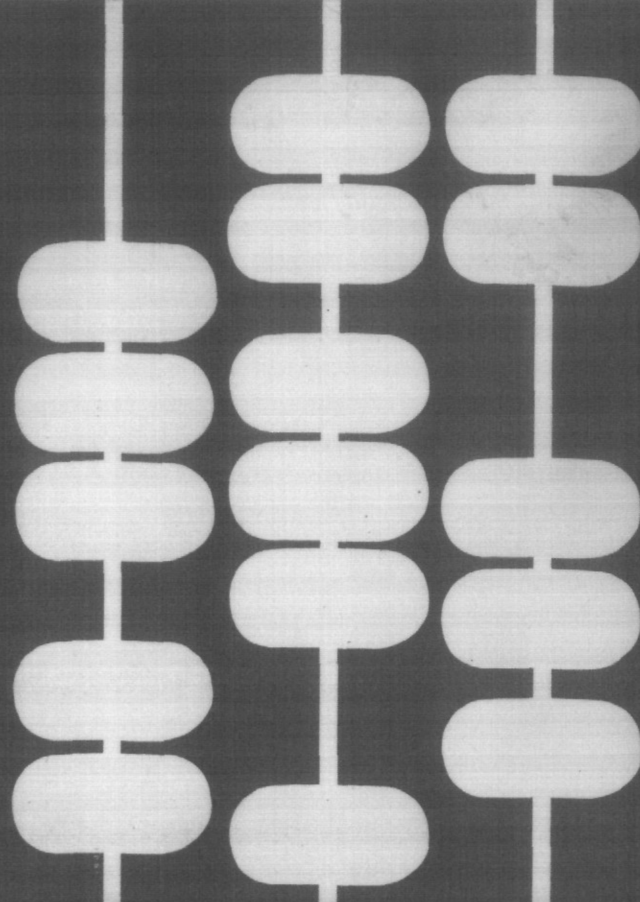
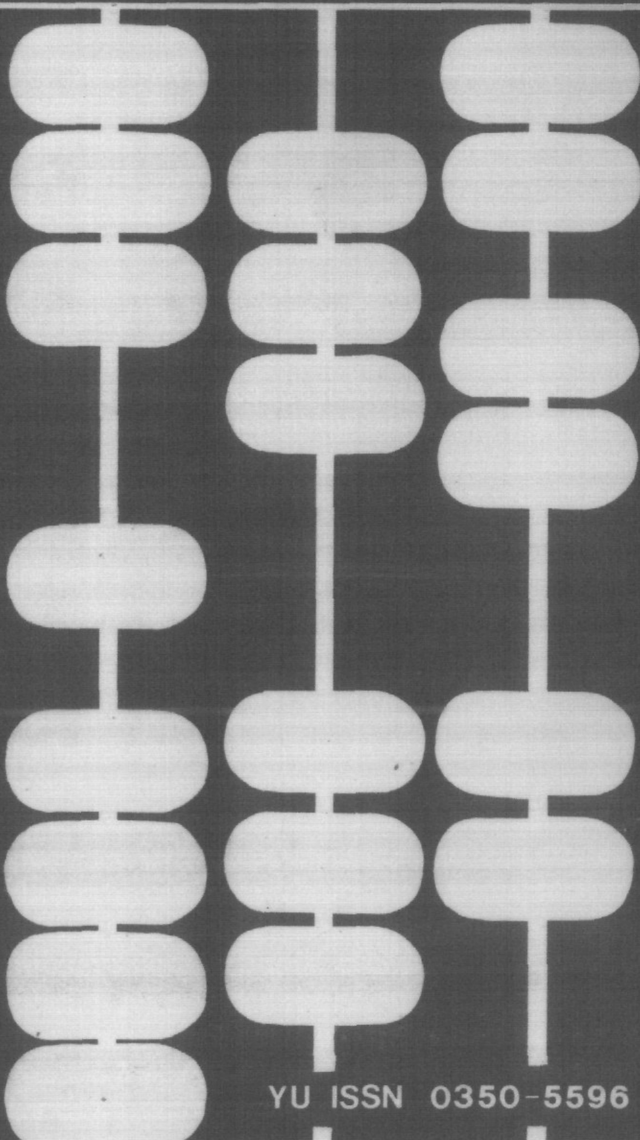


85 informatica 3



YU ISSN 0350-5596

VELIKA KAPACITETA MALEGA MIKRORAČUNALNIKA
PARTNER
Centralna procesna enota 128 KB pomnilnika
Diskovna enota Winchester, zmogljivosti 10 MB
Disketna enota, zmogljivost 1 MB
Serijski vmesnik za tiskalnik
Operacijski sistem CP/M PLUS®
Uporabniška dokumentacija
ISKRA DELTA Tržno komuniciranje Parmova 41
Iskra Delta

informatica

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

YU ISSN 0350–5596

LETNIK 9, 1985 — št. 3

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

UREDNIŠKI ODBOR:

T. Aleksić, Beograd; D. Bitrakov, Skopje; P. Dragojlović, Rijeka; S. Hodžar, Ljubljana; B. Horvat, Maribor; A. Mandžić, Sarajevo; S. Mihalić, Varaždin; S. Turk, Zagreb

GLAVNI IN ODGOVORNI UREDNIK: Anton P. Železnikar

TEHNIČNI ODBOR:

V. Batagelj, D.Vitas -- programiranje
I. Bratko -- umetna inteligenca
D. Ćećez-Kecmanović -- informacijski sistemi
M. Exel -- operacijski sistemi
B. Džonova-Jerman-Blažič -- srečanja
L. Lenart -- procesna informatika
D. Novak -- mikroračunalniki
Neda Papić -- pomočnik glavnega urednika
L. Pipan -- terminologija
V. Rajkovič -- vzgoja in izobraževanje
M. Špegel, M. Vukobratović -- robotika
P. Tancig -- računalništvo v humanističnih in družbenih vedah
S. Turk -- materialna oprema
A. Gorup -- urednik v SOZD Gorenje

TEHNIČNI UREDNIK: Rudolf Murn

ZALOŽNIŠKI SVET:

T. Banovec, Zavod SR Slovenije za statistiko, Vožarski pot 12, Ljubljana
A. Jerman-Blažič, DO Iskra Delta, Parmova 41, Ljubljana
B. Klemenčič, Iskra Telematika, Kranj
S. Saksida, Institut za sociologijo Univerze Edvarda Kardelja, Ljubljana
J. Virant, Fakulteta za elektrotehniko, Tržaška 25, Ljubljana

UREDNIŠTVO IN UPRAVA: Informatica, Parmova 41, 61000 Ljubljana; telefon (061) 312-988; teleks 31366 YU Delta

LETNA NAROČNINA za delovne organizacije znaša 2900 din, za redne člane 790 din, za študente 290 din; cena posamezne številke je 890 din. ŽIRO RAČUN: 50101 - 678 - 51841.

Pri financiranju časopisa sodeluje Raziskovalna skupnost Slovenije.

Na podlagi mnenja Republiškega sekretariata za prosveto in kulturo št. 4210-44/79, z dne 1.2.1979, je časopis oproščen temeljnega davka od prometa proizvodov

TISK: Tiskarna Kresija, Ljubljana

GRAFIČNA OPREMA: Rasto Kirn

V S E B I N A

M. Gerkeš	3	Logični modeli računalniških struktur
R. Murn, S. Prešern, D. Peček, B. Kastelic	15	Odkrivanje napak z Bergerovimi kodi II
Z. Vukajlović	22	Integrisano okruženje programmskog jezika - Alat za udobno i efikasno programir.
S.J.Djordjević	25	Indeksiranje magnetnih traka
S.J.Djordjević	29	Paralelno indeksiranje
J. Berce	33	Izvajalniki za realni čas pri multiprocesorskih sistemih
J. Divjak-Zalokar	39	Bibliografski mikroračunalniški sistem za preiskovanje povzetkov
I. Kononenko	44	Strukturno avtomatsko učenje
I. Komprej, D. Čuk	56	Komunikacija operaterja z lokalno konzolo perifernega mikroračunalnika
M. Kukrika	59	Elementi arhitekture raspodijeljenog sistema za rad u stvarnom vremenu
D. Lecić	66	Multiprogramiranje i merenje I/O čekanja sistema
	68	Nove računalniške generacije
	71	Uporabni programi
	87	Polemika: Umetna inteligenca
	90	Novice in zanimivosti

informatica

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

EDITORIAL BOARD:

T. Aleksić, Beograd; D. Bitrakov, Skopje; P. Dragojlović, Rijeka; S. Hodžar, Ljubljana; B. Horvat, Maribor; A. Mandžić, Sarajevo; S. Mihalić, Varaždin; S. Turk, Zagreb

EDITOR-IN-CHIEF: Anton P. Železnikar

TECHNICAL DEPARTMENTS EDITORS:

V. Batagelj, D. Vitas -- Programming
I. Bratko -- Artificial Intelligence
D. Ćećez-Kecmanović -- Information Systems
M. Exel -- Operating Systems
B. Džonova-Jerman-Blažič -- Meetings
L. Lenart -- Process Informatics
D. Novak -- Microcomputers
Neda Papić -- Editor's Assistant
L. Pipan -- Terminology
V. Rajkovič -- Education
M. Špegel, M. Vukobratović -- Robotics
P. Tancig -- Computing in Humanities and Social Sciences
S. Turk -- Computer Hardware
A. Gorup -- Editor in SOZD Gorenje

EXECUTIVE EDITOR: Rudolf Murn

PUBLISHING COUNCIL:

T. Banovec, Zavod SR Slovenije za statistiko, Vožarski pot 12, Ljubljana
A. Jerman-Blažič, DO Iskra Delta, Parmova 41, Ljubljana
B. Klemenčič, Iskra Telematika, Kranj
S. Saksida, Institut za sociologijo Univerze Edvarda Kardelja, Ljubljana
J. Virant, Fakulteta za elektrotehniko, Tržaška 25, Ljubljana

HEADQUARTERS: Informatica, Parmova 41, 61000 Ljubljana, Yugoslavia
Phone: 61-312-988; Telex: 31366 YU DELTA

ANNUAL SUBSCRIPTION RATE: US$ 22 for companies, and US$ 10 for individuals

Opinions expressed in the contributions are not necessarily shared by the Editorial Board

PRINTED BY: Tiskarna Kresija, Ljubljana

DESIGN: Rasto Kirn

VOLUME 9, 1985 – No. 3

C O N T E N T S

M. Gerkeš	3	Logical Models for Computer Structures
R. Murn, S. Prešern, D. Peček, B. Kastelic	15	Error Detection with Berger Code II
Z. Vukajlović	22	Integrated Programming Language Environment - A Tool for Confortable and Efficient Programming
S.J.Djordjević	25	Magnetic Tape Indexing
S.J.Djordjević	29	Parallel Indexing
J. Berce	33	Executives for Real Time Multiprocessor Systems
J. Divjak-Zalokar	39	Bibliographic Microcomputer System for Abstracts Retrieval
I. Kononenko	44	Inductive Machine Learning
I. Komprej, D. Čuk	56	Operators Communication Using Local Console of a Peripheral Microcomputer
M. Kukrika	59	Elements of Real-Time Distributed Syst. Architecture
D. Lecić	66	Multiprogramming and I/O Waiting Measurments
	68	New Computer Generations
	71	Programming Qiuckies
	87	Polemics: Artif.Intelligence
	90	News

INFORMATICA 3/85

LOGIČNI MODELI RAČUNALNIŠKIH STRUKTUR

MAKSIMILJAN GERKEŠ

UDK: 681.3.517:11/.12

TEHNIŠKA FAKULTETA, MARIBOR
VTO ELEKTROTEHNIKA, RAČUNALNIŠTVO IN INFORMATIKA

Koncept stanja in operacije, kot je znan iz snovanja programske opreme, je izhodiščni koncept z nekoliko širšo interpretacijo. Nabor sestavljenih operacij tvorijo selektorska, sekvenčna in paralelna operacija. Definirana je zančna operacija, ki ima poljubno mnogo ponovitev in nima izhoda. Transformacije, definirane nad sestavljenimi operacijami omogočajo njihovo preoblikovanje, tako da jih lahko modeliramo z izbranimi mikroelektronskimi komponentami male, srednje, velike, pa tudi zelo velike stopnje integracije, vključno z logičnimi mrežami. Na tej osnovi so zgrajeni modeli nekaterih značilnejših logičnih in računalniških struktur.

LOGICAL MODELS FOR COMPUTER STRUCTURES: The concept of state and operation, as it is known from software design is basic concept with extended interpretation. Collection of compound operations consists of select, sequential and parallel operation. An infinite loop operation with no exit terminal is defined. Compound operations can be changed with a set of defined transformations into a different forms which can be simply modeled with SSI, MSI, LSI, or even VLSI structures, including gate arrays. Models for some characteristic logic and computer structures are proposed on that base.

UVOD:

Snovanje računalniških struktur postaja z razvojem mikroelektronske tehnologije vedno bolj kompleksno opravilo. Ob predpostavki, da omogočajo metode programskega snovanja učinkovito reševanje nalog na področju programske opreme, se zdi vprašanje, ali je možno te metode enako učinkovito uporabljati tudi za snovanje strojne opreme povsem utemeljeno. Ekspliciten odgovor na tako zastavljeno vprašanje bi bil zaenkrat precej spekulativen. Lažje je odgovoriti tako, da je možno s smiselno prireditvijo teh postopkov, doseči z njimi dobre rezultate tudi na področju snovanja strojne opreme.

Začetni zapis računalniške strukture, ki jo želimo realizirati običajno pojmujemo kot nekakšno amorfno strukturo, saj zaradi semantične razdalje v splošnem ni možen neposreden prehod na logično izvedbo te strukture. Izkušnje učijo, da ni smotrno vnaprej predpostavljati organizacijo takšne strukture na logičnem nivoju, ampak da jo je potrebno izpeljati iz lastnosti specifikacije, iz katere izhajamo, z upoštevanjem zunanjih parametrov-hitrost, cena, zanesljivost, ...

Ob takšnem izhodišču nas postopki snovanja strojne opreme s pomočjo modelov brez večjih neprijetnih presenečenj vodijo do želene realizacije. Koraki, ki jih pri tem izvajamo, so podobni snovanju programske opreme, le da so osnovne strukture drugačne. Upoštevati pa moramo tudi zunanje parametre, kamor sodijo tudi realne lastnosti mikroelektronskih komponent, ki jih lahko idealiziramo samo na višjih abstraktnih nivojih snovanja. Neupoštevanje zunanjih parametrov lahko povzroči, da moramo sicer korektno zasnovano in logično pravilno rešitev opustiti in poiskati novo, ki bo dovolj upoštevala te zahteve.

Koncept stanja in operacije, kot ga poznamo iz snovanja programske opreme, je izhodiščni koncept, le da ga interpretiramo nekoliko širše. Nabor sestavljenih operacij je drugačen in sestoji v osnovi iz selektorske, sekvenčne in paralelne operacije. Za izgradnjo modelov sekvenčnih krmilnih enot pa je definirana zančna operacija s poljubno mnogo ponovitvami.

Preoblikovanje sestavljenih operacij omogoča nabor transformacij, s katerimi lahko le-te preoblikujemo tako, da najdemo zanje - ali jih sami definiramo - primerne mikroelektronske gradnike, ki so lahko male, srednje, velike, pa tudi zelo velike stopnje integracije, vključno z logičnimi mrežami.

1. STANJA IN OPERACIJE

Koncept stanja in operacije smiselno prilagodimo za potrebe snovanja strojne opreme. S tem bo prehod iz formalizirane specifikacije računalniške strukture na njen logični model razmeroma tekoč. Kot iztočnico uporabimo koncept stanja in operacije, tako kot je specificiran v /1/. Formalnega dela definicije ne bomo spreminjali in

bomo stanje pojmovali samo kot nabor imenovanih vrednosti.

Pojem operacije ponazorimo kot prireditev, ki začetnemu stanju priredi končno stanje.

Formulo, ki določa pogoje za izvajanje operacije povzamemo po / 1/:

$$(\forall s \in S)(Pi(s) \rightarrow Po\ (s,\ ex\ (Op,\ s)))\ . \qquad (1.1)$$

Za vsako stanje s iz prostora stanj S, ki izpolnjuje začetno trditev določeno s predikatom Pi, se z izvajanjem operacije Op določi (izračuna) izhodno stanje ex(Op,s), ki je z začetnim stanjem povezano s končno trditvijo, ki jo specificira predikat Po.

S takšno opredelitvijo stanja in operacije se ne želimo omejiti na krmiljenje operacij, po principu ... izvedi operacijo i, izvedi opredijo i+1 ... .

Če ponovno preberemo (1.1) lahko specificiramo krmilni pogoj za izvajanje operacije tudi takole ..če je zadano stanje s trditev določena s predikatom Pi izpolnjena, tedaj se operacija Op lahko izvede ... . Zadnjo izjavo razširimo takole: ... izvedejo se lahko vse tiste operacije izmed Op_1, Op_2, ..., Op_n, za katere velja, da so pripadajoče trditve Pi_1, Pi_2,..., Pi_n izpolnjene nad stanji s_1, s_2,..., s_n ... .

Za to izjavo najdemo v praksi pogosto naslednji približek: ... izvedejo se lahko vse tiste instrukcije, za katere velja, da imajo veljavne izvorne operande ... .

Če povzamemo, lahko rečemo, da je možno operacije v obeh skrajnostih izvajati s krmilnim oz. podatkovnim pretokom. Formula (1.1) predpisuje samo pogoj, kdaj se operacija lahko izvede, ne predpisuje pa, kdo je tisti, ki ugotavlja izpolnjenost pogoja - človek oz. stroj.

1.1. MODEL PODATKA

Za logični model računalniške strukture prilagojen zapis stanj oz. njihovih komponent, izgradimo model s pomočjo izjav, s katerimi opisujemo elemente izbranih množic, katerih člani so vrednosti komponent stanj, imena komponent stanj, imena operacij, ...

Vzemimo množico elementov D in vsakemu elementu, ki je član te množice, priredimo izjavo, ki le-tega enolično opisuje. Če ima množica D m elementov, je izjav, ki te elemente opisujejo prav tako m. Izjave označimo z D_{m-1}, D_{m-2}, ..., D_0 .

Sedaj pa izberimo še n izjav ob pogoju $m \leq 2^n$, ki jih označimo z A_{n-1}, A_{n-2}, ..., A_0 in zaenkrat ignorirajmo vsebino teh izjav. Če tvorimo vse možne konjunkcije teh izjav, s tem da pravilnostne vrednosti izjavam sami predpišemo dobimo:

$$\bar{A}_{n-1}\wedge\bar{A}_{n-2}\wedge\ldots\wedge\bar{A}_1\wedge\bar{A}_0$$
$$\bar{A}_{n-1}\wedge\bar{A}_{n-2}\wedge\ldots\wedge\bar{A}_1\wedge A_0$$
$$\bar{A}_{n-1}\wedge\bar{A}_{n-2}\wedge\ldots\wedge A_1\wedge\bar{A}_0$$
$$\bar{A}_{n-1}\wedge\bar{A}_{n-2}\wedge\ldots\wedge A_1\wedge A_0 \qquad (1.1.1)$$
$$\vdots$$
$$A_{n-1}\wedge A_{n-2}\wedge\ldots\wedge A_1\wedge\bar{A}_0$$
$$A_{n-1}\wedge A_{n-2}\wedge\ldots\wedge A_1\wedge A_0\ .$$

Sedaj pa tvorimo m ekvivalenc, tako da vsaki izjavi izmed D_0, D_1, ... , D_{m-1} predpišemo kot ekvivalentno izjavo poljubno konjunkcijo izmed (1.1.1), vendar tako, da bo prireditev enolična.

Za zgled predpostavimo, da je $m=2^n$ in tvorimo eno izmed možnih prireditev.

$$\bar{A}_{n-1}\wedge\bar{A}_{n-2}\wedge\ldots\wedge\bar{A}_1\wedge\bar{A}_0 = D_0$$
$$\bar{A}_{n-1}\wedge\bar{A}_{n-2}\wedge\ldots\wedge\bar{A}_1\wedge A_0 = D_1$$
$$\bar{A}_{n-1}\wedge\bar{A}_{n-2}\wedge\ldots\wedge A_1\wedge\bar{A}_0 = D_2$$
$$\bar{A}_{n-1}\wedge\bar{A}_{n-2}\wedge\ldots\wedge A_1\wedge A_0 = D_3 \qquad (1.1.2)$$
$$\vdots$$
$$A_{n-1}\wedge A_{n-2}\wedge\ldots\wedge A_1\wedge\bar{A}_0 = D_{m-2}$$
$$A_{n-1}\wedge A_{n-2}\wedge\ldots\wedge A_1\wedge A_0 = D_{m-1}$$

Izjavam A_{n-1}, A_{n-2},...,A_1, A_0 priredimo pravilnostne vrednosti glede (1.1.2) in dobimo:

A_{n-1}	A_{n-2}	...	A_1	A_0	
0	0	...	0	0	D_0
0	0	...	0	1	D_1
0	0	...	1	0	D_2
0	0	...	1	1	D_3
		.			
		.			
		.			
1	1	...	1	0	D_{m-2}
1	1	...	1	1	D_{m-1}

(1.1.3)

Če si zapomnimo prireditev (1.1.3) lahko z nizi pravilnostnih vrednosti izjav A_{n-1}, A_{n-2},..., A_1, A_0 ponazorimo elemente množice D.

Na opisan način lahko praktično elemente poljubnih mno-

žic priredimo za logični nivo pri izgradnji modelov računalniških struktur.

Doslej nas notranja zgradba izjav ni posebej zanimala. Včasih pa lahko z upoštevanjem notranje zgradbe izjav, izgradimo modele, ki imajo podobne lastnosti kot originalni podatki. Takšen pristop nam včasih olajša izgradnjo modelov operacij nad tako modeliranimi podatki.

Kot zgled uporabimo množico dvojiških števil, ki jo opišemo z izrazom:

$$a_{n-1} \cdot 2^{n-1} + \ldots + a_1 \cdot 2^1 + a_0 \cdot 2^0 , \qquad (1.1.4)$$

kjer je $a_i \in \{0, 1\}_2$ in $i = 0, 1, \ldots, n-1$.

Sedaj specificirajmo n izjav takole:

Koeficient a_i ima vrednost 1_2. (1.1.5)

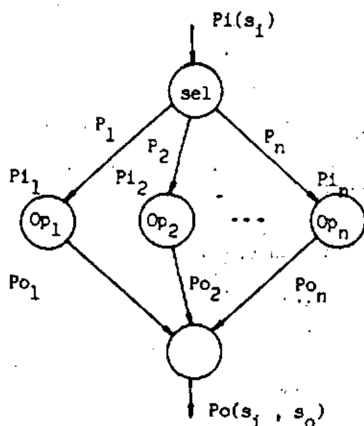
$a_i = 0, 1, \ldots, n-1$

Izjava je pravilna, če je trditev resnična, drugače je napačna. Pri takšnem modelu lahko na primer seštevalnik po mod_2 nad dvojiškimi števili ponazorimo z operacijo logične ekvivalence nad izjavami (1.1.5).

2. SESTAVLJENE OPERACIJE

2.1. SELEKTORSKA OPERACIJA

Selektorsko operacijo ponazorimo z označenim usmerjenim grafom na sliki 2.1.1.



Slika 2.1.1: Graf selektorske operacije

Selektorska operacija je sestavljena operacija, kjer se naenkrat lahko izvede samo ena izmed operacij Op_1, $Op_2, \ldots, Op_n$. Katera operacija se bo izvedla, je odvisno od pravilnosti ene izmed izjav $P_1, P_2, \ldots P_n$, ki jih definiramo takole:

$$P_1 = R_1(e_1(s_i), s_i) \wedge \overline{R_2(e_2(s_i), s_i)} \wedge \ldots \wedge \overline{R_n(e_n(s_i), s_i)}$$
$$P_2 = \overline{R_1(e_1(s_i), s_i)} \wedge R_2(e_2(s_i), s_i) \wedge \ldots \wedge \overline{R_n(e_n(s_i), s_i)} .$$
$$\vdots$$
$$P_n = \overline{R_1(e_1(s_1), s_i)} \wedge \overline{R_2(e_2(s_i), s_i)} \wedge \ldots \wedge R_n(e_n(s_i), s_i) . \qquad (2.1.1)$$

V izrazih (2.1.1) so $R_1, R_2, \ldots, R_n$ predikati, s_i je začetno stanje, e_j, $j = 1, 2, \ldots, n$ pa so funkcije.

Logična pravila, s katerimi opišemo selektorsko operacijo, so:

$$Pi(s_i) \wedge P_1 \rightarrow Pi_1(s_i)$$
$$Pi(s_i) \wedge P_2 \rightarrow Pi_2(s_i)$$
$$\vdots$$
$$Pi(s_i) \wedge P_n \rightarrow Pi_n(s_i) \qquad (2.1.2)$$

$$Pi(s_i') \wedge P_1 \wedge Po_1(s_i', s_0) \rightarrow Po(s_i', s_0)$$
$$Pi(s_i') \wedge P_2 \wedge Po_2(s_i', s_0) \rightarrow Po(s_i', s_0)$$
$$\vdots$$
$$Pi(s_i') \wedge P_n \wedge Po_n(s_i', s_0) \rightarrow Po(s_i', s_0) .$$

Nad spodnjo polovico izrazov (2.1.2) uporabimo formulo $0 \vee \ldots \vee 0 \vee A \vee 0 \vee \ldots \vee 0 = A$ in dobimo:

$$(Pi(s_i') \wedge P_1 \wedge Po_1(s_i', s_0) \vee$$
$$\vee Pi(s_i') \wedge P_2 \wedge Po_2(s_i', s_0) \vee$$
$$\vdots \qquad (2.1.3)$$
$$\vee Pi(s_i') \wedge P_n \wedge Po_n(s_i', s_0)) \rightarrow Po(s_i', s_0) .$$

Če upoštevamo še: $(A \wedge B) \vee \ldots \vee (A \wedge C) = A \wedge (B \vee \ldots \vee C)$ dobimo naslednji izraz:

$$Pi(s_i') \wedge [P_1 \wedge Po_1(s_i', s_0) \vee P_2 \wedge Po_2(s_i', s_0) \vee \ldots$$
$$\ldots \vee P_n \wedge Po_n(s_i', s_0)] \rightarrow Po(s_i', s_0) . \qquad (2.1.4)$$

V izrazih (2.1.2) do (2.1.4) je s_i' spremenjeno stanje s_i, ki ga povzroči operacija Op_1 ali Op_2 ali ... .

Zapis (2.1.4) sicer s stališča snovanja programske opreme ni posebno zanimiv, vendar je njegova zgradba značilna v toliko, da nas navede na definicijo poenostavljene selektorske operacije, ki jo specificirajmo takole:

$$\mathrm{sel}\ (P_j):$$
$$P_1 \rightarrow Op_1$$
$$P_2 \rightarrow Op_2 \qquad (2.1.5)$$
$$\vdots$$
$$P_n \rightarrow Op_n$$

$P_1, P_2, \ldots, P_n$ so izjave, $Op_1, Op_2, \ldots, Op_n$ pa izjave ali podatki modelirani v smislu razdelka (1.1). Za izjave $P_1, P_2, \ldots, P_n$ ponovno velja pogoj, da je lahko naenkrat pravilna samo ena izmed njih.

Z izrazom $A \wedge (A \rightarrow B)$ zožimo pravilnostni prostor implikacije, tako da je enak prostoru pravilnosti konjunk-

cije $A \wedge B$. Napravimo to za implikacije v (2.1.5).

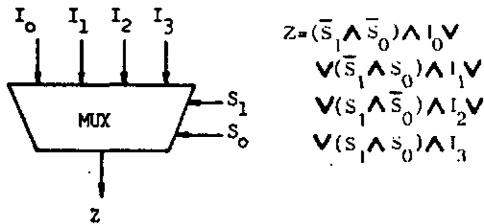
$$P_1 \wedge (P_1 \rightarrow Op_1) = P_1 \wedge Op_1$$
$$P_2 \wedge (P_2 \rightarrow Op_2) = P_2 \wedge Op_2 \qquad (2.1.6)$$
$$\vdots$$
$$P_n \wedge (P_n \rightarrow Op_n) = P_n \wedge Op_n$$

Upoštevamo $0 \vee \ldots \vee 0 \vee A \vee 0 \vee \ldots \vee 0 = A$ in zapišemo:

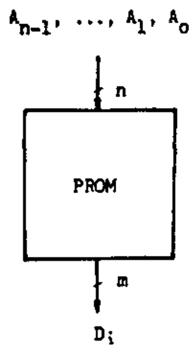
$$P_1 \wedge Op_1 \vee P_2 \wedge Op_2 \vee \ldots \vee P_n \wedge Op_n . \qquad (2.1.7)$$

Izraz (2.1.7) sicer strogo gledano ni ekvivalenten zapisu (2.1.5), vendar se dogovorimo, da bomo (2.1.5) brali takole ... če je Pi pravilna, tedaj je pravilna tudi Op_i .... .

