za vezu rezervacije i kapaciteta sa slike 8a, možemo reći da je (u najboljem slučaju)



(a)



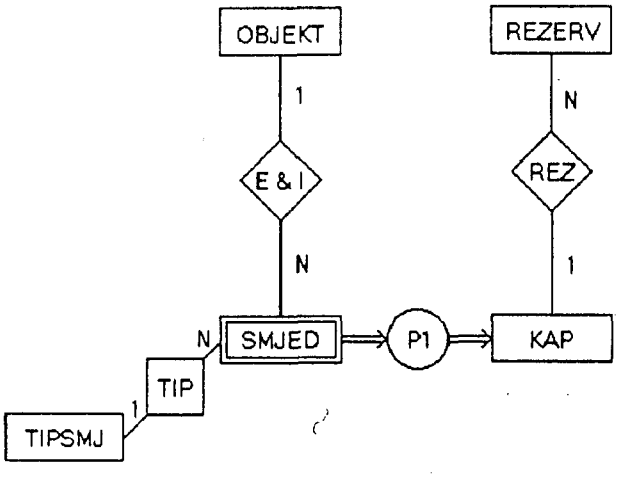
(b)

Slika 8

"implicitna". Naime, postojanje takve veze nipošto nije "očito" iz samog modela, već to postaje tek ako se "prisjetimo" da će se - kao vanjski ključevi - u shemi relacije REZERVacija naći svojstva IDOBJ i IDTIP, te da upravo taj par svojstava tvori identifikator (ključ) veze (relacije) KAPaciteti.

2. Uvođenje simbola *objekt-veza* (<Rad 91>), kako je to učinjeno na slici 8b, omogućuje eksplicitni pokaz vezanosti rezervacije na kapacitet, te stoga uvođenje takvog simbola držimo korisnim.

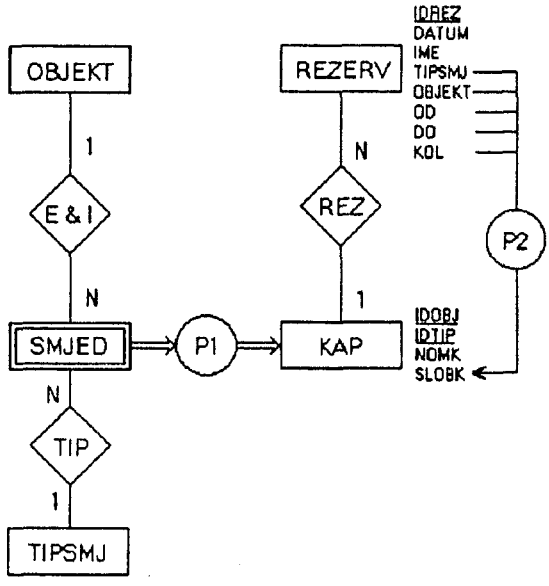
Međutim, nijedan od dvaju navedenih načina ne pokazuje *eksplicitno* da je entitet KAPaciteti *izvedeni* entitet! (Dakle, "formalno redundantan", ali "procesno potreban"!) Stoga, ovdje predlažemo uvođenje "procesnog simbola", koji je već "ispitivan" kod izvedenih svojstava, a čije uvođenje držimo sasma opravdanim kod izvedenih objekata odnosno veza. Naime, izgleda da se tim proširenjem ER jezika ne gubi ništa; model neće postati nepreglednijim, već upravo suprotno, kako to ilustrira slika 9.



Slika 9

Za izvedeni entitet KAPacitet možemo reći da - posredstvom procesa P - slijedi iz (slabog) objekta SMJED. Naime nominalni kapacitet (NOMK) - po smještajnim objektima i tipovima smještajnih jedinica - izračunava se na temelju svojstava objekta (budućih "kolona tabele") SMJED, što modeli sa slike 8 nisu eksplicitno pokazivali.

Razmotrimo i izvedeno svojstvo SLOBK (slobodni kapaciteti). Naime, učinivši "prvi korak" ka "procesualizaciji" modela podataka, postavlja se pitanje zašto ne učiniti i više. Naprimjer, zašto ne pokazati eksplicitno (u modelu) i "način utjecaja" rezervacije na slobodan kapacitet (SLOBK), kako je to učinjeno na slici 10?



Slika 10

Očito, ovakav način prikazivanja "porijekla i procesne prirode" izvedenih svojstava čini model podataka informativnijim. Međutim, u slučajevima gdje ima velik broj izvedenih svojstava, eksplicitan prikaz procesa (načina) njihova formira- nja, doveo bi do toga da model podataka postane nepreglednim.

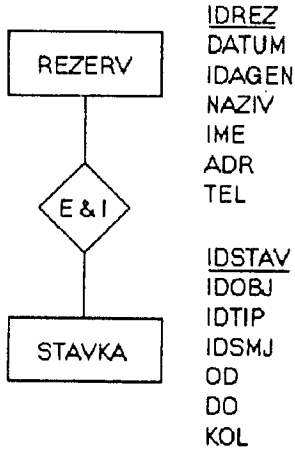
Stoga, uvođenje "procesnih opisa" na razini izvedenih *objekata/veza* držimo sasma opravdanim i korisnim. S druge strane, primjerenost uvođenja takova prikaza na razini *svojstava* zavisi od specifičnih osobina konkretnog modela podataka.

Konačno, želimo li imati oboje: informativnost i preglednost, model možemo prikazati na dvije razine detaljizacije. Dakle, na "global-

noj razini" (bez procesnih opisa za svojstva), i na "razini detalja" (sa takvim opisima).

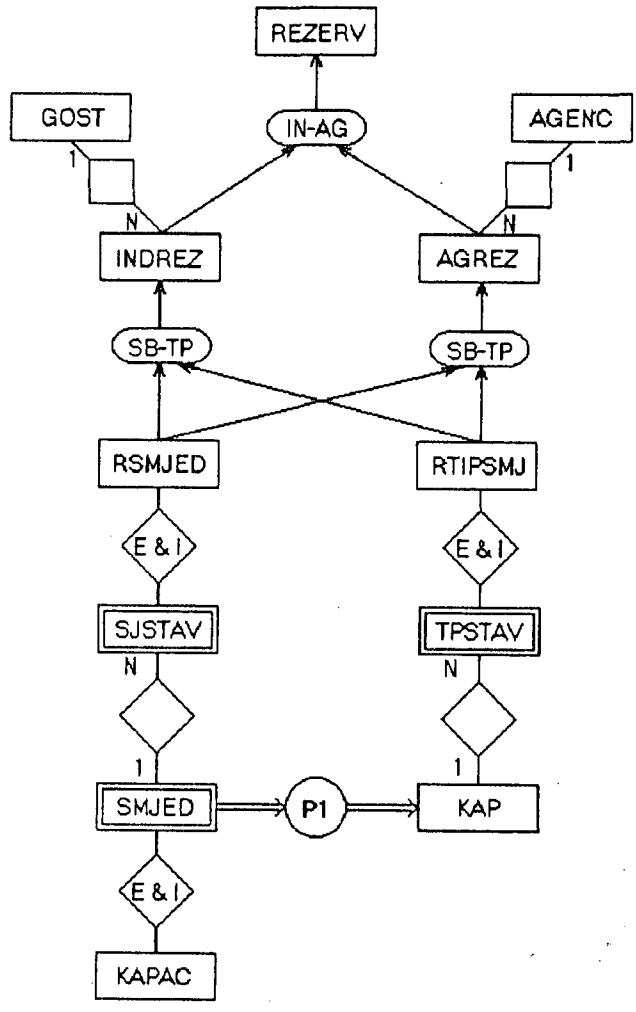
4. Veze i disjunkcija

Sistem rezervacija mora omogućavati rezervaciju ne samo datog tipa smještajne jedinice, već i točno određene smještajne jedinice. Modeli sa slika 6 - 10, očito to ne omogućavaju. Nadalje, rezerviranje može vršiti pojedinac ili agencija; isto tako, valja omogućiti da se jednom rezervacijom rezerviraju bilo razne smještajne jedinice (ili jedinice raznih tipova), bilo pojedinačnih smještajnih jedinica, i to za razna vremenska razdoblja. Naime, bilo bi neprikladno da se za jedno (npr. agencijsko) rezerviranje otvara toliko rezervacija (tj., da se toliko puta ponavlja pisanje "zaglavlja") koliko se (tipova) smještajnih jedinica rezervira. Stoga, objektu REZERVacija pridružujemo slabi objekt STAVKA, kako je to učinjeno na slici 11.



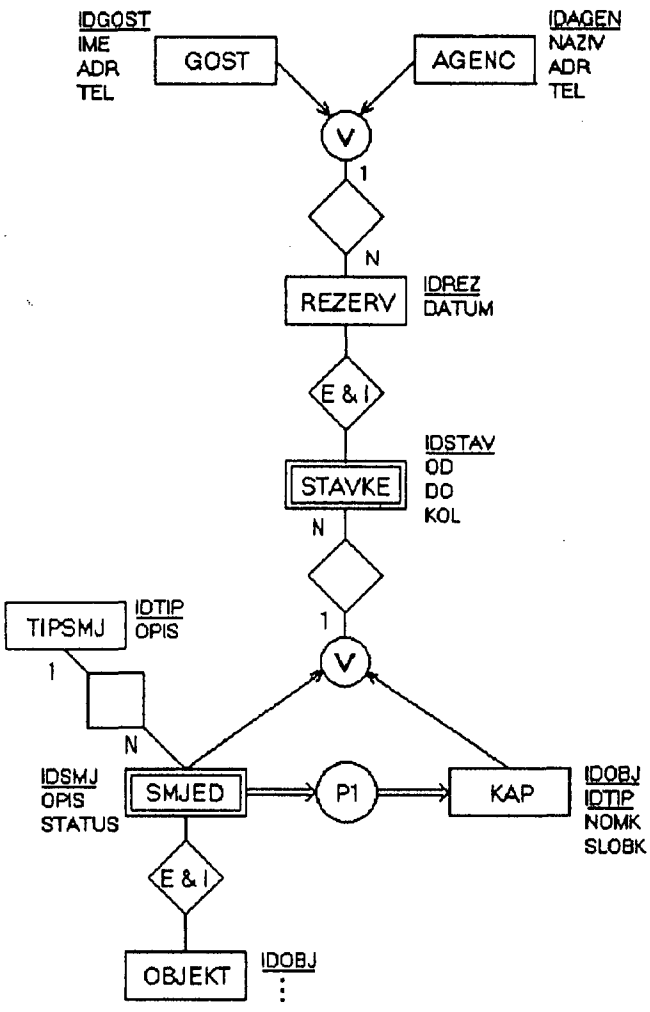
Slika 11

Očito, tako definirani objekti imaju niz svojstava koja se *međusobno isključuju*. Naprimjer, ako rezervaciju vrši agencija, onda je svojstvo IME isključeno ("neprimjenljivo"); jednako neprimjenljivima bivaju svojstva IDAGEN ("šifra" agencije) i NAZIV kada rezerviranje vrši individualni gost. Analogno vrijedi za STAVKE: ako se rezervira tip (IDTIP), onda je broj smještajne jedinice (IDSMJ) neprimjenljiv. U suprotnom, rezervacija točno određene sobe isključuje *potrebu* po eksplicitanju tipa. U takvim situacijama - prema načelu oblikovanja modela podataka - uvode se *podtipovi* (primjereni (disjunktnim) podskupovima instanci datog tipa entiteta). Slijedeći rigorozno to načelo, sačinili bismo model podataka dat na slici 12.



Slika 12

No, takav model ne bi se nipošto odlikovao preglednošću. Nadalje, (nepotrebno) veliki broj relacija (tabela) koje bi iz njega nastale ne bi predstavljao optimalno rješenje ni sa procesnog, a ni sa čisto operativnog (praktičkog) aspekta. Stoga, ovdje predlažemo pojednostavljenje modela (i njegova prikaza!) uvođenjem (simbola) *disjunkcije* na razini veze, kako je to pokazano na slici 13.



Slika 13

Prijevod takova ER modela glasio bi:

GOST(IDGOST,IME,ADR,TEL)
AGENC(IDAGEN,NAZIV,ADR,TEL)
RERERV(IDREZ,DATUM,IDAGEN,IDGOST)
STAVKE(IDREZ,IDSTAV,IDOBJ,IDTIP,
IDSMJ,OD,DO,KOL)
SMJED(IDOBJ,IDSMJ,OPIS,STATUS)
KAPAC(IDOBJ,IDTIP,NOMK,SLOBK)
TIPSMJ(IDTIP,OPIS)
OBJEKT(IDOBJ, ...)

