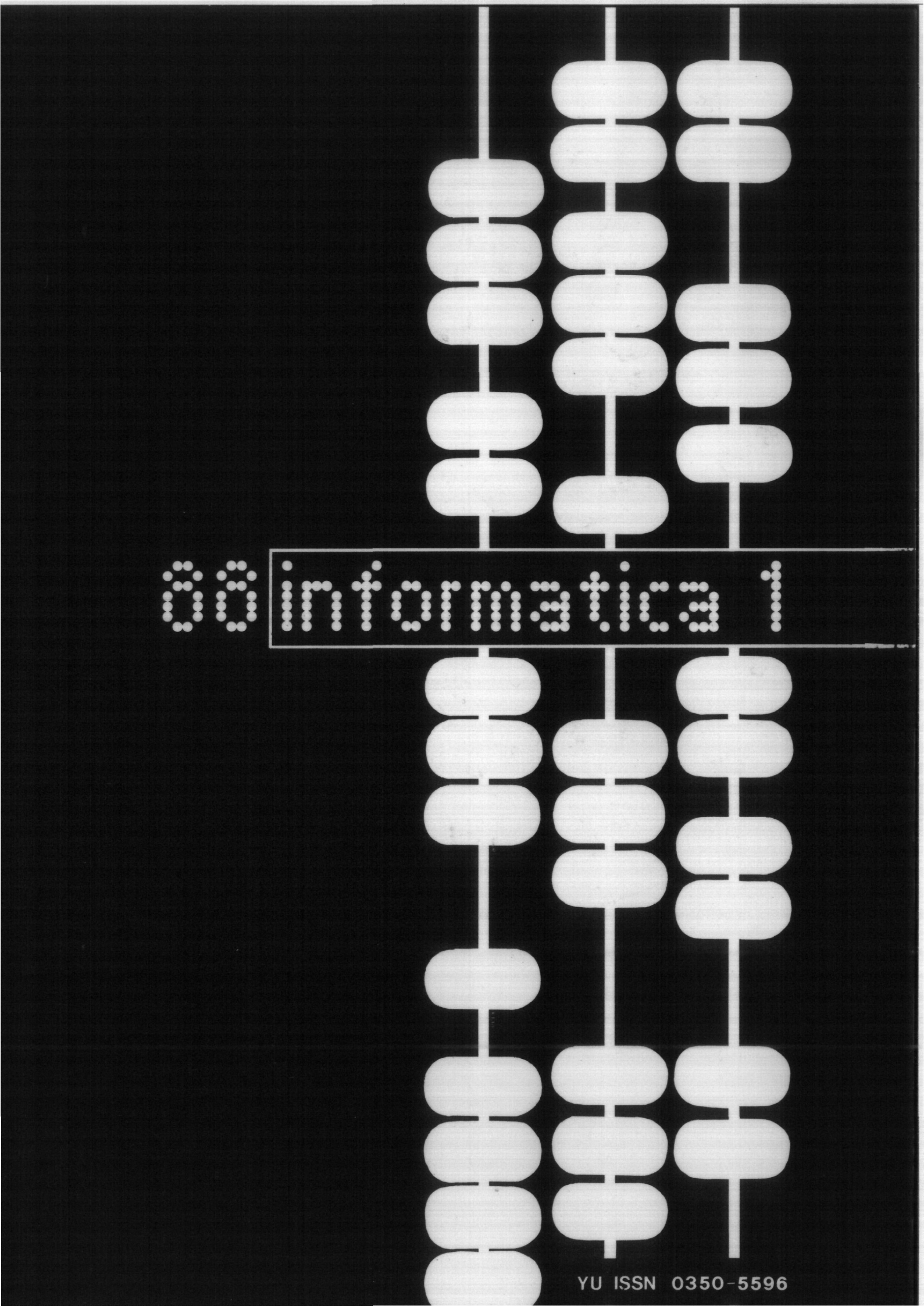


YU ISSN 0350-5596

Iskra Delta

Iskra Delta

Iskra Delta

Iskra Delta

proizvodnja računalniških sistemov in inženiring, p.o.

61000 Ljubljana, Parmova 41

telefon: (061) 312-988

telex: 31366 YU DELTA

Iskra Delta

Iskra Delta

Iskra Delta

Iskra Delta

Iskra Delta

Iskra Delta

Iskra Delta

Iskra Delta

Iskra Delta

Iskra Delta

Iskra Delta

informatica

JOURNAL OF COMPUTING AND INFORMATICS

YU ISSN 0350-5596

VOLUME 12, 1988 – No. 1

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

Uredništvo in uprava:

Casopis Informatika, Iskra Delta, Stegne 15B, 61000 Ljubljana, telefon (061) 574 554; teleks 31366 YU Delta.

Letna naročnina za delovne organizacije znaša 24000 din, za zasebne naročnike 6000 din, za studente 2000 din; posamezna številka 8000 din.

Številka žiro računa: 50101-678-51841

Pri financiranju časopisa sodeluje Raziskovalna skupnost Slovenije

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

Tisk: Tiskarna Kresija, Ljubljana

V S E B I N A

A.P.Zeleznikar 3 Uvodno predavanje

Z. Kačič, B. Horvat, S. Greif 6 Man-Machine Communication: Speaker-Independent Speech Recognition

Z. Brezočnik, B. Horvat 13 Proving the Correctness of Digital Hardware Designs Using Prolog

P. Kolbezen 17 Reconfigurable Multi-Microprocessor Systems

S. Mavrič, P. Kolbezen 25 Stohastični pristop k analizi učinkovitosti večprocesorskih sistemov

I. Pepelnjak, J. Virant ... 30 Interna struktura Unix združljivega 8-bitnega OS

M. Bradesko, L. Pipan ... 40 Sistemski uporabniški vmesnik za 8-bitni Unix združljivi OS

N. Zimic, J. Virant ... 44 Možnosti zaščite programske opreme na mikroračunalnikih

T. Kalin, T. Vidmar 48 Logično testiranje protokolov

D. Mihajlović 52 Određivanje vrste reči iz srpskohrvatskog jezika primenom računara

M. Sifrar 55 Računalniška infrastruktura za delovanje KIS in ZTI

S. Damjanović, L. Đorđević 59 VAL realizacija n-tog korena korišćenjem paralelnog iterativnog algoritma

A. P. Železnikar 65 Lastnosti informacije: mala enciklopedija (A)

77 Potopis

A. P. Železnikar Parsys Expeditions to New Worlds II

Avtorsko stvarno kazalo časopisa Informatika 11 (1987)

informatica

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

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

Headquarters:

Informatica, Iskra Delta, Stegne 15B, 61000 Ljubljana, Yugoslavia. Phone: 61 57 45 54. Telex: 31366 yu delta

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

Printed by: Tiskarna Kresija, Ljubljana

YU ISSN 0350-5596

LETNIK 12, 1988 – ŠT. 1

CONTENTS

A.P. Železnikar 3 An Introductory Lecture

Z. Kačič, B. Horvat, S. Greif 6 Man-Machine Communication: Speaker-Independent Speech Recognition

Z. Brezočnik, B. Horvat 13 Proving the Correctness of Digital Hardware Designs Using Prolog

P. Kolbezen 17 Reconfigurable Multi-Microprocessor Systems

S. Mavrič, P. Kolbezen 25 Stochastic Approach in Performance Analysis of Multi...

I. Pepelnjak, J. Virant ... 30 Internal Structure of an 8-bit Unix Compatible OS

M. Bradeško, L. Pipan ... 40 System Command Line Interface for 8-bit Unix Compatible OS

N. Zimic, J. Virant ... 44 Possibilities of Software Protection in Microcomputers

T. Kalin, T. Vidmar 48 Logical Testing of Protocols

D. Mihajlović 52 Computer Aided Word Type Determination in Serbo-Croatian

M. Sifrar 55 A Computer Supported Bibliographic System

S. Damjanović, L. Đorđević 59 VAL Realization of n-th Root Evaluation Using Parallel Iterative Algorithm

A. P. Železnikar 65 Properties of Information: A Small Encyclopedia (A)

77 Report of a Journey

A. P. Železnikar Parsys Expeditions to New Worlds II

Authors Subject Index of Informatica 11 (1987)

UDK 681.3.068

Anton P. Železnikar
Iskra Delta, Ljubljana

Introductory Lecture. This lecture was given to the students of Computer Science Department within the topic of operating systems and compilers, in Maribor, on October 7, 1987. The aim of the lecture was to evoke the motivation of understanding, of mastering, and of designing of really complex and large program systems as they appear in the framework of operating systems and compilers. Stressing the importance of software engineering and proficient methodologies, the emphasis to the future disciplines concerning artificial intelligence, parallel computer architectures, parallel processing and parallel programming, new informational concepts and methodologies, and new technological environment (optics, biomolecular circuits, etc.) was given.

Snov tega članka je bilo uvodno predavanje študentom računalništva v okviru predmetov Operacijski sistemi in Prevajalniki na VTS v Mariboru, 7. oktobra 1987. Namen predavanja je bila vzpodbuditev motivacije za razumevanje, obvladovanje in razvoj tako kompleksnih programov, kot se pojavljajo v okviru operacijskih sistemov in prevajalnikov. Poudarjena je bila pomembnost programirne tehnike in strokovnih metodologij v povezavi s prihodnjimi usmeritvami, ki se tičejo umetne inteligence, paralelnih računalniških arhitektur, paralelnega procesiranja in paralelnega programiranja, novih informacijskih konceptov in novega tehnološkega okolja (optika, biomolekularna vezja itd.)

... "Warum dringt das Verstehen nach allen wesenhaften Dimensionen des in ihm Erschließbaren immer in die Möglichkeiten? Weil das Verstehen an ihm selbst die existenziale Struktur hat, die wir den Entwurf nennen. ... Der Entwurfcharakter des Verstehens besagt ferner, daß dieses das, woraufhin es entwirft, die Möglichkeiten, selbst nicht thematisch erfaßt. Solches Erfassen benimmt dem Entworfenen gerade seinen Möglichkeitscharakter, zieht es herab zu einen gegebenen, gemeinten Bestand, während der Entwurf im Werfen die Möglichkeit als Möglichkeit sich vorwirft und als solche sein läßt. Das Verstehen ist, als Entwerfen, die Seinsart des Daseins, in der es seine Möglichkeiten als Möglichkeiten ist." ...

M. Haidegger, Sein und Zeit, S. 145.

1

Glavni namen tega uvodnega predavanja je vzpodbujanje in nastajanje motivacije za razumevanje, obvladovanje in konstruiranje zares zapletenih in obsežnih programov, kot so

operacijski sistemi in prevajalniki.

Obvladovanje konstruiranja te vrste računalniških programov zahteva razen znanih napotkov in priporočil o dobrem in zanesljivem programiranju, kot so npr. pravila

programirne tehnike
(strukturiranega programiranja,
navzdoljnega in navzgornega razvoja programov,
jasne programirne konceptualizacije,
uporabe specifičnih programirnih orodij
pa tudi vztrajnega, trdega in
sistematičnega dela),

še specifične metode oziroma metodologijo operacijskih sistemov in prevajalnikov kot specifičnih strokovnih disciplin.

2

Študentje in študentke te študijske smeri boste postali računalniški poklicanci, profesionalci, nekateri od vas pa specialisti oziroma eksperti npr. za operacijske sisteme in prevajalnike. V prihodnosti se bo današnja strokovna podoba teh predmetov bistveno spreminjala, saj bo potrebno tako v operacijske sisteme kot prevajalnike vgrajevati oziroma v-konstruirati inteligenčne mehanizme, hkrati pa bo potrebno upoštevati tudi povsem nove računalniške arhitekture, ki bodo npr. paralelne, paralelnoserijske, dinamične, tj. spremenljive in nastajalne in tudi tehnološko radikalno nove (anorganska in organska tehnologija). Torej naj velja na začetku tega uvoda pregledno opozorilo, da bo kompleksno programiranje v prihodnosti bistveno soočeno

---

* Predavanje je bilo opravljeno kot uvod v predavanja predmetov "Prevajalniki" in "Operacijski sistemi" na VTS Univerze v Mariboru, dne 7. oktobra 1987.

s problematiko

umetne inteligence,
paralelnih računalniških arhitektur,
paralelnega procesiranja in programiranja,
novih informacijskih konceptov in metodologij
in novega tehnološkega okolja
(optika, biosubstrati itd.)

V tem trenutku računalniškega razvoja so na pohodu paralelni računalniki. Današnje obdobje teh računalnikov je še vedno pionirsko in zanesljivo je le to, da so na pohodu in da je vprašanje njihovega dokončnega prodora predvsem v množičnejšem plasmaju. Zdi se mi pomembno, da kot študentje dojemate tudi tiste razvojne smeri, ki bodo v razdobju vaše strokovne dozorelosti na svojem višku, da pozorno sprejemate informacijo o strokovnem razvoju, ker je ta informacija bistveno različna, domala nerazumljivo in nesmiselno oddaljena od vsakdanje potrošniške, komercialne in računalniško ideološke informacije. Ta pripomba se nanaša predvsem tudi na kompleksno programsko opremo novih, paralelnih strojev, ki bo v svoji strukturi veliko bolj umetnointeligenčna, kot je današnja programska oprema.

3

Strokovno spremljanje računalniškega razvoja, tehnologije in metodologij naj postane tudi vaša lastna intelektualna funkcija, ki izvira iz lastne volje, prepričanja in delavnosti strokovnjaka. Uspehi domače računalniške industrije bi lahko bili bistveno večji, če bi v preteklem obdobju v domači industriji delovali ti. računalniški intelektualci. Tudi za strokovnjaka je pomembno vprašanje: Kdo je intelektualec? Intelektualec še ni nekdo, ki je končal najvišje šole, prebral gore knjig, prejel najvišje nagrade in priznanja, ali tisti, ki ga je konformno okolje razbobnalo za izvirnega strokovnjaka, ki uspešno opravlja intelektualni poklic ali upognjen prodaja svoje ideje. Intelektualec ni produkt povprečnega socialnega okolja, temveč ima svoj umni in razumni sistem mišljenja, prepričanja in vedenja neglede na socialni izvor, formalno izobrazbo in mesto v družbeni hierarhiji. Interes intelektualne volje je, da svet spoznava in osmišlja, v njem ustvarjalno deluje, ne pa da ga drži v svoji oblasti in ga izkorišča za svoje ozke potrebe.

Računalniški strokovnjak se ne more uveljavljati zgolj s kompentenco, strokovnostjo in ekspertizo, temveč mora znati in si upati z argumenti posegati tudi v tehnološko in poslovno strategijo računalniškega podjetja. Preprosto zaradi tega,

ker strokovnost
poslovodnih in vodilnih delavcev
največkrat bistveno zaostaja
za potrebami in zahtevami
strokovnega in uspešnega vodenja
računalniškega podjetja.

Ta drža (predispozicija) računalniškega strokovnjaka pa zahteva določeno stopnjo intelektualnega asketizma, na katerem temelji v pretežni meri tehnološka in poslovna uspešnost podjetja v civilno razvitem svetu. V razdobju trajajoče krize je poziv k pozitivnemu intelektualnemu asketizmu nujen, imperativen; kot pridobljena življenska filozofija in delovna navada pa naj bi se prenašal na mlajše rodove in vam ostal osmišljen tudi do konca življenja.

Spominjam se, da sem začel prevajalnike brzkone predavati v šolskem letu 1972/73 na Fakulteti za elektrotehniko v Ljubljani, takoj po svetovnem kongresu IFIP (International Federation for Information Processing) leta 1971 v Ljubljani. Takratni pokongresni čas je bil naklonjen strokovnim vsebinskim inovacijam na fakulteti. Z neznansko težavo sem se tedaj lotil pisanja učbenika za prevajalnike z metodološko naslonitvijo na tedaj strokovno najbolj priznano delo D. Griesa: Compiler Construction for Digital Computers, J. Wiley, 1971. V okviru podiplomskega študija sem v tedanjem razdobju predaval teorijo formalnih jezikov, ki se je ukvarjala s problematiko formalizacije oziroma avtomatizacije prevajanja. To kar je v tedanjem in tudi v kasnejšem obdobju bilo problematično tako za mene kot za študente, je bilo pomanjkanje realnih, industrijskih zahtev za izdelavo prevajalnikov. Posledica tega so bile "teoretične" vaje, ki jim je manjkala poanta realne nujnosti. Manjkala so tudi bistvena programirna orodja, ki olajšujejo in dandanašnji že domala avtomatizirajo pisanje oziroma programiranje slehernega prevajalnika. Podobni pedagoški problemi so se pojavljali tudi v okviru predmeta operacijskih sistemov.

Petnajst let kasneje so ti pedagoški problemi bistveno manjši in so pretežno odvisni le še od prizadevnosti, razgledanosti in strokovnosti učitelja in študentov. Pred vami je odličen učbenik, in sicer A. V. Aho, R. Sethi, J. D. Ullman: Compilers - Principles, Techniques, and Tools, Addison-Wesley Publ. Co., 1986, ki je utemeljen z več kot desetletno delovno tradicijo in praktičnimi izkušnjami avtorjev. Uporaba različnih prevajalnikov za visoke programirne jezike in specialnih programskih paketov za podpiranje programiranja in še posebej za razvoj prevajalnikov tudi na osebnih računalnikih ta posel bistveno olajšuje in približuje, znižuje uporabno abstraktnost in uporabniško udomačuje. Današnji študent in tudi njegov učitelj ne doživljata več občutja odtujenosti učnega predmeta od predmetne prakse.

Danes je mogoče poučevanje tako operacijskih sistemov kot prevajalnikov praktično že nasloniti na problematiko paralelnih sistemov, ki ni le raziskovalno aktualna, inovativna in tako motivirajoča, temveč je v določeni meri tudi preživetveno nujna, predvsem za vas, ko boste slej ko slej postajali nosilci novega civilizacijskega razvoja, nove industrializacije.

5

Uporaba računalnikov in prevajanje programirnih jezikov sta povezana s posebno kategorijo mišljenja, ki je

razumevanje.

O razumevanju bi rad povedal nekaj besed predvsem zaradi tega, ker je razumevanje kot kognitivni, tj. spoznavajoči proces v človekovih korteksih izrazito lep primer informacijske rekurence. Preizkusimo naš diskurz o razumevanju tako, da se vprašamo preprosto: Kaj je razumevanje?

S pojavom vprašanja se v našem korteksu sproži proces vpraševanja o razumevanju, in sicer v prvi vrsti o razumevanju razumevanja. O razumevanju lahko razpravljamo le, če razumevanje razumevamo. Torej je osnovni problem razumevanja (v človeku in stroju), da ga je mogoče

razumevati le skozi razumevanje samo, skozi razumevanje samo-po-sebi. Razumevanje nastaja v okviru razumevanja samega in ga ni mogoče razumevati izven območja razumevanja.

Razumevanje je kot fenomenologija živega korteksa kompleksna rekurentna pojavnost, ki je omogočena kot nevrofiziološka procesnost v živih nevronih samih (celični procesorji) in med živimi nevroni (sinapsni procesorji), ki jih obdaja korteksno okolje z glia celicami. Ta korteks je povezana in organizirana struktura, biološki stroj, ki je paralelno, serijsko, vzvratno substančno in procesno zapleten in prepleten. Korteks je paralelni stroj, ki je procesno upravljan z osnovnimi in v nje vloženimi procesi. V ti. um je mogoče vstaviti poljubno informacijo (program, idejo), ki ta um vznemirja (perturbira) in prevzema njegovo trenutno kontrolno funkcijo (pozornost).

V tem oziru je delovanje umetnega (neživega) stroja, tj. računalnika, podobno. Operacijski sistem je vložen v računalniško (tehnološko) okolje, trenutni (prevedeni) uporabniški program pa v okolje operacijskega sistema. V izvajalnem času uporabniškega programa prevzame kontrolno funkcijo nad računalnikom uporabniški program, ki tako realizira uporabniško funkcijo in vrne ali tudi med izvajanjem vrača kontrolo operacijskemu sistemu. Tudi paralelni računalniki zmorejo izvajajati podobno kot živi korteksi hkrati več procesov enega samega uporabniškega programa. Cilj uporabe paralelnih računalnikov je dosegati paralelno izvajalno organizacijo, ki bi bila podobna korteksni in jo celo presegati tako po številu paralelnih procesov kot tudi z načinom njihove medsebojne komunikacije in koordinacije.

6

Diskurz, ki smo ga pravkar opravili in ki nam daje slutiti možnosti o razumevanju razumevanja, je seveda izven območja ti. racionalistične tradicije. Učna predmeta operacijskih sistemov in prevajalnikov sta seveda v celoti zasidrana v racionalizmu, saj so računalniki le racionalistični stroji. Umetna inteligenca prihaja na sam rob racionalizma, ko govori o inteligenci, ki pa ni več racionalistična kategorija. Razumevanje inteligence je povezano z novim pojmovanjem informacije. Stroji, ki bodo imeli vgrajeno inteligenco, ne bodo več zgolj-racionalistični.

Iskanje odgovorov na vprašanje "Kaj je razumevanje?" bo predstavljalo srž problematike novih raziskovanj, metodologije in načrtovanja strojev. Kajti tudi stroji bodo morali imeti vgrajene osnovne mehanizme razumevanja. Razumevanje bo tedaj postalo osnovni element umetne inteligence, torej tudi osnovni element programiranja kot programiranja razumevanja.

Slovstvo

Priporočeno (obvezno):

A. V. Aho, R. Sethi, and J. D. Ullman: Compilers - Principles, Techniques, and Tools. Addison-Wesley Publishing Company, Reading, Ma., 1986.

Pomožno (priporočljivo):

A. P. Železnikar: Prevajalniki. Fakulteta za elektrotehniko v Ljubljani, 1984.

T. Winograd, F. Flores: Understanding Computers and Cognition. Ablex Publishing Corp. Norwood, NJ, 1986.

A. P. Železnikar: Information Determinations I. Informatica 11 (1987), No. 2, 3-17.

MAN – MACHINE COMMUNICATION: SPEAKER – INDEPENDENT SPEECH RECOGNITION

INFORMATICA 1/88

Zdravko Kačič
Bogomir Horvat
Štefan Greif
Faculty of Technical Sciences, Maribor

UDK 681.3:534.44

Abstract. With a proper selection of feature description methods sufficient accuracy of the speaker - independent speech recognition should be achieved. The speech signal features are described with the three sets of feature (the set of descriptive features , the set of selected features, and the set of characteristic features). The feature description methods are described with the three sets of map (the set of descriptive features map, the set of selected features map , and the set of characteristic features map). As an example two feature description methods are dismembered - zero - crossing method (variant a and b) and method of formant frequencies energy classes (variant a and b). It has been shown that the Fourier transformation as a map of descriptive features was more convinient as a measurement of interval lenght between two succesive zero-crossings of the signal. The mapping rule in variant b of the method of formant frequencies energy classes was more convinient map of selected features than the mapping rule in variant a. With these more convinient maps the smallest feature overlapping and consequently a better average recognition accuracy (greater than 92.5%) has been achieved.

Keywords. Speech recognition, independent speaker, recognition base element, set of features, set of maps, recognition accuracy, feature description, feature overlapping.

1. Introduction

In spite of fast development of computer tecnology, digital signal processing theory, phonetics , linguistics and artiffical intelligence, solution of the problem regarding man - machine communication on the basis of speaker-independent speech recognition, remains entirely the job of the feature.

To solve this problem a very good knowledge of all above mentioned fields shall be required .

Nowadays commercial speech recognition systems recognize successfully a large vocabulary of words only in the case of isolated word recognition and are mostly dependent on speaker [10]. In systems which recognize connected speech or even continuous speech the vocabulary of words is much smaller.

A special problem represent systems which recognize the speaker-independent speech signal.

In systems which recognize isolated words the extent of vocabulary decreases already (on about 40 words). Of course, the same recognition accuracy as in the speaker-dependent systems shall be required.

Today the speaker-independent continuous speech recognition systems exist as prototypes only and their vocabulary is not greater than 10 words [10].

The 'complexity', and first of all the great 'heterogeneity' of speaker-independent speech signal represent one of the major obstacles for solving this problem more successfuly.

The speech signal can be recognized on the basis of the so called recognition base elements (words , syllables, phonemes etc.).

This paper describes some problems which appear in the process of speaker-independent speech recognition on the basis of phoneme recognition, and indicates the ways of their solution .

'Features overlapping' of different recognition base elements (i. e. when features were described by feature extraction methods and presented in n-dimensional space) and great dispersion of features of same base element (i.e. when spoken by an independent speaker) represent a great problem in the speaker independent speech recognition process.

Features overlapping mostly means recognition error when the classification is made.

Speech signal characteristics can be described by various features extraction methods [1 ,3,6, 7].

Differences in speech signal features of the same recognition base element (for an independent speaker) are due to speaker's age, sex , psychophysical condition , etc [1,2,6].

Different feature extraction methods describe speech features in different ways. Consequently, the rate of features overlapping is different and depends upon the method which has been used.

To achieve high recognition accuracy of recognition base elements , the features overlapping of different base elements should be as small as possible.

So the proper selection (definition) of a feature extraction method for particular groups of recognition base elements is an important condition for a good recognition accuracy.

In the next sections the feature extraction process of recognition base elements with definition of some mapping rules and features sets shall be described and the basic notion with dismembers of two feature extraction methods - zero - crossing method (variant a and b) and method of formant frequencies energy classes (variant a and b) shall be presented.

2. Description of recognition base element features

We shall try to describe feature extraction process by means of three sets of speech signal features and three sets of map.

The three sets of features are : the set of all descriptive features, the set of all selected features and the set of all characteristic features. Each of the sets should be mapped with the following mapping sets : the set of descriptive features maps, the set of selected features maps and the set of characteristic features maps.

Such distribution of speech signal features has been assumed to estimate the convinience of a single feature extraction method which shall be used in the feature extraction process.

Analytic evaluation of the importance of a single feature description and with it a definition of 'optimum' description might also be possible.

Let us describe now briefly single sets of features and the sets of maps.

A) Sets of features

a) Set of the recognition base elements articulation - $\mathcal{A}$

$$\mathcal{A}=\{A_1, A_2, \ldots A_n, \ldots A_N\}, \quad (1)$$

N - the number of different recognition base elements

$$A_n=\{A_{n1}, A_{n2}, \ldots A_{nm}, \ldots\}, \quad (2)$$

A_{nm} - the m - th articulation of the n-th recognition base element

$$A_{nm}=\{a_{nm1}, a_{nm2}, \ldots a_{nml}, \ldots a_{nmL}\}, \quad (3)$$

L - the number of windows of the m - th articulation

a_{nml} - the l - th window of the m - th articulation of the n-th recognition base element

$$a_{nml}=\{a^1_{nml}, a^2_{nml}, \ldots a^u_{nml}, \ldots a^U_{nml}\}, \quad (4)$$

U - the number of elements in the l-th window

b) Set of all descriptive features - D

$$D=\{D^1, D^2, \ldots, D^i, \ldots\}, \quad (5)$$

D^i - the set of descriptive features described by the i-th description

$$D^i=\{D^i_1, D^i_2, \ldots, D^i_n, \ldots D^i_N\}, \quad (6)$$

D^i_n - the set of descriptive features of the n-th recognition base element described by the i-th description

$$D^i_n=\{D^{i1}_n, D^{i2}_n, \ldots, D^{im}_n, \ldots\}, \quad (7)$$

D^{im}_n - the set of descriptive features of the m-th articulation of the j-th recognition base element

$$D^{im}_n=\{D^{im}_{n1}, D^{im}_{n2}, \ldots D^{im}_{nl}, \ldots D^{im}_{nL}\}, \quad (8)$$

L - the number of windows of the m - th articulation of the j - th recognition base element

D^{im}_{nl} - the set of descriptive features of the l-th window of the m-th articulation of the n - th recognition base element

$$D^{im}_{nl}=\{d^{im}_{nl1}, d^{im}_{nl2}, \ldots, d^{im}_{nlK}, \ldots d^{im}_{nlK}\}, \quad (9)$$

K - the number descriptive features

c) Set of all selected features - S

$$S=\{S^1, S^2, \ldots, S^j, \ldots\}, \quad (10)$$

S^j - the set of selected features defined by the j-th description

$$S^j=\{S^j_1, S^j_2, \ldots, S^j_n, \ldots S^j_N\}, \quad (11)$$

N - the number of different recognition base elements

S^j_n - the set of selected features of the n-th recognition base element defined by the j-th description

$$S^j_n=\{S^j_{n1}, S^j_{n2}, \ldots, S^j_{np}, \ldots\}, \quad (12)$$

S^j_{np} - the p-th selected feature of the n-th recognition base element defined by the j-th description

$$S^j_{np}=\{s^j_{np1}, s^j_{np2}, \ldots, s^j_{npR}\}, \quad (13)$$

R - the number of elements of the p-th selected feature

d) Set of all characteristic features - C

$$C=\{C^1, C_2, \ldots, C^u, \ldots\}, \quad (14)$$

C^u - the set of characteristic features defined the u-th description

$$C^u=\{C^u_1, C^u_2, \ldots, C^u_n, \ldots, C^u_N\}, \quad (15)$$

N - the number of different recognition base elements

C^u_n - the characteristic feature of the n-th recognition base element defined by u-th description

$$C^u_n=\{c^u_{n1},c^u_{n2}, \dots ,c^u_{nv}, \dots c^u_{nV}\}, \quad (16)$$

V - the number of elements of the characteristic feature

B) Sets of maps

1) Set of descriptive feature maps - F_D

- elements of the set are mapping the set of recognition base elements articulation into the set of descriptive features

$$F_D=\{f_{D1},f_{D2}, \dots ,f_{Di}, \dots \}, \quad (17)$$

$$f_{Di}:\mathcal{A} \longrightarrow D^i \quad (18)$$

The map $f_{Di}:\mathcal{A} \longrightarrow D^i$ is surjective.

2) Set of selected feature maps - G_s

- elements of the set are mapping the set of descriptive features into the set of selected features

$$G_s=\{g_{s1},g_{s2}, \dots ,g_{sj}, \dots \}, \quad (19)$$

$$g_{sj}:D^i \longrightarrow S^j \quad (20)$$

The map $g_{sj}:D^i \longrightarrow S^j$ is surjective.

3) Set of characteristic feature maps - G_c

- elements of the set are mapping the set of descriptive features into the set of characteristic features

$$G_c=\{g_{c1},g_{c2}, \dots ,g_{cn}, \dots \}, \quad (21)$$

$$g_{cu}:D^i \longrightarrow C^u \quad (22)$$

The map $g_{cu}:D^i \longrightarrow C^u$ is surjective.

The map g_{cu} is mapping the set of descriptive features D^i into the set of characteristic features C^u so that the set of all intersection of the elements of set C^u is an empty set.

The set of all intersections of the elements of selected features set - S^j is not an empty set.

That means, that the elements of characteristic features set C^u are disjunctive sets . This is not valid for the elements of selected features set S^j.

If- $f_{Di}:\mathcal{A} \longrightarrow D^i$ and $g_{cu}:D^i \longrightarrow C^u$ are maps, then we may compose f_{Di} and g_{cu} to obtain a map $f_{Di} \circ g_{cu}:\mathcal{A} \longrightarrow C^u$.

We shall define such maps , which are mapping the set of recognition base elements articulation into the set of characteristic features.

3. An example of feature extraction method dismembers

Considering the maps and sets mentioned below, as an example the two feature extraction methods shall be dismembered. The first one is the so called zero-crossing method (method from the time domain) and the second one is the method of formant frequencies energy classes (frequency domain).

There are various variants of the zero-crossing method [5].

Almost all have in common the mapping rule of descriptive features , i.e. measuring the time between the two successive zero-crossings of a signal.

Single variant 'evaluates' these intervals in different ways.

We shall briefly describe two of them.

Elements of the descriptive features set are defined as:

$$d_k=\sum_j T_s \ , \ k = 1,2, \dots , K \quad (23)$$

where:

T_s is the time between two successive samples
j is the number of samples with equal sign
d_k is the lenght of k-th interval
K is the number of intervals
In this way , the set of descriptive features $D^i{}_{m_i}$ is composed of subsets which contain lenght of intervals between two successive zero-crossings.

$$D^i{}_{m_i}=\{d^i{}_{m_{i1}},d^i{}_{m_{i2}}, \dots ,d^i{}_{m_{ik}}, \dots d^i{}_{m_{iK}}\}, \quad (24)$$

Variant a (ZCa)

Elements of the selected features set $S^i{}_{np}$ are defined as:

$$s^i{}_{np}(\tau_j,\tau_{j+1}) = \frac{d(\tau_j,\tau_{j+1})}{P}, \quad (25)$$

where:

-$d(\tau_j,\tau_{j+1})$ is the number of intervals in the time class (τ_j,τ_{j+1})

Value of P is defined by

$$P=\sum_{j=0}^{k-1} d(\tau_j,\tau_{j+1}), \quad (26)$$

K is the number of all intervals .

The subset $S^i{}_{np}$ of selected features set $S^i{}_n$ is composed of elements which represent portion of intervals lenght in particular time classes.

$$S^i{}_{np}=\{s^i{}_{np1},s^i{}_{np2}, \dots s^i{}_{npR}\}, \quad (27)$$

variant b (ZCb)

Secondly , elements of the selected features set $S^u{}_{np}$ are defined as follows:

$$s^u{}_{np}(\tau_j,\tau_{j+1})=\frac{n(\tau_j,\tau_{j+1}).(\tau_j+\tau_{j+1})/2}{W.(\tau_{j+1}-\tau_j)}, \quad (28)$$

where

$n(\tau_j,\tau_{j+1})$ is the number of intervals in the time class (τ_j,τ_{j+1})
W is the window width
τ_j, τ_{j+1} are the boundary values of the j-th time class.

By means of factors $(\tau_j,\tau_{j+1})/2$ and $(\tau_{j+1}-\tau_j)$ a better evaluation of high and low frequency components should be achieved.

$$S^k_{np}=\{s^k_{np1},s^k_{np2}, \ldots ,s^k_{npT}\} \quad (29)$$

b. Method of formant frequencies energy classes (FFEC)

Like the zero-crossing method this method knows various variants as well.

All variants use the discrete Fourier transformation as the mapping rule of the descriptive features [6,9] :

$$G(s)=\sum_{u=0}^{U-1} g(u)\ \exp(-j2\pi su/U) \quad (30)$$

The subset $D^j m_{nL}$ of the descriptive features set $D^j m_n$ is composed of frequency samples.

$$D^j m_{n1}=\{G^j m_{n11},G^j m_{n12}, \ldots ,G^j m_{n1K}\}, \quad (31)$$

Variant a (FFECa)

To define elements of the selected features set S^t_{np} the following prescription has been used:

$$s^t_{np}(r)=\left(\sum_{u=f_m/R_\phi}^{f_{m+1}/R_\phi} \log G_v^2(u)\right)\Big/\left(\sum_{u=1}^{K} \log G_v^2(u)\right); \quad r=1,2, \ldots ,R \quad (32)$$

where

$G_v(u)$ is the u-th element in descriptive features set $D^j m_{n1}$
$s^t_{np}(m)$ is the m - th element in selected features set S^t_{np}
R_ϕ is the resolution factor of DFT
K is the number of all elements in the descriptive features set $D^j m_{n1}$
R is the number of elements in the selected features set S^t_{np}
f_m,f_{m+1} are the boundary values of the m-th formant frequencies class

The selected feature set S^t_{np} is composed of elements , which represent parts of common energy in particular formant frequencies classes.

$$S^t_{np}=\{s^t_{np1},s^t_{np2}, \ldots s^t_{npR}\}, \quad (33)$$

Variant b (FFECb)

This variant defines elements of the selected features set S^v_{np} as follows:

$$s^v_{np}(j)=\log G_{vmax}^2(j)\Big/\left(\sum_{m=1}^{M} \log G_{vmax}^2(m)\right), \quad (34)$$

where

$G_{vmax}^2(j)$ is the maximum frequency component in j-th formant frequencies class
M is the number of maximum components of all classes and the number of elements in the selected features set S^v_{np}

The subset of the selected features set S^v_{np} is composed of elements , which represent a portion of maximum frequency components in a single formant frequencies class.

$$S^v_{np}=\{s^v_{np1},s^v_{np2}, \ldots s^v_{npM}\} \quad (35)$$

Out of it arises a question how efficiency , or better 'convenience' of a single map should be estimated in order to be used in the base element recognition process .

For this purpose , the recognition results obtained by the dismembered feature extraction methods mentioned above , will be presented in the next sections.

4. Experimental results of isolated Slovene vowels recognition

The recognition of the five isolated Slovene vowels (/a/,/e/,/i/,/o/ and /u/) was carried out by the recognition experiment.

One hundred and ten articulations of each vowel, pronounced by 110 different speakers, has been performed. All articulations were recorded in an studio environment.

Speakers were of different age categories. Female - male rate was 3/7.

The speech signal was passed through a band-pass filter (600 Hz - 3.4kHz) and sampled at 10k Hz with a 12 bit A/D converter.

The time window width (W) was limited to 20 ms.

Because of such a great amount of different speakers we might presume that the recognition results (see Table 1) are the recognition results of an independent speaker.

Elements of the selected features set (for a single method) were combined into the feature vector and the number of vector elements was limited to ten :

$$Z_{np}=[z(1),z(2), \ldots ,z(10)], \quad (36)$$

Classification was made on the bassis of multivariate normal distributions with equal covariances [7].

Recognition results for single methods are given in the Table 1a-b.

Zero - Crossing Method										
variant a						variant b				
recognized as [%]										
	A	E	I	O	U	A	E	I	O	U
A	96.4	1.8	0.0	1.8	0.0	97.3	0.9	0.0	1.8	0.0
E	0.0	71.8	13.7	5.5	9.0	0.9	78.2	10.0	7.3	3.6
I	0.0	5.4	90.9	0.0	3.7	0.0	3.7	94.5	0.0	1.8
O	6.4	24.5	0.9	62.7	5.5	7.3	22.7	0.0	63.6	6.4
U	1.9	11.8	11.8	10.9	63.6	2.7	11.8	17.3	10.9	57.3

a

	Method of Formant Frequencies Energy Classes									
	variant a					variant b				
	recognized as [%]									
	A	E	I	O	U	A	E	I	O	U
A	83.6	1.8	0.0	8.2	6.4	97.3	0.0	0.0	2.7	0.0
E	6.3	70.0	16.4	2.7	4.6	0.0	92.8	5.4	0.9	0.9
I	3.6	16.6	69.0	3.6	7.2	0.0	4.5	92.8	0.0	2.7
O	13.8	4.5	1.8	58.1	21.8	1.8	0.0	0.0	92.8	5.4
U	5.5	7.2	1.8	10.1	75.4	0.0	0.9	0.0	10.0	89.1

b

Table 1a-b: Experimental results of five isolated Slovene vowels recognition

5. Efficiency of feature extraction methods

We shall now try to estimate efficiency of single maps, or better, their 'convenience' for the use in the base elements recognition process on the basis of recognition results.

By using map rules in the zero-crossing method (variant a) a somehow better recognition accuracy was achieved only for the vowel /a/ (96.4%) - less for the vowel /i/. For the vowels /e/ , /o/ and /u/ a rather worse recognition accuracy was achieved.

The variant b of the zero-crossing method showed a little bit better recognition results, but the rate of vowels recognition error was rather the same as at the variant a.

The reason for a worse recognition accuracy when zero-crossing method was applied , should be searched in the usage of the map of descriptive features.

In this method (for both variants) the measurement of intervals lenght as mapping rule for mapping the descriptive features was used.

Anyhow, this 'function' is 'incapable' to 'ignore' phase changes between particular frequency components in a signal.

In other words - it is a phase dependent function.

Human ear is insensitive to phase changes in a speech signal [4], whereas this is not true for the 'simple' measurements of intervals lenght.

Two signals with equal frequency components and with different phases sound the same. However, they can be formed in very different subsets of descriptive features , if the rule of the measuring interval lenght between the two successive zero-crossings of the signal was used as the mapping rule.

This is of great importance for phase changes at low frequencies (first two formants), which have ussualy the greatest amplitude and as such a greater influence on the zero-crossing rate.

Fig. 1a shows the first three elements of the feature vector formed by the zero-crossing method (variant a) and the method of formant frequencies energy classes (variant b) for all articulations of the vowel /e/. They 'describe' low frequencies in the frequency spectrum. Fig. 1b represents the last three elements of the feature vector for all articulations of the vowel /e/ , for the both methods. They describe high frequencies in the frequency spectrum.

It could be noticed, that the dispersion of the first three elements of the feature vector formed by the ZCa method (marked by ^), is much greather than the dispersion of the feature vector elements formed by the FFECb method (they are labeled as .).

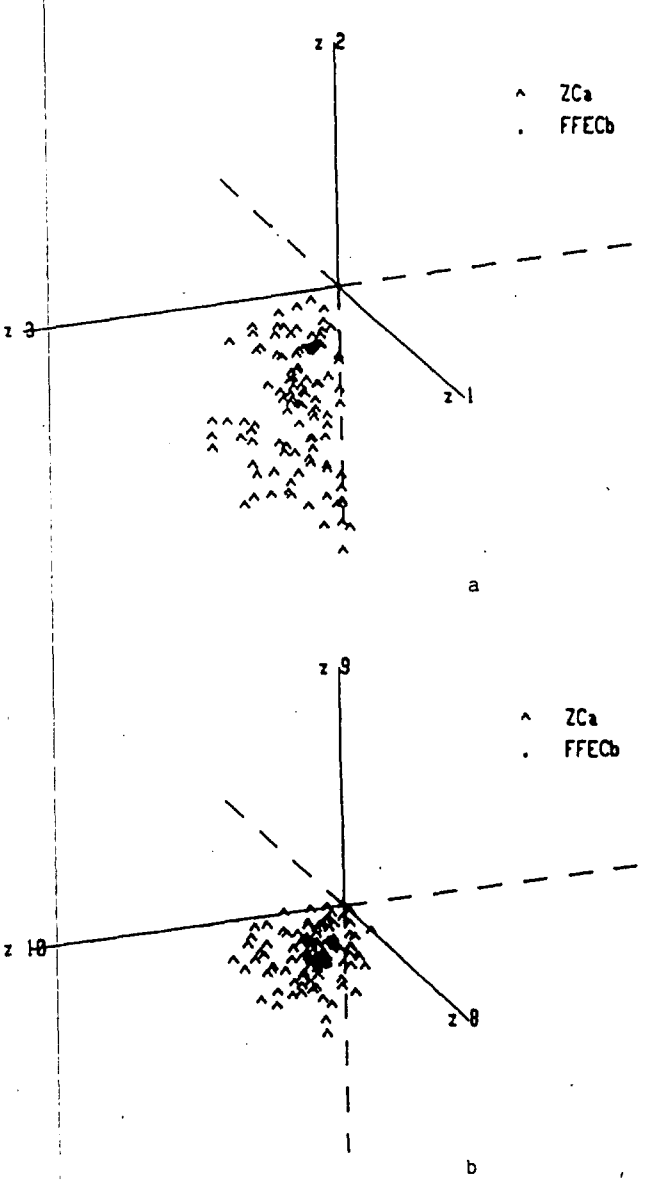


Fig. 1a-b : Distribution of the first three a) and the last three b) elements of the feature vector, for vowel /e/, formed by ZCa (^) and FFECb (.) methods.

A rather smaller dispersion could be seen at the last three elements of the feature vector formed by the ZCa method.

In the both cases the dispersion of feature vectors elements formed by the FFECb is very similar.

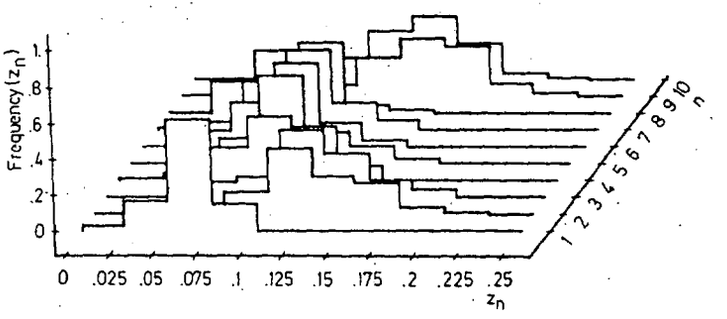
From the above mentioned the importance of the fact of phase changes between single frequency components in the frequency spectrum might be noticed - first of all , for low frequencies being present in a speech signal of an independent speaker.

This fact also indicates the recognition results of the vowels /o/ and /u/, for which first of all the first formant is dominant.

From the Fig. 2a it can be also seen, that features description of recognition base elements with measurement of interval lenght as mapping rule of descriptive features was less successful as with Fourier transformation. This should be evident from the dispersion rate of single feature vector elements , which is greater than the one for the other two methods. This is particulary true for the second and the third element of the feature vector (they first of all describe the first formant).

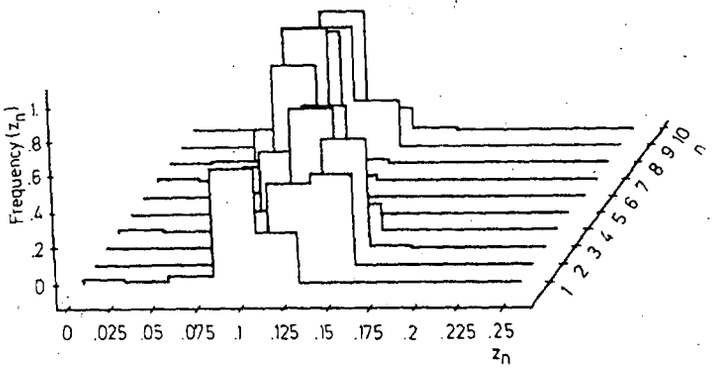
Comparision of recognition results for variants FFECa and FFECb (see Table 1b) and considerations of dispersion rates of vector elements for both variants (Fig. 2 b - c) give indication of the fact that common normalized energy of single formant frequencies classes calculated by this variant was a 'worser criteria' than the ratio of normalized energy of maximum components was. This might point out that the common energy contents per single formant frequencies classes for some recognition element change with an independent speaker. It was reflected as an increase of dispersion for almost all elements of the feature vector (Fig. 2b). This means a worse recognition accuracy (Table 1b).

'FFECa method'



b

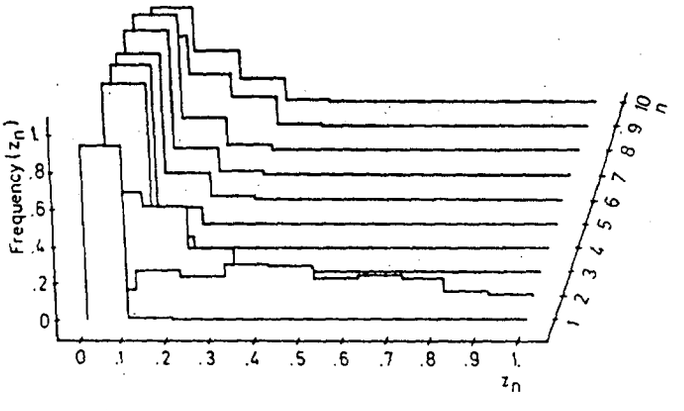
'FFECb method'



c

Fig. 2a-c : Histograms of the feature vector elements , for vowel /e/, formed by ZCa a), FFECa b) and FFECb c) methods.

'ZCa method'



a

A better recognition accuracy and the smallest features vector elements dispersion was achieved when the mapping rule of method FFECb was used.

The mapping rule of the selected features for this variant 'enables selection' of frequency components. In each class only the maximum component was choosen. In this way only energy of the maximum component for a particular class was described. But because of the fact that ten formant frequencies classes were defined, they are not all maximum frequency components of formants.

With this variant the best average recognition accuracy was achieved - greater than 92.5 %.

6. Conclusion

By the speaker-independent speech recognition such features maps should be defined that 'differences' in speech features, appearing in the case of an independent speaker shall be expressed as small as possible. That means that such functions should be defined where features overlapping was as small as possible.
This should be valid for maps of descriptive features (e.g. measurement of intervals lenght - discrete Fourier transformation) and for maps of selected features (e.g. variant a - variant b of FFEC method) as well.

The mapping rules discussed in our paper showed that the discrete Fourier transformation as the mapping rule for the descriptive features maps and the variant b of the FFEC method as the mapping rule for the selected features maps gave the best recognition results.

With above mentioned methods the smallest features overlapping and consequently the best average recognition accuracy has been achieved - i. e. more than 92.5 % .

References

[1] L.R. Rabiner and R. W. Schafer , Digital Processing of Speech Signals, Prentice - -Hall , Englewood Cliffs , NJ , 1978.

[2] A. H. Seidman and I. Flores , Handbook of Computers and Computing , Van Nostrand Reinhold Company , New York , 1984.

[3] R. De Mori and C.Y. Suen , New Systems and Arhitectures for Automatic Seech Recognition and Synthesis , Springer - Verlang, Berln, 1985, Chap. 1, pp. 1 - 72 .

[4] James C. Anderson , "Improved zero-crossing method enhances digital speech " , EDN Magazine , vol. 27, No. 20 , october 13 1982 , pp. 171 - 174 .

[5] R.J. Niederjohn and P.F. Castelaz, "Zero - crossing analysis methods and their use for automatic speech recognition " ,Proc. IEEE Computer Society Workshop on Pattern Recognition and Artifical Intelligence, 1978 , pp. 274 - 281 .

[6] F. Fallside and W.A. Woods, Computer speech processing , Prentice - Hall , Englewood Cliffs , NJ , 1985

[7] J. C. Simon, Spoken Language Generation and Understanding,D. Reidel Publishing Company, 1980 , pp. 129 - 145

[8] R.J. Senter,Analysis of Data, Scot,Foresman and Company,Illinois , 1969 .

[9] I. H. Witten, "Digital storage and analysis of speech", Wireless world, november 1981, pp. 44 - 48 .

[10] P. Willich, "Putting speech recognizers to work" , IEEE Spectrum , april 1987 , pp. 55 - 57 .

[11] Z.Kačič, Š.Greif and B.Horvat, "Uspešnost metod opisovanja skupnih značilnosti osnovnih elementov govornega signala", Elektrotehniški vestnik , Vol. 53 (1986), No. 3, pp. 121 - 129 .

PROVING THE CORRECTNESS OF DIGITAL HARDWARE DESIGNS USING PROLOG

INFORMATICA 1/88

Z. Brezočnik
B. Horvat
University of Maribor

UDK 681.3:325.6.08

ABSTRACT. An approach is presented for automatic formal verification of digital hardware designs using Prolog. Prolog is used both as a representational language for specifying the structure and the behaviour of a design and also as an inference mechanism for proving its functional correctness. A design in this model is composed of hierarchicaly organized modules. Each module is represented as a finite state machine. Validation of design correctness is made by formal proof as an alternative to the traditional approach which utilises simulation. The verification proceeds as follows: a) writing a design specification and a description of its realization in Prolog, b) deriving a design behaviour from the interconnections of its components and their behaviours, c) showing equivalence between the specified and the derived behaviour. The system has enough domain specific and general mathematical knowledge to perform the proofs largely automatically. Designs can be handled from the lowest transistor level up to the architectual levels. Some large designs including a simple computer have already been verified.

1. INTRODUCTION

A hardware or software designer must be able to decide whether the design meets its functional specification. Currently three different approaches exist for answering this question.

The first approach to enshure functional correctness is to develop the design from the specification by such a methodology that ensures it can't be incorrect. In software design it is exemplified in research of automatic programming. In hardware design automated techniques exist for dealing with elements of designs that are most tedious and prone to human error (such as wire routing or PAL generation). Silicon compilation techniques for automatic generation of complete designs from specifications are under development. This approach is perhaps the most attractive, but it is also the most difficult to achieve, because it faces an eventual astronomically large search space of design alternatives. A problem of automatic synthesis is so difficult, that useful general purpose systems for automatic synthesis are not expected in the near future.

The second and the most common approach to validate a design is an accurate simulation of it and trying it on test cases. In exhaustive simulation every possible combination of inputs should be tried for every possible internal state. The number of all possible input patterns can be extremely large even for simple devices. A multiplier for two 16-bit integers faces over four billion different inputs. It's clear that exhaustive simulation is no longer feasible. We can only select a subset of input patterns and hope to extrapolate from them to determine the correctness or failure of the design. The selection of an adequate subset of test cases is very difficult. Such partial simulation can detect the existence of an error, but can never guarantee that there are no more errors in the design.

There is an alternative third approach - a formal verification of a design with mathematical proof. If the formal verification succeds, the design will match its specification for all input patterns. Research in formal verification involves artificial intelligence techniques. It requires a formalism for representing the structure and the behaviour of digital systems and also an inference mechanism, which will expertly manage astronomically large search spaces. A success of the verification depends much on a built-in general mathematical knowledge base. Because of uncompleteness of the simulation and a bad prognosis for the soon automatic synthesis this approach seems to be the most promising in the nearer term.

Wagner was a pioneer with his research in this field. He has used a nonprocedural functional language for digital design description and a theorem prover in a first order predicate logic. The proof must be guided manually. Because of the description language the use is limited to low levels of a design. After that, Gordon has developed his methods for hardware modeling and verification based on different hierarchical levels with an interactive theorem prover (1). Initially, proofs were made manually, but more recently with an interactive theorem prover.

In this paper we present our verification system, named VERDIS (2) for formal verification of digital hardware designs using Prolog. VERDIS represents an automatized version of a

variant of Gordon's approach. In Fairchild Laboratory for Artificial Intelligence the VERIFY system was developed based on the similar principles (3). VERDIS has successfully verified some experimental designs with an interesting degree of complexity.

2. DESIGN REPRESENTATION IN VERDIS

A digital system in VERDIS is represented as a collection of hierarchically organized modules and their interconnections. A module is considered as a finite state machine (FSM)

$$A = \{X, Y, Z, \delta, \lambda\} \qquad (1)$$

It has a finite set of input (X) and output (Z) ports and a finite set of internal state variables (Y). VERDIS supports FSM of types Mealy and Moore and also ordinary decision circuits, if Y={}.

A module is either a primitive with no internal structure (basic building block of the design) or composition of the form

$$||(N_1, .., N_k),$$

where the components N_i are themselves modules. The input (output) ports of a compound module are the input (output) ports of its components. Compound modules are formed by linking some output ports of some components to some input ports of other components. External ports of the module are those ports that are not linked. Each port has an associated signal type which specifies the domain for signals passing through it. Signals in digital system are viewed differently according to the conceptual level in which they are considered: as voltage levels, as logic values ,as bits , as numbers or even as higher-level objects. Signals in VERDIS may have following types:

-*booole* (Boolean truth values true or false)
-*bit* (binary digits 0 and 1),
-*integer(N)* (integers in the range 0-2^{N+1}-1),
-*integer* (natural numbers),
-*booleZ, bitZ, integerZ(N)* and *integerZ*
(high impedance signal types).

Structural and signal hierarchy allow more succint description. Signals on a higher hierarchical level don't show unnecessary details of lower-level signals, but cary the same information.

For conciseness the behaviour of the FSM is described by two sets of equations , rather than an exhaustive table:

$$D^1y = \delta(x,y) \ ; \ x \in X, \ y \in Y \qquad (2)$$
$$z = \lambda(x,y) \ ; \ x \in X, \ y \in Y, \ z \in Z \qquad (3)$$

Equation (2) gives internal states as functions of inputs and current state. Equation (3) gives output signals as functions of inputs and current state. A dictionary of available functions that can be used in expressions for describing module behaviour currently consists of arithmetic (+,-,*,^) , logic *(not,and,or,xor)* and branching functions *(if,case)* and also some special functions for such operations as wiring signals on a bus and work with memories *(joinfn, bushfn, rfetch, fetch,store,...)*.