Na sliki 2.1.2 so podani zgledi modelov nekaterih tipičnih gradnikov s pomočjo poenostavljene selektorske operacije.

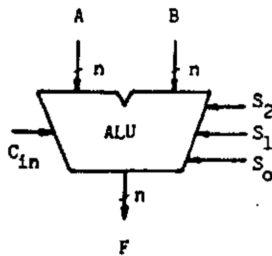


a) model multipleksirnika



$$D = (\bar{A}_{n-1} \wedge \ldots \wedge \bar{A}_1 \wedge \bar{A}_0) \wedge D_0 \vee$$
$$\vee (\bar{A}_{n-1} \wedge \ldots \wedge \bar{A}_1 \wedge A_0) \wedge D_1 \vee$$
$$\vee (\bar{A}_{n-1} \wedge \ldots \wedge A_1 \wedge \bar{A}_0) \wedge D_2 \vee$$
$$\vdots$$
$$\vee (A_{n-1} \wedge \ldots \wedge A_1 \wedge A_0) \wedge D_{2^n-1}$$

b) model bralnega pomnilnika

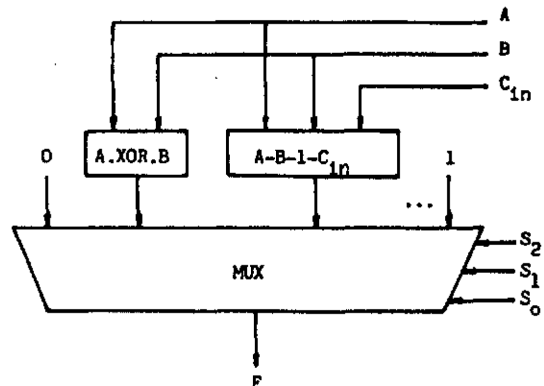


$$F = (\bar{S}_2 \wedge \bar{S}_1 \wedge \bar{S}_0) \wedge \emptyset \vee$$
$$\vee (\bar{S}_2 \wedge \bar{S}_1 \wedge S_0) \wedge (A.XOR.B) \vee$$
$$\vee (\bar{S}_2 \wedge S_1 \wedge \bar{S}_0) \wedge (A - B - 1 + C_{in}) \vee$$
$$\vee (\bar{S}_2 \wedge S_1 \wedge S_0) \wedge (A.AND.B) \vee$$
$$\vee (S_2 \wedge \bar{S}_1 \wedge \bar{S}_0) \wedge (B - A - 1 + C_{in}) \vee$$
$$\vee (S_2 \wedge \bar{S}_1 \wedge S_0) \wedge (A . OR . B) \vee$$
$$\vee (S_2 \wedge S_1 \wedge \bar{S}_0) \wedge (A + B + C_{in}) \vee$$
$$\vee (S_2 \wedge S_1 \wedge S_0) \wedge 1$$

c) model ALU operacijske enote

Slika 2.1.2: Zgledi uporabe poenostavljene selektorske operacije za izgradnjo logičnih modelov

Pri tem smo ponovno predpostavili, da so skrajno desni konjunktivni členi ponazorjeni v smislu razdelka 1.1. Zaradi lažje orientacije je na sliki 2.1.3 podan še eden izmed možnih strukturnih modelov za c) s slike 2.1.2.



Slika 2.1.3: Blokovna shema možnega strukturnega modela za c) 2.1.2

2.2. SEKVENČNA OPERACIJA

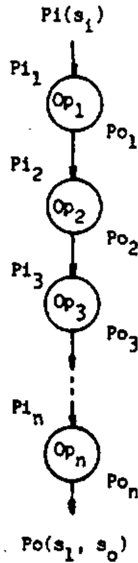
Pri specifikaciji sekvenčne operacije izhajamo iz grafa na sliki 2.2.1.

Logična pravila za sekvenčno operacijo zapišemo takole:

$$Pi(s_i) \rightarrow Pi_1(s_i)$$
$$Pi(s_1) \wedge Po_1(s_1, s_2) \rightarrow Pi_2(s_2) \qquad (2.2.1)$$
$$Pi(s_1) \wedge Po_1(s_1, s_2) \wedge Po_2(s_2, s_3) \rightarrow Pi_3(s_3) \quad \ldots$$
$$Pi(s_1) \wedge Po_1(s_1, s_2) \wedge Po_2(s_2, s_3) \wedge \ldots \wedge Po_{n-1}(s_{n-1}, s_n) \rightarrow$$
$$\rightarrow Pi_n(s_n)$$

$$Pi(s_1) \wedge Po_1(s_1, s_2) \wedge Po_2(s_2, s_3) \wedge \ldots \wedge Po_n(s_n, s_0) \rightarrow$$
$$\rightarrow Po(s_1, s_0).$$

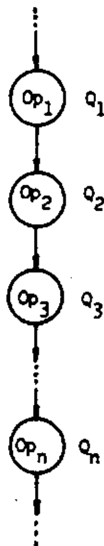
V (2.2.1) smo upoštevali, da lahko Op_1 spremeni kom-



Slika 2.2.1: Graf sekvenčne operacije

ponente v s_i, ki zato preide v s_1.

Do zanimivih zaključkov pridemo, če izgradimo sekvenčno krmiljen model sekvenčne operacije. V ta namen predpostavimo, da je sekvenčna operacija s slike 2.2.1 del sestavljene operacije in jo ponazorimo po sliki 2.2.2.

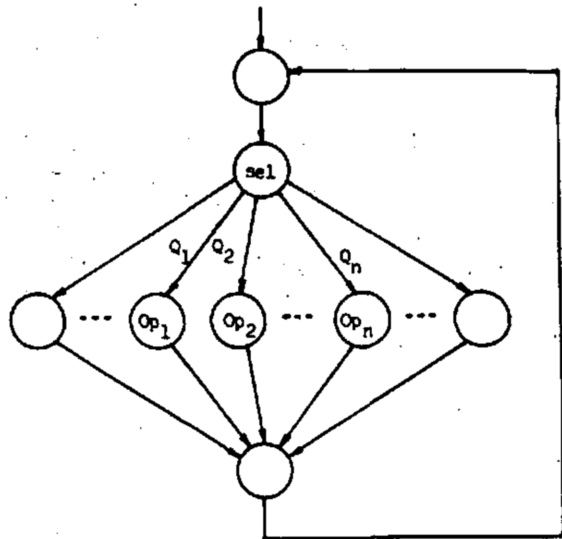


Slika 2.2.2: Prirejen graf sekvenčne operacije

Vozliščem grafa na sliki 2.2.2 smo pripisali izjave $Q_1, Q_2, \ldots, Q_n$. Izjava Q_l, $l = 1, 2, \ldots, n$ je pravilna tedaj in le tedaj, ko je glede na sekvenčno operacijo s slike 2.2.1 izpolnjena pripadajoča izjava $Pi_l(s_l)$ in se operacija Op_l še ni izvedla. Tedaj lahko graf s slike 2.2.2 preoblikujemo v selektorsko operacijo in zapišemo:

$$
\begin{array}{l}
\underline{1}\text{: sel } (Q_l): \\
\quad Q_1 \rightarrow Op_1 \\
\quad Q_2 \rightarrow Op_2 \\
\quad \vdots \\
\quad Q_n \rightarrow Op_n \\
\quad \vdots \\
\qquad \underline{1}\ .
\end{array}
\qquad (2.2.2)
$$

Na sliki 2.2.3 je podan nekoliko prilagojen graf selektorske operacije, s katerim ponazorimo krmiljenje izvajanja sekvenčne operacije s slike 2.2.1.



Slika 2.2.3: Graf modela krmiljenja izvajanja sekvenčne operacije

K dosedanjim izvajanjem sekvenčnega krmilnega modela in sliki 2.2.3 pripomnimo, da bo sekvenčno krmiljenje operacij podrobneje obdelano v razdelku 2.5.

2.3. PARALELNA OPERACIJA

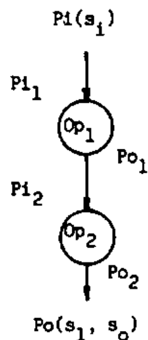
V dosedanjih izvajanjih sekvenčne operacije smo predpostavljali, da je začetna trditev $Pi_j(s_j)$ operacije Op_j pravilna šele, ko se izvedejo vse operacije $Op_1, Op_2, \ldots, Op_{j-1}$. Pri izpeljavi paralelne operacije pa sprostimo ta pogoj.

Izhajajmo iz grafa sekvenčne operacije na sliki 2.3.1.

Logična pravila za takšno sekvenčno operacijo so:

$$
\begin{array}{l}
Pi(s_i) \rightarrow Pi_1(s_i) \\
Pi(s_1) \wedge Po_1(s_1, s_2) \rightarrow Pi_2(s_2) \\
Pi(s_1) \wedge Po_1(s_1, s_2) \wedge Po_2(s_2, s_3) \rightarrow Po(s_1, s_3)\ .
\end{array}
\qquad (2.3.1)
$$

Sedaj pa predpostavimo, da velja:



Slika 2.3.1: Graf dveh sekvenčno povezanih operacij

$$Pi(s_i) \rightarrow Pi_1(s_i)$$

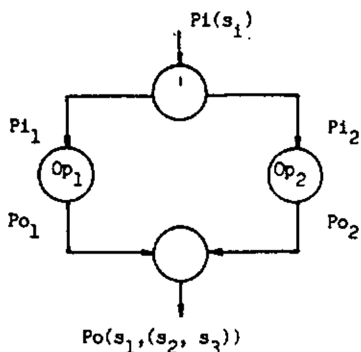
$$Pi(s_i) \rightarrow Pi_2(s_i) \quad , \qquad (2.3.2)$$

da je $Pi_2(s_i)$ pravilen neodvisno od tega, ali se je Op_1 že izvršila ali ne. Za končni pogoj lahko tedaj zapišemo:

$$Pi_1(s_1) \wedge Po_1(s_1,s_2) \wedge Po_2(s_1,s_3) \rightarrow$$

$$\rightarrow Po(s_1,(s_2,s_3)). \qquad (2.3.3)$$

Z (s_2,s_3) smo označili konkatenacijo s_2 in s_3. Za ponazoritev paralelne operacije vpeljemo graf, ki ga podaja slika 2.3.2.



Slika 2.3.2: Graf paralelne operacije

Izvedimo še posplošitev paralelne operacije na n paralelno povezanih operacij in zapišimo logična pravila:

$$Pi(s_i) \rightarrow Pi_1(s_i)$$

$$Pi(s_i) \rightarrow Pi_2(s_i) \qquad (2.3.4)$$

$$\vdots$$

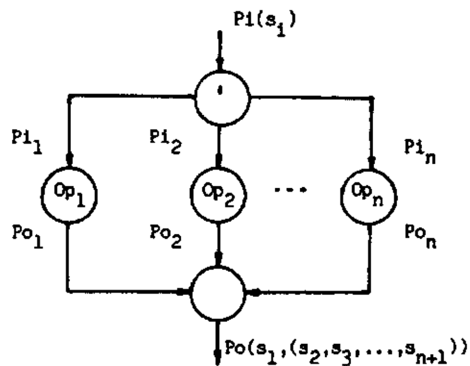
$$Pi(s_i) \rightarrow Pi_n(s_i)$$

$$Pi(s_1) \wedge Po_1(s_1,s_2) \wedge Po_2(s_1,s_3) \wedge \ldots \wedge Po_n(s_1,s_{n+1}) \rightarrow$$

$$\rightarrow Po(s_1, (s_2, s_3, \ldots, s_{n+1})) .$$

Pripadajoč graf podaja slika 2.3.3.

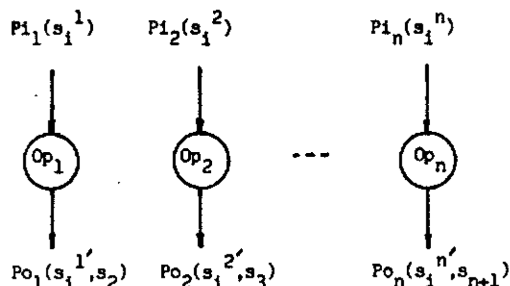
Pogoj za paralelno izvajanje operacij je v bistvu ta, da



Slika 2.3.3: Graf n paralelno povezanih operacij

operacija Op_j ne spremeni tistih komponent stanja s_i, ki so tudi začetne komponente za $Op_1, Op_2, \ldots, Op_{j-1}, Op_{j+1}, \ldots, Op_n$. Enako velja tudi za vse ostale operacije. Operacija tedaj lahko spremeni samo tiste vhodne komponente v stanju s_i, ki so vhodne komponente samo te operacije, sicer se mehanizem paralelnega izvajanja poruši.

Takšno paralelno operacijo lahko tedaj razstavimo na komponente, med katerimi ni več nobene povezave. Slika 2.3.4 podaja tako razgrajeno paralelno operacijo.



Slika 2.3.4: Grafi operacij, ki se lahko izvajajo paralelno

2.4. PREOBLIKOVANJE SESTAVLJENIH OPERACIJ

Razdelek podaja nekatere možnosti preoblikovanja sestavljenih operacij.

a) Preoblikovanje sestavljene selektorske operacije v selektorsko operacijo.

Selektorsko operacijo s slike 2.4.1 zapišimo v disjunktivni obliki.

$$(Q_1 \wedge Q_2 \wedge \ldots \wedge Q_{n-1} \wedge Q_n) \wedge Op_1 \vee$$

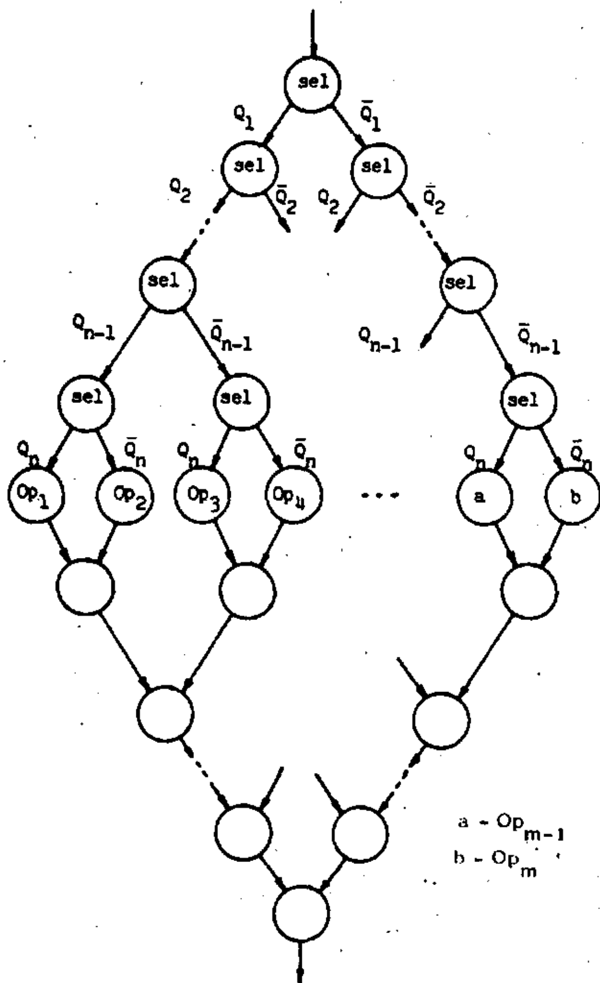
$$\vee (Q_1 \wedge Q_2 \wedge \ldots \wedge Q_{n-1} \wedge \bar{Q}_n) \wedge Op_2 \vee$$

$$\vee (Q_1 \wedge Q_2 \wedge \ldots \wedge \bar{Q}_{n-1} \wedge Q_n) \wedge Op_3 \vee$$

$$\vee (Q_1 \wedge Q_2 \wedge \ldots \wedge \bar{Q}_{n-1} \wedge \bar{Q}_n) \wedge Op_4 \vee \qquad (2.4.1)$$

$$\vdots$$

$$\vdots$$
$$\vee(\bar{Q}_1\wedge\bar{Q}_2\wedge\ldots\wedge\bar{Q}_{n-1}\wedge Q_n)\wedge Op_{m-1}\vee$$
$$\vee(\bar{Q}_1\wedge\bar{Q}_2\wedge\ldots\wedge\bar{Q}_{n-1}\wedge\bar{Q}_n)\wedge Op_m \qquad (2.4.1)$$



Slika 2.4.1: Sestavljena selektorska operacija

Konjunkcije izjav Q_i, $i = 1, 2, \ldots, n$ v (2.4.1) poimenujmo s $P_1, P_2, \ldots, P_m$, $m=2^n$ in dobimo:

$$P_1\wedge Op_1\vee P_2\wedge Op_2\vee P_3\wedge Op_3\vee P_4\wedge Op_4\vee \ldots$$
$$\ldots\vee P_{m-1}\wedge Op_{m-1}\vee P_m\wedge Op_m . \qquad (2.4.2)$$

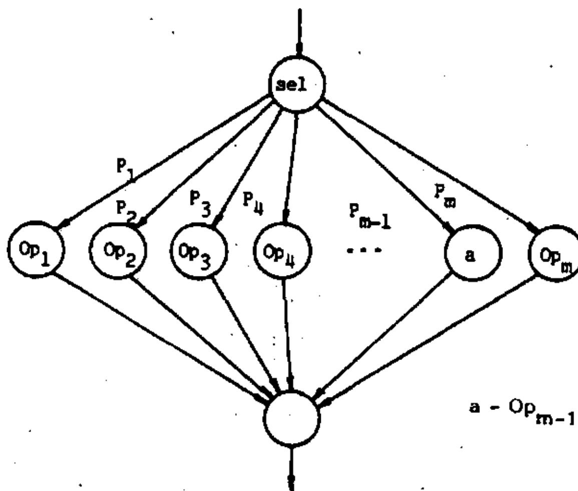
Z izrazom (2.4.2) pa lahko opišemo tudi selektorsko operacijo, katere graf podaja slika 2.4.2.

b) Preoblikovanje paralelne selektorske operacije v paralelno povezane selektorske operacije

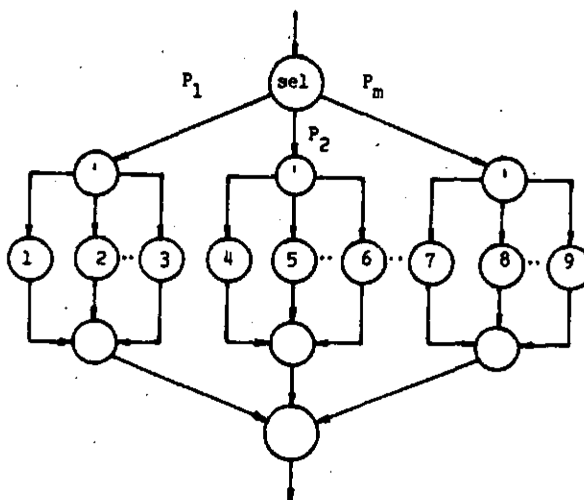
Sliko 2.4.3 opišemo s poenostavljeno selektorsko operacijo:

sel (P_i):

$$P_1 \rightarrow (Op_{11}, Op_{12}, \ldots, Op_{1n})$$
$$P_2 \rightarrow (Op_{21}, Op_{22}, \ldots, Op_{2n}) \qquad (2.4.3)$$
$$\vdots$$
$$P_m \rightarrow (Op_{m1}, Op_{m2}, \ldots, Op_{mn})$$



Slika 2.4.2: Graf selektorske operacije za izraz 2.4.2



1 - Op_{11}	4 - Op_{21}	7 - Op_{m1}
2 - Op_{12}	5 - Op_{22}	8 - Op_{m2}
3 - Op_{1n}	6 - Op_{2n}	9 - Op_{mn}

Slika 2.4.3: Graf paralelne selektorske operacije

in preoblikujemo v disjunktivno obliko:

$$P_1\wedge(Op_{11}, Op_{12}, \ldots, Op_{1n})\vee P_2\wedge(Op_{21}, Op_{22}, \ldots, Op_{2n})\vee\ldots$$
$$\ldots\vee P_m\wedge(Op_{m1}, Op_{m2}, \ldots, Op_{mn}). \qquad (2.4.4)$$

Izraz (2.4.4) zapišemo po komponentah.

$$P_1\wedge Op_{11}\vee P_2\wedge Op_{21}\vee\ldots\vee P_m\wedge Op_{m1}$$
$$P_1\wedge Op_{12}\vee P_2\wedge Op_{22}\vee\ldots\vee P_m\wedge Op_{m2} \qquad (2.4.5)$$
$$\vdots$$

$\vdots$

$P_1 \wedge Op_{1n} \vee P_2 \wedge Op_{2n} \vee \ldots \vee P_m \wedge Op_{mn}$

V izraze (2.4.5) pa lahko preoblikujemo tudi naslednje selektorske operacije:

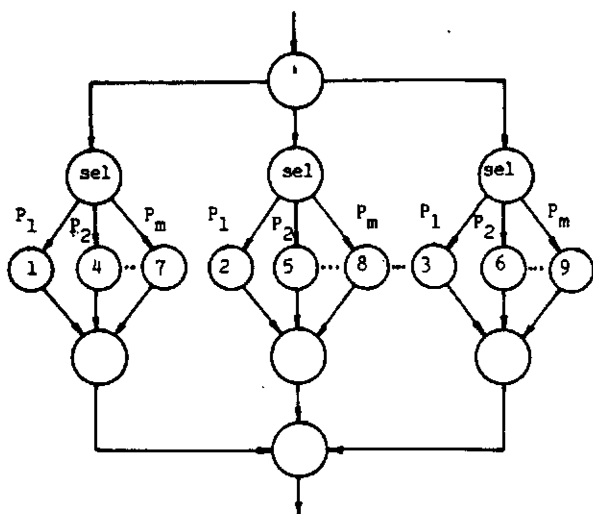
sel (P_i):
$P_1 \rightarrow Op_{11}$
$P_2 \rightarrow Op_{21}$
$\vdots$
$P_m \rightarrow Op_{m1}$

sel (P_i):
$P_1 \rightarrow Op_{12}$
$P_2 \rightarrow Op_{22}$
$\vdots$
$P_m \rightarrow Op_{m2}$

$\vdots$

sel (P_i): (2.4.6)
$P_1 \rightarrow Op_{1n}$
$P_2 \rightarrow Op_{2n}$
$\vdots$
$P_m \rightarrow Op_{mn}$.

Graf selektorskih operacij (2.4.6) je podan na sliki 2.4.4, kot paralelno povezane selektorske operacije.



Opomba: koda operacij je na sliki 2.4.3.

Slika 2.4.4: Graf paralelno povezanih selektorskih operacij

c) Preoblikovanje paralelne selektorske operacije, kadar se operacije ponavljajo

Primerov za ponavljanje operacij v paralelni selektorski operaciji je veliko. Omenimo samo mikroprogramirano krmilno enoto, asociativni pomnilnik, razne registerske strukture, ki omogočajo paralelen dostop do podatkov, procesorje z množico funkcijskih enot, itd.

Ponazoritev takšnih sklopov s paralelno selektorsko operacijo ima svojo težo, saj jih lahko tako na višjih abstraktnih nivojih snovanja ponazorimo v koncentriranem zapisu, ki ga postopoma razgrajujemo z napredovanjem pri izgradnji modela.

Preoblikovanje paralelne selektorske operacije z lastnostjo ponavljanja operacij ponazorimo z zgledom:

sel (P_i):
$P_0 \rightarrow$ A, 1, a
$P_1 \rightarrow$ A, 1, b
$P_2 \rightarrow$ A, 1, c
$P_4 \rightarrow$ A, 2, a
$P_5 \rightarrow$ A, 2, b
$P_6 \rightarrow$ A, 2, c
$P_8 \rightarrow$ A, 3, a
$P_9 \rightarrow$ A, 3, b
$P_{10} \rightarrow$ A, 3, c
$P_{16} \rightarrow$ B, 1, a
$P_{17} \rightarrow$ B, 1, b
$P_{18} \rightarrow$ B, 1, c
$P_{20} \rightarrow$ B, 2, a
$P_{21} \rightarrow$ B, 2, b
$P_{22} \rightarrow$ B, 2, c
$P_{24} \rightarrow$ B, 3, a
$P_{25} \rightarrow$ B, 3, b
$P_{26} \rightarrow$ B, 3, c

(2.4.7)

P_i					
P_0	$\bar{Q}_4$	$\bar{Q}_3$	$\bar{Q}_2$	$\bar{Q}_1$	$\bar{Q}_0$
P_1	$\bar{Q}_4$	$\bar{Q}_3$	$\bar{Q}_2$	$\bar{Q}_1$	Q_0
P_2	$\bar{Q}_4$	$\bar{Q}_3$	$\bar{Q}_2$	Q_1	$\bar{Q}_0$
P_4	$\bar{Q}_4$	$\bar{Q}_3$	Q_2	$\bar{Q}_1$	$\bar{Q}_0$
P_5	$\bar{Q}_4$	$\bar{Q}_3$	Q_2	$\bar{Q}_1$	Q_0
P_6	$\bar{Q}_4$	$\bar{Q}_3$	Q_2	Q_1	$\bar{Q}_0$
P_8	$\bar{Q}_4$	Q_3	$\bar{Q}_2$	$\bar{Q}_1$	$\bar{Q}_0$
P_9	$\bar{Q}_4$	Q_3	$\bar{Q}_2$	$\bar{Q}_1$	Q_0
P_{10}	$\bar{Q}_4$	Q_3	$\bar{Q}_2$	Q_1	$\bar{Q}_0$
P_{16}	Q_4	$\bar{Q}_3$	$\bar{Q}_2$	$\bar{Q}_1$	$\bar{Q}_0$
P_{17}	Q_4	$\bar{Q}_3$	$\bar{Q}_2$	$\bar{Q}_1$	Q_0
P_{18}	Q_4	$\bar{Q}_3$	$\bar{Q}_2$	Q_1	$\bar{Q}_0$
P_{20}	Q_4	$\bar{Q}_3$	Q_2	$\bar{Q}_1$	$\bar{Q}_0$
P_{21}	Q_4	$\bar{Q}_3$	Q_2	$\bar{Q}_1$	Q_0
P_{22}	Q_4	$\bar{Q}_3$	Q_2	Q_1	$\bar{Q}_0$
P_{24}	Q_4	Q_3	$\bar{Q}_2$	$\bar{Q}_1$	$\bar{Q}_0$
P_{25}	Q_4	Q_3	$\bar{Q}_2$	$\bar{Q}_1$	Q_0
P_{26}	Q_4	Q_3	$\bar{Q}_2$	Q_1	$\bar{Q}_0$

(2.4.8)

(2.4.8) je tabela izjav, ki jih priredimo izjavam P_i. Paralelno selektorsko operacijo (2.4.7) lahko sedaj nadomestimo s tremi selektorskimi operacijami.

sel (Q_4):
$\bar{Q}_4 \rightarrow$ A
$Q_4 \rightarrow$ B

sel (Q_3, Q_2):
$\bar{Q}_3 \wedge \bar{Q}_2 \rightarrow$ 1
$\bar{Q}_3 \wedge Q_2 \rightarrow$ 2
$Q_3 \wedge \bar{Q}_2 \rightarrow$ 3

sel (Q_1, Q_0):
$\bar{Q}_1 \wedge \bar{Q}_0 \rightarrow$ a
$\bar{Q}_1 \wedge Q_0 \rightarrow$ b (2.4.9)
$Q_1 \wedge \bar{Q}_0 \rightarrow$ c

(2.4.9) lahko glede na točko b) ponazorimo v grafu kot tri paralelno povezane selektorske operacije.

2.5. MODEL SEKVENČNEGA STROJA

Izhajamo iz implikacij:

$$\wedge p\,(I, Q) \rightarrow Q$$
$$\wedge p\,(I, Q) \rightarrow Z \quad , \qquad (2.5.1)$$

kjer smo z I ponazorili niz izjav $(i_{m-1}, i_{m-2}, \ldots, i_0)$ in z Q niz izjav $(q_{n-1}, q_{n-2}, \ldots, q_0)$. Z $\wedge(I, Q)$ označimo vse možne konjunkcije sestavljenega niza (I, Q),

z $\Lambda p(I, Q)$ pa izbor poljubnega števila konjunkcij iz $\Lambda(I, Q)$, Z pa naj bo znak za niz $(z_{r-1}, z_{r-2}, \ldots, z_0)$.