Promatrano sasma operativno, prilikom otvaranja rezervacije, mogao bi biti uspostavljen neki fiktivni indentifikator (npr. redni broj rezervacije) jer taj identifikator nema neke izravne praktičke primjene. Nadalje, nakon preuzimanja datuma, ako se radi o individualnom gostu onda se IDAGEN "preskače" (neprimjenljivo svojstvo), te se uzimaju podaci o gostu. (Način i razlozi (u)vođenja i (trajnog) čuvanja zapisa o (potencijalnom) gostu ovdje su nevažni). U suprotnom, postupa se analogno sa agencijom. (Napomenimo, da se iz "praktičkih razloga" - pored IDAGEN / IDGOST - u rezervaciju, u pravilu, zapisuju i redundantna svojstva poput naziv/ime, adresa, telefon, ...)

Slijedi upisivanje (jedne ili više) stavki, pri čemu IDSTAV može biti jednostavno redni broj stavke u rezervaciji. Ako se vrši rezervacija na razini kapaciteta, onda broj sobe (IDSMJ) ostaje "prazan"; u suprotnom, ispunjavaju se sva polja, s time da KOLičina biva "1".

Dakle, čini se da uvođenje disjunkcije veza s jedne strane daje znatno pregledniji model poda- taka, a s druge strane (implicitno) odražava i sam proces vršenja rezervacije. Naravno, to nipošto ne znači da model podataka "nameće" detalje same *procedure* korištenja informacijskog sistema (npr. redoslijed operacija kod rezerviranja). Procedura rezerviranja može se zacijelo odvijati i drukčije (npr. pitanje što se i gdje rezervira može prethoditi pitanju tko rezervira), pri čemu model podataka ostaje, naravno, sasma nezavisnim od operativnih postupaka takove vrste.

Naravno, specifičnosti mjesta na kojem se vrši rezerviranje (npr. centralna recepcija, ili pak recepcija pojedinog objekta (hotela)) mogu zahtijevati prilagodbu modela. Naprimjer, ako se rezervacija može vršiti samo za jedan objekt (na recepciji tog objekta), onda podaci o objektu zacjelo neće ulaziti u stavke. U tom slučaju, OBJEKT bi - prema potrebi - mogao biti "vezan" na "zaglavlje rezervacije" (tj. na objekt REZERV).

5. Zaključak

Poput svakog formalnog sistema, jezik za oblikovanje modela (baze) podataka "kreće se" između dvaju - u pravilu, protustavljenih - zahtjeva: *po jednostavnosti* i *po "izražajnosti"* (tj. po "semantičkom bogatstvu"). U ovom članku ukazali smo na potrebu i opravdanost proširenja

ER jezika uvođenjem dvaju novih simbola: *simbola procesa* i *simbola disjunkcije*. Pokazali smo kako ti simboli povećavaju mogućnosti (a i jednostavnost!) oblikovanja modela podataka pomoću ER jezika. Naravno, svjesni smo da uvođenje novih elemenata (posebno *procesnih*) može "zasjeniti" sam model *podataka*. No, mnijenja smo da ovdje iznijeti prijedlozi to ne čine, već da prije doprinose njegovoj preglednosti.

REFERENCE

<Bra 87> Brackett, H.M.:
Developing Data Structured Databases, Prentice-Hall, 1987.

<Che 76> Chen, P.P:
The Entity-Relatinship Model: Toward a Unified View of Data, *ACM Trans. on Database Systems*, No. 1, 1976.

<Dat 90> Date, C.J.:
An Introduction to Database Systems, Vol. 1, Addison-Wesley, 1990.

<Fur 86> Furtado, L.A., Neuhold, J.E.:
Formal Techniques for Data Base Design, Springer-Verlag, 1986.

<Haw 88> Hawryszkiewycz, I.T.:
Systems Analysis and Design, Prentice Hall, 1988.

<Mar 87> Martin, J.:
Recommanded Diagramming Standards for Analysts & Programmers: A Basis for Automation, Prentice-Hall, 1987.

<Nav 86> Navathe, S,, Elmasri, R., Larson, J.:
Integrating Users Views in Database Design, *IEEE Computers*, No. 1 1986.

<Per 89> Perry, T.J., Lateer, G.J.:
Understanding Oracle, Sybex, 1989.

<Rad 89> Radovan, M.:
Modeliranje podataka: ER jezik i normalne forme, *Informatica*, No. 1, 1989.

<Rad 91> Radovan, M.:
Projektiranje informacijskih sistema, Informator, 1991.

<Teo 86> Teorey, 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.

<Tho 89> Thompson, J.P.:
Data with Semantics, Van Nostrand Reinhold, 1989.

<Ull 88> Ullman, D.J.:
Principles of Database and Knowledge-base Systems, Vol. 1, Comp. Sci. Press, 1988.

ALGORITEM ZA DOLOČITEV POVEZOVALNIH FUNKCIJ KRIVULJ B-ZLEPKOV IN NURB KRIVULJ V POLINOMSKEM ČASU

INFORMATICA 3/91

Keywords: B-spline curves, NURB curves, blending functions, de Boor algorithm, time complexity, computer graphics

Borut Žalik, Nikola Guid,
Andrej Tibaut in Aleksander Vesel
Tehniška fakulteta Maribor

Povzetek: Povezovalne funkcije krivulj B-zlepkov in NURB krivulj so definirane z rekurzivno formulo, čigar direktna vgradnja v algoritem pripelje do eksponentne časovne zahtevnosti izračunavanja povezovalnih funkcij glede na njihov red. S pravilno pripravo podatkovnih struktur in upoštevanjem lastnosti povezovalnih funkcij lahko sestavimo algoritem, ki izračuna povezovalne funkcije in nariše krivuljo v polinomskem času. Izdelan algoritem smo primerjali glede na porabljen čas CPU z iterativnim de Boorovim algoritmom, s katerim določimo točke na krivulji brez izračunavanja povezovalnih funkcij. Ugotovimo, da je naš algoritem učinkovitejši celo od de Boorovega algoritma v primeru periodičnih in neperiodičnih krivulj B-zlepkov, v primeru NURB krivulj pa je slabši.

An algorithm for B-spline and NURB blending functions determination in a polynomial time. B-spline and NURB blending functions are defined by recursive formula. Its direct implementation into a program lead us to the exponential time complexity regarding to the degree of blending functions. With paying attention to a correct data structure we have been able to construct an algorithm which evaluates the blending functions and plots the curve in a polynomial time. This algorithm has been compared with iterative de Boor algorithm which determines the points on a curve without blending function calculation. The values of the CPU time spend on curve generation show, that our algorithm is better in the case of periodical and nonperiodical B-spline curves, and in the case of NURB curves it is worse than de Boor algorithm.

1. UVOD

Pri realizaciji algoritmov za izračun in izris krivulj B-zlepkov se srečamo s problemom izračunavanja povezovalnih ali baznih funkcij. Za hiter izris krivulj B-zlepkov in NURB krivulj sicer obstaja iterativni de Boorov algoritem, kjer povezovalnih funkcij sploh ni potrebno izračunavati, vendar je mnogokrat zelo ugodno, če lahko prikažemo tudi povezovalne funkcije, predvsem zaradi učenja in raziskovanja. Seveda je najugodneje, če so povezovalne funkcije izračunane analitično in zakodirane v algoritmu (bazne funkcije so polinomi ustreznega reda). Periodične bazne funkcije nižjih redov (2 ali 3) lahko hitro izpeljemo, najdemo pa jih tudi v literaturi. V primeru uporabe neperiodičnih baznih funkcij moramo vložiti že več napora, da izpeljemo vse potrebne povezovalne funkcije. Le-te že redkeje najdemo (npr. v [TURK80] in [YAMA88]). Določitev koeficientov povezovalnih funkcij višjih redov pa zahteva precej truda.

2. DEFINICIJA KRIVULJ B-ZLEPKOV

Definicijo in lastnosti krivulj B-zlepkov najdemo na mnogih mestih v literaturi ([BÖHM84], [MORT85], [BART87], [GUID88], [PIEG87], [YAMA88]). V nadaljevanju bomo uporabljali oznake in zaključke iz vira [GUID90]. Krivulja B-zlepkov je definirana kot:

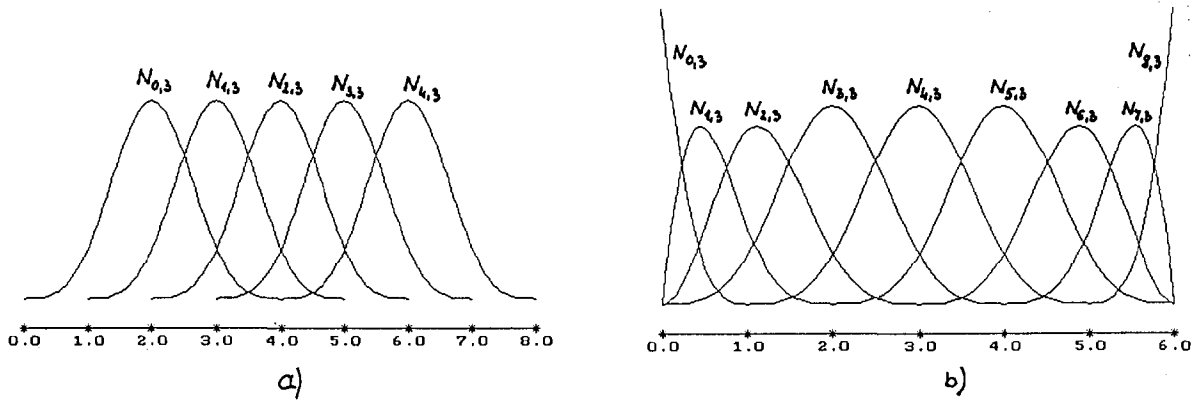
$$p(u) = \sum_{i=0}^{n} r_i \, N_{i,k}(u) \qquad (1)$$

kjer so r_i kontrolne točke in $N_{i,k}$ bazne ali povezovalne funkcije reda k, ki so definirane z rekurzivno formulo:

$$N_{i,0}(u) = \begin{cases} 1, & u_i \le u < u_{i+1} \\ 0, & \text{sicer} \end{cases}$$

in

$$N_{i,k}(u) = \frac{(u - u_i)\, N_{i,k-1}(u)}{u_{i+k} - u_i} + \frac{(u_{i+k+1} - u)\, N_{i+1,k-1}(u)}{u_{i+k+1} - u_{i+1}} \qquad (2)$$



Slika 1 Kubične bazne funkcije, ki jih določa vozliščni vektor
a) $U_{knot} = [0, 1, 2, 3, 4, 5, 6, 7, 8]$
b) $U_{knot} = [0, 0, 0, 0, 1, 2, 3, 4, 5, 6, 6, 6, 6]$

Množico u_i shranimo v vozliščni vektor $U_{knot} = [u_0, u_1, u_2, \ldots, u_m]$. Obravnavali bomo dve obliki vozliščnega vektorja:

a) **Vozliščni vektor, ki določa nesklenjene periodične krivulje B-zlepkov.** Za nas so zanimivi le tisti odseki, kjer sodeluje k+1 povezovalnih funkcij, saj je krivulja B-zlepkov definirana v intervalu $u \in [u_0 + k, u_m - k]$. Definiran je kot [GUID90]:

$$U_{knot} = [u_0, u_1, u_2, \ldots, u_m] = [0, 1, 2, \ldots, m]$$

m je določen s formulo:

$$m = n + k + 1 \qquad (3)$$

kjer je:

n: število kontrolnih točk minus 1

k: red povezovalnih funkcij.

Bazne funkcije za n = 4 in k = 3 nam kaže slika 1a.

b) **Vozliščni vektor, ki določa neperiodične krivulje B-zlepkov.** Definiran je kot [GUID90]:

$$U_{knot} = [u_0, u_1, u_2, \ldots, u_m]$$

$u_i = 0$; če $i \le k$

$u_i = i - k$; če $k+1 \le i \le m - k - 1$

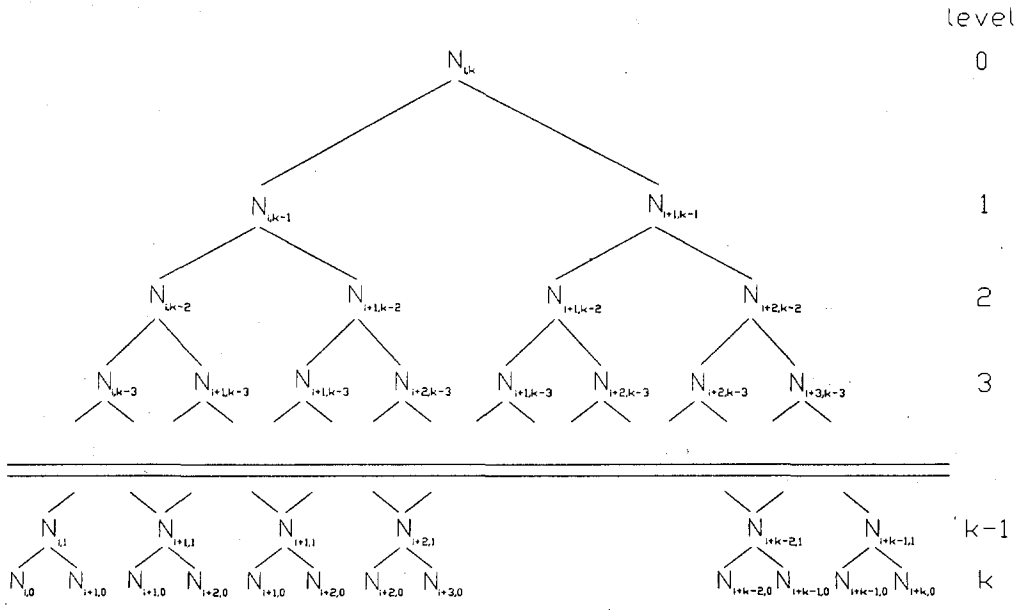
$u_i = m - 2k$; če $m - k \le i \le m$

kjer je m definiran z enačbo 3. Bazne funkcije za n = 8 in k = 3 nam kaže slika 1b.