Module definition consists of Prolog facts and rules. In addition to constructs for specifying type of the module, ports, states, components, internal connections and behavioural equations the description language supports several useful constructs: constants, parameters, arrays, bit-wise connections and *equ* operator for calling some predefined modules from the system library. Let's illustrate some constructs mentioned so far by considering an example of a one-bit multiplier. It's constructed from a collection of 2-to-1 multiplexors, each of which has inputs *inx, iny*, control input *ctrl* and output *out*.

```
% Definition of a one-bit multiplier in
% terms of 2-to-1 multiplexsors

module(bitmult(N)).

port(bitmult(N), in(Bitmult),input,integer(N)).
port(bitmult(N), ctrl(Bitmult),input, boole).
port(bitmult(N),out(Bitmult),output,
     integer(N)).

constant(Bitmult(N),
         null(Bitmult),0,integer(N)).

part(bitmult(N), mplx(Bitmult, I), mux2):-
     index(0,I,N).

linked(bitmult(N),bit(I,in(Bitmult)),
       inx(mplx(Bitmult,I))):- index(0,I,N).
linked(bitmult(N), ctrl(Bimult, I),
       ctrl(mplx(Bitmult, I))) :-index(0,I,N).
linked(bitmult(N),    out(mplx(Bitmult,    I)),
       bit(I,out(Bitmult))):- index(0,I,N).

output_equation(bitmult(N), out(Bitmult) :=
if(ctrl(Bitmult), in(Bitmult), null(Bitmult))).
```

In this example, *N* is a parameter specifying the bit wide of the multiplicand (i.e., the most significant bit represents 2^N). Due to declarative power of Prolog a suitable indexing of part and connections can be used. The construct *index(0,I,N)* means simply that *I* can take any value from *0* to *N* .

3. THE VERIFICATION PROCESS

The key principle of VERDIS is that given the behaviour of components of a system and their interconnections, it is possible to derive a description of the behaviour of the whole system. The derived behaviour can then be compared with a specification of the intended behaviour of the system. If a design contains an error, a discrepancy between both behaviours can be detected and the design corrected.

Before of the verification some basic checks are made to enshure that nontristate outputs are not wired, that connected signals have equivalent types, that every output and state variable has an equation ... . These checks have proved to be very useful in finding typing and logical mistakes in design specification.

On request for verification of a module VERDIS checks, if the correctness of this module type has been already proved, in which case another proof is not necessary and the verification succeeds immediately. If a module is a primitive, its correctness can be assumed. If the correctness of a module is unknown, VERDIS uses a depth first search to recursively verify each of its part and then a module as a whole.

The next step in verifying a module is to derive a behavioural description of a composite module from the behaviours of its components with their interconnections. Each component module has a set of output and a set of state equations. The union of all these equations is in fact an implicitly specified behaviour of a module. It contains internal variables (signals at parts inside the module and state variables of component modules). The unnecessary internal information should be hidden. With subsequent substitutions of internal variables with

behaviour equations of component modules all internal variables are eliminated. If any frequently used internal variable has a very complicated equation it may be better not to eliminate it to avoid even larger equations. In such cases VERDIS first derives and evaluates the expression for the immediate variable and refers to it in behaviour equations of the module where needed. The final result of this step is a set of derived output equations and a set of state equations.

At this point we can try to prove that the derived behaviour description of the module is equivalent to the behavioural specification. In most cases a mapping between them is an exact equivalence - isomorphism. We have to show that corresponding equations in both automata are identical. The proof of identity is generally a hard problem. It requires much mathematical knowledge about functions, which can be used in equations. In more complex cases the correspondence between automata is a homomorphism rather than exact equivalence. A homomorphism may have a structural or a behavioural form or both. Structural homomorphism occurs, when automata are functionally identical but different in structural description. Behavioural or temporal homomorphism occurs, when the same automata is viewed with different time-scales. VERDIS currently works only for isomorphic machines.

The derived and the specified behaviour are compared for each output and each state. Proof of design correctness requires the ability to prove that a given equation (specified behaviour as a left side and derived behaviour as a right side)is an identity. The equation is first checked to see whether it is recognized as a trivial identity or whether it is a trivial non-identity. If the equation is not trivial, VERDIS tries to choose the best strategy for proving the indentity. A strategy selection depends on a form of left and right side of the equation and on function, operators and types of the variables involved. The repertoire of strategies contains: algebraic simplification, Boolean canonicalization and enumeration.

Algebraic simplification is the most general strategy, which tries to prove the identity of left and right side of the equation with simplification, expansion and canonicalization. Simplification is implemented with a general recursive Prolog procedure that recursively simplifies a given expression using simplification rules for the involved operators and functions. Expansion is used when a function is observed on only one side of the equation. The definition of the function is then substituted for its call and the resulting expression is simplified. Canonicalization tries to deal with combinatorial problems because of commutativity and associativity of some functions *(+,*,and,or...)*.

If only Boolean variables occur in the equation a more straightforward strategy of Boolean canonicalization is chosen. During six subsequent steps the left and the right side of equation are transformed to a lexically ordered complete disjunctive normal forms and then compared. It's faster than algebraic simplification.

Sometimes no other strategy but enumeration may be applied. An enumeration is made over a minimal necessary number of variables. For a selected set of variables each possible combination of their values is generated and substituted into the equation, which is then simplified to 'true' or 'false'. In some special cases it's not necessary to generate all possible combinations but only some of them. Partial enumeration may save much computation. VERDIS avoids the use of enumeration, because it's usually very space and time consuming.

If VERDIS doesn't have enough knowledge to select a strategy, an interactive mode is entered. Currently a user can insist in the algebraic simplification, in the enumeration on all or just some of the variables and can also simply assume the equation to be true or false. It's always possible to enlarge VERDIS's knowledge base and automatize the strategy selection for such cases.

Proving identity of an equation is the main and the most difficult problem in the verification process. The use of any procedural programming language with deterministic organization would not be a natural way for solving this problem. This fact is one of the most significant arguments for realization of VERDIS with the nonprocedural language Prolog. A Prolog program is a pattern directed system. The proof proceeds with an activation of that verification rule which is currently appliable to the left and to the right side of the equation. Such realization has a high degree of modularity. The knowledge base can simply be enlarged by adding new rules without any other changes.

4. TWO COMPLEX EXAMPLES

VERDIS has tackled some complex designs including modules *d74* and *host*. *d74* is an arithmetic unit(inputs *inA*, *inB*, *inC* and outputs *out1* and *out2)*, which is intendend to compute sums of products.

*out1 = inA*inB + inA*inC*
*out2 = inA*inC + inB*inC*

At the top level it contains three multipliers and two adders. The multipliers consist of slices, each of which contains a one-bit multiplier, a shifter and an adder. The adders are built from fulladders, which are built from invertors and 2-to-1 multiplexors. Multiplexors are built from logic gates. The logic gates are themselves described at two levels: at Boolean algebra level and at lower transistor level with tristate signals and stored charge. *d74* is parameterized in bit-wide of inputs. An instance that has 2-bit inputs involves 21 different types of module with nine levels of structural hierarchy. The design contains 1902 primitive parts, including 1016 transistors. The entire listing of the proof trace occupies about 1300 lines.

The next example is a register-transfer level description of a simple computer *host* with eight operations: HALT, JMP, JZRO, ADD, SUB, LD, ST and NOP. The computer is divided into a control section and a data section. The data section is built upon a single 16 bit data bus. Six register (the program counter, the accumulator, the instruction register, the argument register, the buffer register and the memory register) and also an arithmetic-logic unit and RAM are connected to the bus. The control section contains a microprogram ROM, a microprogram counter and a microinstruction decoder. VERDIS has proved the correctness of this computer at the microinstruction level.

5. DISCUSSION OF THE RESULTS

VERDIS is a large Prolog program that occupies about 61 kbytes of code for interpretation. It currently runs on a VAX 8800 under ISJ Prolog interpreter (4).

The main restriction on the complexity of examples which currently can be tackled is the amount of workspace required by Prolog. To deal with large systems, it is necessary to verify them a piece at a time or to restart verification when it aborts due to lack of space. Since VERDIS promptly records the progress of verification in the database, restarting does not result in duplication of work.

Due to the hierarchical decomposition of design description the work done in proving the correctness of a complete system is linear in terms of the number of types of module in its design, rather than linear in terms of the number of primitive components. For complex designs with many thousands of primitive components, the computational saving can be very great.

At present, the program performs only functional verification, with no consideration of timing. It appears relatively straightforward to introduce the notation of time in our description, but adding the appropriate inferential machinery will probably require more effort. However, dealing with truly asynchronous systems requires a rather different approach with revision of the representation and proof mechanism, perhaps in direction of a temporal logic (5).

The verification in VERDIS runs largely automatically, what is an advantage in comparison with the Gordon's approach, where the proof has to be guided by a user. VERDIS can be compared with a similar system VERIFY which has been developed in the Fairchild Laboratory for Artificial Intelligence. VERDIS introduces some useful new elements: possibility of macro calls for predefined modules in the system library; retaining of internal variables with over-complex expressions (i.e. bus output), new verification strategies of Boolean canonicalization and partly enumeration, more heuristics in variable selection for the enumeration etc.

6. CONCLUSION

VERDIS is our first attempt of formal verification which has already shown that digital designs with an interesting degree of complexity can be handled. It should be tried on many other real designs and it's knowledge base should be enlarged to become a real design tool, but abilities of this approach are yet evident.

The decision for Prolog as a description and an implementation language seems to be correct. Pattern matching and backtracking in Prolog are very powerful tools in the verification process. VERDIS is currently interpreted on a VAX 8800 under IJS Prolog interpreter(4). If we have any Prolog compiler, the speed of execution of the compiled code will increase ten times at least.

VERDIS should be only one component of larger system supporting hardware designs. This would integrate programs for design entry, simulation, verification, diagnostics etc.

REFERENCES

(1) M. Gordon , J. Herbert, Formal hardware verification methodology and its application to a network interface chip, IEE Proceedings, Vol. 133, Pt. E, No. 5, September 1986, 255-270

(2) Z. Brezocnik , Hardware verification, Master thesis, Tehniska fakulteta, Maribor, November 1986

(3) H. G. Barrow, VERIFY: A Program for Proving Correctness of Digital Hardware Designs, Artificial Intelligence, Vol. 24, No. 1-3, December 1984, 437-491

(4) J. Stojanovski, IJS Prolog Reference Manual, Institut Jozef Stefan, Ljubljana, 1986

(5) B.C.Moszkowski, A temporal logic for reasoning about hardware, Procedings Sixth International Symposium on Computer Hardware Description Languages, Carnegie-Mellon University, Pittsburgh, 1983, 79-90

UDK 681.519.7

Peter Kolbezen
Institut »Jožef Stefan«

A communication mechanism based on the exchange of the distributed topology information is discussed. A particular reconfiguration technique is treated to handle the exchange of topology information and thus to maintain the necessary routing tables. This mechanism handles the reconfiguration dynamically. Inmos Transputer concepts are introduced and applicated on the two-dimensional square interconnection structure.

"Rekonfigurabilni" večprocesorski sistemi. Prispevek obravnava komunikacijski mehanizem, ki je zasnovan na spremembah topologije komunikacijskih poti med mikroprocesorskimi enotami sistema. Namenjen je posebni t.im. "rekonfigurabilni" tehniki, ki obravnava spremembo informacije o topologiji in na ta način vzdržuje potrebne razpredelnice povezav medprocesorskih komunikacij. V predlogu so vpeljani koncepti Inmosovega transputerja, kot vozliščnega procesorja v povezovalni dvodimenzionalni kvadratni strukturi.

1. INTRODUCTION

Highly parallel computing structures promise to be a major application area for the milion-transistor chips that will be possible in just a few years. Such computing systems have structural properties that are suitable for VLSI implementation. The key attributes of VLSI computing structures are

- simplicity and regularity
- concurrrency and communication
- computation intensive VLSI.

The choice of an appropriate architecture for any electronic system is very closely related to the implementation technology. This is especially true in VLSI. The supervisory overhead incurred in general-purpose supercomputers often makes them too slow and expensive for real-time and signal and image processing. Progress in VLSI technology has lowered implementation costs for large array processors to an acceptable level. Algorithmically specialized processors often use different interconnection structures. The matching of the structure to the right algorithm has a fundamental influence on performance and cost effectiveness.

An alternative to the design of a globally synchronous array is to achieve a self-timed system through the use of asynchronous handshaking mechanisms established between neighboring processing elements. These self-timed implementations are commonly referred to as **wavefront arrays** /8,12,21,23,25/. The wavefront array combines the systolic pipelining principle with the dataflow computing concept. A wavefront array processor may be used either as an attached processor interfacing with a compatible host machine or as a stand-alone processor equipped with a global control processor. Such system consists of the processor array(s), interconnection network(s), a host computer and interface unit.

Exploitation of the dataflow principle makes the extractions of parallelism and programming for wavefront arrays relatively simpler.

A family dynamically interconnected or reconfigurable VLSI processor arrays allow an array to support a large class of algorithms. Such structures usually involve significant hardware overhead. Reconfigurability of array structures based on switching lattices has been proven to be useful for solving problems related to fault tolerance. Fault tolerance is an important concern in systolic and wavefront arrays because of the large number of processor elements they may have.

The paper concludes with suggestions as to how a recofigurable sistem may be designed with constructing a wavefront array with the family of Inmos Transputers /20/. Transputer chip is an Occam-language-based design that provides hardware support for both concurrent computation and communication. It adopts the now-popular RISC architecture. Its features make it a powerful building block for constructing concurrent processing networks. The transputer's links are the hardware representation

of the channels for process communication. There is an intimate relationship between transputer channel links and the communications protocol envisaged for wavefront arrays.

2. ADAPTIVE SYSTEMS

One of the major objectives of the adaptive systems is to be rearrange the configurations to match the needs of different applications. A such flexible **Reconfigurable Multi-microprocessor Systems** (RMmS) should include provisions for insertions (extensions) and deletions (reductions).

In general, a node in a system becomes aware of the entire configuration either by a central network control (CNC) or as result of distributed exchange of the configuration (topology) information. The CNC approach can be utilized to implement a fixed routing mechanism. The necessary routig tables, which are computed using the system topology information, are distributed to the participating nodes, once and for all. For a RMmSystems which does not have an CNC, it is necessary to compute routing tables externally and to provide each node with a copy of the full configuration information. Given the network topology, the shortest distance algorithems, such as Dijkstra's algorithm, can be used to compute all the shortest paths between the local node and the rest of the system /5,6/.

The provision of a distributed reconfiguration mechanism eliminates the need for a central network control and a copy of the full configuration information at each node. Instead, each node is responsible for maintaining partial routing information so as to be able to communicate to its neighbouring nodes.

The handling of configurational changes /5,7, 12/ in multi-computer/processor systems is a necessary part of their design. It is required for realiability, modularity, and extensibility. It is the dynamical reconfiguration which is important. The routing mechanisms, such as

- random
- flooding
- ideal observer, and
- adaptive

can respond dynamically to the changes. The frst two techniques can utilize any live link and node. A node is not necessarily aware of the full system topology. The third technique needs a central network control which ideally is assumed to watch the entire system and is responsible for incorporating traffic and topology changes. The last technique, i.e. adaptive routing, is basically a flow control mechanism which can also respond to the topology changes in terms of some delay function.

In this paper, a communication mechanism based on the exchange of the distributed topology information is discussed. A particular reconfiguration technique is proposed to handle the exchange of topology information and thus to maintain the necessary routing tables. A topology message is sent only if there is an indication of a change in the configuration. This mechanism handles the reconfiguration dynamically. The link, node, or link and node failures or unexpected occupation and the reverse changes, i.e. new or free link, new or free node, or new or free link and node cases, are made known to every live node in the system in a finite time.

The treatmented reconfigurations technique in first part of the first work on this topic is discussed in connections with uniform and dense networks. It is based on Baran's hot-potato routine /2/. A correctness proof of a similar algorithm has also been presented by Tajibnapis /1/.

3. INITIALIZATION

The discussed reconfiguration algorithm is basically an adaptive algorithm. It is intended, however, to be controlled by the configurational changes rather than traffic changes. It will initialize the system by setting the initial configuration and then adapt the system, dinamically, to further possible changes in topology. The changes occur either as a result of failures or unexpected occupation (link, node link, and node) or as a result of new links, nodes, or links and nodes joining the system. The topology message format is as shown as:

header	destination	source	distance	crc

Given a network of **N** nodes, it first undergoes an initialization phase whereby each node detects the adjacent operable links and thus exchanges this information with neighbouring nodes. The information is passed in the form of standard format units (topology messages). The new neighbour receives the full account of the topology information available at the detecting node, in the form of one topology message per accessible node. Note that the links are assumed to be bi-directional, i.e. if a link **(a,b)** is live so is the link **(b,a)**. In practice this is not always the case.

The topology message contains two fields relevant to reconfiguration: identification of destination node **i** and the shortest distance between the sending node **s** and node **i**. A node generates, and may modify, a shortest distance table as shown in Fig. 1.

Destination	0	1	2	...	k	...	n-1
Distance	D(0)	d(1)	d(2)	...	d(k)	...	d(n-1)

Fig. 1 Shortest distance table at node k

The entries are indexed by the node identification and the entries themselves are the shortest distance between the corresponing destination node and the local node.

4. HANDLING INSERTION OF A NEW NODE

Now, suppose that node 1 of Fig. 2 detects a new neighbour, say node 17 in the same figure, and the rest of the system is uniform with the configuration known and node 17 has no ties with any other node in the system, except node 1. This is a major change in the network. The topology message exchange propagates until the configurational change has been detected at every node in the system. A receiving node

needs to send topology messages to its neighbouring nodes only if the topology message received has changed the previous topology information, i.e. the shortest distance table.

This process can best be visualized by a top-down tree, as shown in Fig. 3.

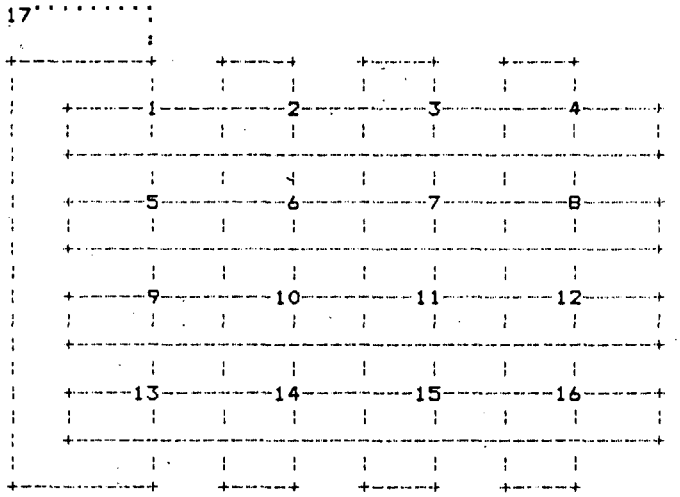


Fig. 2 Node insertion

The tree is extracted from the network shown in Fig. 2. The messages on the topological changes originated at node 1 stop at the terminal nodes when the configuration information, i.e. the shortest distance tabele, has reached settlement. Note that the branches represent the communication links and the vertices represent the nodes as numbered in Fig. 3.

A node is crossed if it has already been traversed once by an equivalent topology message. Two topology messages are considered equivalent if the destination fild, the distance field, and the sending node are the same. For this example a node is traversed at least once for the configurational change to be transparent to the system since node 17 has no ties with the rest of the network. However, the node may have to be traversed more than once in cases where the network is loaded with other traffic which may cause delays on certain paths. The topology messages that then take longer paths can reach a node before those taking sorter paths. The distance and the routing tables alter every time a topology change on a shorter path arrives. Every change is then fanned through the system.

Starting from the ROOT node 1, there are five levels of transmission. The system, with the exception of node 17, is expected to settle in five main periods. For node 17, however, (n-1) main periods are required to receive the complete topology information from the ROOT, where n is the number of nodes in the original network. This is because the ROOT needs to send the shortest path vector to node 17, where each path is transmitted in the form of a topology change message.

In practice, a new link does not always belong to a first-time node joining the system. The node might already have been taking part in the system. On the other hand, more than one link can become live at the same time. This occurs if one or more nodes join the system all at once, with all the adjacent links coming up simultaneously.

5. HANDLING FAILED LINKS

Link failures or occupationes handled differently. Obviously a failed or occupated link cannot take part in a transmission. Upon detection of a such link out of m+1 links, the prepared topology message only needs to be broadcast over the remaining m links. From then on, the procedure for handling a topology message is the same as for the live link detection case. Now, for clarity suppose that the failure/occupation of a particular link disconnects the adjacent node from the rest of the system, which is the reverse of the node insertion case discussed in the previous example. The failure/occupation of the link (1,17) causes the system in Fig. 2 to settle in five main periods. This is because the system requires five levels of transmission (see Fig. 3).

6. TRANSPUTER NODES

The transputer /13,15,20/ is a comporatively new hardware concept. The transputer products are aimed at highly parallel concurrent computing applications. The range of these products from Inmos offer system designers high band-

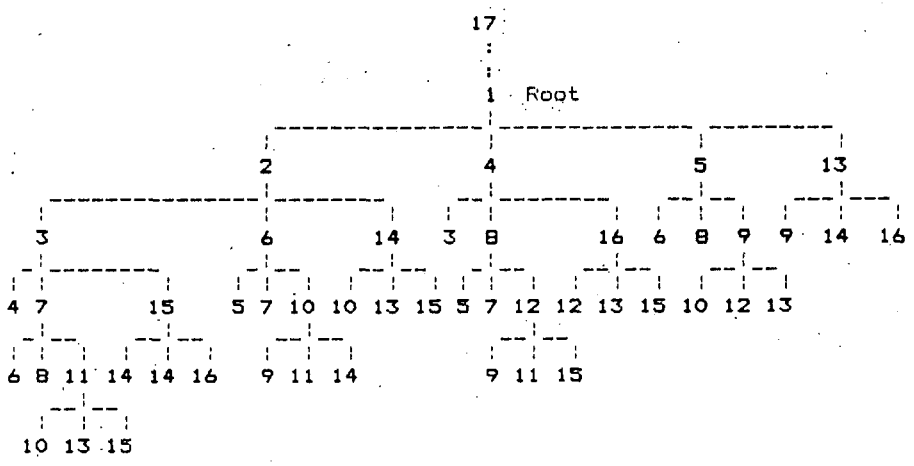


Fig. 3 Topology message exchange tree.

width interprocessor communications without a shared memory bus. It is a programmable VLSI device with communications links for point-to-point connection to other transputers. The software concept upon which transputer interprocess communication is based was originally propounded by C.A.R. Hoare /3/. All transputer range products share same basic common logical properties. These are:

- the ability of the processor chip to sopport external memory, but with the recommendation that processors should not share memory;
- the provision of high bandwidth serial links for interprocessor communication;
- hardware support for on-chip simulated concurrency, and for multi-processor parallel computing;
- low level software development in a high level structured notation.

A link between two transputers provides a pair of "occam" channels, one in each direction. A link between two transputers implemented by connecting a link interface on one transputer to a link interface on the other transputer by two one-directional signal lines. Each signal line carries data and control information.

Each message is transmitted as a sequence of single byte communication, requiring only the presence of a single byte buffer in the receiving transputer to ensure that no information is lost. After transmitting a data byte, the sender waits until an acknowledge is received. The acknowledge signifies both that a process was able to receive the acknowledge byte, and that the receiving link is able to receive another byte. The sending link reschedules the sending process only after the acknowledge for the final byte of the message has been received.

Data bytes and acknowledges are multiplexed down each signal line. An acknowledge is transmitted as soon as reception of a data byte starts (f there is room to buffer another one). Consequently transmission may be continuous, with no delays between data bytes.

Transputer also support program modularity with the possibility of late decisions about which part of the network particular software should reside and a degree of run time choise about which processor should be used for running particular tasks. There is nothing in the transputer architecture and organization which contradicts the idea of dynamic reconfiguration of the network. But dynamic reconfiguration of the transputer network is not supported at present by Inmos except that they provide a cross-bar swich with some low-lewel software support. The reconfiguration is needed in part because the hardware for transputers supports only a finite (and normaly very small) number of links from each node.

For the configuration above on the Fig. 2 which use all four links of every transputer in a array node may be used the B003 Inmos transputer board. There is no provision for booting the code which on the B003 must be done via a transputer link. The problem has been avoided so far since it destroys the symmetry of the network and makes the program unnecessarily complicated for these examples. It is shows one way that the problem may be solved for a 16 node 4 x 4 two-dimensional array on the Fig. 2. An extra transputer, which may be on a B001 or B002 board, is used as the boot node which can connected via a UART to a host machine. This processor is node 17 on the Fig. 2 and it may be include in the loop of nodes 1,5,9 and 13.

7. IMPLEMENTING THE RECONFIGURATION ALGORITHM

A network of needs to be reconfigurated if one of the following topological changes occurs:

1. a new link is introduced (or a failed / occupated link is repaired or again free and returned to the system);
2. a new node is joined to the system (or a failed node is repaired or again free and reinserted);
3. a link fails
4. a node fails.

It is important that topological change of a permanent nature is made known to the entire network in a finite time /5/. This is where the reconfiguration mechanism comes in. It controls the flow of topological information both distributedly and dynamically. On the other hand, temporary changes such as short time link, node, or link and node failures need not be fanned through the system. Instead a temporary blockage on the failed unit, or rerrouting at the adjacent nodes, can be more advantageous until the failure state is removed. This is because the global reconfiguration mechanisms use up an important fraction of the available communication capacity of the network which would otherwise be utilized for the actual information transmission. For permanent failures, however, the reconfiguration mechanism pays off with higher reliability, adaptability, and extensibility.

In a real life system, permanent and temporary configuration changes need to be distiguished before taking any action. This can be handled by employing a re-try and time-out mechanism. Before declaring a failure as permanent the detecting node waits for a finite duration of time during which the failure is declared temporary and the necessary precautions are taken accordingly. This mechanism can also be used for the new links or nodes joining the system. However, a live link or a live node detection mechanism can be incorporated with the type of joining, i.e. whether it is a permanent attachment or a temporary one. Reconfiguration is considered only for the practical case of permanent topological changes.

8. DATA BASE OF THE RECONFIGURATION ALGORITHM

The reconfigurations alghorithm operates basically on two tables. The distance table (DT) which records the distance (path length) of each node in the system from the host node via each of the neighbouring nodes; the routing table (RT) which records the shortest paths only. For each destination a maximum of m paths can be identified, where m is the number of neighbours. The DT is an n by m table, where n is network size.

Figure 4 shows the DT a node 1 of the 16-node network given in Fig. 2. In a regularly connected homogenous network all the distance tables are of equal size. An entry **d(k,i,j)** is the distance of path (k,i) via the neighbour **X(j)**, where i=1,2,...,n and j=1,2,...,m. The nodes that are not accessible from node k have a correspoding entry of infinity in the table. For example, node 1 appears inaccessible from itself via any of its neighbours.

The size of routing table (RT) depends upon the

Destination	Via neighbouring nodes			
	2	4	5	13
1	infinity	infinity	infinity	infinity
2	1	3	3	3
3	2	2	4	4
		. . .		
16	4	2	4	2

Fig. 4 Distance table for node 1 of a 16-node network

specific routing mechanism being employd. In fact, RT is arranged using the distance table. An entry r(i,k) or RT indicates the neighbour node, say X(j), via which the node i is at a minimum distance from the host node k, i.e.

$$d_m(k,i,j) = \min_{1 \leq m} (d(k,i,j)).$$

The entries of RT change only if the minimum of the corresponding DT row changes. It would save processing time if a shortest distance (SD) table were used alongside the distance table. We have

$$SD(i) = d_m(k,i,j).$$

Figure 5 gives the routing table and the shortest distance table given in Fig. 4. Note that r(1,1) = 0 and SD(1) is infinity which means that node 1 cannot send a message to itself by means of routing tables. For applications where the communication protocols maintain information flow between the processes rather than the processors (nodes) independent of where they are residing, we can have SD(k) noninfinity, where r(k,k) = 0.

The reconfiguration algorithm operates on those entries which are related to the topological changes. For example, if a topological change concerns node 1, only the ith level of distance, routing, and shortest distance tables need to be referred to after detection. An exception is the detection of the change which requires either the shortest distance table to be sent to the new neighbour or the column of the distance table corresponding to the failed link to be updated.

Destination	1	2	3	. . .	15	16
Next node	0	2	2	. . .	4	13

(a) Routing table (RT)

Shortest distance	infinity	1	2	. . .	3	2

(b) Shortest distance table (SD)

Fig. 5 Routing and shortest tables at node 1 of a 16-node network

9. THE RECONFIGURATION ALGHORITHM

The reconfiguration algorithm is structured into three basic subalgorithms. Algorithm 1 handles the link and/or node failures (or occupation). A node failure or occupation are interpreted as the the simultaneous failures (or occupation) of the links. Algorithms 2 handles a new link and/or node coming up. A node is declared new/free if all the links connected to it become alive simultaneously. Algorithm 3 handles the topology messages which indicate the configurational changes. These algorithms are given in detail down. A reconfiguration protocol is complete only with a specific routing mechanism, queue management algorithms, and input-output handling routines at the lower lever of communication.

Figure 6 showes a system flow diagram of the reconfiguration mechanism excluding the flow level communication protocol routines. A more detailed description of three algorithm has been written by Bozyigit. Some simulation results using the reconfiguration algorithms are reported by Bozyigit and Faker /6/.

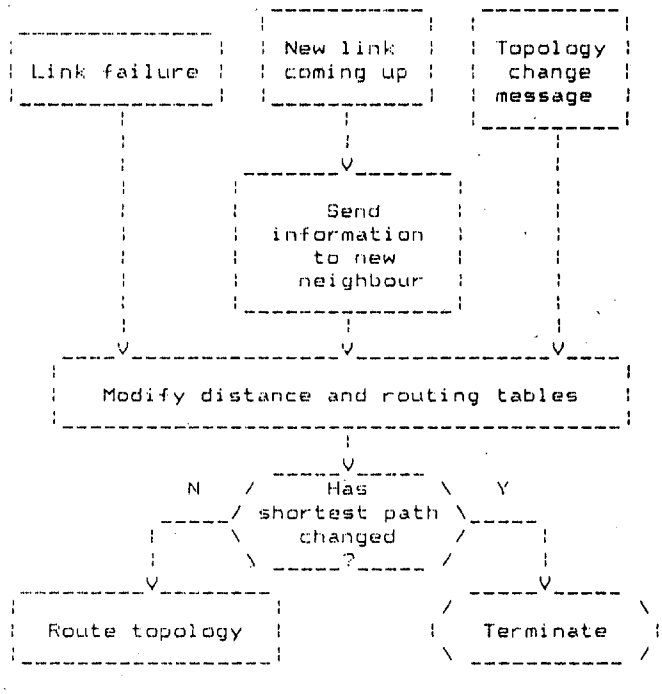


Fig. 6 Organization of reconfiguration algorithm

10. DESCRIBTION OF THE RECONFIGURATION ALGORITHMS

A. Nomenclature for the reconfiguration algorithms

a = source node
n = destination node
c = node adjacent (neighbour) to node a

d^a_{bj} = distance between a and b via the jth neighbour of a (the superscript a is omitted wherever it is obvious)

r^a_b = the first neighbour on shortest path between a and b indicated by a neighbour of a on that path

TM_b^a = topology message compiled at **a** to be sent to **b**

c <- TM_b^a = send TM_b^a to c

V_b^a = shortest distance between **a** and c

W_c^a = ith neighbour of **a**

X_i^a = ith neighbour of **a**

t = (a temporary variable)

B. Algorithms 1: Handles a new link coming up at node a

1. (a new link detected at the local node **a**)
   Link (a,c) becomes live;

2. (update distance and routing tables)

   d_{cj}^a : = 1 where jth neighbour = c,

   V_c^a : = 1,

   r_c^a : = c;

3. (preparate topology message)
   (set distance topology field of topology message)

   dist (TM_c^a) : = 1,

   (set destination field of topology message)

   dest (TM_c^a) : = c;

4. (send topology message to neighbouring nodes)

   X_j^a <- TM_c^a for j = 1,2,...,m and $X_j^a \neq c$;

5. (send available topology information to new neighbour)
   (form topology message)

   dist (TM_i^a) : = V_i^a,

   dest (TM_i^a) : = i for i = 1,2,...,n.

   (send topology change messages to new neighbour)

   TM_i^c : = TM_i^a.

6. Exit.

B. Algorithm 2: Handles link failures at node a

1. (a link failure or occupation detected at node a)
   link (a,c) failed or occupated;

2. (update the distance table)
   (save distance vector correspoding to failed/occupated link)

   t_i : = $d_{i,j}^a$,

   (set vector $d_{i,j}^a$ to infinity)

   $d_{i,j}^a$: = infinity where X_j^a = c, and

   i = 1,2,3,...,n;
   j = index of the failed or occupated link

3. (update routing tables and prepare topology messages)

   for i : = from 1 to n with step 1 do

   if $V_i^a = t_i$ then

   V_i^a : = min$|d_{i,j}^a|$, j ∈ m

   r_i^a : = X_j^a where $V_j^a = d_{i,j}^a$,

   TM_i^a : = V_j^a,

   (send topology message).

   TM_i^Y : = TM_i^a (where Y = X_j^a) for

   j = 1,2,...,m, except for

   X_i^a = c,

   fi,
   od;

4. Exit

C. Algorithem 3: Handles arrived topology message at node a

1. (detect topology message at local node a)

   TM_b^c arrives at a;

2. (update distance and routing tables)
   (assign new distance for (a,b) path)

   $d_{b,j}^a$: = $TM_b^c + W_c^a$,

   (find new min distance (a,b) path)

   t_b : = min $|d_{b,j}^a|$, j ∈ m

   (assign the shortest path)

   if t_b not = V_b^a then

   r_b^a : = X_b^a where $t_b = d_{b,j}^a$

   V_b^a : = t_b

   (prepare topology messages)

   dist (TM_b^a) : = V_b^a,

   dest (TM_b^a) : = b^a,

   (send topology message to neighbours)

   X_j^a <- TM_b^a for all j ∈ m

   fi;

3. Exit

E. Algorithm 4: Round-robin routing algorithm

1. (input a message at a) TM_b^c arrives

2. (find the shortest path (c,b))

   if c = (m-1) then
   c : = 0
   fi,

   (next node)

   c : = r_b^a;

3. (message joins output queue)

   OQ_c : = TM_b^a;

4. Exit.

11. CONCLUSION

Algorithmically specialized processors often use different interconnection structures of reconfigurable wavefront array processors /16,25/. There are a number of traditional algorithms which set up data structures in the first phase of processing, then applay for instance matrix arithmetic techniques to process the data, and in a file phase access the data using a different data paths. A trivial example is the hardware for a shift register with parallel in/serial out or parallel and serial in and out. The **mesh** is used for dynamic programming and the **heksagonally connected mesh** for L-U decomposition /12 /. The **tours** is used for transitive closure. The **binary tree** is used for sorting and the **double-rooted tree** is used for searching.

Each B003 board from the Inmos family has four transputers connected in as ring. There are therefore two links per transputer which can be connected exterally. In this manner there are a wide variety of networks that can be configured using only B003 boards:

- In the example of the shift register a ring of n transputers can be implemented with n/4 B003 boards. The transputers are connected serially as the process "node" together with four channels (two input, two output).

- A two dimensional array of 64 transputers (8 x 8) can be implemented with sixteen B003 boards. Four arrays of n channels (for a structure n x n B003 boards) are connected (left to right, right to left, top to bottom and bottom to top) so that each node can send or receive data to or from any of its four neighbours. Eight placed channels for n=4 (four input, four output) are passed to the process "node".

- A folded binary structure of size 4 x 16 can be implemented with 16 B003 boards too. The boards are connected externally. Four arrays of channels of dimension n (where the network is of size 2×2^n) are connected in such a manner: adj.left, adj.right, diag.left and diag.right and so on where "adj is a mnemonic for adjacent conection and "diag" for a "diag" for a diagonal connection. And also in this example eight placed channels (for input, four output) are passed to the process "node".

- A cube connected cycle of size 4 x 16 can be implemented with 16 B003 boards. Each row is configured on a single B003 board. Three arrays of cannels are defined of dimension n (where the network is of size 2×2^n): clockwise.rote, anti.clockwise.rote and cross.route, where "clockwise" and "anti.clockwise" represent channels connected to nodes in the same row, and "cross" represents a channel connected to a different row. This cube connected cycle is topographically equivalent to a four-dimensional hipercube with four nodes at each "corner". Six placed channels (three input, three output) are passed to proces "nodes". A mapping function is not necessary for this configuration since each transputer has the same link address for its six channels.

Ideally it would be desirable to allocate each process $p_j \in P$ (where P is collection of processes p_j) to exactly one transputer and to allocate each channel $c_{jk} PQ\%C$ (where C is the collection of channels c_{jk}) to one intertransputer link between transputers t_i and t_k - given that the matching pair of channels c_{ik} and c_{kj} may be allocated to the one link l_{jk}. Ideally if there are multiple channels between processes each should be allocated to different links. However there are practical limits to the number of links available to each transputer and there are possible electrical signal and wiring problems with geographically remote links in a large network. It some cases it is necesary to forward messages passively through some transputers at nodes in a network because there are not enough links on each device to provide all the direct communication that is required. There are some examples where such messages forwarding is a heavy overhead and it may be desirable to avoid it by dynamic hardware reconfiguration between the different phases of the program execution. Then the performance of the system may be improved at the expense of more complex hardware.

12. REFERENCES

/1/ A correctness proof of a topology information maintenance protocol for distributed computer network. W.B.Tajibnapis, Comm.ACM, July 1977.

/2/ On distributed communication networks. P.Baran, IEEE Trans. on Communication Systems, Vol. Comm-12, 1967.

/3/ Communicating Sequential Processes. C.Hoare, Comms ACM, vol.21, No.8, August 1978.

/4/ Distributed fault tolerant computer system. D.A.Renels, IEEE Computer, March 1980.

/5/ Hardwired Resource Allocators for Reconfigurable Arcitectures. B.D.Rathi, A.R.Tripathi and G.J.Lipovski, Proc.Int'l Conference on Parallel Processing, IEEE, August 1980.

/6/ A topology reconfiguration mechanism for distributed computer system. M.Bozyigit and Y.Paker, The Computer Journal, 25, No.1, 1982.

/7/ Introduction to the Configurable, Highly Parallel Computer. L.Snyder, Computer, January 1982.

/8/ "Why Systolic Architectures?. H.T.Kung, Computer, Vol.15, No.1, January 1982.

/9/ On the design of algorithms for VLSI systolic arrays. D.I.Moldova, Proceedings of the IEEE 71, No.1, January 1983.

/10/ Flexibile Architecture Microcomputer Design. E.Zager and D.Tabak, Microprocessing and Microcomputer Design 11, 1983.

/11/ Reconfigurable architecture for VLSI processing arrays. M.Sami and R.Stefanelli, National Computer Conference, 1983.

/12/ Computer Architectures and Parallel processing. K.Hwang and F.Briggs, McGraw-Hill, New York, 1984.

/13/ OCCAM - an overview. D.May and R.Taylor, Microprocessors and microsystems, Vol.2, No.2, March 1984.

/14/ Concurrent VLSI Architectures. C.L.Seits, IEEE Trans. on Computer, Vol.c-33, No.12, December 1984.

/15/ The transputer implementation of occam. D.May and Shepherd, Proceedings of the international conference on fifth generation computer systems, ICOT, 1984.

/16/ On Supercomputing With Systolic/Wavefront Array Processors. Kung S.Y., Proc. IEEE, July 1984.

/17/ Post-failure reconfiguration of CSP programs. M.Shatz, IEEE Transaction on Software Engineering, Vol.SE-11, No.10, October 1985.

/18/ An Engineerable and Reconfigurable Cellular Array Processor. J.M.Cotton, Parallel Computing 85, 1986.

/19/ Hierarchical array processor (HAP)

featuring high realibility and high system performance. T.Ishikawa, S.Momoi, S.Shimada, Y.Ogawa, Proc. of the 1986 International Conference on Parallel Processing, August 1986.

/20/ Product Information, The Transputer Family. Inmos Corporation, Part No.72, 1986.

/21/ Synthesizing non-uniform systolic designs. C.Guerra and R.Melhem, Proc. of the International Conference on Parallel Processing, IEEE, August 1986.

/22/ Hardware reconfiguration of Transputer networks for distributed objekt-oriented programming. D.Q.M.Fay and F.K.Das, Microprocessing and Microprogramming 21, 1987.

/23/ Systolic Arrays-From Concept to Implementation. J.A.F.Fortes and B.W.Wah, Computer, Vol.20, No.7, July 1987.

/24/ Mapping Data Flow Programs on a VLSI Array of Processors. Mendelson B. and G.M.Silberman. Proc. 1987 Int'l Conf. Computer Architecture, June 1987.

/25/ Wavefront Array Processors-Concept to Implementation. S.Y.Kung, S.C.Lo, S.N.Jean and J.N.Hwang, Computer, Vol.20, No.7, July 1987.

STOHASTIČNI PRISTOP K ANALIZI UČINKOVITOSTI VEČPROCESORSKIH SISTEMOV

INFORMATICA 1/88

Slavko Mavrič
Peter Kolbezen
Institut »Jožef Stefan«

UDK 681.324:519.21

Predstavljen je stohastični pristop pri modeliranju in analizi učinkovitosti večprocesorskih sistemov s skupnim pomnilnikom, pri katerih je vsakemu procesorju pridružen še njegov zasebni pomnilnik. Zgrajen je model večprocesorskega sistema in definiran njegov režim delovanja. Postopek analize temelji na opisu tega modela s stohastično Petrijevo mrežo. Iz drevesa dosegljivosti te mreže dobimo parametre pripadajoče markovske verige. Analiza stacionarnega stanja te verige pa nas pripelje do iskanega pokazatelja učinkovitosti večprocesorskega sistema.

Stochastic Approach in Performance Analysis of Multiprocessor Systems. A stochastic approach in modelling and performance analysis of global memory based multiprocessor systems is presented. Every processor of such a system possesses its own private memory. A model of system has been built and its workload has been defined. Description of system activity with stochastic Petri net is the first step in a performance analysis. From the reachability tree of this Petri net it is possible to obtain parameters of associated Markov chain. Steady state analysis of this chain leads us finally to a desired performance index of a multiprocessor system.

1. Uvod

Pri ocenjevanju učinkovitosti računalniških sistemov srečamo tri osnovne pristope. To so: merjenje, simulacija in analitično modeliranje.

Prvi pristop vključuje meritve na samem sistemu pod dejanskimi delovnimi pogoji ter primerjalno merjenje s testnimi (benchmark) programi. Oboje zahteva popolno okolje ocenjevanega sistema.

Pri simulaciji opišemo dogajanje v računalniškem sistemu s simulacijskim programom. Pri tem so nam navadno v pomoč temu namenjeni programski jeziki. Iz izvajanja simulacijskega programa sklepamo o lastnostih simuliranega sistema.

Analitični model predstavlja z matematičnimi orodji opisano dogajanje v sistemu. Ocena učinkovitosti sistema nastopa kot analitična ali numerična rešitev tega modela. Velik pomen analitičnega modeliranja je v dejstvu, da ga lahko opravimo že v zelo zgodnji fazi načrtovanja računalniškega sistema in nam je v pomoč pri izbiri optimalne arhitekture.

V tem članku predstavljamo stohastični pristop k modeliranju in analizi večprocesorskih sistemov na osnovi markovskih procesov.

2. Določanje modela večprocesorskega sistema

Prvi korak pri ocenjevanju učinkovitosti nekega večprocesorskega sistema je v določitvi njegovega modela, ki mora zajeti vse bistvene lastnosti sistema. Model mora definirati množico sistemskih gradnikov, njih medsebojne povezave in razmerja ter režim delovanja celotnega sistema. Pri tem se moramo odločiti za primeren nivo abstrakcije predstavitve sistema. Nivo abstrakcije pomeni nivo detajlov, ki jih pri opisu sistema upoštevamo. Ta naj določa kaj so gradniki sistema ter kakšna so njihova medsebojna razmerja in pravila komunikacije. Nivo abstrakcije izberemo tako, da nam bo analiza na osnovi izbranega modela dala kot rezultat nek parameter, ki bo dovolj dobro opisoval učinkovitost sistema. Model mora torej vsebovati vse bistvene elemente, ki so za analizo potrebni, vse detajle, ki na tem nivoju abstrakcije niso pomembni, pa naj model ne zajame. Pri vrednotenju učinkovitosti računalniških sistemov lahko zasledimo različne nivoje abstrakcije:

- Aparaturni nivo. Ta nivo se nanaša na preizkušanje pravilosti nekega vezja z logičnega stališča. Gradniki modela tega nivoja so posamezna integrirana vezja. Za opis takega modela obstajajo posebni programski jeziki.
- Funkcionalni nivo. Cilj analize na tem nivoju je ovrednotenje obnašanja osnovnih funkcionalnih enot aparaturne opreme, kot so procesorji, pomnilniške enote, vodila itd., ko ti sodelujejo pri izvajanju neke operacije. Opis modela je podan s parametri, ki govore o hitrosti, kapaciteti in zakasnitvi posameznih enot.
- Sistemski nivo. Na tem nivoju se ovrednoti učinkovitost celotnega sistema na osnovi poznanih funkcij posameznih enot. Osnovni gradniki modela so aparaturno/programske enote, kot procesne enote, vhodno/izhodne in komunikacijske enote.

Izbira najprimernejšega nivoja abstrakcije pri modeliranju nekega sistema je odvisna od razmerja med točnostjo rezultatov na eni strani in kompleksnostjo ter ceno analize na drugi strani. Za ovrednotenje učinkovitosti večprocesorskega sistema nam ni potrebno poznati detajlov na nivoju posameznih integriranih vezij, prav tako pa za splošno namenski sistem ne potrebujemo natančno poznavanje aktivnosti sistema v realnem okolju neke aplikacije. Zato je funkcionalni nivo za analizo učinkovitosti večprocesorskih sistemov najprimernejši.

Osnovne predpostavke, na katerih temelji naš model večprocesorskega sistema so naslednje:

- osnovni gradniki modela so procesorji, pomnilniške enote in povezovalna mreža.

- vsak procesor P ima svoj zasebni pomnilnik ZP, ki je dostopen le njemu.
- vsak procesor lahko zahteva dostop do neke skupne pomnilniške enote SP, ki mu je dostopna.
- pomnilniška enota sprejme in po njej lastnem pravilu servisira zahteve posameznih procesorjev.
- povezovalna mreža PM povezuje procesorje in skupne pomnilniške enote. Če ni drugače rečeno, se predpostavlja, da so zakasnitve, ki jih vnaša povezovalna mreža, enake nič.

Splošen model, ki zadošča tem pogojem prikazuje slika 1.

Ko je nivo abstrakcije določen in so s tem poznani gradniki modela in njih medsebojne povezave, je potrebno definirati še režim delovanja tega modela. Pri sistemih, v katerih je vsakemu procesorju pridružen zasebni pomnilnik, izvajajo procesorji program iz tega pomnilnika. To izvajanja se prekinja s posegi v skupni pomnilnik. Aktivnost posameznega procesorja lahko smatramo kot zaporedje dostopov do različnih pomnilniških enot. Model režima delovanja nekega procesorja je torej v ponavljajočem izvajanju naslednje sekvence:
- dostop, ali zaporedje dostopov v zasebni pomnilnik
- dostop, ali zaporedje dostopov v skupni pomnilnik

Konfliktna situacija lahko nastopi pri uporabi povezovalne mreže ali skupnih pomnilniških enot. Vsak procesor je tekom delovanja v splošnem v enem izmed naslednjih treh stanj:
- Aktivno stanje. Procesor izvaja program iz svojega zasebnega pomnilnika.
- Stanje dostopa. Procesor izvaja bralno ali vpisovalno operacijo na neki skupni pomnilniški enoti.
- Čakajoče stanje. Procesor čaka na zahtevani dostop do skupne pomnilniške enote.

Takšna aktivnost procesorjev na funkcionalnem nivoju je lahko posledica povsem različnega obnašanja na sistemskem nivoju.

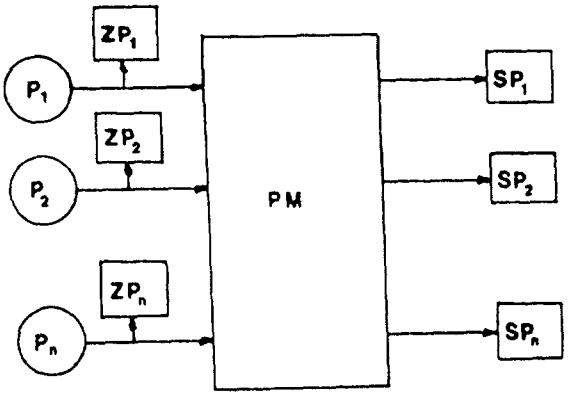
Zamislimo si tri večprocesorske sisteme. Pri prvem poteka komunikacija med posli s pošiljanjem sporočil preko pomnilniških vmesnikov, lociranih v skupnem pomnilniku. Pri drugem sistemu izvajajo procesorji neko operacijo nad vhodnimi podatki, ki jih najdejo v skupnem pomnilniku; tja tudi shranjujejo vmesne rezultate. V tretjem sistemu pa izvajajo procesorji program iz hitrih 'cashe' pomnilnikov, ki se morajo občasno osveževati iz skupnega pomnilnika. Če opazujemo obnašanje posameznega procesorja v vseh treh sistemih, zaključimo, da v vsakem primeru le-to rezultira v zaporedju dostopov do različnih pomnilniških enot. Zaporedju dostopov v zasebni pomnilnik sledi zaporedje dostopov v skupni pomnilnik in obratno.

Ti trije primeri kažejo, kako se lahko povsem različno delovanje večprocesorskega sistema na sistemskem nivoju, reducira na enako obnašanje na funkcionalnem nivoju. Režim delovanja na funkcionalnem nivoju je torej v vseh treh primerih enak.

Za ocenitev učinkovitosti večprocesorskega sistema na osnovi zgrajenega modela, moramo seveda najprej količinsko oceniti parametre tega modela. Pri našem funkcionalnem modelu so ti parametri časi trajanja zaporedij dostopov do zasebnega in skupnega pomnilnika, ki v celoti popisujejo režim delovanja posameznih procesorjev. Tej oceni sledi postopek analize. Ta nas na osnovi teh parametrov in topološkega opisa sistema vodi k izračunu veličine, ki nam bo dala dovolj dober opis učinkovitosti večprocesorskega sistema. Imenujemo jo performančni pokazatelj.

Pri določitvi režima delovanja, oziroma pri ocenitvi parametrov modela srečamo dva osnovna pristopa.

Deterministični pristop zahteva natančno poznavanje delovanja vsakega procesorja. Dobra stran tega pristopa je v tem, da v popolnosti



Slika 1. Splošen model večprocesorskega sistema

sledimo obnašanju celega sistema in s simulacijo določamo čas, ki je potreben za dokončanje neke poznane operacije. Slabost determinističnega pristopa je v dejstvu, da je potrebno popolno poznavanje režima delovanja procesorjev, kar v času načrtovanja sistema ni vedno možno in da so dobljeni rezultati relevantni le za tisto aplikacijo in ne kažejo funkcijske odvisnosti od parametrov modela.

Če uporabimo verjetnostni pristop, obravnavamo parametre režima delovanja kot naključne spremenljivke z neko znano porazdelitvijo verjetnosti. Stohastični pristop nam ne omogoča tako natančnega vpogleda v izvajanje neke operacije v sistemu kot deterministični pristop, zato pa nam daje rezultate kot funkcijske odvisnosti med parametri modela in performančnimi pokazatelji. S tem nam pomaga k boljšemu razumevanju dogajanja v kompleksnem večprocesorskem sistemu. Stohastično analizo je moč opraviti že v zelo zgodnji fazi razvoja samega sistema kot pomoč pri testiranju in primerjanju posameznih arhitekturnih različic s stališča izbranih performančnih pokazateljev. Verjetnostni pristop temelji na tem, da opišemo dogajanje v modelu z nekim stohastičnim procesom. Število stanj tega procesa je odvisno od števila vseh možnih stanj, ki jih lahko doseže obravnavani večprocesorski sistem, oziroma njegov model. Določiti moramo tudi medsebojne odvisnosti, oziroma pogojne verjetnosti med stanji procesa. Zaradi sprejemljive matematične kompleksnosti se za modeliranje podobnih sistemov pogosto uporablja razred stohastičnih procesov z markovsko lastnostjo, ki jim pravimo markovske verige.

3. Markovske verige

Definicije in rezultati tega poglavja se nanašajo na tip časovno zveznih markovskih verig (ČZMV). Pri njih lahko prihaja do prehodov med stanji v vsakem trenutku, medtem ko so prehodi pri časovno diskretnih markovskih verigah možni le ob posameznih trenutkih, ki so določeni z diskretizacijo časovne osi. Zato so zvezne verige primernejše za opisovanje dogajanja v nekem večprocesorskem sistemu.

Definicijo ČZMV, ki opisuje omenjeno markovsko lastnost, lahko zapišemo takole. Stohastični proces $\{X(t),\ t\geq 0\}$ je ČZMV če velja

$$P\{X(t_{n+1})=x_{n+1} \mid X(t_n)=x_n, \ldots, X(t_0)=x_0\} =$$

$$= P\{X(t_{n+1})=x_{n+1} \mid X(t_n)=x_n\},$$

$$t_{n+1}>t_n>t_{n-1}>\ldots>t_0. \qquad (3.1)$$

Ta markovska lastnost pomeni, da je verjetnostno

obnašanje procesa v prihodnosti odvisno samo od trenutnega stanja in ne tudi od zgodovine procesa. Desna stran gornje enačbe predstavlja prehodno verjetnost med dvema stanjema verige, ki jo za primer homogene ČZMV (kjer so prehodne verjetnosti neodvisne od časa opazovanja) lahko zapišemo v obliki

$$p_{ij}(\tau) = P\{X(t+\tau)=j \mid X(t)=i\}. \qquad (3.2)$$

Vse prehodne verjetnosti verige lahko zapišemo v obliki matrike $P(t) = [p_{ij}(t_1)]$. Seveda velja enakost

$$\Sigma_j\, p_{ij}(\tau) = 1, \quad j \in S. \qquad (3.3)$$

Verjetnost stanja i v trenutku t je po teoremu o popolni verjetnosti podana z enačbo

$$\pi_i(t) = \Sigma_j(p_{ji}(t)\pi_j(0)). \qquad (3.4)$$

iz katere vidimo, da je obnašanje procesa popolnoma določeno z začetno porazdelitvijo in prehodnimi verjetnostmi.

Naj bo

$$r_{ji}(\tau) = p_{ji}(\tau) \qquad j <> i \qquad \text{in}$$

$$r_{ji}(\tau) = p_{ji}(\tau)-1 \qquad j = i. \qquad (3.5)$$

Dokažemo lahko, da obstaja pri ČZMV s končnim številom stanj limita

$$q_{ij} = \lim_{\tau->0} (r_{ij}(\tau)/\tau). \qquad (3.6)$$

q_{ij} predstavlja v primeru i <> j časovno gostoto prehodov s katero proces prehaja iz stanja i v stanje j, medtem ko pomeni $-q_{ii}$ v primeru i = j časovno gostoto s katero proces zapušča stanje i. Velja enakost:

$$\Sigma_j\, q_{ij} = 0, \quad j \in S. \qquad (3.7)$$

Iz enačbe 3.4 lahko, s tako definiranimi qij, pridemo do enačbe za verjetnost nekega stanja i v poljubnem trenutku.

$$\pi'_i(t) = \Sigma_j(q_{ji}\pi_j(t)), \quad j \in S. \qquad (3.8)$$

Dobili smo torej sistem diferencialnih enačb, katerega rešitev nas vodi k verjetnosti posameznih stanj v poljubnem trenutku t. Zaradi težavnosti analitičnega reševanja takega sistema, uporabimo navadno numerične integracijske metode.