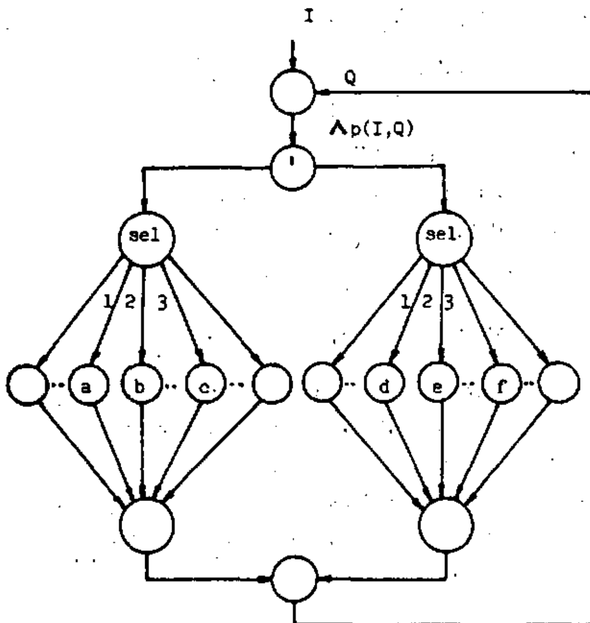
(2.5.1) lahko ponazorimo s paralelno selektorsko operacijo:

sel (P_i):

$$
\begin{array}{l}
\vdots \\
P_i \to Z_i, P_{i+1} \\
P_{i+1} \to Z_{i+1}, P_{i+2} \\
\vdots \\
P_{i+v} \to Z_{i+v}, P_{i+v+1} \\
\vdots
\end{array}
\qquad (2.5.2)
$$

kjer smo s ..., P_i, ... označili konjunkcije iz tabele $\Lambda p(I,Q)$. Pri tem v splošnem ne zahtevamo enoličnosti prireditev ... Z_i ..., niti desne strani ... P_{i+1} ... v (2.5.2), oz. isti Z_i in P_i se lahko na desni pojavijo večkrat, medtem pa mora biti leva stran (2.5.2) enolična.

(2.5.2) ponazorimo z grafom paralelne selektorske operacije na sliki 2.5.1.



a - Z_i d - Q_{i+1} 1 - P_i
b - Z_{i+1} e - Q_{i+2} 2 - P_{i+1}
c - Z_{i+v} f - Q_{i+v+1} 3 - P_{i+v}

Slika 2.5.1: Graf vase zaključene paralelne selektorske operacije

Tako definiramo selektorsko operacijo lahko imenujemo model Mealyjevega stroja, če napravimo primerjavo med (2.5.1) in

$$
\begin{array}{l}
f : I \times Q \to Q \\
g : I \times Q \to Z ,
\end{array}
\qquad (2.5.3)
$$

ter so I - vhodi, Q - stanja, Z - izhodi in f in g preslikavi.

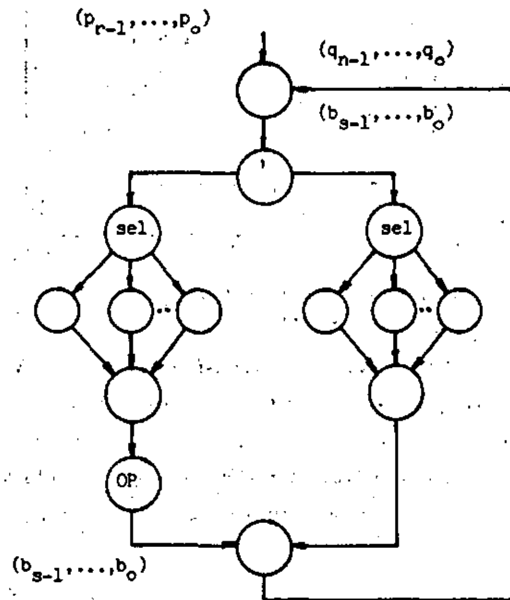
Pravkar opisani model nekoliko dopolnimo, tako da ga bolj približamo obliki s kakršno imamo opravka v praksi snovanja strojne opreme. Za ta namen najprej definirajmo rep in glavo niza $I = (i_{m-1}, i_{m-2}, \ldots, i_j, i_{j-1}, \ldots, i_0)$ ter poimenujmo glavo niza I s P in rep niza I z B in ju označimo takole:

$$
\begin{array}{l}
P = (p_{r-1}, p_{r-2}, \ldots, p_0) \\
B = (b_{s-1}, b_{s-2}, \ldots, b_0) ,
\end{array}
\qquad (2.5.4)
$$

kjer so $r-1 = m-1$, $r-2 = m-2$, ... in $s-1 = j-1$, $s-2 = j-2$, ..

Niz P imenujmo zunanje vhodne izjave, niz B pa notranje vhodne izjave.

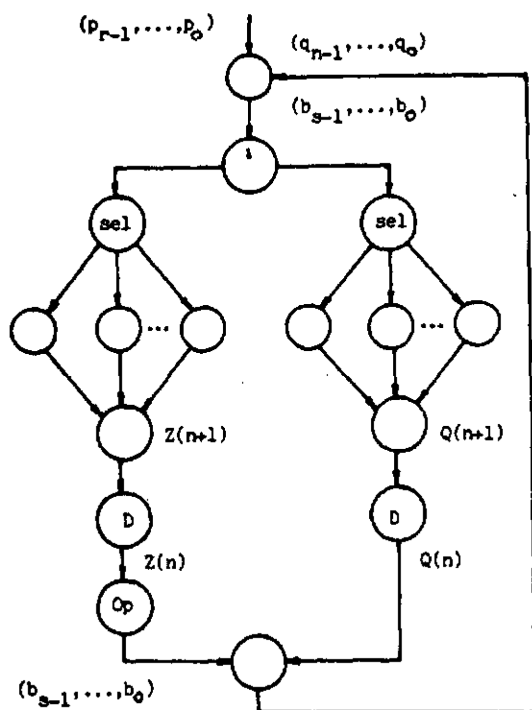
Sedaj pa narišimo nekoliko modificiran graf s slike 2.5.1 na sliki 2.5.2.



Slika 2.5.2: Modificiran graf paralelne selektorske operacije s slike 2.5.1

V sliko (2.5.2) smo vgradili operacijsko enoto OP, ki izvaja operacije določene z Z in kot rezultat daje izjave B o izvedenih operacijah. Podatkovni del operacijske enote nas zaenkrat ne zanima. $(b_{s-1}, b_{s-2}, \ldots, b_0)$ imenujemo vejitvene pogoje, ker omogočajo izvajanje vejitev v grafih operacij, ki jih modeliramo na takšnem modelu.

Sedaj pa vgradimo v naš model še mehanizem, ki omogoča časovno prekrivanje operacij med krmilno in operacijsko strukturo modela, kadar so izjave (b_{s-1}, b_{s-2}, ..., b_0) neaktivne. V ta namen oblikujmo graf našega modela po sliki 2.5.3.



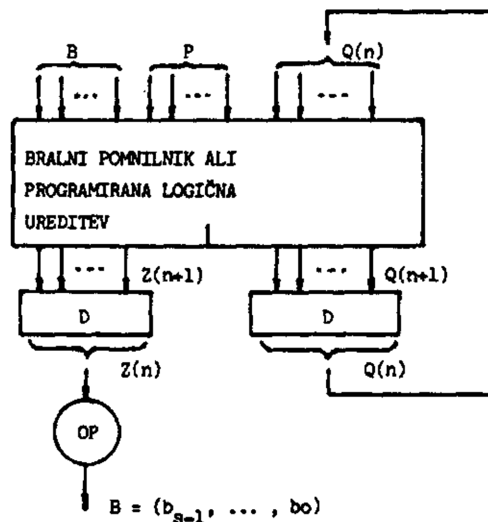
Slika 2.5.3: Model izvajanja operacij s časovnim prekrivanjem

Na sliki 2.5.3 imenujmo D zakasnilni element, ki vhodne izjave prenese na izhod s časovno zakasnitvijo, katere dolžina naj bo zaenkrat določena z zunanjimi pogoji. K grafu, na sliki 2.5.3 narišimo še pripadajočo blokovno shemo, kakršne smo pri snovanju bolj vajeni, na sliki 2.5.4.

Preden nadaljujemo z izgradnjo krmilnega modela definirajmo še modela programirane logične ureditve in bralnega pomnilnika. V ta namen izhajajmo iz poenostavljene selektorske operacije:

$$D_i = A_{m-1} \wedge D_{m-1} \vee A_{m-2} \wedge D_{m-2} \vee \dots$$
$$\dots \vee A_0 \wedge D_0 \ , \qquad (2.5.5)$$

kjer so A_i, i=0, 1,..., m-1 enolično prirejene vsem možnim konjunkcijam nad nizem (a_{n-1}, a_{n-2},..., a_0) in D_i poljubno izbrane - v splošnem ne enolično - iz tabele izjav:



Slika 2.5.4: Blokovna shema modela sekvenčnega stroja z operacijsko enoto

$$\left.\begin{matrix} \bar{d}_{t-1}, \dots, \bar{d}_1, \bar{d}_0 \\ \bar{d}_{t-1}, \dots, \bar{d}_1, d_0 \\ \bar{d}_{t-1}, \dots, d_1, \bar{d}_0 \\ \bar{d}_{t-1}, \dots, d_1, d_0 \\ \vdots \\ d_{t-1}, \dots, d_1, d_0 \ . \end{matrix}\right\} 2^t \qquad (2.5.6)$$

Če "preberemo" iz bralnega pomnilnika z "adrese" A_i "element" D_i lahko to zapišemo takole:

$$D_i = 0 \vee 0 \vee \dots \vee A_i \wedge D_i \vee 0 \vee \dots \vee 0 \qquad (2.5.7)$$
$$= A_i \wedge D_i = 1 \wedge D_i = (d_{t-1}, \dots, d_1, d_0)_i \ .$$

Model programirane logične ureditve je v bistvu identičen s to razliko, da (2.5.5) zapišemo po komponentah in v zapisu izpustimo vse tiste konjunkcije, ki imajo zaradi $d_i = 0$ vrednost 0. V disjunktivne zapise vstavimo tedaj samo tiste komponente d_j, ki imajo vrednost 1, glede na tabelo (2.5.6).

S stališča uporabe lahko tedaj rečemo, da v splošnem ni potrebno razlikovati med bralnim pomnilnikom in programirano logično ureditvijo, saj lahko oba gradnika po potrebi interpretiramo kot bralni pomnilnik oz. programiramo logično ureditev.

Sedaj pa nadaljujmo z izgradnjo modela mikroprogramiranega krmilnika. V ta namen definirajmo selektorsko operacijo, s katero bomo izdelali začetni približek k sekvencerju mikroprogramiranega krmilnika. Iz nizev:

$$P = (p_{n-1}, p_{n-2}, \dots, p_0)$$
$$B = (b_{s-1}, b_{s-2}, \dots, b_0) \qquad (2.5.8)$$
$$Q = (q_{n-1}, q_{n-2}, \dots, q_0)$$

z operacijami glava, rep, delni niz, konkatenacija, glava delnega niza, rep delnega niza, konkatenacija delnega niza tvorimo nize enakih dolžin in jih poimenujmo z $A = (a_{n-1}, a_{n-2}, \ldots, a_0)$. Oglejmo si nekaj primerov tako konstruiranih nizev:

$$
\begin{array}{ll}
q_{n-1}, \ldots\ldots, q_1, q_0 & \\
q_{n-1}, \ldots\ldots, b_{s-1}, b_0 & q_{n-1}, \ldots, b_i, b_j, b_0, b_7, \quad i,j \in \{0, 1, \ldots, s-1\} \\
p_{n-1}, \ldots\ldots, p_1, p_0 &
\end{array}
\qquad (2.5.9)
$$

Pri tem bomo smatrali, da lahko v splošnem vsi elementi tako definiranih nizev zavzamejo vrednosti q_a ali $\bar{q}_a$, b_e ali $\bar{b}_e$ in p_c ali $\bar{p}_c$ in a in $c \in \{0, 1, \ldots, n-1\}$ ter $e \in \{0, 1, \ldots, s-1\}$.

Sedaj pa definirajmo selektorsko operacijo s katero izbiramo nize, ki smo jih konstruirali po zgornjem pravilu.

sel (BR_j):

$$
\begin{array}{l}
BR_1 \longrightarrow \text{niz } 1 \\
BR_2 \longrightarrow \text{niz } 2 \\
\vdots \\
BR_w \longrightarrow \text{niz } w
\end{array}
\qquad (2.5.10)
$$

Za zgled predpostavimo, da ima niz j takšno obliko:

$q_{n-1}, q_{n-2}, \ldots, q_2, b_1, b_0$.

Z BR_j tedaj izberemo enega izmed nizev

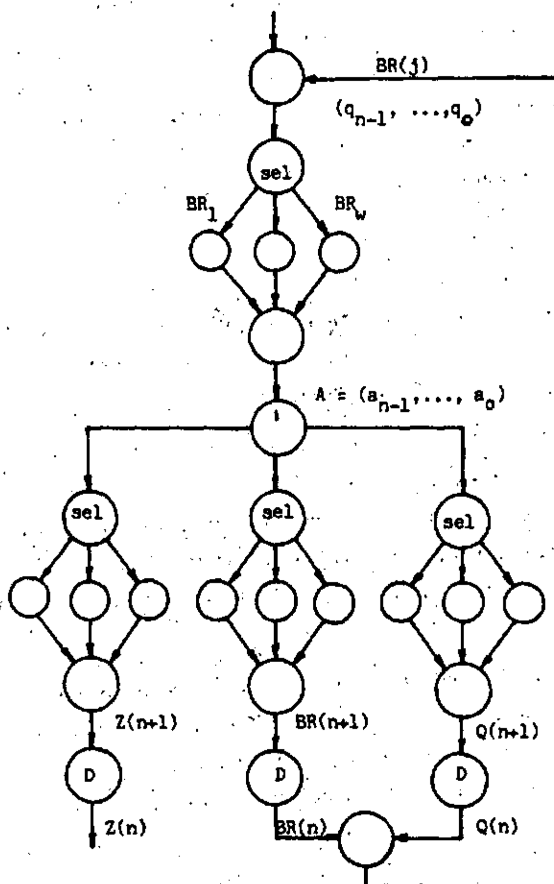
$$
\begin{array}{l}
\bar{q}_{n-1}, \bar{q}_{n-2}, \ldots, \bar{q}_2, \bar{b}_1, \bar{b}_0 \\
\qquad\qquad\qquad\quad \bar{b}_1, b_0 \\
\qquad\qquad\qquad\quad b_1, \bar{b}_0 \\
\qquad\qquad\qquad\quad b_1, b_0 \\
\vdots \\
q_{n-1}, q_{n-2}, \ldots, q_2, b_1, b_0,
\end{array}
$$

odvisno pač od trenutnih vrednosti, ki jih imajo izjave $q_{n-1}, q_{n-2}, \ldots, q_2, b_1, b_0$.

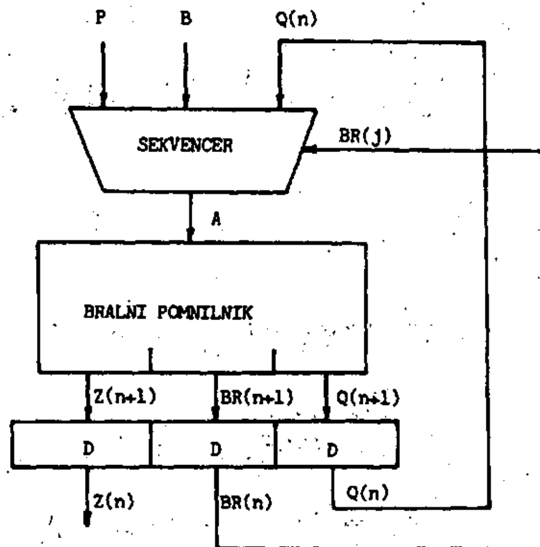
Za selektorsko operacijo (2.5.10) bomo smatrali, da opravlja nalogo sekvencerja adres v začetnem približku k modelu mikroprogramiranega krmilnika, (sl. 2.5.5).

S postopno širitvijo začetnega modela sekvencerja ga lahko opremimo z mehanizmi za strežbo podprogramov, pasti itd., vendar lahko še tako kompleksen sekvencer vedno prevedemo na selektorsko operacijo. Na sliki 2.5.6 je podana blokovna shema krmilnika izdelana na podlagi grafa na sliki 2.5.5.

Omenimo še, čeprav s tem prehajamo zastavljeni okvir tega članka, da se krmilna struktura z izgradnjo mikroprogramskega krmilnika ne konča, ampak v splošnem prehaja v operacijsko enoto, ki smo jo doslej opazovali



Slika 2.5.5: Začetni približek k mikroprogramiranemu krmilniku



Slika 2.5.6: Blokovna shema začetnega približka mikroprogramiranega krmilnika

le kot koncentrirani gradnik strukture. Tudi izgradnjo

modela operacijske enote lahko v splošnem pričnemo s selektorsko operacijo, ki jo razgrajujemo toliko časa, dokler ne pridemo do gradnikov, ki jih s stališča izvajanja operacij lahko pojmujemo kot koncentrirane gradnike. Te gradnike obravnavamo tedaj kot elemente, ki prično izvajati izbrano operacijo z nastopom izvornih operandov oz. v trenutku, ko so izpolnjeni pogoji začetne trditve Pi(s) v (1.1).

3. ZAKLJUČEK

Podani so postopki snovanja z logičnimi modeli računalniških struktur, s katerimi je možen postopen prehod na logično realizacijo izbranih struktur.

Stanja in operacije je možno postopoma razgrajevati v vedno večje detalje, postopek se konča, ko najdemo za sestavljene operacije skupek ustrezno povezanih mikroelektronskih gradnikov, katerih funkcije ustrezajo sestavljenim operacijam, s katerimi smo modelirali izhodiščno strukturo.

Pri predlaganih postopkih snovanja moramo razen notranjih značilnosti strukture upoštevati tudi zunanje parametre - hitrost, cena, zanesljivost, ..., ki v snovanje vnašajo komponento realnega okolja.

Izdelani so modeli sestavljenih operacij, transformacije med njimi in izgrajeni modeli nekaterih realnih logičnih in računalniških struktur.

4. LITERATURA

/1/ C. B. JONES: Software Design: A Rigorous Approach, Prentice-Hall International 1980.

/2/ M. GERKEŠ: Metodologija snovanja računalniških struktur in sistemov z upoštevanjem trendov v razvoju tehnologije in konceptnih rešitev, disertacija, Univerza Edvarda Kardelja v Ljubljani, Fakulteta za elektrotehniko, Ljubljana 1984.

/3/ J. VIRANT: Preklopne funkcije, strukture in sistemi, Univerza Edvarda Kardelja v Ljubljani, Fakulteta za elektrotehniko, Ljubljana 1983.

/4/ M. GERKEŠ, M. PERNEK, M. PAGLAVEC: Aplikacija bipolarnega mikroprocesorja, Poročilo o delu za leto 1984, URP/RP: Računalniška oprema 03-2570, RSS, PORS 3, Visoka tehniška šola, Maribor, 1984.

informatica 85

Posvetovanje in seminarji Informatica '85
Nova Gorica, 24.-27. september 1985

Posvetovanje
18. jugoslovansko mednarodno posvetovanje za računalniško tehnologijo in uporabo
Nova Gorica, 24.-27. september 1985

Seminarji
Izbrana poglavja iz računalniške tehnologije in uporabe

Razstava
Razstava računalniške tehnologije, uporabe, literature in drugih računalniških naprav, z mednarodno udeležbo

Symposium and Seminars Informatica '85
Nova Gorica, September 24th-27th, 1985

Conference
18th Yugoslav International Conference on Computer Technology and Usage

Seminars
Selected Topics in Computer Technology and Usage
Nova Gorica, September 24th-27th, 1985

Exhibition
Exhibition of Computer Technology, Usage, Literature and Other Computer Equipment with International Participation
Nova Gorica, September 24th-27th, 1985

INFORMATICA 3/85

ODKRIVANJE NAPAK Z BERGEROVIM IN PODOBNIMI KODI II

RUDI MURN, SAŠA PREŠEREN, DUŠAN PEČEK, BORUT KASTELIC

UDK: 681.3, 325.6.08

INSTITUT JOŽEF STEFAN, LJUBLJANA
ODSEK ZA RAČUNALNIŠTVO IN INFORMATIKO

Članek opisuje modificiran Bergerov kod in konstrukcijo separabilnega koda, ekvivalentnega Bergerovemu. Analizirane so prednosti modificiranega Bergerovega koda v primeri z običajnim Bergerovim kodom. Podane so vrste napak, ki se pojavljajo v PLA vezjih ter opisano testiranje PLA vezij z modificiranim Bergerovim kodom.

ERROR DETECTION WITH BERGER CODE AND MODIFIED BERGER CODE II: This paper describes modified Berger code and construction of separable code, equivalent to Berger code. Analyzed are advantages of modified Berger code in comparison to Berger code. Listed are types of errors which occur in PLA circuits and a problem of testing PLA cicuits with modified Berger code is studied.

1. UVOD

V prvem članku o Bergerovih kodih (Informatica 4/84) smo opisali Bergerov kod in podali postopke za odkrivanje napak z Bergerovim kodom. Ker se v računalništvu vedno bolj uvaljavljajo PLA vezja z velikim številom izhodov, se pojavlja potreba po testiranju teh vezij. Napake, ki se pri teh vezjih pojavljajo so istoznačne. Prav izpeljanka Bergerovega koda imenovana Modificiran Bergerov kod pa se je izkazala uspešno pri odkrivanju teh napak. Zato bomo obdelali modificiran Bergerov kod ter poudarili prednosti, ki jih ima pred Bergerovim kodom. V zaključku bomo podrobneje predstavili vrste napak, ki se pojavljajo pri PLA vezjih in podali postopke testiranja teh vezij.

2. MODIFIKACIJE BERGEROVEGA KODA

Ogledali smo si že vse prednosti Bergerovega koda. Za primer, ko je teža istoznačne napake (teža napake je število istočasnih napak v posameznih bitih besede, težo napake ena imenujemo enojna napaka, težo napake dve imenujemo dvojna napaka itd.) manjša od števila informacijskih bitov v kodni besedi pa moramo poiskati modificirano rešitev. Prav tako je Bergerov kod v prvotni obliki nepraktičen za primer, ko Bergerov kod ni maksimalne dolžine.

Bergerov kod ne more detektirati istoznačnih napak teže, ki je manjša ali enaka m tako, da zmanjšamo število testnih bitov (m je celo število, ki je manjše od števila informacijskih bitov v kodni besedi). Zato bomo v poglavju 2.1. definirali modificirane Bergerove kode tako, da bodo ti kodi odkrili vse istoznačne napake teže, ki je manjša ali enaka m. Potem bomo ocenili dejansko sposobnost odkrivanja napak teh kodov in opisali TSC testno vezje za modificiran Bergerov kod.

Večina Bergerovih kod ni maksimalne dolžine. Zato bomo v poglavju 2.2. pokazali, da lahko kompletno separacijski kod C" ekvivalenten danemu Bergerovemu kodu nemaksimalne dolžine C1 izpeljemo iz Bergerovega koda maksimalne dolžine.

2.1. Modificiran Bergerov kod

Modificiran Bergerov kod je tisti kod, ki ima testne bite T1 zakodirane s testnimi biti T2. Naj bo

m = maksimalna teža istoznačne napake, ki jo odkrije modificiran Bergerov kod,

J = število bitov v testnem delu kodne besede T1 ali T2.

Dolžina kodne besede pri modificiranem Bergerovem kodu je torej

$$n = I + 2J.$$

Predpostavimo, da so vsi biti, pri katerih je prišlo do napake v delu D(I) v kodni besedi, to je med informacijskimi biti. Če uporabimo I0 mod (m+1) ali I1 mod (m+1) (1<m<I) kot testne simbole, ki jih označimo s T1, bomo lahko s tem kodom detektirali vse istoznačne napake, katerih teža je manjša ali enaka m, ker nobena taka napaka ne more spremeniti ene kodne besede v drugo. V tem primeru potrebujemo

$$J = \lceil \log_2 (m+1) \rceil$$

bitov za testni simbol T1.

Primer 1: število potrebnih testnih bitov J za odkrivanje napak različne teže:

I	Bergerov kod k	modificiran Bergerov kod J	m (maksimalna teža nap)
8	4	1	1 (enkratne napake)
		2	2 (dvojne napake)
		2	3 (trojne napake)
		3	4 (štirikratne nap.)
		3	5 (petkratne nap.)
		3	6 (šestkratne nap.)
		3	7 (sedemkratne nap.)
		4	8 (osemkratne nap.)

16	5	4 15 (petnajstkratno ali manjšo napako)
		5 16 (šestnajskratna nap)

Naj bo Pk (k=0,1,...) podmnožica kodnih besed pri katerih ima vsaka kodna beseda vrednost

$$I1 = k,$$

to je število enic v kodni besedi je enako indeksu k.

Stolpec T1 v Tabeli 1 kaže primer takega koda.

Tabela 1: Primer kodnih besed za I=8, m=7

podmnožica	kodna beseda	I0	T1=I0 mod 8	T2
P0	00000000	8	000	111
P1	00000001	7	111	000
P2	00000011	6	110	001
P3	00000111	5	101	010
P4	00001111	4	100	011
P5	00011111	3	011	100
P6	00111111	2	010	101
P7	01111111	1	001	110
P8	11111111	0	000	111

Problem nastopi z dejstvom, da lahko pride do napake v samih testnih bitih. Na primer napaka lahko spremeni kodno besedo P1 v kodno besedo P0 s spremembo samo 4 bitov (en informacijski in trije testni biti). Ker je število J običajno majhno

$$J = [\log_2 (m+1)],$$

je smiselno uporabiti drugonivojski kod, ki detektira napake v testnih bitih. Za zakodiranje testnega simbola T1 v testni simbol T2 lahko uporabimo katerokoli kod, ki odkriva istoznačne napake. Kod s testnimi simboli T1 in T2 imenujemo "modificiran Bergerov kod". Maksimalna teža napak, ki jih odkrije modificiran Bergerov kod označimo z m. Tabela 1 kaže primer modificiranega Bergerovega koda z m=7. Testni simboli T1 in T2 v tabeli 1 tvorijo dvotirni kod.

Ker v modificiranem Bergerovem kodu vse istoznačne napake v testnih bitih odkrije drugi kod, lahko za testni simbol T1 direktno uporabimo I0 mod (m+1) ali pa I1 mod (m+1). Jasno je, da modificiran Bergerov kod v tabeli 1 (s testnima simboloma T1 in T2) lahko odkrije vse istoznačne napake, ki imajo vrednost teže manjšo ali enako 7. Ta sposobnost odkrivanja napak je učinkovita neodvisno od števila informacijskih bitov v kodu.

Iz definicije modificiranega Bergerovega koda vidimo, da modificiran Bergerov kod odkrije vse istoznačne napake razen tistih, ki vplivajo le na informacijske bite in imajo vrednost teže enako mnogokratniku (m+1).

Poglejmo verjetnost za neodvisno napako. V tem primeru je napaka na izhodu neodvisna od statusa ostalih izhodov. Naj bo verjetnost, da je prišlo do napake na enem izhodu enaka p in, naj bo ta verjetnost enaka za vsak izhodni bit. Verjetnost, da ni prišlo do napake je q=1-p. Torej izrazimo verjetnost, da je prišlo do kakršnekoli istoznačne napake z Bernoullijevo porazdelitvijo in je enaka

$$\sum_{i=1}^{I1+J} \binom{I1+J}{i} p^{i} q^{n-i} + \sum_{i=1}^{I0+J} \binom{I0+J}{i} p^{i} q^{n-i} =$$

$$= (1-q^{I1+J})\, q^{I0+J} + (1-q^{I0+J})\, q^{I1+J} =$$

$$= q^{I0+J} + q^{I1+J} - 2q^{n} =$$

$$\doteq n\, p \qquad (p \ll 1).$$

Verjetnost, da je prišlo do neodkrite istoznačne napake je enaka

$$\binom{I1}{m+1} p^{m+1} q^{n-(m+1)} + \binom{I1}{2(m+1)} p^{2(m+1)} q^{n-2(m+1)} + \ldots$$

$$+ \binom{I0}{m+1} p^{m+1} q^{n-(m+1)} + \binom{I0}{2(m+1)} p^{2(m+1)} q^{n-2(m+1)} + \ldots =$$

$$\doteq \left[\binom{I1}{m+1} + \binom{I0}{m+1} \right] p^{m+1} \qquad (p \ll 1).$$

Torej je pogojna verjetnost P, da je prišlo do istoznačne napake, pa ta ni bila odkrita enaka kvocientu obeh prejšnjih izrazov

$$P = \left[\left(\binom{I1}{m+1} + \binom{I0}{m+1} \right) / n \right] p^{m}.$$

Številka, ki jo dobimo, ko izračunamo P, je pri različnih kodnih besedah različna, toda vedno je zelo majhna za razumne vrednosti p in za m>1. Ta verjetnost eksponentno pada, če večamo m. Vzrok za to je, da model neodvisnih napak predvideva manjšo verjetnost za pojav večkratne napake. Čeprav za nekatere primere, ko na primer kombinacijsko vezje, model neodvisnih napak ni najboljši, je v splošnem res, da je manjša verjetnost, da bo prišlo do napake v več bitih [DON82].