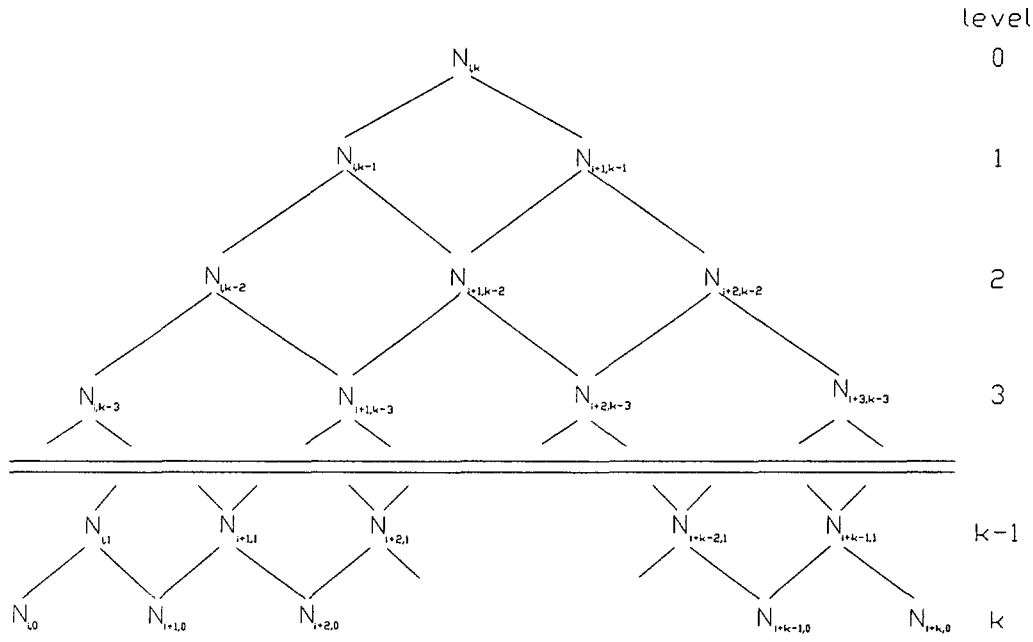
3. PRIPRAVA PODATKOVNE STRUKTURE ZA DOLOČITEV POVEZOVALNIH FUNKCIJ

Potek izračunavanja povezovalnih funkcij z algoritmom, ki neposredno rešuje enačbo 2, nam kaže slika 2. Rekurzivni klici nam ustvarijo polno dvojiško drevo. Število vozlišč v takem drevesu je določeno z enačbo

$$\sum_{i=0}^{k} 2^i = 2^{k+1} - 1. \qquad (4)$$



Slika 2 Drevo rekurzivnih klicev funkcije $N_{i,k}$



Slika 3 Modificirana struktura za izračun funkcije $N_{l,k}$

2^k vrednosti povezovalnih funkcij na globini k določimo brez računanja (enačba 2), vse preostale $2^k - 1$ vrednosti povezovalnih funkcij pa moramo izračunati, kar vodi k eksponencialni časovni zahtevnosti

$$T(k) = 2^k - 1 = O(2^k). \qquad (5)$$

Če pa boljše pogledamo sliko 2, vidimo, da se nekatere povezovalne funkcije, shranjene v vozliščih drevesa, ponavljajo. Z združevanjem vozlišč, ki hranijo iste povezovalne funkcije, dobimo novo strukturo, ki jo vidimo na sliki 3. Število vozlišč v taki strukturi je določeno z enačbo

$$\sum_{i=0}^{k} (1 + i) = \frac{1}{2} (k^2 + 3k + 2). \qquad (6)$$

Tudi tokrat k + 1 začetnih vrednosti povezovalnih funkcij na globini k določimo brez računanja, tako da nam ostane le $\frac{1}{2}$ k(k + 1) izračunov, ki pa jih lahko opravimo v polinomskem času:

$$T(k) = \frac{1}{2} k(k+1) = O(k^2) \qquad (7)$$

Da bi lahko določili vrednost povezovalne funkcije glede na izbiro parametra u v polinomskem času, moramo pripraviti ustrezno podatkovno strukturo. Funkcijo na globini j ($0 \le j < k$) izračunamo s pomočjo dveh funkcij iz globine j+1. Ugodno bi bilo našo shemo s slike 3 obrniti tako, da bodo prej dostopna vozlišča z večjo globino. Podatkovno strukturo tvorimo kot seznam enosmerno povezanih seznamov, kjer imamo na vrhu k+1 vozlišč s povezovalnimi funkcijami reda 0 in na dnu samo eno vozlišče reda k. Strukturo deklariramo na naslednji način:

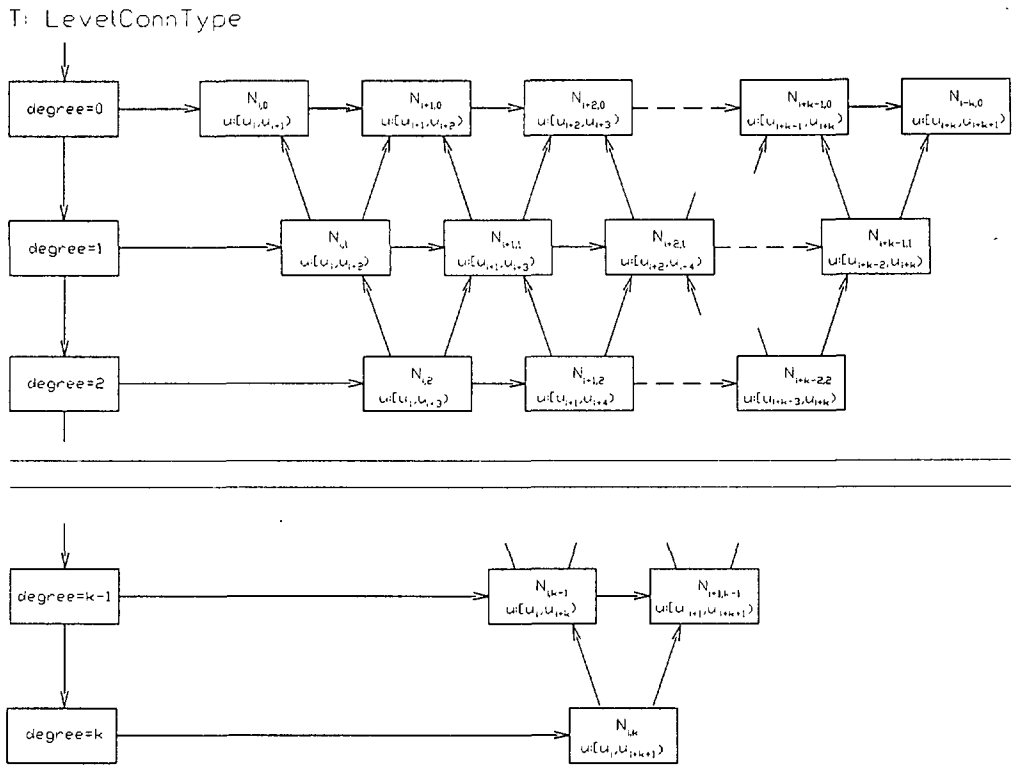
```
TreePointer = ^TreeType;
TreeType = record
   Ni: integer;                  {indeks povezovalne funkcije}
   LeftValue,                    {spodnja meja določena z Uknot}
   RightValue,                   {zgornja meja določena z Uknot}
   LDenominator,                 {imenovalec levega izraza}
   RDenominator,                 {imenovalec desnega izraza}
   Nu: real;  {vrednost povezovalne funkcije pri izbranem u}
   PermanentLL, {zastavica na globini 0; če TRUE,potem Nu=0}
   Permanent: Boolean;           {če TRUE, izračun Nu ni potreben}
   Lson, Rson,                   {kazalca na zapise višje v strukturi}
   list: TreePointer;            {kazalec na naslednji seznam}
end; {record TreeType}

LevelConnType = ^ListOfPointers;
ListOfPointers = record
   ToLevel: TreePointer;         {kazalec na seznam TreePointer}
   Nk: Integer;                  {stopnja povezovalne funkcije}
   NextLevel: LevelConnType;     {kazalec na naslednji zapis}
end; {record ListOfPointers}
```

Slika 4 nam kaže uporabljeno podatkovno strukturo.

Slika 4 Podatkovna struktura za izračun povezovalnih funkcij v polinomskem času

Inicializacija podatkovne strukture

S tako pripravljeno podatkovno strukturo, ki jo vidimo na sliki 4, lahko izračunamo poljubno povezovalno funkcijo glede na postavljeni vozliščni vektor, le prave podatke moramo vpisati v posamezne zapise. Podatkovno strukturo začnemo polniti pri zapisih, ki hranijo povezovalne funkcije stopnje 0 in so na vrhu seznama seznamov. Najprej za vsako povezovalno funkcijo na globini 0 preverimo mejni vrednosti iz vozliščnega vektorja in s tem določimo interval, v katerem je posamezna povezovalna funkcija definirana. Če sta obe vrednosti enaki, postavimo zastavico PermanentLL na TRUE, vrednost povezovalne funkcije shranjene v komponenti Nu pa je 0. V primeru, ko se vrednosti LeftValue in



Slika 4 Podatkovna struktura za izračun povezovalnih funkcij v polinomskem času

RightValue razlikujeta, postavimo zastavico PermanentLL v FALSE, Nu dobi vrednost 1, vrednosti iz vozliščnega vektorja vpišemo v LeftValue ter RightValue in določimo konstanti v imenovalcih, ki ju shranimo v komponenti zapisa LDenominator in RDenominator. Nato napolnimo še preostale zapise v podatkovni strukturi s slike 4 tako, da se spuščamo po seznamu tipa LevelConnType. Če imata oba zapisa na višjem nivoju v podatkovni strukturi zastavico PermanentLL TRUE, dobi tudi zastavica pravkar obravnavanega zapisa vrednost TRUE, Nu pa vrednost 0, sicer pa postavimo zastavico PermanentLL v FALSE. V tem primeru določimo interval, v katerem ta povezovalna funkcija deluje, kot unijo dveh sosednjih intervalov z nižjega nivoja dostopnih preko kazalcev Lson in Rson, nato pa z njihovo pomočjo določimo še imenovalca.

Določitev vrednosti povezovalne funkcije pri izbranem parametru

Pri izbrani vrednosti parametra u v večini primerov sodeluje pri izračunu funkcije $N_{1,k}$ manjše število povezovalnih funkcij, kot smo jih označili z zastavico PermanentLL. Tako npr. na globini 0 sodeluje kvečjemu le ena povezovalna funkcija, ki ima po enačbi 2 vrednost 1. Zato v vsakem zapisu tipa TreeType uvedemo še zastavico Permanent, ki ima vrednost TRUE pri vseh funkcijah $N_{1,k}$, kjer velja $u \notin$ (LeftValue, RightValue). Seveda dobi zastavica Permanent takoj vrednost TRUE, če ima takšno vrednost tudi PermanentLL. Zastavice na globinah večjih kot 0 določimo v primeru, ko ima zastavica PermanentLL vrednost FALSE, z naslednjim logičnim izrazom

Permanet := Lson^.Permanent and Rson^.Permanent.

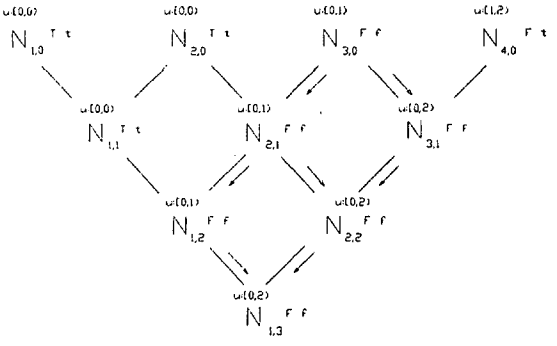
Postavitev zastavic spremenimo, ko tekoči parameter u spremeni interval v vozliščnem vektorju U_{knot}. Izračun vrednosti povezovalnih funkcij pri parametru u sedaj opravimo tako, da od globine 1 do k izračunamo le povezovalne funkcije v tistih zapisih, ki imajo postavljeno zastavico Permanent na vrednost FALSE.