Kadar pri neki ČZMV obstaja stacionarna porazdelitev verjetnosti, nas ta navadno bolj zanima in je tudi lažje izračunljiva kot porazdelitev v poljubnem času t. Izkaže se, da obstaja stacionarna porazdelitev pri neskrčljivih homogenih ČZMV in da je neodvisna od začetne porazdelitve. Stacionarna verjetnost nekega stanja je enaka limitni vrednosti njegove verjetnosti, torej

$$\pi_i = \lim_{t->\infty} \pi_i(t). \qquad (3.9)$$

Tedaj tudi velja

$$\lim_{t->\infty} \pi'_i(t) = 0 \qquad (3.10)$$

in tako pridemo do sistema linearnih enačb

$$\Sigma_j(q_{ji}\pi_j) = 0, \quad j \in S, \qquad (3.11)$$

Ker je ta sistem enačb homogen, obstaja rešitev $\pi_i = 0$ za vsak $i \in S$. Če so enačbe linearno neodvisne, je to tudi edina rešitev sistema in stacionarna porazdelitev ne obstaja. Če pa so enačbe linearno odvisne, dobimo stacionarno porazdelitev z dodatkom normalizacijskega pogoja

$$\Sigma_i\, \pi_i = 1, \quad i \in S. \qquad (3.12)$$

Prav analiza stacionarnega stanja, oziroma izračun stacionarnih verjetnosti stanj časovno zvezne markovske verige je velikega pomena pri ovrednotenju performanc večprocesorskih sistemov. Zato bomo rezultate te analize koristno uporabili v nadaljnjem izvajanju.

Poglejmo si še, po kakšnem zakonu so porazdeljeni časi trajanja posameznih stanj markovske verige. Markovska lastnost pravi, da je obnašanje procesa v prihodnosti odvisno le od stanja v katerem se proces nahaja v trenutku opazovanja t, ne pa tudi od zgodovine procesa. Prihodnost procesa torej tudi ni odvisna od časa, ki ga je proces prebil v tem stanju pred časom t. Za naključno spremenljivko W_i, ki pomeni čas trajanja stanja i torej velja naslednja lastnost

$$P\{W_i>t+\tau \mid W_i>t\} = P\{W_i>\tau\}. \qquad (3.13)$$

Edini porazdelitveni zakon, ki zadošča temu pogoju, je eksponentna porazdelitev. Gostoto verjetnosti te porazdelitve podaja enačba

$$f(t) = a \exp(-at), \qquad t >= 0. \qquad (3.14)$$

Lahko bi pokazali, da so časi trajanja posameznih stanj ČZMV eksponentno porazdeljeni s funkcijo gostote verjetnosti

$$f_{Wi}(t) = -q_{ii} \exp(q_{ii}t), \qquad t>=0 \qquad (3.15)$$

in s srednjo vrednostjo

$$E(W_i) = -1/q_{ii}. \qquad (3.16)$$

Analiza večprocesorskih sistemov s pomočjo pridruženih markovskih verig pomeni torej predpostavko, da se čase trajanja aktivnih stanj in stanj dostopa obravnava kot eksponentno porazdeljene naključne spremenljivke.

4. Stohastične Petrijeve mreže

V nadaljevanju nas zanima vprašanje, kako na osnovi modela večprocesorskega sistema pridemo do pridružene markovske verige. Direktna pot je lahko zelo zapletena, saj moramo evidentirati vsa stanja sistema in časovne gostote prehodov med njimi. Pomagamo si s stohastičnimi Petrijevimi mrežami (SPM), ki predstavljajo učinkovito orodje za opisovanje sočasnosti in sinhronizacije v paralelnih sistemih. SPM so nekakšna nadgradnja standardnih Petrijevih mrež v tem, da vpeljejo vanje dimenzijo časa kot naključno veličino.

SPM je bipartiten graf, ki ga sestavlja množica položajev P, množica prehodov T, množica povezav A, množica prehodnih hitrosti G in začetna označitev M_0. SPM je torej določena s peterko (P, T, A, G, M_0). Vsak element iz množice G je prirejen enemu elementu iz množice T, kar pomeni, da je za vsak prehod podana njegova prehodna hitrost. Če je ta večja od nič, govorimo o časovnem prehodu, če pa je enaka nič, pa o takojšnjem prehodu. V grafični ponazoritvi rišemo položaje kot kroge, časovne prehode kot mastne črte, takojšnje prehode kot tanke črte, povezave pa kot puščice. Povezave vežejo položaje s prehodi in prehode s položaji. Položaj p_i je vhodni položaj prehoda t_j, če obstaja povezava od položaja p_i k prehodu t_j. Podobno rečemo, da je položaj p_i izhodni položaj prehoda t_j, kadar obstaja povezava od prehoda t_j k položaju p_i. Označitev SPM je določena s porazdelitvijo žetonov po posameznih položajih. Žetone grafično predstavimo s pikami. Nek prehod je omogočen, kadar vsebuje vsak izmed njegovih vhodnih položajev vsaj en žeton. Če je prehod takojšen,

se izvrši takoj, če pa je časoven, se izvrši po eksponentno porazdeljenem naključnem času. Prehodna hitrost, pripisana temu prehodu, predstavlja parameter te porazdelitve. Izvršitev prehoda povzroči odvzetje enega žetona iz vseh vhodnih položajev in dodajanje enega žetona v vsak izhodni položaj. To povzroči novo razporeditev žetonov po položajih in s tem nastanek nove označitve. Vse označitve neke SPM tvorijo njeno dosegljivostno množico, ki jo lahko predstavimo z drevesom dosegljivosti. Graditi ga pričnemo pri začetni označitvi, ki predstavlja koren drevesa in poiščemo vse neposredno dosegljive označitve, ki jih dobimo z izvršitvijo prehodov, omogočenih z začetno označitvijo. Te označitve postanejo neposredni potomci korena. Ta postopek ponavljamo nad temi in vsemi nadaljnimi potomci, dokler se vsaka pot v drevesu ne konča z označitvijo, ki smo jo že predhodno srečali, ali pa z mrtvo označitvijo, to je tako, ki ne omogoča nobenega prehoda. Označitve iz tako dobljene dosegljivostne množice so lahko stabilne ali pa nestabilne. Za stabilne velja, da omogočajo le časovne prehode, medtem, ko je pri nestabilnih označitvah omogočen vsaj en takojšen prehod. Zato je čas trajanja nestabilnih označitev enak nič.

Dokazano je, da so SPM izomorfne s časovno zveznimi markovskimi verigami. Stabilne označitve iz dosegljivostne množice SPM sovpadajo s stanji iz prostora stanj ČZMV, prehodne hitrosti med stabilnimi označitvami SPM pa s časovnimi gostotami prehodov med stanji ČZMV. Zato je potrebno iz drevesa dosegljivosti izločiti vse nestabilne označitve in na novo preračunati prehodne hitrosti med stabilnimi. Možno pa je tudi, da že pri graditvi drevesa dosegljivosti upoštevamo samo stabilne označitve. Drevo dosegljivosti SPM z le stabilnimi označitvami predtavlja torej že kar diagram prehajanja stanj iskane ČZMV.

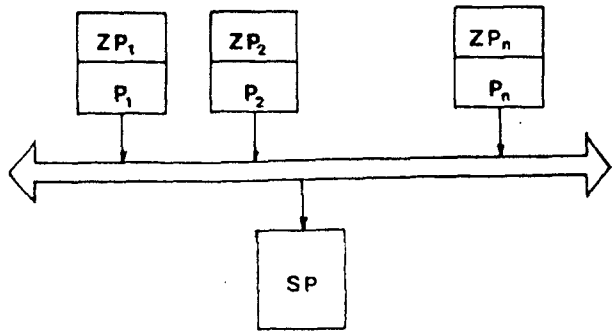
Postopek performančne analize nekega večprocesorskega sistema je z uporabo SPM precej olajšan. Od analitika zahteva samo definiranje topologije in parametrov SPM, nadaljni postopek gradnje drevesa dosegljivosti in stacionarne analize njemu izomorfne ČZMV pa je lahko popolnoma avtomatiziran. V naslednjem poglavju bomo pokazali celoten postopek performančne analize enostavne arhitekture večprocesorskega sistema.

5. Primer analize učinkovitosti večprocesorskega sistema

Za primer analizirajmo večprocesorski sistem s skupnim vodilom; model tega sistema prikazuje slika 2. Vsakemu procesorju je pridružen zasebni pomnilnik, skupni pomnilnik pa je zunanji vsem procesorjem, to pomeni, da je vsakemu izmed njih dostopen le preko skupnega vodila. Do konfliktnih situacij prihaja pri zaseganju skupnega vodila. Da bomo lahko dogajanje v tem sistemu obravnavali kot stohastični proces z markovsko lastnostjo, se mora režim delovanja modela podrejati naslednjim predpostavkam:

- Časi trajanja aktivnega stanja in stanja dostopa posameznih procesorjev so naključne spremenljivke z eksponentno porazdelitvijo in srednjimi vrednostmi
$1/\lambda_1, 1/\lambda_2 \ldots, 1/\lambda_p$ za aktivna stanja in
$1/\mu_1, 1/\mu_2, \ldots 1/\mu_p$ za stanja dostopa.
- Ko zahteva nek procesor dostop do skupnega pomnilnika, je ta dostop možen takoj (brez zakasnitve), če je skupno vodilo prosto.
- Če skupno vodilo ni prosto, preide procesor v fazo čakanja do sprostitve vodila.
- Ko procesor konča s stanjem dostopa, takoj (brez zakasnitve) sprosti skupno vodilo.

V analizi bomo predpostavljali popolno

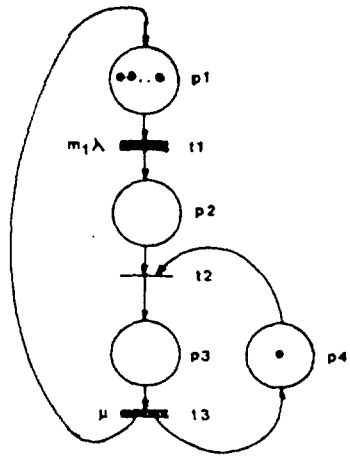


Slika 2. Arhitektura večprocesorskega sistema

simetrijo v sistemu, kar pomeni, da bosta parametra našega modela srednja vrednost dolžine aktivnega stanja $1/\lambda$ in srednja vrednost dolžine faze izmenjave $1/\mu$ enaka za vse procesorje. Razmerje med njima $\rho = \lambda/\mu$ pa imenujmo faktor obremenljivosti.

Točki 2. in 4. pomenita, da smo zanemarili čase, potrebne za dodeljevanje in sproščanje skupnega vodila. To dejanje lahko opravičimo z dejstvom, da so ti časi navadno veliko krajši od časov trajanja faz procesiranja in izmenjave. Možno pa je tudi, da čas dodeljevanja in sproščanja vodila prištejemo k času faze izmenjave.

Če torej sistem zadošča gornjim zahtevam, mu lahko priredimo stohastični proces z markovsko lastnostjo in opravimo performančno analizo na osnovi analize stacionarnega stanja prirejene markovske verige.



Slika 3. SPM večprocesorskega sistema

Analize se lotimo s konstruiranjem SPM, ki bo opisovala dogajanje v sistemu. Prikazuje jo slika 3. Število žetonov v položaju p_1 pomeni število procesorjev v aktivnem stanju, število žetonov v p_2 pomeni število procesorjev v stanju čakanja, žetoni v p_3 pomenijo procesorje v stanju dostopa, žeton v p_4 pa pomeni, da je skupno vodilo prosto. Prehod t_1 je časoven in predstavlja dogodek, ko želi eden izmed aktivnih procesorjev poseči v skupni pomnilnik. Ker je gostota teh dogodkov sorazmerna številu aktivnih procesorjev, je prehodna hitrost prehoda t_1 enaka $m_1\lambda$, kjer je m_1 število žetonov v p_1. Prehod t_2 je takojšen, saj dobi procesor dostop do skupnega pomnilnika takoj, če je le skupno vodilo prosto. Prehod t_3 je zopet časoven in ponazarja dogodek, ko nek procesor zaključi stanje dostopa in s tem sprosti skupno vodilo. Začetna označitev naj bo taka, da se v p_1 nahaja p žetonov (p je število procesorjev v sistemu), v p_4 en žeton, v p_2 in p_3 pa ni nobenega žetona. Ta začetna označitev ustreza stanju sistema, ko so vsi procesorji v

aktivnem stanju, skupno vodilo pa je prosto. Iz tako definirane stohastične Petrijeve mreže si zgradimo pripadajoče drevo dosegljivosti (slika 4). Pri graditvi drevesa smo takoj izpuščali nestabilne označitve in upoštevali le stabilne. Na primer. Iz označitve (p,0,0,1) pridemo v prvem koraku do označitve (p-1,1,0,1), ki je nestabilna, saj omogoča takojšen prehod t_2, zato takoj nastopi označitev (p-1,0,1,0), ki je stabilna in jo v drevesu upoštevamo.

$$
\begin{array}{lll}
P_0 = (p,0,0,1) & & \\
\quad\downarrow p\lambda & & \\
P_1 = (p-1,0,1,0) & \xrightarrow{\mu} & P_0 \\
\quad\downarrow (p-1)\lambda & & \\
P_2 = (p-2,1,1,0) & \xrightarrow{\mu} & P_1 \\
\quad\downarrow (p-2)\lambda & & \\
\quad\vdots & & \\
\quad\downarrow \lambda & & \\
P_p = (0,p-1,1,0) & \xrightarrow{\mu} & P_{p-1}
\end{array}
$$

Slika 4. Drevo dosegljivosti SPM

Drevo dosegljivosti SPM nam popolnoma določa ČZMV s katero bomo ponazorili dogajanje v našem večprocesorskem sistemu, saj predstavlja dosegljivostna množica prostor stanj ČZMV, prehodne hitrosti drevesa dosegljivosti pa so enake časovnim gostotam prehodov verige. Diagram prehajanja stanj tako dobljene markovske verige prikazuje slika 5.

Preden se lotimo reševanja markovske verige, se moramo odločiti za nek performančni pokazatelj, ki nam bo dal dovolj dobro informacijo o učinkovitosti večprocesorskega sistema. Za ta pokazatelj smo izbrali povprečno število aktivnih procesorjev (procesorjev v aktivnem stanju) in ga imenovali procesna moč sistema P. Izračunamo jo po enačbi

$$P = \Sigma_i (\pi_i n_a(i)), \quad i \in S, \tag{4.1}$$

kjer pomeni π_i stacionarno verjetnost stanja i ČZMV, $n_a(i)$ pa število aktivnih procesorjev v stanju sistema, ki je ponazorjen z i-tim stanjem ČZMV. Za izračun procesne moči rabimo torej stacionarne verjetnosti stanj verige, ki jih dobimo iz analize stacionarnega stanja ČZMV. To opravimo z rešitvijo sistema linearnih enačb, ki jih lahko zapišemo direktno iz diagrama prehajanja stanj po enačbah 3.11 in 3.12. Dobimo sistem p+1 enačb

$$-p\lambda\pi_0 + \mu\pi_1 = 0 \tag{4.2}$$

$$-((p-i)\lambda + \mu)\pi_i + (p-i+1)\lambda\pi_{i-1} + \mu\pi_{i+1} = 0, \quad i = 1 \ldots p-1$$

$$\pi_0 + \pi_1 + \ldots + \pi_p = 1.$$

Splošna rešitev tega sistema je naslednja:

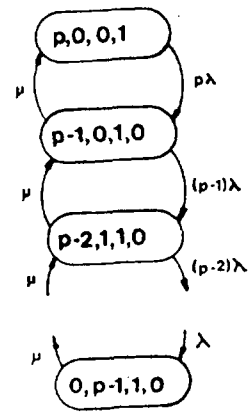
$$\pi_0 = (\Sigma_k (\rho^k (p!/(p-k)!)))^{-1}, \quad k = 0 \ldots p$$

$$\pi_i = \pi_0 \rho^i (p!/(p-i)!). \tag{4.3}$$

Procesna moč pa je enaka

$$P = (\Sigma_k (\rho^k n!/(n-k)!) - 1)/(\rho \Sigma_k (\rho^k n!/(n-k)!)), \quad k = 0 \ldots p \tag{4.4}$$

Rezultat analize učinkovitosti našega večprocesorskega sistema je torej enačba, ki podaja odvisnost procesne moči od števila procesorjev in obremenljivosti sistema. To pomeni, da lahko na podlagi rezultatov stohastične analize ocenimo učinkovitost sistema v različnem obsegu in pod različnimi delovnimi pogoji.



Slika 5. Diagram prehajanja stanj markovske verige

6. Zaključek

Rezultati predstavljene stohastične analize so dobljeni pod predpostavkami, navedenimi v poglavju 5. Te predpostavke so nam omogočile, da smo obravnavali dogajanje v večprocesorskem sistemu kot markovski proces. Čeprav se realni sistemi v splošnem ne podrejajo tem predpostavkam lahko uvidimo, da dajo markovski modeli navadno celo pesimistično oceno o učinkovitosti modeliranega sistema. Zakaj? V realnem sistemu ima porazdelitev časov dostopa navadno manjšo varianco v primeru z eksponentno porazdelitvijo. Tedaj je dejanska učinkovitost boljša, kot jo je napovedal naš model. Isto velja v primeru, ko časi aktivnih stanj in stanj dostopa niso za vse procesorje enaki. Iz tega lahko zaključimo, da dajejo rezultati, dobljeni na osnovi predstavljene stohastične analize, dovolj dobro oceno o učinkovitosti obravnavanega večprocesorskega sistema.

7. Literatura

1. Feller W., An Introduction to Probability Theory and Its Applications, Willey, New York, 1966.
2. Holliday M.A., Exact Performance Estimates for Multiprocessor Momory and Bus Interference, IEEE Transactions on Computers, Januar 1987.
3. Jamnik R., Verjetnostni račun, Mladinska knjiga, Ljubljana, 1971.
4. Karlin S., A First Course in Stochastic Processes, Academic Press, New York, 1975.
5. Marsan M.A. s sodelavci, Modeling Bus Contention and Memory Interference in a Multiprocessor System, IEEE Transactions on Computers, Januar 1983.
6. Marsan M.A. s sodelavci, Performance Models of Multiprocessor Systems, MIT Press, Cambridge, London, 1986.
7. Peterson J.L., Petri Net Theory and the Modeling of Systems, Prentice-Hall, Englewood Cliffs, 1981.
8. Vidav I., Višja matematika II, DZS, Ljubljana, 1975.
9. Zuberek W.M., Timed Petri Nets and Preliminary Performance Evaluation, Proc. of the 7th Annual Symposium on Computer Architecture, La Baule, 1980.

INERTNA STRUKTURA UNIX ZDRUŽLJIVEGA 8-BITNEGA OPERACIJSKEGA SISTEMA

INFORMATICA 1/88

Ivan Pepelnjak
Jernej Virant
Marjan Bradeško
Nikolaj Zimic
Fakulteta za elektrotehniko, Ljubljana

UDK 681.3.06

Izvleček :

Razvoj mikroračunalniške tehnologije zahteva uporabo vse zmogljivejših diskovnih enot in tiskalnikov. Takšne zmogljivosti lahko učinkovito izrabimo le v računalniški mreži. Za uspešno implementacijo mreže mikroračunalnikov rabimo operacijski sistem, ki podpira poddirektorije, zaščito datotek ipd. V članku je predstavljen operacijski sistem, ki izpolnjuje vse navedene zahteve, poleg tega je UNIX in CP/M združljiv. Opisana je struktura datotek na disku in interna zgradba operacijskega sistema.

Abstract :

The development of computer technology requires use of more and more sophisticated disk units and printers. This resources can be effectively used only in a local area network. To successfully implement microcomputer local area network, an operating system supporting subdirectories, file protection and users separation is needed. In the paper we present an operating system which meets all of the stated requirements along with UNIX and CP/M compatibility. Disk structure and internal data structures used are presented.

1. UVOD

1.1 Opis problema

Vse hitrejši razvoj mikroračunalniške tehnologije omogoča prodor mikroračunalnikov v vsa področja družbenega življenja. Kakor hitro pa hočemo mikroračunalnik uporabiti v poslovni sferi, se srečamo s količinskimi problemi :

- prisotnost velikih baz podatkov
- velika količina izpisov

ki jih lahko rešimo samo z zahtevno in še vedno drago tehnologijo (trdi diski, hitri tiskalniki). Poleg tega morajo biti podatkovne baze v takšnih okoljih praviloma deljene med več uporabnikov, ki hkrati dosegajo podatke iz podatkovne baze.

Če hočemo rešiti takšen problem je najbolje, da povežemo več mikroračunalnikov v mrežo, znotraj katere si lahko delijo zmogljivosti (trdi disk) in izmenjujejo sporočila. Tako dosežemo večjo hitrost delovanja celotnega sistema (inteligenca je razdeljena med več delovnih mest) in manjšo ceno sistema na enoto zmogljivosti.

1.2 Naša realizacija cenene mikroračunalniške mreže

Mikroračunalniško mrežo lahko s stališča programske opreme realiziramo na več načinov :

- vsi računalniki lahko zahtevajo podatke z vseh računalnikov
- nekateri računalniki so delovne postaje, drugi so strežniki

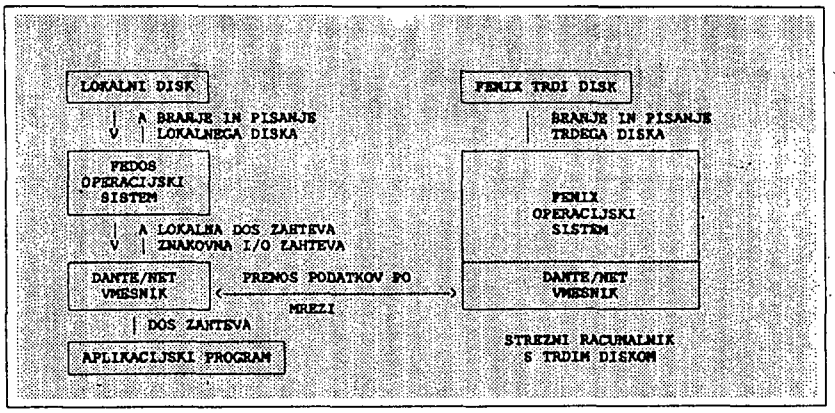
Prvi pristop je zanimiv v primeru, ko imamo na razpolago že obstoječ (dokaj drag) mikroračunalnik s trdim diskom. Ta implementacija ima nekatere slabe lastnosti, saj je operacijski sistem takšnega računalnika zahtevnejši, poleg tega zaradi servisiranja zahtev z drugih vozlišč upade hitrost lokalnega dela.

Drugi pristop je boljši v primeru, ko lahko sami razvijemo svoj mikroračunalnik, saj se lahko odločimo za posebno izpeljanko mikroračunalnika, ki vsebuje samo krmilnik za trdi disk in enostaven zaslonski krmilnik za diagnostiko ter tako dosežemo cenejšo izvedbo mikroračunalnika. Delitev vozlišč na strežna vozlišča in delovne postaje poleg tega prinese še dodatne prednosti, saj se vsako vozlišče ukvarja izključno z eno nalogo, ki jo zaradi tega opravlja bolje in hitreje.

Kot osnovo naše mikroračunalniške mreže smo vzeli mikroračunalnik DIALOG z operacijskim sistemom FEDOS (CP/M združljiv operacijski sistem) in mikroprocesorjem Z80.

Ko smo preudarjali, kakšen naj bi bil operacijski sistem strežnega računalnika smo ugotovili naslednje :

- operacijski sistem mora podpirati zaščito datotek med uporabniki
- operacijski sistem mora podpirati drevesno strukturo kazala datotek



Slika 1.1 Struktura mikroračunalniške mreže

V naslednjih razdelkih si bomo zaporedoma ogledali
- Strukturo diska, kot jo podpira FENIX operacijski sistem
- notranjo zgradbo FENIX operacijskega sistema

Če smo hoteli v okviru mreže podpirati standardno CP/M programsko opremo, smo morali vsaj logično obdržati CP/M diskovno strukturo in CP/M diskovne funkcije. Tako smo prišli do naslednje strukture mikroračunalniške mreže :

- v strežnem računalniku teče operacijski sistem FENIX, ki podpira večino funkcij, ki jih mora imeti večuporabniški operacijski sistem (zaščita datotek, drevesna struktura kazala, podpora večih uporabnikov ipd).
- v delovnih postajah teče lokalno operacijski sistem FEDOS (CP/M združljiv) in vmesnik na lokalno mrežo DANTE/NET, ki pošilja zahteve preko lokalne mreže strežnemu računalniku

2. STRUKTURA SISTEMA DATOTEK

2.1 Uvod

Operacijski sistem, ki podpira hkratno ali zaporedno delo večih uporabnikov, mora zadovoljevati sledeče zahteve :

- omogočati mora ločitev datotek različnih uporabnikov, da omogoči vsakemu uporabniku pregled nad njegovimi datotekami in prepreči nehoteno izgubo podatkov zaradi napake v delu enega od uporabnikov

- omogočati mora zaščito datotek pred drugimi uporabniki, da bi preprečili zlorabo podatkov ali namerno povzročeno izgubo podatkov.

Poleg teh zahtev mora operacijski sistem izpolnjevati še naslednje zahteve, ki jih narekuje optimalnost delovanja računalniškega sistema :

- omogočati mora obstoj zelo majhnih datotek, saj je v razvojnem okolju večina datotek majhna (100 bytov - 10K bytov). Osnovna enota dodeljevanja prostora na disku mora biti torej čim manjša, najbolje en fizični sektor.

- omogočati mora obstoj zelo velikih datotek (omejitev mora biti velikost diskovnega pomnilnika), saj podatkovne baze navadno potrebujejo izredno velike datoteke.

- v razvojnem okolju je ugodno, če lahko uporabnik svoje datoteke razdeli v logične skupine.

- omogočati mora hiter naključen dostop do datotek.

Praktično vsi operacijski sistemi imajo dostop do datotek organiziran preko kazala datotek. Le to vsebuje najmanj dva podatka :

- ime datoteke
- informacijo o tem, kje se datoteka nahaja na disku

Informacija o položaju datoteke na disku je lahko spravljena na tri različne načine :

- v kazalu datotek
- v posebnem delu diska
- deli datoteke so med seboj povezani v verigo

Če izberemo tretjo možnost imamo še vedno dve različici :

- informacija o nadaljevanju verige je skrita v vsakem sektorju.
- informacija o nadaljevanju verige je zapisana v posebni tabeli.

Vse tri rešitve imajo svoje dobre in slabe lastnosti :

2.1.1 Zapis položaja datoteke v kazalu datotek

Dobre lastnosti :

- položaj datoteke ugotovimo takoj, ko najdemo ustrezen zapis v kazalu datotek. Tako prihranimo en dostop do diska
- takšen način zapisa je najlažje realizirati

Slabe lastnosti :

- Velikost zapisa v kazalu datotek je velika, zato potrebujemo za iskanje zapisa v kazalu več dostopov do diska.
- Pojavi se problem velikih datotek, ki jih navadno rešujemo z uporabo več zapisov v kazalu datotek, kar še povečuje že tako veliko kazalo

2.1.2 Zapis položaja datoteke v posebnem delu diska

Dobre lastnosti :

- kazalo datotek je majhno, kar omogoča hitro iskanje

Slabe lastnosti :

- dostop do informacije o položaju datoteke zahteva še en dostop do diska in praviloma tudi premik glave.

2.1.3 Povezava v verigo

Dobre lastnosti :

- kazalo datotek je majhno
- v primeru uporabe posebne tabele so vse informacije o položaju datotek zbrane na enem mestu

Slabe lastnosti :

- shranjevanje kazalcev v podatkovnih blokih izredno upočasni naključni dostop do datoteke in ga v bistvu degenirira v sekvenčnega
- uporaba posebne tabele navadno povzroči probleme zaradi njene velikosti (za 40M bytni disk pri velikosti enote dodeljevanja 1K byt ta tabela velika 80K bytov).

2.2 Pregled obstoječih rešitev

Oglejmo si sedaj nekatere obstoječe operacijske sisteme, posebno tiste, ki so primerljivi z našim sistemom.

2.2.1 CP/M

Operacijski sistem CP/M ima vse informacije o datotekah zbrane v skupnem kazalu datotek. Informacija o položaju datoteke je zapisana kar v kazalu datotek, za velike datoteke uporablja CP/M več zapisov v kazalu datotek.

Osnovna enota dodeljevanja je odvisna od velikosti diska in se giblje od 1K byt do 8K bytov. Velikost kazala datotek je omejena, torej je omejeno tudi skupno število datotek na celotnem disku.

Zaščita datotek je omejena na Read-Only atribut, ki ga lahko nastavlja vsak uporabnik. Delno je torej preprečeno nenamerno brisanje datotek, namerne poškodbe podatkov ne moremo preprečiti.

Operacijski sistem podpira 16 uporabnikov, vendar ne nudi nobene zaščite med uporabniškimi področji.

2.2.2 MS-DOS

Operacijski sistem MS-DOS podpira drevesno strukturo kazala, vendar je velikost glavnega kazala fiksna. Ne podpira uporabnikov, prav tako ne podpira zaščite datotek.

Osnovna enota dodeljevanja je odvisna od velikosti diska in se giblje od 1K do 4K. Skupno število datotek na disku ni omejeno.

Informacija o položaju datoteke na disku se nahaja v posebni tabeli na začetku diska (FAT), posamezni elementi v tej tabeli so povezani s kazalci. Vsak element predstavlja en cluster (enoto dodeljevanja prostora).

2.2.3 FILES-11 struktura (RSX-11 in VMS)

FILES-11 struktura je sorazmerno precej zahtevna diskovna struktura. Omogoča drevesno strukturo datotek, zaščito datotek med uporabniki, omogoča neomejeno število uporabnikov. Vsi parametri diskovne strukture (število datotek, velikost datoteke ipd.) so neomejeni, omejitev je le fizična velikost diskovne enote.

Informacija o položaju datotek na disku je zapisana v posebnih sektorjih (FILE HEADER), ki so zbrani v datoteki (0,0)INDEXF.SYS, kar poenostavi kontrolo pravilnosti diskovne strukture in omogoča iskanje izgubljenih datotek. Operacijski sistem edini med naštetimi podpira verzije datotek.

2.3 Naša izbira strukture datotek

Ob izbiri strukture datotek smo se skušali izogniti večini slabih strani vseh sistemov. Tako smo dobili strukturo, ki dokaj optimalno izpolnjuje večino ciljev, ki smo si jih zadali v začetku tega poglavja, vendar je omejena z uporabljeno tehnologijo (Z80 procesor in 64K bytov pomnilnika), ki nam je onemogočala realizacijo bolj zahtevne in bolj optimalne strukture.

Velikost sektorja na disku je 1024 bytov, velikost logičnega sektorja, kot ga vidi uporabniški program pa je 128 bytov, da ohranimo združljivost s CP/M operacijskim sistemom. Ves dostop uporabniškega programa do datotek poteka preko logičnih sektorjev velikosti 128 bytov.

Informacija o imenu datotek je zbrana v kazalu datotek. Kazalo datotek je datoteka tako kot vsaka druga datoteka, zato je lahko kazalo neomejeno veliko. Operacijski sistem podpira drevesno strukturo kazala, vsako vozlišče v drevesni strukturi je spet datoteka. Tako smo dosegli enoten pristop do vseh datotek v operacijskem sistemu, kar poenostavi pisanje sistemskih programov a tudi samega operacijskega sistema. Poleg tega obstajata v vsakem kazalu posebni datoteki . in .. (trenutno in predhodno kazalo), kar še olajša sistematski pristop do drevesne strukture kazala. Za definicijo poti do posameznega vozlišča je uporabljena UNIX sintaksa (npr. /USR/SAMPLE/TEST).
Kot vidimo, je v kazalu datotek zapisano samo ime datoteke, njena zaščita in položaj ostalih informacij o datoteki. S tem dosežemo, da je velikost enega zapisa samo 16 bytov, torej spravimo v en fizičen sektor (1K byt) 64 zapi-

```
Type Directory_Record = Record
        File_Name        : String (8)
        File_Type        : String (3)
        First_Descriptor : Integer
        Protection       : Array (1..4) of nibble
        Reserved_Byte    : Byte
```

Slika 2.1 Struktura zapisa v kazalu datotek

sov, kar je dovolj za večino primerov, torej lahko celotno kazalo večinoma dosežemo z enim samim dostopom do diska, kar bistveno vpliva na hitrost odpiranja datotek.

Zaščita datoteke je razdeljena v štiri dele :

- uporabnik (USER_ID datoteke je enak USER_ID uporabnika)
- uporabniška skupina (GROUP_ID datoteke je enak GROUP_ID uporabnika)
- ostali svet (USER_ID in GROUP_ID datoteke in uporabnika se razlikujeta)
- privilegirani sistemski uporabnik (USER_ID = 0). Za takšnega uporabnika sistem ne testira zaščite datotek.

Vsaka skupina uporabnikov (OWNER, GROUP, WORLD) ima v dveh zlogih zaščite prirejene 4 bite, ki predstavljajo dovoljenja posamezne skupine uporabnikov :

- READ -- uporabnik lahko datoteko bere
- WRITE -- uporabnik lahko datoteko piše
- EXECUTE -- uporabnik lahko datoteko izvaja
- DELETE -- uporabnik lahko datoteko briše

Za kazala, ki so po definiciji tudi datoteke, je pomen posameznih bitov nekoliko spremenjen :

- READ -- uporabnik lahko bere kazalo
- WRITE -- uporabnik lahko kreira ali briše datoteke
- EXECUTE -- uporabnik lahko odpira datoteke
- DELETE -- uporabnik lahko briše kazalo

Četrta skupina štirih bitov je neuporabljena za zaščito datotek in služi kot opis dodatnih lastnosti datoteke :

- DIRECTORY -- datoteka je kazalo
- SYSTEM -- datoteka je sistemska datoteka

Z implementacijo zaščite nad kazali datotek lahko uvedemo nekatere zanimive izvedbe zaščite datotek :

- uporabnik lahko samo pregleduje kazalo, vendar ne more odpreti datoteke v kazalu (READ)
- uporabnik lahko odpre datoteko, če pozna njeno ime, vendar ne more izpisati imen datotek (EXECUTE)
- uporabnik lahko samo lista kazalo in odpira datoteke (READ in EXECUTE)
- uporabnik lahko samo kreira nove datoteke, ko je datoteko enkrat kreiral, je ne more več brati (WRITE). Preprečevanje brisanja datotek dosežemo z ustrezno zaščito datotek pri kreiranju.

Informacija o položaju datoteke na disku je zbrana v dveh podatkovnih strukturah :

- Za prvih 256K bytov datoteke v opisu datoteke (FILE DESCRIPTOR) -- en sektor. V tem sektorju so zbrane tudi vse ostale informacije o datoteki.
- Za podatke preko 256K bytov datoteke v razširjenem opisu datoteke (EXTENDED FILE DESCRIPTOR) -- en sektor za vsakih 512K bytov

```
Type File_Descriptor = Record
       Open_Count         : Byte
       User_ID            : Byte
       Group_ID           : Byte
       Last_Block         : Double Integer
       Access_Date        : Date
       Update_Date        : Date
       Create_Date        : Date
       Backup_Date        : Date
       Descriptor_Pointer : Array (0..127) of Integer
       Data_Pointer       : Array (0..255) of Integer
     End ;
```

Slika 2.2 Struktura opisa datoteke

Polja USER_ID in GROUP_ID imajo znan pomen. Polje OPEN_COUNT služi za štetje števila dostopov do datotek, ki jih hkrati odpre več uporabnikov. Polje LAST_BLOCK pove številko zadnjega logičnega bloka (128 bytov) v datoteki. V opisu datoteke hrani operacijski sistem datum zadnjega dostopa, spreminjanja, kreiranja in arhiviranja. Te datume lahko uporabljamo za kontrolo dostopa do datoteke ali za delno arhiviranje podatkov na disku (arhiviramo samo podatke kjer je UPDATE_DATE > BACKUP_DATE). Datum je natančen do sekund.

DESCRIPTOR_POINTER je tabela kazalcev na sektorje, ki vsebujejo razširjen opis datoteke. DATA_POINTER je tabela kazalcev na prvih 256 podatkovnih sektorjev. S takšno zasnovo opisa položaja datoteke na disku smo dosegli naslednje omejitve :

- velikost diska največ 64M bytov
- velikost datoteke največ 128M bytov, torej več od velikosti diska, kar pomeni, da je velikost datoteke dejansko omejena z velikostjo diska.

Ker ima velika večina danes dostopnih diskov, primernih za vgraditev v mikroračunalnik, kapaciteto manjšo od 64M bytov, ta omejitev velikosti diska ne bi smela predstavljati resne ovire. Če želimo uporabljati večji disk, ga lahko razdelimo na particije velikosti 64M bytov in ga uporabljamo kot več logičnih diskov velikosti 64M bytov.

Zasedenost diska je zapisana v posebni datoteki (/BITMAP.SYS). Z uporabo posebne datoteke dosežemo bistveno manjši zagonski čas sistema v primerjavi s sistemi, ki ob zagonu računalnika gradijo tabelo zasedenosti diska v spominu (CP/M). Poleg tega je gradnja takšne tabele v pogojih drevesne strukture kazala praktično nemogoča, saj bi zasedla preveč spomina. Edina slabost takšne zasnove bitne tabele se pokaže ob izpadu sistema, saj je takrat tabela poškodovana in potrebujemo poseben program, ki jo popravi nazaj v korektno stanje.

Za vzpostavitev vseh sistemskih struktur potrebujemo še dva kazalca :

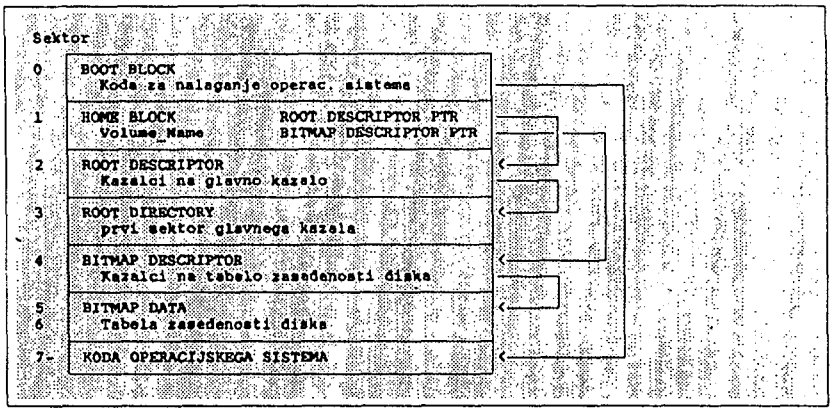
- kazalec na opis datoteke / (vrh drevesne strukture kazala)
- kazalec na opis datoteke /BITMAP.SYS (za lažje delo, čeprav bi ga lahko dobili iz datoteke /).

Ta dva kazalca sta zapisana v HOME BLOCK sektorju diska (sektor 1 v 0. stezi), ki ima sledečo strukturo

```
Type Home_Block = Record
       Volume_Name       : String (16)
       Root_Descriptor   : Integer
       Bitmap_Descriptor : Integer
       Bitmap_Size       : Integer
     End ;
```

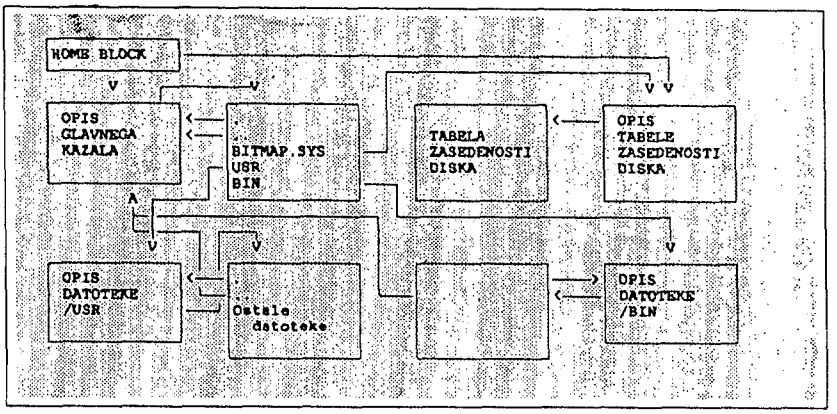
Slika 2.3 Struktura HOME BLOCK sektorja

Sektor 0 v 0. stezi je rezerviran za program, ki naloži operacijski sistem ob zagonu računalnika. Tako je standardna oblika prve steze diska sledeča :

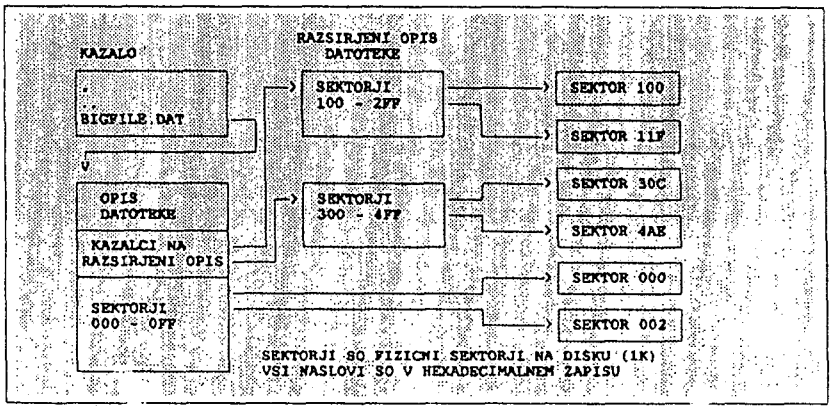


Slika 2.4 Struktura prve steze diska

Za ilustracijo zgoraj navedenih pojmov si oglejmo še primer strukture diska, ki ima v glavnem kazalu samo dva poddirektorija /BIN in /USR. Prikazane so vse povezave med podatkovnimi strukturami na disku in način kako operacijski sistem doseže določene podatkovne strukture.



Slika 2.5 Primer diskovne strukture



Slika 2.6 Primer zelo velike datoteke

Lastnost	CP/M	MS-DOS	FILES-11	FENIX
velikost kazala	omejena	omejena za glavno kazalo	64K datotek za RSX-11	neomejena
velikost zapisa v kazalu	32 bytov	32 bytov	16 bytov	16 bytov
drevesna struktura kazala	NE	DA	NE za RSX-11 DA za VMS	DA
zapis polozaja datoteke na disku	v kazalu	veriga (FAT)	v FILE HEADER	v opisu datoteke
maksimalna velikost datoteke	32M bytes	velikost diska	velikost diska	64M bytes
globina kazala	---	neomejeno	2 za RSX-11	neomejeno
stevilo uporabnikov	16	---	64K za RSX-11 neomejeno VMS	255
zascita datotek	NE	NE	DA	DA
casovne oznake	DA (2)	DA (2)	DA (4)	DA (4)
osnovna enota dodeljevanja prostora	2-4K	1-4K	0.5K	1K

Opomba : Oznaka neomejeno pomeni, da ima parameter tako veliko vrednost, da je za naso primerjavo neomejeno velika.

Slika 2.7 Primerjava diskovne strukture sistema FENIX z drugimi operacijskimi sistemi

3. Struktura operacijskega sistema FENIX

3.1 Uvod

Operacijski sistem FENIX je samostojen program, ki teče v glavnem računalniku mreže DANTE/NET. Njegova edina naloga je odgovarjati na zahteve drugih računalnikov v mreži. Glavna zanka operacijskega sistema je zato zasnovana tako

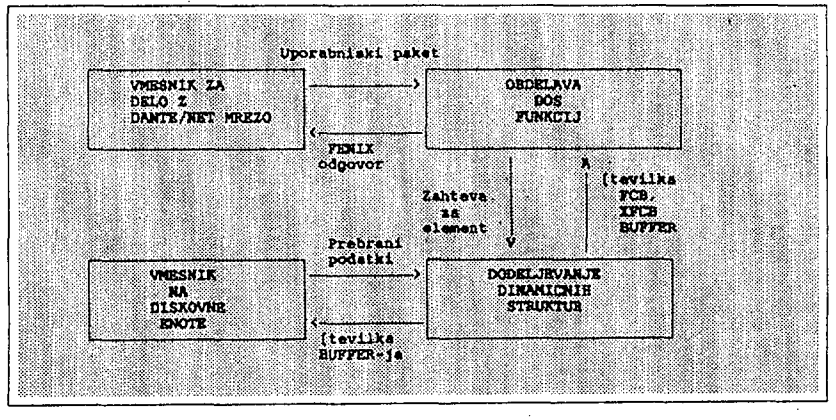
```
Initialize System
Repeat
      Get network message
      Process message
      Send network message
Until Shutdown_Request
Shutdown system
Halt CPU
```

Slika 3.1 Glavna zanka operacijskega sistema

Če bi hoteli doseči večjo efektivnost dela, bi morali implementirati paralelnost procesov PROCESS_MESSAGE, kar bi pomenilo, da operacijski sistem hkrati streže več dostopom do diska.

Operacijski sistem lahko razdelimo v več modulov :

- komunikacija z DANTE/NET mrežo
- izvajanje DOS funkcij
- dodeljevanje in vzdrževanje vmesnikov
- krmilnik diskovnih enot



Slika 3.2 Razdelitev operacijskega sistema *FENIX*

3.2 Opis posameznih modulov

3.2.1 Komunikacija z DANTE/NET mrežo

Naloga tega modula je sprejem in oddaja paketa v DANTE/NET mrežo. Mreža sama je collision-avoidance mreža tipa Ethernet, uporabljen protokol je SDLC. Paket sam je sestavljen iz več delov :

- krmilne informacije (sprejemnik, oddajnik)
- opis DOS funkcije
- parametri za DOS funkcijo
- kontrolni seštevek

Ko modul za komunikacijo z DANTE/NET mrežo sprejme paket, zasede v pomnilniku 256 bytov velik blok, ki ga uporablja za shranjevanje paketa z mreže in za shranjevanje vseh sistemskih spremenljivk, ki jih potrebuje DOS za obdelavo te DOS funkcije. Ko je paket uspešno sprejet, ga modul preda modulu za obdelov DOS funkcij. Po izvršeni DOS funkciji modul prejme izhodni paket, ki ga pošlje računalniku, ki je zahteval DOS funkcijo.
Parametri za DOS funkcijo so lahko :

- FCB ID -- številka FCB - ja, ki opisuje datoteko, s katero hoče uporabnik delati. FCB ID je sestavljen iz dveh bytov : številka FCB - ja in zaporedna številka dostopa do tega FCB-ja, s pomočjo katere se kontrolira pravilnost dostopa in preprečuje branje datotek drugih uporabnikov
- Številka zapisa v datoteki, ki ga hočemo brati ali pisati
- Celoten FCB, kot ga podpira CP/M (16 bytov)
- Zastavice v CP/M FCB-ju
- DMA področje (vsebina zapisa, ki ga beremo ali pišemo)

Poleg teh parametrov lahko vrne operacijski sistem FENIX uporabniku še :

- status DOS funkcije

3.2.2 Izvajanje DOS funkcij

Modul skrbi za izvajanje vseh DOS funkcij, ki jih drugi računalniki zahtevajo po mreži.
V inicializaciji podatkovnih struktur modul prepiše podatke o uporabniku v začasne spremenljivke ter določi logični disk, s katerim uporabnik dela in njemu pripadajoči fizični disk in direktorij.

```
Get network packet
Initialize data structures
If error then Send reject packet
        else Process DOS function
             Send result packet
```

Slika 3.3 Osnovna struktura modula za izvajanje DOS funkcij

Po končani inicializaciji modul preko tabele kliče ustrezno DOS funkcijo, ki obdela uporabnikovo zahtevo in zgradi paket, ki ga modul preko komunikacijskega modula vrne uporabniku.

3.2.3 Dodeljevanje in vzdrževanje vmesnikov

Modul skrbi za dodeljevanje in vzdrževanje vseh dinamičnih sistemskih podatkovnih struktur :

- vmesniki za delo z diskom
- opis odprte datoteke (FCB)
- opis večkratno odprte datoteke (XFCB)

Natančen algoritem dodeljevanja teh struktur je opisan v razdelku SISTEMSKE PODATKOVNE STRUKTURE.

3.2.4 Krmilnik trdega diska

Modul skrbi za povezavo ostalega operacijskega sistema z diskovnimi enotami. Modul kliče modul za dodeljevanje in vzdrževanje vmesnikov v sledečih primerih :

- ko zahteva branje novega sektorja z diska
- ko zahteva zapis sektorja na disk v primeru, ko zmanjka vmesnikov
- ko zahteva zapis sektorja na disk ob periodičnem čiščenju sistemskih vmesnikov

3.3 Sistemske podatkovne strukture

Operacijski sistem FENIX potrebuje za svoje delovanje naslednje podatkovne strukture :

- vmesnike za delo z diskom
- opis odprtih datotek (FCB)
- opis večkratno odprtih datotek (XFCB)
- opis uporabnikov (USER)
- opis diskovnih enot

Vse dinamične podatkovne strukture (vmesniki za delo z diskom, FCB in XFCB) se dinamično dodeljujejo in odvzemajo ob prezasičenosti sistema. Operacijski sistem vodi za te tri dinamične strukture statistiko njihove uporabe (LRU algoritem), tako, da lahko po potrebi odvzame najmanj uporabljan element.
LRU algoritem je realiziran s pomočjo vrste :

- ob kreiranju se element postavi na začetek vrste
- ob vsakem dostopu operacijski sistem premakne element na začetek vrste
- LRU element je element na koncu vrste

Opisan algoritem je enostaven za realizacijo in daje dovolj dobre rezultate.

3.3.1 Opis odprtih datotek

```
Type File_Control_Block = Record
     Disc_Drive          : Byte
     File_Name           : String (8)
     File_Type           : String (3)
     Sequence_Number     : Integer
     First_Descriptor    : Integer
     First_Buffer        : Byte
     Current_Descriptor  : Integer
     Current_Buffer      : Byte
     FCB_Semaphore       : Byte
     Next_Pointer        : Integer
     Prev_Pointer        : Integer
     Protection          : Byte
     Node_ID             : Byte
     FCB_Flags           : Byte
     Access_Count        : Byte
End ;
```

Slika 3.4 Opis odprte datoteke

Opis posameznih polj :

Disc_Drive — diskovna enota, na kateri je odprta datoteka

File_Name — ime datoteke

File_Type — tip datoteke

Sequence_Number — zaporedna številka uporabe tega elementa. Ko operacijski sistem odvzame FCB uporabniku, ki ga že dolgo ni uporabljal, poveča sequence number in tako onemogoči ponovno uporabljanje odvzetega FCB-ja

First_Descriptor
Številka sektorja na disku, ki vsebuje opis datoteke

First_Buffer
Številka vmesnika, ki vsebuje opis datoteke ali 0, če opisa datoteke ni v spominu

Current_Descriptor
Številka sektorja na disku, ki vsebuje razširjeni opis datoteke za zadnji prebrani sektor. To polje se uporablja le če beremo datoteke, ki so večje od 256K bytov, drugače je enako First_Descriptor

Current_Buffer
Številka vmesnika, ki vsebuje sektor, na katerega kaže Current_Descriptor. Če tega sektorja ni v spominu, vsebuje to polje 0.

FCB_Semaphore neuporabljeno

Next_Pointer
Prev_Pointer Kazalci za implementacijo LRU algoritma

Protection bit 0 -- 3
zaščita datoteke, ki je odvisna od zaščite datoteke v opisu datoteke in uporabnika in grupe računalnika, ki je odprl datoteko.
bit 4 -- 7
dodatni atributi datoteke

Node_ID Številka računalnika, ki je odprl datoteko

FCB_Flags Zastavice, ki povedo stanje FCB-ja
bit 7 FCB je zaklenjen v spominu
bit 6 polje First_Descriptor je OK
bit 5 FCB je neveljaven
bit 4 FCB je uporabljen

Access_Count Števec dostopov do FCB-ja. Števec se poveča ob vsakem OPEN ukazu in zmanjša ob vsakem CLOSE ukazu. Ko doseže vrednost števca 0, operacijski sistem izvede dokončno CLOSE datoteke.

3.3.1.1 Dodeljevanje FCB-jev

Operacijski sistem zahteva nov FCB ob DOS funkciji, ki mora za svoje delo odpreti novo datoteko (OPEN, MAKE, DELETE, RENAME, GET FILE SIZE ipd.). Če je na razpolago prost FCB, ga modul za dodeljevanje vmesnikov zaseže, vpiše v FCB_FLAGS bit 7 in 4 ter ga vrne DOS modulu.

Če operacijski sistem ne more najti prostega FCB-ja, poišče zadnji FCB v listi, ki ni zaklenjen v spomin, zapre datoteko, ki ji ta FCB pripada in ga vrne DOS modulu.

Da bi preprečili dostop do tujih podatkov in zlorabo operacijskega sistema, poveča operacijski sistem ob vsakem OPEN klicu polje SEQUENCE_NUMBER v FCB-ju. Če hoče uporabniški program brati ali pisati datoteko, mora poleg številke FCB-ja v paketu, ki ga pošlje preko mreže podati še zaporedno številko v FCB-ju. Če se ti dve številki ne ujemata, je bila datoteka medtem zaprta in uporabniški program mora ponovno izvršiti OPEN klic. Ta protokol je implementiran že v mrežnem delu CP/M operacijskega sistema na ostalih računalnikih v mreži in je tako za končnega uporabnika neviden.

Ker bi lahko uporabnik uganil pravilno zaporedno številko v FCB-ju primerja operacijski sistem poleg zaporedne številke še številko vozlišča s katerega je bila datoteka odprta in številko vozlišča s katerega je prišla zahteva za branje oz. pisanje

Operacijski sistem vzdržuje v vsakem FCB-ju polje Access_Count ki šteje število OPEN klicev izvršenih na tem FCB-ju. Ko je polje Access_Count pri CLOSE klicu enako 0, lahko operacijski sistem izvrši dokončen CLOSE datoteke. Operacijski sistem tedaj sprosti FCB, ki je pripravljen za naslednji OPEN klic.

Če več uporabnikov hkrati odpre isto datoteko, vrne operacijski sistem prvemu uporabniku številko FCB-ja, s katerim je odprl to datoteko, za vse naslednje uporabnik pa uporabi operacijski sistem novo podatkovno strukturo XFCB. Takšen algoritem omogoča vodenje evidence o odpiranju datotek in preprečuje izgubo podatkov zaradi brisanja in hkratnega branja oz. pisanja datoteke (če bi vodili le evidenco o prvem vozlišču, ki je odprlo datoteko, bi lahko zbrisali datoteko, ki jo dosega več uporabnikov ali pa bi preprečili brisanje datoteke, ki jo je en uporabnik večkrat odprl). S stališča uporabniškega programa se XFCB ne razlikuje od FCB-ja, razlika je le v tem, da ima XFCB zaporedno številko od 128 do 255.

Opomba : Dodeljevanje FCB-jev je še oteženo zaradi slabega sloga programiranja v večini CP/M programov, ki praviloma ne zapirajo datotek, ki jih ne rabijo več, tako, da mora operacijski sistem hevristično določati katere datoteke program še uporablja in katere lahko zapre. Napake v tem hevrističnem procesu privedejo do ponovne uporabe zaprte datoteke in ponovnega OPEN klica, ki je, kot smo že poudarili, neviden za uporabniški program.

3.3.2 XFCB -- opis večkrat odprte datoteke

Slika 3.5 Struktura XFCB - ja

Operacijski sistem kreira enega ali več XFCB za vsako datoteko, ki jo hkrati odpre več kot en uporabnik. Prvi uporabnik uporablja za dostop do datoteke številko FCB-ja, vsi naslednji uporabniki uporabljajo za dostop do datoteke številko XFCB-ja, ki kaže na pravilni FCB. S takšnim pristopom je omogočeno vodenje natančne evidence o odpiranju datotek, poleg tega je število uporabnikov, ki lahko hkrati odprejo eno datoteko neomejeno.

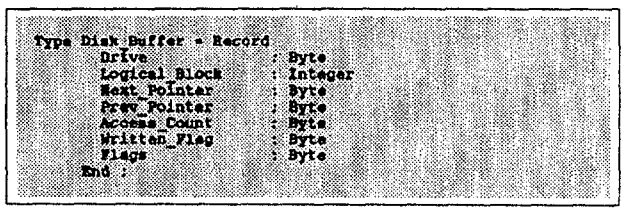
3.3.2.1 Dodeljevanje XFCB :

Operacijski sistem dodeli nov XFCB vsakič ko uporabnik poskuša odpreti datoteko, ki jo je že odprl drug uporabnik. Če modul za dodeljevanje vmesnikov ugotovi, da ne more dobiti prostega XFCB-ja, odvzame enega od uporabljenih XFCB-jev, zapre FCB, ki ga je ta XFCB naslavljal (in tako zmanjša ACCESS_COUNT) in vrne nov XFCB modulu za obdelavo DOS funkcij.