Poglejmo še drugačen model napak. Sedaj naj bo verjetnost za pojav kakršnekoli istoznačne napake, ne glede na to koliko bitov zajema, enaka. Naj bo tudi verjetnost, za vse kodne besede, da se pojavijo na izhodu, enaka. Število vzorcev napak v posamezni kodni besedi je enako

$$\sum_{i=1}^{I1+J} \binom{I1+J}{i} + \sum_{i=1}^{I0+J} \binom{I0+J}{i} =$$

$$= (2^{I1+J} - 1) + (2^{I0+J} - 1).$$

Število kodnih besed za I1 in I0 je

$$\binom{I}{I1} = \binom{I}{I0}$$

Celotno število vzorcev napak je torej

$$E = \sum_{I1=0}^{I} \left[(2^{I1+J} - 1) + (2^{I0+J} - 1) \right] \binom{I}{I1} =$$

$$= 2 \sum_{I1=0}^{I} (2^{I1+J} - 1) \binom{I}{I1}.$$

število neodkritih napak za vsako kodno besedo je

$$\sum_{0<j(m+1)\le I1} \binom{I1}{j(m+1)} + \sum_{0<j(m+1)\le I0} \binom{I0}{j(m+1)} .$$

Celotno število neodkritih napak je enako D

$$D = \sum_{I1=0}^{I} \left[\sum_{0<j(m+1)\le I1} \binom{I1}{j(m+1)} + \sum_{0<j(m+1)\le I0} \binom{I0}{j(m+1)} \right] \binom{I}{I1} =$$

$$= 2 \sum_{I1=0}^{I} \sum_{0<j(m+1)\le I1} \binom{I1}{j(m+1)} \binom{I}{I1} .$$

Sposobnost odkrivanja napak z modificiranim Bergerovim kodom za vse istoznačne napake je enaka

$$1 - \frac{D}{E}$$

Tabela 2 kaže odstotek odkritih napak za nekatere modificirane Bergerove kode, kjer T1 in T2 tvorita dvotirni kod

Tabela 2: Primerjava odkrivanja napak

I	modificiran Bergerov kod m+1	2J	odkril napake (v procentih)	Bergerov kod k	odkril nap. (procent)
16	4	4	93,74	5	100
32	4	4	93,75	6	100
48	4	4	93,75	6	100
64	4	4	93,75	7	100
16	8	6	99,04	5	100
32	8	6	98,54	6	100
48	8	6	98,33	6	100
64	8	6	98,47	7	100

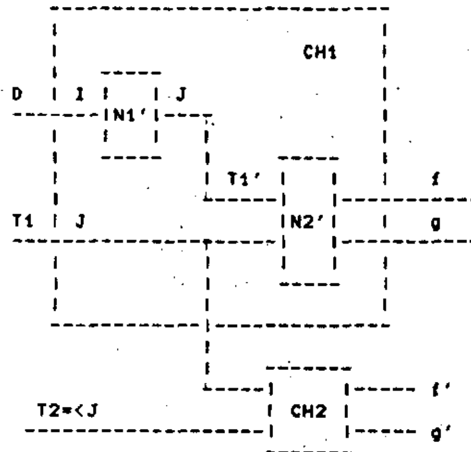
Vidimo, da ko število informacijskih bitov raste, ostane procent odkritih napak na približno isti vrednosti (93 oz. 98 odstotkov). To predstavlja veliko vrednost pri tistih aplikacijah, kjer ima vezje veliko število izhodov. Za praktično uporabno vezje bo vrsta oz. vzorec napake odvisen od funkcije in strukture tega vezja. V splošnem mora modificiran Bergerov kod odkriti večino istoznačnih napak, ki so v danem vezju možne.

Struktura testnega vezja za modificiran Bergerov kod

TSC testno vezje za modificiran Bergerov kod sestoji iz dveh glavnih delov (slika 1).
Vezje CH1 testira informacijske bite in sicer tako, da z vezjem N1' tvori komplement testnih bitov T1' in to primerja s T1 (v vezju N2'). Naj bo testni simbol T1 definiran kot

$$T1 = (2^{J} - 1) - (I1 \bmod m+1).$$

Z drugimi besedami je T1 komplement izraza I1 mod (m+1). V tem primeru vezje N1' imenujemo generator modula teže medtem, ko običajne generatorje teže imenujemo generator polne teže.



Slika 1.: Struktura testnega vezja za modificiran Bergerov kod.

Vezje N2' na sliki 1 je testno vezje za dvotiren kod. J izhodov iz vezja N1', ki jih označimo s T1', primerjamo v testnem vezju N2' s testnim simbolom T1, ki je del kodne besede. Ker modificiran Bergerov kod zagotavlja popoln kodni prostor za testno vezje N2' za dvotirni kod, je testno vezje CH1 TSC testno vezje. Drugi nivo kodiranja T1 in T2 testiramo v vezju CH2. T1 in T2 lahko tvorita bodisi dvotirni kod ali drug Bergerov kod. Če T1 in T2 tvorita dvotirni kod, potem je CH2 enak kot N2'. Ko ne pride do napak, potem sta T1 in T2 dopustni kodni besedi, ter prav tako T1 in T1'. Velja T1'=T2 in f=f', g=g'.

Tabela 3 daje primerjavo hardwareskih stroškov za generator testnih simbolov za modificiran Bergerov kod (m+1=4) in Bergerov kod. Ta prihranek v hardware-u je ocenjen v obliki števila elementov za PLA implementacijo [DON82].

št. bitov informacije	Berg. kod FA	HA	modif. Ber. k. FA	HA	XOR	prihranek
15	11	0	7	0	3	22,7%
16	11	4	7	2	3	25,6%
31	26	0	15	0	7	28,8%
32	26	5	15	2	7	30,7%
63	57	0	31	0	15	32,5%
64	57	6	31	2	15	33,6%

FA = popolni seštevalnik
HA = polovični seštevalnik
XOR = XOR vrata s tremi vhodi

Tabela 3.: Primerjava hardwareskih stroškov za generator testnih simbolov.

2.2. Konstrukcija separabilnega koda ekvivalentnega Bergerovemu kodu

Definicija: Separabilen kod C je ekvivalenten Bergerovemu kodu C1, če imata C in C1 enako dolžino, enako število informacijskih in testnih bitov in, če C detektira vse istoznačne napake.

Postopek 1 podaja konstrukcijo separabilnega koda C", ki je ekvivalenten danemu Bergerovemu kodu nemaksimalne dolžine.

Postopek 1: Naj bo C1 Bergerov kod nemaksimalne dolžine z dolžino n1, i1 informacijskimi biti in $[\log_2(i1+1)]$ testnimi biti.

Naj bo C Bergerov kod maksimalne dolžine s

$k=[\log_2(i1+1)]$ testnimi biti,

$I=(2\exp[\log_2(i1+1)])-1$ informacijskimi biti

dolžino n.

Naj bo C' kod, ki ga definiramo na sledeč način

C'=(X: X∈C in
(I-i1+1) = (n-n1+1) levih pozicij v X so vsi 0 ali pa vsi 1)

Iz C' dobimo kod C", ki je ekvivalentna danemu Bergerovemu kodu nemaksimalne dolžine C1 na sledeč način

C" = (Y: Y dobimo tako, da spustimo skrajnih levih (i-i1)=(n-n1) pozicij kodne besede C').

Primer:
Naj bo Bergerov kod nemaksimalne dolžine

C1 = (0010, 0101, 1001, 1100)

Potem je Bergerov kod maksimalne dolžine za k=2

C =(00011,00110,01010,10010,01101,
10101,11001,11100)

Z definicijo za C' izberemo samo ustrezne kodne besede iz C:

C'=(00011,00110,11001,11100) in

Za kod C" dobimo:

C"=(0011,0110,1001,1100)

Jasno je, da je C" popolno separabilen kod ekvivalenten C1 pri tem, da C1 ni popolno separabilen kod.

Za vsak Bergerov kod nemaksimalne dolžine C1 obstaja ekvivalenten separabilen kod C" za katerega je testno vezje tipa 1 TSC testno vezje. Ker večina Bergerovih kod ni maksimalne dolžine lahko s tem postopkom izpeljemo ekvivalenten separabilen kod.

STRUKTURA PLA VEZIJ

PLA (Programmable logic Arrays) se pogosto uporabljajo kot LSI/VLSI logične enote namesto ROM-ov ali običajne TTL logike. Testiranje PLA vezja postaja z večjo kompleksnostjo LSI/VLSI vezij vse zahtevnejše. Več avtorjev je obravnavalo probleme generiranja testov za PLA vezje. Največ pozornosti je bilo posvečeno eksplicitnemu testiranju, kjer določeni vhodni vzorci služijo kot testi, ki se izvajajo v različnem času kot uporaba PLA vezij. Študij pa je bil posvečen tudi implicitnemu (sočasnemu) testiranju PLA vezij, kjer se vezje testira med običajnim delovanjem PLA vezja.

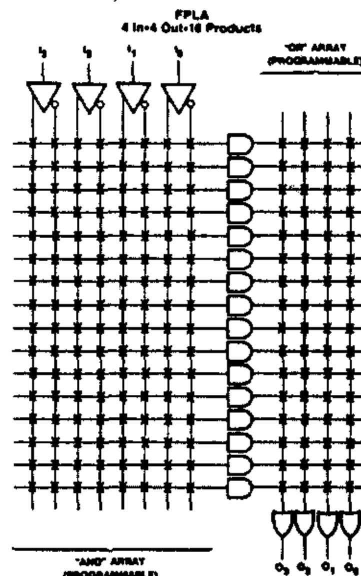
ROM, PAL in PLA vezja so si v sorodu. Poglejmo kaj jim je skupno, ter bistvene značilnosti, ki jih ločijo.

Osnovna logična struktura PROM-ov sestoji iz fiksnega polja AND vrat, katerih izhodi vodijo v programirno polje OR-vrat.

Pri uporabi PROM-ov je vhod v PROM adresa pomnilnika in izhod je vsebina te pomnilniške lokacije.

Osnovna struktura PAL sestoji iz programirnega polja AND vrat, katerih izhodi vodijo v fiksno polje OR vrat.

Osnovna struktura PLA sestoji iz programirnega polja AND vrat, katerih izhodi vodijo v programirno polje OR vrat (slika 2).



Slika 2.: Logična struktura PLA vezij.

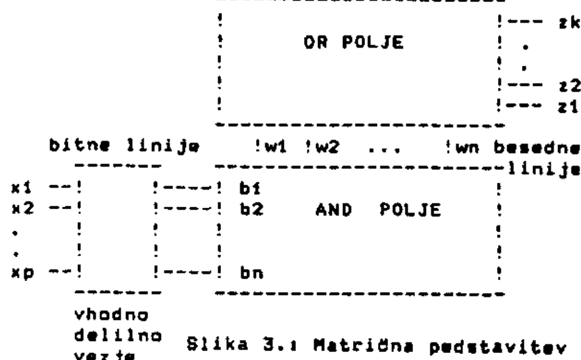
Ker imamo popoln nadzor nad vsemi vhodi in izhodi, omogoča PLA vezje popolno fleksibilnost pri implementaciji logičnih funkcij. Ta prožnost povzroča, da so PLA relativno dragi in funkcije težje razumljive, programiranje pa je drago (zahtevajo specialne programatorje).

3. MATRICNA PREDSTAVITEV PLA VEZIJ

PLA vezje ponavadi sestoji iz treh delov:

- vhodnega delilnega vezja,
- AND polja in
- OR polja.

Vhodne linije v AND polje imenujemo bitne linije, izhodne linije iz AND polja imenujemo besedne linije ali produktne linije (sl. 3)

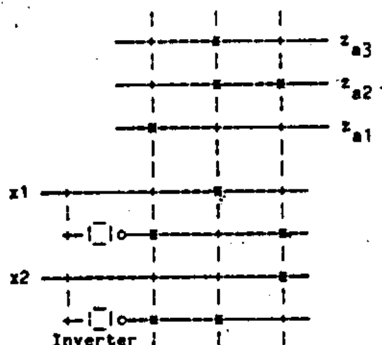


Slika 3.: Matrična pedstavitev

Vhodno delilno vezje na sliki 4 razdeli vhodne signale v dve skupini:

- prvotni signali in
- invertirani signali.

Vsak "x" na sliki 4. imenujemo križišče in pomeni,da sta dve liniji v stiku.



Slika 4.: AND-OR implamantacija PLA vezja.

4. VRSTE NAPAK V PLA VEZJIH

V tem razdelku bomo študirali zvezo med različnimi tipi napak v PLA vezju in vzorcu napak na izhodih, ki je posledica teh napak.

Obravnavali bomo tri tipe napak v PLA vezjih:

- stuck-at napake,
- napake na križiščih in
- kratkostične napake.

Pri NOR-NOR PLA vezju smatramo, da se stuck-at napake na besednih linijah in izhodnih linijah pojavijo samo na izhodih invertorjev. Napaka na križiščih je lahko posledica manjkajočega elementa ali pa dodatnega elementa v polju vezja katero spremeni uporabno križišče v neuporabno ali obratno. Kratkostična napaka med dvema sosednjima linijama povzroči, da sta vrednosti obeh linij enaki in lahko prevlada bodisi logična vrednost 1 ali logična vrednost 0, če sta prvotni vrednosti obeh linij različni. Poglejmo najprej nekaj osnovnih definicij:

Da se dokazati (DON82), da vsaka istoznačna napaka v vhodnem vektorju AND polja ali OR polja v PLA vezju lahko rezultira le v istoznačnih napakah v izhodnem vektorju tega polja in, da so vse enojne napake v PLA vezju razen vhodne napake tipa stuck-at, ki lahko povzročijo le istoznačne napake na izhodu iz PLA vezij.

Napake tipa stuck-at na vhodu bo vezje obravnavalo kot drugačne vhodne kombinacije. Edini način za sprotno testiranje vhodnih napak tipa stuck-at, je ta, da imamo vhodno informacijo zakodirano ter uporabljamo testno vezje (checker), ki odkrije kakršno koli napako v vhodni kodi.
V nadaljnem si poglejmo večkratne napake. Pri napakah tipa stuck-at v konvencionalnem kombinacijskem vezju, je poddel teh napak istoznačnih, saj so vse posamezne linije z večkratnimi napakami na isti vrednosti. Pokazano je bilo, da lahko v vezju brez invertorjev vsaka istoznačna napaka povzroči le istoznačne napake (SMI77). Ker imajo PLA vezja zelo regularno strukturo, lahhko razdelimo vse enojne napake, ki smo jih zgoraj obravnavali, v dve skupini napak: F1 in F0. Ti skupini sta definirani v Tabeli 4.

skupina napak F1

- vse napake tipa stuck-at-1
- dodatni členi na križiščih
- kratki stiki, ko prevlada "1"

skupina napak F0

- vse napake tipa stuck-at-0
- manjkajoči členi na križiščih
- kratki stiki, ko prevlada "0"

Tabela 4.: Skupina napak F1 in F0 za PLA vezje tipa AND-OR.

5. POSTOPKI TESTIRANJA

Ker vse enojne napake razen napak tipa stuck-at na vhodu povzročijo v PLA vezju le istoznačne napake na izhodih, lahko katerikoli kod, ki odkrije istoznačne napake uporabimo za sprotno testiranje PLA vezij. Taki kodi so:

- kod m-od-n,
- dvotirni kod,
- Bergerov kod.

Včasih ima PLA vezje lahko veliko število izhodnih linij. V takih primerih je predraga uporaba zgoraj naštetih kodov. Modificiran Bergerov kod (DON82), (DP3189), (MUR84) pa lahko odkrije vse istoznačne napake teže, ki ni enaka mnogokratniku v naprej določenega celega števila M.

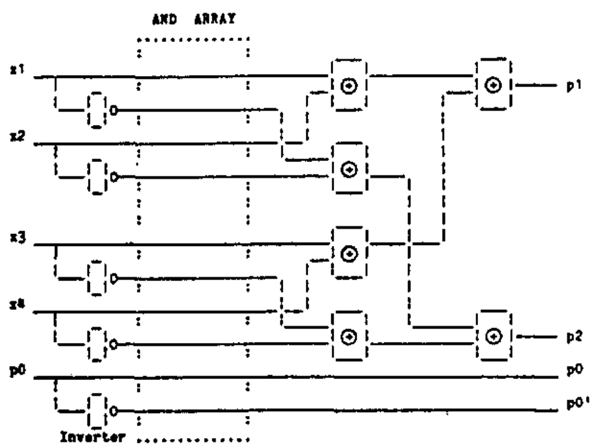
Drugi postopek, ki je bil obravnavan za testiranje PLA vezij, je metoda s paralelno vezanimi signaturnimi analizatorji (HAS83).

5.1. Sprotno testiranje PLA vezij z modificiranim Bergerovim kodom

V tem poglavju si bomo ogledali shemo PLA vezja za sprotno testiranje. S sprotnim testiranjem PLA vezij so se ukvarjali (DON82), (KHA82) in drugi. Iz prejšnjih diskusij vemo, da napake tipa stuck-at na vhodu v PLA vezje lahko sprotno testiramo, če vhodni podatek zakodiramo. Tukaj privzemimo, da vhode v PLA vezje zakodiramo z bitom za kontrolo parnosti (parity check bit). Na ta način bomo vse enojne napake tipa stuck-at na vhodu, kot tudi enojne napake v vhodnih podatkih odkrili z vgrajenim vezjem za testiranje parnosti v samen PLA vezju. Če uporabimo drugo XOR drevo, ki je prikjučeno na vhodne invertorje (kot kaže Slika 5), potem bodo vse enojne napake tipa stuck-at ali kratki stiki na bitnih linijah (vključno z napakami v invertorjih) testirane s tem dupliciranim vezjem za testiranje parnosti.
Modificiran Bergerov kod je primeren za testiranje velikih PLA vezij, ker je število testnih bitov (check bits) v modificiranem Bergerovem kodu neodvisno od celotnega števila informacijskih bitov in zavisi le od števila M. Za dano število M, je potrebno največ 2(log M) testnih bitov (logaritem z osnovo 2). Pri tem je (L) najmanjše celo število, ki je večje ali enako L. Na primer, za M=4, so potrebni štirje testni biti za konstrukcijo modificiranega Bergerovega koda, pri čemer bo v kodni besedi s 64 informacijskimi biti odkrito 93,75 odstotkov vseh možnih istoznačnih napak.

Modificiran Bergerov kod je separabilen kod. Kodna beseda za modificiran Bergerov kod sestoji iz dveh delov:

- podatkov D (informacijski biti) ter
- testnih bitov C.



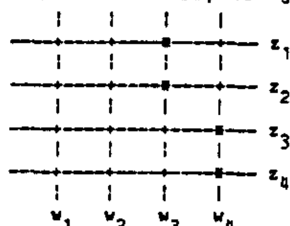
Slika 5.: Testiranje parnosti za vhodne napake.

Testni biti C so definirani na nasleden način:

Naj bo I1 število enic in I0 število ničel v informacijskem delu D kodne besede. Definiramo C(I1) = I1 modul M in C(I0) = I0 modul M. Potem dobimo testne bite C za kodno besedo modificiranega Bergerovega koda tako, da vključimo C(I1) ali C(I0) v kodno besedo, ki odkrije istoznačne napake.

V našem načrtu za PLA vezje ni težko ugotoviti koliko izhodnih bitov bo napačnih, ko pride do enojne napake. Ker napake na bitnih linijah testiramo s testnim vezjem za kontrolo parnosti (parity checker) in ker vsaka enojna napaka v OR polju vpliva največ na en izhodni bit, so vse kar moramo upoštevati, napake na produktnih linijah. Ko napaka spremeni vrednost produktne linije, so lahko samo tisti izhodi, ki so odvisni od te produktne linije, vključeni v napako. Naj bo t(i) število uporabljenih križišč produkta w(i) v polju OR. V najlažjem primeru izberemo celo število m tako, da je m > maksimuma za vse i od t(i). Potem bo vsaka napaka, ki je posledica enojne napake v AND polju ali v OR polju imela težo manjšo kot m in bo odkrita z modificiranim Bergerovim kodom. Ponavadi izberemo m = 2 exp N, kjer je N celo število. Na ta način bo enostavna implementacija testnega vezja za modificiran Bergerov kod.

V primeru, da ima en produkt (kot na primer kontrolna linija) t(i) dosti večji kot ostali, lahko uporabimo dve ali več istih besednih linij za ta produkt. Izhodi, ki uporabljajo ta produkt, so razdeljeni v skupine in vsaka grupa je povezana samo na eno od teh besednih linij. To kaže slika 6, kjer besedni liniji w(3) in w(4) tvorita isti produkt. Povdariti je treba, da bo vse napake, ki imajo težo različno od mnogokratnika M odkril Bergerov kod. Ni potrebno, da ima vsaka besedna linija manj kot M uporabljenih križišč v OR polju. V vsakem primeru, zaradi izbire malega M za modificiran Bergerov kod, lahko vedno uporabimo simulacijo napak (v našem primeru je treba obravnavati samo napake na besednih linijah), da preverimo, da nobena možna napaka ne bo imela teže, ki je deljiva z M.

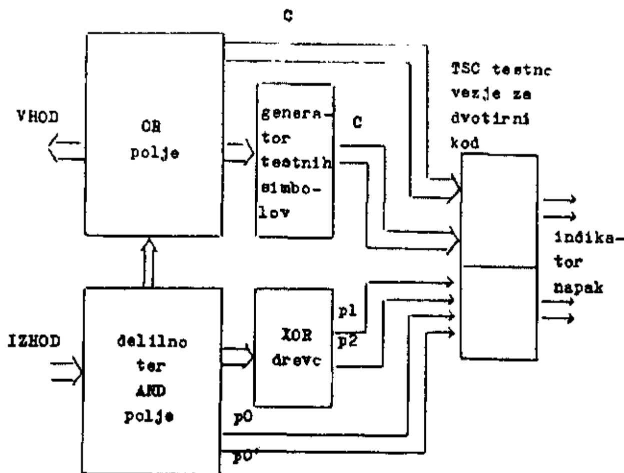


Slika 6.: Uporaba dveh besednih linij za en produkt.

Bločna shema PLA vezja je prikazana na Sliki 7. Ta sestoji iz dveh glavnih delov:

- PLA vezja z vkodiranimi izhodi in
- popolnoma samotestno vezje za kod.

Testni simbol C je definiran z modificiranim Bergerovim kodom in je implementiran v PLA kot običajni izhodi. Informacijski del D v izhodih PLA vezij je ponovno zakodiran v generatorju testnih simbolov, kot je to opisal (DON82).



Slika 7.: Načrt PLA vezja za sprotno testiranje.

Izhodi iz generatorja testnih simbolov, ki jih označimo s C*, potem primerjamo s testnim simbolom C v popolnoma samotestnem dvotirnem testnem vezju (AND71). Vsako neujemanje teh dveh signalov pomeni, da obstaja napaka bodisi v PLA vezju ali pa v testnem vezju. Dokazano je bilo, da je ta vrsta testnih vezij popolnoma samotestna, če in samo če se vsaka od 2 exp(k) binarnih k-teric pojavi v testnem simbolu neke kodne besede, kjer je k število testnih bitov. Take kode imenujemo kompletno separabilni kodi (OP3018). V primeru modificiranih Bergerovih kodov, kjer je testni simbol modul teže informacijskega dela D, je zelo verjetno, da ustrežemo tem pogojem za popolno samotestiranje. Pri vsakem dogodku, je treba preveriti, če je kod kompletni kod. Če ni, potem morajo biti nekateri nespecificirani izhodi uporabljeni za kompletiranje koda.

6. ZAKLJUČEK

Analizirali smo modificiran Bergerov kod, ki odkrije vse istoznačne napake katerih teža ni enaka mnogokratniku v naprej določenega celega števila m+1. Prednost modificiranega Bergerovega koda pred Bergerovim kodom je manjše število testnih bitov od običajnih Bergerovih kod, ter s tem cenejše oz. manjše testno vezje. Rezultat tega je sicer zmanjšana sposobnost detekcije napak kot z Bergerovim kodom, če vedno pa ti kodi detektirajo večino istoznačnih napak, ki se lahko pojavijo v danem vezju. Ker je število testnih bitov v modificiranem Bergerovem kodu neodvisno od celotnega števila informacijskih bitov je uporaba modificiranega Bergerovega koda primerna za vezja z velikim številom izhodov (kot na primer PLA vezja). Procent odkritih napak je približno konstanten (93 odstotkov oz. 98 odstotkov), ko število informacijskih bitov raste (32,48,64). Hkrati je implementacija TSC testnih vezij za modificiran Bergerov kod približno 20 - 30 odstotkov cenejša od vezij za Bergerov kod.

Ker vse enkratne napake razen napak tipa stuck-at na vhodu v PLA povzročijo le istoznačne napake na izhodu iz PLA, lahko katerikoli kod, ki odkrije istoznačne napake (kot na primer m-od-n kod, dvotirni kod ali Bergerov kod) uporabimo za sprotno testiranje PLA. Ponavadi pa imajo PLA veliko število izhodov, zato je v takih primerih uporaba teh kodov predraga. Modificirani Bergerov kod lahko odkrije vse istoznačne napake, katerih teža ni enaka mnogokratniku danega celega števila m. Modificirani Bergerovi kodi so primerni za testiranje velikih PLA, ker je število testnih bitov v modificiranem Bergerovem kodu neodvisno od celotnega števila informacijskih bitov in zavisi le od števila m. Za dano število m, potrebujemo največ $2\lceil \log m \rceil$ testnih bitov, kjer je $\lceil L \rceil$ najmanjše celo število, ki je večje ali enako L. Na primer, za m=4 potrebujemo 4 testne bite za tvorbo modificiranega Bergerovega koda in 93,75 procentov vseh možnih istoznačnih napak bomo odkrili v kodni besedi s 64 informacijskimi biti.

Odvisno od zgradbe PLA bomo ugotovili koliko izhodnih signalov bo napačnih, če je prišlo do neke napake v PLA. S tem določimo število m in modificiran Bergerov kod bo napako odkril. Ponavadi izberemo m=2 exp(N), kjer je N celo število, s čemer dosežemo enostavnejšo implementacijo TSC.

Predlagan je načrt PLA vezja za sprotno testiranje. Ta načrt lahko odkrije vsako enojno napako, vključno z napakami na vhodu ter določene večkratne napake v PLA vezju med normalnim delovanjem PLA vezja. Ta načrt ne potrebuje spremembe konfiguracije obstoječega PLA vezja. Namesto tega dodamo vezje za zakodiranje in testiranje. Modificirani Bergerovi kodi in vezje za testiranje parnosti so uporabljeni v tem načrtu. S tem je ta pristop manj drag, kot pa je uporaba drugih kodov kot na primer dvotirnih kodov ali Bergerovih kodov. Načrt testnega vezja za modificiran Bergerov kod je opisan v (DON82). Ta načrt je še posebej atraktiven v primerih ko:

- ima PLA veliko število izhodov in
- vsako besedno linijo si deli relativno majhen del izhodov.

7. LITERATURA

(ASH76) M.J.Ashjaee in S.M.Reddy: On Totally-Self-Checking Checkers for Separable Codes, Int. Symp. on Fault-Tolerant Computing, pp.151-156, 1976.

(BOS82A) B. Bose in D. K. Pradhan: Optimal Unidirection Error Detecting/ Correcting Codes, IEEE Transactions on computers, Vol. c-31, No. 6, June 1982.

(BOS82B) Bella Bose in Thammavaram R. N. Rao: Theory of Unidirectional Error Correcting/Detecting Codes, IEEE Trans. on Computers, Vol. C-31, No. 6, June 1982.

(CAR80) W.C.Carter in A.B. Wadia: Design and Analysis of Codes and their Self-Checking Circuit Implementation for Correction and Detection of Multiple b-Adjacent Errors, IEEE, pp. 35-40, 1980.

(DON82) H.Dong: Modified Berger Codes for Detection of Unidirectional Errors, IEEE, pp 317-320, 1982.

(DON82) Hao Dong in Edward J. McCluskey: Concurrent Testing of Programmable Logic Arrays, CRC Technical Report No. 82-11, Stanford University, Center for Reliable Computing, June 1982.

(DP3189) Murn, Peček, Kastelic, Prešern: Odkrivanje istoznačnih napak z Bergerovimi in podobnimi kodi, Institut Jožef Stefan, december 1983, DP - 3189.

(HAS83) Syed Zahoor Hassan in Edward J. McCluskey: Testing PLAs Using Multiple Parallel Signature Analysers, FTCS 13th Annual Intern. Symp. Fault Tolerant Computing, Milano 1983, pp.422-425. 9, September 1979.