Na sliki 5 vidimo primer, ko želimo izračunati neperiodično povezovalno funkcijo $N_{1,3}$ s slike 1b. Če je $u \in [0,1)$, moramo opraviti 5 izračunov (vrednost funkcije reda 0 je določena že vnaprej in je ne upoštevamo). Potek izračuna vidimo na sliki 5, če sledimo puščicam. Vrednost zastavice PermanentLL smo označili z velikima črkama T in F, vrednost zastavice Permanent pa z malima črkama t in f. Če je $u \in [1,2)$, bi morali opraviti le tri izračune. Na sliki 6 vidimo algoritem, ki uporablja opisano podatkovno strukturo, za izračun vrednosti povezovalne funkcije.

4. PRIPRAVA PODATKOVNE STRUKTURE IN ALGORITEM ZA IZRIS KRIVULJ B-ZLEPKOV

Priprava podatkovne strukture

Vsako povezovalno funkcijo bomo izračunali v diskretnih točkah. Naj bo na intervalu $[u_l, u_{l+1})$ dovolj NoOfSteps izračunov. Ker se vsaka povezovalna funkcija razteza največ na k+1 intervalih (slika 1a in 1b), bo za posamezno povezovalno funkcijo potrebnih največ (k+1) * NoOfSteps izračunov [ŽALI90]. Za vsako bazno funkcijo si shranimo še dejansko število izračunanih vrednosti. Vse podatke shranimo v zapis BfType.



Slika 5 Potek izračuna povezovalne funkcije $N_{1,3}$

```
BfType = record
           Bf: array [0..UValuesMax] of real;
           NoOfUvalues: integer;
end

UValuesMax ≥ (k+1) * NoOfSteps
```

Da bi lahko direktno uporabili enačbo 1 za določitev krivulj B-zlepkov, bomo vse funkcije $N_{i,k}$ predstavili kot kazalce na zapise tipa BfType in jih shranili v polje tipa AllBFunctionType:

```
AllBfunctionType = array [0..NoOfBf] of ^BfType.
```

Izračun in izris periodičnih krivulj B-zlepkov

Potrebujemo n+1 enakih periodičnih povezovalnih funkcij, ki so med seboj le premaknjene (slika 1a). Zato je dovolj, če izračunamo le eno povezovalno funkcijo in zanjo rezerviramo prostor, za vse ostale n bazne funkcije pa postavimo le kazalce na zapis tipa BfType. Potrebno podatkovno strukturo vidimo na sliki 7a.

Časovna in prostorska zahtevnost določitve povezovalnih funkcij sta neodvisni od števila kontrolnih točk, tako da lahko zapišemo:

$$T(n) = 1$$
$$S(n) = 1 \qquad (8)$$

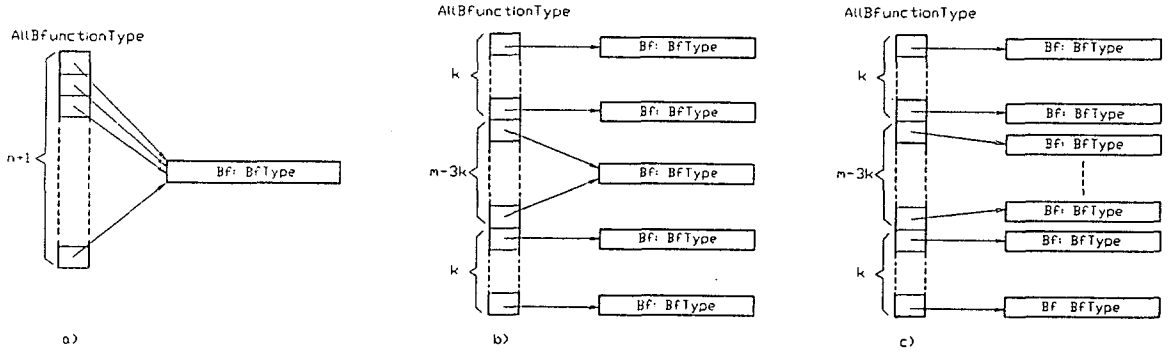
Funkcijo za izračun baznih funkcij pokličemo le ((k+1)NoOfSteps) - krat.

```
function ModifiedNf(t: LevelConnType; u: real): real;
{ Funkcija izračuna vrednost povezovalne funkcije krivulje
  B-zlepkov.
  - t: kazalec na podatkovno strukturo,
  - u: vrednost parametra, pri kateri bomo izračunali
                                        povezovalno funkcijo }
  var
    bufferNu: real;
    q: TreePointer;

begin (* ModifiedNF *)
  t := t^. NextLevel;
  while t <> nil do
  begin
    q := t^.ToLevel;
    while q <> nil do
      with q^ do
      begin
        if not permanet then
          Nu := LeftDenominator * (u - LeftValue) * Lson^.Nu +
                RightDenominator * (RightValue - u)* Rson^.Nu;
        end;
        BufferNu := Nu;
        q := q^.list;
    end;
    t := t^.NextLevel;
  end;
  ModifiedNf := BufferNu;
end; { ModifiedNF }
```

Slika 6 Algoritem za izračun vrednosti povezovalne funkcije

Odsek krivulje B-zlepkov narišemo tako, da upoštevamo le tiste povezovalne funkcije, ki na tem odseku nastopajo in jih zmnožimo s pripadajočimi kontrolnimi točkami. V začetku algoritma za izris krivulj B-zlepkov določimo, katere povezovalne funkcije sodelujejo pri odseku, ki ga trenutno rišemo (slika 8). Prvo in zadnjo povezovalno funkcijo, ki hkrati določata tudi indeks ustrezne kontrolne točke, shranimo v spremenjivki firstBf in lastBf. V drugem delu algoritma določimo še indeks diskretne vrednosti vsake povezovalne funkcije, shranjene v zapisu tipa BfType, ki jo pri izračunu točk na krivulji po enačbi 1 potrebujemo.

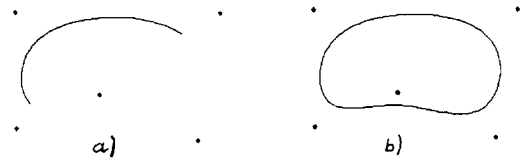
Nesklenjeno periodično krivuljo B-zlepkov, čigar povezovalne funkcije so na sliki 1a, vidimo na sliki 9a. Z majhno spremembo pri pripravi podatkov lahko s tem algoritmom narišemo tudi sklenjeno krivuljo B-zlepkov, ki jo vidimo na sliki 9b.



Slika 7 Podatkovna struktura baznih funkcij za:
a) periodične krivulje B-zlepkov
b) neperiodične krivulje B-zlepkov
c) NURB krivulje

```
Procedure PlotBSpl(k,m: integer; periodic: boolean;
                   Pts: AllBfunctionType; rx,ry: PointsType);
{  - k: red povezovalnih funkcij,
   - m: število vozliščnih vrednosti,
   - periodic: TRUE, če rišemo periodične krivulje B-zlepkov
   - Pts: polje kazalcev na povezovalne funkcije,
   - rx,ry: polje kontrolnih točk.
var
  x, y: PolyLineType;
   j, ui, ni, ind, firstBf, lastBf: integer;
  a: real;
begin
  firstBf := -1;                    lastBf := k - 1;
  for ui := 0 to m-2*k-1 do
  begin
    firstBf := firstBf + 1;         lastBf := lastBf + 1;
    for j := 0 to NoOfSteps do
    begin
      x[j] := 0;     y[j] := 0;
      for ni := firstBf to lastBf do
      begin
        if periodic then
          ind := (lastBf - ni) * NoOfSteps + j
        else
          if ni <= k then
            ind := (lastBf - k) * NoOfSteps + j
          else
            ind := (lastBf - ni) * NoOfSteps + j;
        a  :=  Pts[ni]^.Bf[ind];
        x[j] := x[j] + rx[ni] * a;
        y[j] := y[j] + ry[ni] * a;
      end;
    end;
    gPolyLine(NoOfSteps+1, x, y);
  end;
end; { PlotBspl }
```

Slika 8 Algoritem za izris krivulj B-zlepkov



Slika 9 a) Nesklenjena periodična krivulja B-zlepkov
b) Sklenjena periodična krivulja B-zlepkov

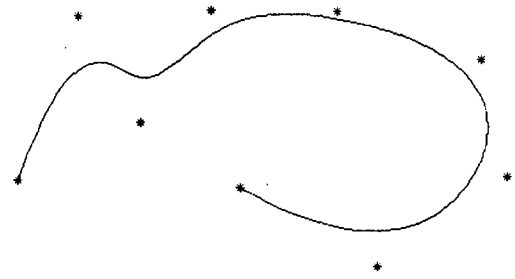
Izračun in izris neperiodičnih krivulj B-zlepkov

V primeru neperiodičnih krivulj B-zlepkov je število povezovalnih funkcij, ki jih moramo izračunati, večje. V najslabšem primeru moramo določiti k+1 povezovalnih funkcij, od katerih jih je k neperiodičnih in ena periodična. Na primeru s slike 1b, ko je k = 3 in n = 8 vidimo, da je prvih k baznih funkcij ($N_{0,3}$, $N_{1,3}$, $N_{2,3}$) zrcalnih zadnjim k baznim funkcijam ($N_{7,3}$, $N_{6,3}$, $N_{5,3}$). Zato je dovolj, če izračunamo le prve, druge pa dobimo z enostavno zamenjavo mest koeficientov v polju. Med neperiodičnimi baznimi funkcijami se pojavijo periodične povezovalne funkcije, če je n ≥ 2k [GUID90]. Kot smo že videli, pa te zahtevajo samo en izračun.

Tudi tokrat je prostorska in časovna zahtevnost določitve povezovalnih funkcij neodvisna od števila kontrolnih točk, tako da velja enačba 8. Funkcijo za izračun baznih funkcij v najslabšem primeru pokličemo

$$1*NoOfSteps + 2*NoOfSteps + ... + (k+1)*NoOfSteps =$$
$$= 0.5 * (k^2 + 3k + 2) * NoOfSteps.$$

Potrebno podatkovno strukturo vidimo na sliki 7b, ki jo uporablja algoritem za izris krivulje B-zlepkov (slika 8). Na sliki 10 vidimo primer neperiodične krivulje B-zlepkov, katere povezovalne funkcije so na sliki 1b.



Slika 10 Neperiodična krivulja B-zlepkov

5. DEFINICIJA NURB KRIVULJ

NURB krivulje ("nonuniform rational B-spline curves") so s svojo univerzalnostjo pritegnile mnogo raziskovalcev ([WAYN83], [BÖHM84], [PIEG87], [PIEG89], [ROGE90]). Z njimi lahko eksaktno predstavimo tako daljice, stožnice, kot poljubno oblikovane krivulje. NURB krivulje B-zlepkov uporabljajo tudi nekateri komercialni modelirni sistemi. Definirane so kot:

$$p(u) = \frac{\sum_{i=0}^{n} N_{i,k}(u)\, w_i\, r_i}{\sum_{j=0}^{n} N_{j,k}(u)\, w_j} = \sum_{i=0}^{n} R_{i,k}(u)\, r_i \qquad (9)$$

kjer so r_i kontrolne točke in $R_{i,k}$ racionalne povezovalne funkcije, ki so definirane s kvocientom:

$$R_{i,k}(u) = \frac{N_{i,k}(u)\, w_i}{\sum_{j=0}^{n} N_{j,k}(u)\, w_j} \qquad (10)$$

V enačbi 10 nastopajo bazne funkcije $N_{i,k}$, ki so rekurzivno definirane z enačbo 2. Z vsoto v imenovalcu izraza 10 funkcijo $R_{i,k}$ pri vsaki vrednosti parametra u normaliziramo. Množico vrednosti w_i, $w_i \geq 0$, $0 \leq i \leq n$ imenujemo uteži. Shranimo jih v vektor uteži $W = [w_0, w_1, ..., w_n]$. Število uteži je enako številu točk, tako da točko r_i utežimo z utežjo w_i. Vozliščni vektor je definiran podobno kot tisti, ki določa neperiodične krivulje B-zlepkov (poglavje 2):

$$U_{knot} = [u_0, u_1, ..., u_{m-1}, u_m],$$

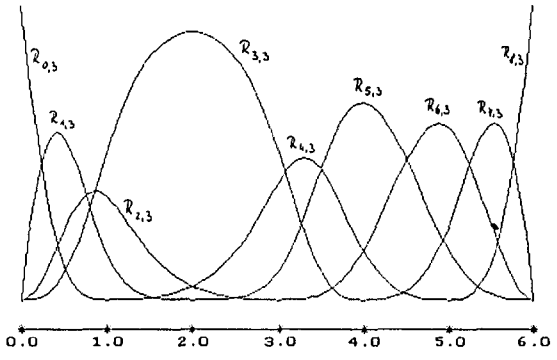
kjer velja: $u_i = 0$; če $i \leq k$ (a)

$u_i = m-2k$; če $m-k \leq i \leq m$ (b)

$u_i \geq u_k$; če $i > k$ in $k+1 \leq i,k \leq m-k-1$ (c)

Za vozle na intervalu (b) velja, da so lahko neuniformni. To pomeni, da razmaki med vozli niso nujno konstantni. Vrednosti sosednjih vozlov so lahko enake, vendar pri tem

velja, da ne sme biti enakih več kot k+1 sosednjih vozlov. Če vsem utežem w_i, $0 \le i \le n$ dodelimo vrednost 1, preide NURB krivulja v krivuljo B-zlepkov. Torej so $R_{i,k}$ posplošitev povezovalnih funkcij $N_{i,k}$. Primer povezovalnih funkcij za NURB krivulje vidimo na sliki 11.