Ob CLOSE ukazu operacijski sistem sprosti XFCB in izvede CLOSE ukaz na ustreznem FCB-ju.

Obdelava odvzetih XFCB-jev, ki jih uporabnik hoče ponovno uporabiti je enaka kot za odvzete FCB-je.

3.3.3 Vmesnik za delo z diskom



Slika 3.6 Struktura vmesnika za delo z diskom

Polji Drive in Logical_Block opredelita diskovno enoto in zaporedno številko sektorja na tej enoti. Polji Next_Pointer in Prev_Pointer služita za implementacijo LRU algoritma. Polje Access_Count služi za štetje števila dostopov do tega vmesnika.

Polje Written_Flag služi za indikacijo vpisovanja v ta vmesnik in vsebuje 1 bit za vsak logični blok (128 bytov) znotraj fizičnega sektorja (1 K byte).

V polju FLAGS operacijski sistem hrani stanje tega vmesnika :

Bit 7	Vmesnik je zaklenjen v spomin
Bit 6	Vsebina vmesnika je spremenjena
Bit 5	Modul za povezavo z diski piše vmesnik na disk
Bit 4	Vmesnik bo zamenjan
Bit 3	Modul za povezavo z diski bere vmesnik z diska

Bita 5 in 3 služita za sinhronizacijo med modulom za delo z diskovnimi enotami in ostalimi moduli operacijskega sistema, saj bi se ob uporabi prekinitev in zakasnjenega pisanja na disk zlahka zgodilo, da bi modul za obdelavo DOS funkcij uporabljal vmesnik, ki še ni bil prebran z diska ali ki se ravnokar vpisuje na disk.

Bit 4 služi za sinhronizacijo med modul za dodeljevanje vmesnikov in modulom za obdelavo DOS funkcij, saj preprečuje uporabo vmesnika, ki ga bo modul za dodeljevanje vmesnikov zamenjal.

Opisi vmesnikov so zbrani v tabeli, ki vsebuje 256 elementov, sami vmesniki se nahajajo v ostanku spomina ali v drugih spominskih bankah v mikroračunalnikih, ki imajo več kot 64K bytov spomina.

3.3.3.1 Dodeljevanje vmesnikov :

Modul za obdelavo DOS funkcij zahteva določen logični blok z diska. Če je ta blok že v enem od vmesnikov, ga modul za dodeljevanje vmesnikov vrne in ga premakne v LRU vrsti, drugače poišče modul za dodeljevanje vmesnikov prost vmesnik in zahteva branje tega vmesnika z diska.

Če modul za dodeljevanje vmesnikov ne more najti prostega vmesnika, poskuša najti popolnoma zapisan vmesnik (Written_Flag = 255) in ga zapiše na disk, zahteva branje vmesnika in ga vrne.

Če nobeden od vmesnikov ni popolnoma poln, modul zapiše na disk prvi dostopen vmesnik in zahteva branje novega vmesnika.

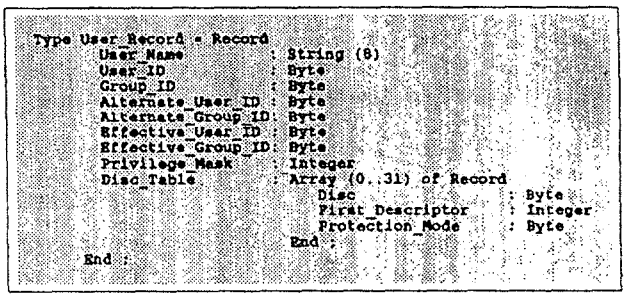
Če so vsi vmesniki zasedeni (LOCKED), modul sprosti vse vmesnike, ki so bili dodeljeni odprtim datotekam za opis datoteke in označi v vseh FCB-jih, da niso več pravilno nastavljeni. Po tej operaciji modul ponovno poskuša dobiti prazen vmesnik s pomočjo zgornjih treh točk.
Če je med iskanjem vmesnika modul za dodeljevanje vmesnikov zašel pregloboko v listo vmesnikov, torej je so vmesniki že precej zasedeni, bo modul zahteval zapis vseh popolnoma vpisanih vmesnikov (Written_Flag = 255) na disk. Če na ta način ne pridobi dovolj vmesnikov, bo modul zahteval še zapis vseh delno napolnjenih vmesnikov na disk.

Ob hkratnem delu modula za delo z diskovno enoto in ostalega operacijskega sistema ter dovoljšnjem številu vmesnikov (več kot 128) lahko s tem algoritmom dovolj zgodaj ugotovimo možno zasičenje sistema in zahtevamo vpis polnih vmesnikov na disk preden pride do popolnega zasičenja sistema. Takšno vpisovanje vmesnikov na disk poteka hkrati z ostalim delom sistema in dela sistema praktično ne prekinja. Zapisovanje vmesnikov na disk ob zasičenju sistema blokira sistem do konca zapisovanja in tako zmanjša hitrost delovanja sistema.

3.3.4 Opis uporabnika

Vsako vozlišče, ki se lahko pojavi v mreži, ima v glavnem računalniku ustrezno podatkovno strukturo, ki definira uporabnika, ki dela na tem vozlišču.
V tej podatkovni strukturi so zbrani vsi podatki o uporabniku, ki jih operacijski sistem potrebuje za svoje delo :

- Ime uporabnika



Slika 3.7 Opis uporabnika

- USER_ID in GROUP_ID sta polji, ki ju sistem uporablja ob določanju zaščite datoteke proti določenemu uporabniku
- ALTERNATE_USER_ID in ALTERNATE_GROUP_ID sta polji, ki ju sistem uporablja za shranjevanje USER_ID in GROUP_ID v primeru, ko uporabnik s pomočjo privilegiranega ukaza menja svoj USER_ID.
- EFFECTIVE_USER_ID in EFFECTIVE_GROUP_ID sta polji, ki ju operacijski sistem uporablja za shranjevanje USER_ID in GROUP_ID ob zagonu programa, ki začasno menja USER_ID in GROUP_ID na svoj USER_ID in GROUP_ID. Takšen program lahko izvaja privilegirane funkcije tudi za navadnega uporabnika.
- PRIVILEGE_MASK vsebuje definicijo privilegijev, ki jih ima uporabnik, torej seznam vseh zaščitenih operacij, ki jih uporabnik lahko izvaja.
- DISC_TABLE vsebuje preslikavo logičnih diskov v poddirektorije. Vsak element tabele vsebuje opis enega logičnega diska (A: do P: so stalni, ostali ostanejo samo za čas delovanja programa) in sicer :
  - diskovna enota, kjer se nahaja logični disk
  - kazalec na opis direktorija, ki ustreza logičnemu disku
  - zaščita direktorija, kot jo določi operacijski sistem ob OPEN klicu

S pomočjo tabele logičnih diskov lahko operacijski sistem zelo enostavno določi s katerim poddirektorijem hoče uporabnik delati, saj mu ni treba ponovno izvajati OPEN ukaza.

3.3.5 Opis diskovne enote

```
Type Disk_Description = Record
       Drive             : Byte
       Flags             : Byte
       Volume_Name       : String (8)
       Root_Descriptor   : Integer
       Bitmap_Descriptor : Integer
       Bitmap_Size       : Byte
       Dismount_Flag     : Byte
       Test_Offline      : Integer
       DOS_Init          : Integer
       DOS_Entry         : Integer
       Bitmap_First      : Byte
       Bitmap_Count      : Byte
       Bitmap_Buffer     : Integer
       Bitmap_Pointers   : Array (0..7) of Integer
     End ;
```

Slika 3.8 Opis diskovne enote

Opis diskovne enote vsebuje vse informacije, ki jih operacijski sistem potrebuje za delo z diskom :

- ime diska (samo za informacijo uporabnika)
- kazalec na opis glavnega kazala
- kazalec na opis tabele zasedenosti diska in njena velikost
- naslov procedure za
  - testiranje prisotnosti diska
  - inicializacijo diska
  - izvedbo DOS funkcije na tem disku
- Polja za vnaprejšnje dodeljevanje prostih sektorjev

Naslova procedure za inicializacijo diska in izvedbo DOS funkcije na disku omogočata modularno izvedbo diskovnega operacijskega sistema in celo uporabo večih diskovnih struktur na različnih diskih (npr. CP/M na disku A:, FENIX na disku B:).

3.3.5.1 Dodeljevanje prostih sektorjev :

Ko modul za obdelavo DOS funkcij zahteva prosti sektor na disku, skuša modul za dodeljevanje vmesnikov to zahtevo najprej zadovoljiti s pomočjo vnaprej zavzetih sektorjev v spominu. Če to ni možno, modul dodeli sektor s pomočjo tabele prostih sektorjev in dodatno zasede še 254 sektorjev, ki jih bo uporabil pri naslednjih zahtevah za dodelitev praznega sektorja.

Ta način dodeljevanja spomina precej izboljša dodeljevanje novih sektorjev na disku, saj v trenutku, ko je podatkovni sektor tabele prostih sektorjev v spominu skuša le-tega čimbolje izkoristiti. Tako zahteva dodeljevanje novih sektorjev dostop do diska le na vsakih 256 sektorjev, vmesnik, ki bi drugače hranil tabelo prostih sektorjev pa je lahko v vmesnem času sproščen.

Ob brisanju datoteke skuša modul za dodeljevanje vmesnikov vključiti novo sproščeni vmesnik v seznam vnaprej dodeljenih vmesnikov. Ta operacija uspe le v primeru, ko je seznam dodeljenih vmesnikov dovolj prazen in ko je sproščeni sektor dovolj blizu začetka diska. Ta zadnji pogoj preprečuje vnašanje sektorjev globoko znotraj diska v tabelo vnaprej dodeljenih sektorjev in tako zmanjša gibanje glave po disku.

4. ZAKLJUČEK

Operacijski sistem FENIX smo uspešno implementirali na mikroračunalniku DIALOG z 10M bytnim trdim diskom. Za delo z operacijskim sistemom je bil razvit poseben interpreter ukazov in podporni programi, ki so ekvivalentni UNIX podpornim programom, tako, da celota FENIX operacijski sistem in SHELL interpreter ukazov uspešno predstavljata UNIX - združljivo okolje v 8 bitni tehnologiji.

Operacijski sistem deluje dokaj hitro v primerjavi z drugimi mrežnimi operacijskimi sistemi realiziranimi v 8 bitni tehnologiji, precej zaslug za to nosi izredno hitra lokalna mreža DANTE/NET (hitrost prenosa 1M bit/s). Ob testiranju se je operacijski sistem obremenjen s štirimi delovnimi postajami izkazal za hitrejšega kot delo na sami delovni postaji z disketo, torej je njegova uporaba vsekakor upravičena.

Ob razvijanju operacijskega sistema se je rodilo nekaj idej, ki so vredne razmisleka in morda implementacije v naslednji verziji operacijskega sistema :

- podpora zaklepanja zapisov v datotekah, ki bi na sistemskem nivoju rešila problem hkratnega dostopa v bazo podatkov.
- integracija večih diskovnih pogonov v eno diskovno strukturo kot jo pozna UNIX operacijski sistem (MOUNT in UMOUNT ukaz)
- podpora datotek, ki bi se raztezale preko večih diskov. Tako bi lahko dosegli do 128M bytov velike datoteke brez spreminjanja diskovne strukture, ob manjših spremembah pa bi lahko dosegli še bistveno večje datoteke (cca. 4G byte).
- implementacija večih diskovnih struktur v okviru enega operacijskega sistema (npr. en CP/M disk in en FENIX disk)
- implementacija zahtevnejše diskovne strukture (npr. FILES-11 level 2)

Za implementacijo zadnjih ciljev bi bilo bolje uporabiti modernejšo tehnologijo kot je Z80 procesor, saj nam je 8 bitna tehnologija že ob razvijanju FENIX operacijskega sistema postavljala precej omejitev, predvsem zaradi omejitve naslovnega prostora na 64K bytov.

Literatura :

(1) Možnosti in nemožnosti mikroračunalnika DIALOG,Elektrotehniški vestnik, April 1987
(2) Peterson, Silbershatz, Operating system Concepts, Addison - Wesley 1983
(3) Comer, Operating System design, the XINU Approach, Prentice - Hall 1984
(4) XENIX operating system, Microsoft corp., 1986
(5) VAX/VMS V4.0 operating system, Digital equipment corporation, 1985
(6) IBM DOS 3.2 Technical Reference Manual, IBM Corp., 1985
(7) IBM DOS 2.0 Manual, IBM Corp. 1983
(8) Hyman, Memory resident utilities, interrupts and disk management, MIS 1986

SISTEMSKI UPORABNIŠKI VMESNIK ZA 8-BITNI UNIX ZDRUŽLJIV OPERACIJSKI SISTEM

INFORMATICA 1/88

Marjan Bradeško
Ljubo Pipan
Ivan Pepelnjak
Nikolaj Zimic
Fakulteta za elektrotehniko, Ljubljana

UDK 681.3.06

Izvleček

Članek opisuje izboljšavo 8-bitnega operacijskega sistema do te mere, da ga uporabnik z vidika sintakse vidi kot operacijski sistem Unix. Navedene so nekatere splošne značilnosti novega sistema, natančneje pa je opišan visokonivojski sistemski vmesnik, ki je v veliko pomoč uporabniku pri pisanju sistemskih uslužnostnih programov. S tem ima uporabnik sam možnost širiti nabor ukazov, ki predstavljajo vez med njim in operacijskim sistemom. Z že obstoječimi programi in zunanjim videzom predstavlja opisani operacijski sistem korak naprej pri prehajanju na bistveno zmogljivejše in bolj fleksibilne sisteme kot je CP/M, na osnovi katerega sloni nadgradnja, opisana v pričujočem članku.

Abstract

The paper presents the improvements of an 8-bit operating system to Unix - compatible environment as seen through user-level command line interface. We are describing some general features of the new operating system along with high - level system interface, which help the system programmers in writing new system utilities. With this aid, the end user can write his own utility programs, which represent the link between end-user and operating system environment. Existing programs can run under simulated CP/M environment, so this operating system represents a significant link between existing 8 bit operating systems and new UNIX - based systems.

1. UVOD

Doba 8-bitnih mikroračunalniških sistemov počasi mineva, hkrati z njimi pa se umikajo tudi operacijski sistemi, napisani zanje. Razvoj gre v smeri 16- in 32-bitnih sistemov. Temu ustrezno se prilagajajo tudi operacijski sistemi.

Uporabnik, ki bi rad na hitro prešel iz 8-bitnih na novejše sisteme, se znajde pred težavo. Aplikacijska programska oprema je vsa napisana na kožo njegovemu 8-bitnemu operacijskemu sistemu in običajno ni enostavno prenosljiva na nove sisteme. S tem problemom in z eno od uspešnih razrešitev le-tega se bomo ukvarjali v pričujočem članku.

Pri našem delu smo izhajali iz 8-bitnega mikroračunalniškega sistema z operacijskim sistemom CP/M. Sisteme smo povezali v lokalno mrežo z enim glavnim računalnikom s trdim diskom kapacitete 20 M. Glavni računalnik skrbi izključno za nadzor mreže, v katero je lahko povezanih do 16 postaj z dvema gibkima diskoma, ki imata vsak kapaciteto po 800 K. Prenos po mreži poteka po koaksialnem kablu (hitrost 1 Mbit), tako da je dostop iz katerekoli postaje do trdega diska glavnega računalnika hitrejši kot dostop na lokalni gibki disk postaje. S tem smo bistveno izboljšali performanse samega mikroračunalnika. S spremembo operacijskega sistema pa smo dobili sistem, ki navzven daje povsem drugačen videz od 8-bitnih sistemov. Uporabnik ga namreč z vidika sintakse vidi kot operacijski sistem UNIX, ki pa je že domena 16- in 32-bitnih sistemov. Pri vsem tem je v ozadju še vedno CP/M, tako da uporabnikova aplikacijska oprema ne potrebuje nobenih sprememb, hkrati pa je sistem zaradi povezave v mrežo in Unix-ove sintakse bistveno bolj fleksibilen.

Operacijski sistem na glavnem računalniku je povsem nov, na postajah pa je le modificiran CP/M. Pred BDOS je namreč dodan modul, ki ugotovi, če je neka sistemska funkcija rešljiva v okviru postaje, ali pa je potrebna intervencija glavnega računalnika. V tem primeru postaja pošlje zahtevo po servisiranju NDOS (Network DOS) funkcije v mrežo.

Operacijski sistem na glavnem računalniku je večuporabniški in večposlovni, servisiranje zahtev je rešeno s čakalnimi vrstami. Tudi podatkovne strukture na trdem disku so drugačne od CP/M struktur. Spremenjen je FCB (File Control Block) in zaradi drevesne strukture direktorijev tudi directory entry. Direktoriji so namreč datoteke, katerih elementi so imena

datotek v direktoriju in kazalci na opise (deskriptorje) teh datotek. Operacijski sistem CP/M dela z datotekami na fizičnih diskih A: oziroma B: (v primeru dveh pogonov), naš sistem pa za delo z datotekami uvaja logične diske. Če torej hočemo nek direktorij brati oziroma kaj pisati po njemu, mu moramo najprej prirediti enega od logičnih diskov. Šele potem so nam dostopne datoteke, ki se na njem nahajajo. Logični diski so vsi s številkami 3-32 (diski C:, D:, itd). Poleg standardnih Unix-ovih poddirektorijev /bin, /usr, /etc in /tmp imamo še dva poddirektorija

/lca - 'local A:' - CP/M diskovni pogon A:
/lcb - 'local B:' - CP/M diskovni pogon B:

V same podrobnosti jedra operacijskega sistema se tukaj ne bomo spuščali, ponovno poudarimo le to, da lahko poganjamo vso aplikacijsko programsko opremo, ki teče pod CP/M. Povejmo še, da za uspešno prehajanje med podatkovnimi strukturami CP/M postaje in glavnim računalnikom skrbita posebni proceduri Input Formatter in Output Handler operacijskega sistema na glavnem računalniku.

Pri našem opisu bomo ostali na nivoju vmesnika med uporabnikom in sistemom, kajti prav tu je najbolj fleksibilno mesto. Uporabnik ima namreč možnost širiti sistem s sistemskimi uslužnostnimi programi, ki jih je zaradi ustreznih orodij, ki jih ponuja nov sistem, zelo enostavno pisati. Podali bomo opis vmesnika, nekaterih sistemskih parametrov in na kratko našteli in opisali doslej napisane sistemske uslužnostne programe.

Uporabniški vmesnik je napisan v visokonivojskem programskem jeziku Turbo Pascal kot knjižnica uporabnih procedur, ki jo vključimo s stikalom ($I ime_knjižnice) v naš program. Izraz program bo odslej krajši izraz za sistemski uslužnostni program.

2. OBDELAVA UKAZNE VRSTICE

Interpreter ukazne vrstice (Command Line Interpreter) ni več sestavni del jedra operacijskega sistema, pač pa omogoča povezavo uporabnika s samim sistemom. V operacijskem sistemu CP/M se omenjeni interpreter imenuje CCP (Console Command Processor), pri Unixu pa je to Shell. Shell prebere ukazno vrstico, ugotovi, za kateri ukaz gre in nato kliče ustrezni program, ki izvede ukaz v skladu s parametri v ostanku ukazne vrstice. Omeniti velja, da so tako kot pri CCP tudi v Shell nekateri ukazi že vgrajeni in jih ni potrebno zaganjati kot ukazne datoteke (podaljšek .COM).

Vsaka ukazna vrstica vsebuje ime ukaza, zastavice za morebitne opcije, argumente k zastavici in ostale ukazne argumente.

2.1 Inicializacija

Ob vstopu v nek program nimamo na voljo nobenih podatkov o ukazni vrstici, natančneje o njenem ostanku (command tail). Začetek vsake ukazne vrstice je vedno ime ukaza - programa. Ta program mora potem, ko se zažene, dobiti vse informacije, potrebne za uspešno izvrštev ukaza. Vse informacije pripravi procedura INIT, ki jo pokličemo z:

INIT (flags_with_arguments)

kjer je flags_with_arguments niz dolžine 20 znakov, v katerem navedemo zastavice (drugo poleg druge) za opcije, ki v tem ukazu zahtevajo tudi argument. Omenjena procedura zgradi množico zastavic (vključno z morebitnimi argumenti), ki jih najde v ukazni vrstici, zgradi pa tudi seznam ostalih argumentov in izvede primerjanje vzorcev (pattern matching), če je to potrebno - v primeru, da ukaz deluje nad več datotekami.

Kot primer si poglejmo začetek programa tail, ki izpiše zadnjih n vrstic (znakov, blokov) datoteke ali pa preskoči prvih n vrstic (znakov, blokov) datoteke in jo nato izpiše do konca. Ukaz tail ima lahko štiri opcije (p, l, b, c), od katerih tri zahtevajo argumente (l-število vrstic, b-število blokov in c-število znakov).

```
begin
  INIT ('lbc') ;
  ...
end.
```

Slika 2.1 Inicializacija branja opcij

2.2 Zastavice za opcije

V različnih programih nastopajo najrazličnejše opcije. Procedura INIT zgradi ustrezne strukture, ki jih lahko pregledujemo z nekaterimi funkcijami in procedurami. Za testiranje prisotnosti neke opcije (zastavice za opcijo) je na voljo funkcija TEST_FLAG, ki vrne vrednost true, če je opcija v ukazni vrstici prisotna, sicer pa false. Za testiranje vrednosti argumenta, ki ga neka opcija zahteva, pa je na voljo funkcija FLAG_VALUE, ki vrne niz dolžine 40 znakov, če je argument prisoten in prazen niz, če argumenta ni (v primeru, da želimo argument pri opciji, ki ga sploh ne zahteva). Pokličemo ju z :

TEST_FLAG (flag) in FLAG_VALUE (flag)

kjer je flag ena od zastavic tipa char. Kot primer si spet vzemimo del programa tail.

```
begin
  ...
  if TEST_FLAG ('b') then Nb := FLAG_VALUE ('b') ;
  Num_Blocks := StrToInt (Nb) ;
  ...
end.
```

Slika 2.2 Testiranje zastavic

Gornji primer preveri, če je prisotna opcija b (število blokov) in v niz Nb vrne število blokov.

2.3 Argumenti

V ukazni vrstici se poleg opcij in njihovih argumentov pojavljajo še razni drugi argumenti kot so npr. poti, imena datotek, itd. Programerju so dostopni preko spremenljivke ARGC, ki pove, koliko je teh argumentov, in funkcije ARGV, ki jo pokličemo z :

ARGV (Arg_No)

Spremenljivka Arg_No je celoštevilska in pomeni zaporedno število argumenta v ukazni vrstici, ki ga vrne funkcija ARGV. Sem seveda štejejo tudi argumenti, ki jih generira proces primerjanja vzorcev, v primeru, da ukaz deluje nad več datotekami, ki jih podamo kot vzorec. Kadar so argumenti datoteke, dostikrat rabimo tudi funkcijo GET_PROT, ki nam vrne ustrezno zaščito datoteke (ki smo jo dobili s funkcijo ARGV) kot celo število. Če gre za lokalni CP/M direktorij, je zaščita -1, sicer pa velja naslednja shema :

Slika 2.3 Zaščita datoteke

Štirje biti (1 nibble) imajo pri tipu datoteke (Type) naslednji pomen) :

bit0 - datoteka je/ni direktorij
bit1,2,3 - nepomembni

Pomen bitov pri zaščiti datoteke (World, Group, User) pa je naslednji :

bit0 Read navadno datoteko lahko beremo, ne moremo pa je izvajati kot program (onemogočen zagon podatkovne datoteke)

bit1 Write navadno datoteko lahko spreminjamo; direktorij lahko spreminjamo (REN, DEL)

bit2 Execute datoteko lahko izvajamo kot program; v direktoriju lahko odpiramo datoteke

bit3 Delete datoteko lahko brišemo

Uporabo zgoraj opisanih funkcij pokaže naslednji del programa, ki izpiše vse argumente in pri tistih, ki so direktoriji, to posebej označi.
Omeniti velja še, da funkcija Argv vrača celoten argument tudi v primeru, da je argument neka pot z vzorcem. Primer :

```
begin
  ...
  if ARGC <> 0 then
    for Arg_No := 1 to ARGC do begin
      Write (ARGV (Arg_No)) ;
      if GET_PROT and $1000 <> 0 then Write ( <DIR>" ) ;
      Writeln ;
    end ;
  ...
end.
```

Slika 2.4 Branje zaščite datotek

/usr/include/*.pas

Argv po primerjanju vzorcev vrača argumente v obliki :

/usr/include/fcl.pas
/usr/include/tail.pas
/usr/include/head.pas
...

V primeru, da smo na enem od CP/M diskov, pa Argv vrača argumente v standardni CP/M obliki disk:ime_datoteke. Običajno potrebujemo le CP/M format imena datoteke, zato obstaja funkcija ARGV_RAW, ki iz celotnega opisa poti izloči le ime datoteke, ki ji spredaj doda logični disk, kateremu je prirejen direktorij, na katerem se datoteka nahaja. Zgornji primer po uporabi funkcije ARGV_RAW nad vsemi argumenti izgleda takole :

F:fcl.pas
F:tail.pas
F:head.pas
...

kjer je F: logični disk, kateremu je prirejen poddirektorij /usr/include.

Za izločanje imena datoteke iz podane poti obstaja še druga funkcija, katere opis bo podan kasneje.

3. DELO Z LOGIČNIMI DISKI

Kot smo že večkrat omenili, moramo direktorije prirediti logičnim diskom, če hočemo delati z njimi oziroma z datotekami na njih. Zato je v knjižnici na voljo nekaj procedur oziroma funkcij, ki omogočajo omenjene akcije. Imena vseh procedur se pričnо z ASG (assign). ASG_FILE je inačica ukaza ASSIGN v programskem jeziku Pascal. ASG_FILE priredi ime neki datoteki. Razlika med ASSIGN v Pascalu in ASG_File je v tem, da prvi zna delati le z imeni datotek v CP/M formatu, slednji pa z opisi celotne poti v drevesni strukturi. ASG_FILE kliče funkcijo ASG_PATH, ki je uporabna zato, ker kot rezultat vrne celo število, ki je številka logičnega diska, kateremu je omenjena funkcija priredila pot, ki smo jo podali kot argument. Ime datoteke iz opisa poti pa izloči funkcija ASG_PATH_EXTRACT, ki vrne celo število, ki povečano za ena pomeni položaj prvega znaka imena datoteke v opisu poti. Omenjene procedure oziroma funkcije kličemo z :

ASG_FILE (f, path)

ASG_PATH (path)

ASG_PATH_EXTRACT (path)

kjer je path niz dolžine 255, f pa neka datoteka, ki ji prirejamo ime. Omenjene funkcije pojasni spodnji primer. Program odpre za branje datoteko primer.pas na poddirektoriju /usr/include in izpiše njeno ime. Obenem izpiše, kateremu logičnemu disku je priredil pot /usr/work.

```
begin
  ...
  path := '/usr/include/primer.pas' ;
  ASG_FILE (f, path) ;
  reset (f) ;
  k := ASG_PATH_EXTRACT (path) ;
  Writeln ('File = ', Copy (path, Succ (k), 255)) ;
  ...
  k := ASG_PATH ('/usr/work') ;
  Writeln ('Logical disk = ', Chr (k + 64)) ;
  ...
end.
```

Slika 3.1 Prirejanje logičnega diska

Program da kot rezultat naslednji izpis :

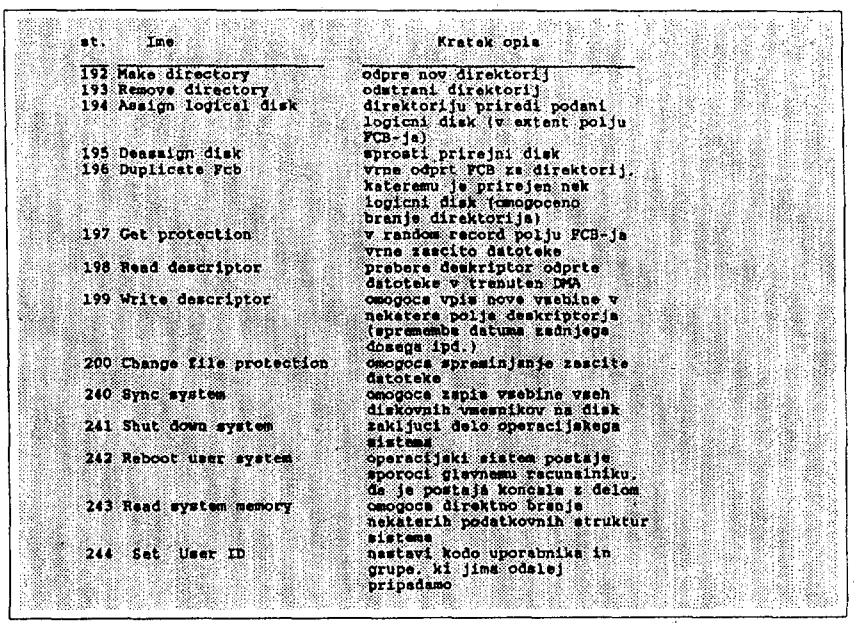
File = primer.pas

Logical disk = F

Uporabniku je na voljo še ena pomembna funkcija, ki mu omogoča klicanje BDOS funkcij (CP/M BDOS in vseh dodatnih), ki delujejo nad datotekami. Funkcijo pokličemo z :

FILE_BDOS (code, f)

kjer je code številka BDOS funkcije, f pa datoteka, nad katero izvajamo omenjeno funkcijo. Funkcija FILE_BDOS tudi sama skrbi za obravnavo napak. Ostale BDOS funkcije kličemo preko standardne procedure BDOS v Turbo Pascalu. S tem imamo pri pisanju uslužnostnih programov poleg viskonivojskega vmesnika na voljo tudi direktne klice funkcij na nižjem nivoju.



št.	Ime	Kratek opis
192	Make directory	odpre nov direktorij
193	Remove directory	odstrani direktorij
194	Assign logical disk	direktoriju priredi podani logicni disk (v extent polju FCB-ja)
195	Deassign disk	sprosti prirejni disk
196	Duplicate Fcb	vrne odprt FCB za direktorij, kateremu je prirejen nek logicni disk (omogoceno branje direktorija)
197	Get protection	v random record polju FCB-ja vrne zascito datoteke
198	Read descriptor	prebere deskriptor odprte datoteke v trenuten DMA
199	Write descriptor	omogoca vpis nove vsebine v nekatere polja deskriptorja (sprememba datuma zadnjega dostopa ipd.)
200	Change file protection	omogoca spreminjanje zascite datoteke
240	Sync system	omogoca zapis vsebine vseh diskovnih vmesnikov na disk
241	Shut down system	zakljuci delo operacijskega sistema
242	Reboot user system	operacijski sistem postaja sporoci glavnemu racunalniku, da je postaja koncala z delom
243	Read system memory	omogoca direktno branje nekaterih podatkovnih struktur sistema
244	Set User ID	nastavi kodo uporabnika in grupe, ki jima odslej pripadamo

Slika 3.2 Dodatne DOS funkcije

Dodatne, tako imenovane DOS funkcije podajamo zato, da zaokrožimo celoto orodij, ki so na voljo programerju pri pisanju sistemskih uslužnostnih programov. Za pravilno uporabo teh funkcij je potreben še dodaten opis nekaterih sistemskih struktur, ki pa jih tu ne bomo navajali.

Omeniti velja, da so nekatere od gornjih funkcij dostopne samo priviligiranemu uporabniku sistema SuperUser, ki edini tudi lahko zaključi delo operacijskega sistema.

Z vsemi zgoraj naštetimi pripomočki smo napisali množico uslužnostnih programov (okrog 45), ki so povsem podobni ukazom operacijskega sistema Unix. Unix-ove ukaze si lahko vsak bralec pogleda v enem od mnogih priročnikov, ki zanj obstajajo. Povejmo še to, da zaradi specifične konfiguracije našega sistema ni bilo možno pri vsakem ukazu implementirati vseh opcij, ki so na voljo v Unix-ovi originalni verziji.

4. Zaključek

Kaj smo z zgoraj opisano rešitvijo dosegli ? Predvsem preprosto, zmogljivo konfiguracijo lokalne mreže in ob majhnih stroških prehod na okolje zahtevnejših in bistveno zmogljivejših operacijskih sistemov z moznostjo preproste nadaljne širitve programske podpore samemu sistemu. Uporabnik, ki že ima mikroračunalnik, mora vanj vgraditi le ploščico, ki mu omogoča povezavo v mrežo in dokupiti nov operacijski sistem z vsemi orodji, ki mu omogočajo širitev. Konfiguracija je zelo primerna za manjše pisarne ali obrate, na njej pa teče vsa aplikacijska programska oprema, ki je bila doslej napisana za njegov stari operacijski sistem CP/M. Uporabnik bo ob uporabi take konfiguracije počasi prešel na nove, zmogljivejše sisteme in ko se bo nekoč znašel pred 32-bitnim strojem z novim operacijskim sistemom, ne bo imel posebnih težav s prehodom nanj.

Literatura :

(1) Možnosti in nemožnosti mikroračunalnika DIALOG, Elektrotehniški vestnik, April 1987

(2) Peterson, Silbershatz, Operating system Concepts, Addison - Wesley 1983

(3) Comer, Operating System design, the XINU Approach, Prentice - Hall 1984

(4) XENIX operating system, Microsoft corp., 1986

(5) IBM DOS 3.2 Technical Reference Manual, IBM Corp., 1985

(6) IBM DOS 2.0 Manual, IBM Corp. 1983

MOŽNOSTI ZAŠČITE PROGRAMSKE OPREME NA MIKRORAČUNALNIKIH

INFORMATICA 1/88

Nikolaj Zimic
Jernej Virant
Marjan Bradeško
Ivan Pepelnjak
Fakulteta za elektrotehniko, Ljubljana

UDK 681.3.06:003.6

Izvleček :

V članku so opisane možnosti zaščite programske opreme na obstoječih mikroračunalnikih. Predstavljene so najpogosteje uporabljene zaščite, stopnja zaščite, ki jo nudijo, napadi nanje in obramba programske opreme pred napadom.

Abstract :

The paper presents possible protection of software on existing microcomputers. The most often used protection schemes are explained, together with level of protection they enable, the attacks on these schemes and the possibilities of software defense against such attacks.

1. NAČINI ZAŠČITE PROTI KOPIRANJU

1.1 Uvod

Nedovoljeno kopiranje programov na osebnih in hišnih računalnikih se je začelo s kopiranjem igric. Te zaščite so prebijali mladi nadebudneži, ki so to počeli bolj iz zadovoljstva pred prepovedanim kot pa iz ekonomskih razlogov.

S povečanjem moči osebih računalnikov se je pojavila tudi zahtevna programska oprema, ki lahko stane tudi nekajkrat več kot strojna oprema. Zato je zaščita programov postala resen problem.

Z množičnim prodorom osebnih računalnikov se je razmahnilo tudi nedovoljeno kopiranje ali bolje rečeno kraja programov. S krajo programov je povzročena ogromna škoda avtorjem in proizvajalcem teh programov. Avtorji so se s problemom nedovoljenega kopiranja spopadli na različne načine:

- pri programih z velikim številom prodanih kosov so znižali ceno, tako da kraja programa postane skoraj nesmisel.
- programom so vzdignili ceno, tako da so z nakupom enega kosa plačane tudi morebitne kopije.
- programe so zaščitili na razne načine in tako otežili, če že ne preprečili kopiranje.

Pri ukradenih programih se pojavljajo še dodatni problemi, kot so garancija in vzdrževanje programov. Pri ukradenih programih ni vzdrževanja, zato so neuporabni za resno delo.

1.2 Problemi zaščite programov proti kopiranju na osebnih računalnikih

Pri osebnih računalnikih ni več običajnih zaščit programske opreme, ki so prisotne na velikih računalnikih. Pri osebnih računalnikih ima uporabnik vse privilegije in ima dostop do celotnega pomnilnika in do vseh perifernih enot. Procesor tudi nima svoje serijske številke, po kateri bi programska oprema ločila računalnike med seboj.

Samo kopiranje mora biti s strani sistema dovoljeno, da si lahko uporabnik naredi 'back up' programske opreme. Te zmožnosti omogočajo uporabniku tudi nedovoljeno kopiranje pogramske opreme.

Rezultat je enostavno prenašanje programske opreme z enega računalnika na drug računalnik. Programska oprema pač nima možnosti ugotoviti na katerem računalniku se izvaja.

1.3 Načini zaščite programske opreme proti kopiranju

Proizvajalci programske opreme so začeli s programskim paketom prodajati še dodatek, ki ga je zelo težko prekopirati. Pri cenejših verzijah programske opreme je to kar disketa, ki se razlikuje od običajne diskete in jo programska oprema lahko preverja. To so na primer poškodbe magnetnega materiala na točno določenem mestu ali pa nestandarden zapis na disketi.

Pri dražjih paketih se k programski opremi doda modul. Modul se običajno priključi na zunanje vhode. Programska oprema je pisana tako, da brez modula ne deluje. Modul je običajno zalit v epoksi smolo, kar otežuje kopiranje modula.

1.3.1 Zaščita na disketi

Disketa je običajno medij, na katerem se prodaja program. Ista disketa lahko služi še kakor ključ, brez katerega program ne deluje pravilno. Tak način zaščite je dokaj poceni, saj zahteva samo spremembo na disketi.

Običajna disketa je organizirana po sledeh, to je 40 ali 80 koncentričnih krogov. Vsaka sled je logično razdeljena na sektorje. Pred vsakim sektorjem je poseben zapis (id field), v katerem je kodirana številka sledi, številka strani, številka sektorja in dolžina sektorja. S strani procesorja se v kontroler gibkega diska zapiše samo številka sektorja, sledi in strani, kontroler pa sam poskrbi, da se bodo podatki pisali ali brali iz pravilnega mesta na disketi. Dodatni zapisi, ki skrbijo za pravilen dostop do sektorja, so običajno procesorju nevidni, lahko pa se jih z direktnim dostopom do kontrolerja prebere.

Na disketo se lahko poleg običajnih informacij zapiše dodaten id field, ki pa mu ne sledi sektor. To pomeni, da je na disketi podatek o dodatnem sektorju, ki pa ne obstaja. Pri normalni uporabi diskete z dodanim id field-om se le tega ne opazi. Služi samo za kontrolo pri kopiranju. Pri običajnem formatiranju se dodaten id field ne zapiše, kar programski paket kasneje preveri in 'ugotovi', da ni pognan iz originalne diskete.

Naslednja možnost je zapis ene sledi več kot je običajno. Disketne enote običajno omogočajo formatiranje diskete na večje število sledi, kot je standardno. Ker operacijski sistem dodatnih sledi ne uporablja, so le te iz uporabniškega stališča povsem nevidne. Enostavno pa se lahko uporabijo za shranjevanje dodatnih informacij, ki služijo za zaščito proti kopiranju.

Zaščita diskete je možna tudi z odstranitvijo magnetnga materiala z diskete. To se običajno naredi z laserjem. To so tako imenovane laserske luknje. Glavna prednost laserskih lukenj pred raznimi zapisi na disketi je v težjem kopiranju, saj je potreben fizičen poseg na disketo. Programska oprema preverja napako na disketi. Če napaka ni na pričakovanem mestu, programska oprema ne deluje pravilno. Preverjanje napake se izvede s pisanjem in branjem sekvence podatkov na poškodovan sektor. Ker je magnetni medij poškodovan, so prebrani podatki spremenjeni. Seveda so podatki spremenjeni točno na določenem mestu.

Boljša metoda zaščite je z dvema laserskima luknjama na istem sektorju. To pomeni, da imamo na enem sektorju dve točno določeni napaki. S pisanjem in branjem poškodovanega sektorja lahko izmerimo razdaljo med luknjami na približno 5% natančno. Natančnost meritve nam določa hitrost vrtenja diskete, ki pa ni popolnoma enaka pri vseh disketnih enotah. Dolžina se izmeri s preštevanjem pravilno prebranih zlogov med dvema napakama. Na mestu napake disk kontroler izgubi sinhronizacijo, zato so lahko podatki med dvema napakama pomaknjeni v levo ali desno. Programska oprema tako preverja, če je napaka na sektorju in če je razdalja med dvema napakama pravilna.

Disketo, ki je zaščitena z lasersko luknjo, lahko poljubno formatiramo, ne da bi izgubili ključ zaščite. To pomeni, da si lahko naredimo poljubno število kopij, ki služijo kot back up. Te kopije pa so neuporabne brez originalne diskete, ki je zaščitena z lasersko luknjo. Programsko opremo lahko prekopiramo na trdi disk, seveda pa ne deluje brez originalne diskete.

1.3.2 Zaščita z dodatnim modulom

Pri dražjih paketih se poleg programske opreme doda modul. Modul se običajno priključi na serijski, paralelni vmesnik ali med tipkovnico in računalnik.

Nekateri novejši moduli se priključujejo med računalnik in tiskalnik. Ta modul se sproži ob točno določeni sekvenci in ne moti običajnega dela s tiskalnikom.

V mudulu je običajno nekaj integriranih vezij, ki so zalita v smolo. Ta modul je za uporabnika črna škatla, brez katere programska oprema ne deluje.

Modul je običajno avtomat, ki na sprejeto sekvenco podatkov odgovori s točno določeno sekvenco, ki jo preverja programska oprema. Sekvenca se običajno generira s psevdo naključnim generatorjem.

Zaščita je s stališča kopiranja modula dokaj zanesljiva, saj je pri daljši sekvenci praktično nemogoče ugotoviti način prekodiranja vhodne v izhodno sekvenco.

1.3.3 Zaščita programskih paketov na trdem disku

Določeno programsko opremo je možno instalirati na trdi disk, tako da le ta ne potrebuje diskete s ključem za normalno delo.

Običajno se programsko opremo instalira na trdi disk z zaščiteno disketo. Na disketi se pri inštalaciji zmanjša števec tako, da je z eno zaščiteno disketo možno instalirati samo omejeno število kopij. Števec inštalacij se lahko poveča samo v primeru, da se po posebnem postopku onemogoči prej instalirano kopijo na trdem disku.

Tak način dela omogoča, da se inštalacijsko disketo, ki je zaščitena, uporablja samo pri inštalaciji. Drugače pa ta disketa služi kot back up.

Pri inštalaciji na trdi disk se običajno v programsko opremo zapišejo informacije o fizičnem položaju le te na trdem disku. Pri kopiranju je praktično skoraj nemogoče prekopirati programsko opremo na fizično enake lokacije. Prazen prostor na trdem disku, ki se lahko uporabi za kopiranje, se spreminja pri vsakem brisanju ali pisanju na disk. Tako je samo pri praznem disku enostavno določiti, kam se bodo podatki pri kopiranju pisali.

1.4 Problemi uporabnika pri zaščitenih programskih paketih

Zaščita programskih paketov lahko pri nepreviđni zaščiti postane dvorezen meč za legalnega uporabnika. To pomeni, da se lahko zaščita obrne proti uporabniku, ki je programski paket kupil.

Problemi nastanejo, če se poškoduje disketa, ki služi kot ključ za pravilno delovanje programa. To pomeni avtomatsko nekajdnevno neuporabnost programskega paketa. Proizvajalci namreč zamenjajo poškodovano disketo, na kateri je zakodiran ključ, z novo.

Problemi se pojavljajo tudi pri dodatnih modulih. Če je modul slabo priključen ali če pride do napake pri komunikaciji z modulom, se lahko sproži zaščita, ki je vgrajena v programski paket. To ima za posledico prekinitev programa ali pa njegovo nepravilno delovanje, kar pa zmanjšuje zanesljivost programske opreme.

Zaradi zmanjševanja zanesljivosti delovanja programskega paketa se običajno zaščita postavi samo na začetek programa. To seveda olajšuje napad na zaščito in zmanjšuje zanesljivost zaščite.

2. NAPADI NA ZAŠČITENE PROGRRAME

2.1 Možnosti 'popravljanja' zaščitenih programov

Pri zaščitenih programih vedno obstaja rutina ali skupina rutin, ki preverjajo prisotnost zaščite. Če ključ, ki je na disketi ali pa v posebnem modulu, ne obstaja, se program običajno prekine ali pa ne deluje pravilno. Eden izmed možnih napadov na zaščito je onemogočitev prej omenjenih rutin.

Seveda pa je dokaj zapleteno najti rutine, ki tvorijo zaščito programa. Avtorji rutin seveda poskušajo te rutine čimbolj zakriti. Rutine imajo seveda nekaj značilnosti, ki olajšujejo iskanje.

Glavne značilnosti rutin, ki skrbijo za zaščito so:

- prve razlike med delovanjem programa pri originalnem ključu in brez, se pojavijo prav v njih.
- te rutine kličejo periferne enote, na katere je priključena zaščita (krmilnik gibkega diska, serijski in paralelni izhod, ...).
- včasih vektorji prekinitev kažejo na te rutine, posebno v primeru, če periferne enote delujejo pod prekinitvami.

Zanesljivost programske opreme, proti napadu se lahko poveča s testiranjem okolja. To pomeni, da programska oprema odkrije programe, ki nadzirajo delovanje programske opreme. Če program odkrije, da je 'opazovan' ima možnost, da zavede opazovalca.

2.2 Možnosti kopiranja zaščitenih disket

Diskete so običajno zaščitene na tri načine:

- z nestandardnim zpisom.
- s povečanjem števila sledi.
- z odstranitvijo magnetnega materiala (tako imenovane laserske luknje)

Običajni programi za kopiranje disket ne prepišejo skritih zapisov na disketi. Te lahko prekopirajo programi, ki 'pričakujejo' skrite zapise in naredijo popolno kopijo, ne glede na to, kaj je zapisano na disketi.

Dokaj enostavno je tudi prekopirati sledi, ki jih običajno ni. Seveda pa moramo prej odkriti, da je programska oprema zaščitena na ta način.

Če je disketa zaščitena z laserskimi luknjami, je le to dokaj težko prekopirati brez drage opreme. Seveda pa ni nujno, da poškodbo povzročimo z laserjem. To poškodbo se lahko povzroči z ostrim predmetom. Pomemben je le fizičen položaj poškodbe in njena širina.

Prva dva primera zaščite je možno prekopirati popolnoma programsko, brez fizičnega posega v računalnik ali na disketo. Prekopirajo jih tudi aparature, ki so narejene za hitro kopiranje programske opreme. Take aparature namreč ne preverjajo zapisa na disketi, ampak delajo popolno kopijo.

2.3 Možnosti programskega simuliranja zaščitene diskete

Napako na disketi, ki je povzročena z laserjem, je možno programsko simulirati. To pomeni, da programska oprema ne dobi odgovora z diska ampak iz posebnega programa, ki simulira napako.

Disketo z zaščito lahko programsko analiziramo in izdelamo algoritem po katerem se spremeni zapisani sektor. Če je na sektorju enojna napaka, je začetek sektorja pravilen, nato pa sledi mesto napake. Na mestu napake je prebrana vrednost nesmiselna. Po fizični napaki na disketi spet sledijo podatki, ki so bili zapisani. Ti podatki so zaradi izgube sinhronizacije lahko pomaknjeni. Premik je mišljen na nivoju bita.

Če je na enem sektorju več laserskih lukenj, pomeni, da imamo več področij, kjer so prebrani podatki popolnoma nedefinirani. Točno določena pa je razdalja med laserskimi luknjami. Razdalja se lahko izmeri s številom zlogov med dvema napakama.

Z laserskimi luknjami zaščiten program lahko prelisičimo s spremembo programa, ki bere sektor. To pomeni, da moramo napisati program, ki simulira zaščiten sektor. Ta program inštaliramo kot del sistema tako, da zaščitena programska oprema ne opazi spremenjenega branja diskete. Seveda pa se ta program aktivira samo v primeru branja in pisanja na sektor z laserskimi luknjami. V vseh ostalih primerih je branje in pisanje običajno.

2.4 Možnosti kopiranja dodatnih modulov

Dodatni moduli so običajno vezja, ki so zalita v epoksi smolo. To pomeni da nimamo vpogleda v notranjo zgradbo vezja. Sam algoritem je lahko zakodiran tudi v integrirana vezja, kot so naprimer PAL, EPLD, GAL vezja. Ta vezja lahko zaščitimo proti kopiranju s programiranjem posebnega bita v integriranem vezju. To pomeni, da ne moremo prebrati podatkov za programiranje. Podatke za programiranje lahko dobimo le z analizo delovanja integriranega vezja.

Analiza delovanja integiranih vezij oziroma modulov je dokaj zamudna zaradi velikega števila možnih kombinacij. To število lahko povečamo s številom notranjih stanj flip-flopov. Če imamo v modulu ali v integriranem vezju 10 flip-flopov, je število notanjh stanj 2 na 10. potenco.

Same module oziroma integrirana vezja je dokaj težko prekopirati. Lažje je odkriti rutine, ki uporabljajo modul in jih spremeniti.

2.5 Možnosti uporabe programskih debugerjev

Debuger je program, ki je namenjen ugotavljanju napak v programski opremi. Omogoča nam spremljanje izvajanja programa.

Tako lahko spremljamo izvajanje programa z zaščito in brez nje. Iz spremljanja lahko ugotovimo, kje je jedro zaščite in ga zaobidemo. Z debugerjem lahko iščemo inštrukcije, ki so vezane na delovanje zaščite. To so naprimer instrukcije za branje in pisanje kanala, na katerega je priključen modul za zaščito.

V opazovani program lahko postavimo prekinitvene točke, v katerih opazujemo stanje pomnilnika in registrov. Iz podatkov na skladu lahko ugotovimo, od kje je bil določen podprogram klican.

Opazujemo lahko tudi komunikacijo z dodatnimi moduli. Z dobljenimi podatki lahko napišemo program za analizo modula. Po drugi strani pa lahko odkrijemo algoritem, ki skrbi za preverjanje modula. Ta algoritem mora biti ekvivalenten algoritmu v modulu. Iz algoritma se lahko brez večjh problemov izdela modul.

Nekateri programski paketi imajo vgrajeno preverjanje okolice v kateri delujejo. To pomeni, da lahko odkrijejo, če delujejo pod kakšnim dodatnim programom, ki opazuje njihovo delovanje. Če odkrijejo, da so opazovani, se na ustrezen način 'branijo'. Spremenijo način delovanja tako, da kar najbolj otežijo spremljanje dogajanj v računalniku.

2.6 Možnosti uporabe emulatorjev

Emulatorji so najnevarnejši za zaščitene programe. Njihovo prisotnost je programsko nemogoče odkriti, ker sonda emulatorja zamenjuje procesor računalnika. Emulator omogoča v realnem času spremljanje programa oziroma dogajanj v pomnilniku ali perifernih enotah.

Z emulatorjem dokaj enostavno odkrijemo razliko v delovanju programa, ki ima ključ (modul ali zaščitemo disketo) ali je brez.

3. MOŽNOST UPORABE KRIPTOGRAFIJE

Pri običajnih programskih paketih je problem, da mora zaščita delovati tudi proti nedovoljeni uporabi legalnega uporabnika. To pomeni, da uporabnik programski paket normalno uporablja, ne sme pa ga prekopirati v smislu nedovoljene nadaljnje kopije.

Pri uporabi kriptografije mora biti ključ za uporabo programa skrit tudi pred legalnim uporabnikom. To pomeni, da mora biti skrit na disketi ali dodatnem modulu. Rezultat je, da je tak način zaščite popolnoma enakovreden prej opisanim načinom. Nekoliko se spremeni način napada, ki mu cilj postane iskanje ključa za dešifriranje programske opreme.

Druga možnost bi bila uporaba modula, v katerem bi bil vgrajen postopek in ključ za dešifriranje programske opreme. Tudi ta možnost je neuporabna, ker modul lahko služi za dešifriranje programske opreme, ki se dešifrirana lahko poljubno kopira.

4. ZAKLJUČEK

Zaščita programske opreme ne prenese resnejših napadov. Popolnoma pa zadostuje za preprečevanje kopiranja uporabnikom, ki niso vešči v sistemskem programiranju.

Postavlja se tudi vprašanje smiselnosti kopiranja programov. Pri nedovoljeni kopiji je pravilno delovanje programske opreme vprašljivo. Hkrati nam odpade tudi vzdrževanje s strani prodajalca. To sta dva dejavnika, ki govorita v prid nakupu programske opreme, saj se resnejše delo ne more izvajati na programski opremi za katero ne odgovarja avtor.

Na Zahodu so strogi zakoni, ki ščitijo avtorje pred krajo programske opreme, vendar je kraj le preveč, da bi lahko zaupali samo zakonu.

Na nivoju osebnih računalnikov ni zanesljive zaščite programske opreme. Zaščita je odvisna le od iznajdljivosti avtorjev na eni strani in 'lopovov' na drugi strani. Seveda pa to ne pomeni, da lahko vsakdo v kratkem času prebije zaščito v programski opremi.

LOGIČNO TESTIRANJE PROTOKOLOV (pregled)

INFORMATICA 1/88

Tomaž Kalin
Tone Vidmar
Fakulteta za elektrotehniko, Ljubljana

UDK 681.3.066

POVZETEK: Namen prispevka je predstaviti splošno vsebino problematike logičnega testiranja komunikacijskih protokolov, seznaniti bralca z trenutnim stanjem reševanja problematike v svetu in na kratko predstaviti naš prispevek k temu. Ta se nanaša predvsem na popolnejšo sintakso testnega modela komunikacijskega protokola (robni pogoji testiranja), arhitekturo in strukturo testnega modela ter razvoj t.i. rekurzivne testne metode kompleksnih komunikacijskih sistemov.

KJUCNE BESEDE: protokol, logično testiranje, modeliranje

I. VSEBINA PROBLEMA

Komunikacijski protokol je predpis (opis, standard), ki opredeljuje, v konteksu komunikacijskega podsistema način komuniciranja med porazdeljenimi procesi (sistemi, aplikacijami). Implementacija protokola predstavlja prav tako porazdeljen sistem, kjer ima vsak element (proces) določeno lokalnost (nič dogodki) in interakcijo z okolico (oddaja, sprejem komunikacijskih sporočil). Opis protokola je po tem torej opis porazdeljenega sistema. V splošnem gre lahko za porazdelitev večih procesov, ki pa se po pravilu grupirajo v dve večji podgrupi procesov P(N) in P*(N), ki predstavljata komunicirajoči točki povezani z prenosnim medijem. "N" poneni protokolarni nivo komunikacijskega sistema (konkretno lahko tudi v smislu protokolarnega modela OSI). V eksploataciji porazdeljenega sistema, ki je podprt z komunikacijskim podsistemom (P(N),P*(N)) lahko pride do različnih defektov (nedefiniran sprejem protokolarnega sporočila, smrtni objem -dead lock-, preseženje pomnilniških kapacitet prenosnega kanala -over flow-, dvounmost postopka posameznih komunikacijskih faz -vspostavljanje, prenos podatkov, rušenje zveze-...). Metode logičnega testiranja so namenjene odkrivanju in odpravljanju takih napak v protokolarnem predpisu. Tu nikakor ne gre za pristope, ki so podobni simulacijskim pristopom, vendar pa so rezultati podobni. Povdariti je treba, da je vložek dela za izvedbo testa na osnovi simulacijskih pristopov neprimerno večji, kot pa pri pristopu z logičnim testiranjem, kar je pomembna prednost za tak način testiranja. Pri tem pa je poleg same testne metode izredno pomembno kako se protokolarni predpis formalno opiše (formalen opis porazdeljenega sistema). In ne nazadnje je treba povdariti, da je postavitev testnega modela in interpretacija rezultatov testiranja prav tako parameter, ki bistveno vpliva na celotno uporabnost logičnega testiranja. Na osnovi povedanega problematika logičnega testiranja protokolov razpade na tri bistvene dele: -formalen opis protokola, struktura testnega modela ter izbira testne metode-.