(KHA82) Jadav Khakbaz in Edward J. McCluskey: Concurrent Error Detection and Testing for Large PLA's, IEEE Transactions on Electron Devices, Vol. ED-29, No. 4, April 1982.

(KHA83) Javad Khakbaz: A Testable PLA Design with Low Overhead and High Fault Coverage, FTCS 13th Annual Intern. Symp. Fault Tolerant Computing, Milano 1983, pp.426-429.

(MAR78) M.A. Marouf in A.D. Friedman: Design of Self-Checking Checkers for Berger Codes, The Eight Annula Int. Conf. on Fault-Tolerant Computing, pp.179-184, June 1978.

(MUR84) R. Murn, S. Prešern, D. Peček in B. Kastelic: Odkrivanje napak z Bergerovim in podobnimi kodi I, Informatica 4/84, letnik 8, pp.68 - 74.

(OST79) Daniel L. Ostapko: Fault Analysis and Test Generation for Pragrammable Logic Arrays (PLA's), IEEE Trans. on Computers, Vol. c-28, No.9, september 1979.

(PRA80) D.K. Pradhan: A New Class of Error-Correcting/Detecting Codes for Fault-Tolerant Computer Applications, IEEE Trans. on Comp., vol. C-29, pp. 471-481, June 1980.

(SMI77) J.E. Smith in drugi: The Design of Totally Self-Checking Combinational Circuit, Proc. 7th Ann. Symp. on Fault Tolerant Computing, Los Angeles, June 1977, pp.130-134.

(WAK78) J. Wakerly: Error Detecting Codes, Self-Checking Circuits and Applications, Elsevier North-Holland,Inc. The Computer Science Library, 1978.

INFORMATICA 3/85

INTEGRISANO OKRUŽENJE PROGRAMSKOG JEZIKA – ALAT ZA UDOBNO I EFIKASNO PROGRAMIRANJE

ZLATIBOR VUKAJLOVIĆ

UDK: 519.682.8

ELEKTROTEHNIČKI FAKULTET SARAJEVO

KRATAK SADRŽAJ: U ovom radu su prikazane konceptualne osnove tzv. integisanog okruženja programskog jezika, programskog sredstava koje omogućuje udobno i efikasno programiranje, omogućujući simultano unošenje, izmjenu, testiranje i prevođenje programa. Opisan je način njegovog korištenja, osnovni dijelovi i njihova međusobna funkcionalna povezanost.

INTEGRATED PROGRAMMING LANGUAGE ENVIRONMENT - TOOL FOR CONFORTABLE AND EFFICIENT PROGRAMMING: In this paper are shown conceptual basses for integrated programming language environment, programming tool which provide comfortable and efficient programming, providing simultan input, testing, editing and compiling program. It is described way of its using, its parts and their mutual linking.

1. UVOD

U procesu nastajanja izvršive verzije programa postoji nekoliko koraka. Prvo je potrebno formirati tekst izvornog programa (npr. pomoću standardnog editora teksta), a potom ga prevesti. Rezultat prevođenja, iz praktičnih razloga, nije uvijek i izvršiva verzija programa, pa je stoga, nakon prevođenja, potrebno program i povezati. Povezivati se mogu dijelovi programa koji su napisani u različitim programskim jezicima i/ili neovisno prevedeni.

Na taj način formiran program ne mora biti i korektan. Korektnost se provjerava testiranjem programa. Za testiranje su projektovana određena programska sredstva, koja olakšavaju testiranje, tzv. simbolički kontrolni testni-programi ("debugger"). Eventualne greške se moraju popraviti (pomoću editora), program ponovo prevesti, povezati, testirati .... .

U jednom ovakvom, tradicionalnom, pristupu se (znatno) gubi vrijeme na višestruko prevođenje korektnih dijelova programa, na procese povezivanja ili pak višestruko unošenje nekih sekvenci koje se ponavljaju u različitim programima (a što se može jednostavnije postići njihovim inkorporiranjem u vlastiti program).

Stoga se postavlja pitanje opravdanosti projektovanja tzv. integrisanog okruženja programskog jezika, koje bi omogućilo da se, relativno jednostavnim korištenjem, proces nastajanja korektne izvršive verzije programa posmatra kao jedinstven proces. Okruženje bi trebalo da olakša pisanje programa, na taj način, što bi: programera "vodio" u procesu prevođenja, "trenutno" reagovao na leksičke, sintaksne i semantičke greške i što bi omogućavao simultano unošenje, izmjenu i testiranje (jezički orijentisanim simboličkim kontrolno-testnim programom) korisničkih programa.

Konceptualni osnovi jednog takvog integrisanog okruženja programskog jezika su dati u ovom radu. U njemu su opisane funkcije okruženja, njegovi osnovni dijelovi, uloga i međusobna povezanost istih. Pored toga prikazano je i konceptualno rješenje realizacije i načina korištenja integrisanog okruženja.

2. FUNKCIJE OKRUŽENJA I NAČIN KORIŠTENJA

Kao što smo već napomenuli, integrisano okruženje programskog jezika treba da integriše procese unošenja, izmjene, testiranja i prevođenja programa. Pored toga, okruženje nudi i neke dodatne mogućnosti kao što su: listanje statičke strukture programa, listanje korištenih objekata (procedura, varijabli, konstanti ...) sa naznakom gdje se isti definišu i koriste, kao i frekvencije izvršenja pojedinih dijelova (iskaza) programa. Pri tome, korištenje okruženja treba da bude relativno jednostavno i efikasno, i da se ostvaruje interakcijom korisnika sa okruženjem.

Osnovu baze podataka okruženja čini prelazna forma u širem smislu, koja se sastoji od tri dijela: tabele simbola, tabele koda i tabele teksta. Date tabele se mogu nalaziti u centralnoj memoriji, mogu biti organizovane kao datoteke sa direktnim pristupom, a moguće je pronaći i neko kompromisno rješenje.

Komandama okruženja vrše se operacije nad prelaznom formom. Tipične komande su: L (list), I (include), C (change), O (object code), D (define), S (save), K (kill), P (path), H (help), M (move), E (edit), G (go), R (cross references) i Q (quit). Dati skup komandi je invarijantan u odnosu na programski jezik na kojeg se okruženje odnosi, a izborom odgovarajućih opcija i drugih parametara (imena datoteka) se postiže veoma visoka fleksibilnost. Pored ovih, koriste se i komande zadane preko funkcionalnih tipki (npr. , ,-, - i sl.) koje omogućuju prolaženje kroz tekst izvornog programa, pri čemu tzv. kursor pokazuje na trenutno referenciranu poziciju u tekstu.

Tekstualne komande imaju niže navedene opise i efekte (parametri navedeni unutar uglatih zagrada se mogu izostaviti):

L [parametri]
Data komanda u definisanu datoteku (ili standardan izlaz, ako ime datoteke nije navedeno) ispisuje tekst izvornog programa, a u

formatu definisanog opcijama komande (sa ili bez broja linije i sl.).

I [parametri]
Ovom komandom se u korisnički program (prelaznu formu) inkorporira sadržaj naznačene datoteke (datoteka) ili tekst koji se unosi sa standardnog ulaza. Način inkorporiranja, sadržaj datoteke (tekst izvornog programa ili prelazna forma u širem smislu), kao i reakcije na pojavu greške se definišu opcijama. Linije koje započinju sa znakom $, tretiraju se kao (indirektne) komande okruženja. (Na taj način se koriste tzv. komandne datoteke.)

K [parametri]
Datom komandom se iz prelazne forme brišu njeni dijelovi. Početna pozicija dijela koji se izbacuje je određena pozicijom kursora, a krajnja pozicija i druge relevantne informacije (npr. informacija da treba izbrisati okvirnu proceduru), kao i reakcije na greške, se zadaju opcijama komande.

P parametri
Data komanda postavlja kursor na novu poziciju, naznačenu tzv. putem kursora. Put kursora se definiše imenima procedura ili drugih objekata definisanih simboličkim imenima, odnosno brojevima iskaza unutar tijela procedure. Elementi puta se odvajaju znakovima "/". Ako se izostavi prvi element, početna pozicija je trenutna pozicija kursora, a u suprotnom, naznačeni vodeći element mora biti objekat definisan na globalnom nivou. Npr. /p1/p2/3 označava da se kursor postavlja na treći iskaz procedure p2, ugnježdene u proceduru p1, a koja je (opet) ugnježdena u proceduru na koju (deklaraciju ili neki iskaz unutar tijela) trenutno pokazuje kursor.

M [parametri]
Datom komandom se dijelovi programa, definisani pozicijom kursora i opcijama komande, prebacuju sa jednog na drugo mjesto. Odredište se izabira: parametrima (definisanjem puta kursora) ili pomjeranjem kursora pomoću funkcionalnih tipki, i pritiskom na odgovarajuću (funkcionalnu) tipku nakon toga.

C parametri
Ovom komandom se omogućuje izmjena dijela prelazne forme u širem smislu. Izmjena se vrši na nivou jednog iskaza, odnosno deklaracije. Opcijama se, između ostalog, opisuju reakcije na pojavu greške.

S parametri
Ovom komandom se omogućuje da se dio (ili kompletna) prelazne forme u širem smislu sačuva u eksternim datotekama (datoteci). Sve datoteke, koje odgovaraju tabelama koda, simbola i teksta, imaju isti prvi dio imena, a različite sufikse.

G
Datom komandom se aktivira modul interpretera sa jezički orijentisanim simboličkim kontrolno testnim programom, a koji je niže detaljnije opisan.

O parametri
Datom komandom se aktivira modul koji na osnovu prelazne forme generiše objektni (ili pak asemblerski) kod. Parametrima se definiše oblik objektnog koda i ime odgovarajuće datoteke

R [parametri]
Ovom komandom se, na jednokratno izvršenje, poziva modul koji omogućuje izlistavanje statičke strukture programa, popis svih korištenih objekata definisanih u programu, a sa naznakom gdje se isti definišu i koriste. Opcijama se definišu željeni podaci, a izostavljanje imena datoteke označava listanje na standardnom izlazu.

E parametri
Ovom komandom se aktivira editor teksta izvornog programa (skromnijih mogućnosti). Po izlasku iz datog editora kontrola se vraća na okruženje.

H [parametri]
Ovom komandom se na standardnom izlazu ili u naznačenu datoteku izlistava upustvo za upotrebu.

D makro definicija
Ovom komandom se definiše makro sa parametrima (kao u C-u). Svaki poziv definisanog makroa vrši odgovarajuće makroproširenje.

Q [ime datoteke]
Komanda za izlazak iz okruženja. Ime datoteke označava skup datoteka u kojim će se prelazna forma sačuvati. (Razlikuje se od komande S po tome, što se, ako se tabele: koda, simbola i teksta nalaze na disku, vrši samo njihovo reimenovanje.) Ako, u parametrima, nije zadano ime datoteke, potrebno je, zadavanjem nove komande Q, izvršiti verifikaciju iste.

Interpreter sa jezički orijentisanim kontrolno testnim programom, treba da omogući jednostavno, pregledno i efikasno interaktivno testiranje programa. Naime, on treba da omogući da se interpretiranje obavlja na jedan od načina: koračnom, iskaznom, tragovnom ili normalnom.

U koračnom načinu rada, izvršava se iskaz po iskaz, i to na taj način što se na standardnom izlazu ispisuje tekst koji odgovara iskazu i rezultati izračunavanja iskaza, nakon čega se, do zadavanja nove komande načina rada, proces prekidanja prekida.

Iskazni način rada se, od koračnog, razlikuje po tome što se poziv nestandardne procedure ili funkcije, izvršava u jednom koraku, a ne iz onoliko koraka, koliko ima u tijelu procedure definisanih iskaza.

U tragovnom načinu rada (na standardnom) izlazu se ispisuju dijelovi programa koji se interpretiraju, i to bez prekidanja procesa interpretacije i bez ispisa rezultata izračunavanja.

U normalnom načinu rada se vrši, bez prekida i bez ispisa teksta programa i rezultata izračunavanja pojedinih iskaza, interpretiranje prelazne forme.

Normalni, iskazni ili tragovni način rada se, zadavanjem nove komande, može prekinuti i definisati novi način rada. Naime, u procesu interpretiranja, kod svakog iskaza, na početku se vrši provjera da li je u međuvremenu zadana nova komanda.

Pored toga mogu se zadati i tzv. trag i prekidne tačke. To znači da se mogu obilježiti iskazi kod kojih se, prilikom njihovog interpretiranja, interpretiranje prekida i/ili ispisuje tekst koji odgovara datom iskazu. Posebnom komandom se vrši izlistavanje brojeva iskaza i odgovarajućih tekstova.

Posebna mogućnost interpretera je štampanje vrijednosti u simboličkom obliku, kao i izmjena vrijednosti promjenjivih definisanih u programu, te definisanje tzv. uslovnih prekidnih tačaka (npr. prekid ako je neka varijabla promijenila vrijednost, ili ako je varijabla poprimila negativnu vrijednost i sl.). Pored toga treba da postoji mogućnost

redirekcije standardnih ulaza i izlaza korištenih u programu koji se testira (zbog toga što se za interakciju korisnika sa okruženjem isti koriste).

Komandama interpretera se mogu obilježiti i oštampati iskazi čija se frekvencija izvršavanja želi pratiti, a takođe postoji i komanda za izlazak iz ovog modula.

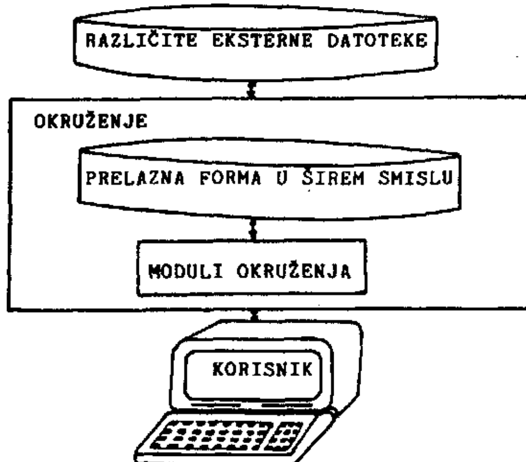
3. MODULI (DIJELOVI) OKRUŽENJA I NJIHOVA POVEZANOST

Iz predhodnog teksta se moglo zaključiti da se okruženje sastoji od nekoliko međusobno nezavisnih dijelova (modula). Već su spomenuti moduli interpretera i generatora objektnog koda, kao i modul koji omogućuje izlistavanje statičkih statističkih podataka o programu. Intuitivno su se mogli naslutiti moduli koji odgovaraju komandama: H, E, L, H, C, S i K. Komandom I se vrši definisane ulazne datoteke u inkrementalni prevodilac, odnosno leksički analizator, dok se komanda M sastoji od komandi L, K i I, a uz korištenje neke privremene datoteke.

Ostatak okruženja sadrži najkompleksniji i najveći modul inkrementalnog prevodioca, odnosno sintaksno-semantičkog editora. Ovaj modul podržava ostale komande, inkrementalno obavlja funkcije makro predprocesora, leksičku, sintaksnu i semantičku fazu prevođenja, kao i fazu generisanja prelazne forme (u širem smislu), modifikuje prelaznu formu, podržava funkcionalne tipke i poziva ostale module na jednokratno izvršenje.

Svi moduli manipulišu sa prelaznom formom kao bazom podataka. Pojedini elementi unutar tabela su međusobno spregnuti, tako da tvore neku strukturu: tabela koda drvo, tabela simbola drvo ili listu i tabela teksta niz listi. Takođe postoji povezanost elemenata iz različitih tabela: iz tabele koda se pointira na elemente tabele teksta i tabele simbola, a iz tabele simbola se vrši pointiranje na tabelu koda i tabelu teksta.

Izvan okruženja programskog jezika, ali opet u nekom svojevrsnom okruženju, se nalazi korisnik koji koristi različite eksterne datoteke, odnosno uređaje kao specijalan oblik datoteka. To je šematski prikazano na slici:



4. ZAKLJUČAK

Integrisano okruženje programskog jezika, kroz simultano unošenje, izmjenu, testiranje i prevođenje programa, treba da omogući udobno i efikasno programiranje, a izborom odgovarajuće prelazne forme (drveta) i generisanje optimalnog objektnog koda. Mogućnost praćenja frekvencija izvršenja pojedinih iskaza daje programeru dodatnu informaciju o tome na koje dijelove programa (prilikom pisanja) treba obratiti posebnu pažnju, a lista kros referenci daje potpuni uvid u definisanost i korištenje objekata.

Sama koncepcija okruženja omogućuje relativno lako "uvođenje" u korištenje istog. Za početak je dovoljno koristiti komande: I za prevođenje teksta izvornog programa u prelaznu formu, G za startanje interpretera sa jezički orijentisanim kontrolno testnim programom i O za generisanje objektnog koda, a nakon toga postepeno "usvajati" i ostale komande. Bitno je napomenuti da kod prelaska na okruženje nekog drugog jezika skup komandi ostaje isti (mijenjaju se eventualno parametri).

Nadalje, moduli okruženja su međusobno neovisni i dobro definisani, pa stoga okruženje može biti veoma lako realizovano od strane više programera, čiji su poslovi međusobno različiti. Takođe, cjelokupno okruženje se može realizovati u jeziku visokog nivoa, što okruženje čini veoma prenosivim. Glavni problem kod prenošenja je pisanje novog modula za generisanje objektnog koda, dok se ostali moduli nešto modifikuju.

Put ka realizaciji može biti i takav da se prvo realizuje neintegrisano okruženje, a takođe je moguća postupnost u realizaciji: prvo se realizuje okruženje za minimalan podskup jezika, koji se postepeno proširuje.

5. LITERATURA

Grupa autora: Software Engineering - An Advanced Course, Springer-Verlag, 1975

Medina-Mora: An Incremental Programming Environment, IEEE Transactions on Software Engineering, septembar 1981

INFORMATICA 3/85

INDEKSIRANJE MAGNETNIH TRAKA

SINIŠA J. DJORDJEVIĆ

UDK: 681.327.6

U radu je prikazana mogućnost organizovanja indeks sekvencijalnih datoteka na magnetnim trakama. Mogućnost proističe iz različitih brzina pristupa slogovima na magnetnoj traci u zavisnosti od organizacije podataka na traci. Ovakve organizacije podataka ne nude se korisnicima računarskih sistema zbog teškoća održavanja ali odredjeni aspekti primene opravdavaju njihovo organizovanje.

MAGNETIC TAPE INDEXING: This paper presents possibility of organization of the index sequential files on the magnetic tape. The possibility is given, depending of the data organization on the tape, by the different accessing speeds. This organizations are not given to the users.

I UVOD

INDEKSIRANJE U ŠIREM SMISLU PREDSTAVLJA ODREDJENI PROSTOR ISPUNJEN TAČKAMA (ENTITETIMA ODNOSNO ODGOVARAJUĆIM JEDINICAMA PODATAKA). KAKO SE SVAKI PROSTOR MOŽE PRESLIKATI NA JEDNODIMENZIONALNI PROSTOR TO SE MOŽE USVOJITI DA JE JEDNODIMENZIONALNI I INDEKSNI PROSTOR. RASTOJANJE DVEJU TAČAKA MOŽE SE DEFINISATI I KAO UREDJENI PAR ČIJI JE PRVI ELEMENT SAM PAR TAČAKA (ILI NJIHOVE KOORDINATE). AKO JE OVAJ UREDJENI PAR, RASTOJANJE DVEJU TAČAKA U PROSTORU, RELACIJA TADA IMA SMISLA PROSTOR ORGANIZOVATI KAO INDEKSNI. SUŠTINA JE U TOME DA SE IZBOROM RASTOJANJA DVEJU TAČAKA (IZBOROM KRETANJA) ODREDJUJE MINIMALNO RASTOJANJE DVEJU TAČAKA.

U OVAKO ODREDJENIM USLOVIMA ZA PREPOZNAVANJE INDEKSNIH PROSTORA MOŽE SE I MAGNETNIM TRAKAMA PRISTUPITI KAO INDEKSNIM PROSTORIMA JER JE MOGUĆE PREPOZNATI RASTOJANJE MEDJU PODACIMA (TAČKAMA PROSTORA) KAO RELACIJU.

INDEKSNE PROSTORE U ORGANIZACIJAMA PODATAKA PREDSTAVLJAJU PRE SVEGA INDEKS SEKVENCIJALNE DATOTEKE.

SOFTVERSKA PODRŠKA INDEKS SEKVENCIJALNIH DATOTEKA NIJE PREDVIDJENA OPERATIVNIM SISTEMOM KADA SU U PITANJU MAGNETNE TRAKE JER JE NA MAGNETNIM TRAKAMA MOGUĆ JEDINO SEKVENCIJALNI PRISTUP. U SUŠTINI, PROBLEM JE U NEPRIHVATLJIVIM VREMENIMA PRI PRISTUPU SLOGOVIMA NA OSNOVU INDEKSIRANJA KOJE PRETPOSTAVLJA DIREKTAN PRISTUP NA MEDIJUMU NA KOJEM SU SMEŠTENI PODACI. I AKO SE DIREKTNI PRISTUP UVEK MOŽE SIMULIRATI SEKVENCIJALNIM ZBOG ČEGA JE SVAKA ORGANIZACIJA PODATAKA NA JEDINICAMA SA DIREKTNIM PRISTUPOM MOGUĆA I NA JEDINICAMA SA SEKVENCIJALNIM PRISTUPOM VREME PRISTUPA JE ISKLJUČIVI, OGRANIČAVAJUĆI FAKTOR. MEDJUTIM, OPERATIVNI SISTEMI NE ISKLJUČUJU MOGUĆNOST OVAKVIH ORGANIZACIJA PODATAKA KOJE SU NA OSNOVU VREMENA PRISTUPA NEPRIHVATLJIVE ZA NAJČEŠĆE PRIMENE JER POSTOJE I TAKVI SLUČAJEVI GDE BAŠ OVAKVE ORGANIZACIJE POVEĆAVAJU EFIKASNOST. NA PRIMER, DOS VS NA NIVOU ASEMBLERSKIH MACRO-INSTRUKCIJA OMOGUĆAVA ORGANIZOVANJE INDEKS SEKVENCIJALNIH DATOTEKA NA MAGNETNIM TRAKAMA. MEDJUTIM, OVO DALJE ZNAČI VRLO OTEŽANO ILI NEMOGUĆE KORIŠĆENJE OVIH DATOTEKA IZ VIŠIH, PROBLEMSKI ORIJENTISANIH, PROGRAMSKIH JEZIKA. NAIME, OSTAJU TRI MOGUĆNOSTI ZA KORIŠĆENJE TIH DATOTEKA OD KOJIH SU PRVE DVE VEZANE ZA VIŠI PROGRAMSKI JEZIK DOK JE TREĆA KORIŠĆENJE DATOTEKA NA NIVOU PROGRAMIRANJA NA KOJEM SU TE DATOTEKE I DEFINISANE. PRVA MOGUĆNOST JE DVE DATOTEKE. DATOTEKE ZA KOJE NE POSTOJI SOFTVERSKA PODRŠKA U OPERATIVNOM SISTEMU NA SVIM NIVOIMA PROGRAMIRANJA, ODGOVARAJUĆIM PREDPROGRAMIMA A NA OSNOVU ZAHTEVA KONKRETNE OBRADE REORGANIZOVATI U DATOTEKE SA ODGOVARAJUĆOM PODRŠKOM I TAKVE, SKRAĆENE, DALJE IH PROSLEDITI. DRUGA MOGUĆNOST JE KORIŠĆENJE GOTOVIH RUTINA ZA PRISTUPE I OBRADE IZ NIVOA PROGRAMIRANJA NA KOJEM SU DATOTEKE DEFINISANE U VIŠIM NIVOIMA PROGRAMIRANJA ALI SE OVIME OTVARA PROBLEM POVEZIVANJA PROGRAMSKIH JEZIKA KOJI ZAHTEVA DODATNE NAPORE. I AKO TREĆA MOGUĆNOST TAKODJE SADRŽI NEDOSTATAK, MALO JE OBUČENIH PROGRAMERA ZA NIŽE NIVOE PROGRAMIRANJA A I SAMO PROGRAMIRANJE JE KOMPLIKOVANIJE, NJENOM PRIMENOM POSTIŽE SE NAJVEĆA EFIKASNOST U SLUČAJEVIMA KADA JE OPRAVDANO ORGANIZOVANJE INDEKS SEKVENCIJALNIH DATOTEKA NA MAGNETNIM TRAKAMA. U TEKSTU SE POREd DETALJNOG OPISA ORGANIZACIJE INDEKS SEKVENCIJALNIH DATOTEKA NA MAGNETNIM TRAKAMA RAZMATRA I EFIKASNOST PRIMENE U SLUČAJEVIMA ZA KOJE SE PREDLAŽE OVAKVO ORGANIZOVANJE PODATAKA. KAO ŠTO ĆE BITI POKAZANO, INDEKS SEKVENCIJALNE DATOTEKE NA MAGNETNIM TRAKAMA MOGU POVEĆATI EFIKASNOST KORIŠĆENJA ODREDJENIH ARHIVA, MOGU OMOGUĆITI AUTOMATIZOVANJE EVIDENCIJE I ISKORIŠĆENJA KAPACITETA MAGNETNIH TRAKA I OMOGUĆITI ARHIVIRANJE PROCEDURA SA NIŽOM FREKVENCIJOM UPOTREBE KOJE SE MOGU IZVODITI DIREKTNO SA TRAKE

KOMPLETNA DALJA ANALIZA IZVEDENA JE NA OSNOVU OPERATIVNOG SISTEMA DOS VS ALI TO NIJE NIKAKVO OGRANIČENJE JER JE ANALIZA U STVARI BAZIRANA NA HARDVERSKOM ODREDJENJU JEDINICA MAGNETNIH TRAKA PA JE ZA DRUGE OPERATIVNE SISTEME MOGUĆE PO ANALOGIJI IZDVOJITI ODGOVARAJUĆE ELEMENTE.

II ORGANIZACIJA INDEKS SEKVENCIJALNIH DATOTEKA NA MAGNETNIM TRAKAMA

NA MAGNETNIM TRAKAMA MOGUĆ JE SAMO "STROGO" SEKVENCIJALNI PRISTUP ALI SE NA OSNOVU HARDVERSKE ORGANIZACIJE TRAKA MOGU RAZLIKOVATI TRI

VRSTE OVOG PRISTUPA KOJE SE ZNATNO RAZLIKUJU U BRZINI. PRVA MOGUĆNOST JE PREMOTAVANJE TRAKE NEZAVISNO OD SADRŽAJA DELA KOJI SE PREMOTAVA. DRUGA MOGUĆNOST JE PREMOTAVANJE TRAKE SLOG (FIZIČKI SLOG) PO SLOG BEZ PRENOSA SADRŽAJA TOG SLOGA U OPERATIVNU MEMORIJU. TREĆA MOGUĆNOST JE ČITANJE SLOGOVA SA TRAKE ODNOSNO PREMOTAVANJE SLOG PO SLOG SA PRENOSOM SADRŽAJA SLOGA U OPERATIVNU MEMORIJU. PRVA MOGUĆNOST JE PREMOTAVANJE NA OSNOVU TEIP MARKI (TAPE MARK U DALJEM TEKSTU U OZNACI TM). TM JE FIZIČKI SLOG KOJEI ISPISUJE KANAL I KOJI I SLUŽI ZA AUTOMATSKO PREMOTAVANJE TRAKE ODNOSNO ZA ODVAJANJE FIZIČKIH DELOVA TRAKE KOJE PROSTAVLJAJU POSEBNE LOGIČKE ORGANIZACIJE PODATAKA. PRVA MOGUĆNOST, PRVI NAČIN PREMOTAVANJA, JE DALEKO BRŽA OD PREOSTALE DVE(ODNOS BRZINA DIREKTNO ZAVISI I OD BROJA SLOGOVA NA FIZIČKOM DELU TRAKE KOJI SE PREMOTAVA) JER NEMA ZAUSTAVLJANJA I AKTIVIRANJA KRETANJA TRAKE NA SVAKOJ PRAZNINI IZMEDJU DVA FIZIČKA SLOGA.DRUGA MOGUĆNOST, DRUGI NAČIN PREMOTAVANJA BRŽI JE OD TREĆEG NAJMANJE ZA VREME PRENOSA PODATAKA U OPERATIVNU MEMORIJU A U USLOVIMA MULTIPROGRAMINGA ODNOS BRZINA SE I POVEĆAVA U ZAVISNOSTI OD DRUGIH OBRADA. NA OSNOVU REČENOG LAKO SE UOČAVA DA SE BRZINSKO OPTIMIZIRANJE ISKORIŠĆENJA MAGNETNIH TRAKA SASTOJI U ODGOVARAJUĆEM KOBINOVANJU PREMOTAVANJA TRAKE KOJE U OVOM SLUČAJU PREDSTAVLJA IZBOR PRISTUPNE METODE.UTICAJ PRISTUPA NA ORGANIZACIJU PODATAKA ZAHTEVA POSEBNU I VRLO SLOŽENU ANALIZU PA JE ZBOG TOGA U OVOM TEKSTU DAT SAMO PRIKAZ KONKRETNIH JEDNOSTAVNIJIH PRAKTIČNO UPOTREBLJIVIH REŠENJA.