Slika 11 Kubične racionalne povezovalne funkcije določene z:

U_{knot} = [0, 0, 0, 0, 1, 2, 3, 4, 5, 6, 6, 6, 6] in
W = [1, 1, 1, 5, 1, 1, 1, 1, 1]

6. ALGORITEM ZA IZRAČUN NURB KRIVULJ

Za izračun povezovalnih funkcij $N_{i,k}$ uporabljamo že znano dinamično drevesno strukturo (poglavje 3). Tudi za shranjevanje vrednosti $R_{i,k}$ vzamemo isti tip podatkovne strukture kot za $N_{i,k}$:

NURB: AllBFunctionType,

s to razliko, da ima tokrat vsaka funkcija $R_{i,k}$ kazalec na svoj zapis tipa BfType (slika 7c).

Za izračun racionalnih povezovalnih funkcij upoštevamo naslednje:

- potrebujemo n+1 povezovalnih funkcij $N_{i,k}$, ki so lahko vse, zaradi morebitnega neuniformnega vozliščnega vektorja, različne;
- ker v splošnem niti dve racionalni povezovalni funkciji nista enaki, moramo določiti tudi vseh n+1 $R_{i,k}$.

Pseudo kod algoritma za izračun povezovalnih funkcij NURB krivulj vidimo na sliki 12.

Pri analizi časovne zahtevnosti za izračun neperiodične krivulje B-zlepkov smo ugotovili, da je le-ta neodvisna od števila točk. Tokrat pa so zaradi neuniformnega vozliščnega vektorja vse funkcije $N_{i,k}$ med seboj različne. Vsako izmed njih izračunamo v polinomskem času (enačba 7), za izračun vseh funkcij $N_{i,k}$ pa moramo poklicati rutino za izračun baznih funkcij

$$2*(1*NoOfSteps + 2*NoOfSteps + \ldots + k*NoOfSteps) +$$
$$+ (k+1)(n+1-2k)*NoOfSteps = (k+1)(n-k+1)*NoOfSteps.$$

Število klicev je tokrat v linearni odvisnosti od števila kontrolnih točk krivulje. Ko smo izračunali vse povezovalne funkcije $N_{i,k}$, je potrebno izračunati še racionalne povezovalne funkcije $R_{i,k}$. Vsaka $R_{i,k}$ se razteza največ čez k+1 segmentov vozliščnega vektorja. Za vsako vrednost u

```
Type
  Ptr = ^TypePtr;
  TypePtr = array [0..knotmax] of record
                                    knot  : real;
                                    index : integer;
                                  end;

procedure MakeNURBf(ABF: AllBFunctionType;

      Var NURB: AllBFunctionType; Wei: PointerToWeights;
   Knt: PointerToKnots; NoAllBf, k: integer; var indexs: Ptr);

{ - ABF: kazalec na povezovalne funkcije Ni,k,
  - NURB: kazalec na NURB povezovalne funkcije,
  - Knt: kazalec na vozliščni vektor,
  - Wei: kazalec na polje, kjer so shranjene uteži,
  - NoAllBf: število neperiodičnih pov. funkcij v ABF,
  - k: stopnja racionalnih povezovalnih funkcij
  - indexs: kazalec na polje vozlov.

var
  i, num, c, m, NoUv : integer;

  function Summa(indexs: Ptr; i,j,k,m: integer;
           N: AllBFunctionType; W: PointerToWeights;
           Knt: PointerToKnots): real;
    { - i: indeks funkcije, ki se trenutno izračunava,
      - j: indeks vrednosti povezovalne funkcije,
      - k: stopnja povezovalnih funkcij,
      - N: kazalec na polje vrednosti povezovalnih funkcij,
      - W: kazalec na polje uteži.
  var
    Sum, u          : real;
    Bfu, ind, kplus1 : integer;
  begin
    Kplus1 := 1; Bfu := 0;
    u := GetuValue(Knt,indexs,i,j);
    while (kplus1  ≤ k + 1) and (Bfu ≤ NoAllBf) do
    begin
      if (u ≥ Knt^[Bfu]) and (u ≤ Knt^[Bfu + k +1]) then
      begin
        ind := GetIndex(Knt,indexs,i,j,Bfu);
        Sum:= Sum + N[Bfu]^.Bf[ind] * W^[Bfu];
        kplus1 := kplus1 + 1;
      end;
      Bfu := Bfu +1;
    end;
    Summa:= Sum;
  end;  { Summa }

begin { MakeNURBf }
  new(indexs);
  indexs^[0].index := 0;    indexs^[0].knot := Knt^.[0];
  m := NoAllBf + k + 1;     c := 0;
  for i := 0 to k do
  begin
    num := i;
    while (num ≤ m - k) and (c ≤ m) do
    begin
      if num ≤ k then
      begin
        indexs^[num + 1].index := ABF^[num]^.NoOfUValues;
        indexs^[num + 1].knot := Knt^[i + k + 1];
      end
      else
      begin
        indexs^[num + 1].index := indexs^[num - k].index +
                                  ABF^[num].NoOfUValues;
        indexs^[num + 1].index := Knt^[num + k + 1];
      end;
      num := num + k + 1; c := c + 1;
    end;
  end;
  GetDiffKnots(indexs,NoAllBf + k + 1,NoOffDiff);
  for i := 0 to NoAllBf do
  begin
    m := NoAllBf + k + 1; NoUv := ABF[i]^.NoOfUValues;
    NURB[i]^.NoOfUValues := NoUv;
    for j := 0 to NoUv do
      NURB[i]^.Bf[j] := ABF[i]^.Bf[j] * Wei^.[i] /
                        Summa(indexs,i,j,k,m,ABF,Wei,Knt);
  end;
end; { MakeNURBf }
```

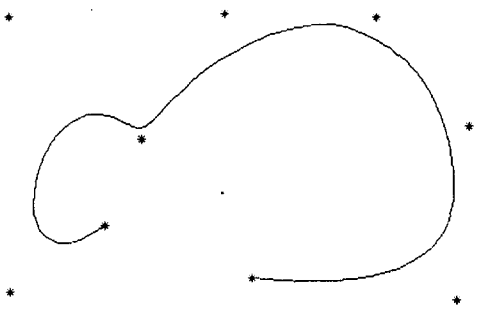
Slika 12 Algoritem za izračun NURB krivulj

moramo določiti vsoto imenovalca v enačbi 10, ki je seštevek k+1 vrednosti funkcij $N_{i,k}$ pomnoženih z utežmi w_i, kar opravimo v polinomskem času.

Algoritem za izris NURB krivulj, ki uporablja podatkovno strukturo s slike 7c, vidimo na sliki 13. Primer NURB krivulje določene s povezovalnimi funkcijami na sliki 11 pa vidimo na sliki 14.

```
procedure PlotNURBSpline (m, n, k : integer;
                          Knt : PointerToKnots; rx, ry : PtoPoints;
                          NURB : AllbFunctionType; indexs : Ptr);
{ - m: število vozlišč vozliščnega vektorja,
  - n: število kontrolnih točk in število povez. funkcij,
  - k: stopnja racionalnih povezovalnih funkcij,
  - Knt: kazalec na vozliščni vektor,
  - rx,ry: kazalca na x- in y-koordinate kontrolnih točk,
  - NURB:  kazalec na vrednosti povezovalnih funkcij
  - indexs: kazalec na polje vozlov.                          }
var
  x, y     : PolyLine_Type;
  i,j,ind,NoOfKnt,firstRBf,lastRBf,func,ni,Steps: integer;
  first    : boolean;
begin
  NoOfKnt := NoOfDiffKnots(indexs);
  for i:=1 to NoofKnt-1 do
  begin
    first:=true;
    for j:=0 to n do
      if  (Knt^[j]<=rKnt[i]) and (rKnt[i+1]<=Knt^[j+k+1])
      then if first then
        begin
          firstRbf:=j;
          first:=false;
        end
        else lastRbf:=j;
    Steps:=round((rKnt[i+1]-rKnt[i])*NoofSteps)+1;
    for ni:=1 to Steps do
    begin
      x[ni]:=0;
      y[ni]:=0;
      for func:=firstRbf to lastRbf do
      begin
        ind:=round((rKnt[i]-Knt^[func])*NoofSteps)+ni-1;
        x[ni]:=x[ni]+ NURB[func]^.Bf[ind]*rx^[func];
        y[ni]:=y[ni]+ NURB[func]^.Bf[ind]*ry^[func];
      end;
    end;
    gPolyLine(Steps,x,y);
  end;
end; { PlotNURBSpline }
```

Slika 13 Algoritem za izris NURB krivulj



Slika 14 NURB krivulja

7. ZAKLJUČEK

Opisana algoritma za izris krivulj B-zlepkov in NURB krivulj smo uporabili v programskem orodju za študij krivulj v CAGD, ki smo ga izdelali v našem laboratoriju [GUID91]. Časovno zahtevnost obeh izdelanih algoritmov smo primerjali z de Boorovim iterativnim algoritmom [YAMA88] in iterativnim racionalnim de Boorovim algoritmom [FARI89], ki izračunata ustrezno krivuljo brez določevanja povezovalnih funkcij. Rezultati so prikazani v tabeli 1 za primer, ko imamo 35 kontrolnih točk. Vidimo, da je naš algoritem za izris krivulj B-zlepkov hitrejši od de Boorovega algoritma, saj s sestavo primerne podatkovne strukture izkorišča lastnosti krivulj B-zlepkov oziroma njihovih povezovalnih funkcij (t.j. da je potrebno določiti samo k+1 različnih povezovalnih funkcij). De Boorov racionalni iterativni algoritem za izris NURB krivulj pa je hitrejši od našega algoritma, saj je tokrat potrebno izračunati vseh n povezovalnih funkcij. Seveda pa de Boorov algoritem ne omogoča prikaz povezovalnih funkcij, ki pa smo jih želeli vključiti v omenjen paket za študij krivulj.

Tabela 1 Čas (msek) porabljen za izris krivulj B-zlepkov in NURB krivulj z različnimi algoritmi

		algoritem								
		A	B	C	D	E	F	G	H	I
red	2	1.59	1.71	1.74	1.83	3.46	3.47	22.95	5.54	22.96
	3	1.97	2.18	2.14	2.59	5.27	5.33	34.56	9.50	34.88
	5	2.74	4.07	4.04	7.84	10.49	10.60	63.25	20.65	66.96
	8	4.40	22.84	11.53	87.49	20.87	21.26	119.91	43.11	197.65

A: naš algoritem - periodične krivulje
B: Cox-de Boorova formula - periodične krivulje
C: naš algoritem - neperiodične krivulje
D: Cox-de Boorova formula - neperiodične krivulje
E: de Boorov algoritem - periodične krivulje
F: de Boorov algoritem - neperiodične krivulje
G: naš algoritem za NURB krivulje
H: de Boorov racionalni algoritem za NURB krivulje
I: rekurzivni algoritem za NURB krivulje

Opisan algoritem lahko brez težav priredimo tudi za izračun in izris drugih tipov krivulj in ploskev, kar smo storili tudi v našem programskem paketu za študij krivulj v CAGD. Algoritem smo napisali v pascalu in za izris uporabili funkcije standarda GKS. Izdelali smo ga na skromni aparaturni opremi: IBM PC/AT-286/16Mh z uporabo matematičnega koprocesorja.

8. LITERATURA

[BART87] Bartels, R. H., J. C. Beatty, and B. A. Barsky, "An Introduction to Splines for Use in Computer Graphics and Geometric Modeling", Morgan Kaufman Publishers, Los Altos, California, 1987.

[BÖHM84] Böhm, W., G. Farin, and J. Kahmann, "A survey of curve and surface methods in CAGD", Comput. Aided Geometric Design, Vol. 1, No. 1, 1984, pp. 1-60.

[FARI89] Farin, G., "Rational Curves and Surfaces", in Mathematical Methods in Computer Aided Geometric Design, (eds. T. Lyche and L. L. Schumaker), Academic Press, 1989, pp. 215-238.

[GUID88] Guid, N., "Računalniška grafika", Tehniška fakulteta Maribor, Maribor, 1988.

[GUID90] Guid, N. and B. Žalik, "Contributions to practical considerations of B-splines", Automatica, Zagreb, Vol. 31, No. 1-2, 1990, pp. 83-88.