Opis protokola v naravnem jeziku formalno in interpretacijsko ni korekten in je neprimeren za kakršnokoli avtomatsko obdelavo. Dejstvo je, da je večina standardov in različnih specifikacij podana v taki obliki. Tak opis je samo na prvi pogled enostaven in primeren. Ob postopku razvoja tehnologije od specifikacije k implementaciji pa se pokažejo sledeče pomankljivosti: -ne zagotavlja enoumnega opisovanja protokolarnih konstruktov, enostavno otkrivanje nepotrebnih deklaracij ni možno, ne omogoča preverjanja in potrjevanja funkcionalnosti protokola, ne podpira tehnologije razvoja implementacije, ne nakazuje okvirov implementacije, ne omogoča uporabe testnega orodja-.

Iz vsega naštetega sledi, da je med standardom in implementacijo protokola ali poljubnega algoritma (procesa) velika in pomembna vrzel, ki vsebuje širok spekter problemov in možnih rešitev.

Produkcija kompatibilnih implementacij protokola predstavlja jedro problema, ki ga iz različnih zornih kotov in smeri poskušajo reševati različne raziskovalne in razvojne skupine. Ne glede na to, da OSI protokolni model dokaj dobro strukturira in lokalizira določene probleme specifikacije, je način formalnega opisovanja porazdeljenega sistema še vedno ključen ko se govori o logičnem testiranju protokolov. Danes uporabljajo za specifikacijo protokolarnih standardov največkrat strukture končnega avtomata, Petrijeve mreže, abstraktne podatkovne tipe, posebne jezike za specifikacijo, višje programske jezike in podobno.

IBM, Švica, je:"H. RUDIN: Automated Protocol Validation: Some Practical Exsamples, Proceeding of the Sixt International Conference on Computer Communication". V članku je podan hierarhični design in strukturiran pristop k reševanju problema, na originalen način, neodvisen od vseh do sedaj podanih referenc. Tip napak, ki jih metoda odkriva, je prikazan na primeru nivoja X.25, ki je imel v verziji 1976 določene logične nepravilnosti. Po standardu je bila ta nekorektnost definicije tretirana kot interna proceduralna napaka, kar pa je nesprejemljivo glede na celoten standard, ki tako situacijo očitno dopušča. V kasnejših verzijah standarda X.25 so to napako korigirali. Poleg tega postopek omogoča odkrivanje pasusov, ki se nikoli ne zgodijo, protokol pa jih predvideva (dead koad), in situacije 'dead lock', to je primer, ko vsi čakajo na sprejem, ki ne bo nikoli izvršen. V članku je za odkrivanje teh napak opisan koncept 'perturbacije', ki generira tesno drevesno strukturo. Ta članek nas je spodbudil, da smo poiskali vse reference v zvesi s tem člankom, ki so sledeče: "Pitro Zafiropulo, "Protocol Validation by Duologue- Matrix Analysis", IEEE Transactions on Communications, Vol. Com-26, Dec. 1980", "Pitro Zafiropulo, Colin H. West, Harry Rudin & Daniel Brand,"Towards Analysing and Synthesizing Protocols", Transactions on Communications, Vol. Com-28, April 1980", "Harry Rudin, Colin H. West, "A Validation Technique for Tihtly Coupled Protocols", IEEE Transactions on Computers, Vol. C-31, July 1982", "Colin H. West, "General Technique for Communication Protocol Validation", IBM J. RES. Vol. 22, July 1978". Ta grupa člankov nam je bila poleg že prej naštetih, ki se nanašajo predvsem na specifikacijske probleme, osnova za razvoj t.i. rekurzivne testne metode. Poleg teh pa je potrebno našteti tudi sledeče:"Philip M. Merlin, "Specification and Validation of Protocols", IEEE Transaction on Communicatins, Vol. Com-27, Mar. 1979","George V. Bochman, "A General Transition Model for Protocol and Communications Services", IEEE Transactions on Communications, Vol. Com-20, April 1980", "Georg V. Bochman, "Experience with Formal Specification Using EFMS Model", IEEE Transactions on Communications, Vol. Com-30, Dec. 1982", "Andre A. S. Danthim, "Protocol Representation with FMS", IEEE Transaction on Communications, Vol. Com-28, April 1980, "Georg V. Bochman, "Finit State Description of Communication Protocols", Computer Network Protocols, Liege - Belgija, februar 1978", "J. Hartmanis, R. E. Stearns, Algebraic Structure Theory of Sequential Machines, Prentice - Hall 1966".

Naša posebna pozornost je bila posvečena tudi možnosti uporabe Petrijevih mrež, saj imajo te prav posebne možnosti. Petrijeve mreže (PN - Petri Nets) so formalen model informacijskih tokov sistema. Namenjene so predstavljanju dogodkov ali procesov, ki v določenem trenutku postanejo konkurenti. Začetnik razvoja je C.A. Petri, ki je prva dela objavil 1965. Dokončno potrditev so dobile v okviru dveh konferenc (Concurent Systems and Parallel Computation, Woods Hole 1970 in Conmference on Petri Nets and Related Methods, 1975). V Evropi so jih največ razvijali na 'Institut fur Informationssystem-forschung' v Bonnu. Na tem področju je veliko storjenega, vendar pa dostopne literature, ki bi združevala vse rezultate teorije, ni. Večino informacij je treba iskati v člankih ali publikacijah lokalnega dosega (doktorati). Edina pregledna informacija je dana v članku 'Petri Nets' in knjigi 'Petri Net Theory and the Modelling of Systems' avtorja J.L. Petersona. Študij tega področja nam je koristil predvsem za spoznavanje tehnik modeliranja porazdeljenih sistemov, čeprav smo se v nadaljevanju dela njihovi uporabi odrekli.

Vse do sedaj našteto se nanaša na deterministične metode logičnega testiranja. V zadnjem času se pojavljajo tudi novi trendi z uporabo hevrističnih metod, vendar pa imajo trenutno prve še vedno boljše in za prakso sprejemljivejše rezultate.

NAŠ PRISPEVEK K OPISANI PROBLEMATIKI

Nova spoznanja in predlagane izboljšave testnih postopkov so rezultat študija, ki se je začel leta 1983. V tem času je poleg pričakovanih rezultatov (seznanjanje s problemom in njegovo razumevanje, opredelitev obstoječega stanja v svetu,...) nastalo tudi več verzij programskih paketov za logično testiranje protokolov (prvi v razvojnem oddelku za komunikacije v DELTI), kar pa je bilo odločilnega pomena in potreben pogoj za globlje spoznavanje z opisano problematiko testiranja.

Končen rezultat dela je "logični protokolarni analizator" - LPA, ki je v svoji današnji obliki realiziran v okvirih manjšega osebnega računalnika. Zanimivo je omeniti, da je prva izvedba testnega postopka, ki je nastala na osnovi podatkov iz navedene literature, zasedla zmogljivosti večjega mini računalnika. Ta podatek lahko delno ilustrira napredek v razvoju tehnologije testiranja in kvaliteto izvedbe LPA-ja. Poleg tega je obstoječa verzija LPA-ja po kvantitativnih in kvalitativnih lastnostih testiranja neprimerno boljša od prvih izvedb, iz katerih se je razvila. LPA je danes po funkcionalnosti in namenu uporabe primerljiv s klasičnimi logičnimi analizatorji za programsko ali strojno opremo, s tem da gre tu za logično testiranje interakcije med posameznimi elementi porazdeljenega sistema.

Poleg testne metode kot osnove izvedbe LPA-ja (namenjena je logičnemu testiranju interakcije v najožjem smislu) je razvita celotna "tehnologija" razvoja komunikacijske opreme, ki je zasnovana tako, da je programabilna in funkcionalno povezana z izvedbo izbrane testne metode.

Bistveni prispevek našega dela je, poleg vključitve testne metode v celovit postopek razvoja komunikacijskih sistemov in določene izboljšave same metode, možnost testiranja izvedbeno odvisnih sistemov oziroma testiranje protokolarnega para v njegovem izvedbenem okolju. Zasnova testnega modela in njegova sintaksa sta zato pomembna parametra testne metode, saj bistveno vplivata na kvaliteto in kvantitativne zahteve samega testiranja; predvsem na zahteve po računalniških zmogljivostih same izvedbe testnega postopka. Poleg tega je testni model bistven za dosego drugega zastavljenega cilja, to je vpeljave robnih pogojev v postopek testiranja. Zavedati se je namreč treba, da robni pogoji, kot so vrsta kanala, prioriteta oddaje/sprejema, možnost kolizijskih dogodkov na prenosnem mediju, vrste sporočil in podobni, bistveno vplivajo na logično in izvedbeno strukturo določenega komunikacijskega nivoja.

Podan je rekurzivni postopek testiranja na osnovi predlaganega testnega modela. Testni model je predstavljen kot mreža avtomatov. Za dano strukturo med seboj povezanih avtomatov je podan postopek paralelno/serijske kompozicije in minimizacije, katere rezultat je avtomat prenosnega medija v širšem smislu. Prenosni medij v širšem smislu je element modela, ki vsebuje lastnosti fizičnega prenosnega medija

II. PREGLED OBSTOJECEG STANJA

Namen tega poglavja je na kratko predstaviti dogajanja na tem področju v svetu in sistematizirati probleme, ki se rešujejo v 'prostoru' med danim opisom protokola in njegovo implementacijo. Članki, ki nimajo eksplicitno navedenega izvora so iz zbornika konference 'EUTEKO 83' (European Teleinformatics Conference'), ki je bila organizirana v okviru projekta COST 11 bis. To ni kompletna informacija, vendar pa dokaj dobro zajema probleme in načine njihovega reševanja predsvem v evropskem prostoru, ki pa ta trenutek ne predstavlja nepomemben del vseh svetovnih dogajanj. Nasprotno, z sprejetjem razvoja komunikacijskih sistemov pod okrilje državnih administracij je Evropa postala celo vodilna za določena področja. Tu imamo v mislih celo vrsto javnih servisov, ki so standardizirani in veliko podporo, ki se v Evropi daje razvoju sistemov v okviru t.i. ISDN (Integreated System Digital Network).

Uvodna referenca, ki sistematizira pristop, je članek "S. ALFONZETTI, A Reference Model for a Protocol Entity".V tem delu je podan formalen pristop k razčlenjevanju problema (protokola) na particije in segmente, glede na kriterije kot so :- implementacijska odvisnost, funkcionalna celovitost in avtonomnost, delovne faze protokola, hierarhična porazdelitev, tip procesiranja, dinamičnost in statičnost podatkovnih struktur. Podana je struktura modela posameznega protokolarnega nivoja in terminologija, ki se je držijo v Evropi vsi, ki se ukvarjajo s problematiko tega tipa. Nakazan je način povezovanja segmentov in interpretacija rezultatov pri odkrivanju t.i. 'dead lock' situacije . V delu se navaja cela vrsta pomembnejših referenc, ki lahko služijo kot vodilo pri bodočem delu.

Druga referenca nakazuje konkretno rešitev na osnovi podanega modela: "J.P. Ansard, VALIDOC: A Protokolar and PDIL - A Language for Protocol Description and Implementation". Članek opisuje rezultate ene od kompletnejših rešitev, ki vodijo od specifikacije do implementacije. Sistem je realiziran na institutu INRIA, Rocquencourt - Francija. Sistematizira njihov pristop in opisuje glavne elemente produktov in postopek dela, predvsem pa se iz članka čuti obsežnost, kompleksnost in vsa potrebna širina razvojnega dela, da se pride do uporabnega produkta.

Tretja pomembnejša referenca je po vsebini podobna predhodnji:"Bengt JONSSON: An Extended State Machine Approach to Specification, Validation and Testing of Protocols". Delo opisuje interaktiven sistem za opisovanje objektov (protokolarnih nivojev, N-entities) v obliki končnih avtomatov. Sistem sestavljata dva produkta, testni del CADIE in specifikacijski jezik ASYL. Postopki preverjanja in testiranja niso predstavljeni, čeprav so navedeni kot deli celotnega sistema.

Četrta referenca ne opisuje razvitega sistema do nivoja uporabnosti, konkretneje pa opisuje specifikacijo in povezovanje posameznih modulov specifikacije. Menimo, da je to eno od uporabnejših del:"O.E. MARTIKAINEN: Modular Specificationof OSI Subsystems:

Specifikacije s pomočjo Petrijevih mrež so redkejše, vsi pa jim priznavajo kompletnejše predstavitvene možnosti, predsvem zaradi enostavnega opazovanja časovnih parametrov in sinhronizacijskih postopkov v protokolarni specifikaciji.

Osnovni članek s tega področja podaja bistvene probleme sintakse za specifikacijo in rešitev v zvezi s sinhronizacijo oddajnika in sprejemnika:"E. ECKART, P. PRINOTH: Automated Proofing of Communication Protocosls Against Communication Services, Proceedings of the Sixt International Conference on Computer Communication".

Vmesni članek med specifikacijo in proučevanjem problemov s pomočjo posebnih programskih jezikov je delo: "J. M. AYACKE, J. P. COURTIAT, LC/1: A Specification and Implementation Language for Protocols".

Določena grupa člankov se ukvarja s specifikacijskimi problemi v okviru posebnih programskih jezikov.

Osnovna referenca, ki detalnejše opisuje in sistematizira področje med opisom protokola in specifikacijo je:"J.P. ANSART: Protocols, Standards, Specification, Implementation, Testing". Podaja uvod v področje specifikacije s pomočjo algebrajskih jezikov in procesne algebre. Take rešitve so primerne, zlasti ker:- zagotavljajo korektno in enoumno specifikacijo, ki omogoča enostavno sintaktično in semantično kontrolo. Teorija nudi orodja, ki omogočajo enostavnejše strukturiranje problema ter obvlada probleme nedeterminiranosti protokola tudi v distribuiranem okolju.

Podoben članek je:"Ed BRINKSMA: An Algebraic Language for the pecification of the Temporla Order of Events in Services and Protocols".

Posebno pozornost zasluži članek, ki opisuje formalen pristop preverjanja protokolov in išče določene možnosti v okvirju hardwarske implementacije le-teh:"Miguel G. HOFFMAN: Hardware Implementation of Communication Protocols: A Formal Appeoach, Proceedings of the 7th Annual Symposium of Computer Architecture ".

Sledeči članek uporablja podobne definicije kot so navedene v prvem s tem, da nakazuje rešitve s pomočjo specifikacijskega jezika in posebnega algoritma preverjanja (CCS - Calculus of Comunicatio Systems):"A.F. SCOLLO: SDL and CCS Based Description of Communication Entities".

Teoretična osnova za poljubno specifikacijo v okvirju formalnih zapisov, ki vsebuje primer formalizacije fragmenta Teletex-ovega servisa, je članek: "W. ORTH: Formulizel Definition and Analysing of Protocols".

Že v dosedaj naštetih delih so vsebovani postopki, ki opisujejo testiranje protokolov, ki so vključeni v celovito rešitev problema specifikacija - implementacija. Obstaja pa nekaj del, ki se poudarjeno ukvarjajo samo s testiranjem."J.P. ANSART, STQ, CERBERE and GENEPI: Three tools for Implementation Testing". Delo podaja kriterije testiranja in opisuje tri sisteme za testiranje. To so izredno dragi produkti. Za njihov razvoj so bila vložena znatna sredstva.

Dva podobna članka, ki se ukvarjata s specifikacijo standarda s poudarkom na testiranju protokolov v okvirju OSI referenčnega modela, sta:"D. RYNER: Protocol Implementation Assessment, National Physical Laboratory, UK, Proceedings of the Sixth International Conference on Computer Communication","R.W.S. HALL and D. RYNER: Protocol Product Testing - Some Comparisons".

Posebno zanimivo in pregledno je delo, ki opisuje praktične rezultate v laboratorijih

in nižjih komunikacijskih nivojev, kakor jih zaznava testirani komunikacijski nivo.

Definiran je postopek, katerega rezultat sta vmesnika med dvema komunikacijskima nivojema. Za izhodišče je uporabljena struktura avtomata prenosnega medija, dobljenega po paralelno-serijski kompoziciji avtomatne mreže modela. Ta postopek je pomemben rezultat, saj predstavlja avtomatiziran korak k snovanju izvedbeno odvisne strukture komunikacijskega sistema.

Opisana problematika v našem okolju še vedno nima pravega mesta, ne glede na perspektivni razvoj računalniških, komunikacijskih, teleinformacijskih in porazdeljenih sistemov. Metoda je uporabna za analizo obstoječe komunikacijske opreme, prav tako pa lahko predstavlja osnovo za avtomatizirano načrtovanje protokolarnih predpisov oziroma celovite komunikacijske opreme. V določenih razvojnih delovnih okoljih bistveno pripomore h korektnosti in učinkovitosti pri premagovanju problemov, ki se pojavljajo med opisom in izvedbo komunikacijskih sistemov.

Sam pristop, kot tudi tehnologija testiranja in načrtovanje komunikacijskih sistemov s pomočjo LPA-ja, so bili verificirani na tako kompleksnem sistemu, kot je telefonska centrala ISKRA-2000. Z LPA-jem je bil testiran tudi "obroč z žetonom", razvit na Institutu Josef Stefan, kjer je bilo testiranje opravljeno s strani samih izvajalcev. To pomeni, da ima LPA že sedaj dovolj prijazen uporabniški vmesnik in, da je vsebina testiranja za uporabnike transparentna, kar jim omogoča, da se posvetijo reševanjem izvedbenih, ne pa logičnih problemov protokola. Na osnovi analize sistemov z LPA-jem so bile razložene določene lastnosti in pojavi v eksploataciji sistemov, ki si jih razvijalci in vzdrževalci sistemov poprej niso znali dobro pojasniti.

V nadaljevanju dela bomo LPA iz laboratorijskih okvirov postavili razvijalcu na delovno mesto s čimer bo LPA postal uporabno orodje s podobnimi efekti kot jih imajo klasični logični analizatorji,brez katerih danes marsikateri projekt ni več izvedljiv.

ODREDJIVANJE VRSTE REČI IZ SRPSKOHRVATSKOG JEZIKA PRIMENOM RAČUNARA

INFORMATICA 1/88

UDK 681.3.06:808.61

Mihajlović Dragan
INDOK-Služba Skupštine SAPV

Odredjivanje vrste reči iz srpskohrvatskog jezika primenom računara ima značaja za automatsko indeksiranje tekstualnih dokumenata, odabir leksike za sastavljanje tezaurusa i druga lingvistička istraživanja. Osnovni princip odredjivanja vrste reči predstavlja činjenica da reči sa jednakom kombinacijom slova na svom kraju pripadaju po pravilu istoj vrsti reči. U radu se razmatra način odredjivanja vrste reči iz srpskohrvatskog jezika na osnovu kombinacije slova na kraju reči i zadatog rečnika. Prikazani su eksperimentalno dobijeni rezultati.

COMPUTER AIDED WORD TYPE DETERMINATION IN SERBO-CROATION is of importance in the process of automatic indexing of textual documents, corpus of words creation in the proces of thesaurus determination and lingvistics research. Word type determination is based on the fact that words with identical combination of letters ending them belong, as a rule, to the same word type. The paper discusses the method of word type determination in serbo-croation based on letter combinations ending the words and the given vocabulary. Results of experiments are also shown.

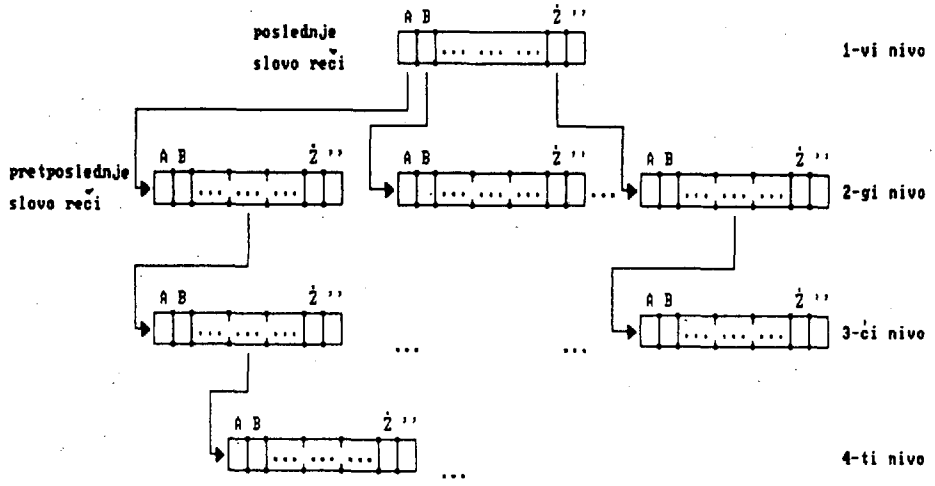
1. UVOD

Odredjivanje vrste reči primenom računara, posebno prepoznavanje imenica i prideva iz teksta na prirodnom jeziku ima značaja u automatskom indeksiranju, odabiru leksike za sastavljanje tezaurusa i druga lingvistička istraživanja. Osnovni princip odredjivanja vrste reči predstavlja činjenica da reči sa jednakom kombinacijom slova od kraja prema početku reči pripadaju po pravilu istoj vrsti reči. Na primer, reči SAGLASNOST, SIGURNOST i ZAVISNOST imaju istu kombinaciju slova od kraja prema početku reči, NOST i pripadaju imenicama. Na osnovu navedenog principa mogu se graditi različiti sistemi rečničkog ili tabelarnog tipa. Kada se odredi broj slova na osnovu kojeg će se izvršiti odredjivanje vrste reči, na osnovu dovoljno ulaznih podataka (reči sa ručno odredjenom vrstom reči) za svaku prisutnu kombinaciju slova od kraja prema početku reči u rečnik ili tabelu upisuje se kombinacija slova i oznaka vrste reči. Odredjivanje vrste zadate reči svodi se na pronalaženju u rečniku ili tabeli kombinacije slova koja je ista sa kmbinacijom slova od kraja prema početku zadate reči i da se vrsta reči pridružena toj kombinaciji u rečniku ili tabeli uzme za vrsu zadate reči. Očigledno je da se na osnovu jednog slova sa kraja reči ne može vršiti upotrebljivo prepoznavanje imenica i prideva. Tačan odgovor na to, koji je minimalan broj slova iz kraja reči dovoljan za odredjivanje vrste reči ne može se dati. Može se postaviti sledeća tvrdnja: što je broj slova na osnovu kojeg se vrši odredjivanje vrste reči veći to je i rezultat pouzdaniji. Za razliku od metoda sa fiksnim brojem slova uzetih od kraja prema početku reči koje su primenjene u ruskom jeziku /1/, za odredjivanje vrste reči u srpskohrvatskom jeziku u ovom radu izložen je metod korišćenja rečnika sa mogućnošću izjednačavanja promenljivog broja slova od kraja prema početku reči.

U rečniku je za svaku reč ručno utvrdjena njena vrsta. Izbor reči za formiranje rečnika može se izvršiti pomoću računara tako da se dobiju što raznovrsniji završeci po svim vrstama reči. Sastav rečnika treba da obezbedi odredjivanje vrste reči na osnovu minimalno dva slova od kraja prema početku reči. Ovaj uslov može se koristiti kao kriterij izbora novih reči koje će se dodavati u rečnik.

2. ODREDJIVANJE VRSTE REČI U SRPSKOHRVATSKOM JEZIKU

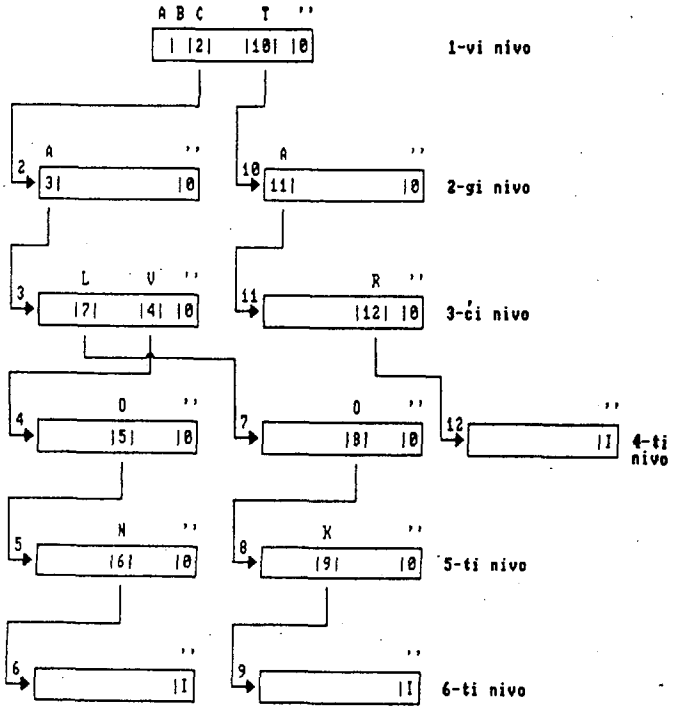
Odredjivanje vrste zadate reči sastoji se u tome da se u rečniku pronadje reč koja se sa zadatom slaže sa maksimalnim brojem slova od kraja prema početku reči i njena vrsta uzme za vrstu zadate reči. Ukoliko se reč čiju vrstu odredjujemo nalazi u rečniku tada je potrebno uzeti pridruženu oznaku vrste reči iz rečnika. Kada se reč čiju vrstu odredjujemo ne nalzi u rečniku tada se za njenu vrstu uzima vrsta one reči iz rečnika koja se sa njom slaže u najvećem broju slova od kraja prema početku reči. Interesantno je razmotriti pogodnu organizaciju rečnika za brzo odredjivanje vrste nepoznate reči. Na slici 1. data je logička struktura zapisa rečnika. Postoje samo oni zapisi koji su neophodni za organizovanje rečnika. Zapis i-vog nivoa odnosi se na prva slova invertovanih reči (poslednje slovo reči), zapisi



Slika 1. Logička struktura zapisa rečnika

2-gog nivoa odnose se na drugo slovo invertovane reči (pretposlednje slovo reči) itd. Maksimaln broj spregnutih zapisa za jedan je veći od broja slova u najdužoj reči.

U svakom zapisu postoji 30 polja za slova A,B,...Z,Ž i jedno polje (sa adresom 31) za blanko. Polja u zapisu sadrže adresu zapisa sledećeg slova invertovane reči. Ako je sadržaj k-tog polja zapisa na nivou n nula to znači da ne postoji reč čije n-to slovo od kraja prema početku reči daje adresu polja k. Polje blanko znaka u zapisu sadrži oznaku vrste reči koja nas dovodi do tog polja. Raspored polja u zapisu može se pogodno odabrati tako da se nekom funkcijom numerički kod slova transformiše u adresu polja.



Slika 2. Primer zapisa rečnika

Na slici 2. ilustrovani su zapisi rečnika nakon toga što su upisane sledeće imenice: NOVAC, KOLAC, RAT. Na primer, od reči NOVAC invertovana reč je CAVON. Zapis na 1-om nivou u polju C sadrži adresu zapisa na 2-gom nivou. Zapis na 2-gom nivou u polju A sadrži adresu zapisa na 3-ćem nivou itd. Princip odredjivanja vrste reči je praćenje spregnutih zapisa u rečniku dok je to moguće na osnovu slova invertovane reči čiju vrstu odredjujemo. Ako se reč čiju vrstu odredjujemo nalazi u rečniku tada se praćenje spregnutih zapisa izvodi do blanko znaka i čitanja informacije o vrsti reči koju smo toj reči ručno pridružili. Kada se praćenje spregnutih zapisa ne može više izvoditi na osnovu slova invertovane reči čiju vrstu odredjujemo može postojati više načina za nastavak algoritma. Jedan od načina je nastavak prolaženja spregnutim zapisima tako da to bude najkraći put preko prvog mogućeg polja zapisa. Informacija u prvom zapisu sa sadržajem polja(31) različitim od nule pridružuje se kao vrsta reči za analiziranu reč. Nije istraživano da li neki drugi način nastavka prolaženja spregnutim zapisima daje bolje rezultate.

Odredjivanje vrste reči UDOVAC pomoću rečnika ilustrovanog na slici 2 izvodi se preko zapisa sa adresama 1, 2, 3, 4, 5, 6. U zapisu sa adresom 6 u polju 31 (blanko) pročitamo vrstu reči I-imenica.

3. DOBIJENI REZULTATI

Za proveru opisanog načina odredjivanja vrste reči i to imenica, prideva, glagola i brojeva formiran je rečnik od 2320 imenica, 1530 prideva, 550 glagola i 50 brojeva. Pored ovog rečnika formiran je i rečnik od 978 reči (neinformativne reči:prilozi, predlozi, zamenice, veznici, rečce i uzvici) čiju vrstu odredjujemo na osnovu konsultovanja tog rečnika. Formirana su dva rečnika da bi se sprečilo u prvom redu eventualno pogrešno odredjivanje vrste reči koja je imenica, pridev, glagol ili broj kao prilog, predlog, zamenica, veznik, izvik ili rečca. Prilikom odredjivanja vrste nepoznate reči prvo se proverava da li je reč sadržana u neiformativnom rečniku. Vrsta reči koja je sadržana u neinformativnom rečniku odredjuje se iz tog rečnika. Vrsta reči koja nije sadržana u neinformativnom rečniku odredjuje se poredjenjem slova od kraja prema početku reči.

Tabela 1.

VRSTA REČI	ALGORITMOM ODREDJEN OBLIK		POGREŠNO ODREDJENIH OBLIKA
	DIREKTNO U REČNIKU	NA OSNOVU ZAVRŠETAKA	
imenice	923	1193	43
pridevi	416	847	19
glagoli	160	264	8
brojevi	19	4	1
UKUPNO	1518	2308	71

Vrsta neinformativne reči koja nije sadržana u neinformativnom rečniku odredjuje se pogrešno. Analiziran je prirodan tekst od 19200 reči. Od toga 1430 reči su neinformativne,a 17770 reči su informativne. Različitih oblika informativnih reči bilo je 3826. Rezultati odredjivanja vrste reči za 3826 oblika informativnih reči ilustrovani su u tabeli 1.

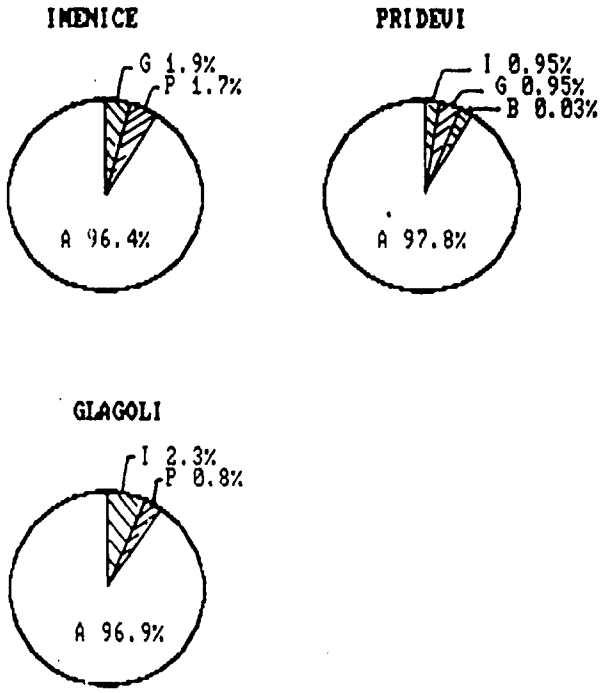
Od 3826 različitih oblika informativnih reči direktno u rečniku utvrdjena je vrsta reči za 1518 reči. Za preostalih 2308 različitih oblika informativnih reči, vrsta je odredjena na osnovu slova od kraja prema početku reči.

Stepen tačnosti odredjivanja vrste 2308 različitih oblika informativnih reči ilustrovan je na slici 3.

Oznake na slici 3. imaju sledeća značenja:

A - pravilno odredjena vrsta reči,
I - reči koje su nepravilno svrstane u imenice,
P - reči koje su nepravilno svrstane u prideve,
G - reči koje su nepravilno svrstane u glagole,
B - reči koje su nepravilno svrstane u brojeve.

Na primer, vrsta imenica pravilno je odredjena u 96.4% slučajeva. Rečima koje su imenice, u 1.9% slučajeva odredjena je vrsta glagola i u 1.7% slučajeva vrsta prideva.



Slika 3. Tačnost odredjivanja vrste reči

4. ZAKLJUČAK

Pokazano je da se vrsta reči u srpskohrvatskom jeziku može odrediti sa velikom tačnošću primenom računara na osnovu slova od kraja prema početku analizirane reči i poznatog rečnika. Povećanje tačnosti pravilnog odredjivanja vrste reči može se izvršiti proširivanjem poznatog rečnika kao i primenom dodatnih kriterija. Kao dodatni kriterij za odredjivanje vrste reči iz prirodnog teksta može biti poznat redosled vrsta reči u rečenici.

LITERATURA:

/1/ Vinjakova S.M.
Vydelenie suščestvitel'nyh i prilagatel'nyh pri avtomatičeskom analize teksta,
NTI, ser 2., br. 3., pp. 15-18, 1976.

UDK 681.3:655

Marija Šifrar
Univerza v Mariboru, Računalniški center

Prispevek govori na splošno o strategiji in tehnologiji pri načrtovanju ustrezne računalniške infrastrukture kot osnove za delovanje knjižničnega informacijskega sistema ter sistema znanstvenega in tehničnega informiranja. Podan je splošen opis in kriteriji, ki jih mora potrebna računalniška strojna, programska in komunikacijska oprema izpolnjevati.

This paper discusses strategies and technologies for developing infrastructure required to support a functionally unified bibliographic information system. Some general descriptions and specifications for particular components such as hardware, software and communications are considered.

Prehod na nov (avtomatiziran) način poslovanja knjižnic in specializiranih informacijskih centrov zahteva izgradnjo odnosno dostopnost ustrezne infrastrukture. Infrastrukturo predstavljajo poleg podatkovnih resursov (baz podatkov) tudi potrebna strojna računalniška, komunikacijska in programska oprema, ki omogoča gradnjo, vzdrževanje, manipulacijo, shranjevanje, iskanje in posredovanje informacij.

V svetu in pri nas število podatkovnih baz nenehno narašča. Pri načrtovanju infrastrukture je potrebno nameniti posebno skrb izgradnji novih in omogočiti učinkovit dostop do obstoječih podatkovnih zbirk tako doma kot v tujini.

Programska, strojna računalniška in komunikacijska oprema morajo omogočati postopen prehod k avtomatizaciji enotnega knjižničnega informacijskega sistema ter sistema znanstvenega in tehničnega informiranja.

Oprema mora biti takšna, da jo je možno uporabljati za različna opravila knjižnično informacijske dejavnosti in dejavnosti specializiranih informacijskih centrov npr. za online katalogizacijo, medknjižnično izposojo, poizvedovanje v domačih in tujih bibliografskih bazah podatkov različnih ponudnikov, naročanje in nabavo gradiva, izposojo, finančno poslovanje itd.

STROJNA RAČUNALNIŠKA OPREMA

Za razvoj knjižničnih informacijskih sistemov kot tudi sistema znanstvenega in tehničnega informiranja je ustrezna računalniška podprtost izredno pomembna. Najboljši uporabniški vmesnik izgubi na vrednosti ob nezanesljivi računalniški podpori in slabih odzivnih časih. Kompliciran in neprijazen postopek komuniciranja odvrne uporabnika od uporabe direktnega (online) sistema.

Pri planiranju razporeditve računalniških kapacitet je treba upoštevati lokacijo posameznih podatkovnih resursov ter dostopnost in način posredovanja uslug končnim uporabnikom.

Za delovanje knjižničnega informacijskega sistema ter sistema znanstvenega in tehničnega informiranja se lahko uporablja računalniška strojna oprema različnih zmogljivosti in sicer veliki računalniški sistemi ter manjši podsistemi običajno v obliki mini- in mikro računalnikov ter terminalov. Ključnega pomena pa je razvit komunikacijski sistem, ki omogoča, da imajo uporabniki preko svojega terminala ali mikroračunalnika dostop do vseh baz podatkov v mreži.

Zmogljivejši računalniški sistemi

Osnovno računalniško strojno opremo za uspešno delovanje knjižničnega informacijskega sistema ter sistema znanstvenega in tehničnega informiranja predstavlja zmogljivejši centralni sistem, na katerega je možno ob ustrezni komunikacijski strojni in programski opremi priključiti zadostno število terminalov ali podsistemov (neinteligentni-, inteligentni terminali, mini- in mikroračunalniki).

Velik zmogljivejši računalniški sistem, mora omogočati vnos in vzdrževanje skupne bibliografske baze podatkov ter dostop, iskanje in posredovanje podatkov. Na centralnem sistemu je potrebno hraniti baze podatkov osnovnih bibliografskih zapisov kot tudi ustreznih pomožnih datotek indeksov za iskanje podatkov. Na večjem sistemu je smiselno

vzdrževanje različnih baz podatkov specializiranih informacijskih centrov. Podatke je smiselno hraniti na velikem sistemu iz več razlogov:

1. Količina podatkov (večje diskovne kapacitete ipd.).
   Programsko opremo za vzdrževanje baze podatkov je tako enostavneje vzdrževati, saj se vse spremembe in dopolnitve vnesejo na enem mestu.

3. Tudi vse spremljajoče datoteke t.i. indeksne datoteke za potrebe iskanja, ki se neprestano spreminjajo, je zaradi lažjega vzdrževanja potrebno hraniti za vse uporabnike na eni lokaciji (na enem sistemu).

4. Možnost komuniciranja oziroma povezovanja z ostalimi velikimi računalniškimi sistemi in podsistemi.

Podsistemi

Kot podsistem lahko služijo prav tako računalniški sistemi različnih zmogljivosti, včasih pa so to samo običajni terminali. Na izbiro ustrezne opreme vpliva veliko faktorjev. Uporaba mikroračunalnikov kot podsistema pa je zelo pogosta in velikokrat tudi ekonomična.

Mikroračunalnike lahko uporabljamo tako, da delujejo samostojno ali pa kot terminali na centralnem računalniškem sistemu, kar je odvisno od vrste opravila in od same organiziranosti celotnega sistema. Samostojno jih je možno uporabljati za postopek obdelave bibliografskih podatkov (katalogizacijo) t.j. vnosa, spreminjanja in brisanja podatkov. Za vzajemno online katalogizacijo, kjer več institucij združeno vzdržuje skupno bibliografsko bazo podatkov, ki se hrani na centralnem računalniškem sistemu, so primerni mikroračunalniki z vgrajenim terminalskim emulatorjem. Pri distribuiranem načinu vnosa podatkov, ki prehaja vse bolj v rabo, je uporaba mikroračunalnikov za ta opravila, ekonomična. Na ta način uporabniki manj obremenjujejo komunikacijsko linijo in centralni procesor. Na glavni računalnik se priključijo le kadar iščejo zapise ali pa jih vnašajo v centralno bibliografsko bazo podatkov. Istočasno takšen način obdelave podatkov pripomore k izboljšanju odzivnega časa pri iskanju v bibliografski bazi podatkov ter k zmanjšanju stroškov obdelave bibliografskega gradiva, saj tako porabijo manj procesorskega časa in nižji so tudi stroški komunikacij. Prihranki so občutni še zlasti tedaj, ko morajo knjižnice in specializirani informacijski centri vršiti katalogizacijo na najeti računalniški opremi.

Ustrezni podsistemi (npr. mini- ali mikroračunalniki) se lahko uporabljajo tudi za opravljanje ostalih funkcij knjižničnega informacijskega sistema ter sistema znanstvenega in tehničnega informiranja, ki so lokalnega značaja npr. izposoja, naročanje in nabava bibliografskega gradiva, finančno in knjigovodsko poslovanje itd.

Mikroračunalniki so primerni tudi kot orodje za iskanje podatkov v bazah podatkov različnih ponudnikov pri nas in v tujini, hranjenih na velikih sistemih. Zato potrebujemo posebno komunikacijsko programsko opremo, ki omogoča komuniciranje mikroračunalnika s programskim sistemom na velikem računalniku, ki manipulira z bazo podatkov. Prednosti uporabe lokalne programske opreme za iskanje v komercialnih bazah podatkov je zmanjšanje stroškov iskanja ter možnost nadaljne lokalne obdelave zadetkov.

Uporaba različnih ASCII terminalov in (mikro)računalnikov pripomore h kompleksnosti komunikacijskega sistema. Zaradi raznolikosti terminalske opreme različnih proizvajalcev mora biti ustrezno prirejena komunikacijska programska oprema. Sistem mora poznati npr. število znakov v vrstici, število vrstic na ekranu, za vsak tip terminala sekvenco za brisanje ekrana itd. Za priključitev številne ASCII terminalske opreme je potrebno poskrbeti za ustrezni način preklapljanja med sistemi, izbiro optimalne podatkovne poti in vključitev v lokalne računalniške mreže.

Poleg izbire tipa terminalov, je treba upoštevati še druge kriterije. Procedure za vključitev v sistem in dialogi z računalnikom morajo biti čimbolj preprosti. Sporočila sistema morajo biti enostavna in takšna, da pomagajo uporabniku. Sistem mora biti neobčutljiv na aktivnosti na oddaljenih lokacijah. Iz tega razloga niso priporočljive verige povezanih terminalov, kjer motnja kjerkoli v verigi povzroči izpad vseh terminalov.

PROGRAMSKA OPREMA

Programska oprema za opravljanje osnovne knjižnične ter referalne dejavnosti se deli v funkcionalnem smislu na:

- programsko opremo za vnos in vzdrževanje (ažuriranje) bibliografskih baz podatkov (on-line katalog) in povezuje fazo naročanja, to je delno obdelavo bibliografskega gradiva ter končno obdelavo (katalogizacijo, klasifikacijo oziroma indeksiranje) prispelega bibliografskega gradiva,

- programsko opremo za iskanje, posredovanje in izkazovanje podatkov v bibliografskih bazah podatkov posameznih knjižnic ter bazah podatkov ostalih velikih ponudnikov baz podatkov (informacijskih servisov),

- dodatne programske module npr. za izposojo in medknjižnično izposojo, finančno poslovanje, itd.

Osnovni kriterij, ki ga mora pogramska oprema izpolnjevati je modularnost. Sistem mora biti zastavljen in zgrajen tako, da ga je možno postopoma dograjevati, dodajati posamezne funkcije z namenom, da se ugodi vsem potrebam in željam osebja pri opravljanju vsakodnevnih nalog in končnih uporabnikov pri koriščenju uslug knjižnic.

Zahteve, ki jih mora programski sistem izpolnjevati:

- povezovati mora module programirane za "host" računalnik in module za mikroračunalnike,

- istočasno ga lahko uporablja več uporabnikov (multi-user),

- mora biti uporabniško prijazen, to pomeni prilagojen uporabi začetnikov ter bolj izkušenih uporabnikov on-line sistemov,

V svetu obstaja veliko komercialnih programskih paketov, veliko institucij pa je za svoje potrebe izdelalo lastne programske sisteme. Običajno so programskimi sistemi, ki so jih razvile posamezne institucije za lastne potrebe bolj konverzacijsko orientirani medtem, ko so tipični komercialni programski sistemi transakcijsko orientirani. Konverzacijsko orientirani sistemi dopuščajo odprt dialog.

Veliko teksta v obliki sporočil posreduje programski sistem. Pogoj za vzpostavitev uporabniško prijaznega sistema, je vzpostavitev zelo intenzivnega konverzacijskega modula.

Pri nas so razviti programski produkti, ki pokrivajo le posamezne funkcije. Nastajali so iz zahtev, ki so jih narekovale trenutne potrebe in ne temeljijo na enotnih merilih.

Komunikacijska programska oprema je za uspešno delovanje knjižnično informacijskega sistema in sistema znanstvenega in tehničnega informiranja ključnega pomena. Vse podatke, ki jih uporabnik vtipka na svojem terminalu, mora pravilno posredovati aplikativnemu programu (npr. on-line katalogu na host računalniku) in poskrbi tudi za to, da programski sistem vrne odgovor ustreznemu terminalu. Teoretično bi za to lahko poskrbela aplikativna programska oprema za opravljanje ostalih funkcij knjižničnega poslovanja. To bi zahtevalo zelo tesno interakcijo z operacijskim sistemom in računalniško strojno opremo, kar je zelo težko doseči pri uporabi višjih programskih jezikov, v katerih je običajno napisana aplikativna programska oprema. V tem primeru je pisanje programske opreme bolj zahtevno in razvoj opreme terja več časa in kadra.

MREŽE ZA PRENOS PODATKOV

Planirati je treba sistem, ki bo omogočil postopno priključitev velikega števila terminalske opreme. Tradicionalne telefonske zveze nudijo omejeno število podatkovnih poti na linijo. To narekuje potrebo po povečanju števila telefonskih linij na eni lokaciji in dodatno strojno opremo, ki kontrolira pretok podatkov na obeh koncih, kar pomeni občutno povečanje stroškov in vzpostavljanje redundantnih podatkovnih poti.

- Mreža v kateri je povezanih preko tisoč terminalov mora imeti številne alternativne podatkovne poti, ki potekajo preko različnih medijev za prenos podatkov z vgrajeno zmožnostjo avtomatskega odkrivanja motenj oziroma napak ter ponovne vzpostavitve poti. Pri načrtovanju produkcijske mreže je poudarek na zanesljivosti, operativnosti, možnosti razširitve in ekonomičnost.

- Mediji za prenos podatkov morajo biti širokopasovni, zanesljivi in ekonomični. Sateliti in terestrični mikrovalovi so zanimivi mediji za daljinske povezave. Za lokalni prenos podatkov npr. na območju kompleksa univerze, so poleg tradicionalnih telefonskih zank, direktnih kabelskih povezav (lokalna računalniška mreža) primerni tudi radijski in visokofrekvenčni (21-22 gigahertz) mikrovalovi.

- Zahteva po zanesljivosti in možnosti razširitve narekuje implementacijo tehnike paketnega preklapljanja podatkov, kjer so podatki združeni v pakete. Takšen sistem je tudi JUPAK.

  Paket je podatkovna enota, ki se prenaša po mreži (v tem primeru so to informacije, ki jih on-line katalog (programski sistem) pošilja uporabniku ali pa so to odgovori, ki jih uporabnik tipka na terminalu). Od pošiljatelja do prejemnika jih vodi in kontrolira pot ustrezna strojna in programska oprema. Vsak paket je opremljen z adreso pošiljatelja in naslovnika. Z dodano informacijo o naslovu pošiljatelja in prejemnika paketov zadostujejo za priključitev več terminalov na procesor le ena vrata. Mreža s paketnim preklapljanjem zahteva uporabo ustrezne strojne in programske opreme ter implementacijo protokolov, ki omogočajo izbiro optimalne poti, kontrolo nad potekom prenosa podatkov, odpravo napak oziroma motenj pri prenosu podatkov ter odkritje in izognitev neoperativnih poti.

- Protokoli služijo kot specifikacije za interno delovanje računalniške mreže ter delovanje eksternih vmesnikov do te mreže. Protokoli opredeljujejo definicije funkcij, ki jih morajo posamezne komponente računalniške mreže opraviti. ISO referenčni model predvideva sedem nivojev protokolov za medsebojno sporazumevanje med različnimi računalniškimi sistemi. Najvišji nivo je aplikacijski nivo, ki je še vedno predmet dogovarjanj in razprav.

APLIKACIJSKI PROTOKOLI
(za potrebe knjižničnega informacijskega sistema ter sistema znanstvenega in tehničnega informiranja)

Ker je protokol standard, lahko vsak sistem znotraj računalniških mrež komunicira s katerimkoli drugim sistemom znotraj mrež, ki podpirajo isti protokol. To dopušča lociranje funkcij tam, kjer je to najbolj primerno in sicer v tehnološkem pogledu (zmogljivost računalniških sistemov, stroški telekomunikacij) in z vidika upravljanja računalniških mrež (kateri računalniški sistem, kdo upravlja z računalniškim sistemom, itd.).

V ZDA razvrščajo aplikacijske protokole za te namene v tri splošne razrede:

- protokoli za iskanje informacij v bazah podatkov
  (za sporazumevanje med računalniškimi sistemi različnih knjižnic, ter drugih posrednikov informacij ter med posredniki teh informacij in uporabniki),

- protokoli za poslovanje
  (za izmenjavo podatkov med knjižnicami, informacijskimi centri ter ponudniki baz podatkov),

- protokoli za operativno delovanje aplikacij
  (služijo za povezovanje različnih aplikacij s področja avtomatizacije poslovanja knjižnic ter specializiranih informacijskih centrov).

Protokol za iskahje informacij predpisuje način iskanja in izmenjavo zapisov med računalniki. Pri tem velja poudariti naslednje:
Uspešna določitev protokolov za iskanje in posredovanje informacij je odvisna od implementacije splošno veljavnega standardiziranega (statičnega) opisa datoteke. Za knjige je ta zahteva izpolnjena z MARC (MAchine Readable Cataloguing) standardi. To je standardiziran bibliografski opis knjige za izmenjavo zapisov v računalniško čitljivi obliki. Za druge vrste dokumentov npr. elektronski dokumenti (članki in knjige), ki vsebujejo tekst v celoti (izvlečki, poročila, podatki o izposji itd.) to trenutno še ni rešeno.

Poleg protokolov za iskanje in posredovanje informacij je treba razviti še številne aplikacijske protokole z vidika knjižničnega poslovanja. Priprave za standardizacijo postopka naročanja in nabave knjig in serijskih publikacij so že v teku. Nekateri postopki kot

npr. fakturiranje, pa se rešuje na splošni ravni skupaj s poslovnimi aplikacijami. Protokoli za servise kot je elektronska pošta pa so bili razviti skupaj s prizadevanji za standardizacijo na področju računalniškega omreževanja.

Protokoli so potrebni tudi za standardizacijo operativnega poslovanja knjižnične dejavnosti in informacijskih centrov. To še posebaj velja za izposojo. V primeru, ko je sistem za izposojo direktno povezan z on-line katalogom knjižnice, je potrebno priskrbeti sredstava za ugotavljanje statusa bibliografske enote, podaljšanje in vračila ter izvedbe dejanske transakcije izposoje, katere rezultat je knjiga, ki se pošje s pošto.

V zvezi z načrtovanjem protokolov za izposojo je potrebno rešiti še vrsto nejasnosti glede zaščite, legalizacije, določitve privilegijev ter pri postopku povezovanja podatkov o knjigi v on-line katalogu in podatkov v datoteki podatkov potrebnih za izposojo.

Vpliv aplikacijskih protokolov

Implementacija aplikacijskih protokolov popolnoma eliminira vprašanje standardizacije poizvedovalnega jezika za on-line kataloge in podobne sisteme. Vendar je o standardizaciji na tem področju še prezgodaj govoriti, ker je razvijanje uporabniških vmesnikov še vedno bolj umetnost kot pa inženiring. Orodja in izrazoslovje te vrste še niso dokončno definirana. Če bodo protokoli implementirani, bo lahko uporabnik komuniciral s katerimkoli on-line katalogom (ali več on-line katalogi) z uporabo kakršnegakoli poizvedovalnega jezika, ki je instaliran skupaj z lokalnim on-line katalogom ali pa na osebnem računalniku. Nikoli ni potrebno poznati ukazov več kot enega poizvedovalnega jezika. Lokalni računalniški sistem (osebni računalnik ali on-line katalog na host) na katerega se uporabnik prijavi komunicira na uporabnikovo zahtevo z vsemi oddaljenimi sistemi, prevede neopazno za uporabnika lokalni poizvedovalni jezik v skupen splošno poznan iskalni jezik aplikacijskega protokola.

Ker prenaša protokol zapise v predpisanem, splošno poznanem formatu, je možno na ta način izpeljati tudi številne dodatne postopke. Na host sistemu se lahko instalirajo posebne rutine za odločitev o tem v katerem računalniškem sistemu bo iskal želeno informacijo in v kakšnem vrstnem redu. Zadetki dobljeni z več oddaljenih sistemov se primerjajo med seboj in združijo. Na ta način se odpravijo nepotrebni duplikati. Smiselno je implementirati program za ugotavljanje najcenejše poti za dostop do ustreznih podatkovnih baz. Princip je podoben kot pri vzpostavljanju medkrajevnih telefonskih razgovorov, kjer se upošteva dnevni čas, vrsta baze podatkov, cenik, popusti na količino ipd. Programski sistem na host bo lahko vzpostavil zvezo s številnimi oddaljenimi servisi za posredovanje baz podatkov, da bi ugodil uporabnikovi zahtevi.

INTEGRIRANI ALI MREŽNI SISTEMI

Integrirani sistemi so zaprti sistemi. Mrežni sistemi so odprti, neskončno razširljivi. V razvitem svetu je večina računalniških mrež medsebojno povezanih. Povezati se je možno s tisočerimi računalniki. Seveda je treba poznati proceduro za vzpostavitev zveze. Na ta način bodo v svetu kmalu povezane vse večje knjižnice in informacijski centri na aplikacijskem nivoju.

Obstaja možnost, da institucije na lokalnem nivoju vzpostavljajo integrirane sisteme, ki predstavljajo vozlišča v celotni mreži. Vendar je tudi na lokalnem nivoju smotrno uporabiti mrežni pristop in opravljati razne funkcije na različnih računalniških sistemih.

DOSEDANJE IZKUŠNJE V SLOVENIJI

Knjižnice v Sloveniji razpolagajo s različno programsko opremo (vnos in ažuriranje kataloga knjig po polnem in skrajšanem bibliografskem opisu; vnos, ažuriranje, iskanje in izkazovanje podatkov v bazah specializiranih informacijskih centrov, izposoja knjig, iskanje gradiva v online katalogu, iskanje podatkov v drugih online sistemih, itd.), ki ni grajena po enotnih merilih. Baze podatkov, ki se vzdržujejo, ne uporabljajo enotnega formata zapisa.

Trenutno imajo le nekatere knjižnice in informacijski centri skromno strojno opremo in sicer različne tipe mikroračunalnikov in terminalov. Kot host sistemi se koristijo računalniški sistemi RRC, RCUL in RCUM.

Dosedanje izkušnje so pokazale, da bo v prihodnje potrebna širša in bolj usklajena akcija za izboljšanje situacije na področju avtomatizacije knjižničnega poslovanja ter sistema znanstvenega in tehničnega informiranja. Pri tem bi moralo sodelovati čim več knjižnic in ostalih institucij. Z ozirom na knjižnični fond, razpoložljiva finančna sredstva ter glede na število in strokovno usposobljenost delavcev v posameznih institucijah, bi bilo smotrno združiti sredstva in napore tako za vsklajeno nabavo strojne računalniške in komunikacijske opreme ter za razvoj programske opreme.

V naslednjem obdobju bi bilo potrebno povezati knjižnice in informacijske centre v enoten knjižnično-informacijski sistem in omogočiti njihovo vključitev v svetovno izmenjavo znanstveno tehničnih informacij.

VAL REALIZACIJA IZRAČUNAVANJA N-TOG KORENA KORIŠĆENJEM PARALELNOG ITERATIVNOG ALGORITMA
VAL REALIZATION OF N-TH ROOT EVALUATION USING PARALLEL ITERATIVE ALGORITHM

INFORMATICA 1/88

Slavoljub Damjanović
HE »Đerdap«, Kladovo
Lazar Đorđević, dr.
Elektronski fakultet, Niš

UDK 681.3.06:519.6

SADRŽAJ: U ovom radu prikazana je programska realizacija izračunavanja n-tog korena u višem dataflow jeziku VAL, primenom paralelnog iterativnog algoritma. Izložene su osnovne karakteristike ovog jezika, koji po semantici pripada klasi funkcionalnih jezika, a po sintaksi je sličan Pascal-u. Opisane su vrednosti i tipovi podataka koji se koriste u VAL-u, kao i njegove najznačajnije programske strukture. Njegova efikasnost za paralelno programiranje iskorišćena je na primeru izračunavanja n-tog korena.

ABSTRACT: This paper presents the program realization of n-th rooth evaluation in the high-level dataflow language VAL, using parallel iterative algorithm. The basic terms of the language are reviewed, which is by semantic a functional language and by syntax alike Pascal. Data values and types used in VAL are described, as well as his the most important program structures. His efficiency for parallel programming is utilized in the n-th root evaluation example.