MOŽE SE APROKSIMATIVNO PRIHVATITI DA JE DIREKTAN PRISTUP PREMOTAVANJE MAGNETNE TRAKE NA OSNOVU TM S OBZIROM NA ODNOSE BRZINA PREMOTAVANJA. LAKO SE UOČAVA ANALOGIJA SE MAGNETNIM DISKOVIMA GDE JE KRETANJE GLAVA ZA ČITANJE I PISANJE NA DISKU OZNAČENO KAO DIREKTAN PRISTUP. PRISTUPU SLOGU NA OSNOVU OPISANOG SLOGA STAZE ODNOSNO NA OSNOVU REDNOG BROJA SLOGA NA STAZI ODGOVARALO BI PREMOTAVANJE TRAKE SLOG PO SLOG BEZ PRENOSA SADRŽAJA SLOGOVA U OPERATIVNU MEMORIJU.

IZMEDJU INDEKS SEKVENCIJALNE DATOTEKE NA DISKU I TAKVE DATOTEKE NA TRACI U ORGANIZACINOM POGLEDU NE MORA BITI NIKAKVE RAZLIKE.RAZLIKA JE U TOME ŠTO SU NA DISKU SA STANOVIŠTA VREMENSKE OPTIMIZACIJE ORGANIZACIONA PODRUČJA INDEKS SEKVENCIJALNE DATOTEKE (PODRUČJA INDEKSA I PODRUČJE PODATAKA) ODREDJENA HARDVERSKIM CELINAMA (STAZA.CILINDAR I DISK) DOK SE NA TRAKAMA HARDVERSKE CELINE PROIZVOLJNO (NA OSNOVU KONKRETNIH ZAHTEVA ORGANIZACIJE) ODREDJUJU IZUZEV RELACIJE DISK-KOTUR TRAKE. KOD INDEKS SEKVENCIJALNIH DATOTEKA NA DISKU HARDVERSKE CELINE ODREDJUJU I BROJ NIVOA INDEKSA (ODNOSI SE NA DATOTEKE ZA KOJE POSTOJI SOFTVERSKA PODRŠKA NA SVIM NIVOIMA PROGRAMIRANJA) DOK SE NA TRAKAMA TAJ BROJ TAKODJE MOŽE PROIZVOLJNO ODREDJIVATI. U DALJEM IZLAGANJU PREDPOSTAVIĆE SE DA JE BROJ NIVOA INDEKS DVA JER SE ODGOVARAJUĆA GENERALIZACIJA MOŽE PO ANALOGIJI JEDNOSTAVNO IZVESTI. PODRUČJA INDEKS SEKVENCIJALNE DATOTEKE NA TRACI ODVAJAJU SE SA TM I MOGU SE SMATRATI POSEBNIM SEKVENCIJALNIM DATOTEKAMA. NAJPRE DOLAZI PODRUČJE PRVOG NIVOA INDEKSA (VIŠI NIVO) U KOJEM SE NALAZE INDEKSI KOJI IDENTIFIKUJU REDNI BROJ PODRUČJA PODATAKA U KOJEM SE NALAZI TRAŽENI SLOG. IZA OVOG PODRUČJA (SEKVENCIJALNE DATOTEKE) DOLAZE PO DVA PODRUČJA, PRVO ZA NIŽI NIVO INDEKSA I DRUGO ZA SLOGOVE SA PODACIMA. U PODRUČJU NIŽEG NIVOA INDEKSA NALAZI SE REDNI BROJ FIZIČKOG BLOKA U KOJEM SE NALAZI TRAŽENI SLOG A PRISTUP SE U PODRUČJU FIZIČKIH BLOKOVA (PODRUČJU PODATAKA) OSTVARUJE PREMOTAVANJEM SLOG PO SLOG BEZ PRENOSA SADRŽAJA SLOGA U OPERATIVNU MEMORIJU. NARAVNO,PODRUČJE VIŠIH INDEKSA MOŽE DA PRATI I PO N DATOTEKA (SEKVENCIJALNIH) PRI ČEMU JE UVEK PRVA OD NJIH PODRUČJE INDEKSA KOJI SADA NE IDENTIFIKUJU FIZIČKI BLOK VEĆ SEKVENCIJALNU DATOTEKU. U TOM SLUČAJU VIŠI INDEKSI IDENTIFIKUJU N-TORKU PODRUČJA ODNOSNO SEKVENCIJALNIH DATOTEKA. MOGUĆNOST SA INDEKSIRANJEM FIZIČKIH BLOKOVA NAVEDENA JE ZBOG TOGA ŠTO SE FIZIČKI SLOGOVI NA TRACI MOGU ELASTIČNIJE PROJEKTOVATI NEGO NA DISKU. PROJEKTOVANJE VELIČINA ODGOVARAJUĆIH PODRUČJA I BROJA NIVOA INDEKSA VRŠI SE NA OSNOVU ZAHTEVA KONKRETNE ORGANIZACIJE. NA TRACI JE ČESTO VRLO EFIKASNO PODRUČJE VIŠIH INDEKSA PONAVLJATI PRE SVAKE N-TORKE DATOTEKE ŠTO NAROČITO DOLAZI DO IZRAŽAJA AKO ULAZNI SLOGOVI KOJI SE TRAŽE NISU SORTIRANI. OVA SE REDUDANTNOST NE ODRAŽAVA ZNAČAJNO U NAJČEŠĆIM SLUČAJEVIMA, NA ISKORIŠĆENJE MEMORIJSKOG KAPACITETA TRAKE.

ORGANIZOVANJE PODRUČJA PREKORAČENJE JE KOD INDEKS SEKVENCIJALNIH DATOTEKA NA TRAKAMA DALEKO IZRAŽENIJI PROBLEM JER JE SPREZANJE SLOGOVA (SLOGOVI U SEBI SADRŽE POKAZIVAČ KOJI PREDSTAVLJA FIZIČKU ADRESU ILI NEKO PRESLIKAVANJE FIZIČKE ADRESE SLOGA) NA MAGNETNIM TRAKAMA PRIHVATLJIVO JEDINO AKO SE RADI O VRLO MALOM, NEZNATNOM,BROJU SLOGOVA. SPREZANJE SLOGOVA ZAHTEVALO BI ČESTA PREMOTAVANJA TRAKE NAPRED-NAZAD ŠTO BI ZNATNO UTICALO NA VREME PRISTUPA SLOGOVIMA INDEKS SEKVENCIJALNE DATOTEKE NA MAGNETNOJ TRACI. KAKO SE PODRUČJE PREKORAČENJA MOŽE ORGANIZOVATI I BEZ SPREZANJA SLOGOVA TO SE OVAKVO REŠENJE PREDLAŽE ZA INDEKS SEKVENCIJALNE DATOTEKE NA MAGNETNIM TRAKAMA. POred TOGA, U ORGANIZACIJI INDEKS SEKVENCIJALNIH DATOTEKA NA MAGNETINIM TRAKAMA SIGURNO JE POVOLJNIJE PREDVIDETI SAMO NEZAVISNO PODRUČJE PREKORAČENJE,PODRUČJE PREKORAČENJA ZA INDEKSNU DATOTEKU U CELINI,JER SE I INDEKS SEKVENCIJALNE DATOTEKE NA TRAKAMA RAZMATRAJU SAMO U SLUČAJEVIMA KADA JE BROJ SLOGOVA PREKORAČIOCA, SLOGOVA KOJE BI TREBALO DODATI U VEĆ FORMIRANU INDEKSNU DATOTEKU, DOVOLJNO MALI. NARAVNO,MOGUĆE JE I SA STANOVIŠTA ORGANIZACIJE I PROGRAMIRANJA LAKO IZVODLJIVO PROJEKTOVATI I POSEBNA PODRUČJA PREKORAČENJA, PODRUČJA PREKORAČENJA ZA SVAKU INDEKSIRANU N-TORKU SEKVENCIJALNIH DATOTEKA, KAO POSEBNE SEKVENCIJALNE DATOTEKE IZA SVAKE INDEKSIRANE N-TORKE.

TRAŽENJE SLOGOVA U INDEK SEKVENCIJALNOJ DATOTECI NA MAGNETNOJ TRACI ZASNOVANO JE NA ISTOM PRINCIPU KOJI JE PRISUTAN I KOD SVIH OSTALIH INDEKSNIH DATOTEKA. NAJPRE SE SEKVENCIJALNO TRAŽI U SEKVENCIJALNOJ DATOTECI VIŠIH INDEKSA REDNI BROJ N-TORKE SEKVENCIJALNIH DATOTEKA KOJA SADRŽI TRAŽENI SLOG. REDNI BROJ PREDSTAVLJA FIZIČKU ADRESU I ZA SEKVENCIJALNE DATOTEKE I ZA SLOGOVE NA TRACI ŠTO JE DEFINISANO FIZIČKIM MEHANIZMIMA PRISTUPA. NA OSNOVU DOBIJENOG REDNOG BROJA LAKO SE IZRAČUNAVA BROJ SEKVENCIJALNIH DATOTEKA ODNOSNO BROJ TM KOJE SE PREMOTAVAJU.

OVO PREMOTAVANJE JE RELATIVNO IZUZETNO BRZO JER NE ZAVISI OD BROJA SLOGOVA NA DELU TRAKE KOJI SE PREMOTAVA. POSTUPAK SE DALJE NASTVLJA DOK SE NE PRONADJE TRAŽENI SLOG.

ZA EFIKASNOST INDEKS SEKVENCIJALNE DATOTEKE NA MAGNETNOJ TRACI NAROČITO JE VAŽNAS SORTIRANOST ULAZNIH SLOGOVA KOJI SE TRAŽE. AKO SU ULAZNI SLOGOVI U POTPUNOSTI SORTIRANI NA ODGOVARAJUĆI NAČIN ONDA JE PREMOTAVANJE TRAKE UNAZAD SVEDENO NA MINIMUM ILI SE MOŽE U POTPUNOSTI IZBEĆI. U POTPUNOSTI SE MOŽE IZBEĆI AKO SE SVI POTREBNI PARMETRI ZA TRAŽENJE PRATE PARALELNO ZA SVE SLOGOVE KOJI SE TRAŽE ŠTO JE KOD INDEKSNIH DATOTEKA NA TRACI MOGUĆE JER SE PREDPOSTAVLJA DA SE UVEK TRAŽI RELATIVNO MALI BROJ SLOGOVA. SADA SE LAKŠE UOČAVA KAKO PONAVLJANJE VIŠIH INDEKSA POVEĆAVA EFIKASNOST PRIMENE INDEKS SEKVENCIJALNIH DATOTEKA NA MAGNETNIM TRAKAMA.SMANJUJE SE KRETANJE UNAZAD ALI SE I POVEĆAVA BROJ POTREBNIH UPISA. REČENO VAŽI BEZ OBZIRA DA LI JE SLOG PRONADJEN ILI GA TREBA DODATI U DATOTEKU. JASNO JE, MEDJUTIM, DA KOD DODAVANJA SLOGOVA ORGANIZACIJA PODRUČJA PREKORAČENJA ZAJEDNO SA SORTIRANOŠĆU ULAZNIH SLOGOVA UTIČE NA BRZINU PRISTUPA. AKO SU ULAZNI SLOGOVI SORTIRANI I AKO IH JE MOGUĆE PARALELNO PRATITI ONDA JE PRISTUP BRŽI SA NEZAVISNIM PODRUČJEM PREKORAČENJA ALI AKO ULAZNI SLOGOVI NISU SORTIRANI (I NE PRATE SE PARALELNO) ONDA POSEBNA PODRUČJA PREKORAČENJA IZA SVAKE N-TORKE SMANJUJU KRETANJA NAPRED-NAZAD PA TIME POVEĆAVAJU BRZINU PRISTUPA. KONAČNO REŠENJE PROJEKTA INDEKS SEKVENCIJALNE DATOTEKE NA MAGNETNOJ TRACI ZAVISI OD KONKRETNOG PROBLEMA. AŽURIRANJE SLOGOVA U INDEKS SEKVENCIJALNOJ DAT-

OTECI NA TRACI JE NEDOSTATAK OVE ORGANIZACIJE PODATAKA JER SE ZBOG AŽURIRANJA SMANJUJE KAPACITET TRAKE. AŽURIRANJE PODRAZUMEVA ULAZNO-IZLAZNE (UPDATE) DATOTEKE U KOJIMA JE MOGUĆE I ČITANJE I PISANJE SLOGOVA A AŽURIRANJE JE I JEDNA OD PREDPOSTAVKI INDEKSNIH DATOTEKA. NAPOMENIMO DA ZA ULAZNO-IZLAZNE DATOTEKE NA MAGNETNIM TRAKAMA NE POSTOJI SOFTVERSKA PODRŠKA U VIŠIM PROGRAMSKIM JEZICIMA U OPERATIVNOM SISTEMU. RADI SE O TOME ŠTO SE KOD PISANJA SLOGOVA IZA SLOGA OBRAZUJE VEĆA PRAZNINA PA JE POTREBNO IZA SVAKOG SLOGA KOJI ĆE SE AŽURIRATI REALIZOVATI DODATNU PRAZNINU ZA ŠTA POSTOJI ODGOVARAJUĆA INSTRUKCIJA. DA BI DODATNE PRAZNINE ŠTO MANJE UTICALE NA KAPACITET TRAKE POTREBNO JE DA FIZIČKI SLOGOVI BUDU ŠTO DUŽI. POVEĆANJE DUŽINE SLOGOVA (BLOKOVA) POVEĆAVA I BRZINU PRISTUPA U INDEKSNIM DATOTEKAMA AKO SE TRAŽENJE SLOGOVA NA NIVOU BLOKA VRŠI U OPERATIVNOJ MEMORIJI ŠTO NAROČITO DOLAZI DO IZRAŽAJA KOD INDEKSNIH DATOTEKA KOJE SE POSEBNO PROJEKTUJU.

III NEKI PRIMERI PRIMENE INDEKS SEKVENCIJALNIH DATOTEKA NA MAGNETNIM TRAKAMA

JEDNU OD NAJZNAČAJNIJIH PRIMENA, SA STANOVISTA SISTEM ANALIZE, INDEKS SEKVENCIJALNE DATOTEKE NA MAGNETNIM TRAKAMA NALAZE KOD ORGANIZOVANJA VELIKIH ARHIVA KOJE SU ZBOG ZAHTEVA ORGANIZACIJE I OBRADE INDEKSIRANE. RADI SE DAKLE O ARHIVAMA KOJE SU KAO INDEKS SEKVENCIJALNE DATOTEKE ORGANIZOVANE NA MAGNETNIM DISKOVIMA ILI DOBOŠIMA. OSNOVNI PROBELEM JE ŠTO JE ZA MNOGE OD TIH ARHIVA NEEKONOMIČNO DRŽATI IH I ČUVATI STALNO NA DISKU A POTREBNO JE KOPIRATI IH I ZBOG ZAŠTITE. OVO ZNAČI DA SE PRE I POSLE SVAKE OBRADE OVE DATOTEKE KOPIRAJU NA DISK I SA DISKA VRAĆAJU NA TRAKU. TU SE I NALAZE MOGUĆNOSTI UŠTEDE U VREMENU JER JE MOGUĆE ZA RELATIVNO MALI BROJ SLOGOVA KOJE TREBA OBRADITI KORISTITI TE DATOTEKE KAO INDEKS SEKVENCIJALNE DATOTEKE SA TRAKE. EKONOMIČNOST INDEKS SEKVENCIJALNIH DATOTEKA NA TRAKAMA SE LAKO ODREDJUJE NA OSNOVU STATISTIČKIH VREDNOSTI ZAHTEVA OBRADE I VREMENA PRISTUPA KOD INDEKSNE DATOTEKE NA TRACI ODNOSNO VREMENA KOPIRANJA DISK-TRAKA I TRAKA-DISK. DODATNI SOFTVERSKI NAPOR MOŽE I DA SE NE UZIMA U OBZIR AKO JE TO ISKLJUČIVA MOGUĆNOST POVEĆANJA VREMENSKIH KAPACITETA RAČUNARA.

ZBOG POJAVE NOVIH ORGANIZACIJA PODATAKA, VSAM I BANKE PODATAKA, MOŽE SE UČINITI DA JE OVA ANALIZA ZASTARELA I BEZ ZNAČAJA. OVO SIGURNO NIJE TAČNO IZ VIŠE RAZLOGA. PRE SVEGA, MALI JE BROJ CENTARA KOJI USPEŠNO PRIMENJUJU BANKE PODATAKA (JOŠ SU U UPOTREBI I RAČUNARI NA KOJIMA ZBOG KAPACITETA NIJE NI MOGUĆE INSTALIRATI ODGOVARAJUĆI SOFTVER ZA PODRŠKU BANKI PODATAKA) A SIGURNO MANJI BROJ RAČUNSKIH CENTARA KOJI SU SVOJ KOMPLETAN APLIKATIVNI SOFTVER BAZIRALI NA BANKAMA PODATAKA . DALJE, UVEK ĆE POSTOJATI TAKVE APLIKACIJE RAČUNARA U KOJIMA ĆE DATOTEKE KAO ORGANIZACIONE JEDINICE PODATAKA BITI EKONOMIČNIJE OD BANKI PODATAKA.

S DRUGE STRANE, VSAM DATOTEKE I INDEKS SEKVENCIJALNE DATOTEKE (ISAM) ZAJEDNO U OSNOVI IMAJU ISTI LOGIČKI MEHANIZAM (PRINCIP) ORGANIZACIJE A TO JE INDEKSIRANJE. OVE DVE ORGANIZACIJE PODATAKA RAZLIKUJU SE SAMO U ORGANIZACIJI PODRUČJA PREKORAČENJE (VSAM SADRŽI I DISTRIBUCIJU SLOBODNOG PROSTORA) ALI OVO NIJE DOVOLJNO ŠIROKO POZNATA ČINJENICA. ZBOG TOGA SE ČESTO NEOPRAVDANO DAJE ISKLJUČIVA PREDNOST VSAM-U (ISAM SE POGREŠNO SMATRA ISLUŽENIM) A TOME DOPRINOSE I NEKE ''DODATNE MOGUĆNOSTI'' VSAM-A KOJE NISU NIŠTA DRUGO NEGO ODGOVARAJUĆE ORGANIZACIJE SEKUNDARNIH KLJUČEVA.

INDEKSIRANJE KAO PRINCIP ORGANIZACIJE PODATAKA NA MAGNETNIM TRAKAMA SVOJE ZNAČAJNIJE PRIMENE MOŽE NAĆI U SISTEM PROGRAMIRANJU KAO IZDVOJENOM POSEBNOM OBLIKU PRISTUPA ODRŽAVANJU RAČUNARSKIH SISTEMA. BIĆE OPISANE DVE OBLASTI PRIMENE ALI SU MOGUĆNOSTI ZNATNO ŠIRE.

U MNOGIM RAČUNSKIM CENTRIMA EVIDENCIJA O STANJU NA MAGNETNIM TRAKAMA PREDSTAVLJA PRAVI PROBLEM. JEDINO REŠENJE U VEZI SA PRAĆENJEM STANJA NA MAGNETNIM TRAKAMA, BROJ DATOTEKA NA TRACI, VELIČINE DATOTEKA I SLOGOVA, ROKOVI ZAŠTITE DATOTEKA, ORGANIZACIONA PRIPADNOST DATOTEKA ....SU KARTOTEKE (ILI DATOTEKE ALI TO NIJE ZADOVOLJAVAJUĆI NIVO OPTIMIZACIJE JER SU OVE DATOTEKE SAMO ODGOVARAJUĆE KOPIJE KARTOTEKA) KOJE JE VRLO ČESTO VRLO TEŠKO ODRŽAVATI ZBOG BROJA KOJIMA CENTRI RASPOLAŽU. INDEKSIRANJE NA MAGNETNIM TRAKAMA OMOGUĆAVA POTPUNO RAZREŠENJE OVOG PROBLEMA. NA OSNOVU INDEKSIRANJA MOGUĆE JA NA SVAKOJ TRACI ORGANIZOVATI POSEBNO PODRUČJE (JEDNU RELATIVNO VRLO MALU SEKVENCIJALNU DATOTEKU) U KOJE BI SE NALAZILI JEDINO UPISI O SADRŽAJU TRAKE (POTPUNA ANALOGIJA SA VTOC-OM NA DISKU) A KOJE BI OMOGUĆILO AUTOMATIZOVANJE EVIDENCIJE O MAGNETNIM TRAKAMA.

U PODRUČJU KOJE BI SADRŽALO OPIS SADRŽAJA TRAKE PODACI SE MOGU ORGANIZOVATI I INDEKSIRATI NA RAZLIČITE NAČINE ALI JE NAJBOLJE DA SVAKU DATOTEKU NA TRACI IDENTIFIKUJE FIZIČKA DUŽINA DELA TRAKE NA KOJEM SE OVA DATOTEKA NALAZI. OVO ZBOG TOGA ŠTO BI SE NA OSNOVU TOG PODATAKA NAJEKONOMIČNIJE MOGLA KORISTITI SLOBODNA PODRUČJA NA TRACI (FREE EXTENTS) KOJA MOGU NASTATI ZBOG RAZLIČITIH ZAHTEVA OBRADE. PRISTUP OVAKO ORGANIZOVANIM DATOTEKAMA TRAKE MOŽE SE ODGOVARAJUĆIM, VRLO JEDNOSTAVNIM PROGRAMOM POTPUNO AUTOMATIZOVATI. PROGRAM SAM BIRA NAJEKONOMIČNIJE PODRUČJE ZA SMEŠTAJ DATOTEKE ILI JAVLJA O MOGUĆOJ REORGANIZACIJI TRAKE ILI TRAŽI NOVI TRAKU (AKO SE RADI O KREIRANJU NOVE DATOTEKE) ILI JEDNOSTAVNO I AUTOMATSKI PRISTUPA TRAŽENOJ DATOTECI. TAKODJE JE VRLO JEDNOSTAVAN PROGRAM KOJI VRŠI REORGANIZACIJU TRAKE ODNOSNO KOJI JEDNOSTAVNO PREPIŠE POTREBNE DATOTEKE SA TRAKE KOJA SE REORGANIZUJE NA NOVU TRAKU BEZ SLOBODNIH PODRUČJA. BEZ MNOGO NAPORA MOGUĆE JE I REDOSLED DATOTEKA KOD REORGANIZACIJE PODREDITI STATISTIČKIM ZAHTEVIMA OBRADE ČIME SE MOŽE UŠTEDETI U VREMENU PRI PREMOTAVANJIMA. INDEKSIRANJE SE NA MAGNETNIM TRAKAMA MOŽE VRLO EFIKASNO PRIMENJAVATI I KOD ARHIVIRANJA PROCEDURA. OVO SE ODNOSI KAKO NA PROCEDURE KOJE SE ČUVAJU NA TRAKAMA ZBOG EVENTUALNIH OŠTEĆENJA BIBLIOTEKE PROCEDURA TAKO I NA PROCEDURE KOJE SE NE UPOTREBLJAVAJU DOVOLJNO ČESTO DA BI SE DRŽALE U BIBLIOTECI. INDEKSIRANJE OMOGUĆAVA POTPUNO AUTOMATSKI PRISTUP NA OSNOVU IMENA PROCEDURA NA TRACI KOJA SE DALJE MOŽE DIREKTNO SA TRAKE IZVODITI.

IV INDEKS NESEKVENCIJALNE DATOTEKE NA MAGNETNIM TRAKAMA

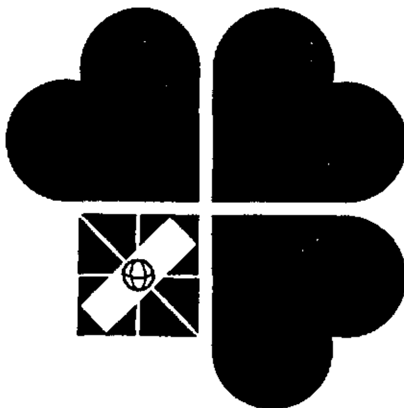
KOD INDEKS NESEKVENCIJALNIH DATOTEKA U PODRUČJU INDEKSA NALAZE SE ADRESE (ILI NEKA PRESLIKAVANJA ADRESA) ENTITETA KOJIMA SE PRISTUPA. PRVI PROBLEM VEZAN ZA INDEKS NESEKVENCIJALNE DATOTEKE NA MAGNETNIM TRAKAMA JE DODAVANJE NOVIH SLOGOVA U OBA PODRUČJA, I U PODRUČJU INDEKSA I U PODRUČJU PODATAKA. OVO DODAVANJE SE KOD INDEKS NESEKVENCIJALNIH DATOTEKA NAJČEŠĆE VRŠI S KRAJA DATOTEKE. ZBOG BRZINE TRAŽENJA I PRETRAŽIVANJA JEDINO PRIHVATLJIVO REŠENJE JE DA SE PODRUČJE INDEKSA NALAZI NEPOSREDNO ISPRED PODRUČJA PODATAKA. SLOBODNI PROSTOR IZA PODRUČJA INDEKSA MOŽE SE OSTVARITI REALIZOVANJEM PRAZNINA ILI FIKTIVNIM UPISIVANJEM FIZIČKIH BLOKOVA ILI NA NEKI DRUGI NAČIN. U PODRUČJU INDEKSA PRISTUPA SE SEKVENCIJALNO, CELO PODRUČJE MOŽE DA BUDE I SAMO JEDAN FIZIČKI BLOK, DOK SE U PODRUČJU PODATAKA KORISTE I DRUGI, BRŽI OBLICI PRISTUPA U ZAVISNOSTI OD DOBIJENIH ADRESA FIZIČKIH BLOKOVA KOJIMA JE POTREBNO PRISTUPITI.

KAKO JE OSNOVNA KARAKTERISTIKA INDEKS NESEKVENCIJALNIH DATOTEKA NESORTIRANOST U PODRUČJU PODATAKA TO JE PREDPOSTAVKA ORGANIZOVANJA INDEKSNESEKVENCIJALNIH DATOTEKA NA MAGNETNIM TRAKAMA SEKVENCIJALNO TRAŽENJE I PRETRAŽIVANJE U PODRUČJU PODATAKA KOJE SE LAKO OSTVARUJE SORTIRANJEM ZAHTEVA ZA SLOGOVE KOJI SU DOBIJENI IZ PODRUČJA INDEKSA.

V ZAKLJUČAK

SVRHA OVOG RADA BILA JE DA POKAŽE DA INDEKSIRANJE NA MAGNETNIM TRAKAMA I TE KAKO IMA SMISLA I DA JE SA ORGANIZACIONOG STANOVIŠTA VRLO LAKO IZVODLJIVO. NI SOFTVERSKA PODLOGA NE BI TREBALO DA PREDSTAVLJA VEĆI PROBLEM JER JE ZA PROGRAMIRANJE REČENOG DOVOLJAN PROSEČAN ASEMBLERSKI NIVO.

RAČUNARI JESU MAŠINE KOD KOJIH JE OTVOREN PROBLEM EKONOMIČNOST ISKORIŠĆENJA KAPACITETA KUPOVNOM NOVE MOĆNIJE MAŠINE ILI OPTIMIZACIJA ISKORIŠĆENJA STARE. OVO DRUGO POSTAJE DOVOLJNO SKUPO ZNATNO SE POVEĆAVA SLOŽENOST SOFTVERA, DA SAMO SEBE DOVODI U PITANJE. MEĐUTIM U USLOVIMA OTEŽANE NABAVKE ILI ZAMENE SISTEMA, KADA NEMA IZBORA, JEDINO REŠENJE JE UPRAVO OPTIMIZACIJA ISKORIŠĆENJA POSTOJEĆIH KAPACITETA. TIME I ANALIZA IZNEŠENA U OVOM TEKSTU DOBIJA SVOJ PRAVI ZNAČAJ.



IFIP CONGRESS 86

DUBLIN IRELAND 1-5 SEPTEMBER 1986

10th WORLD COMPUTER CONGRESS

Programme Streams

The programme is divided into four streams.

Design of Components

Applications

Design of Systems

Informatics in a Developing World.