[GUID91] Guid, N., B. Žalik, and A. Vesel, "A software teaching tool for curve methods in CAGD", Proc. of Computer Graphics and Education '91, Barcelona, 1991, pp. 62 - 70.

[MORT85] Mortenson, M. E., "Geometric Modeling", John Wiley & Sons, New York, 1985.

[PIEG87] Piegel, L. and R. F. Sproull, "Curve and Surface Construction Using Rational B-splines", Comput. Aided Design, Vol. 19, No. 9, 1987, pp. 485 - 498.

[PIEG89] Piegl, L., "Modifying the shape of rational B-splines. Part 1: curves", Comput. Aided Design, Vol. 21, No. 8, 1989, pp. 509-518.

[ROGE90] Rogers, D. F. and L. A. Adlum, "Dynamic rational B-spline surfaces", Comput. Aided Design, Vol. 22, No. 9, 1990, pp. 609 - 616.

[TURK80] Turk, S., "Računarska grafika. Osnovi teorije i primjene", Školska knjiga Zagreb, Zagreb, 1980.

[WAYN83] Wayne, T., "Rational B-splines for Curve and Surface Representation", IEEE Comput. Graphics and Application, Vol. 3, No. 6, pp. 61-69.

[YAMA88] Yamaguchi, F., Curves and Surfaces in Computer Aided Geometric Design", Springer-Verlag, Berlin, Heidelberg, 1988.

[ŽALI90] Žalik, B. and N. Guid, "Časovna in prostorska zahtevnost pri izrisu krivulj B-zlepkov", Proc. 34. konferenca ETAN, Zagreb, Vol. 8, 1990, pp. 65-72.

ANALIZA ZMOGLJIVOSTI MEHANIZMA ZA KONTROLO PRETOKA V MREŽAH ZA PRENOS PODATKOV

INFORMATICA 3/91

Keywords: packet-switched data networks, window flow-control, queueing network, performance analysis

Marjeta Pučko
Institut Jožef Stefan
Jamova 39, 61111 Ljubljana

Povzetek

V prispevku je opisan strežni model virtualne zveze in mehanizem za kontrolo pretoka po metodi okna kot nadgradnja tega modela. Predstavljeni sta dve različici mehanizma za kontrolo pretoka po metodi okna. Poseben poudarek je na analizi zmogljivosti obeh različic mehanizma, izdelana pa je tudi primerjava med njima glede na najpomembnejša merila za zmogljivost.

Abstract

In this paper, a virtual circuit model, represented by the queueing network is described. On this model, the window control mechanism is superimposed. Two alternatives of the window flow-control mechanism are discussed. A central issue in this discussion is the performance analysis. As a result, both alternatives are compared in terms of the most important performance measures.

1 Uvod

Nalogi mrežnega nivoja v komunikacijski arhitekturi OSI sta poleg izvajanja drugih funkcij tudi usmerjanje in kontrola pretoka. Obe funkciji zagotavljata pravilno dostavo paketov od izvornega do ponornega vozlišča mreže za prenos podatkov. Procedure za kontrolo pretoka, vključene v operacije na mrežnem nivoju, preprečujejo, da bi prišlo do *nasičenja*. V primeru, da bi do nasičenja prišlo, bi se to pokazalo na dva načina:

- časovne zakasnitve bi se izrazito povečale,
- propustnost, merjena v številu paketov na enoto časa, bi izrazito upadla.

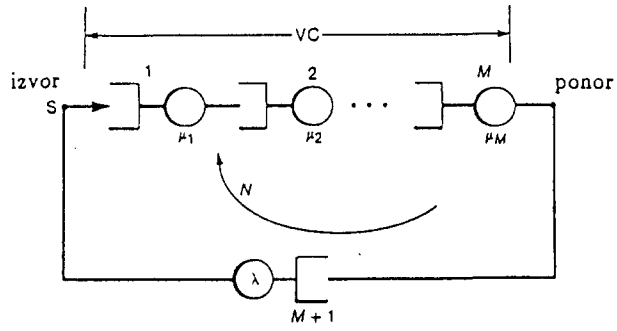
Ob dovolj velikem povečanju bremena se napolnijo vsi izravnalniki, promet se ustavi in propustnost pade na ničlo. V takšnem primeru pride do smrtnega objema (deadlock). Če so mehanizmi za kontrolo pretoka zasnovani pravilno, do smrtnega objema ne more priti.

Pri izdelavi analize zmogljivosti mehanizma za kontrolo pretoka moramo najprej zgraditi *model virtualne zveze*. V strežnem modelu virtualne zveze, ki je opisan v naslednjem razdelku, bomo uporabili neskončne izravnalnike, saj bi uporaba končnih (sicer realnih izravnalnikov) analizo zelo otežila. V nadaljevanju bomo model virtualne zveze nadgradili z modelom mehanizma za kontrolo pretoka po metodi okna in določili njegove značilnosti glede propustnosti in časovnih zakasnitev, ki predstavljata najpomembnejši standardni merili za zmogljivost mrež za prenos podatkov.

2 Model virtualne zveze

Virtualna zveza (VC) pokriva M *store-and-forward* vozlišč od izvora do ponora v mreži s preklapljanjem paketov. V modelu takšno virtualno zvezo predstavimo z M *čakalnimi vrstami*. Parameter λ pove povprečno hitrost prihajanja paketov v virtualno zvezo (porazdeljena je po Poissonu). Zaradi enostavnosti vsako vrsto servisira en sam strežnik.

Prehodno zakasnitev zanemarimo. Izvora zakasnitve sta čas čakanja v vrsti in oddajni čas, ki je odvisen od dolžine paketa in kapacitete oddajne linije. Za i-to čakalno vrsto v virtualni zvezi



Slika 1: Zaprti sistem

$(1 \leq i \leq M)$ je oddajna hitrost oz. kapaciteta enaka μ_i paketov/sek. Zanemarimo tudi vpliv ponovnih oddaj. Bolj splošen (veljavnejši) model celotne mreže bi dobili, če bi med seboj povezali čakalne vrste za posamezne virtualne zveze, vendar bi kompleksnost analize s tem ustrezno narasla.

Kljub enostavni obliki je model še vedno izredno težko analizirati, dokler ne sprejmemo še ene dodatne predpostavke. Količina $1/\mu_i$ v i-tem vozlišču predstavlja povprečni čas za oddajo paketa. Predpostavimo, da je dolžina paketa, ki potuje skozi kaskado čakalnih vrst, izbrana naključno in neodvisno vsakič, ko paket pride v novo čakalno vrsto. To predpostavko o neodvisnosti je prvi uporabil L. Kleinrock [KLEI 74]. Eden od razlogov za veljavnost predpostavke je tudi ta, da so "tokovi" paketov na dani izhodni povezavi pogosto multipleksirani, tako da je prispevek enega toka paketov majhen v primerjavi z ostalimi. Če imajo dolžino vsi enako porazdeljeno, se lahko nek paket v katerikoli vrsti obnaša tako, kot da bi se njegova dolžina spreminjala naključno. Simulacije, izvedene za primerjavo zmogljivosti različnih kontrolnih mehanizmov za eno virtualno zvezo po metodi okna, so pokazale [SCHW 87], da ta predpostavka velja, če M ni prevelik. Za večje $M(M \geq 6)$ se rezultati analize bistveno razlikujejo od rezultatov simulacije.

Če sprejmemo predpostavko o neodvisnosti in eksponentno porazdelitev dolžine paketov, analiza postane izvedljiva. Model ene virtualne zveze postane kaskada neodvisnih $M/M/1$ vrst. Eksponentna porazdelitev dolžine paketov pomeni tudi izstope iz čakalnih vrst, porazdeljene po Poissonu. Po Poissonu porazdeljeni prihodi v naslednjo vrsto pa zopet zagotavljajo $M/M/1$ karakteristiko. Naš model virtualne zveze je ob uporabi naštetih predpostavk le poseben primer *odprte mreže čakalnih vrst* v *produktni obliki*, kar pomeni, da verjetnost stanja v celotni mreži čakalnih vrst lahko zapišemo kot produkt verjetnosti stanj za posamezne vrste.

3 Model s premičnim oknom

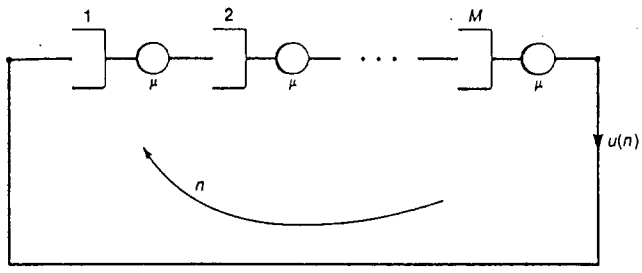
Ko že imamo zgrajen model virtualne zveze, se pojavi vprašanje, kako na vrh tega modela postavimo še mehanizem za kontrolo pretoka.

Najprej določimo mehanizem – to je *kontrola s premičnim oknom*. Mehanizem deluje tako, da je vsak paket posebej potrjen takoj, ko doseže ponor. N predstavlja velikost okna. Če je bilo oddanih že vseh N paketov, do izvora pa še ni prispela nobena potrditev, oddajnik preneha z oddajanjem paketov. Ko potrditev doseže izvor, pomakne oddajno okno za 1 naprej in omogoči oddajo naslednjega paketa.

Predpostavimo, da se potrditve pošiljajo nazaj k izvoru z najvišjo prioriteto. Ta predpostavka za primer potrditev X.25 vmesnika DTE–DCE sicer ne velja, velja pa za IBMovo SNA. Kadar to ne velja, je analiza izvedljiva tudi tako, da virtualno zvezo predstavimo kot zaporedje čakalnih vrst. Predpostavka o pošiljanju potrditev z najvišjo prioriteto zagotavlja najboljši primer glede zmogljivosti.

S to predpostavko zanemarimo tudi časovno zakasnitev pri potrditvah v smeri od ponora do izvora, kar nam omogoča, da zgradimo model s premičnim oknom, nadgrajen na modelu virtualne zveze, kot *model zaprtega sistema*. Izvor in ponor sta povezana z dodatno čakalno vrsto $M+1$ s hitrostjo servisiranja λ. Po zaprtem sistemu kroži fiksno število paketov N. Kadar je vseh N paketov na virtualni zvezi (v zgornjih M vrstah s slike 1, je vrsta $M+1$ prazna. Takoj, ko en od N-tih paketov doseže ponor, se pojavi v vrsti $M+1$ (zaradi predpostavke, da zakasnitve pri potrditvah ni). Izvor sedaj lahko pošilja pakete s hitrostjo λ. Na podlagi tega modela lahko določimo časovno zakasnitev od izvora do ponora in propustnost tega mehanizma za kontrolo pretoka.

Brez kakršnekoli nadaljnje analize lahko podamo oceno kvalitete mehanizma s premičnim



Slika 2: Uporaba Nortonovega teorema za računanje propustnosti

oknom. Pri naraščajoči hitrosti prihajanja paketov se bodo vrste v vozliščih vzdolž virtualne zveze začele polniti, časovne zakasnitve bodo naraščale in prišlo bo do nasičenja. Mehanizem za kontrolo pretoka preprečuje, da bi bilo na virtualni zvezi več kot N paketov hkrati. Čim manjši je N, tem manjše so tudi časovne zakasnitve. Cena, ki jo je treba plačati za to, pa je ustrezno manjša propustnost. Takšno razmerje se pojavlja v vseh mehanizmih, ki preprečujejo nasičenje. Najboljša je torej takšna kontrolna shema, ki pri dani propustnosti zagotavlja najmanjšo časovno zakasnitev oz. najvišjo propustnost pri dani časovni zakasnitvi. Jasno je, da sta oba parametra odvisna tudi od M (števila čakalnih vrst od izvora do ponora). S povečevanjem M narašča minimalna časovna zakasnitev, ker mora vsako vozlišče paket shraniti in poslati naprej, manjša pa se propustnost, ker narašča minimalni čas za prehod skozi mrežo.

Če N še naprej povečujemo, se povečevanje propustnosti ustavi pri neki maksimalni vrednosti, časovna zakasnitev pa še naprej narašča. Torej obstaja neka optimalna vrednost N, kjer se propustnost dovolj poveča, časovna zakasnitev pa se še ne poveča preveč. Takšna vrednost je $N \doteq M$.

V nadaljevanju se moramo zateči h kvantitativni analizi, kar zahteva vpeljavo metod za analizo zaprtega sistema s slike 1. Ta zaprti sistem je le poseben primer *zaprte mreže čakalnih vrst.* Rešitve so zopet v produktni obliki – podobno kot za odprto mrežo čakalnih vrst. Določiti želimo zmogljivost mehanizma s premičnim oknom, ki jo predstavljata propustnost in časovna zakasnitev. Nanašata se na celotno virtualno zvezo. Časi čakanja v vrsti in dolžina vrst nas v tem primeru posebej ne zanimajo, zato lahko uporabimo Nortonov teorem.