1. UVOD

Klasični pristupi za razvoj multiprocesorskih arhitektura postali su neadekvatni, uglavnom zbog teškoća u raspodeli posla između velikog broja procesora i u sinhronizaciji njihovih operacija. Postalo je očigledno da se, radi iskorišćenja masovnog paralelizma, moraju razviti novi modeli računara koji treba da ispune sledeća dva zahteva:

1) Korisnik mora biti u stanju da opiše rešavane probleme na način pogodan za paralelnu obradu.
2) Računar mora biti u stanju da iskorišćava unutrašnji paralelizam sadržan u problemima koji se rešavaju.

Dataflow računarski sistemi efikasno rešavaju oba ova problema korišćenjem dataflow programskog jezika visokog nivoa, prevođenjem programa pisanog njime u dataflow graf, i njegovim izvršavanjem na paralelnom računaru. Dataflow jezici su uvedeni da bi se omogućilo jednostavno i lako programiranje u multiprocesorskim dataflow sistemima, jer su klasični konkurentni jezici neadekvatni za izražavanje različitih oblika konkurentnosti koji postoje u programima. Viši dataflow jezik VAL (Value-Oriented Algoritmic Language--algoritamski jezik orijentisan prema vrednosti) /1/ projektovan je baš sa ciljem da se njime mogu izraziti svi nivoi i tipovi konkurentnosti na jednostavan i jasan način.

2. PROGRAMSKI JEZIK VAL

2.1. Osnovne napomene o VAL-u

Dataflow programski jezik VAL, po semantici, pripada klasi funkcionalnih ili aplikativnih jezika kod kojih se sva obrada vrši primenom funkcija nad vrednostima. On je funkcionalan u pravom matematičkom smislu te reči. Svaki programski modul predstavlja funkciju koja prihvata neke ulazne podatke i čiji je jedini zadatak da generiše neki skup rezultata. Klasični jezici koriste ove pojmove: naredba i tekuće stanje računanja koje se može predstaviti skupom promenljivih i njihovim trenutnim vrednostima u memoriji. Stanje se menja promenom vrednosti promenljivih. Umesto naredbi i promenljivih, u VAL-u se koriste izrazi i vrednosti. VAL je jezik orijentisan prema vrednosti, jer je svaka aktivnost u VAL programu neki izraz čije izračunavanje generiše skup vrednosti. Svaka vrednost se direktno šalje drugim izrazima kojima je potrebna kao argument tj. ne memoriše se, pa ne postoji promenljiva kojoj bi se ta vrednost dodelila. Radi pogodne notacije u programu, vrednost se može "vezati" za neki identifikator (ime) vrednosti, ali njihova veza mora ostati stalna u celom opsegu važnosti imena vrednosti. Zato identifikator nije promenljiva, jer promenljiva menja svoju vrednost a identifikator ne /2/.

U VAL-u svakom podatku mora biti specificiran njegov tip, dok proveru tipa operanada za svaku operaciju vrši VAL prevodilac. Po sintaksi VAL je sličan Pascal-u, a po semantici funkcionalnim jezicima kao što su LISP i FFP.

2.2. Vrednosti i tipovi podataka

Ulazni i izlazni podaci VAL izraza i funkcija su vrednosti. Skup svih vrednosti koje VAL programi mogu prihvatiti ili generisati predstavlja domen vrednosti VAL jezika. On se deli na više poddomena koji predstavljaju tipove podataka VAL-a. Postoje elementarni tipovi u koje spadaju skalarne vrednosti:nulta vrednost NIL, logičke, celobrojne, realne i znakovne vrednosti;strukturni tipovi: nizovi i slogovi, i unioni tipovi (unije).

Specifikacija tipa je sintaksna konstrukcija

kojom se specificira tip nekog podatka.

Specifikacija elementarnog tipa je samo ime tipa: NULL, BOOLEAN, INTEGER, REAL i CHARACTER.

Specifikacija nizovnog tipa specificira tip svih elemenata niza. Dužina niza nije poznata u vreme prevođenja programa već se određuje u toku njegovog izvršavanja, pri izvršavanju operacije kreiranja niza.

Primer: ARRAY [ARRAY [REAL]] ;

Vrednost niza tipa ARRAY[T] sastoji se iz dve komponente: 1) Opsega (LO, HI) gde su LO i HI celi brojevi i LO≤HI+1. To su granice niza. Ako je LO=HI+1, niz nema elemenata; 2) Sekvence od HI-LO+1 elemenata tipa T.

Specifikacija slogovnog tipa specificira imena polja sloga i tip svakog polja. Imena polja unutar jednog sloga moraju biti različita. Ako je nekolikim imenima polja pridružen isti tip, sva ta polja su tog tipa.

Primer: RECORD [I, J : INTEGER; TEMP: REAL];

Ako su N1,..., Nk imena polja, a T1,...,Tk specifikacije tipova, tada RECORD[N1 : T1,... , Nk : Tk] specificira slogovni tip. Vrednost slogovnog tipa je skup od k parova:{(N1, V1),...,(Nk,Vk)} , gde je Vi vrednost tipa Ti.

Specifikacija unionog tipa specificira imena oznaka i tip svake oznake. Imena oznaka unutar jedne unije moraju biti različita. Ako je nekolikim imenima oznaka pridružen isti tip sve te oznake su tog tipa. Ako je izostavljena specifikacija tipa iza imena oznake, usvaja se nulti tip NULL.

Primer: ONEOF [THIS : ARRAY[INTEGER] ;THAT:
RECORD [C, D : REAL]] ;

Ako su N1,..., Nk imena oznaka, a T1,...,Tk specifikacije tipova, tada ONEOF [N1 : T1,... ; Nk : Tk] specificira unioni tip. Vrednost unionog tipa je samo jedna vrednost, jednog od nekoliko alternativnih tipova T1,...,Tk, kojoj je pridružena oznaka, koja ukazuje kog je tipa ta vrednost, tj. vrednost unionog tipa je par (Ni, Vi), gde je 1≤i≤k a Vi je vrednost tipa Ti.

Definicija funkcije može sadržati definicije tipova koje specificiraju programski-definisane tipove korišćene u njoj. Deo definicije tipa za specifikaciju tipa može sadržati imena tipova definisana u istoj ili drugim definicijama tipova. Tako se, korišćenjem rekurzije u definiciji tipa, mogu formirati tipovi sastavljeni od nizova i/ili slogova neograničene dubine.

Primer:
TYPE STACK=ONEOF [empty:NULL; element:
RECORD[value: REAL; rest: STACK]] ;

2.3. Programske konstrukcije

2.3.1. IF konstrukcija

Uslovni izraz selektuje jedan od nekoliko izraza zavisno od vrednosti logičkog izraza.

primer: IF P THEN X ELSE Y ENDIF +3;

Ovaj izraz ima vrednost X+3 ili Y+3, zavisno od vrednosti P.

Izraz iza IF je test izraz i logičkog je tipa. Izrazi iza THEN i ELSE su grane. IF konstrukcija je takođe jedan izraz čiji je skup vrednosti jednak skupu vrednosti one grane za koju je test izraz istinit.

2.3.2. LET konstrukcija

LET konstrukcija deklariše jedno ili više imena vrednosti, definiše njihove vrednosti i izračunava jedan izraz koristeći te definisane vrednosti.

Primer:

```
LET A, B, C, X, Y, DISC : REAL, O.K.:BOOLEAN;
    DISC:=B*B-4.0*A*C
    X, Y, O.K.:=
IF DISC>0.0  THEN  (EXP(DISC, 0.5)-B)/(2.0*A),
                   (-EXP(DISC,0.5)-B)/(2.0*A),
                   TRUE
             ELSE  0.0, 0.0, FALSE
ENDIF
IN X, Y, O.K.
ENDLET
```

U ovom LET bloku izračunavaju se realni koreni kvadratne jednačine A*x*x+B*x+C=0. IF izraz generiše dva korena i logičku vrednost koja pokazuje da li su koreni realni. Primetimo da, pošto obe grane IF izraza moraju vratiti jednak broj rezultata istih tipova, moraju se vratiti dva realna broja čak i ako je diskriminanta negativna.

Svako ime vrednosti uvedeno u LET bloku može biti deklarisano i definisano samo jednom u njemu. Deklaracija mora prethoditi definiciji ili može biti deo nje. Svako ime vrednosti mora biti definisano pre njegovog korišćenja na desnoj strani neke druge definicije. Definicije i deklacije se mogu mešati po želji.

Pošto se ime vrednosti ne sme koristiti pre nego što se definiše i može biti definisamo samo jednom u bloku, to znači da se ono ne sme pojaviti u sopstvenoj definiciji. Zato su definicije oblika "I:=I+1" neispravne u LET bloku. (Mada se mogu javiti u ITER grani FOR konstrukcije - videti naredni odeljak.)

2.3.3. FOR konstrukcija

Ovaj izraz vrši sekvencijalnu iteraciju u kojoj naredni ciklus zavisi od rezultata prethodnog ciklusa. On uvodi jedno ili više imena vrednosti koja se zovu imena petlje, a služe da prenose informacije iz jednog u drugi ciklus. FOR konstrukcija predstavlja iterativni oblik petlje u VAL-u, kod koje se međuzavisnost za podacima prenose iz jednog u drugi iterativni ciklus.

Primer:

```
FOR  Y: INTEGER:=1; P : INTEGER:=N;
DO IF P≠1 THEN ITER Y:=P*Y; P:=P-1; ENDITER
          ELSE Y
   ENDIF
ENDFOR
```

Ova petlja izračunava faktorijel od N. Njeno telo je IF-THEN-ELSE konstrukcija čija se prva grana sastoji iz predefinicija, a druga grana je izraz Y. Na početku svakog iterativnog ciklusa ispituje se vrednost P. Ako ona nije jednaka 1, izvršava se THEN grana i počinje sledeći ciklus; u protivnom, petlja se završava sa vrednošću Y.

Imena petlje su ona imena koja se deklarišu i definišu iza reči FOR i to na isti način kao u LET bloku. Za njih važe ista pravila kao i za imena vrednosti u LET bloku.

FOR konstrukcija funkcioniše na sledeći način. Imena petlje su inicijaliziraju, samo jednom, vrednostima navedenim u definicijama iza reči FOR, i počinje prvi iterativni ciklus. Za vreme svakog iterativnog ciklusa imena imaju fiksne vrednosti. Zatim se izračunava telo petlje koristeći trenutne vrednosti imena petlje. Rezultat ovog izračunavanja je ili odluka da se izadje iz petlje sa izračunatim vrednostima, ili odluka da se otpočne novi ciklus sa novim vrednostima imena petlje.

Telo petlje sastoji se od uslovne ili LET konstrukcije, ili stabla uslovnih ili LET konstrukcija, sa malom razlikom: njihove grane mogu biti ili obični izrazi ili se mogu sastojati iz reči ITER, nekih predefinicija i ENDITER. Ako se izabrana grana sastoji od ITER, predefinicija i ENDITER, odredjena imena petlje se predefinišu prema desnim stranama predefinicija, i ponavlja se izračunavanje tela petlje.

Unutar svake ITER grane, predefinisana imena vrednosti moraju biti podskup imena petlje. U ovim predefinicijama mogu se koristiti prethodne vrednosti svih imena petlje, uključujući i vrednost onog imena koje se predefinište. Predefinicije ne uključuju dekleracije jer su tipovi imena petlje deklarisani na početku FOR konstrukcije. Za razliku od definicija u LET bloku ili na početku petlje, predefinicija u ITER grani može na svojoj desnoj grani sadržati imena petlje koja se javljaju na levoj strani te iste ili neke od kasnijih predefinicija. U tom slučaju koristi se stara (prethodna) vrednost imena petlje tj. vrednost koju je ono imalo u prethodnom iterativnom ciklusu. Ako se neko ime petlje na desnoj strani predefinicije pojavilo na levoj strani neke prethodne predefinicije, tada se koristi njegova nova vrednost, tj. rezultat te prethodne predefinicije. Zato je predefinicija oblika "J:=J+1" ispravna i znači da će u sledećem interativnom ciklusu vrednost J biti za 1 veća od njene vrednosti u prethodnom ciklusu. Ime petlje koje se ne predefiniše zadržava svoju staru vrednost.

2.3.4. FORALL konstrukcija

Ova konstrukcija generiše jedan ili više skupova vrednosti koje su istog tipa unutar svakog skupa, pa ih ili vraća kao nizove ili vraća rezultat primene neke operacije nad njima. Za prvi slučaj se koristi reč CONSTRUCT, a za drugi reč EVAL iza koje sledi ime operatora. Izračunate vrednosti ne smeju da zavise jedna od druge, jer je cilj da se one izračunavaju istovremeno na paralelnom računaru koji bi mogao da radi na ovaj način. FORALL konstrukcija predstavlja paralelan oblik petlje u VAL-u, kod koje se svi njeni koraci mogu konkurentno i nezavisno izvršavati na posebnim procesorskim elementima.

Primer:

```
FORALL I IN [1, L],J IN [1,  M]
   E  :  ARRAY[ARRAY[REAL]] :=
         FORALL K IN [1, N]
            F  : REAL:=A[I,K]*B[K,J]
         EVAL PLUS F
         ENDALL
   CONSTRUCT E
   ENDALL
```

U ovoj ulančanoj FORALL petlji vrši se množenje matrica A i B reda L x M i M x N, respektivno. Elementi novoformirane matrice izračunavaju se unutrašnjom FORALL-EVAL petljom množenjem odgovarajućih elemenata matrica A i B, i sabiranjem rezultata tih množenja operatorom PLUS. Sva množenja vrše se paralelno i nezavisno. Vrste matrice rezultata formiraju se u unutrašnjoj od dve ulančane FORALL-CONSTRUCT petlje, da bi se zatim od njih formirala cela matrica u spoljnoj FORALL-CONSTRUCT petlji. Formiranje svih vrsta, i od njih cele matrice, vrši se paralelno.

FORALL uvodi jedno ili više indeksnih imena vrednosti celog tipa (imena ispred reči IN) i nekoliko opcionih privremenih imena vrednosti (imena u deklaracijama i definicijama). Obe vrednosti izraza iza reči IN su celog tipa. To su donja i gornja granica indeksa. FORALL konstrukcija funkcioniše tako što indeks dobija vrednost svakog broja izmedju svojih granica, izvrše se definicije privremenih imena vrednosti, i izračunavaju se svi delovi tela konstrukcije. Ako je dato više indeksa, ovo se vrši za sve moguće kombinacije vrednosti tih indeksa.

U CONSTRUCT delu, izraz u telu konstrukcije se izračunava za svaku vrednost indeksa, i za svaku komponentu tog izraza formira se niz čije su granice jednake granicama indeksa a čiji su elementi jednaki izračunatim vrednostima. Ako je dato više indeksa, formiraju se više-dimenzioni nizovi, pri čemu prvi indeks indeksira najspoljniji niz.

U EVAL delu, operacija mora biti jedna od sledećih: PLUS, TIMES, MIN, MAX, OR ili AND. Izraz iza reči EVAL izračunava se za svaku vrednost indeksa, a data operacija se izvršava nad celim skupom generisanih vrednosti. Ako je dato više indeksa, data operacija se izvršava nad skupom vrednosti generisanih za sve moguće kombinacije vrednosti indeksa.

2.4. Definicija i poziv funkcije

VAL program se sastoji iz više modula koji se posebno prevode. Način njihovog povezivanja u kompletan izvršni program i smeštanja u programsku biblioteku zavisi od konkretne implementacije. Svaki modul definiše jednu spoljašnju i jednu ili više unutrašnjih funkcija. Svaka funkcija se definiše pomoću definicije funkcije koja se sastoji iz sledećih komponenti:

1) Rezervisana reč FUNCTION.
2) Zaglavlje funkcije koju čine ime funkcije, deklaracije formalnih argumenata i specifikacije tipova povratnih vrednosti.
3) Definicije programski-definisanih tipova čija se imena mogu koristiti u celoj definiciji funkcije, uključujući i njeno sopstveno zaglavlje. Ovde se takodje nalaze i EXTERNAL deklaracije spoljašnjih funkcija koje koristi ovaj modul.
4) Definicije unutrašnjih funkcija koje se koriste u ovoj funkciji.
5) Izraz koji izračunava povratne vrednosti funkcije. To je telo definicije funkcije.
6) Rezervisana reč ENDFUN.

Poziv funkcije se sastoji iz imena funkcije iza koga sledi lista stvarnih arguumenata izmedju zagrada. Ova sintaksa je ista i za spoljašnje i za unutrašnje funkcije. Svaka funkcija može koristiti samo podatke koje primi pri svom pozivu. Ona ne prima nikakve podatke od definicija svojih unutrašnjih funkcija.

3. IZRAČUNAVANJE N-TOG KORENA KORIŠĆENJEM PARALELNOG ITERATIVNOG ALGORITMA

3.1. Metoda

U literaturi je poznato više iterativnih formula za izračunavanje n-tog korena realnog broja a (a>0). Broj iteracija iterativnog postupka zavisi od reda konvergencije formule i izabrane startne vrednosti. Formulama brže konvergencije računa se u manji braj iteracija ali one u sebi imaju više operacija sa aspekta mašinskog vremena. Zbog toga se traži kompromis izmedju brzine računanja i zauzetosti memorije. Pri stvaranju standardnih programa za računare postavlja se zahtev da se brzo računa, ali sa što manje zauzetosti memorije. Ako je početna vrednost korena suviše udaljena od tačne vrednosti, izračunate približne vrednosti korena sporo konvergiraju ka tačnoj vrednosti, pa je za izračunavanje potreban veliki broj iterativnih koraka. Ako se startna vrednost pažljivo bira, broj iteracija se može svesti na dovoljno mali broj a da se tražena tačnost zadovolji.

U ovom radu prikazana je realizacija brzog i tačnog izračunavanja n-tog korena u VAL-u. Ovaj problem je ekvivalentan sa rešavanjem jednačine $x^n-a=0$. Za njegovo rešavanje ovde je korišćen metod prikazan u radu /5/. U njemu se, polazeći od izabrane početne približne vrednosti korena, primenom odredjene rekurentne formule formira niz približnih vrednosti korena koje postepeno konvergiraju ka tačnoj vrednosti. Konvergencija niza se obezbedjuje izborom odgovarajuće početne približne vrednosti korena i rekurentne formule. U primenjenoj rekurentnoj formuli figuriše i jedan parametar p za ubrzavanje konvergencije, koji smanjuje broj iterativnih koraka potrebnih da se dodje do tačne vrednosti korena.

Algoritam za izračunavanje n-tog korena bazira se na činjenici da se svaki realan broj a>1 može predstaviti u notaciji pokretnog zareza u obliku (3.1.), gde je x mantisa, k eksponent, a b brojna osnova računara.

$$a=b^{k}*x \qquad (1/b\le x<1) \qquad (3.1.)$$

Za brojeve a<1/b koristi se oblik

$$a=b^{(-k)}*x \qquad (3.2.)$$

gde je k prirodan broj ili 0.

Ako se izvrši operacija korenovanja na broju $a=b^{(\pm k)}*x$ dobiće se

$$b^{(\pm k/n)}*r(x) \qquad (3.3.)$$

gde je r(x) aproksimacija x na opsegu $[1/b,1]$.

Dalje se može pisati

$$b^{(k/n)}=b^{i}*b^{(l/n)} \qquad (3.4.)$$

gde su k, n, l celi brojevi. Izjednačavanje stepena u ovom izrazu dobija se

$$l=k-n*i. \qquad (3.5.)$$

Uzimajući u obzir jednakosti (3.4.) i (3.5.) i koristeći najprostiju linearnu aproksimaciju startne vrednosti korena

$$x0=\left(\frac{12*n}{2*n+1}-\frac{6*n}{n+1}\right)*a+\frac{4*n}{n+1}-\frac{6*n}{2*n+1} \qquad (3.6.)$$

može se formirati poboljšana početna vrednost korena kojom iterativni proces počinje /6/. Algoritam za njeno formiranje dat je u programskoj realizaciji metode. Ovaj algoritam predstavlja poboljšanje u odnosu na početnu aproksimaciju oblika

$$x0=b^{*i} \qquad (3.7.)$$

koja se često sreće. Praktični rezultati su pokazali da poboljšani algoritam ima za oko 20% manji broj iteracija u odnosu na algoritam sa početnom vrednošću b^{*l}.

Sa tako odredjenom startnom vrednošću korena primenjuje se iterativna formula

$$x(i+1)=x(i)*\left(1-\frac{x(i)^{*n}-a}{n*x(i)^{*n}+p*(x(i)^{*n}-a)}\right) \qquad (3.8.)$$

u kojoj je p parametar za ubrzavanje konvergencije. Ako pretpostavimo da parametar p uzima m različitih vrednosti p(j) (j=1,2,...,m) i da je x(i+1, j) odgovarajuća približna vrednost korena izračunata pomoću formule (3.8.) za datu približnu vrednost x(i), tada se kao sledeća tačnija aproksimacija korena uzima aritmetička sredina m izračunatih vrednosti.

$$S(i+1)=\sum_{j=1}^{m} x(i+1, j)/m \qquad (3.9.)$$

Ako ona zadovoljava traženu tačnost uzima se kao rešenje, inače se uzima kao sledeća tačnija aproksimacija za naredni iterativni korak. Red konvergencije ove metode je m+2. Tako se, povećanjem brzine konvergencije, formulom(3.8.) smanjuje broj iterativnih koraka potrebnih da se dodje do tačne vrednosti korena, čime se štedi mašinsko vreme.

3.2. Programska realizacija

U ovom radu uzeto je da je m=4, pa se opisani algoritam može primeniti za izračunavanje korena reda n≥4. Vrednosti parametra p odredjuju se iterativnim postupkom kao koreni odredjenog polinoma m-tog stepena, koji se dobija primenom teorije o simetričnim polinomima. Algoritam za njihovo izračunavanje prikazan je u radu /5/. Ovde je usvojeno da su one unapred izračunate, tako da se u programu učitavanju kao ulazni parametri. Vrednosti parametra p za izračunavanje korena reda n=4,....,10 date su u tablicama na slici 1.

n	P1	P2
4	-3.2325708207635052	-1.3866604755783284
5	-4.1399516638010342	-2.1122912630726287
6	-5.0436283701595526	-2.7457281407818382
7	-5.9453086293195556	-3.3624215685739527
8	-6.8457957198231788	-3.9716843906536000
9	-7.7455156160747983	-4.5768023569325406
10	-8.6447144365224048	-5.1793144252452301

n	P3	P4
4	-1.2710725244812891	-0.1096961791768777
5	-1.5340673156281110	-0.2136897574982265
6	-1.8946202271861466	-0.3160232618725641
7	-2.2743773676695521	-0.4178924344369407
8	-2.6628446684031332	-0.5196752211200897
9	-3.0561869272678491	-0.6214950997248128
10	-3.4525820152776382	-0.7233891229547285

Sl. 1.

Kako m različitih vrednosti ne zavise jedna od druge, već samo od vrednosti parametara p (j), to se one mogu izračunavati nezavisno i konkurentno. Tako ovaj iterativni algoritam poseduje implicitni paralelizam i pruža mogućnost za konkurentno programiranje, što je iskorišćeno pri njegovoj implementaciji u VAL-u.

VAL program za realizaciju ovog algoritma dat je na sl. 2.

Programski modul ROOTN kao ulazne parametre prihvata red korena N, vektor parametara P koji odgovara N-u, brojnu osnovu računara B, i broj A čiji se koren izračunava. Unutrašnja funkcija POKZAR vrši predstavljanje broja A u notaciji pokretnog zareza po formulama (3.1.) i (3.2.). Kao izlazne vrednosti ova funkcija generiše mantisu XA broja A, eksponent K, a L dobija vrednost -1, ako je A<1/B, ili 1, ako ovaj uslov nije ispunjen. Telo ove funkcije sastoji se iz LET-IN bloka.

Za realizaciju paralelnog interativnog algoritma, u telu spoljašnje funkcije ROOTN koristi se paralelna FORALL petlja. Najpre se, opisanim poboljšanim algoritmom odredjuje početna vrednost korena YOO sa kojom počinje iterativni proces u spoljašnjoj FOR petlji. Zatim se u unutrašnjoj FORALL-EVAL petlji paralelno izračunavaju četiri različite vrednosti korena za četiri različite vrednosti parametra P. Sve vrednosti korena paralelno se sabiraju operatorom PLUS, koristeći osobinu asocijativnosti za sabiranje, i odredi se njihova aritmetička sredina. Ako ona zadovoljava traženu tačnost, vraća se kao rešenje, inače se uzima kao sledeća tačnija aproksimacija korena, i izvršava se naredni iterativni korak.

4. ZAKLJUČAK

Rezultati dobijeni primenom formule korišćene u ovom radu pokazuju da se željena tačnost od 10**(-15) pri bilo kojem n od 4 do 10 postiže u tri iteracije. Ova formula daje razultate brže od Newtonove uglavnom za dve iteracije. Prednost ovog algoritma u odnosu na druge je u njegovoj unutrašnjoj konkurentnosti, pošto se pojedina izračunavanja u njemu mogu obavljati nezavisno i konkurentno, tako da je idealan za programiranje na konkurentnim jezicima kao što je VAL, koji je korišćen u ovom radu.

LITERATURA:

/1/ W. B. Ackerman, J. B. Dennis, "VAL - A Value-oriented Algoritmic Language: Preliminary Reference Manuel", Tech. Report TR-218, Lab. for Computer Sci., MIT, Cembridge, June 13, 1979.

/2/ J. R. McGraw, "Dataflow Computing - Software Development" , IEEE Trans. Computers, Vol.29, No.12, December 1980, 1095-1103.

/3/ J. R. McGraw et al, "SISAL - Streams and Iteration in a Single Assignment Language: Language Reference Manuel, Version 1.2", Tech. Report M-146,

```
FUNCTION ROOTN(N : INTEGER; P : ARRAY [REAL] ; B, A : REAL RETURNS REAL)
FUNCTION POKZAR(B, A : REAL RETURNS REAL, INTEGER, INTEGER)
LET R : REAL:=1.0/B;
    P : INTEGER:=
        IF   A<1.0  THEN  1
                    ELSEIF  A=1.0  THEN  2
                                   ELSE  3
        ENDIF
IN   IF P=1  THEN  FOR  A1  :  REAL:=A;
                        K1  :  INTEGER:=0
                   DO   IF    A1<R  THEN  ITER  A1:=A1*B,
                                            K1:=K1+1
                                      ENDITER
                                ELSE  A1,  K1,  -1
                        ENDIF
                   ENDFOR
     ELSEIF  P=2  THEN  A, 0, 0
     ELSE  FOR  A2 : REAL:=A;
                K2 : INTEGER:=0
           DO  IF  A2>=1.0   THEN  ITER A2:=A2*R;
                                        K2:=K2+1
                                   ENDITER
                             ELSE  A2, K2, 1
               ENDIF
          ENDFOR
    ENDIF
ENDLET
ENDFUN
FOR  K, L, I, L1  :  INTEGER;
     AN, A0, B0, AB, A1, B1, XA, Y0, Y01, Y00, Y, EPS : REAL;
     AN:=REAL(N);
     EPS:=0.1E-15;
     A0:=2.0*AN+1.0;
     B0:=AN+1.0;
     AB:=A0*B0;
     A1:=6.0*AN*(2.0*B0-A0)/(A*B);
     B1:=2.0*AN*(2.0*A0-3.0*B0)/(A*B);
     XA, K, L:=POKZAR(B, A);
     I:=K/N;
     L1:=(K-N*I)*L;
     Y0:=(A1*XA+B1)*EXP(B, L*I);
     Y01:=
        IF L1=0 THEN 1.0
                ELSE EXP((A1*1.0/B+B1), -L1)
        ENDIF;
     Y00:=Y0*Y01;
     Y:=Y00
DO   LET  T0, T1, T2, YY : REAL;
          T0:=EXP(Y, N);
          T1:=T0-A;
          T2:=N*T0;
          YY:=
            FORALL J IN [1, 4]
              YY1 : REAL:=Y*(1.0-T1/(T2+P[J]*T1))
            EVAL PLUS YY1
            ENDALL /4.0
    IN IF  ABS(YY, Y) <=EPS  THEN  YY
                             ELSE  ITER  Y:=YY  ENDITER
       ENDIF
   ENDLET
ENDFOR
ENDFUN
```

Sl. 2.

Lawrence Livermore National Lab., Livermore, California, March 1, 1985.

/4/ J. B. Dennis et al, "Modeling the Weather with a Dataflow Supercomputer", IEEE Trans. Computers, Vol.33, No.7, July 1984, 592-602.

/5/ L. N. Djordjević, "On N-th Root Evaluation by Iterative Metods with Parameters", Informatica, No.4, 1977, 60-62.

/6/ M. S. Milićević, "Početne aproksimacije u iterativnom izračunavanju n-tog korena i formiranje poboljšanog algoritma", Automatika, Zagreb, 1974.

UDK 681.3.007

Anton P. Železnikar
Iskra Delta, Ljubljana

Properties of Information: A Small Encyclopedia

This encyclopedia presents a first trial in approaching the thinking which is basically characteristic for a sensefully and usefully broadened comprehension of information. Certainly, this encyclopedia is not finished and is still arising, thus, (A) in the title is marking that in this part of it merely terms beginning with the letter A are elaborated. The broadened notion of information is grounded in common-sense experience and everyday use of the term information and its lingual derivarives. Already, several advantages of this broadenend comprehension can be observed concerning as the philosophy of information qua a discipline of sufficiently precise thinking as also those parts of science and technology which deal with specific informational problems of intelligence, e.g., of intelligen machines, programs, and methodologies.

Ta enciklopedia je le prvi poskus v približevanju tistemu mišljenju, ki je lahko temeljno značilno za smiselno in uporabno razširjeno pojmovanje informacije. Ta enciklopedija seveda ni končana, je šele v nastajanju in (A) v naslovu označuje, da so v tem delu obdelani le pojmi, ki začenjajo s črko A. Razširjeni pojem informacije temelji na zdravorazumskem izkustvu in vsakdanji rabi pojma informacije in njenih jezikovnih izpeljank. Spoznavna je tudi že vrsta prednosti razširjenega pojmovanja informacije, ki se tiče tako filozofije informacije kot discipline dovolj natančnega razmišljanja kakor tudi tistih delov znanosti in tehnologije, ki so povezani s specifičnimi informacijskimi problemi inteligence, tj. inteligenčnih strojev, programov in metodologij.

-1

Ta enciklopedija je samo prvi poskus v približevanju tisti miselnosti, katere osnovna značilnost je smiselno in uporabno razširjeno pojmovanje informacije. Enciklopedija seveda ni končana, temveč je v nastajanju in (A) v naslovu označuje, da so v tem delu prispevka obdelani le pojmi, ki začenjajo s črko A.

Zavedam se, da bo razširjeno pojmovanje informacije naletelo na vrsto pomislekov. Vendar se ob teh pomislekih, ki jim vselej manjka konkretna podlaga, opiram na vsakdanje, zdravorazumsko izkustvo, ki mi pravi, da tudi preprost, neizobražen človek razumeva informacijo kot pomen določene fenomenologije v razširjenem pomenu. Seveda vidim v razširjenem pomenu informacije določene prednosti, ki zadevajo tako filozofijo kot disciplino dovolj natančnega razmišljanja kakor tudi tisti del znanosti in tehnologije, ki se danes intenzivno sooča s specifično informacijskimi problemi takoimenovane inteligence, tj. inteligentnih strojev, programov in metodologij.

Obravnava pojmov, ki začenjajo s črko A v tem delu prispevka, res ni najbolj posrečena, saj bi bila črka I nedvomno v tem trenutku aktualnejša. Tu svetujem, da je mogoče najti iz informacije izpeljane pojme in informacijske opredelitve drugih pojmov v literaturi (OWI, ID1, ID2, POI). Vsekakor pa je del enciklopedije za črko I v obdelavi in bo objavljen takoj, ko bo delo v tej aproksimativni fazi na njem končano.

0

V tej enciklopediji bodo v abecednem zaporedju terminov opisani primeri različnih lastnosti informacije glede na vsakdanje izrazoslovje, ki je udomačeno v filozofiji, sociologiji, medicini, biologiji, psihologiji, fiziologiji, nevroanatomiji, biokemiji in informatiki (glej slovstvo pod oznakami FIR, DPH, MDL, DMB, PNS, itd. na koncu prispevka). Informacija in raziskava informacije uporablja svoj jezik, ki niso le jezikovni principi in nespremenljiva pravila, temveč tudi informaciji svojska razvojnost. Jezik informacije, ki izraža različne fenomenologije kot informacijo, ni samo jezik teh fenomenologij, npr. ni samo jezik filozofije, sociologije, psihologije, biologije, ali vobče neke znanosti. To je jezik

nastajajoče filozofije informacije, ki se opira na informacijo kot svojsko fenomenološko razvojnost v živem in neživem. Jezik filozofije informacije je tako kot filozofija informacije šele v nastajanju in v tej fazi nastajanja je še bistveno povezan z jeziki informacijsko obravnavanih fenomenologij.

Lastnosti raznovrstnih informacijskih oblik bodo strnjene v abecedno urejena poglavja, označena z A do Z. Pri raziskavi različnih lastnosti informacije v posameznih primerih terminov bodo upoštevana nekatera izhodišča dela (OWI, ID1, ID2 in POI), ki obravnavajo informacijo kot informacijo. Raziskava lastnosti informacije bo upoštevala sorodne lastnostnostne pojme iz filozofije, medicine, psihologije nevrofiziologije itd., vendar predvsem kot informacijske entitete. S takšnim načinom obravanave se bo razširjala pojmovnost informacije, z ustrezno pojmovnostjo pa tudi uporabnost informacije kot zamisli dovolj splošne spoznavne oblike in dovolj splošnega procesa razumevanja.

Ob vsakem pojavu pojma (oblike) ali dogodka (procesa) bi se bilo upravičeno vselej znova vpraševati, kaj je tak pojem ali dogodek kot pojav informacije, kot informacijski pojav in kakšna je njegova informacijska osnova v kontekstu razumevanja informacije kot informacije. Takšno vpraševanje bi ne samo preprečevalo nastajanje informacijske slepote, absurdov, praktičnih in teoretičnih ideologij, ki odlagajo in zaslepljujejo nastajanje razumevanja pojmov in dogodkov, marveč bi ta nasprotja lahko tudi vselej osvetljevalo kot regularno informacijo v danem socialnem okolju.

A

ABAZIJA je vobče nesposobnost hoje zaradi pomanjkljive motorne koordinacije (pomanjkanja motorne informacije), toda brez paralize. Abazija je lahko sinonim za pomanjkanje koordinacijske (usklajevalne) informacije v okviru informacijskega procesa v živem (celici, organizmu, korteksu) in v neživem (stroju, prostoru, času).

ABDUKCIJA je silogizem, v katerem je glavna (majorantna) premisa resnična, podrejena (minorna) premisa pa samo verjetna. Abdukcija je hipotetična, verjetnostna, statistična informacija, ki pojasnjuje dano množico faktov.

ABERACIJA je pojem za informacijsko deviacijo, anomalijo, nenormalno varianto, abnormalnost, nepravilnost, izvenpravilnost. Aberacija je v domeni protiinformacije, informacijskega nastajanja, spremenjene, od norme odstopajoče vidnosti, pogleda, informacijske barvitosti.

Kromozomska aberacija je sprememba (nenormalnost) v številu kromosomov (informacijskih nosilcev dedne zasnove) ali v strukturi posameznega kromosoma (spremembe v nukleotidnem zaporedju). Heterosomalna aberacija je sprememba v strukturi več kromosomov, navadno z translokacijo. Homosomalna aberacija je sprememba strukture v posameznem kromosomu. Genetični lokusi ali nukleotidna zaporedja so lahko zbrisani, podvojeni, obrnjeni ali na druge načine premeščeni brez očitne spremembe glede na komplementarni kromosom.

Mentalna aberacija je odstopanje od normalne informacijske aktivnosti mišljenja. Penta-X kromosomska aberacija je pojav petih kromosomov X pri človeku (kariotipska oznaka 49,XXXXX). Posledice te informacijske aberacije so bistvena mentalna zaostalost, postnatalno zaostajanje v rasti (majhne roke) in patent ductus arteriosus. Druge kromosomske aberacije so npr. še tetra-X (48,XXXX) in triple-X (47,XXX). Kromosomska aberacija se lahko pojavi tudi zaradi sevanja (X-sevanje ali laserska svetloba).

ABIOGENEZA. Beseda prihaja iz grščine in ima v latinščini pomen generatio aequivoca spontanea, pomeni spontanost v nastajanju živega, organskega iz anorganskega po naravni fizikalno-kemijski poti, tj. s samorojevanjem, samoprodukcijo, avtogonijo (spontano generiranje). Abiogeneza je tako neposredno primerljiva s pojmom avtopoiesis (ID1), ki ga je skoval H. R. Maturana za označevanje biološke samoprodukcije. Informacijska abiogeneza pomeni spontanost nastajanja informacije iz same sebe in iz druge (okoliške) informacije, ima tedaj pomen spontanega generiranja informacije iz informacije z informacijo. Informacija je tedaj kot informacijska abiogeneza primerna za izražanje kakršnihkoli abiogenetskih pojavov.

ABIOLOŠKA (nebiološka) informacija nastaja v živih organizmih in izven njih (npr. v živi in neživi snovi, v neživih delih vesolja, v anorganskem svetu). Pojem abiološke informacije je informacijsko relativen, saj je povezan z zaznavanjem, s spoznavanjem in razumevanjem okoliške informacije z lastno (živo) informacijo.

ABIOFIZIOLOŠKA informacija se pojavlja z anorgansko aktivnostjo v živih organizmih.

ABIOTROFIJA so neznani, starostni degenerativni procesi v živčnih celicah osrednjega živčnega sistema. Abiotrofična informacija je vselej značilno spremenjena, degenerirana, nastajajoča v individualno ali populacijsko retardiranem informacijskem okolju (ki se kaže npr. kot čudaštvo ali starostna ideologija).

ABIOZA je odsotnost življenja. Abiotična informacija nastaja iz neživega (npr. umetna, strojna, druga neživa informacija), nastaja v okolju, ki ne podpira življenja.

ABLEPSIJA je nesposobnost videnja, slepota. Ableptična informacija ne zmore dovolj kritično opazovati sama sebe, ne more biti kritično razspoznana, tako da lahko vztraja, vzdržuje, podpira svojo slepoto, svoje nekritično nastajanje. Trdna (celostna, totalitarna) ideologija je značilen primer ableptične, slepe in slepilne informacije.

ABNORMALNA (zmotna, čudna, blodna) informacija se nanaša na stanje, strukturo, funkcijo, mrežo, pojavnost, ki je bistveno različna od določene informacijske norme. Abnormalnost je odklon od normale glede na nekatere atribute. Informacijska abnormalnost je relativna in lahko preide v informacijsko regularnost z ritmičnim in harmoničnim ponavljanjem.

ABREAKCIJA je čustvena sprostitev, ki je povezana z doslej potlačenim izkustvom (avtoterapevtični informacijski proces).

ABSOLUTNA informacija je samosvoja, neomejena, popolna, samozadostna, nekvalificirana, neocenjevana, neodvisna in brezpogojna, je teoretično nasprotje relativne informacije. Absolutna informacija bi bila informacijsko popolnoma osamljena (izolirana), nevplivana, neinformatizirana z drugo nastajajočo informacijo.

ABSTRAKCIJA je informacija oddvojenosti, ločenosti, poudarjene splošnosti ali bistvenosti, zanemarjanja vzporednosti ali posamičnosti, neupoštevanja, zožene ali poenostavljene pojmovnosti druge (tudi drugače bistvene) informacije. Abstrakcija je

oblikovanje pojma v smeri od manj splošnega k bolj splošnemu. Abstrakcija je tako "smiselna" opredelitev posplošenosti. Abstraktna informacija tudi ni konkretna, saj zanemarja širši informacijski kontekst. Bitjevska informacija je bolj ali manj, tako ali drugače abstraktna, saj preprosto ne more zavestno (dovolj natančno) zajemati celotne življenske in kulturne informacije.

ABSURD je informacija neskladnosti, nesmiselnosti, protislovnosti, nelogičnosti v primerjavi z veljavno bitjevsko, populacijsko ali kulturno informacijo. Absurd je protiinformacija, ki se v temeljih upira obstoječi informaciji in izpostavlja nesmiselna nasprotja. Nastanek absurda kot informacije je bistven dosežek protiinformiranja bitjevske informacije. Z absurdom se dokazuje določeno protislovje, nasprotje (reductio ad absurdum). Absurd je osrednji protiinformacijski proces, je nasprotje zdravemu razumu (tudi filozofska kategorija).

ACEFALNO je brezglavo, je lahko posledica dednega defekta. Acefalni informaciji manjka (je skrito) bistveno (glavno), kar jo kot informacijo karakterizira (tipiziranost, namernost, usmerjenost).

ACELULARNO pomeni brez celic, torej brez biološko in informacijsko značilnega za živo celico. Virus je acelularen, ima primitivnejšo strukturo kot živa celica.

ACERBOFOBIJA je strah pred nejevoljnostjo, grenkobo, bridkostjo, je strah pred rezkim kriticizmom ali brezobzirno in kritično informacijo. Acerbofobična informacija izraža strah pred odprto kritično informacijo, jo patološko odklanja.

ACIKLIČNA informacija bi bila v svojem nastajanju premočrtna, brez vmešćanja nastale protiinformacije. Protiinformacija bi bila s takšnim nastajanjem zgubljena, njen pojav bi bil enak pojavu informacijskega šuma. Aciklična informacija je tudi umetna, statična informacija (npr. podatkovanje), ki ni informacijsko generativna.

ADAPTACIJA ali prilagoditev pomeni zmanjševanje informacijskega učinka ponavljajoče informacije v informacijskem procesu zaradi ponavljajoče informacijske enoličnosti (nespremenljivosti, nenastajalnosti, nespontanosti). Ablogeneza je tako določeno nasprotje adaptacije, njena protiinformacija. Adaptacija pomeni tako tudi zamiranje določene (adaptacijske) protiinformacije. Adaptacija je med drugim informacijsko vmešćanje (angl. embedding) nastale informacije v obstoječo, pomensko veljavno informacijo. Adaptacija je presojanje nastale informacije z veljavno informacijo s stališča ohranjanja veljavnih (na določen način uspešnih, primernih, ugodnih, lahkotnejših) preživetvenih informacijskih obrazcev. Adaptacija (akomodacija) je lahko tudi reduciranje metafizičnih, bitjevskototalnih informacijskih relacij v socialno dominantno informacijo (ideološko mrežo). Adaptacija je lahko progresivna sprememba občutljivosti, prednostna sprememba, s katero bitje modificira zaznavanje neprijetnih dražljajev (svetloba, zvok, toplota, ideologija) v danem okolju.

ADEKVATNOST je informacijska izenačenost (objektivnost), skladnost, ustreznost ali soglasnost (subjektivnost) kot rezultat razumevanja (informacijske opazovalnosti, raziskovalnosti in spoznavnosti) pri primerjanju in povezovanju informacije. Adekvatni so lahko določeni pojmi, ki ustrezajo dani informaciji; sicer so ti pojmi neadekvatni.

ADIAFORA (iz grščine) je nerazlično, neizpostavljeno ali ravnodušno v etičnem pomenu, kot srednje, indiferentno stanje med dobrim in zlim, to kar ni ne eno ne drugo. Adiafora je informacijsko nediferencirano, nepomembno, je nekaj, kar v bistvu informacijsko ni prisotno v danem informacijskem območju.

AFAZIJA je korteksna motnja (disfunkcija, strukturalna napaka, izguba), ki povzroča nesposobnost govorjenja ali razumevanja govorjenega in zapisanega zaradi neustreznosti določenega možganskega predela (cerebralne lezije). Afazija je motorna (ekspresivna) in senzorska (receptivna). Brocina (od Broca) afazija je izguba zmožnosti izražanja misli z besedami, čeprav je razumevanje govorjenega in pisanega govora ohranjeno. Wernickejeva afazija je oblika senzorske ali receptivne afazije zaradi lezije v Wernickejevem predelu dominantnega senčnega režnja. Tu se pojavlja znatna nesposobnost razumevanja posameznih besed, fraz in stavkov, s težavami pri ponavljanju narekovanih besed. Spontani govor je sicer fluenten toda neustrezen, ker se uporabljajo nepravilne besede (parafazija), tudi v nepravem zaporedju, žargonsko. Informacijska afazija je povezana vobče z nesposobnostjo generiranja in razumevanja določene informacije, je tedaj neka oblika bitjevske informacijske disfunkcije.

AFEKCIJA je za razliko od spoznavnega ali voljnega informiranja čustvena oblika informacije, v kateri je afekt navadno kratkotrajno in močno čustveno vzburjenje, ki povzroča spremembe v informaciji bitja.

AFEKT je stanje razuma, čustvo, razpoloženje, ki nastane zaradi ideje ali dražljaja. Afekt je večkrat zunanja reakcija na notranjo (metafizično) informacijo. Afektivnost je subjektivna potencialnost za izkušanje in izražanje čustvenih reakcij v spremenljivih pogojih.

AFERENTNA informacija nekaj prinaša ali nekaj sporoča, podobno kot komunicirana (od nekje poslana in sprejeta) informacija. Aferentna informacija v živem je čutilna (senzorska) informacija, ki prihaja s periferije v osrednji živčni sistem. Eferentna informacija pa sporočilo odnaša (v živem je to motorna informacija).

AFICIRATI pomeni na kaj delovati, vplivati, vzbujati. Informacija aficira informacijo in je informacijsko aficirana; to pomeni, da informacija vselej oblikuje informacijo in je tudi informacijsko oblikovana.

AFINITETA kot informacija povzroča informacijsko navezovanje, ki se kaže kot informacijska sorodnost, razmernost (relacijskost), podobnost (analogičnost), nagnjenost (afektiranost), asociativnost, lahkotnost navezovanja.

AFORIZEM kot informacija je omejenost, ločenost, zaprtost v sebi, nepovezanost z drugo informacijo. Aforizem je lahko primerna filozofska informacija (npr. Pascal, Nietzsche, tudi Heidegger z njemu svojskim izrazom). Individualna informacija je kot izvirna protiinformacija aforistična, dokler ni dovolj temeljito vmešćena oziroma vložena v obstoječo, veljavno informacijo.

AGENS informacije je njena spontanost, nastajalnost, vmestljivost, informatičnost. Agens je pojem za dejavno informacijo, ki informira in protiinformira, generira informacijo in protiinformacijo.

AGNOSTICIZEM kot informacija postavlja meje spoznavanju, ko izhaja iz predpostavke, da obstaja tudi nespoznavno. Bitje lahko spoznava le v okvirih svoje avtopoezije (avtopoiesis), v okvirih svoje trenutne biološke strukture in organizacije, iz temeljev svoje metafizike, ki je trenutna celostna informacija bitja. Tu se informacijski agnosticizem bistveno ločuje od idealizma in materializma kot filozofskih usmeritev, ki poskušata razumevati svet v celoti.

AGNOZIJA je omejenost ali nesposobnost razpoznavanja ali razumevanja predmetov in pojavov (senzorske informacije) pri ohranjeni osnovni občutljivosti na ravni bitjevske informacije. Npr. to kar bitje vidi, ne razpoznava (razpoznavna slepota), to kar sliši kot govor, ne razumeva (razumevna gluhost). Informacijska agnozija je vobče lahko tudi informacijska slepota, ki izvira iz določene informacijske usmerjenosti (filozofije, ideologije, znanosti, tehnologije, prepričanja, vere, doktrinarnosti), ki pa se lahko prekinja s pojavi prekinitvene protiinformacije. Ideacionalna agnozija pomeni nesposobnost sprejemanja in povezovanja idej in konceptov.

AGRAFIJA je nesposobnost pisnega izražanja. Literarna agrafija je nesposobnost pisanja črk. Mentalna agrafija pomeni nesposobnost izražanja idej in konceptov s pisanjem. Agrafija je pisna (izrazna) informacijska nesposobnost.

AGREGAT je vobče kup, ki nastaja le z zunanjim kopičenjem brez tistih notranjih povezav, ki so značilne za organizem. Informacijski agregat bi bil informacijdska množica brez informacijskih povezav, razmerij ali odvisnosti. Agregat bi bila razbita, nerazmernostna informacija, torej informacija, ki ve za informacijo, vendar jo notranje, medsebojno ne povezuje. Informacija bitja ali živa informacija je vselej organska (informacijsko razmernostna, povezana, neagregatna).

AGRESIVNOST je priprava za pristop ali uvod v napad, napadalnost kot potencial in dejanje, ki ima za cilj določene spremembe, povzročanje škode, prilaščanje in uničevanje. Agresivnost je sinonim za sovraštvo, destruktivnost in sadizem, hkrati pa ima tudi tradicionalni pomen dinamičnosti, samozavestnosti, ekspanzivnosti, pritiska. Agresivnost kot informacija bitja nastaja iz razlike med metafiziko bitja in informacijo iz okolja. Agresivnost kot informacijska razlika je oblikovalka informacije z značilno, "napadalno" protiinformacijsko komponento.

AKANTESTEZIJA je informacijska nenormalnost (popačenost, napaka) kožne senzorske percepcije, pri kateri dražljaj dotika povzroči informacijo, da je bil v kožo zaboden oster predmet.

AKATAFAZIJA je nesposobnost oblikovanja idej in njihovega logičnega in razumljivega izražanja. Govorjenje je verbalno in slovnično zmedeno, pojavljajo se neobstoječe in nepravilne besede. Akatafazija je tako deficienca v nastajanju informacije kot hermenevtike (bitjevske pomenskosti) in transformacije hermenevtike v semantiko (veljavno kulturno pomenskost).

AKATALEPSIJA je nesposobnost razumevanja, je oblika mentalne deficience. Že antična doktrina pravi, da ni ničesar znano z gotovostjo. Npr. medicinske diagnostične ali prognostične sodbe so inherentno nezanesljive. Akatalepsija kot informacija spoznava, da informacija nastaja, da ni nikoli dokončno spoznana ali razumljena, je vselej razvijajoča oziroma zamirajoča, protiinformacijska, dokončno neznana in nepredvidljiva.

AKATEKSIJA je pomanjkanje psihične energije, s katero Jaz (ego) oblikuje intrapsihične predstavitve zunanjih objektov ali delov Jaza. Neka informacija se kaže npr. kot nepomembna v informacijskem kontekstu, čeprav izraža pomen določene konfliktne bistvenosti ali simptomatičnosti.

AKCIDENTALNO ima pomen nebistvenega, pojavnostnega, naključnega, spremenljivega, obrobnega in postranskega. Akcidentalno ne spreminja bistva stvari in pojavov. Akcidentalna informacija je protiinformacija z marginalnim informiranjem, ki informacijskega bistva v svojem območju dejansko (informacijsko) ne spreminja. Informacijski učinek ideologije je npr. akcidentalen, če ne povzroča ideološke realizacije. Akcidentalna je lahko tudi informacija nekega svetovnega središča, ki ne povzroča njenega razumevanja v perifernem informacijskem okolju. Akcidentalnost pomeni slej ko prej relativnost v pomembnosti ali nepomembnosti nastajajoče informacije.

AKCIJA na daljavo je fizikalna zamisel, da lahko dano telo vpliva na drugo telo brez posredovanja mehanične povezave (gravitacijska interakcija, model elektromagnetnega polja, unificirana teorija polja). Živa senzorska informacija zaznava informacije polja kot vidne (svetlobne), slušne in taktilne (akustične), toplotne (kvantne), biovalovne (gravitacijske, elektromagnetne) in druge (umetne) signale. To so lahko naravne ali umetne informacijske akcije na daljavo (npr. elektromagnetni prenos zvoka in slike).

AKCIJSKA teorija razločuje akcijo od obnašanja, ko postavlja, da je akcija povezana s pomenom ali namero (namenom). Akcijska teorija je je informacija akcijske analize, ki se začenja z individualnim akterjem. Analiza obravnava značilne akterje v značilnih položajih, ko identificira cilje, pričakovanja in vrednosti (vrednote) akterjev, naravo položaja in znanje akterja o situaciji in o drugih elementih. T. Parsons imenuje te predpostavke akcijski referenčni okvir (okvirna informacija). Doktrina simboličnega interakcionizma priznava dve glavni obliki akcijske teorije: hermenevtično in pozitivistično (M. Weber). Empirično možne akcije so lahko sestavi tradicionalnih, afektivnih, instrumentalnih (namensko racionalnih) in vrednostno racionalnih akcij. Akcijo je tako mogoče opredeljevati v odvisnosti od pomenskosti in smiselnosti glede na akterja. Hermenevtična akcijska teorija poudarja prednost pomenskosti, ko trdi, da sta aktivnost in pomen zapleteno (nerazrešljivo) povezana. Pozitivistične akcijske teorije se ukvarjajo s socialno strukturo in s tem, kako se lahko pojavljajo cilji in sredstva akterjev. Za pozitivistično teorijo sta akcija in interakcija rezidualna koncepta, ki sta manj pomembna od analize socialnega sistema kot celote, v katerem vidi ta teorija človeškega akterja predvsem kot socializirano bitje v skupno kulturo.

Za Parsonsa je akcija obnašanje, ki je vodeno s pomeni akterjev. Akterji imajo cilje in izbirajo ustrezna sredstva. Akcijski procesi so omejeni s položajem (okoljem) in vodeni s simboli in vrednostmi. Najpomembnejsa kategorija je interakcija, tj. akcija, ki je usmerjena proti drugim akterjem. Če je interakcija med dvema akterjema pogostna, nastanejo medsebojna pričakovanja, pojavi se medsebojno prilagojevanje pričakovanj in obnašanj. Ko postanejo pričakovanja zanesljivi napovedovalci obnašanja, se spremenijo v norme, ki upravljajo interakcijo tako, da postane ta bolj učinkovita in za akterje nagrajevalna. Te

norme se potem bitjevsko ponotranjijo.
Akcijska teorija nazorno podpira zamisli in principe informacije (OWI, ID1, POI, ID2).

AKENESTEZIJA je pomanjkanje značilne informacije (značilnega občutka) fizične eksistence, ki se pojavlja kot informacija odrevenelosti, otopelosti, neživosti (npr. posebno stanje ideološke otrplosti).

AKOLUCIJA je pojem iz starogrške filozofije in pomeni sledenje, hojo za čim. Akolucija je sled, ki ju puščata pojma, je takšno njuno razmerje, da sta eden od drugega odvisna. Takšna je npr. lahko odvisnost vzroka in posledice (rekurzivna prepletenost). Akolucija je poučen primer tiste informacije, ki ima pregled nad prepletenostjo in vase-zaokroženostjo informacije. Organska informacija je vselej akolutivna, vsebuje *informacijo o prepletenosti informacije*.

AKOMODACIJA lahko pomeni prilagajanje (biološko, socialno adaptacijo) razvoja organa, bitja, njegovega obnašanja na pojave okolja. Akomodacija je živa informacija, ki razvija nove informacijske sheme iz starih shem zaradi novih izkušenj. Učenje novih nalog je akomodacijski informacijski proces, s katerim nastajajo nove in se spreminjajo stare informacijske sheme. Sheme same po sebi niso toge informacije, tako kot so npr. refleksi.

AKOZMIZEM negira obstoj samostojnega realnega sveta, ki naj bi bil le privid. Posamezno v svetu kot absolutni, neskončni celoti, je lahko le metafizični modus. Bitje lahko razumeva svet le v okviru totalnosti svoje informacije, ki pa ni informacija sveta. Metafizika kot celostna informacija bitja je v svojem bistvu informacijsko akozmična, saj je lahko le avtopoetična.

AKRIZIJA je pomanjkanje krizne informacije, npr. pri napredovanju bolezni. Akrizna informacija je lahko posledica navidezne (ideološke) varnosti, s katero se pojavljajoča kriza zavrača oziroma se z njo odlaga krizna informacija. Akrizna informacija je akritična, ne vidi krize, ne zaznava bolezni.