I. Design of Components

This stream emphasises the individual disciplines which form the basis of any system design and end user application. The results presented in this stream will eventually make new innovations feasible in the following areas

Theoretical Computer Science

- semantics of new program and system concepts
- new optimisation and transformation techniques
- algorithms for new computer structures (e.g. parallel, distributed, VLSI)
- algorithms for new applications in knowledge engineering, image processing etc

Programming Science and Methodology

- theory and methodology of programming
- program and system specification
- formal systematic program development
- logic, functional and concurrent programming
- verification and rewrite systems

Software Engineering

- programming support and system design environments
- program quality, reliability and robustness measures
- issues related to mass production of software
- application system development by the end user
- new management practices
- engineering for non-functional requirements

Computer Engineering

- supercomputer architectures
- dataflow machines
- ultra-high speed and VLSI
- innovative microprocessor developments
- systolic array architectures
- communication devices machine-machine, graphics, video, sound, touch-sensitivity
- secure systems, high-level language machines

II. Design of Systems

This stream emphasises methods, techniques and technologies which relate the various disciplines cooperating in a system. These are

Distributed Systems

- architectural models of distributed systems
- specification and verification methods
- electronic mail
- network operating systems
- satellite communication
- computer based communication and the PTT

Information Systems

- requirements analysis for data modelling
- techniques and tools for specification
- databases and techniques for office systems
- database management systems for small machines
- new approaches database machines and innovative data modelling techniques

Artificial Intelligence

- knowledge representation techniques
- reasoning
- tools and methods for knowledge engineering
- major application areas, natural language interfaces, vision, speech, learning
- how to get started in artificial intelligence

III. Applications

There will be case studies showing developments of general interest, particularly those which have achieved a high degree of relevance to user needs at reasonable cost and full reliability, and also those which demonstrate innovative applications of informatics

Computer Integrated Manufacturing

- computer aided design engineering and manufacturing applications
- design problems and their solutions
- problems of inter-disciplinary approaches
- prospects and consequences

New Informatics Applications

- new office automation applications
- integrated publishing
- electronic journals, funds transfer etc
- electronic distribution of software products
- recreational applications

IV. Informatics in a Developing World

This stream emphasises the legal, social and economic aspects of informatics. Subjects include issues of concern to industrialised nations, to developing countries and the impact of information technology on international policy and planning. Also discussed will be the varied reactions of countries in terms of informatics research and development programmes

Exhibition

In conjunction with the Irish Business Equipment Trade Association (IBETA) a major exhibition of computer systems and services will be held in the Royal Dublin Society. If you wish to obtain further details of this exhibition please complete and return the attached reply card.

If you wish to receive the final call for papers and third announcement please complete the attached reply card and return to:

Congress Secretariat,
IFIP Congress '86,
44 Northumberland Road,
Dublin 4, Ireland.
Tel. (01) 688244
Telex 31098
Telegrams: Congrex, Dublin

PARARELNO INDEKSIRANJE

SINIŠA J. DJORDJEVIĆ

UDK: 519.683.4

U tekstu se razmatra indeksiranje u kojem presek indeksnih grupa slogova nije prazan skup već se organizacija indeksiranja prilagodjava preseku indeksiranih grupa slogova.

PARALLEL INDEXING: This paper will consider indexing in which section of index record groups is not an empty set but indexing organization is subordinated to the section of indexed record groups.

1. UVOD

Indeksiranje je preslikavanje grupe slogova na ključ sloga sa najvećom ili najmanjom adresom u okviru te grupe. Time se traženje ili pretraživanje prepušta algoritmu kojim se omogućuje smanjenje srednjeg broja pristupa što dalje omogućava prilagodjenja u okviru realnog vremena upotrebe računara. Za indeksiranja koja se u praksi upotrebljavaju karakteristično je da je presek indeksnih grupa slogova jednak nuli, da je to prazan skup. Drugim rečima, slog pripada samo jednoj indeksiranoj grupi.

U ovom tekstu razmatra se indeksiranje u kojem presek slogova iz indeksnih grupa slogova nije prazan skup već se organizacija indeksiranja podredjuje preseku indeksiranih grupa slogova. Ovakvo indeksiranje razmatra se sa aspekta memorijskog prostora i srednjeg broja pristupa s ciljem da se pokaže da indeksiranje u kojem presek slogova indeksiranih grupa slogova nije prazan skup može da ima prednosti u odnosu na postojeće široko primenjivane oblike indeksiranja.

Ovakvo indeksiranje podrazumeva novi algoritam kako u samom organizovanju indeksiranja tako i u projektovanju metoda traženja i pretraživanja. Te metode direktne su implikacije organizacije indeksiranja pa nisu posebno razmatrane.

2. MEMORIJSKI PROSTOR KOD INDEKSIRANJA SA VIŠE STABALA SA JEDNIM NIVOOM

Pod paralelnim indeksiranjem podrazumevaće se indeksiranje u kojem presek indeksiranih grupa slogova nije prazan skup. Presek indeksiranih grupa slogova nije ni proizvoljan skup u odnosu na broj slogova u tom skupu jer se razmatra paralelno indeksiranje koje može, uz odredjene predpostavke u organizaciji, da pruži odredjena poboljšanja. Za dalju analizu pretpostavljaće se da je presek indeksiranih grupa slogova ili prazan skup ili skup sa konstantnim brojem slogova. Jednostavan primer ove organizacije prikazan je na slici 1.

U ovom delu teksta razmatraće se samo stabla sa jednim nivoom indeksiranja jer uz male algoritamske napore pružaju zadovoljavajuće rezultate.

S druge strane, indeksiranje u kojem je presek indeksiranih grupa prazan skup posmatraće se sa najopštijih aspekata i poslužiće kao referentno indeksiranje jer je u praksi jedino upotrebljavano indeksiranje.

Za indeksiranje sa jednim stablom i više nivoa, indeksiranje u kojem je presek prazan skup, za memorijski prostor ima se:

$$M = \sum_{i=1}^{n} P_i \quad ; \quad P_i = \sum_{j=1}^{P_{i-1}} h_{i,j} \qquad \ldots (1)$$

gde je M broj zapisa na svim nivoima, P_i broj zapisa na i-tom nivou, $P_o = 1$, i n broj nivoa. Memorijski prostor dat je indirektno preko broja zapisa jer se iz tog broja i dužine zapisa koja je unapred definisana može jednostavno odrediti. $h_{i,j}$

je broj zapisa j-te indeksirane grupe na i-tom nivou. Relacija (1) važi u opštem slučaju, svaka indeksirana grupa može imati proizvoljan broj zapisa. Ograničenje je da je presek indeksiranih grupa na jednom nivou prazan skup i da je unija indeksiranih grupa na tom nivou jednaka skupu koji sadrži sve zapise na tom nivou.

Ako je broj zapisa u indeksiranim grupama na jednom nivou konstantan, $h_{i,j}$ ne zavisi od j, onda je:

$$M = \sum_{i=1}^{n} \prod_{k=1}^{i} h_k \quad , h_k = h_i \quad \text{za } k=i \quad \ldots \quad (2)$$

Za više stabala (s) sa jednim nivoom ima se:

$$M = \sum_{i=1}^{s} \lceil Q/r_i \rceil \quad \ldots \quad (3)$$

Relacija $\lceil a/b \rceil$ u skupu realnih brojeva daje najmanji prirodan broj koji je veći od a/b , ako a/b nije prirodan broj inače daje sam taj broj a/b . Q je broj zapisa (slogova) u datoteci koja se indeksira a r_i je korak indeksiranja odnosno broj slogova u indeksiranim grupama i-tog stabla. Uslovi u (3) nisu sasvim opšti jer se pretpostavlja da indeksirane grupe u jednom stablu imaju isti broj slogova. Sasvim opšti uslovi nemaju nemaju značaja za analizu koja će u ovom tekstu biti iznešena.

Kada je r_i = const. , kada indeksirane grupe u svim stablima imaju jednak broj zapisa onda je:

$$M = s\lceil Q/r \rceil \quad \ldots \quad (4)$$

U zavisnosti od algoritma traženja, da bi se obezbedio algoritam sa zadovoljavajućom jednostavnošću, stabla sa jednim nivoom se formiraju na sledeći način:

Indeksira se svaki r-ti zapis za prvo stablo i $(i\lceil r/(s-1)\rceil + kr)$-ti zapis za (i+1)-to $(1 \leq i \leq s-1)$ stablo, $k \in (0,1,\ldots,s-1)$. Korak indeksiranja je $\lceil r/(s-1)\rceil$ i algoritam traženja postaje jednostavan. Minimalna vrednost $\min(a_1,\ldots,a_i,\ldots,a_s)$ je (a_i – indeks za neki zapis iz i-tog stabla) najbliži indeks i korak je $\lceil r/(s-1)\rceil$.

Potrebno je uporediti obe metode indeksiranja. Na osnovu (2) za korak indeksiranja dobija se:

$$r = \lceil Q/(\prod_{i=1}^{n} h_i) \rceil \quad \ldots \quad (5)$$

Ako se upotrebi korak indeksiranja r_1 za indeksiranje sa više stabala za isti korak indeksiranja kao kod jednog stabla sa više nivoa potrebno je $\lceil r_1/r \rceil + 1$ stabala. r_1 je korak indeksiranja svakog stabla kod indeksiranja sa više stabala a r je korak indeksiranja u datoteci koji se ostvaruje primenom opisanog algoritma traženja za paralelno indeksiranje.

Ako se uporede brojevi svih zapisa u obe organizacije onda se poredi:

$$\sum_{i=1}^{n-1} \prod_{k=1}^{i} h_k \quad \text{i} \quad (\lceil r_1/r \rceil + 1)\lceil Q/r_1 \rceil \quad \ldots \quad (6)$$

Kako je:

$$\sum_{i=1}^{n} \prod_{k=1}^{i} h_k = \sum_{i=1}^{n-2} \prod_{k=1}^{i} h_k + \prod_{k=1}^{n-1} h_k + \prod_{k=1}^{n} h_k \quad \ldots \quad (7)$$

i kako se može uzeti:

$$\prod_{i=1}^{n} h_i \approx Q/r \quad , \quad \lceil r_1/r \rceil + 1 \approx s \quad ,$$

$r_1 \approx (s-1)r$, to se dobija:

$$\sum_{i=1}^{n-2} \prod_{k=1}^{i} h_k + Q/h_n r + Q/r \geq sQ/(s-1)r \quad \ldots \quad (9)$$

Relacijom (9) pokazuje se da indeksiranje sa više stabala sa jednim nivoom sadrži manji broj zapisa. Ispravnost relacije (9) jednostavno se pokazuje, desna strana postaje $Q/r + Q/(s-1)r$ pa je izmedju ostalih uslova dovoljno da je $h_n < s-1$ da bi relacija (9) važila.

Na slici 1. prikazane su dve organizacije indeksiranja, sa dva stabla sa jednim nivoom i organizacija sa jednim stablom i tri nivoa. Broj zapisa za paralelno indeksiranje je 9 a za indeksiranje sa jednim stablom broj zapisa je 14 pri čemu je isti korak indeksiranja za obe organizacije i iznosi 2. Kod paralelnog indeksiranja reorganizacija, umetanje i brisanje, distribucija slobodnog prostora i organizacija područja prekoračenja podrazumevaju nove algoritme ali se ti algoritmi jednostavno organizuju jer je i paralelno indeksiranje sa stanovišta organizacije vrlo jednostavno.

3. SREDNJI BROJ PRISTUPA KOD INDEKSIRANJA SA VIŠE STABALA SA JEDNIM NIVOOM

Srednji broj pristupa kod sukcesivnog traženja i po indeksima i u datoteci manji je kod jednog stabla sa više nivoa nego kod više stabala sa jednim nivoom.

Za srednji broj pristupa Z ima se:

$$Z = \sum_{i=1}^{n} p_i c_i = (1/n) \sum_{i=1}^{n} i \quad , \quad p_i = 1/n \quad , \quad c_i = i \quad ,$$

$$Z = (n+1)/2$$

gde je p_i verovatnoća traženja i-tog sloga a c_i broj pristupa do i-tog sloga.

Za jedno stablo sa više nivoa izraz postaje:

$$Z_1 = \sum_{i=1}^{n} (h_i+1)/2 + (r+1)/2 =$$

$$(1/2)\sum_{i=1}^{n} h_i + (n+r+1)/2$$

Za više stabala sa jednim nivoom izraz postaje:

$$Z_2 = s(\lceil Q/r_1 \rceil + 1)/2 \approx s(Q/((s-1)r)+1)/2 \approx$$

$$Q/2r + s/2$$

Za uporedjivanje Z_1 i Z_2 potrebno je uporediti:

$$\sum_{i=1}^{n} h_i + n + r + 1 \qquad i \qquad Q/r + s$$

Za uporedjivanje Z_1 i Z_2 koristiće se, za odredjene parametre, vrednosti definisane teorijom traženja.

Iz teorije traženja poznato je da Z_1 ima minimum kada je h_i = const. a kako je s druge strane $Q/r = h^n$ to se dobija:

$$n(h+1) + r + 1 \leq h^n + s \quad , \quad h > 2 \ \ldots \quad (10)$$

Relacija (10) važi jer je u praksi $s > r+1$ a indukcijom može da se pokaže da je za $n > 1$ i $h > 2$, n i h prirodni brojevi, ispunjen uslov $n(h+1) < h^n$ kojim se dokazuje relacija (10).

Medjutim, u organizacijama indeksiranja sa više stabala sa jednim nivoom, ako su stabla sortirana prema prvom zapisu u stablu i ako su stabla organizovana tako da je moguć direktni pristup u stabla onda sukcesivno traženje za paralelno indeksiranje ima manji srednji broj pristupa. Ne radi se o čisto sukcesivnom traženju već se u stabla uskače nekom od metoda direktnog traženja a onda se postupak traženja nastavlja sukcesivno.

I druge metode traženja takodje daju bolje rezultate kod paralelnog indeksiranja ali je to izvan tematskih okvira ovog teksta.

Ako identifikujemo j-ti zapis u i-tom stablu onda jedini mogući zapisi iz (i+1) - tog stabla mogu biti j-ti ili (j-1)-ti zapis. Ako je to (j-1)-ti zapis traženje se završava.

Dokaz je trivijalan.

Umesto (j-1)-tog zapisa traženje može da upućuje i na (j+1)-ti zapis što zavisi od parametra indeksiranja. Bitno je da je potrebno u svakom stablu izuzev prvog tražiti samo u okviru dva zapisa koja se unapred mogu odrediti.

Za srednji broj pristupa ako se u svakom stablu izuzev prvog traži na nivou dva zapisa dobija se:

$$Z_2' = (1/2)(\lceil Q/r_1 \rceil + 1) + (s-1)(2+1)/2 =$$

$$(1/2)(\lceil Q/r_1 \rceil + 3s - 2)$$

Ako se ne traži u svim stablima već se traženje završi u stablu u kojem je rezultat (j-1)-ti zapis onda se za srednji broj pristupa dobija:

$$Z_2'' = (1/2)(\lceil Q/r_1 \rceil + 1) + ((s-1)/2)((2+1)/2) =$$

$$(1/2)(\lceil Q/r_1 \rceil + 3s/2 - 1/2)$$

Jedna od promenljivih, s ili r_1, je nezavisno promenljiva što znači da bi uporedjivanjem Z_1 i Z_2'' trebalo tu promenljivu odrediti tako da Z_2'' ima manju vrednost. U opštim uslovima to nije složen ali je glomazan matematički proračun te neće biti naveden u ovom tekstu. Uz uslov da je $Q/r_1 = r_1$, broj slogova u indeksiranoj grupi slogova u datoteci jednak je broju zapisa u svakom stablu, odrediće se takvo s za koje je $Z_2'' < Z_1$.

Uz navedeni uslov važi relacija:

$$Q = (s-1)^2 r^2 = h^n r$$

na osnovu koje se dobija nejednačina:

$$n(h+1) + r + 1 > (s-1)r + 3s/2 - 1/2$$

iz koje se na kraju dobija:

$$s < (2n(h+1) + 4r + 3)/(2r + 3)$$

4. VIŠE STABALA SA VIŠE NIVOA

Osnovna varijanta je više stabala sa više nivoa takvih da u okviru svakog stabla ne postoji presek (broj slogova jednak je nuli) izmedju indeksiranih grupa. Presek postoji samo kod indeksiranja u datoteci. Ovo znači da se indeksiranje sa više stabala sa jednim nivoom dalje nastavlja tako što se zapisi u svakom stablu indeksiraju jednim stablom sa više nivoa u kojem je presek indeksiranih grupa na jednom nivou prazan skup.

Problem više stabala sa više nivoa zahteva posebnu analizu pa se navode samo osnovne relacije za osnovnu varijantu:

$$M = s \sum_{j=1}^{n_1} \prod_{k=1}^{j} h_k \qquad ; \qquad h_k = h_j \quad \text{za} \quad k=j$$

$$r_1 = \lceil Q / \prod_{j=1}^{n_1} h_j \rceil \qquad ; \qquad \lceil r_1/r \rceil = s-1$$

5. ZAKLJUČAK

Indeksiranje u automatskoj obradi podataka predstavlja mehanizam pristupa i kao takvo vrlo je značajno. Organizacije podataka definisane sistemskim softverom (tu su uključena i indeksiranja) gube značaj kod složenijih zahteva za organizovanje podataka pa se mora pristupiti individualnom indeksiranju čime teorija indeksiranja postaje vrlo važna. Kao argument dovoljno je samo pomenuti organizovanje podataka preko sekundarnih ključeva koje u stvari predstavlja odredjeni skup indeksiranja.

Ne navodi se nikakva literatura jer je analiza u celini originalna.

4	8	12	16
a_4	a_8	a_{12}	a_{16}

2	6	10	14	16
a_2	a_6	a_{10}	a_{14}	a_{16}

$Z_2 = 7$

$Z_2' = Z_2' = 4$

$M = 9$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}

$a_i > a_{i-1}$

1': | 2 | 4 | / | a_2 | a_4 |

2': | 6 | 8 | / | a_6 | a_8 |

3': | 10 | 12 | / | a_{10} | a_{12} |

4': | 14 | 16 | / | a_{14} | a_{16} |

1'': | 1' | 2' | / | a_4 | a_8 |

2'': | 3' | 4' | / | a_{12} | a_{16} |

$M = 14$

$Z_1 = 6$

1''	2''
a_8	a_{16}

SLIKA 1.

INFORMATICA 3/85

IZVAJALNIKI ZA REALNI ČAS PRI MULTIPROCESORSKIH SISTEMIH

J. BERCE DIPL. ING.

UDK: 681.3.012

ISKRA – DELTA, PARMOVA 41

Arhitektura standardnih mikroprocesorjev je najbolj prilagojena potrebam enega programa z enim uporabnikom. Za bolj zahtevna izvajanja z več uporabniki in s tem tudi več programi je bila razvita široka paleta sistemov tako glede aparaturne kot programske opreme.

EXECUTIVES FOR REAL TIME MULTIPROCESSOR SYSTEMS

The architecture of standard microprocessor is best suited to single user, single task operation. For more demanding multi - user, multi - tasking operation a variety of systems design solutions have been proposed involving both HW and SW.

1. UVOD

Napredek pri razvoju elektronskih vezij in čedalje večja integracija elementov ter vedno hitrejša preklopna vezja, ob zmeraj boljših pomnilniških elementih, ponuja bolj in bolj obsežne sisteme, ki zahtevajo manjše napore pri izdelavi aparaturne opreme (AO).

Nizke cene mikroprocesorjev omogočajo drugačen pristop pri realizaciji doslej le rezerviranim tvorcem mikroračunalnikov in ne le minimizacijo po Von Neumannovem principu. Nove arhitekture uporabljajo dodatno AO, da bi poudarile velikokrat spregledane vidike, kot so: vzporednost (parallelism), zanesljivost (realiability), "neobčutljivost na okvare" (fault-tolerant). Večračunalniški sistem omogoča rešitev teh problemov, saj se pomnožene sposobnosti opravljanja nalog približujejo vzporednemu poteku delovanja. Hkrati zaradi same narave sistema, omogočajo zanesljive in na okvare malo občutljive izvedbe.

Seveda pa takšni sistemi zahtevajo drugačen pristop pri izdelavi, saj morajo biti prav tako uporabni pri povezavi z enim, kot tudi z več računalniki za opravljanje nalog.

Če kot primer vzamemo robota, pri katerem ima roka več stopenj svobode ter vsaka os svoj servo motor, bi integrirani krmilnik s svojim hitrim zaporednim delovanjem, lahko ustvaril vtis o sočasnosti gibov. Program sam bi bil lahko napisan kot vsota se ponavljajočih manjših podprogramov. Navidezna konkurenčna izvedba, bi zahtevala od eno-procesorskega računalnika, podporo pri več programskem delovanju, ki ga daje izvedba z izvrševalcem za realni čas (real time executive-RTX). Dinamičen razpored izvajalnih programov - IP (task) se izvaja glede na njihovo prednost. Procesi, ki so pripravljeni za izvajanje, se uvrstijo v čakalno vrsto, da bi se v trenutku, ko pridejo na vrsto izvršili. Tako se lahko hkrati izvaja tudi več neodvisnih izvajalnih programov, kar pa še ne pomeni, da se izvajajo sočasno.

Sočasnost je namreč določitev za več procesorski sistem (več kot en procesor na vodilu) in ne za več programsko neodvisni enoprocesorski sistem.

Vsakdo, ki je že sestavljal sistem z vzajemnim delovanjem v strojni kodi, bo priznal kakšen napredek imajo takšni mikroprocesorski izvajalniki. Toda žal tudi najhitrejši procesorji niso dovolj hitri (število izvršenih instrukcij v časovni enoti), da bi zadovoljili vse potrebe uporabnikov, zaradi česar so se začeli uveljavljati več procesorski sistemi z izvajalniki za realni čas.

2. IZVAJALNIKI ZA REALNI CAS ZA MIKROPROCESORJE

Sistemi za realni čas, so lahko razloženi kot skupek samostojnih združenih izvajalnih programov, ki so odgovorni za posamezno ali skupinsko povezano odvisno spremenljivko. Zato, da oskrbuje in nadzoruje več kot eno zunanjo napravo, mora biti nadzorni sistem sposoben voditi vzporedne programe, kot sodelujoče. Z enim procesorjem je resnično hkratno sodelovanje nemogoče, kajti procesor je sposoben izvršiti naenkrat le en ukaz. Zato mora obstajati nekakšen mehanizem za dodeljevanje procesorskega časa programom ter hkrati nadzorovati izvedbo in časovno skladnost.

2.1. OSNOVE DELOVANJA IZVAJALNIKA (EXECUTIVE)

Slika 2.1. prikazuje osnovno shemo za delovanje izvajalnika za realni čas za katerega obstajajo naslednje osnovne funkcije:

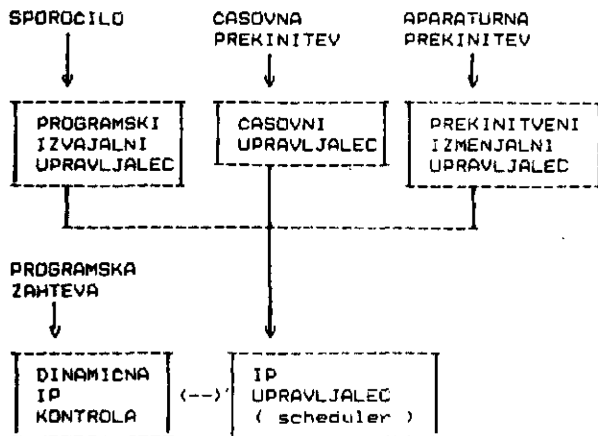
inicializacija programa in ponovno razvrščanje (nadzor nad skladom),

medprogramska povezava in časovna skladnost delovanja,

takt,

vodenje prekinitev,

spremenljiv nadzor nad programi.



Slika 2.1. Shema delovanja izvajalnika za realni čas

2.1.1. Začenjanje programov in ponovno razvrščanje programov

Vsak program ima svojo prednost (priority) pri izvajanju ter v vsakem trenutku določeno stanje, ki ga nadzira izvajalnik kar je prikazano v sliki 2.2. Dolžnost izvajalnika je določanje začetka, poteka in delovanja programov, glede na dogodke, ki se izvršijo.

Pripravljeni (ready) program je tisti, ki bi se izvršil takoj, ko bodo izvršeni vsi programi z višjim prvenstvom.

Čakajoči (waiting) program je tisti, ki potrebuje zunanji dogodek, da ga obudi in postavi v vrsto pripravljenih programov.

Program, ki se izvršuje (running) ima trenutno najvišjo prvenstvo med pripravljenimi programi.

Začasno odstranjen (suspended) program je tisti, ki ne čaka na zunanji dogodek - prav tako ni pripravljen in se ne poteguje za delovnje.

Zaključen (terminated) program je tisti, ki se je izvršil oziroma opravil.

2.1.2. Medprogramska povezava in časovna skladnost

Izvajalnik skrbi za komunikacijo med programi preko sporočil, prenosa podatkov ter nadzornim mehanizmom. Sporočilo samo je poslano preko kazalcev, ne pa ob morebitni zahtevi programa od izvajalnika, da mu omogoči pozitivni (send message primitive) osnovni znak (primitive). Način sam zmanjšuje neizkoriščenost, vendar pa zahteva določeno stopnjo reda: poslano sporočilo se ne sme preoblikovati oziroma spremeniti, dokler ga ni sprejemnik potrdil. Tudi način, ko je sporočilo poslano točno določenemu sprejemnemu programu, ni preveč zaželjen. Uporablja se način, ko se sporočila uvrščajo po pravilu FIFO (first in first out - prvi vstopaš prvi izstopaš) in si jih v vrstnem redu programi posredujejo (dodeljujejo).

Med programska sporočila lahko zato razdelimo v dve skupini (navidezna sorodnost z vhodno/izhodnimi podatki, zastavicami (flag)) :

sporočilo pri katerem je najbolj pomembna vsebina le tega in ne tako pomembna prisotnost,

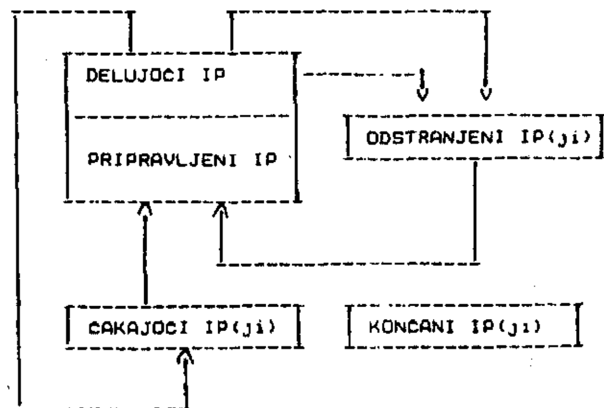
sporočilo katerega prisotnost je najpomebnejše, ker odloča o izvršitvi oziroma pravilni delitvi nekega dogotka ali podatkov glede na pomembnost - čas dostopa oziromo uporabe.

2.1.3. Takt (TIMNING)

Aparaturni takt (clock interval) je uporabljen za vodenje in razvrščanje ter "enakomerno" porazdeljevanje centralno procesne enote-CPE vsem programom v sistemu za realni čas zaradi:

programske časovne zanke so onemogočene s prekinitvami,

programi z višjo prednostjo, ki so v stanju zaseden in čakajoč (busy - waiting), bi onemogočali izvajanje ostalih programov, ki bi se med tem lahko izvajali, zato potrebuje izvajalnik enakomerne prekinitve (timeout) za pravilno dodelovanje CPE programom. Vendar morajo biti časovni razmiki za "aplikacijske" programe čim krajši zaradi natančnosti časovnega poteka. Del razmika uporabi izvajalnik za svoje delovanje tako, da je normalni obseg časovnih razmikov med 10-100ms.



Slika 2.2. Stanja izvajalnih programov (IP)

2.1.4. Vodenje prekinitev

Prekinitve so nepričakovane za izvajalnik, ki jih uvršča v navidezno vhodno/izhodno pozivno vrsto, da bi jih nato progam obdelal kot ostale pozive (le-ta ima lahko posebne prekinitvene znake / znamenja). Slaba stran takšnega načina obdelave prekinitev je v tem, da mora

izvajalnik shraniti "vsebino" prekinjenega programa in uvrstiti prekinitveni program. Veliko bolje je, če obidemo izvajalnik tako, da uvedemo kratke programe, ki obdelajo mimo njega prekinitev ter mu nato posredujejo potrditev obdelave prekinitve. S tem združimo nesočasne dogodke v povezano zgradbo in ne le "kar tako dodane" v izvajalnik.

2.1.5. Spremenljiv nadzor nad programi

V zahtevnem sistemu je ugodno, če lahko nadzorni programi nadzirajo ostale programe, saj je program lahko:

ustvarjen (spoznan izvajalniku in vključen v listo pripravljenih,

začasno odstavljen,

ponovno začet,

odstavljen (dokončno odstranjen za izvajanje).