3.1 Nortonov teorem

Teorem je sicer imenovan tudi *dekompozicijski teorem za mreže čakalnih vrst*, prvi so ga dokazali K.M. Chandy, U. Herzog in L.S. Woo [SCHW 87]. Velja za mreže s produktno obliko, M-stopenjski model virtualne zveze z eksponentnimi strežniki je le poseben primer.

Če je mreža s čakalnimi vrstami produktne oblike, lahko celotno mrežo predstavimo z eno vrsto med dvema točkama. "Kratkostično" propustnost $u(n)$ lahko določimo z rekurzivnimi metodami, ki jih bomo opisali pozneje. Nortonovemu modelu lahko določimo še bolj poenostavljen ekvivalentni model,kjer vrsta s hitrostjo servisiranja $u(n)$ in stanjem vrste n (število zahtev v vrsti) nadomešča celotno mrežo med dvema točkama. Verjetnosti stanja sta v obeh primerih enaki.

N je celotno število paketov v zaprtem sistemu. Za stanje te vrste veljajo enačbe splošnega *rojstno-smrtnega procesa* z intenzivnostjo rojevanja zahtev λ in intenzivnostjo umiranja zahtev oziroma hitrostjo servisiranja $u(n)$. Verjetnost, da je vrsta v stanju n, je enaka

$$p_n = p_0 \lambda^n / \prod_{i=1}^{n} u(i) \qquad (1)$$

Verjetnost, da je vrsta prazna p_0, je določena z običajnim pogojem

$$\sum_{n=0}^{N} p_n = 1 \qquad (2)$$

Preostane nam le še to, da povežemo uporabo Nortonovega teorema in mehanizma s premičnim oknom. Za začetek analizo še malo poenostavimo s tem, da določimo vsem hitrostim servisiranja $\mu_1, ..., \mu_M$ isto vrednost. Za ta primer sedaj velja

$$u(n) = n \frac{\mu}{n + (M-1)} \qquad (3)$$

n paketov je porazdeljenih v M vrstah (slika 2). Propustnost $n(u)$ bo vedno enaka ali manjša od μ. Podana pa je s produktom

$$u(n) = \mu \cdot verjetnost(vrsta\ ni\ prazna) \qquad (4)$$

Zaradi simetrije so vse vrste identične in verjetnost, da vrsta ni prazna, je enaka za vse vrste. S pomočjo enačbe 3 dobimo

$$\frac{p_n}{p_0} = \rho^n \begin{pmatrix} M-1+n \\ n \end{pmatrix} \quad (5)$$

kjer je izkoristek $\rho \equiv \lambda/\mu$, verjetnost p_0 pa

$$\frac{1}{p_0} = \sum_{n=0}^{N} \rho_n \begin{pmatrix} M-1+n \\ n \end{pmatrix} \quad (6)$$

Propustnost γ virtualne zveze s kontrolo pretoka s pomočjo okna je podana s povprečjem vseh N možnih hitrosti servisiranja:

$$\gamma = \sum_{n=1}^{N} u(n) p_n \quad (7)$$

Po Littlovi formuli [KLEI 74] je časovna zakasnitev $E(T)$ na virtualni zvezi enaka razmerju med povprečnim številom paketov $E(n)$ in propustnostjo γ.

$$E(T) = E(n)/\gamma = \sum_{n=1}^{N} n p_n/\gamma \quad (8)$$

V primeru $\lambda \to \infty$ lahko analizo zmogljivosti tega mehanizma za kontrolo pretoka tudi nekoliko poenostavimo. Za ta primer velja $E(n) = N$, za *propustnost* velja

$$\gamma = u(N) = \frac{N\mu}{N + (M-1)}, \ \lambda \to \infty \quad (9)$$

za *časovno zakasnitev* pa

$$E(T) = N/\gamma = [M - 1 + N]/\mu \quad (10)$$

Medsebojna odvisnost propustnosti in časovne zakasnitve pri kontroli pretoka s premičnim oknom je prikazana na sliki 3 (primerjava z drugim mehanizmom za kontrolo pretoka - potrjevanjem za celotno okno).

Minimalna časovna zakasnitev $E(T) = M/\mu$ se pojavlja pri $N = 1$ (na sliki je $\mu E(T) = 3$ za $M = 3$). S povečevanjem velikosti okna N časovna zakasnitev proporcionalno narašča, prav tako tudi propustnost γ. Ko propustnost doseže vrednost μ, začne naraščati počasneje kot raste N. Karakteristika je naslednja:

$$\mu E(T) = \frac{M-1}{1-(\rho/\mu)}, \ \lambda \to \infty \quad (11)$$

Enačba je veljavna seveda samo za tiste N, ki so cela števila. Čim bolj se γ približuje μ (s povečevanjem N), tem bolj imajo majhne spremembe γ za posledico večje spremembe $E(T)$.

Kakšna naj bi bila vrednost N ? Različni kriteriji vodijo k isti vrednosti, to je $N = M - 1$. Naraščajoči N se odraža v enakomernem naraščanju časovne zakasnitve in hitrem upadanju propustnosti. Ustrezen kriterij za iskanje "dobre" vrednosti N predstavlja iskanje maksimalne vrednosti razmerja $\gamma/E(T)$ ali njegovega normaliziranega ekvivalenta $\gamma/\mu/\mu E(T)$, imenovanega tudi "moč" sistema. V primeru, ko je $\lambda = \mu$, je bolje izbrati $N = M$.

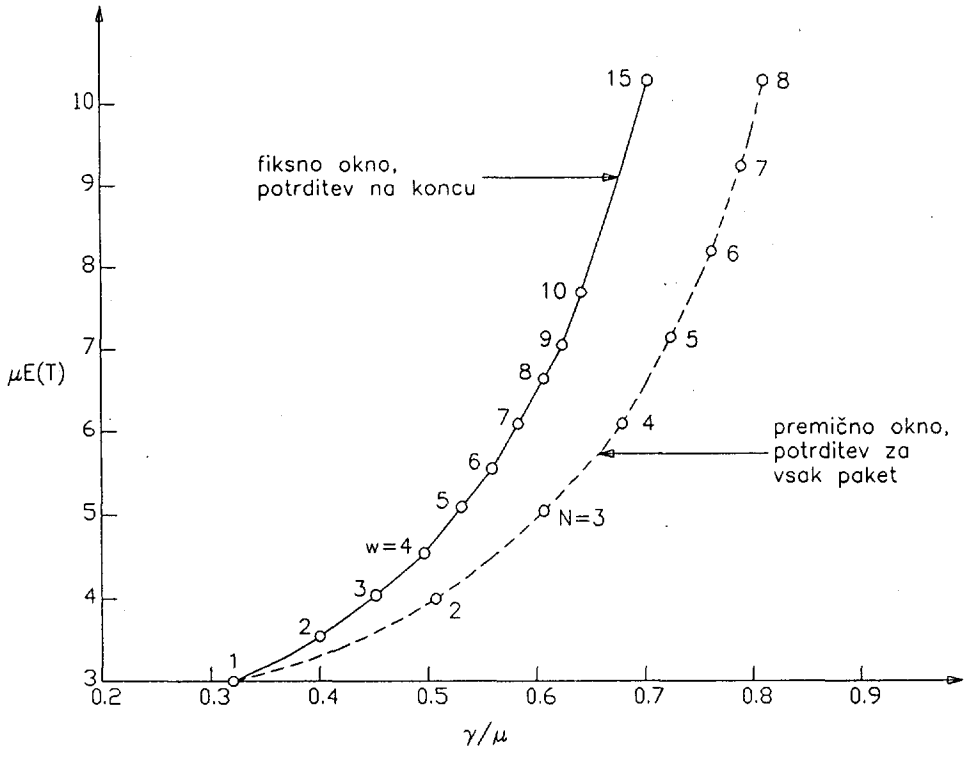
Ni nujno, da mora biti mehanizem za kontrolo pretoka takšen, da je potrjen sprejem vsakega paketa posebej. Število potrditev je mogoče tudi zmanjšati tako, da sprejemnik pošlje potrditev za vse predhodne nepotrjene pakete iz enega okna hkrati.

4 Model s potrjevanjem za celotno okno

V primeru modela s potrjevanjem za celotno okno ni potrebno potrjevanje za vsak prispeli paket. Sprejemnik lahko zadrži potrjevanje, dokler ni pripravljen na sprejem paketov, nato pa z eno potrditvijo potrdi sprejem vseh prispelih paketov. Prednost takšnega načina potrjevanja je v možnosti usklajevanja hitrosti oddajnika in sprejemnika ter zmanjšanju števila potrditev.

Pri primerjavi tega modela z modelom s premičnim oknom nas zanima propustnost v ekstremnem primeru, ko sprejemnik potrjuje sprejem vseh paketov iz enega okna hkrati.

Ker so velikosti okna v različnih mehanizmih za kontrolo pretoka lahko različne, v tem primeru označimo velikost okna z w. Izravnalnik w shrani $w-1$ paketov. Ko prispe še w-ti paket, sproži potrditev za vseh w paketov in postavi števec paketov C (z vrednostjo 0) zopet na vrednost w. C pravzaprav predstavlja čakalno vrsto z intenzivnostjo prihajanja zahtev λ. Izravnalnik w pomeni tudi



Slika 3: Odvisnost propustnosti in časovne zakasnitve, $M = 3, \lambda \to \infty$

edino razliko v primerjavi z modelom s premičnim oknom, ki pa analizo propustnosti zelo oteži. Vsebina izravnalnika w se namreč prestavi v čakalno vrsto naenkrat, zato lastnost produktne oblike za to mrežo ne velja več in uporaba Nortonovega teorema ne zagotavlja eksaktne rešitve. Če Nortonov teorem kljub temu uporabimo, pa lahko dobimo *aproksimacijo* [KURO 88].

Za predstavitev stanja strežne vrste C in izravnalnika w potrebujemo števili i in j, za kateri velja $n = w - (i + j)$. Sedaj lahko zapišemo dvodimenzionalne ravnotežne enačbe za verjetnosti stanj p_{ij}. Sistem enačb (njihovo število je reda $w^2/2$) rešujemo numerično.

V nekaterih primerih lahko računanje tudi poenostavimo, takšna primera sta posebni vrednosti λ, $\lambda \to \infty$ in $\lambda = \mu$. V primeru največjega prometa, $\lambda \to \infty$, dvodimenzionalne enačbe (odvisne od i in j) prevedemo na enodimenzionalne (odvisne samo od j). Računamo le verjetnosti stanj p_j izravnalnika w oziroma zgornje vrste, $p_n = p_{w-j}$:

$$\begin{aligned} u(w)p_0 &= u(1)p_{w-1} \\ u(w-1)p_1 &= u(w)p_0 \\ &\vdots \\ u(1)p_{w-1} &= u(2)p_{w-2} \end{aligned}$$

Za $u(n)$ velja enako kot pri modelu s premičnim oknom (enačba 3)

$$u(n) = n\frac{\mu}{n + (M-1)}$$

Če rešimo enačbe za p_j, dobimo

$$p_j/p_0 = u(w)/u(w-j) \qquad (12)$$

glede na vsa stanja pa

$$p_j = \frac{(w-j) + (M-1)}{(w-j)[w + (M-1)T_w]} \qquad (13)$$

kjer je T_w končna vsota

$$T_w = \sum_{j=1}^{w} \frac{1}{j} \qquad (14)$$

Enačbo 13 uporabimo pri računanju propustnosti γ in povprečne zakasnitve $E(T)$. *Propustnost* je podana z enačbo

$$\gamma = \frac{\mu w}{[w + (M-1)T_w]} \qquad (15)$$

Ob primerjavi te enačbe z ustrezno enačbo za propustnost pri modelu s premičnim oknom

(enačba 9) lahko vidimo, da imata povsem enako obliko, le $(M-1)/N$ je nadomeščen z $(M-1)T_w/w$.

Pri določanju povprečne časovne zakasnitve zopet uporabimo Littlovo formulo. Naj bo $p(n)$ verjetnost stanja Nortonovega ekvivalenta virtualne zveze. Potem je *povprečna časovna zakasnitev* enaka

$$E(n) = \sum_{n=1}^{w} np(n) = \frac{\gamma}{\mu}\left[\frac{1+w}{2} + (M-1)\right] \quad (16)$$

kjer seveda velja $n = w - j$ in $p(n) = p_{w-n}$, njena normalizirana vrednost pa je podana z enačbo

$$\mu E(T) = \frac{E(n)}{\gamma/\mu} = \left(M - 1 + \frac{1+w}{2}\right) \quad (17)$$

Če to enačbo zopet primerjamo z ustrezno enačbo pri modelu s premičnim oknom, ugotovimo, da smo velikost okna N nadomestili z $(1+w)/2$. Ko primerjamo zmogljivost obeh mehanizmov (slika 3), lahko vidimo, da zmogljivost mehanizma s potrjevanjem za celotno okno močno zaostaja za zmogljivostjo mehanizma s premičnim oknom. Razlika med njima s povečevanjem okna postaja še večja. Pri primerjavi obeh mehanizmov bi podobne rezultate dobili tudi v primeru $\lambda = \mu$. V primeru mehanizma s premičnim oknom predstavlja optimalno velikost okna $N = M$, v primeru mehanizma s potrjevanjem za celotno okno pa $w = 2M - 1$.