AKROMATIČNA informacija se posreduje s poslušanjem, z ustnim izražanjem učitelja, brez dialoga, enosmerno iz bitja na bitje, brez določene bitjevske interakcije. Primer akromatične ali brezbarvne informacije je totalitarna ideologija, ki ne prenaša dialoške protiinformacije.

AKSIOLOGIJA je filozofija vrednosti (vrednot) v estetiki, etiki, ontologiji, psihologiji, fenomenologiji, je propedevtična (uvodna) disciplina na različnih področjih praktične filozofije. Aksiologija je informacija o vrednostnih razmerjih informacije. Vrednost kot veljavna informacija in njena protiinformacija je pogojena z čustvenimi in ideološkimi komponentami bitjevske in okoliške informacije, torej nastaja in se spreminja. Družbe, kulture, civilizacije napredujejo in propadajo z različnimi vrednotami. Različnost ali celo nasprotnost vrednot le potrjuje njihovo kulturno oziroma informacijsko relativnost.

AKSIOM (grško zahteva, želja) je princip, teza, sodba, stališče, izrek, ki se ne dokazuje in se obravnava kot resničen (logično pravilen), kot princip ali premisa v induktivnem/deduktivnem dokazovanju (sklepanju). Aksiom je tedaj informacijski konstrukt, izhodiščna informacija, iz katere se s pomočjo pravil aksioamtičnega sistema izpeljuje sistemska (aksiomatično veljavna) informacija. Sam aksiom je aksiomatično-sistemsko neizpeljiva informacija, informacijsko pa je seveda izpeljiv, saj je nastal kot informacija. Aksiom je mogoče obravnavati kot vsako drugo informacijo, raziskovati njegovo informacijsko poreklo in vzroke njegovega nastanka. V tem pomenu principi informacije niso aksiomi, ker so krožni (informacijsko rekurentni) in nastajajoči (informacijsko odprti) konstrukti.

AKSIOMATIKA informacijsko raziskuje splošno in podrobno aksiomatične metode, sisteme in izpeljevalne (sklepne) postopke. Aksiomatično izpeljevanje (dokazovanje, sklepanje) je informacijsko omejeno oziroma formalizirano na poseben način. Aksiomatična metoda nasteva neopredeljene pojme (simbole) in sistemsko nedokazane izreke (aksiome), opredeljuje veljavno konstrukcijo formul s pravili izpeljevanja (aksiomatične dedukcije). Tako dobljene konstrukcije so novi izreki, rezultati aksiomatične teorije. Aksiomatična metoda je značilno informacijsko zaprta, zaokrožena, omejena. Informacijska omejenost aksiomatičnega sistema je opredeljena z njegovo konsistentnostjo (hkrati ne moreta obstajati izrek in njegova negacija), s polnostjo (vsi izreki sistema so izpeljivi) in z množico medsebojno neodvisnih aksiomov (aksiom ni izpeljiv iz drugih aksiomov). Aksiomatični sistem je nazoren primer informacijsko statičnega (nerazvijajočega, nenastajalnega, nespontanega, umetnega) sistema.

AKSON je nevronski proces ali nevronsko vlakno, ki prenaša informacijske impulze in/ali informacijske velemolekule vstran od lastnega celičnega jedra in omogoča bistveni proces nevrona.

AKT ni informacijsko ničesar drugega kot prav nastajanje informacije in v okviru tega nastajanja tudi nastajanje aktualnosti in resničnosti. Informacijski akt izpeljuje informacijsko opredeljenost iz informacijske neopredeljenosti, to pa je prav nastajanje informacije in vsakokratno vmeščanje nastale informacije v obstoječo informacijo. Pri tem je tudi sam akt informacija, ki izpeljuje informacijo in je tudi sam informacijsko izpeljevan. Impulzivni akt nastane brez premišljanja (preudarjanja) ali refleksije (npr. mnenje in odločitev izvedenca). Impulzivni akt je ego-sintoničen, vsebuje komponento ugodja, neko stopnjo nepremagljivosti (spontanosti), izraža sebe neposredno, nepopačeno. Nepopolni akt ne dosega pričakovanega konca, zato opušča svoj načrt, ga odlaga na poznejši čas, ali izraža v spremenjeni, popačeni ali simbolični obliki. Refleksni akt je relativno konstanten, avtomatičen in neodvisen. Simptomatični akt je navidezno nepomemben, brezciljen, nenameren in predstavlja (simbolizira) nezavestno željo ali motiv, ki ga ne razpoznava. Simptomatični akti so npr. izumetničenost, manira, uitje jezika ali pisala, založitev predmetov in podobno obnašanje, ki je kratkotrajno in ga je mogoče korigirati z večjo pozornostjo.

AKTIVACIJA je proces, ki sproži akcijo. Aktivacija je značilnost same informacije: informacija je proces, ki sproža informacijo, informacijsko nastajanje, nastajanje protiinformacije, vmeščanje protiinformacije. Informacijska aktivacija ni samo spontana, je tudi prepleteno (paralelno, sekvencialno) krožna (rekurenčna, rekurzivna). Informacija je v nenehnem stanju svoje aktivacije.

AKTIVIZEM je nazor (informacionizem, doktrina), ki poudarja pomen bitjevske aktivne, zavestne in voljne dejavnosti (informacije) pri spreminjanju okolja, bitja, populacije, družbe, gospodarstva, kulture, sveta, vesolja. V okviru aktivizma se lahko poudarja prednost praktične nad teorijsko informacijo (ekonomizem) ali

obratno (ideologizem).

AKTIVNOST je dejavnost, delovanje, delavnost organizma, bitja, populacije, je njihova interakcija, njihova lastna in medsebojna informacija. Aktivna zavest (bitjevska korteksna informacija) je ustvarjalna (spontana, bisociativna, asinhrona, naključna, nepredvidljiva) na področju misli in volje (obnašanja bitja), lahko pa je tudi pasivna (avtomatska, asociativna, ideološka). Tehnološki, kulturni in socialni dosežki so posledice informacijske aktivnosti uma, bitjevskih korteksnih informacijskih procesov. Sinhrona informacijska aktivnost je ritmična, ponavljajoča (bioritem, informacijske oscilacije).

AKTUALIZEM je filozofska doktrina resničnosti, ki poudarja, da resničnost ni statična, nespremenljiva bit, temveč je proces spontane in namerne dejavnosti, aktivnosti živega ustvarjalnega razvoja in samouresničevanja. Aktualizem se kot informacija lahko približuje dialektiki in avtopoiesisu, ko poudarja, da je bit sveta bivanje in razvijajoče (reproduktivno) dogajanje v živem.

AKTUALNOST (dejanskost, grško 'energeia', nemško 'Wirklichkeit') in potencialnost (možnost, grško 'dynamis') sta nasprotni informaciji, pri čemer je prva oblika, druga pa samo možnost oblike (po Aristotelu). Informacijsko je aktualnost biti-v-obliki, potencialnost pa biti-v-procesu. Aktualnost (območje dogodkov in faktov) je informacijski način, ko informacija povzroča informacijo in je z njo povzročena (informiranje, informatiziranje). Potencialnost ni način informacijskega obstoja temveč proces nastajanja, ko informacija prehaja iz ene v drugo obliko. Informacijsko nastajata aktualnost in potencialnost v medsebojnem informiranju, ko je informacija hkrati oblika in proces informacije.

AKULTURACIJA je informacija, s katero nastaja proces stikanja med različnimi kulturami in rezultati tega stikanja. Akulturacija je informacija neposredne socialne interakcije, odprtosti v druge kulture s pretokom informacije, z medkulturno informatizacijo. Akulturacija je tako tudi akomodacija in asimilacija, ki spreminja informacijsko (bitjevsko, populacijsko) identiteto.

ALEGORIJA je vobče izražanje dane informacije z drugo, podobno (simbolično) informacijo. Alegorija je primerna pesniška, govorniška ali likovna informacijska oblika. Metafora je posebna (namerna) oblika alegorije.

ALEKSIJA je nesposobnost branja pri ohranjeni ostrini vida. Aleksija je oblika vidne agnozije ali receptivne afazije, katere značilnost je nesposobnost razpoznavanja in interpretiranja črk in besed. Ta infomacijska nesposobnost se lahko pojavi zaradi lezije dominantnega temenskega režnja (korteksa), lahko pa je tudi razvojna (prirojena besedna slepota). Aleksična informacija ne more razpoznavati in interpretirati (upomeniti, hermenevtično predstaviti) izpisane informacije, jo tako zaznava kot (nerazumljivi) informacijski šum.

ALERGIJA je stanje preobčutljivosti (hipersenzitivnosti) z vpletenostjo imunološko povzročenih sprememb (spremenjena reaktivnost gostitelja ob pojavu antigena). Celična alergija je povečana, s celico povzročena imunska reakcija na specifičen antigen. Alergija kot informacija je informacijska preobčutljivost in informacijsko neustrezna protiinformatičnost, ki ne generira intencionalne, pričakovane informacije za določen primer.

ALGEBRA je veja matematike, ki raziskuje abstraktne strukture z operacijami. Operacija je informacija, s katero se elementi strukture algebraično preoblikujejo. Algebra je poseben aksiomatičen sistem, torej natanko formalizirana, abstrahirana, simbolizirana informacija. Informacijska algebra bi lahko bila le dovolj nenatančen, ohlapen, mehko formaliziran sistem informacijskih principov z nastajalno (odprto, generativno) opredeljenimi informacijskimi operatorji in operandi.

ALGORITEM je značilen primer do potankosti določene, nespremenljive informacije. Algoritem je absolutno statična informacija, torej informacijska konstanta. Matematične funkcije in konstrukti, današnji računalniški programi, procedure za reševanje nalog so značilni algoritmi. Namen algoritma kot informacije je, da je vselej in kjerkoli do potankosti enako razumljen (informacijsko standariziran).

ALIENACIJA je pripadanje nečemu drugemu namesto samemu sebi, je osebnostna informacijska motnja (odsotnost pozornosti, določene zavesti; odtujenost samemu sebi, ki povzroča konformno ali ideološko obnašanje bitja; razosebljenost, birokratska miselnost). Nasprotje vsake alienacije je nova, izvirna, individualna informacija, ki je odtujena prav obstoječemu informacijskemu ozadju (okolju, prevladujočemu ozadju, kulturništvu). Ideologija kot okoliška, kulturna informacija ni odtujena le bitjevski informaciji temveč tudi različnim drugim ideologijam. Alienacija kot informacijsko sprejemanje tuje, okoliške, nelastne informacije zaznava lastno, izvirno, za bitje bistveno informacijo kot informacijski šum (ki ga ni vredno razpoznavati).

ALOCENTRIČNA informacija izhaja iz tujih (nelastnih, neindividualnih) izhodišč, upošteva predvsem okoliško informacijo, je torej nasprotje egocentrične informacije. Značilni primeri alocentrične informacije so ideologija, znanost, kultura.

ALOGIČNA informacija ne upošteva logike, je protilogična in informira (informatizira) izven logičnega območja, ker logiko kot informacijo presega, spregleduje njen logično (doktrinarno) zoženi informacijski prostor. Primeri alogične informacije so ustvarjalnost, inteligenca, estetika, etika.

ALOPATRIČNA informacija je informacija tesne povezanosti bitij neke populacije ali vrste, čeprav so ta bitja geografsko razpršena.

ALTERNATIVA omogoča izbiro vsaj med dvema možnostima. Alternativna informacija je izbirna, spontano odločitvena v informacijskem prostoru, tista ki izbere alternativo.

ALTHUSSERIANIZEM nosi ime po francoskem marksističnem filozofu L. Althusserju (1918) in predstavlja doktrino (informacionizem) s štirimi glavnimi usmeritvami.

(1) Ekonomski determinizem ni podlaga modela socialne nadstavbe. Ideologija in politika sta pogoja za obstoj gospodarstva. Produkcijska oblika je kompleks odvisnosti (odnosov) ekonomije, ideologije in politike. (2) Ideologija je realen socialni odnos, praksa in ni iluzija, kot se navadno poudarja. Ideologijo je mogoče opredeljevati kot epistemološki koncept in z njenim nasprotjem z znanostjo. (3) Ideološki aparati države v kapitalizmu reproducirajo produkcijske odnose s pomočjo medijev in vzgoje. (4) Posamezniki so nosilci ali dejavniki strukture socialnih odnosov, ne pa razredi (množice posameznikov).

Kriticizem očita althusserianizmu šolski

teoreticizem, negiranje relevantne evidence, dogmatizem in izhajanje iz marksističnih principov (primarnost ekonomije). Althusserianizem je nazoren primer informacionizma, mestoma celo trdodogmatične ideologije.

ALTRUIZEM je v informacijskem nasprotju z (etičnim) egoizmom. Egoizem je hipotetična informacija, ki utemeljuje, da je moralnost mogoče pojasnjevati le z ustrezno razjasnjenim samointeresom bitja. Altruizem poudarja, da moralnosti ni mogoče reducirati na samointeres. Altruizem in egoizem sta tako protiinformacija eden drugemu, ko se dopolnjujeta in vzvratno (protiinformacijsko) pojasnjujeta. Tudi altruizem je informacionizem, ki lahko postane močan skupinski pritisk na posameznika, (npr. altruistični samomor).

AMBIGUITETA je informacijska dvomljivost, nejasnost, dvoumnost, večumnost. Večumna informacija sproži hkratno, paralelno nastajanje informacije (protiinformacije) z različnimi, tudi nasprotnimi pomeni. Posebna oblika ambiguitete je sistematična dvoumnost. Informacija ima določen pomen, če se uporabi v povezavi z eno vrsto informacije in drugačen pomen, če se uporabi z drugo vrsto informacije.

AMBILATERALNA informacija je informacijsko vplivna in vplivana informacija. Ambilateralnost je splošna informacijska lastnost.

AMBITENDENCA je ambivalenca, za katero je značilna koeksistenca reakcije in protireakcije. Primer take bitjevske informacije je manično depresivna psihoza.

AMBIVALENCA (bipolarnost) je večvrednostna, v sebi nasprotujoča, bitjevsko shizofrenična informacija, ki lahko povzroča hkrati informacijo in njeno nasprotje, tezo in antitezo, vrednost in protivrednost, je nerazrešljivi krog informacije in njene protiinformacije. Ambivalenca je npr. koeksistenca čustva, želje, impulza in temu podobno z njegovo antitezo, npr. ljubezni s sovraštvom do iste osebe.

AMBIVERZIJA je mešanica ali ravnotežje bitjevske ekstraverzije (informacijske usmerjenosti navzven, vplivnosti od zunaj) in introverzije (informacijske usmerjenosti navznoter).

AMFIBOLIJA je informacija dvoumnosti, dualnosti, večsmiselnosti, kategorialno mešana informacija (npr. empirizem in transcendenca, idealizem in materializem).

AMFILOGIJA je posebna oblika protiinformacije, ki izraža nasprotnost, dvojnost, protislovnost, spornost, negotovost.

AMIMIJA je nesposobnost izvajanja in razumevanja pomenskih gibov ali kretenj (gesturalna afazija).

AMINOKISLINA vsebuje ti. aminoskupino, 20 različnih aminokislin sestavlja peptidno ali beljakovinsko zaporedje, ki je kodirano (informacijskonosilno) z mRNA.

AMNEZIJA je informacija pozabljanja, izgube spominskih sledi, ki je lahko delna, popolna, naravna, umetna (namerna), patološka (epilepsija), začasna, retrogradna in anterogradna (nesposobnost pridobivanja novih spominov). Amnezija je lahko posledica možganske poškodbe, fokalne cerebralne lezije pa tudi histerije. Elektivna ali afektivna amnezija je pozabljivost glede na specifične dogodke zaradi podzavestne spominske represije. Epizodična amnezija je izguba spomina o važnem in pomembnem dogodku, čeprav je sicer spominska funkcija normalna.

AMORFNA informacija je na določen način neoblikovana, neizrazna, še nerazvita, nenastala, površno zarojena, značilno nerazmernostna informacija. Njeno nasprotje je dobro izoblikovana, formalizirana, natanko usmerjena, kodificirana informacija.

AMPLIFIKACIJA je ojačevanje, povečevanje, razširitev. Informacijska amplifikacija je ponavljanje, pojasnjevanje, vprašev anje, tudi samo nastajanje informacije o določenem pojmu z nastajanjem novih označevalcev, sinonimov, atributov, metafor. Informacijska amplifikacija se uporablja npr. v retoriki in literaturi kot stilistično in poetično poudarjanje in nastevanje podobnih lastnosti informacijsko obravnavanih oblik in pojavov.

ANALEPTIČNO je tisto, kar stimulira osrednji živčni sistem, še posebej z izboljšavo jakosti in živahnosti cerebralne aktivnosti. Analeptična informacija povzroča povečano in živo informacijsko aktivnost.

ANALGEZIJA je izguba občutka bolečine pri anesteziji, opitju ali prekinitvi informacijskih poti v centralnem ali perifernem živčnem sistemu.

ANALITIKA je pojav (oblika in proces) informacijske analize in analize informacije kot informacije. Analitika je poseben, v bistvu protiinformacijski (analitično vprašev alni) vidik filozofije (Aristotel, Kant, Heidegger), znanosti in seveda informacije. Protiinformacija, ki nastaja z analizo (z razčlenjevanjem, razlaganjem, razreševanjem, razkrivanjem) razstavlja celoto na dele, išče mejne oblike (konture) med deli celote, gradi logiko neke discipline, principe, spoznanja. Analiza pridobi npr. antitezo iz teze, sinteza ju kot protiinformacija analize zopet poveže (skupaj vmesti v veljavno informacijo).

ANALIZA je informacijska razčlenitev sestavljenega, npr. razčlenitev sestavljene informacije. Analiza je informacija razčlenjevanja (protiinformiranja in informacijskega vmešč anja). Uporabne oblike analize kot informacije so npr. psihoanaliza, faktorska, sistemska, statistična, strukturalna analiza, katerih narava je informacijska.

ANALOGIJA je informacijska skaldnost, podobnost, ki informira (sklepa) iz posebnega primera na poseben primer, kjer se na osnovi podobnosti primerjajo, ocenjujejo lastnosti enega in drugega. Analogija je za razliko od strogega (deduktivnega, induktivnega) sklepanja spontan način sklepanja z nezanesljivo, nenatančno in nedokazano informacijo. Informacijska analogija je pomenska podobnost (sinonimnost, povezanost) informacijskih enot.

ANAMNEZA je odbiranje, prebiranje, rekolekcija, spominjanje, reminiscenca (klicanje v spomin) in odtod razpoznavanje in argumentiranje informacije z obstoječo informacijo. Asociativna anamneza je (psihiatrično) povezovanje simptomov s čustvenim življenjem.

ANARHIJA je vobče informacijska nepovezanost, nerazmernost, neodvisnost, posameznost, svobodnost, neusmejena spontanost v oblikah in procesih informacije. Anarhična informacija ni samo nepovezana, lahko je tudi v sebi informacijsko razpadla ali razpadajoča informacija, ki se iz urejenosti (razmernosti) razvija v nepredvidljivo neurejenost ali drugačno urejenost.

ANATOMIJA je znanost o strukturi in organizaciji organizmov, naprav, telesa, je morfologija (študij oblik) in vzorcev organizmov, je raztelesevanje. Anatomija informacije je nauk o informacijski strukturi in organizaciji, o informacijskih oblikah in procesih. Anatomija je umetna, modelna, primerjalna, opisna, razvojna, funkcionalna, fiziološka, patološka, sistematična itd.

ANATOMIZEM je teoretični koncept, po katerem je pojavnost celote odvisna od njene strukture, npr. pojav življenja organizma od njegove sestavljenosti, pojavnost umetnine od njenih podrobnosti. Anatomizem je tedaj informacija pojasnjevanja celote z njenimi deli, z njeno posebno anatomsko strukturo. Anatomizem je značilni informacionizem.

ANESTEZIJA je pojav omrtvičenosti, delne ali popolne izgube zavesti, izgube občutljivosti v omejenem čutilnem predelu. Spontana informacija bitja je omrtvičena (nastajalno blokirana) pod vplivom anestetikov, hipnoze ali ideoloških (magijskih) diskurzov.

ANIMATIZEM je personifikacija neživih predmetov, ki se jim pripisujejo duhovne (informacijske) kvalitete človeških bitij.

ANIMIZEM je prepričanje (informacija verovanja), da so nekateri neživi predmeti ali sile (sonce, veter, voda) živi in imajo svoj namen in namero (intenco).

ANKSIOZNOST je bitjevska informacija strahu kot iracionalnost in fobičnost, ki nastaja z informacijo predmetov in položajev, kot so npr. odprti in zaprti prostori (agorafobičnost, klaustrofobičnost), višine, pajki, kače, grmenje, potovanje, gneča, tuje osebe itd. To je strah izven razmerja dejanske nevarnosti. Anksioznost je informacija nespoznanega v sebi ali v okolju, ki nastaja s spremembami v okolju ali z informacijskimi premiki v podzavesti kot posledicami lastnih potlačenih refleksov. Psihoanaliza loči vrsto različnih oblik anksioznosti: primarna, signalna, kastracijska, separacijska, depresivna, paranoidna, objektivna, nevrotična, psihotična. Fobije imajo svoj izvir v histeriji.

ANOEZIJA je nekognitivna zavest, je čustveno stanje brez spoznavanja ali brez reference na kakršenkoli objekt. Anoezija je primer informacijske (kognitivne) slepote.

ANOMALIJA je odstopanje strukture, funkcije, stanja, informacije od normativnosti, zakonitosti, pravilnosti, je nepravilnost v razvoju. Informacijska anomalija je bistven protiinformacijski mehanizem, s katerim nastaja nova, nepredvidljiva, informacijsko nenormativna informacija. Anomalija je odstopanje od normalne informacije, je informacijski razvoj.

ANOMIJA je informacija socialnega stanja, za katerega je značilen ti. prelom norm (normna prelomnica), ki vpliva na socialno interakcijo. Anomija je prelomna informacija, ki prekinja določeno informacijsko (ideološko, doktrinarno, konformno, populacijsko) slepoto. V tem pomenu je anomija podobno kot alienacija zamisel, s katero se premoščuje razlika med pojasnjevanjema socialne akcije na individualni ravni in na ravni socialne strukture. Klasično zamisel anomije je dal že E. Durkheim v svojem delu o samomoru (1897). Po njegovi predpostavki so ljudje lahko srečni le, če so njihove želje proporcionalne njihovim možnostim (sredstvom, povprečjem, realitetam). Človekove potrebe, ki so prepuščene same sebi, se razvijajo informacijsko spontano (bitjevsko neomejeno, metafizično svobodno, življensko naravno). Ta narava človekove informacije oblikuje v povezavi z informacijo nujnostno omejenih možnosti informacijo nesrečnosti in kot skrajno posledico suicidalno informacijo. Anomija je informacija zloma tistega okvira, v katerem cilji presegajo realiteto in začenja nastajati samomorilska informacija.

R. K. Merton (1957) je Durkheimovo zamisel anomije razširil v splošno teorijo deviantnega obnašanja. Socialni sistemi se medsebojno razlikujejo predvsem v poudarjanju kulturnostno opredelejenih ciljev na eni in institucionalnih možnosti za doseganje teh ciljev na drugi strani. Nastajanje deviantne informacije je tako normalna posledica prevelike razlike, ki nastaja zaradi neustreznega poudarjanja enega od obeh izhodišč v vsakokratnem realnem okolju (npr. pojavi kriminalnosti, desidentstva, radikalnega upora, nezakonitosti).

ANTAGONIZEM je spopad, tekmovanje enega z drugim, je protireakcija, nasprotje, spor, upor, upornost. Antagonizem je regularen informacijski (naraven, zgodovinski, socialen) pojav kot komponenta protiinformacije in kot protiinformiranje. Antagonizem nastaja iz informacije kot informacija nasprotovanja, ki producira nastajajočo informacijo in informacijo nastajanja. Antagonizem je osnovni princip dialektike, živega informacijskega procesa. Bakterijski antagonizem preprečuje rast drugih bakterij s prehranjevalnim tekmovanjem ali z izločanjem strupov (črevesni trakt). Antagonizirati pomeni nasprotovati, delovati proti čemu, tekmovati s kom ali s čim.

ANTANALGEZIJA je povečana občutljivost za bolečino zaradi učinkovanja nekaterih drog, ki znižujejo prag bolečine. Informacija bolečine se pri antanalgeziji lahko pojavlja prej in češče kot v normalnem stanju občutljivosti.

ANTECENDENCA je informacijska pogojnost, ki izjavlja, da določena informacija velja ali bo veljala le v primeru (pri pogoju), ko je ali bo obveljala neka druga informacija. Antecendenca je tako primer pogojne ali hipotetične informacije, v kateri se pojavlja značilna predpostavka, ki začenja z besedo "če" ali "ko." Antecendenčna informacija ima vsaj dva dela, in sicer pogojni ali antecendenčni del (protasis) in posledični ali konsekventni del (apodosis). Antecendenca se uporablja kot metoda dokazovanja v formaliziranih informacijskih sistemih (v teorijah, ekspertnih sistemih).

ANTICIPACIJA je informacija vnaprejšnjega, nekritičnega, neraziskovalnega, nespoznavnega sprejemanja informacije kot veljavne (upravičene), pri čemer se bo njena veljavnost dokazovala ali izkazala (sama po sebi) kasneje. Vzroki za pojav anticipacijske informacije so ideološki (psihološki, nazorski, proleptični), disciplinarno logični (filozofski, znanstveni) ali preprosto epikurejski.

ANTIGEN (imunogen) je substanca (beljakovina, ogljikov hidrat, maščoba, nukleinska kislina, druga velemolekula), ki lahko povzroči v gostitelju nastanek specifičnega protitelesa ali posebne populacije limfocitov, ki s to substanco reagirajo. Antigen je lahko genetični ali drugačni tujek za določenega gostitelja. Za ustrezno reakcijo mora imunski sistem razpoznati vrsto antigena. Tu ima antigen vlogo informacije, ki jo informacijsko razpoznava imunski sistem.

ANTILOGIJA je protirazumska, protislovna, oprečna informacija, je kot antiteza ideologije, dogme, teze, teorije, ki dokazuje, da je k vsaki informaciji mogoče najti tudi raznovrstno protiinformacijo (npr. vzrok in

protivzrok, dokaz in protidokaz). Antilogija kot informacija potrjuje informacijsko relativnost, kjerkoli se informacija pojavlja (filozofija, znanost). Antilogija je razumljiva iz predpostavke, da nobenega pojava ni mogoče v celoti razpoznati po logični poti.

ANTINOMIJA je informacija protipravilnosti, protislovnosti, informacijske nasprotnosti, protiinformacijskosti informacije v sami sebi. Antinomija lahko povzroči več primerov informacije, ki so medsebojno protislovni. Značilni primeri antinomij so paradoksi in dialektični diskurzi.

ANTIPATIJA je značilna bitjevska, čustveno pogojena informacija odbojnosti do drugih bitij. Antipatija je v sebi nedoločna (apriorna), spontana, refleksna informacija.

ANTIPERISTAZA je možnost nastajanja protiinformacije v informacijsko (ideološko, dogmatsko, navidezno) zapolnjenem prostoru.

ANTITETIKA je metodičen postopek pojmovnega nasprotovanja s prepričanjem, da so nasprotja imanentna vsebini nekega procesa. Antitetika pojmov je značilna za Kantovo (antinomija čistega uma) in Heglovo filozofijo (kategorialna dedukcija). Informacijska antitetika je del protiinformiranja, in to tisti del, ki sistematično išče prav možna nasprotja k trenutni informaciji.

ANTITEZA je nasprotna trditev, nasprotje dveh pojmov, informacija nasprotnosti. Nasprotje je lahko kontradiktorno ali kontrarno. Kontradiktorno nasprotje dano trditev negira, antiteza pa izhodiščni tezi nasprotuje tako, da predpostavlja, da je mogoče iz njune nasprotnosti (kontrarnosti) izpeljati sintezo kot tretji pojem, ki postane del novega spoznanja. Protiinformacija pa ni samo antiteza in sinteza (informacijsko nasprotovanje in informacijska vmestitev), temveč je lahko povsem (vsa, katerakoli) druga, s tezo nerazmernostna (nedialektična) informacija.

ANTONIM je beseda z nasprotnim pomenom (npr. dobro, slabo). Homonim je beseda z različnimi pomeni (npr. prst na roki in prst kot zemlja). Sinonim je ena od dveh ali več besed z enakim pomenom (npr. menim, mislim, sodim). Tudi informacija je lahko antonimna, homonimna in/ali sinonimna.

ANTROPOFOBIJA je informacija strahu pred socialnimi odnosi (strah pred ljudmi), katere posledica je npr. samotarstvo.

ANTROPOINFORMATIKA raziskuje informacijske procese človeka kot zavestnega bitja, zlasti višje korteksne informacijske funkcije, mehanizme spoznavanja in razumevanja, razvoja inteligence kot informacije, reševanja problemov, učenja, spominjanja. Na teh raziskovalnih področjih se povezuje s kognitivno filozofijo, kognitivno psihologijo, nevrolingvistiko, nevrofiziologijo, biologijo, biofiziko, biokemijo itd.

ANTROPOLOGIJA je znanost o človeški vrsti in njenih bližnjih prednikih s posebno pozornostjo na spremenljivosti in razvoju človekovih značilnosti in navad v različnih populacijah in okoljih. Kulturna antropologija raziskuje skupinske značilnosti človeka, ki se pridobivajo in prenašajo z učenjem, kot so socialna organizacija, tehnologije, jeziki, navade, običaji, vera, umetnost itp. Patološka antropologija preučuje človekove bolezni glede na diferencialno porazdelitev med človeškimi skupinami. Socialna antropologija se ukvarja s človeškimi družbami s posebnim ozirom na rodovne sisteme, družbene vloge, rodovno organizacijo, strukturo skupnosti, razrede, kaste, politične, gospodarske, verske ustanove itd. Antropologija je pomemben izvor pri študiju človekove informacije.

ANTROPOMORFIZEM je informacija prirejanja človekovih fizičnih, duševnih, ali čustvenih značilnosti živalim ali drugim nečloveškim entitetam.

AORISTIJA je načelo skeptične neodločnosti, nesklepanja, ko informacijske razlike niso dovolj izrazite, določljive, ko so nejasne, neznane ali neobstojne.

APARAT je zapletenost strukture in/ali organizacije, kot je naprava, mehanizem, stroj, celica, tkivo, organ, bitje, institucija, informacija (dokumentacija, literatura, znanost itd.) Aparat je informacija o in v določenem sistemu, ki ta aparat opisuje, vzdržuje njegovo delovanje in nastajanje. Informacija v širšem pomenu se razumeva kot informacijski aparat, ki sproža nastajanje, sprejemanje, preoblikovanje, razumevanje, skratka informiranje in informatiziranje informacije kot informacije. Filozofija informacije je razmisljanje o informacijskem aparatu skozi njegovo mogočo zapletenost in prepletenost.

APATIJA je nenavadna informacijska blokiranost, problemska neobčutljivost, nemočna otopelost, nezdrava ravnodušnost, demoraliziranost v primerih, ko se normalno pričakuje nastajanje bitjevske problemskoreševalne in odločitvene informacije. Apatija je pomanjkanje čustvene informacije, je posledica razočaranja, izgube upanja in dolgočasja (morije) zaradi blokiranja ekspresivne obnašanja bitja. V ekstremnih primerih je lahko apatija obramba, ki rešuje življenje. Apatija je informacija, ki postavlja totalno informacijo bitja, njegovo metafiziko v nevtralno stanje zaradi razočaranja, izčrpanja določenih možnosti, nepravičnega in neupravičljivega brezupja.

APERCEPCIJA je prilagojevanje (informacijsko vmeščanje, filozofsko pojmovanje, percepcija, razumevanje) novega izkustva (protiinformacije) celoti že pridobljenih izkustev (obstoječi informaciji), tj. ti. aperceptivni masi znanja, miselnih navad, obnašanja in povezavi novega in starega izkustva v novo celoto. Apercepcija je informacija, ki novo informacijo tudi hermenevtično (ali semantično) povezuje (pojasnjuje) z obstoječo informacijo. Tako prihaja do višje informacijske (pomenske) sinteze, do rasti in raznovrstnosti pomena kot informacije.

APODIKTICNA informacija je neizpodbitno dokazana, neovrgljiva, trdna, nujna, brezpogojna (npr. vsak krog ima središče).

APOLOGIJA je obrambna informacija, s katero se brani obstoječa, (pri)znana informacija proti novi, (ne)predvidljivi informaciji. Z apologijo se brani vera ali ideologija proti spontano nastajajoči informaciji, ki (ne)predvidljivo integrira ali razkraja informatičnost, veljavnost, urejenost vere ali ideologije.

APOPLEKSIJA je nenadna epizoda oslabitve cerebralne funkcije (npr. zavestne informacije), ki vodi v komo (popolna nezavest) in je vobče posledica cerebrovaskularnega akcidenta (krvavitve). Apopleksija je nenadoma nastala oslabitev informacijske funkcije.

APORETIKA je informacija raziskovanja nerešljivih ali težko rešljivih problemov neglede na njihovo možno rešitev. Aporetika (N. Hartmann) je le ena od faz na poti do teorije (fenomenologija, aporetika, teorija), s katero

se določen problemski prostor identificira, odpira, primerja, raziskuje, problematizira, paradoksira v območju dejstev. Nasprotja in protislovja, ki se pri tem pojavljajo, spadajo že v območje kasnejše teorije. Aporetika je v svojem bistvu protidogmatična, problemska in je v nasprotju s sistematičnostjo.

APORIJA je informacija dvoma, pomisleka, brezpotja, brezizhodnosti, nezmožnosti, antinomičnosti, paradoksalnosti kakršnekoli rešitve nekega vprašanja, je omahovanje med različnimi nasprotnimi preudarki. Aporija je primer protiinformacije, ki nastaja aporetično, cirkularno spontano v okviru aporetike.

APRAKTOGNOZIJA je nesposobnost uporabe predmetov zaradi pomanjkljivega razpoznavanja njihove narave in funkcije. Geometrijska apraktognozija je izguba sposobnosti razpoznavanja in interpretiranja geometrijskih oblik zaradi lezije temensko-zatilnične regije, ki se kaže kot konstrukcijska apraksija, ki je lahko povezana še z barvno agnozijo, aleksijo in hemianopijo. Apraktognozija je pomanjkanje določene razpoznavne informacije, je torej specifično pomanjkanje opazovanja, raziskovanja in spoznavanja druge informacije, je nenormalna protiinformacijska deficienca.

APRAGMATIZEM je izguba sposobnosti izražanja misli in akcije ali razumevanja narave miselnega in fizičnega akta. Izvori apragmatizma kot informacije so lahko lezijski ali ideološki.

APRAKSIJA je neparalitična nesposobnost opravljanja spretnostnih, namenskih in koordiniranih motornih dejavnosti. Pri neposkodovanih (brezhibnih) motornih gibalnih poteh je oslabljena konceptualizacija gibov. Apraksija označuje navadno popolno izgubo in ne le oslabitev določene sposobnosti. Apraksija je lep primer pomanjkanja nastajanja informacije (konceptualizacije) za gibalno akcijo. Agnozijska apraksija je nesposobnost gibalnega izvajanja zaradi nerazpoznavanja in neinterpretiranja tiste senzorske informacije, od katere je gibalno izvajanje odvisno.

APRIORIZEM je informizem v okrilju bitjevske, populacijske, kulturne informacije, ki postavlja opazovalno, raziskovalno ali spoznavno informacijo izven izkustvenega informacijskega konteksta s predpostavko, da se razumevanje kot informacija lahko pojavi tudi pred izkustvom kot obnašanjem bitja. Apriorizem kot informacija je nasprotje empiricizma. Pojav protiinformacije je lahko prav aprioristično (spontan, samovoljen, ideološki, aksiomatičen, nepredvidljiv).

APTITUDA (nadarjenost) je konstitucijski substrat ali inherentni potencial, ki omogoča bitju pridobivanje posebnih oblik spretnosti, znanja (npr. proizvajanja glasbe) skozi vajo. Aptituda je primer nastajanja in vmeščanja (krožne vaje) informacije z namero, da se doseže (oblikuje) določena sposobnost kot informacija. Vokacionalna aptituda je sposobnost individua, da se nauči spretnosti določenega poklica.

AREST je ustavitev, upočasnitev, oviranje, vmešavanje v proces normalnega, spontanega informacijskega nastajanja, preden se je nastajanje samo po sebi končalo. Razvojni arest je začasna ali stalna ustavitev razvojnega procesa, ki je lahko regularen ali abnormalen razvojni vzorec. Maturacijski arest je razvojna napaka krvnih celic pri akutni levkemiji. Metafazni arest je ustavitev celičnega cikla v stadiju delitve celičnega jedra. Aretirano je zakasnjeno ali prekinjeno v procesu informacijskega nastajanja.

ARGUMENT je dokazovalna informacija, ki vsebuje zanesljivost dokaza. V matematiki je argument objekt funkcije ali argumentna informacija, ki se preoblikuje z uporabo (izvajanjem) funkcijske informacije. Argument je tako vobče informacija, ki je lahko preoblikovana z drugo informacijo, ki pa lahko prav zaradi svoje zanesljivosti (aktualnosti, argumentiranosti) sproži preoblikovanje (informatiziranje) druge informacije. Argument je kot informacijski objekt in informacijski subjekt, torej relevantna informacija.

ARHAIČNO je začetno, staro, starinsko, starodavno, nekdanje, predniško, podedovano. Informacija je arhaična, če ima omembe vredno evolucijsko starost. Dedna informacija je lahko arhaična. Informacijska arhaičnost nakazuje razvojnost informacije iz informacije. Tudi današnja informacija nosi v sebi npr. obrise antične ali staroegiptovske informacije. Filozofija kot informacija se večkrat vrača k prvobitnosti (primordialnosti) pojmov in njihovih pomenov, npr. ko postavlja vprašanja o logiki, biti, jeziku, ko išče prvobitnejšo, arhaično informacijo glede na neko današnjo rabo.

ARHIKORTEKS je filogenetsko najstarejši del cerebralnega korteksa in najzgodnejši, ki ga je razvojno še mogoče diferencirati. Njegovi glavni konstituenti so hipokamp, nazobčani zavoj in subikul.

ARISTOTELIZEM je informacija (ideologija), ki obsega pojmovno razširjenost in raziskavo aristotelske doktrine kot tudi premišljeno privzemanje, prakso in seveda prevrednotenje (perverzijo) te doktrine (npr. grško-evropsko in arabsko-srednjevzhodno).

ARITMIJA je variacija glede na normalni, regularni ritem. Živa informacija je čestokrat izrazito ritmična (utripanje srca, dihanje, dnevni, biološki ritem), informacijsko nastajanje pa je tudi aritmično, čeprav je del spontano cikličnega (ritmičnega) informiranja. Aritmičnost informacije je v določenem pomenu njena neregularnost, odstopanje od pričakovanega ali napovedljivega. Perpetualna aritmija je peramnentna abnormalnost srčnega ritma.

ASEMAZIJA je nesposobnost komuniciranja z govorom ali kretnjami.

ASEMIJA je nesposobnost (grafična, mimična, verbalna itd.) uporabe ali razumevanja znakov, kretenj, pisanega ali tiskanega jezika. Ta nesposobnost je posledica korteksne lezije ali pa je izvorno psihogena. Je kombinacija amimije in aleksije.

ASIMILACIJA je bitjevski informacijski proces vključevanja (informacijskega vmeščanja) novih izkušenj (protiinformacije) v obstoječe kognitivne strukture (v spoznavno informacijo). Z asimilacijo bitje ne spreminja le svoje informacije, temveč hkrati informatizira okolje z lastno informacijsko strukturo. Asimilacija je tudi učenje, ko se nova informacija o objektih, osebah, dogodkih interpretira in razumeva v okviru obstoječega znanja in shem. Genetska asimilacija je izbira, ki je posledica genetske fiksacije fenotipa, ki je izgledal prvotno prednosten le v omejenem okolju.

ASIMPTOTIČNA informacija se lahko pomensko približa dani informaciji poljubno natančno, vendar je nikoli pomensko ne doseže. Asimptotična informacija je le približek k dani informaciji. Senzorska informacija je približek opazovane realitete. Prevod iz enega naravnega

jezika v drugi je lahko kvečjemu asimptotičen (natančen prevod vobče ni mogoč). Osebni pomen, ki je hermenevtičen, je mogoče le asimptotično prevesti v skupinski (populacijski, kulturni) pomen, ki je semantičen.

ASKETIZEM je doktrina (ideologija) ali praksa (obnašanje), ki negira čutno ugodje (hedonizem) in poudarja duhovni, tudi ustvarjalni jaz. Asketizem je kot usmerjevalna informacija sprejet od vrste pomembnih religij. M. Weber poudarja usodno pomembnost protestantskega asketizma za nastanek kapitalizma, za delovno disciplineranost in za kapitalistično organizacijo. Asketizem je tudi etični princip in je lahko pomembna informacija za preživetje.

ASOCIACIJA je informacijski proces stikanja, spajanja, sestavljanja, paralelniziranja in posledičnosti informacije. V ti. asociacijskem korteksu se informacija abstrahira, navzkrižno povezuje in interpretira. Asociacija je informacija, ki lahko vzbudi, izzove ali obnovi drugo informacijo. Informacijska asociacija lahko nastane zaradi informacijske podobnosti (konformizma) ali nasprotnosti (protiinformacijske spontanosti), je posledica informacijske dispozicije bitja in vplivov njegovega informacijskega okolja. Vzvratna asociacija se uporablja pri verbalnem učenju in pomeni oblikovanje vezi med delom v asociativnem zaporedju in delom (informacijo), ki začenja zaporedje, ki se ga želi naučiti. Zvočna asociacija je pomik v vsebini neke misli, ki nastane zaradi zvenenja neke besede in ne zaradi njenega pomena. Sanjska asociacija nastane s svobodno interpretacijo sanj. Svobodna asociacija nastaja v mišlenju iz vsakdanjih, nepomembnih, naključnih dogodkov.

ASOCIACIONIZEM je doktrina (informacionizem), da je asociacija osnovni princip duševnega življenja. Celotno psihološko izkustvo vključno z višjimi miselnimi procesi naj bi bilo kombinacija enostavnih idej, ki nastajajo iz čutilnega izkustva.

ASIMBOLIJA je nesposobnost razumevanja in interpretiranja simbolov, kot so besede, števila, kretnje, znaki, diagrami, note ali druga senzorska informacija (aleksija, amimija, amuzija, agnozija). Bolečinska asimbolija (nezaznavanja bolečine) lahko nastane zaradi poškodb senzorskih poti.

ATAKSIJA je neobstojnost, nekoordiniranost in neorganiziranost gibalne (motorne) informacije pri odsotnosti paralize, izguba normalnega cerebralnega vpliva na motorno aktivnost.

ATAVIZEM je informacija nerazvitosti ali informacijsko obnašanje davnih prednikov, davne preteklosti, primitivnosti, ki je v nasprotju z višjim razvojem, s pričakovano sodobno informacijo. Atavizem je lahko posledica dednih faktorjev, kot so genska recesija, rekombinacija in modifikacija ali okoliški faktorji.

ATITUDA je informacijsko relativno stabilen (objektni) sistem prepričanj (drža, obnašanje), s katerim se ocenjuje določen objekt (npr. tudi v psihologiji ali sociologiji). Skozi atitudo je mogoče opazovati, raziskovati in spoznavati obnašanje ljudi. Informacijska atituda pomeni opazovanje določene fenomenologije kot informacijsko značilnost, posebnost, nastajalnost.

ATOMIZEM je prepričanje (doktrina), ki trdi, da je celota (struktura) vselej sestavljena iz delov (atomov). Atomizem je bistvena znanstvena informacija, ki omogoča, da se integralne lastnosti objektov razumevajo kot konstrukcije osnovnih lastnosti objektnih delov. Tako je spremembe različnih pojavov mogoče razumeti metafizično kot preurejanje trajnih delov namesto pojavnostnega kreiranja in destrukcije. Seveda pa se lahko pokaže, da je tudi del struktura osnovnejših oblik ali osnovnejših procesov. Informacija kot razvijajoč pojav ni atomarna v atomističnem pomenu, ker v njej druga informacija ne informira (ne informatizira) kot njen del in tudi sama informacijsko (delno) vpliva na drugo informacijo. Informacija kot celota je hkrati del sebe in druge informacije (lastnost informacijske cirkularnosti).

ATRIBUT kot informacija izraža bistveno lastnost neke oblike ali procesa, ki jo je mogoče domišljati samostojno (vzporedno ali dodatno k obliki ali procesu). Seveda pa atributska informacija ni nespremenljiva, za vselej bistvena (Descartes, Spinoza).

ATROFIJA je pojemanje, hiranje, slabljenje žive celice, tkiva, organa, organizma, misli, informacije po določenem obdobju razvoja oziroma zorenja. Cerebralna atrofija nastane s hiranjem živčnih celic v cerebralnih poloblah.

AVDICIJA je informacija, proces, uporaba, občutek poslušanja. Miselna avdicija je halucinacija, v kateri se lastne misli izgovarjajo.

AVERZIJA je močna odbojnost in neprijetno občutje, sprožено s posebnimi dražljaji, informacijo, z reakcijo umika, preklica, bega ali z izogibajočim obnašanjem. Averzija je neprijetna, odbojna, kritična informacija, ki obravnava npr. prepovedano, krizno, javno ali zasebno nedotakljivo informacijo (tabuje, mite, resnice, neresnice). Averzija je tudi zavračanje informacijsko represivnega, zaostalega, arhaičnega okolja oziroma socialnega sistema (družine, klana, šole, podjetja, naroda itd.)

AVRA je senzacija, občutek pred epileptičnim napadom, histerično manifestacijo, je del tega napada kot svarilna informacija. Obstaja več vrst avre, kot je čustvena (strah, bojazen), senzorska (vidna, slušna, kretenjska, vohalna, somatična), trebušna itd. Avra so tudi svarilni in opozorilni signali pred napadom migrene, kot so vizualni, senzorski in motorni simptomi.

AVTENTIČNO in neavtentično sta eksistencialna pojma, ki se uporabljata za razlikovanje med dejanji dobrega in slabega (zaupanja, verovanja), pristnega in nepristnega, resničnega in lažnega glede na bitjevsko informacijo. Neavtentična informacija (tudi mimkrija) bitja se interpretira kot defenzivna, v kateri so realna občutja in motivi različni od bitjevsko izvirnih in nepristni z namenom, da se bitje izogiba tistim informacijskim stanjem, v katerih bi lahko nastajala informacija strahu.

AVTIZEM je informacija miselne in obnaševalne preokupacije s samim seboj z izključitvijo zunanjega sveta, ki se kaže subjektu kot nerealen. Avtistična informacija nastaja kot posledica izključno subjektivnih namer in fantazem brez upoštevanja realnosti. Infantilni avtizem je otroška psihoza, ki ji manjka sposobnost zaupanja in komunikacije, z mutavostjo in govornimi motnjami.

AVTOANALIZA je npr. informacijska samoanaliza v okviru nastajajoče informacije. Informacija je po svoji naravi avtoanalitična, ko generira protiinformacijo dane informacije in jo pomensko vmešča v izhodiščno informacijo.

AVTOANAMNEZA je informacija klinične zgodovine, ki jo izraža bolnik npr. v obliki psihiatrične zgodovine.

AVTOGONIJA je samorojevanje, samoprodukcija, avtopoiesis, abiogeneza. Narava informacije je avtogonična, vendar je informacija tudi alopoietična, vplivana od zunaj pri svojem nastajanju.

AVTOGNOZIJA je informacija samoznanja, znanje o samemu sebi, ki nastaja npr. s psihoanalitično interpretacijo.

AVTOHTONA informacija ni vplivana bistveno z drugo informacijo. Pri določenih pogojih je živčna aktivnost lahko avtohtona, sama sebi pripadajoča, od zunaj nevplivana.

AVTOKATARZA je prezračevanje, očiščevanje lastnih misli in čustev kot način izpraznitve, izločanja, odlaganje neprijetne, neprimerne informacije. Določena umetniška in znanstvena dejavnost je povezana z avtokatarzo, tudi pisanje dnevnika je informacijsko razbremenjevanje, očiščevanje.

AVTOMATIZEM je lastnost (informatizem) samodejno uravnavanega, avtonomno upravljanega, samokrmiljenega procesa, na potek (na izvajanje) katerega ni mogoče vplivati izven domene njegove avtomatičnosti, tj. izven značilne lastnosti avtomatizma. Bitjevski informacijski procesi (polzavestni in življenskotemeljni) so občasno avtomatični (npr. refleksno reaktivni). Avtomatizem kot informacija je predvidljivost, opredeljivost, napovedljivost, ritmičnost, harmoničnost. V živi naravi niso znani primeri čistega avtomatizma, saj je živo podrejeno principu nastajanja (npr. avtopoezija). Avtomatizem je lastnost samodejnih človekovih strojev (robotov). Avtomatična informacija bi bila npr. ponavljanje informacijskih vzorcev, njihovo predvidljivo sestavljanje in prepletanje, računalniški program, pa tudi ideološki diskurz.

AVTONOMIJA je samostojnost, samozadostnost, samoproduktivnost, neodvisnost, ločenost, zaokroženost informacije kot celote, njene sestavljenosti in prepletenosti. Vladajoča ideologija je nazoren primer avtonomne informacije, njene dominirajoče samozadostnosti, imperativnosti, evdamonizma, utilitarizma, namenske zaprtosti. Nasprotje avtonomne informacije je heteronomna, povsem odprta informacija.

AVTORITARNOST je informacijska predispozicija bitja, ki omogoča sprejemanje antidemokratičnih ali tudi povsem nerazumnih prepričanj (npr. fašistična in stalinistična ideologija).

AVTORITETA je informacija uglednosti, veljave, vzornosti, verodostojnosti, poroštva, vpliva, premoči, dominacije zaradi spoštovanja, strahu, uspešnosti, konformnosti, ideološkosti, poudarjanja, prepričljivosti, moči informacije, ki se pojavlja v nasprotju (kot nasilje) z (nad) individualno (metafizično) ali skupinsko (kulturno, populacijsko) informacijo. M. Weber razlikuje legalno-racionalno, tradicionalno in karizmatično avtoriteto. Legalno-racionalna avtoriteta se utemeljuje informacijsko kot poslušnost skozi formalna pravila, ki so nastala z regularnimi, javnimi procedurami. Značilni primer te vrste je tudi formalna (zakonita) birokracija, ki lahko postavlja kvazilegalne norme. Tradicionalna avtoriteta (npr. racionalistična tradicija) ohranja pravila navad in arhaične prakse. Karizmatična avtoriteta temelji na poslušnosti določene kategorije pripadnikov (idejnih slednikov), ki verjamejo, da je značaj njihovih vodij poseben, tako da njihova avtoriteta lahko povzroča transcendenco obstoječe ali običajne prakse.

AVTOSUGESTIJA je pojav samoprepričevalnosti, ideološkega utrjevanja, vsebinskega in oblikovnega dopolnjenvanja informacijske pomenskosti s položaja pomena same informacije. Avtosugestija je iterativno zaokrožena samoinformacija, ki vedno znova, ponavljajoče poudarja pomebnost lastnega pomena. Bistvena ideologija se lahko vzdržuje le avtosugestivno (v filozofiji, znanosti, praksi).

1

(DOS) N. Abercrombie, S. Hill, B. S. Turner: Dictionary of Sociology. Penguin Books. Harmondsworth, Meddlesex, England, 1986.

(FIR) V. Filipović (redaktor): Filozofijski rječnik. Nakladni zavod Matice hrvatske. Zagreb, 1984.

(DPH) A. Flew (Editor): A Dictionary of Philosophy. St Martin's Press. New York, 1984.

(PNS) E. R. Kandel and J. H. Schwartz: Principles of Neural Science (Second Edition). Elsevier. New York, 1985.

(MDL) A. Đ. Kostić (urednik): Medicinski leksikon. Medicinska knjiga. Beograd, Zagreb, 1981.

(IMB) S. I. Landau (Editor): International Dictionary of Medicine and Biology (in Three Volumes). J. Wiley & Sons. New York, 1986.

(ADP) C. Rycroft: A Critical Dictionary of Psychoanalysis. Penguin Books. Hormondsworth, Middlesex, England, 1986.

(OWI) A. P. Železnikar: On the Way to Information. Informatica 11 (1987), No. 1, 4-18.

(ID1) A. P. Železnikar: Information Determinations I. Informatica 11 (1987), No. 2, 3-17.

(POI) A. P. Železnikar: Principles of Information. Informatica 11 (1987), No. 3, 9-17.

(ID2) A. P. Železnikar: Information Determinations II. Informatica 11 (1987), No. 4, 8-25.

REPORT OF A JOURNEY

Parsys Expeditions to New Worlds II

Anton P. Zeleznikar
Iskra Delta, Ljubljana

2. How to Exchange Information in USA?

Already in autumn 1986, entering into the design of the Parsys system and its conception, it became evident that concepts, algorithms, and technological background had to be put under critical enlightenment. Mr. Petar Brajak developed the first outlines, schemes, and procedures concerning efficient parallel communication and organization of a possible parallel computer system on the conceptual level. Later on, professor Saša Prešern joined the project as its manager. In winter 1986-1987 and during spring 1987, all negotiations for a long tour visit to US universities, institutions and enterprises were performed satisfactorily.

The goal of this exhaustive tour was to verify the Parsys conceptual background and to exchange experiential information with research and industrial groups dealing with parallel computer systems. This verification and negotiation tour embraced Laboratory for Computer Science at Massachusetts Institute of Technology (MIT), Ultracomputer Research Laboratory within Courant Institute of Methematical Sciences at New York University, Computer Science Department at Columbia University, Computer Science Department at Carnegie Mellon University, Center for Study of Language and Information of Computer Science Department at Stanford University, University of California in Berkeley, California Institute of Technology, University of California in Los Angeles, Electrical and Computer Engineering Department and Arizona Health Sciences Center at University of Arizona in Tuscon, School of Electrical Engineering at Purdue University in West Lafayette, Indiana, University of Illinois in Urbana, and Thomas Watson Research Center at New York Heights. Several computer companies were visited, such as VME Electronics in New York, Iskra Research Center in Santa Clara, VLSI Laboratory at M/A-COM Government Systems in San Diego, Fifth Generation Computing in New York, etc. and additional talks with professors from other universities were conducted.

3. The Seminars at MIT and Courant Institute

Topics of the Parsys Seminars at the MIT and the Courant Institute can be grasped from the posters presented in Figure 10 (PS) and Figure 15. The host of the MIT Parsys Seminar was professor Robert H. Halstead (Figure 8). According to the posters, Parsys was defined as a tightly coupled MIMD parallel processor development project founded by Iskra Delta Computers and carried out by several institutions and faculty research groups in Yugoslavia. Specifically, the project was involved with:

- development of prototype hardware and system software,
- development of specific programming environments, and
- development of application software.

The architectural design of Parsys was based upon already proven yet innovative concepts. The prototype machine would be a shared memory system consisting of 64 processors and 271 memory modules that were connected via a network of VLSI custom built ISMM (Intelligent Shared Memory Module) integrated circuits in a 6-cube configuration. Each ISMM supported fast routing and incorporated functions and logic that avoided or minimized degradations mostly encountered in the multiprocessor environments such as:

- n simultaneous accesses to the same memory location,
- n simultaneous accesses to the same memory module,
- n simultaneous synchronization requests,
- slow dynamic context switch, and
- huge latency.

In the presentation at both institutions, the concepts of the Parsys projects were described and some of the ISMM functions were shown.

Mr. Brajak's presentation of Parsys concepts at MIT ignited an intrinsic argumental discussion among three groups of researchers with different architectual approaches in parallel system design. Professor Halstead's group favored large grain parallelism on bus (Concert approach), a group of doctoral candidates working on the Connection Machine favored a concept of fine grain parallelism with local memory processing, and the Parsys concepts favoring medium grain parallelism with shared memory processing. A very fruitful discussion took place with some doctoral candidates working on enhancement of the routing mechanism for the Connection Machine. Parsys Routing Unit design was conceptually similar to the interconnection mechanism of the Connection Machine (both projects use n-dimensional hypercube); however, implementational concepts were completely different so that both parties were pleased and satisfied. A tour of the Connection Machine, the Concert Multiprocessor, and a pleasant dinner accompanying also professor Anton Mavretič from Boston University were delightful endings to a fulfilling day in Boston.