5 Zaključek

Predstavitev virtualne zveze z mrežo čakalnih vrst in njena nadgradnja z modelom mehanizma za kontrolo pretoka po metodi okna nam omogočata učinkovito analizo zmogljivosti. Ob primerjavi najpomembnejših meril za zmogljivost vidimo, da zmogljivost mehanizma s potrjevanjem za celotno okno močno zaostaja za zmogljivostjo mehanizma s premičnim oknom. Mehanizem s potrjevanjem za celotno okno ima sicer dve pomembni prednosti: manjše število potrditev in možnost usklajevanja hitrosti oddajnika in sprejemnika. Cena, ki jo je treba plačati za to, pa je nižja propustnost. Pojavi se vprašanje, ali je mogoče povečati propustnost do te mere, da bi bila skoraj enaka kot pri modelu s premičnim oknom, obenem pa bi obdržali prednost ene same potrditve za celotno okno. Kompromis med obema rešitvama je mogoče doseči, primer takšnega mehanizma je IBM SNA mehanizem za kontrolo pretoka.

6 Viri

[KING 90] P.King: *Computer and Communications Systems Performance Modeling*, Prentice – Hall, 1990,

[KLEI 74] L.Kleinrock: *Queueing Systems*, Vol. I: Theory, John Wiley & Sons, 1974,

[KLEI 76] L.Kleinrock: *Queueing Systems*, Vol. II: Computer Applications, John Wiley & Sons, 1976,

[KONH 80] A.G.Konheim: *A Queueing Analysis of Two ARQ Protocols*, IEEE Trans. on Comm., COM-28, no. 7, July 1980, pp. 1004 – 1014,

[KURO 88] J.F.Kurose, H.T.Mouftah: *Computer- Aided Modeling, Analysis, and Design of Communication Networks*, IEEE Journal on Selected Areas in Communications, vol. 6, no. 1, January 1988, pp. 130 – 145,

[PUCK 90] M.Pučko: *Propustnost javnih X.25 mrež za prenos podatkov*, IJS DP-5799, Ljubljana, marec 1990.

[SCHW 87] M.Schwartz: *Telecommunication Networks: Protocols, Modeling and Analysis*, Addison – Wesley, 1987,

[STAL 91] W.Stallings: *A practical guide to queueing analysis*, Byte, February 1991, pp. 309 – 316,

[STUC 85] B.W.Stuck, E.Arthurs: *Computer and Communications Systems Performance Modeling*, Prentice – Hall, 1985.

Knjižna recenzija

O KNJIZI I NJENOM AUTORU:

Anton P. Železnikar, »On the Way to Information«, Slovensko društvo Informatika, 1990.

Pojava nove knjige, kao i rođenje novog deteta, radostan je događaj. Međutim, kada se dete rodi u vreme rata, onda je radost obično pomešana sa brigom i strahom. Kada se jedan značajan intelektualni rezultat pojavi u trenutku kada apsurd i bezumlje odnose pobedu za pobedom, a turbulencija društvenog fluida na površinu izbacuje talog velike specifične težine, situacija je slična. Knjiga »On the Way to Information« Antona P. Železnikara zaslužuje pažnju i kao fenomen koji u izvesnom smislu ne pripada ni vremenu ni prostoru u kome je nastao. Cilj ovog osvrta je da, verujem sa razlogom, donekle odstupi od uobičajenog prikaza pojave nove knjige. Potaknut knjigom, sadašnjim vremenom, a svakako i opkruženjem u kome je knjiga nastala, nameravam da se pored prikaza sadržaja knjige nešto šire osvrnem na delo u svom prostoru i na autora u svom vremenu.

Knjiga »On the Way to Information« nosi naslov po prvom od niza eseja koji čine njen sadržaj. Ovaj prvi odeljak ima uvodni karakter i prikazuje informaciju u njenom raznolikom okruženju. Da ova knjiga namerava da udari temelje jedne nove discipline postaje jasno u njenom drugom odeljku koji sa maksimalnim mogućim nivoom granularnosti definiše dvadesetak koncepcija i pojmova koji svi u korenu reči imaju informaciju i informisanje. U ovom delu knjige učinjen je očigledan doprinos semantici engleskog jezika. Autor ovde čini pokušaj da obrazuje konzistentno definisani skup apstraktnih komponenti da bi otvorio prostor da međusobni odnosi ovih komponenti u raznim okruženjima postanu predmet jedne specifične teorije. Naravno, ovo je dobar i lako prepoznatljiv početak svih novih disciplina. Ostaje međutim nejasno hoće li finoća granularnosti pojmova biti kontraproduktivna za teoriju koju trebaju pored njenog autora asimilirati i dalje poneti i drugi ljudi. U drugom odeljku ove knjige čini se da broj nijansi prevazilazi onu granicu koju normalan posmatrač može konsumirati bez ulaganja izuzetnih napora.

Posle drugog odeljka koji je centralan po položaju i najobimniji po zapremini slede još tri odeljka. Treći odeljak ispituje principe informacije, informacione forme, osobine inteligencije i intelekta, a takođe i sučelje koje povezuje informaciju sa njenim folozofskim i fizičkim okruženjem. Četvrti odeljak pokušava da metafizički pojam Boga dopuni sa svim onim što je tom pojmu nedostajalo u odsustvu jedne teorije o sveobuhvatnom značenju i dometu informacija u sadašnjem svetu, kao i o informacionoj suštini ljudskog intelekta. Poslednji, peti odeljak ispituje razne aspekte informatičkog društva, kao i to u kojoj meri se civilizacija, kultura i ljudska intelektualnost oslanjaju na informaciju i prepliću sa njenim raznim formama. Zaključak ovog dela je veoma kratak jer se radi o prvom tomu višetomne knjige, pa će se poruka celine pojaviti tek onda kada se pojavi i celina. U ovom zaključku se daje obećanje da će drugi tom biti posvećen mogućnostima formalizacije raznih informacionih koncepata.

Imajući u vidu sadržaj ove knjige jasno je da ona raste na intelektualnom terenu koji je izuzetno slabo naseljen. Radi se o tromeđi filozofije, sociologije, i tehnologije, o ograničenom prostoru do koga iz pozadine stalno dopiru daleki odijeci računarske tehnike, a pored njih i sve ostale komponente intelektualne arome kasnog dvadesetog veka. Iako je ovu knjigu pisao inženjer to nije knjiga za inženjere, posebno ne za one koji inženjerstvo (pravilno) shvataju kao most između riznice teorijskog znanja i čovekovih praktičnih potreba. Najzadovoljniji čitaoci biće verovatno filozofi, kulturolozi, i psiholozi, a pored njih i ostali koji društvo posmatraju sa aspekta komunikacija i upotrebe informacija.

Knjiga je napisana na engleskom jeziku, iz istih razloga iz kojih je Boškovićeva Theoria Philosophiae Naturalis pisana na latinskom. To je iz više razloga dobro. Pre svega, očigledno je da autor veruje da delo ove vrste zaslužuje da dođe do auditorijuma koji nije ograničen samo na njegovu neposrednu sredinu. Nadalje, ulazeći u nove prostore autor je bio prinuđen da definiše novu terminologiju, i nije se plašio da to učini na jeziku koji mu nije maternji. Konačno, izborom engles-

kog jezika izbegnut je razlog za bilo kakve klasifikacije ove knjige u odnosu na gabarite jugoslavenskog prostora, učinjen je doprinos veoma oskudnom jugoslavenskom stručnom izdavaštvu na engleskom jeziku, a takođe je ponuđen i jasan putokaz drugim autorima sa ovih prostora.

Knjiga nije pisana da bi bila roba u izlogu knjižare (i ne predskazujem joj nikakav komercijalni uspeh), već je to kulturni i društveni fenomen koji stoji u specifičnom odnosu prema vremenu i prostoru u kome je nastao. Stoga se na ovu knjigu ne sme osvrtati sa stanovišta potrošačkog prikaza čiji je cilj da čitaoca obavesti da li mu se »isplati« da knjigu nabavi — ovakva, nažalost česta, vrsta prikaza je verovatno i u širim razmerama najbolji način da se knjizi oduzme svako dostojanstvo i njena dimenzija intelektualnog ostvarenja i umetničkog dela, i pored toga što moramo priznati da mnoge savremene knjige iz praktičnog računarstva ne zaslužuju nikakav bolji tretman.

Budući da je ova knjiga prevashodno jedno autorsko delo, vredi je posmatrati i u kontekstu celokupnog opusa njenog autora. Tačnije, svaka prava knjiga predstavlja jedan od intelektualnih autoportreta njenog autora. Železnikarova knjiga je autoportret čoveka koji ide svojim putem bez obzira na vremneske prilike koje ga na tom putu prate. Vremena su, kako se to banalno kaže, teška. To su istovremeno i idealni uslovi u kojima naučnici kojima je njihova nauka teška i umetnici kojima je njihova umetnost teška žure da se prihvate drugih poslova, čiji su efekti po pravilu direktno ili indirektno štetni toj njihovoj nauci, odnosno umetnosti. Takvi instinkti, po svemu sudeći, daleko su od intelektualnog sveta Antona Železnikara.

Obzirom na vreme kada se ova knjiga pojavila vredi se i šire osvrnuti na delovanje autora, Antona P. Železnikara. Kratko rečeno, Železnikar je slovenački naučnik koji je tridesetak godina intenzivno i efikasno delovao na jugoslavenskom prostoru. U momentu kada je taj prostor razbijen u paramparčad valja pomenuti delo onih koji su ga nekada gradili, i valja im se zahvaliti. Železnikar je svojim radom u oblasti računarskog hardvera, razvoja industrijskih proizvoda, teorije programskih jezika, filozofije informatike, naučne publicistike i padagogije, pa sve do organizacije jugoslavenskih publikacija i naučnih skupova sigurno naučnik iz oblasti računarske tehnike sa najširim stručnim spektrom ne samo u slovenačkim nego i u jugoslavenskim okvirima. Ostavio je brojne pozitivne tragove na jugoslavenskom prostoru. Bio je glavni organizator izuzetno značajnog jugoslavenskog međunarodnog kongresa Informatica koji je tokom sedamdesetih godina bio u svome zenitu i bio u stanju da svake godine okupi na Bledu po 500 istraživača ne samo is Jugoslavije nego i iz brojnih drugih zemalja. Bila je to velika škola za sve njene učesnike. Za to mu svi sa ovih prostora moraju biti zahvalni. Nažalost, nismo mu zahvalni što je početkom osamdesetih godina istu tu manifestaciju uništio njenim prenosom u Ljubljanu i tematskim sakaćenjem u okviru Iskra Delte. Istini za volju, to je bilo u jugoslavenskom duhu, jer se jugoslavenski prostor nikad nije podigao do one civilizacijske razine na kojoj se javljaju manifestacije grčevite borbe za očuvanje dobrih tradicija.

Jedan deo eseja objavljenih u okviru knjige »On the Way to Information« publikovan je pre toga u časopisu Informatica. Anton Železnikar je osnivač i glavni urednik ovog uglednog jugoslavenskog stručnog časopisa još od 1977. godine. Ovaj časopis ima nesumnjivu reputaciju na jugoslavenskim prostorima u oblasti računarstva i informatike, i obzirom na svoje dimenzije prolazi danas kroz ozbiljna iskušenja. Poželimo ovom vrednom časopisu i njegovom glavnom uredniku da neozledeni produ kroz apsurde sadašnjeg vremena. U protivnom, teško je poverovati da ćemo dočekati pojavu drugog toma knjige »On the Way to Information«.

Prof. Jozo J. Dujmović
Elektrotehnički fakultet, Beograd

Dvajsetletnica kongresa IFIP'71

Letos septembra mineva dvajset let od svetovnega kongresa Mednarodne federacije za procesiranje podatkov (IFIP) v Ljubljani, na katerem se je zbralo 2300 udeležencev iz 32 držav. V tistih časih je Slovenija še imela upanje za prodor na mednarodni trg, liberalizem politično-ekonomskih struktur še ni bil zatrt. Stane Kavčič, predsednik IS Slovenije, je bil dobrohoten gostitelj mednarodne smetane v vili Podrožnik in v Vladni palači. Mestni svet je gostil vse udeležence v prostorih Magistrata. Samo kongres z rastavo računalniške opreme na GR in s tiskanjem zbornika je prinesel Ljubljani (brez hotelskih storitev Bleda in Ljubljane) devizni priliv pol milijona dolarjev.

A.P. Železnikar

Informatica

Informatica *is published four times a year in Winter, Spring, Summer and Autumn by the Slovene Society Informatika, Tržaška cesta 2, 61000 Ljubljana, Yugoslavia.*

Volume 15 Number 3 August 1991 YU ISSN 0350 – 5596

Informatica

A Journal of Computing and Informatics

CONTENTS