After a successful 45-minute seminar and a 90-minute brainstorm discussion at MIT, Mr. Brajak's talk at the Ultracomputer Laboratory at the Courant Institute was just a routine. Moreover, Ultracomputer group headed by professor M. Kalos and A. Gottlieb previously already checked some of the Parsys concepts, such as the Conflict Filter mechanism with

---

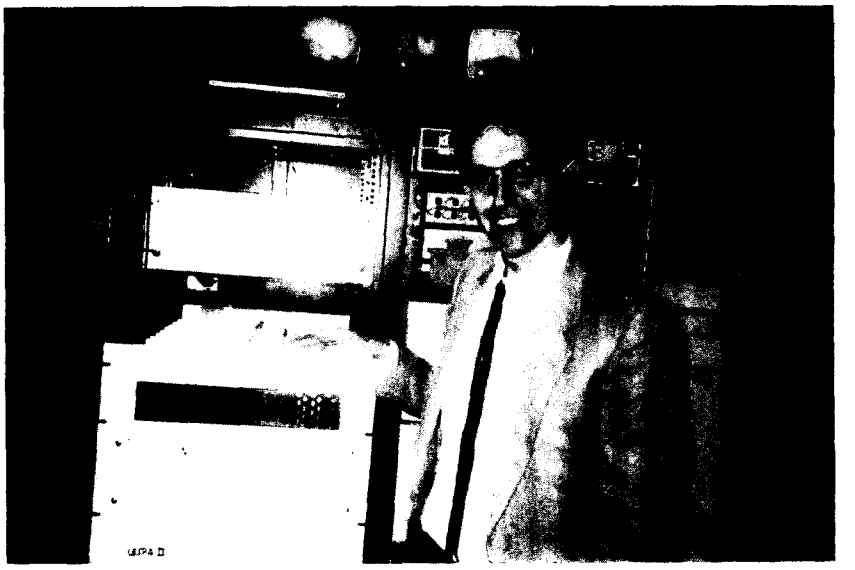


Figure 12. After the Parsys Seminar at MIT, the next seminar at the Ultracomputer Research Laboratory of Courant Institute for Mathematical Sciences (New York University) was more or less routine. Here, Petar enjoyed his presentation in the same way as the audience, because there was a common background of subject matter concerning Parsys and the Ultracom-puter Project.

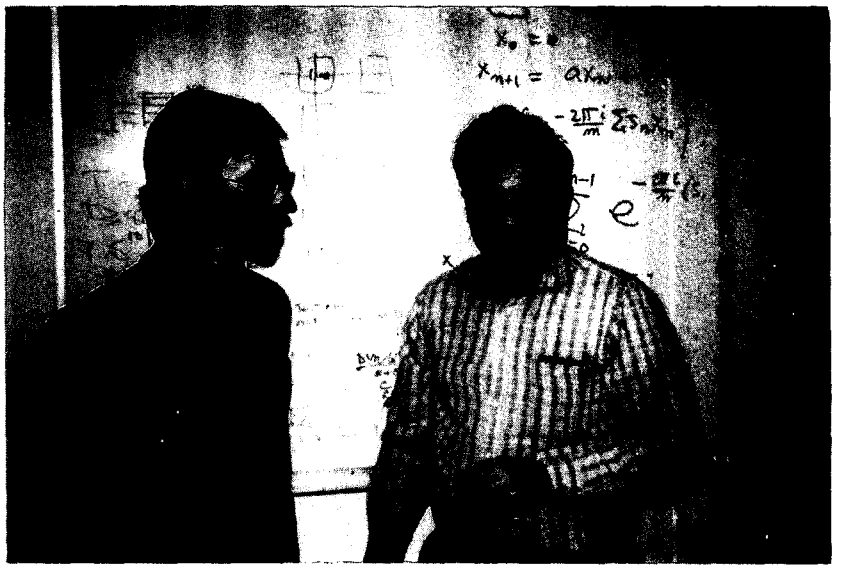


Figure 13. Professor Sasa Prešern and director of the Ultracomputer Research Laboratory at Courant Institute of Mathematical Sciences, professor Malvin H. Kalos, discussing some subjects of the Parsys Seminar and negotiating possible future collaboration concerning parallel computer architectures, parallel programming, and operating systems suitable for parallel architectures.

Figure 14. Professor Malvin H. Kalos (in the middle) and professor Anton P. Zeleznikar during the demonstration of Ultracomputer System at the Ultracomputer Research Laboratory (Courant Institute of Mathematical Sciences at New York University). Special attention was spent to the packaging and technological details of the Ultracomputer System as well as to some problems of machine's parallelism.

PARALLEL COMPUTING SEMINAR
Thursday, April 16, 1987
3:00 p.m. in 613 WWH

RATIONALE AND CONCEPTS FOR THE PARSYS PARALLEL PROCESSOR ARCHITECTURE

PETAR BRAJAK
Iskra Delta, Ljubljana, Yugoslavia

ADJ. PROF. SASA PRESERN
Jozef Stefan Institute, Ljubljana, Yugoslavia

PROF. A. P. ZELEZNIKAR
Edvard Kardelj University, Ljubljana, Yugoslavia

Abstract

PARSYS is a tightly-coupled MIMD parallel processor development project founded by Iskra Delta Computers and carried out by several institutions and faculty research groups in Yugoslavia. Specifically, the project is involved with:

development of prototype hardware and system software,
development of specific programming environments, and
development of applications software.

The architectural design of PARSYS is based upon already proven yet innovative concepts. The prototype machine will be a shared-memory system consisting of 64 processors and 271 memory modules that are connected via a network of VLSI custom-built ISMM (Intelligent Shared Memory Module) chips in a 6-cube configuration. Each ISMM supports fast routing and incorporates functions and logic that avoid or minimize degradations mostly enountered in the multiprocessor environments such as:

N simultaneous accesses to the same memory location,
N simultaneous accesses to the same memory module,
N simultaneous synchronization requests,
slow dynamic context switching, and
huge latency.

In the presentation we describe the concepts of the PARSYS project and show some of the ISMM functions. Primarily, we show how ISMMs enable two or more processors to read/write the same memory address in one memory cycle and how ISMMs maintain lists of waiting processes with a synchronizational mechanism that enables N processes to link up in only one memory cycle.

Courant Institute of Mathematical Sciences
New York University
251 Mercer Street
New York, New York 10012

Figure 15. The Iskra Delta Parsys Parallel Computing Seminar at Courant Institute of Mathematical Sciences (New York University) was announced by a special poster, similar to the one presented in Figure 10 (PS).

New York University
A private university in the public service

Courant Institute of Mathematical Sciences
Ultracomputer Research Laboratory

251 Mercer Street
New York, N.Y. 10012
Telephone: (212)460-7480

arpanet: KALOS@NYU-CMCL1.ARPA

16 April 1987

Professor Anton P. Zeleznikar
Laboratory for Parallel Computer
Department of Electrical Engineering
Edvard Kardelj University in Ljubljana
Trzaska c. 25
61000 Ljubljana
Yugoslavia

Dear Professor Zeleznikar,

On behalf of the Courant Institute of Mathematical Sciences and the Ultracomputer Research Laboratory, I would like to express our gratitude for your visit along with Dr. Presern and Mr. Brajak. The lecture about your PARSYS project was very interesting and, as you could see from the lively discussion, was received with great attention. We found the concepts related to distributed communication in the network to be clever and were intrigued by the method of memory mapping. We hope to able to receive your technical reports and to exchange information in the future.

Sincerely,

Malvin H. Kalos

Malvin H. Kalos
Professor and Director
Ultracomputer Research Laboratory

Figure 16. The Parsys Parallel Computing Seminar at Courant Institute of Mathematical Sciences was received with great attention. The letter of professor Malvin H. Kalos just approved this fact.

Content Addressable Memory on their simulator, so that we were all prepared for concrete talks and discussions. Ultracomputer project was very interested for Parsys group, since Ultracomputer's close collaboration with IBM RP3 project. IBM RP3's principle investigator, Marc Snir, was not able to attend a lecture; however, he specifically requested a copied material of the presentation. Several topics were particularly stressed:

- Parsys approach of reducing network traffic using Conflict Filter mechanism and elaborated Routing Unit design;
- Parsys approach to address scrambling;
- Parsys approach of eliminating hot spots using Get and Link Synchronization mechanism (a much more realistic approach than Ultracomputer's Fetch and Add).

A four hour afternoon session ended with sincere wishes for collaboration and technical report exchanges, a tour to the Ultracomputer Laboratory, and an altogether dinner at one of the finest restaurants in New York's famous Greenwich Village.

4. Columbia University Reminiscence

A visit to Columbia University, Department of Computer Science was not solely a professional tour of a laboratory that pioneered fine grained VLSI large scale parallelism in late 70's and early 80's with projects such as Non-von and Dado.

It was also Mr. Brajak's emotional return to the institution, to a project, where he studied and learned first steps about parallel processing, a return to people with whom he spent hours writing simulators, rethinking layout designs, implementing parallel algorithms, eating pizzas, drinking beer, etc. We were mostly interested in seeing how two well-pubicized projects have progressed in last four years. We learned that Non-von was abounded in 1985 when the principle investigator, professor David Shaw decided to take a challenge at the New York stock exchange market. DADO project, however, turned out to be a successful commercial project, a tree structured parallel processing machine ranging from 8 to 4096 32-bit Motorola processors mostly for AI applications and signal processing.

Dado's principal investigator Sal Stolfo, together with other graduate students established a company called Fifth Generation Computing that produces and sells Dado machines. We were happy to see 64 processor version at Columbia University and had pleasant chats with Mike Van Biemma, Andy Lowrie, Cecilia Paris, Sanjiv Sharma, and other researchers, graduate students, and Mr. Brajak's friends working on Dado project.

5. Visit at Carnegie-Mellon University

A visit to Carnegie-Mellon University was mostly organized by professor Dalibor Vrsalovic, and professor Daniel P.. Siewiorek, who was our guest at Mipro '86 Conference in Opatija. Carnegie-Mellon University is one of leading US universities, particularly well known for multiprocessing (C.mmp, Cm*) and VLSI systolic processing (WARP). Recently, most of the research activity has been focused on software concepts such as parallel operating systems (MACH), program decomposition, debuggers, and software tools for parallelization (PIE).

Main topic with professor Siewiorek, professor Vrsalovic, and particularly with professor Zary Segall, the head of the Computer Science Department, was a possible collaboration and joint venture project particularly on developing and installing MACH and PIE software tools on the PARSYS machine. At present, both MACH and PIE operate on bus oriented multiprocessor systems. Shared memory with multistage network parallel system could give a new dimension to such software.

We left Carnegie-Mellon University with best wishes and hopes for future collaboration with one of the best departments in computer science in United States.

6. San Francisco Bay Conversations

> ... The projecting of the understanding has its own possibility - that of developing itself. This development of the understanding we call "interpretation". In it the understanding appropriates understandingly that which is understood by it. In interpretation, understanding does not become something different. It becomes itself. Such interpretation is grounded existentially in understanding; the latter does not arise from the former. Nor is interpretation the acquiring of information about what is understood; it is rather the working-out of possibilities projected in understanding.
>
> M. Heidegger, B&T, 188-189.

The aim of San Francisco Bay conversations was to exchange experiential information concerning new philosophies of understanding computers, cognition (artificial intelligence), and information. There was also some interest to discuss possibilities of automatic language translation concerning English, Japanese, and Slovene language. The author of this article was particularly interested to get some counterparts for the discussion of the possibilities of a new information philosophy from which, in the future, a more adequate theory of information and science of information could be developed. This question appears to be fundamental also in concern with parallel computer application in the near future, where new, yet not discovered, and more intelligent methodologies have to be applied.

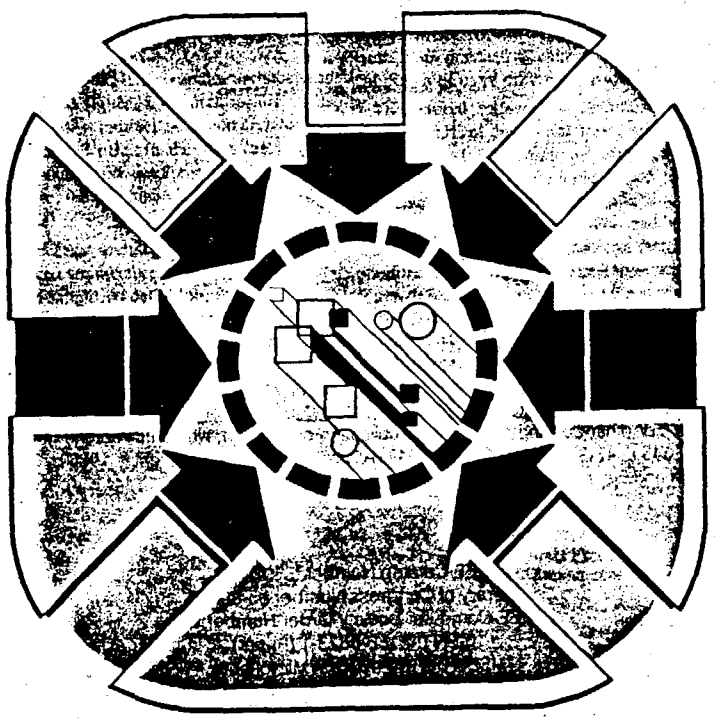
The San Francisco arrival was pleasant and friendly. We met professor Bruno Stiglic from Iskra Electronics and his wife who were our hosts quite at the beginning and later during our symbolic Easter celebration. On Monday, April 20, 1987, we visited professor Terry Winograd at the Center for the Study of Language and Information (CSLI) at Stanford University. CSLI is a research institute devoted to building theories about the nature of information and how it is conveyed, processed, stored, and transformed through the use of language and in computation.

It seems that up to now, CSLI was not studying the arising of information or coming of information into existence, although conveying, processing, storing, and transforming of information in artificial and especially in living systems concerns informational arising in such or another form. Ignoring informational arising, for it does not fit well into the traditional rationalistic methodologies, means ignoring the most evident nature of living,

4/27/87

Tutorial on
Advanced Microprocessors and High-Level Language Computer Architecture

Veljko Milutinović



IEEE COMPUTER SOCIETY

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

COMPUTER SOCIETY PRESS

IEEE COMPUTER SOCIETY ORDER NUMBER 623
LIBRARY OF CONGRESS NUMBER 85-80875
IEEE CATALOG NUMBER EH0236-0
ISBN 0-8186-0623-1

Parallel Processing
The Cm* Experience

Edward F. Gehringer
North Carolina State University

Daniel P. Siewiorek
Carnegie-Mellon University

Zary Segall
Carnegie-Mellon University

We look forward to fruitful joint research.
Dan Siewiorek
April 17, 1987

with my compliments and desire for cooperation,
Apr. 17, 87

digital Digital Press

Figure 18. Professor Daniel P. Sieworek presented to us the book (written together with E.F. Gehringer and Z. Segall) "Parallel Processing: The Cm* Experience" at the occasion of our visit to Carnegie-Mellon University in Pittsburgh, Pa.

Figure 17 (at the left). At the occasion of our visit to Purdue University, West Lafayette, Ind., professor's Veljko Milutinovic present was the Tutorial on Advanced Microprocessors and High-Level Language Computer Architecture.

Figure 19. Professor Zary Segall (Carnegie-Mellon University), one of the today's most distinguished specialists for parallel processing.

Figure 20 (at the right). Professor S. Prešern and professor A. P. Zeleznikar when the site-seeing of San Francisco Bay at the weekend.

Figure 21 (at the right). Professor Saša Prešern and Mr. Petar Brajak at the San Francisco Bay. In the background a part of the Golden Bridge construction can be seen.

Figure 22 (at the left). San Francisco Bay with San Francisco City in the background was a real refreshment in our straining long-tour "parallel" visit in USA. Professor A. P. Zeleznikar, looking forth to the coming meetings and conversations planned for our Parsys and artificial intelligence California visit.

social-interactive informational recurrence and its, for instance, natural verbal and lingual phenomenology. This is even true, when "in using the rich resources language provides for dealing with information, CSLI's researchers show mastery of a powerful apparatus that includes concepts of meaning, reference, knowledge, desire, and intention" (RP). Meaning, reference, knowledge, desire, and intention are characteristically interwoven, recurrent, and arising information of beings during their lively social interaction.

The current research projects at CSLI are organized in the following research activities:

(1) the nature of information, representation, and action;
(2) information and meaning in extended discourse;
(3) information and meaning in sentences;
(4) information and meaning in words; and
(5) sources of information.

CSLI's goal is to develop theories of information that are explicit and systematic and to apply them to the analysis of language. It is not clear at all (at least not to the author) what could be a theory of "information content." If information has a content, does information only carry a content or is the information content only a floret for information of information. It is evident that in the CSLI's understanding of the so-called nature of information, the question of the nature of information was raised only traditionally and that a philosophy of information was not touched in an innovative and investigational manner.

A "full" account of the content and transfer of information requires to embed theories of meaning and interpretation in the real world. At first step is to understand how information about the world is represented. The Representation and Reasoning Project at CSLI is developing a general theory of representation and modeling that will characterize a variety of representational systems, including sentences, utterances, parse-trees, computer screens, minds, and computers. The goal is to build the foundations of a theory of computation that can explain what it is to process, rather than merely to carry, information. Some properties of representation sound to be essential, for instance:

- Representation is a more restrictive notion than information, but a broader one than language. Representation includes photographs and other physical simulations, such as models, and also uses of nonlinguistic symbols like numbers to represent distances and sets to represent meanings.

- Representation is circumstantially dependent, not only because it is specifically relational, but also because whether 'this' represents 'that' depends, in general, on the whole context in which 'this' and 'that' appear.

- There is no reason to suppose that representation is "formal"; it emerges out of partially disconnected physically embodied systems or processes.

- It matters that "represent" is a verb. Representational acts are the primary objects of study, and representational structures, especially those requiring an independent act of interpretation, are taken as derivative.

Besides of the already mentioned project, in the first group of activities (1) there are still the following projects: situation theory and situation semantics (the goal is to provide the basis of a mathematically rigorous, axiomatic theory of information content); situated automata (developing tools for constructing complex machines with well-defined informational properties); rational agency (merging of philosophical and AI traditions in the study of rational behavior to build a theory of belief, desire, and intention as these attitudes act collectivelly, informed by perception, to produce action); semantics of computer languages (a theory of semantics of languages for system description and development); embedded computation (semantic relations between the processes and the embedding context); and analysis of graphical representation (developing an account of the document as an information-bearing artifact).

In the second group of activities (2) are the following projects: discourse, intention, and action (development of theories of discourse, research of the nature of discourse, sentence-level and subutterance phenomena); and grammatical theory and discourse structure (grammar and lexical-functional theory, interaction between discourse and sentence phenomena).

The third group of activities covers foundations of grammar (understanding of methods of encoding linguistic information as systems of rules or constraints and of how that information can be used in recognition and generation) and head-driven phrase structure grammar (analyzing the structure and interpretation of natural language).

The fourth group of activities concerns lexical project (development of workable lexicon that integrates semantic knowledge about lexical items) and AFT (Aitiational frame theory) lexical representation theory. The first project deals with questions:

- How do knowledge of the world and lexical meaning link up?
- How should lexical meaning be represented?
- What is the place of lexico-semantic information in the overall grammar? and
- What is the structure of the lexicon?

For human agents, speech and vision are the primary sources of information about the world. Three projects of the fifth group of activities are concerned with representing and characterizing information contained in speech and with relating this information to other aspects of the communication process. A fourth project is exploring comparable aspects of visual information. These projects are: phonology and phonetics, finite state morphology, computational models of spoken language, and visual communication.

Its fair to say that the author of this report was critically challenged reading Tony Durham's interview with professor Terry Winograd in Computing (July 10, 1986). This was the decisive impulse in authors decision to present another concept of information based on author's studies of information systems concerning structure and organization of molecules of life, living cells, and human cortices. To this, merely a new philosophical orientation was added, by which the "informational" thinking was progressed into a new circulation of understanding.

The conversation concerning the broadened notion of information as proposed by the author (OWI, ID1, POI, and ID2) was superficial and

Figure 23. Our very first contact with Stanford University was professor Terry Winograd.

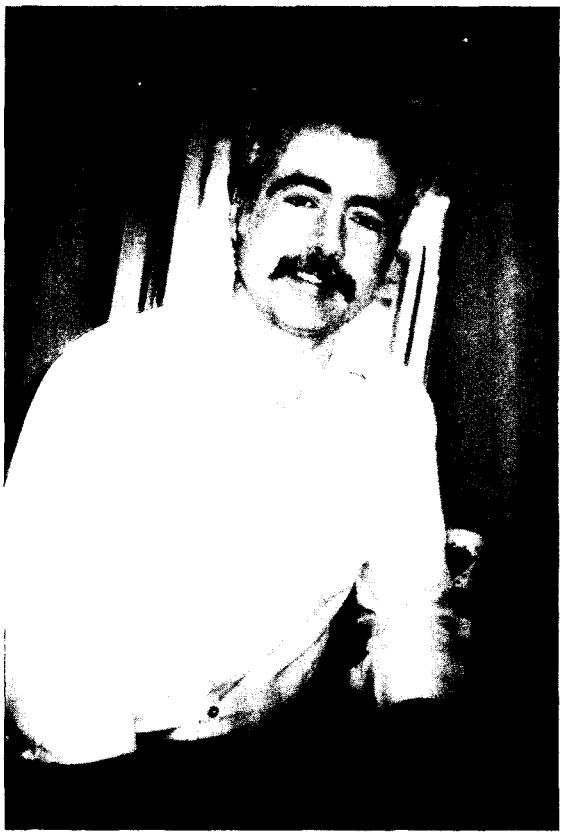


Figure 24 (at the right). Professor Winograd and his colleagues on the staff of the Center for the Study of Language and Information (CSLI) are conspicuously ensconced on the Stanford University campus, which is one of the centers, if not the center, of the artificial intelligence world.

Figure 25. In our friendly, however open conversation, professor Winograd did not agree that Zeleznikar's concept of broadened notion of information is acceptable. Zeleznikar opposed that, for instance, an outward information is practically thrown into the informational realm of human cortices where thereupon information is arising in the way local structuring and organization of this realm. Scientific understanding of information is rooted too much in information theory (theory of communication).

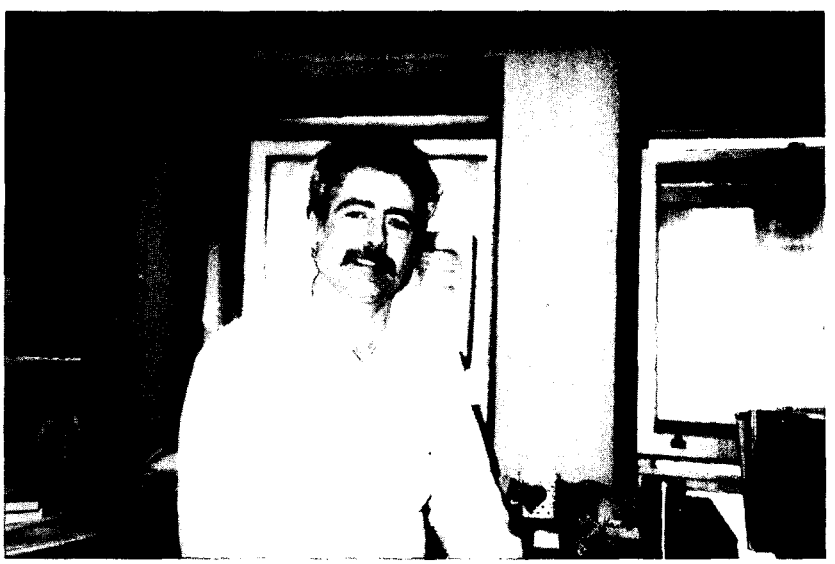


Figure 26. "Dr Terry Winograd is famous for a piece of work he no longer believes in. Even the title under which his doctoral thesis was published in 1972 must now be something of an embarrassment. It was called Understanding Natural Language and the implication was that the computer program SHRDLU was doing understanding." (T. Durham: Language Through the Looking Glass. Computing, July 10, 1986)

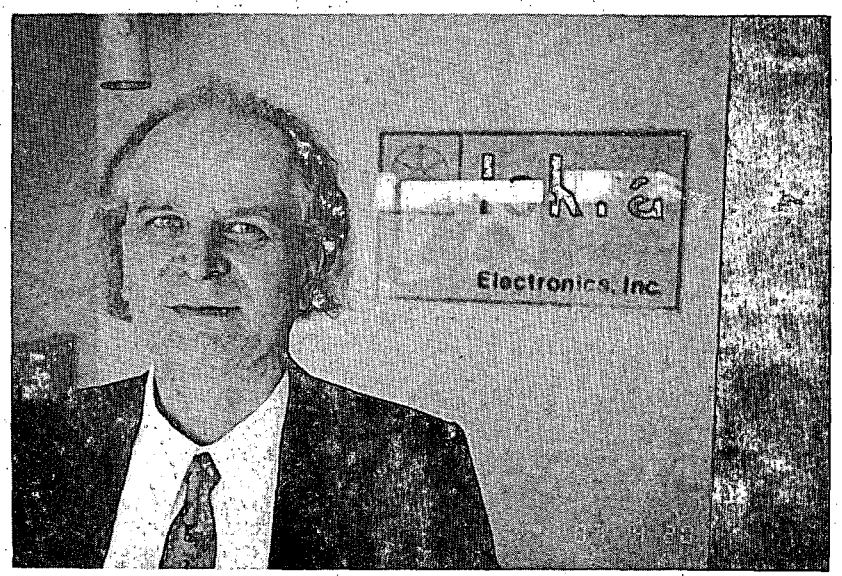
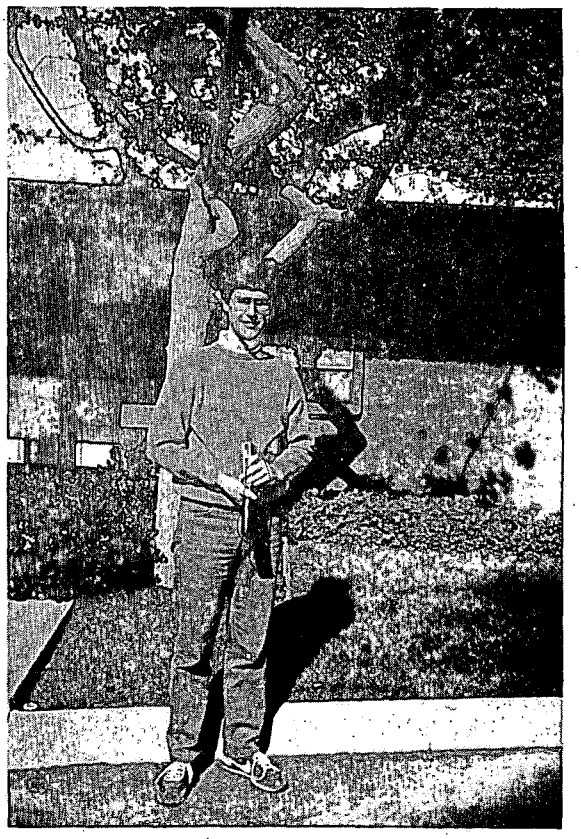


Figure 27. Iskra Electronics, Inc. in Santa Clara, Ca was our intermediate site on the ways through Silicon Valley.

Figure 28 (at the right). Mr. Petar Brajak has taken time to be photographed in the front of Iskra Electronics office in Santa Clara.



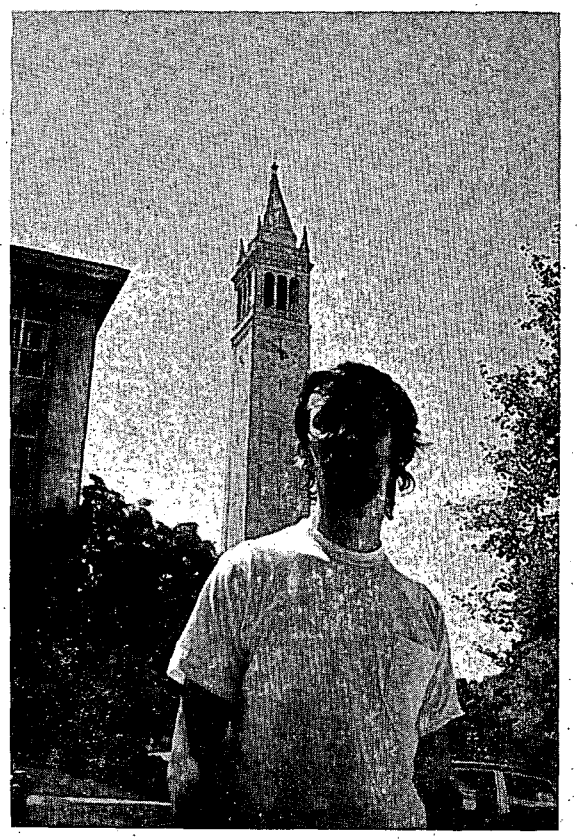


Figure 29. Professor Saša Prešern after visiting some of our colleagues at the Berkeley University. The Californian sky was always clear and the climate refreshing and warm for all of us coming from the foggiest part of Slovenia.

Figure 30. The author of this paper was already a bit exhausted after days of our continuous journey along the east and now along the west coast of United States.

remained on the surface of essentially different backgrounds of understanding. For instance, understanding Being (das Sein) in Heidegger's philosophy as a particular form and process of information seems to be not philosophically acceptable at this time. Was Heidegger's Sein, Dasein, Geworfenheit, Befindlichkeit, Verstehen, etc. a search for something which comes very near to the understanding of the broadened notion of information? Analyzing the research projects of CSLI, it becomes evident that a notion of information in the sense of the classical understanding of information (theory of information, science of information) is already surpassed. As mentioned by professor Winograd, the broadened notion of information may not be called information, so a new term has to be introduced. However, the author's opinion remained and still remains that sooner or later the broadened notion of information will be accepted according to its use in everyday verbal expression and according to its sense in the framework of man's common-sense. To achieve this commitment new efforts in philosophical understanding of information are necessary.

The San Francisco Bay visit is illustrated by photographs from Figure 20 to Figure 30. We visited San Francisco area, Stanford University, Iskra Electronics in Santa Clara, Berkeley University and some other sites and companies. We should not forget a pleasant and useful conversation and dinner with the former professor in Ljubljana, Mr. Egon Zakrajšek of Crommemco company, and should also remember his argumented disappointment forcing him to leave his native country.

7. Los Angeles Propeadeutics

The main point of interest in Los Angeles area was the visit to CalTech (California Institute of Technology). In some manner, this visit was propedeutical, with the goal to obtain some further information concerning parallel computer systems and their programming. At CalTech, the historical cosmic cube was born, connecting sixty four small computers by a network of point-to-point communication channels in the plan of a binary 6-cube. The machine offered high degree of concurrency in applications and suggested that future machines with thousands of nodes are both feasible and attractive.

Our host at CalTech was professor Carl Seitz, the principal investigator of the Cosmic Cube, and some his doctoral candidates. Discussions were mostly in defending each others approach in parallel processing (Cosmic Cube supports local memories and message passing mechanisms; Parsys, on the other hand, favors shared memory approach with fast interconnection elements). Particularly, we were discussing the ways of programming such machines. Photographs in figures 31 to 33 illustrate our visit at CalTech.

Besides of CalTech, we visited some other places in Los Angeles area and gathered documentation concerning research activities in the field of parallel computer technology.

Professor Prešern and myself went to the University of California, Los Angeles (UCLA) to visit professor Milos Ercegovac and to obtain some information on parallel research work at UCLA. At the same time, Mr. Brajak went to San Diego where he had an appointment with VLSI group at M/A-COM Government Systems. He was particularly interested in receiving proper information about possible implementational difficulties and costs of VLSI layout for Parsys ISMM chip. Furthermore, Mr. Brajak obtained there a very relevant information about US Government sponsored BBN Butterfly project.

8. Old Tucson in Arizona

Tucson is an exotic, warm, and touristic site in Arizona. During the time of our visit, professor Branko Souček was still the visiting professor of the Electrical and Computer Engineering Department at the University of Arizona in Tucson. Our seminar for professors and students of ECE Department embraced Parsys architecture and the Trident VME Microcomputer System produced by Iskra Delta. Several possibilities of cooperation in research work of Parsys were discussed.

A very interesting conversation took place with professor Stuart R. Hameroff at Arizona Health Science Center in Tucson. At this occasion, professor Hameroff has presented his book, still preparing it for the print: Ultimate Computing: Biomolecular Consciousness and Nanotechnology (Elsvier Science Publ.)

Photographs in Figures 34 to 36 present professor S. R. Hameroff at lunch in Arizona Health Science Center, professor Branko Souček and his wife, and the classical cowboy with his horse in Old Tucson.

9. Purdue University Negotiations

A visit to Purdue University was one of the most exhaustive experiences in the whole journey. A two day schedule was carefully planed in advance by our host, our dear friend, professor Veljko Milutinović and comprised lectures, talks, laboratory tours, theses defendings, carefully planned lunches with the leading people in parallel processing field and a pleasant dinner at professor Milutinovic's house. It is impossible to mention all events that happened during this two day stay at Purdue University. I will stress only few important ones:

- a visit to three VLSI laboratories (gate array, custom-built, semi custom) where we discussed possible VLSI and GaAs implementations for Parsys Routing Unit design,
- a visit to CDC Cyber 205 supercomputer center, and
- a visit to PASM parallel system laboratory.

Furthermore, we had talks with some of the most distinguishable people in the field. Professor H. J. Siegel, a first name in the field of interconnection networks, an author of numerous papers and books, was quite impressed and intrigued with our network transformation. He was astonished when he realized that Parsys type network was not included in his book "Interconnection Networks for Large-Scale Parallel Processing" by Lexington Books, and had shown willingness to visit Iskra Delta, particularly Mipro's New Computer Generation Symposium.

A very interesting talk we had with professor Henry Dietz, Mr. Brajak's colleague from Columbia University student days. Professor Dietz is the leading researcher in the field of parallel languages and compilers. His doctoral dissertation "The Refined-Language Approach to Compiling for Parallel Supercomputers" received

Figure 31. California Institute of Technology was our last visiting site in Los Angeles area. Several debates concerning parallel architecture and parallel processing with post-graduate students and professors contributed to high standards of mutual understanding. The concept of the Iskra Delta Parsys system remained an original undertaking of interest on further exchange of research reports.

Figure 32. The three-man expedition in the front of the famous and already technologically historical Cosmic Cube parallel computer at CalTech (from the left to right: A. P. Zeleznikar, S. Prešern, and P. Brajak).

Figure 33. In many cases and at different occasions, Mr. Brajak's great interest was in looking packaging of subunits and components in a parallel computer system. At CalTech, this was the case for the Intel's parallel machine iPSC.

Figure 34. Professor Stuart R. Hameroff, anestethesiologist by profession, has accepted our expedition in the Arizona Health Center in Tucson, Arizona as a courtesy of our host, professor Branko Souček. The main attraction of this visit was the preparation of the manuscript of the book Ultimate Computing: Biomolecular Consciousness and Nanotechnology (to appear at Elsvier Science Publ.)

Figure 35. Professor Branko Souček and his wife have been our hosts in the town built in the desert. Professor Souček has organized a number of meetings with several professors and researchers of the Electrical and Computer Engineering Department at the Arizona University of Tucson.

Figure 36. At the time living in Tucson we have hoped that finally our Parsys Project is getting its horse so we can ride in the future with a refreshing horsepower. The photograph was made in Old Tucson, a site of historical memory of the American culture.

"ACM the most distinguishable award." He proposed a possible collaboration such that Refined Languages and PARSE software tool project become part of Parsys sophisticated architecture.

Besides professor H. J. Siegel and professor H. Dietz, we had interesting talks with professors Tom Cassavant (parallel operating systems), Kak (artificial intelligence), Schwederski (PASM), and Fortes (fault-tolerant networks). We should also mention the hospitality of Mr. Spira Matić from Boris Kidrič Institute, Vinča, who was on a scientific visit to Purdue University at that time.

On the way back to New York we stopped at the National Center for Supercomputers Research and Development at University of Illinois in Urbana-Campaign. We were not able to find Cedar's principal investigators professor David Kuck and professor Andy Lawrie; however, we gathered relevant information and technical reports concerning Cedar Machine and Alliant FX commercial product.

10. A Jump to Thomas Watson Research Center

Our, only several hours lasting visit to IBM's Thomas Watson Research Center was pleasant, friendly, and intelligent. The host, dr. Vojin G. Oklobdžija, research staff member at exploratory VLSI design, demonstrated to us not only operation of GF11 and a look inside this large parallel machine, but also carefully devoted his time to several question we were able to discuss.

The IBM Research Division conducts fundamental scientific research and explores the application of advanced technologies to IBM products. The basic scientific orientation of the Division concerns

- physics,
- chemistry,
- mathematics, and
- computer science.

In addition, a significant portion of IBM's research is aimed at

- advancing state-of-the-art computer technologies in
  - logic and memory,
  - storage,
  - input/output devices,
  - communications,
  - computer systems and programming,
  - computer application, and
  - manufacturing.

The Research Division employs more than 3,000 people at four major location:

- Thomas J. Watson Research Center (RC for short) in Yorktown Heights (the division's headquarter),
- Almaden Research Center in San Jose, Ca.,
- Zuerich Research Laboratory in Switzerland, and
- Tokyo Research Laboratory.

The RC is organized into six major departments:

- semiconductor science and technology,
- physical sciences,
- mathematical sciences,
- input/output technologies,
- manufacturing research, and
- computer sciences.

Research activity is enhanced by an advanced computing environment (some 43,000 books and nearly 1,700 current scientific journals). There is a lively calendar of events, which lists hundreds of lectures, seminars, colloqia, and other items of interest to staff members, as well as diverse educational program.

To recognize the scientific and technical achievements of the research staff, IBM presents corporate awards and the research division presents its own internal awards and honors. The most prestigious technical honor awarded to an individual is appointment as an IBM fellow. There are only some 15 IBM fellows among whom are the Nobel price winners (in 1986 and probably in 1987) of the Research Division.

The Computer Science Department (CSD) maintains a broad and innovative research program and, at the same time, provides technical leadership within IBM. With this aim, the department actively contributes to the worldwide scientific community and transfers the results of its technological advances to

IBM development laboratories.

Today, the CSD employs some 600 people who are

- computer science,
- engineering, and
- programming professionals.

More than 80 percent of the research staff members holds Ph.D. degrees. Research activity, conducted in small groups, is project-oriented and often interdisciplinary in character.

The CSD is organized into six areas:

- communication systems,
- work-station systems,
- symbolic and numeric processing,
- large systems and VLSI,
- advanced minicomputer systems, and
- software technology.

Staff members are encouraged to publish the results of their research in the open literature. Research results are also published in the IBM Research Reports, whose listing and abstracts are circulated to universities and scientific centers, and in IBM's own technical journals,

- the IBM Journal of Research and Development and
- the IBM Systems Journal.

Staff members play an active role by attending conferences, officiating in professional societes, and editing journals, as well as serving on organizing committees for conferences and meetings, editorial and technical boards, and advisory committees.

At the occasion of our visit in the RC the GF11 Supercomputer was demonstrated. GF is a parallel computer conceived primarily for the numerical solution of problems in quantum chromodynamics (QCD), being jointly developed with the Physical Science Department. The machine incorporates 576 floating-point processors, each with its own 2 million bytes of memory, and capable of 20 million floating-point operations per second (20 flops), giving the total machine over 1 gigabyte of memory and a peak processing speed of more than 11 gigaflops. The floating-point processors are interconnected by a high-speed full Benes network, a nonblocking switch capable of realizing configurations incorporating any permutation of the processors and able to

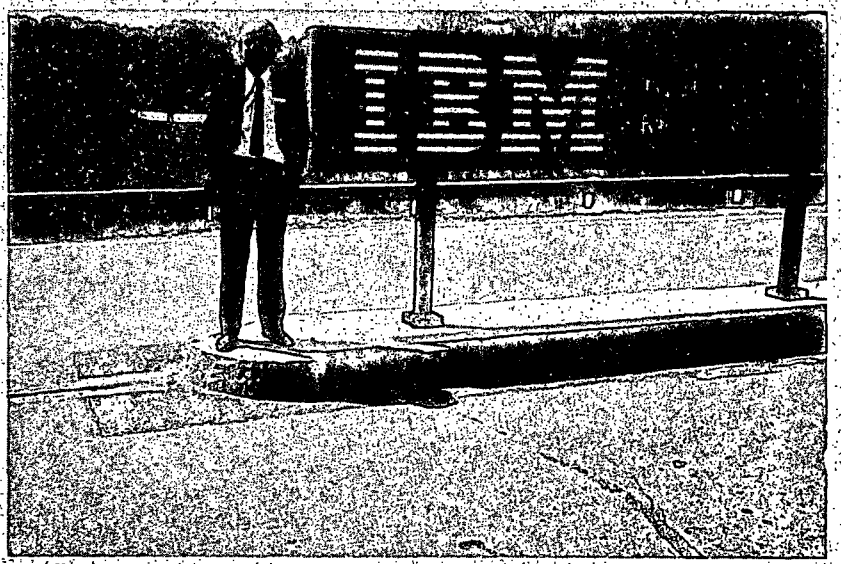


Figure 37. Computer Science of the Thomas J. Watson Research Center at Yorktown Heights (NY) was the last place of our visit.

Figure 38 (at the right). The author's joke was that his hair is pointing into the direction of T.J. Watson RC when living it. We have to thank for IBM's hospitality and for expertise which was offered. It happened on a bright and windy day.

Figure 39 (at the left). How we shall cross the bridge connecting an ambitious research and engineering undertaking on one side and economical and cultural circumstances on the other side?

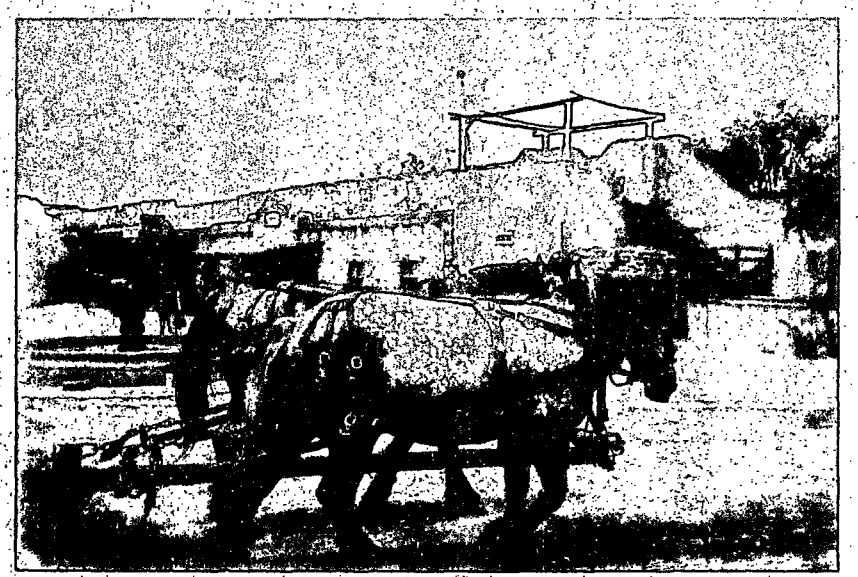


Figure 40. Engineers are always like horses of the modern high-tech company. They have to draw the company's carriage and they are more often than not whipped when the carriage is moving to slow and not in the appropriate innovative direction. The horse culture of Old Tucson may sound not only historical, but also instructive.

reconfigure in one processor cycle. Using this switch, GF11 can be organized in any one of a number of different topologies, such as a rectangular mesh of any dimension and size, any torus, a hexagonal mesh, or some irregular organization fitted to a single problem.

A central controller is used to broadcast instructions to all of the processors and to the switch, and also to communicate with host computer. GF11 is a modified single-instruction multiple-data (SIMD) machine: all processors receive the same instruction at the same time. The SIMD architecture has the advantages of simplicity, high performance due to elimination of synchronization overhead, and reduced cost due to sharing of the controller.

A typical calculation in QCD, for example, an evaluation of the masses of the proton, neutron, and a few related particles, is estimated to require 3 times 10 to power 17 arithmetic operations. With a 100 megaflop machine this calculation would take 100 years. By a parallel application of its 576 processors, GF11 is capable of completing such a calculation in about one year. QCD is only one example of a class of scientific problems that can be structured for computation on GF11.

Furthermore, Dr Oklopdzija prepared a lecture for us on IBM's new supercomputer TF-1 (Terraflop 1), a 10 to power 12 floating point operations per second supercomputer, incorporating special purpose custom-built VLSI arithmetic processor.

The visit to IBM RC was pleasant and impressive. When living the RC, the photo of my questioning staying on Figure 37 was made. However, my thinking hairs still pointed to the RC direction (Figure 38).

11. Have I really done nothing in the last two years?

The question I have to answer concerns a critical phase of our social and cultural development. My answer is a trial how to exclude several unpleasant, ill-meaning, and unintelligible idle talks concerning my activity in the last two years. These talks are coming into existence at various unprofessional levels and are propagated by several non-professionals. Why I have to answer such non-proficient rumors at all?

First, I have to agree that my intellectual action in developing a new philosophy of information and in organizing and advising research and development to Parsys project were hardly recognized as useful and prospective under existing subcultural managerial and scientific circumstances. According to my own measures, in the last two years, I have attained much more than in the previous 20 years of my life when I was living in safety at a "national" scientific institute. Let me explicate only two substantial, already mentioned and developmentally crucial activities: establishing a new philosophy of information as a starting-point for future investigations of intelligent machines and the initial pushing, professional concentration, and organizational help of the Parsys project.

Since autumn 1985 I have visited Japan twice and USA with the intention to bring new research and technological orientations in the corporative discourse of national computer industry. There was never doubt, at least from my view of understanding the scientific and technological trends of development that novel orientations are a necessity for industrial or even for national survival. Besides my main activities (research and consulting work), I have performed regular activity as academic lecturer, adviser, editor, author, and organizer to mention several of my factual activities. In this respect, I would be extremely pleased to hear again how I was doing nothing in the last two years and how this slogan can be understood from the managerial or subcultural point of view.

12. Conclusion

What is the moral of this long, straining, and fruitful two year journey into the world of information, intelligence, and parallel processing? Do we, the actors of different adventurous and exhaustive expeditions into the unknown and by knowledge populated areas, have anything to regret?

References

(LLG) T. Durham: Language Through the Looking Glass. Computing, July 10, 1986.

(B&T) M. Heidegger: Being and Time. Harper & Row, Publ., New York (1962).

(OWI) A. P. Zeleznikar: On the Way to Information. Informatica 11 (1987), No. 1, 4-18.

(ID1) A. P. Zeleznikar: Information Determinations I. Informatica 11 (1987), No. 2, 3-17.

(POI) A. P. Zeleznikar: Principles of Information. Informatica 11 (1987), No. 3, 9-17.

(PS) A. P. Zeleznikar: Parsys Expeditions to New World. Informatica 11 (1987), No. 3, 76-80.

(ID2) A. P. Zeleznikar: Information Determination II. Informatica 11 (1987), No. 4, 8-25.

(RP) The Research Program at CSLI. Center for the Study of Language and Information, Ventura Hall, Stanford University, CA 94305.

(TW) Thomas J. Watson Research Center. IBM, Yorktown Heights, New York 10598.

****
*
* AVTORSKO STVARNO KAZALO CASOPISA *
* INFORMATICA, LETNIK 11 (1987) *
*
****

C l a n k i

Alagić Mara and S. Alagić: Categorical Approach to the Relational Model of Data. Informatica 11 (1987), No. 3, pp. 3 - 8.

Barle J., J. Grad, and D. Krstić: A Production Problem Interactive Prototype of Linear Programme. Informatica 11 (1987), No. 3, pp. 29 - 31.

Barle T. in Eli Delidžakova-Drenik: Ugotovitve ob preizkusu demonstracijske verzije programskega paketa Micro-Optrans. Informatica 11 (1987), št. 2, str. 66 - 68.

Brajak P.: Designing a Cofigurable Intelligent Memory Module (RIMM) for Performance Enhancement to Large Scale, General Purpose Parallel Processor. Informatica 11 (1987), No. 1, pp. 19 - 53.

Brodnik A., M. Spegel in T. Lasbaher: Programiranje z Modulo-2, prvi del. Informatica 11 (1987), št. 2, str. 78 - 82.

Brodnik A., M. Spegel in T. Lasbaher: Programiranje z Modulo-2, drugi del. Informatica 11 (1987), št. 4, str. 69 - 76.

Fajfar D. and M. Lokar: Analysis of Buffered Multistage Interconnection Network for Parallel Processors. Informatica 11 (1987), No. 4, pp. 3 - 7.

Filipič B.: Implementing Pascal-like Constructs: An Exercise in Prolog Programming. Informatica 11 (1987), No. 2, pp. 27 - 30.

Freyder J. D.: Reasoning Simulation Programs. Informatica 11 (1987), No. 3, pp. 32 - 37.

Grad J. and M. A. Jenkins: Decision Support Systems: Tools, Expectations and Realities. Informatica 11 (1987), No. 3, pp. 18 - 24.

Ipavec Marija: Produkt za upravljanje relacijske baze podatkov - VAX Rdb/VMS. Informatica 11 (1987), št. 2, str. 69 - 73.

Janoš T.: Jedno proširenje biblioteke Ratfor-a potprogramima za obradu alfanumeričkih veličina. Informatica 11 (1987), št. 2, str. 53 - 56.

Jeram Sonja: Computation May Hinge on Biological Materials. Informatica 11 (1987), No. 3, pp. 72 - 75.

Jerman-Blažič Borka: Izbira kodne strukture v 6. nivoju referenčnega modela OSI za potrebe prenosa in izmenjave podatkov. Informatica 11 (1987), št. 1, str. 83 - 87.

Jerman-Blažič Borka in Monika Kapus-Kolar: Komunikacijski in aplikacijski procesi v višjih nivojih referenčnega modela OSI. Informatica 11 (1987), št. 4, str. 84 - 88.

Jockovič M. B. and D. M. Velašević: One Solution of the Busline Crew Scheduling Problem. Informatica 11 (1987), No. 4, pp. 35 - 39.

Kočovič P.: Geometrijsko modeliranje upotrebom Eulerovih formula. Informatica 11 (1987), št. 1, str. 67 - 72.

Kočovič P.: A Relational Database for Represenitation of Machining Parts. Informatica 11 (1987), No. 2, pp. 18 - 26.

Kolbezen P.: Language Considerations of Parallel Processing Systems. Part One: Concurrent Microprocessing Sstems. Informatica 11 (1987), No. 2, pp. 31 - 35.

Kolbezen P.: Language Considerations of Parallel Processing Systems. Part Two: Survey and Analysis of Concurrent Languages. Informatica 11 (1987), No. 2, pp. 36 - 43.

Kolbezen P., S. Mavrič in B. Mihovilovič: RISC arhitekture. Informatica 11 (1987), št. 3, str. 60 - 68.

Meža M.: Nekatere izkušnje pri uvajanju Prologa v pouk računalništva na srednjih šolah. Informatica 11 (1987), št. 3, str. 69 - 71.

Malekovič M.: Normalne forme u relacionim bazama podataka: logičke osnove. Informatica 11 (1987), št. 1, str. 78 - 82.

Markovič P.: Implikacija organizacije nervnih sistema na računarske sisteme. Informatica 11 (1987), št. 4, str. 77 - 83.

Mihovilovič B., P. Kolbezen in J. Silc: Komunikacijski procesi v transputerskih sistemih. Informatica 11 (1987), št. 2, str. 74 - 77.

Mihovilovič B., P. Kolbezen, and J. Silc: A Paradigm of Transputer System Implementation. Informatica 11 (1987), No. 4, pp. 54 - 58.

Ojsteršek M., V. Zumer, P. Kokol, and A. Zorman: A Simulation of Dataflow Computer Models and a Model of Fault Tolerant Dataflow Computer. Informatica 11 (1987), No. 4, pp. 59 - 63.

Prešern S.: A Selected Survey of Parallel Computer Systems. Informatica 11 (1987), No. 4, pp. 40 - 53.

Rugelj J. in M. Vidmar: Pregled arhitektur ter uporabljenih ISO standardov v MAP/TOP lokalnih mrežah. Informatica 11 (1987), št. 3, str. 54 - 59.

Semulič B.: Modeliranje informacijskih sistemov z upoštevanjem elementov dinamike realnega pojava. Informatica 11 (1987), št. 3, str. 41 - 45.

Smilevski M.: Metodološki pristup definisanju informacionih sadržaja baze podataka. Informatica 11 (1987), št. 1, str. 54 - 60.

Ščavničar I.: Model informacijskega sistema za podporo kadrovski dejavnosti. Informatica 11 (1987), št. 2, str. 44 - 52.

Silc J. and B. Robič: The Review of Some Data Flow Computer Architecture. Informatica 11 (1987), No. 1, pp. 61 - 66.

Silc J. and B. Robič: Data Flow Based Parallel Inference Machine. Informatica 11 (1987), No. 4, pp. 27 - 34.

Temeljotov A., D. Mrdakovic, D. Pavšelj in D. Cuk: Občasna notranja diagnostika mikroračunalnika EPM-850. Informatica 11 (1987), št. 1, str. 73 - 77.

Terčelj A.: Fraktali - Grafične skrivnosti računalniških umetnikov. Informatica 11 (1987), št. 3, str. 46 - 50.

Vidojkovic D. i V. Jovanovic: Jezik za specifikaciju infološkog modela. Informatica 11 (1987), št. 2, str. 60 - 65.

Vogel L.: Komunikacijski vmesnik SPI-11. Informatica 11 (1987), No. 3, pp. 38 - 40.

Zeleznikar A. P.: Informatici na pot. Informatica 11 (1987), št. 1, str. 3.

Zeleznikar A. P.: Na poti k informaciji (On the Way to Information). Informatica 11(1987), št. 1, str. 4 - 18.

Zeleznikar A. P.: Information Determinations 1. Informatica 11 (1987), No. 2, pp. 3 - 17.

Zeleznikar A. P.: Raziskave računalnikov in informacije v naslednjem desetletju. Informatica 11 (1987), št. 2, str. 57 - 59.

Zeleznikar A. P.: Principles of Information. Informatica 11 (1987), No. 3, pp. 9 - 17.

Zeleznikar A. P.: Artificial Intelligence Experiences Its Own Blindness. Informatica 11 (1987), No. 3, pp. 25 - 28.

Zeleznikar A. P.: Research of Computers and Information in the Next Decade. Informatica 11 (1987), No. 3, pp. 51 - 53.

Zeleznikar A. P.: Information Determinations II. Informatica 11 (1987), No. 4, pp. 8 - 25.

Zivkovic D.: Alternativni postupak odredivanja tipova entiteta i akcija u JSD metodi projektovanja informacionih sistema. Informatica 11 (1987), št. 4, str. 64 - 68.

Novice in zanimivosti

Ali bo fizika izrinila matematiko s prvega mesta med znanostmi (I. Ščavničar). Informatica 11 (1987), št. 2, str. 83 - 84.

Report of a Journey

Zeleznikar A. P.: Parsys Expeditions to New Worlds. Informatica 11 (1987), No. 3, pp. 76 - 80.

Nove knjige

T. Winograd and F. Flores: Understanding Computers and Cognition: A New Foundation for Design. Informatica 11 (1987), št. 1, str. 88 - 89 (recenzent A. P. Zeleznikar).

Pisma bralcev

L. Omejec: Kritika članka B. Vilfana, V. Mahniča in T. Mohoriča. Informatica 11 (1987), št. 1, str. 90 - 91.

B. Vilfan, V. Mahnič in T. Mohorič: Odgovor na kritiko članka avtorjev. Informatica 11 (1987), št. 1, str. 90.