Programi, ki se izvršujejo lahko spreminjajo sestavo sporočila, tako da ustvarjajo ali ukinjajo izmenjave. Spremenljivi nadzor se največ uporablja, tako da je dodana uporabniškim programom "funkcijska" knjižnica. Spremenljivo se ustvarjajo oziroma ukinjajo izmenjave med programi, ki jih knjižnica vsebuje. Uporabnik določi le vodilni program v svojem sistemu.

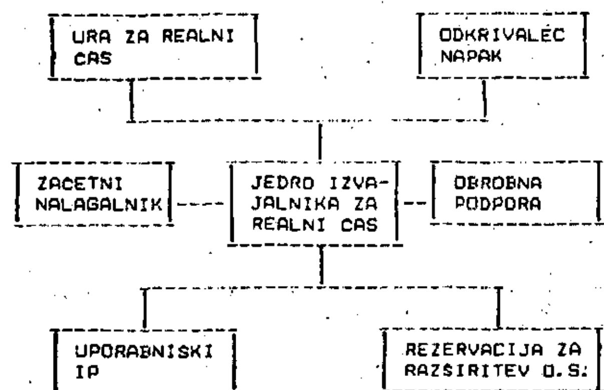
2.2. DRUGOTNE FUNKCIJE IZVAJALNIKA

Drugotne funkcije izvajalnika so vgrajene kot izvajalni programi, ki jih kliče uporabnik.

Na tržišču so izvajalniki, ki nudijo od osnovne zbirke programov, s katerimi lahko izvajamo razvojni ali aplikacijski sistem oz. oba, kar prikazuje slika 2.3., do zahtevnejših z večjimi sposobnostmi izvajanj.

3. VEČRAČUNALNIŠKI IN VEČPROCESORSKI SISTEMI

Obstoja veliko večračunalniških razporeditev, ki nudijo vzporednost in sočasnost pri izvajanju, vendar to niso večprocesorski sistemi. Za slednje je značilno, da obstaja

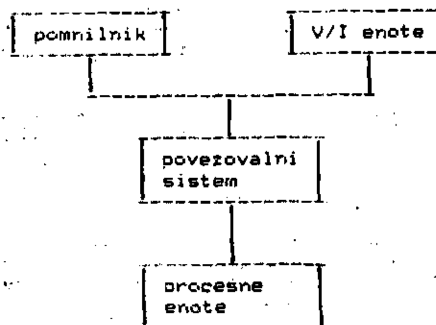


Slika 2.3. Razdelitev izvajalnika za realni čas

delitev skupnega pomnilniškeg prostora med vsemi procesorji in delitev vhodno/izhodnih enot med vsemi pomnilniškimi in procesorskimi sestavi.

Pravila za aparaturne in programske povezave v pravi večprocesorski sistem lahko strnemo v:

večprocesorski sklop vsebuje dva ali več procesorjev s približno primerljivimi sposobnostmi,



Slika 3.1. Razdelitev aparaturnih enot in opreme pri večprocesorskem sistemu

vsi procesorji si dele vse dostope vhodno/izhodnih kanalov, nadzorne enote in priključene naprave,

cel sistem nadzira samo operacijski sistem, ki oskrbuje vzajemno delovanje med procesorji in njihovimi programi pri delu, korakih, podatkih in osnovnih podatkovnih nivojih.

3.1. ORGANIZACIJA APARATURNE OPREME VEČPROCESORSKEGA SISTEMA

Po Enslow-u obstajajo tri, v osnovni različne razporeditve za "pravi večprocesorski" sklop: časovno razdeljivo/skupno vodilo (Time shared/common bus),

pregradni preklopnik (Crossbar switch),

večvratni pomnilnik (Multiport memory).

Čeravno obstajajo tudi sistemi, ki dosegajo vzporednost po drugi poti:

nesimetrični ali neenoviti (Asymmetrical or non-homogenus) sistemi,

razvrščeni ali usmerjeni procesorji (Array or vector processor),

kanalizirani procesorji (Pippeline processor),

sistemi neobčutljivi na okvare (Fault-tolerant systems),

združljivi procesorji (Associative processors).

3.1.1. Časovno razdeljivo/skupno vodilo

Pri skupnem vodilu je le-to večinoma povsem pasivno. Nanj se vežejo ostali deli sistema, kar pomeni, da ne obstajajo stikala in tudi ne ojačevalniki. Prenos opravil opravlja izključeno le vmesniško vodilo (bus interface) na sprejemni in oddajni strani. Torej mora sprejemnik poznati le svoj naslov ter odgovarjati na nadzor oddajnika.

Zaradi pomanjkljivosti takšnega sistema, so kasneje dodali še dodatna vodila, kar je povzročilo, da vodilo sedaj potrebuje tudi stikala, logiko in ostale nadzorne funkcije in s tem ni več popolnoma pasivno.

3.1.2. Crossbar stikalo

Z večanjem števila vodil, je prišlo do tega, da ima vsaka pomnilniška enota svojo povezavo. To je že osnova nezapornega crossbar stikala, ko je število prenosov omejeno s pomnilniškimi enotami in ne s sposobnostjo stikal. Vsako križišče (cross - point) mora biti sposobno, ne le preklapljati celotno vzporedno oddajo, ampak se tudi odločati pri večjem številu zahtevkov za posege v isto pomnilniško enoto, ki se pojavijo v enem samem pomnilniškem ciklu.

Kot naravni tok, so se v nadaljevnju crossbar stikala začela uporabljati tudi pri povezavi vhodno/izhodnih enot, po istih pravilih.

3.1.3. Pomnilnik z več vrati

Če nadaljujemo in nadzor ter preklopno logiko razdeljeno v crossbar stikalni mreži, združimo v pomnilniške enote, dobimo večvratni pomnilnik. Da rešimo stalno nesoglasje pri posegih v pomnilnik, določimo prednostno vrsto za dostop do pomnilniških vrat. Ker so vrata pač vrsta enakih električnih vodnikov je vseeno ali priključimo nanje vhodno/izhodne enote ali procesorje. Tak način zato tudi dopušča, da obstojajo zasebne računalniške enote, vezane na procesorje oziroma na vhodno/izhodne enote, kar povečuje varnost pri shranjevanju podatkov.

3.2. PROGRAMSKE ZAHTEVE PRI VEČRAČUNALNIŠKIH IN VEČPROCESORSKIH SISTEMIH

Večuporabniški sklop potrebuje določene zahteve kot so:

pogovorno neodvisno usmerjeno in istočasno delovanje večjega števila uporabnikov,

vzporednost pri delovanju in programih,

skupne in zasebne vire,

modularnost in aparaturne ter programske razširitve,

povezava z zunanjimi enotami in napravami,

krajevno omejena razdelitev aparaturnih enot,

visoka zanesljivost, uporabnost in neobčutljivost na okvare.

Tem zahtevam odgovarja skupek mikroprocesorjev združenih v prosto povezanih (loosely coupled) in porazdeljenih (distributed) sistemih, ko so sposobni tudi neodvisnega delovanja. Program se lahko odvija v enem samem sklopu ali pa se dodeli kateremukoli prostemu sklopu. Nezasedeni sklopi uspešno vrše nadzor nad izvajanjem. Operacijski sistem mora podpirati decentralizirano delovanje vsakega računalniškega sklopa. Tako vsebuje vsak tak računalniški sklop, svoj podsistem, ki je sposoben občevati z ostalimi. Dosežena je vzporednost izvajanja dela, saj lahko vsak uporabnik zažene enega ali več del na enem ali več računalniških sklopih.

3.2.1. Programska povezava sklopov

Komunikacija med porazdeljenim računalniškim sistemom mora izpolnjevati:

izvrševanje vsebine operacijskega sistema (sistemski pozivi),

skrb za poljubne razsodniške poteke pri komunikaciji z vzajemnim soorazumevnjem.

Zaradi tega, ker je pozivna operacija lahko zaključena, še predno sprejemna odgovori na poziv, je s tem dosežena neodvisnost. Sprejemna operacija določa, kakšna vrsta sporočil je dovoljena in s sprejemom le-tega se tudi zaključi. Doseženo je, da se programi lahko dodeljujejo različnim mikroračunalniškim sklopom dinamično, oziroma statično. V kolikor upoštevamo namen, da se ne uporabljajo centralni elementi, pridemo do tega, da nimamo pri dodeljevanju centralne čakalne vrste, ampak je zamenjana v razširitveno funkcijo.

Komunikacija med procesi se izvaja preko manjših podatkovnih paketov ali sporočil, ki potujejo preko logičnih povezav naslovljenih kot vrata. Vrata se "ustvarijo" na začetku dela ter se jim dodelijo standarndne povezave s sistemskim procesom. Vendar pa le-ta ni obremenjen z njimi drugače kakor preko kataloga.

3.2.2. Časovno kritični sklopi

Današnje aplikacije za realni čas z mikroračunalniki zahtevajo tja do stotine MIPS (milion instruction par second), kar zahteva čim krajši sistemski odziv, ki ga nudijo vzajemno delujoči napredno porazdeljeni sistemi. Vendar brez dobrih programskih orodij, bi večanje mreženja mikroprocesorjev znižalo sistemsko propustnost in zmanjšalo odzivni čas sistema, zaradi neuravnoteženosti uporabe mikroprocesorjev (večanje veriženja programov in s tem pogojeno čakanje na izvajanje) in povečanja medprocesorskih povezav (prenos velikega števila podatkov med programi dodeljenimi različnim procesorjem, povečuje njihovo vzajemno komunikacijo). Zaradi tega zahtevajo porazdeljeni povezani mikroračunalniški aplikacijski sistemi sledeče funkcije :

močno povezanost sistemov (cenovno-učinkovita minimalna mikroprocesor/pomnilniška modularna komunikacija),

aplikacijsko opisni jezik, ki se najbolj približuje danim zahtevam aplikacije, in

programsko razvojno orodje, ki združuje in dodeljuje predhodno opisana pogoja.

Postopek imenovan izvajalno programsko razdeljen model TAM (task allocation model) - "vejalno in skakalno" heuretična metoda, je razvojno programsko orodje, ki se uporablja pri porazdeljenih mikroračunalniških sistemih in omogoča sledeče funkcije :

minimizira medprocesorske komunikacijske okoliščine,

uravnotežuje uporabnost vsakega mikroprocesorja in

je primerno inženirsko programsko aplikacijsko orodje.

V časovno kritičnih sklopih (TCA - time critical application) so programi urejeni v izvajalnih "vrstah", ki morajo doseči "port-to-port" (vrata do vrata) časovne zahteve. Zato je PTP določen kot celoten čas izvajanja v "vrstah" pri sklopu vnaprej določenih časovnih omejitvah. Torej dodelitev programa izvajalnemu mikroprocesorju, mora najprej zagotoviti te PTP časovne zahteve, da program smatramo za dodeljen. Pri čemer ima pravilno dodeljevanje mikroprocesorjem močan vpliv (število "omogočitev" programa, povezovalni pogoji med programi in dolžina programov so osnovni pogoji za željene informacije o programu).

Zaradi porazdeljene mreže je potrebno opisati še tri osnovne PTP dejavnike :

izvajalni čas programa, je določen z dolžino programa (strojne instrukcije) in mikroprocesorsko hitrostjo izvajanja (merjeno v MIPS),

zakasnitve pri vrstah, se pojavijo ko se več kakor en program izvaja v danem mikroprocesorju in mora drugi program čakati na izvajanje. Določene so s številom programov, njihovo dolžino ter števila "omogočanj" izvajanja programa,

medprocesorsko komunikacijski čas, ki se pojavi, ko si morajo nameščeni programi v mikroprocesorjih med seboj izmenjavati podatke. Zavisi od povezovalnega pogoja (število prenešenih besed) med izvajalnimi programi in medprocesorskih "prometnih" okoliščin.

Glede na vsa navedena dejstva moramo zagotoviti za PTP zahtevke sledeče pogoje :

zmanjšanje izvajalnega časa programa izvedemo tako, da "dolge" programe prenesemo v "najmočnejši" mikroprocesor,

zmanjšanje "zakasnitve v vrstah" izvedemo s tem, da programe z "dolgimi instrukcijami" in pogosto izvajane programe razdelimo med različne mikroprocesorje,

medprocesorski komunikacijski čas zmanjšamo s tem, da močno povezane programe izvajamo v istem mikroprocesorju.

Pri čemer se moramo opreti na aparaturno opremo (izbira najboljše AO konfiguracije), dodeljevalni nivo (določimo poti za časovno kritične poteke, "ostali naj počakajo"), programski nivo (visok nivo prvenstva za prekinitvene programe z njihovim vklapljanjem) in prevajalni del (omogočiti in izbrati instrukcije z najkrajšim izvajalnim časom).

4. Z A K L J U Č E K

Počasi se končuje "obdobje" eno procesnih mikroračunalnikov na večini polj, saj pomnoženi procesorji cenejo dosežejo boljše karakteristike in ostale tako zahtevane lastnosti pri uporabi mikroračunalnikov. Tako so sistemi z večprocesorji in z večračunalniki čedalje cenejši in varnejši za delovanje in shranjevanje podatkov. Tudi komunikacija s človekom in ostalimi enotami, ki so bila ozka grla za osrednji sistem, se s tem olajša (predvsem pri vedno bolj inteligentnih terminalih).

Da lahko razvojni inženir čim bolj izkoristi prednosti več mikroračunalniškega sistema, ne sme privzeti prvotno oblikovanega sistema, temveč mora biti svoboden predvsem glede aparaturne opreme; s pogojem, da mu pri tem ni potrebno popolnoma preoblikovati napisanih programov, temveč so (vhodno/izhodna) vrata le sistemske spremenljivke. Na ta način smo pri aplikaciji ločili določitve programov (algoritme - poteke, podatke itd) od aparaturne opreme in lahko s programsko opremo predvidimo aparaturne zahtevke. Tako je osnovna zahtevana programska oprema - za več računalniško (kakor tudi eno računalniško), pri razvijanju razdeljena na :

programirni jezik s prevajalnikom,

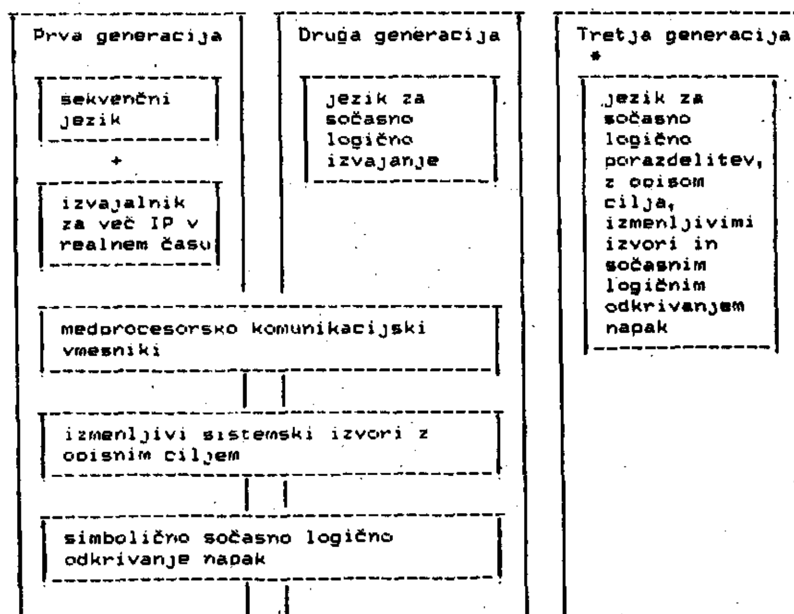
možnost programskega določanja aparaturne opreme,

izvore dodeljevanja pri mrežni obliki,

izbira in določanje izvajalnega časa sistemske konfiguracije,

testno sledenje programom (debugging) in

kontrola izvajanja.



Slika 4.1. Razvojne generacije za programsko opremo

Kar lahko strnemo v zbir orodij, katere naj večračunalniški sistem podpira za razvoj aplikacijskega paketa programov :

visoko-nivojski zaporedno izvajalni jezik,

sočasno logično izvajanje (concurrency),

porazdeljeno dodeljevanje in

V/I obdelave.

Slika 4.1. prikazuje razvojne generacije glede na zgoraj navedene zahtevke, ki jih je programska oprema prešla s tem, da se čim bolj "osvobodi" odvisnosti aparaturnih zahtevkov ter je s tem večnamenska in lahko prenosljiva. Prav tako, je aplikacija že v razvojnem delu samostojen "izdelek" in ni potrebno prilagajati odvisnost med "operacijsko sistemskim delom" in aplikacijo, ter omogoča enostavnejše testiranje, nameščanje in kontroliranje.

LITERATURA:

1. A Model to Solve Timming-Critical Application Problems in Distributed Computer Systems, R.Perny-Yi Ma, " Computer ", jan. 1984;

2. Design and implementation of fault-tolerant multimicrocomputer systems, D.Bernhard and E.Schmitter, "Microprocessors and Microsystems", maj 1981;

3. Functional architecture, J. McGrath and P.Strzelecki, "SI", jan. 1982;

4. Introduction to Multiprogramming, M. Dahmke, "Byte", sept. 1979;

5. Multiprocessors and other parallel systems : an introduction and overview, P. H. Enslow, "Infotech State of the Art Report", 1976;

6. Multiple - Microprocessor Programming Techniques : MML, a New Set of Tools, M.Boari, S.C-Reghizzi, A.Dapra, F.Maderna, A.Natali, " Computer ", jan. 1984;

7. Process communication within a distributed multimicrocomputer system, F. Eser and F. Schmidtke, "Microprocessors and Microsystems", maj 1981;

8. Real-time executives for microprocessors, F. von der Linden and I. Wilson, "Microprocessors and Microsystems", jul./avg. 1980.

BIBLIOGRAFSKI MIKRORAČUNALNIŠKI SISTEM ZA PREISKOVANJE POVZETKOV

JANEZ DIVJAK ZALOKAR

VISOKE VOJNE TEHNIČKE ŠKOLE, ZAGREB

UDK: 681.3:011/016

Racunalniski bibliografski sistemi omogocajo sirok spekter najrazlicnejsih operacij nad bazami podatkov o publikacijah. V clanku je opisana ideja ter navodila za realizacijo manjsega sistema na mikroracunalniku, namenjenega predvsem iskanju povzetkov s pomocjo kljucnih besed povezanih s podatki o publikaciji. Prikazane so tudi osnovne strukture podatkov, realizirani algoritmi in okvirna zmogljivost takega sistema.

Kljucne besede : povzetek, bibliografija, kljucna beseda, mikroracunalnik

BIBLIOGRAPHIC MICROCOMPUTER SYSTEM FOR ABSTRACTS RETRIEVAL. Bibliographic computer systems offers wide variety of operations over publications data bases. The main idea of smaller system mainly dedicated for use in abstract retrieval, based on keywords assigned to publication data, and instructions for the implementation on microcomputer system are described in this article. Fundamental implemented algorithms, data structures and global capabilities of such a system are also given.

Keywords : abstract, bibliography, keyword, microcomputer

I. U V O D

Zagotovo vsaka ustanova ali delovna organizacija razpolaga z manjso ali vecjo kolicino strokovne literature (knjig in revij). Pregled nad kolicino in vsebino najveckrat ni tako enostavno realizirati. Najelegantnejsa resitev je vsekakor racunalnisko cuvanje in preiskovanje osnovnih podatkov o obstojecih publikacijah. Med osnovne podatke sodijo naslov, podatki o avtorju in zalozbi, leto izdaje, interna klasifikacijska koda in krajsi povzetek vsebine. Kadar pa zelimo kar se da hitro poiskati vse publikacije z doloceno tematiko, je potrebno osnovne podatke povezati s tako imenovanimi kljucnimi besedami (v dalnjem tekstu "kljuci"), ki specificirajo vsebino vsake publikacije, ter na njihovi osnovi vspostaviti sistem za ucinkovito iskanje opisov publikacij (v dalnjem tekstu "zapisov") s presekom zahtevanih kljucev. V tem clanku ne bomo obravnavali ostale aspekte vecjih bibliografskih informacijskih sistemov, temvec se bomo predvsem zadrzali v okvirih moznosti, ki jih nudi mikroracunalnik. Osnovni cilj nam bo izgradnja sistema, ki bo sposoben v najkrajsem casu odgovoriti na vprasanje : "Katere publikacije govore o doloceni tematiki?".

II. IZBIRA RACUNALNIKA IN JEZIKA

Relativno enostavno je mogoce vspostaviti majhen toda efikasen bibliografski informacijski sistem tudi na mikroracunalniku z vsaj eno disketno enoto, malo vecjim pomnilnikom (vsaj 50K znakov) in programskim jezikom, ki omogoce direktno pozicioniranje na podatke v datotekah.

Vsekakor daje programiranje v strojnem jeziku obicajno najboljse rezultate glede hitrosti in ekonomicne uporabe pomnilnika, vendar so dandanes le redki entuzijasti pripravljeni zgraditi informacijski sistem samo s pomocjo nekega asemblerja.

Zato se bomo odlocili za visji programski jezik, pri katerem bo posebno izrazena sposobnost manipuliranja s sestavljenimi strukturami podatkov, nizi znakov in moznost logicnih operacij nad polji bitov (to so osnovne operacije v racunanju unij in presekov mnozic kljucev ter zapisov). Eden od najprimernejsih jezikov, ki jih srecamo tudi na mikroracunalnikih, je PASCAL s svojimi strukturami RECORD, ARRAY OF CHARS in SET. V skrajnem slucaju pa je mogoce uporabiti tudi prevajalnike FORTRAN ali BASIC z moznostjo klicev strojnih podprogramov.

V dalnjem tekstu opisani algoritem je implementiran na mikroracunalniskem sistemu CDC 110 VIKING zasnovanem na procesorju Z80, z operativnim sistemom CP/M 2.2 s 53K znakov uporabniskega pomnilnika ter v konfiguraciji z disketno enoto za 1,2M znakov in graficnim tiskalnikom. Program je napisan v jeziku PASCAL/M (malo egzotike firme SORCIM). Disketa zadosca za cuvanje tabele 1000 razlicnih kljucev dolzine 15 znakov ter 1000 zapisov z osnovnimi podatki (4*70 znakov), povzetkom (8*70 znakov), letnico in poljubno kombinacijo povezav kljucev s zapisom. Varijanta brez povzetkov omogoca cuvanje 2000 zapisov, kar je hkrati tudi meja za implementacijo danega algoritma na tem mikroracunalniskem sistemu.

III. IDEJNA RESITEV

Osnovni problem mikroracunalniske implementacije nekoliko bolj zapletenega algoritma, ki manipulira z veliko mnozico podatkov, sta hitrost in omejen pomnilnik. Vedno se postavlja vprasanje, kaj cuvati v pomnilniku, kaj pa na zunanjih enotah, da hitrost ne bi bila bistveno zmanjsana. Kompromis ni zmeraj ravno lahko najti. Velikokrat je pravzaprav vse odvisno od osnovne ideje algoritma, ki je v nasem primeru sledeca:

Pri iskanju mnozic kljucev povezanih z zapisi podatkov je najfrekventnejsi pristop do samih kljucev ter podatkov o vezi kljucev z zapisi. Zato jih bomo cuvali loceno od samih zapisov: tabelo kljucev v pomnilniku, zveze pa v obliki polj bitov (SET OF 1..nkey, kjer vsak clen pove, ce je kljuc iz tabele povezan z zapisom ali ne) katere bomo po potrebi nalozili v pomnilnik z diskete v blokih. Pokazalo se je, da blok z zvezami za 100 zapisov podatkov omogoca se dovolj hitro iskanje (vec niti ni bilo mogoce istocasno cuvati v pomnilniku). V skrajnem primeru bi bilo mogoce razdeliti na vec blokov tudi tabelo kljucev, seveda na racun hitrosti iskanja.

Drugi problem se javlja, ko se je potrebno odlociti o nacinu cuvanja kljucev. Lahko bi jih cuvali tako, kot so vneseni za vsak zapis, kar bi omogocilo mgzaktno kasnejse iskanje zapisov. Na zalost pa bi se v tabeli kopicili kljuci z istim pomenom a razlicnimi prefiksi (ednina, mnozina, varijante istih pridevnikov in samostalnikov), zato bi bilo potrebno iskati zapise z unijo vseh varijant kljuca. Zato se lahko odlocimo za izbor sistema, pri katerem vhodni kljuci (ki se vezejo na zapise ali po katerih iscemo zapise) predstavljajo podnize ("podkljuce") ze vnesenih kljucev v tabeli. Vhodni kljuc se vnese v tabelo le, ce ne obstaja kot podniz vsaj enega od ze obstojecih. Prav tako se obstojeci kljuc zamenja z vhodnim ce predstavlja njegov podniz. Pri iskanju se poiscejo vsi zapisi, ki so povezani s kljuci, kateri vsebujejo v sebi vhodni podkljuc. Tak sistem sicer lahko pripelje do zanimivih logicnih napak v iskanju (vhodni podkljuc "VOD" privlece na plan zapise povezane s kljuci "VODOVOD", "SPREVOD" ali "NAVODILO"...), vendar je pri pazljivi organizaciji kljucev ucinkovitejsi kot sistem fiksnih kljucev.

Ponazorimo ta sistem s primerom :

V prazno bazo povzetkov pravljic vnesimo 4 zapise ter vsak zapis povezimo s sledecimi podkljuci-

1. zapis : DEKLICA, VOLK, GOZD
   (vsi kljuci se vnesejo v tabelo)

2. zapis : METKA, JANKO, CARATI, GOZD
   ("GOZD" se ze nahaja v tabeli)

3. zapis : SNEGULJCICA, ZACARATI, GOZDOVI
   ("CARATI" se zamenja z "ZACARATI",
   "GOZD" se zamenja z "GOZDOVI")

4. zapis : SNEG, VZIGALICE, DEKLICA
   ("SNEG" se ze nahaja v "SNEGULJCICA",
   "DEKLICA" se ze nahaja v tabeli)

Sedaj imamo v tabeli kljuce : DEKLICA, VOLK, GOZDOVI, SNEGULJCICA, METKA, JANKO, ZACARATI in VZIGALICE.

Pri iskanju bosta za vhodni podkljuc "CAR" prikazana drugi in tretji zapis, za "SNEGULJ" tretji in cetrti zapis, za "GOZD" pa prvi, drugi in tretji....

Tretji problem pri oblikovanju algoritma je odlocitev o nacinu iskanja zapisov. Nekateri obstojeci sistemi uporabljajo metodo formiranja logicnih izrazov v katerih so kljuci povezani z logicnimi operatorji. Seveda je za vas prevajalnik takih izrazov dovolj kompliciran, da se bomo (iz majhne lenobe) odlocili za poenostavljeno metodo, ki jo lahko opisemo na naslednji nacin:

a) predpostavi, da so vsi zapisi kandidati za ispis

b) precitaj skupino alternativnih podkljucev (za vsak podkljuc poisci v tabeli vse pripadajoce kljuce, poisci zapise, ki so povezani z vsaj enim od teh kljucev in doloci njihov presek z dosedanjimi kandidati)

c) ponavljaj (b) za vse skupine, s cimer je pravzaprav izvrsen naslednji (in najpogostejsi) logicni izraz:

(S) := (S1) in (S2) in (S3) ....

kjer so :

(Si) - mnozice kandidatov v skupini
(Si) := (ki1) ali (ki2) ali (ki3) ....

(kij) - mnozice zapisov povezanih z vhodnimi podkljuci kij

in, ali - logicni operatorji
i, j - indeksi

d) ispisi vse preostale kandidate iz (S)

Ostali problemi so vec ali manj estetske narave. Verjetno bi zeleli najti dolocene zapise na podoben nacin tudi za druge osnovne podatke v bazi. Vendar to ni realna zahteva. Vse kar se lahko naredimo je to, da prikljucimo veznemu zapisu kljucev se podatek o letnici izdaje publikacije in eventualno klasifikacijsko kodo (v kolikor jo definiramo z nekaj znaki) ter tako omogocimo hitro iskanje za ta dva parametra. Ostale podatke bomo morali iskati s pomocjo sekvencijalnega citanja baze in vzporejanja vsebine z vhodnim nizom znakov, kar pa je dokaj pocasen proces. Cas citanja lahko prepolovimo tako, da oddvojimo osnovne podatke od povzetka in jih cuvamo v posebni datoteki.

IV. REALIZACIJA

V tem delu bodo prikazane uporabljene strukture podatkov in neki najbolj kriticni algoritmi implementirani na navedenem mikroracunalniku.

Sistem ima vgrajene funkcije za:

- dodajenje in brisanje zapisov iz baze
- azuriranje podatkov in kljucev v zapisu
- iskanje zapisov po preseku skupin kljucev
- iskanje zapisov po intervalu publiciranja
- sekvencijalno iskanje zapisov po podnizih osnovnih podatkov in povzetka
- listanje sortirane tabele vnesenih kljucev
- listanje kljucev za dani podkljuc

Sistem cuva bazo vseh potrebnih podatkov na stirih datotekah:

- datoteka vseh kljucev (1 blok)
- datoteka osn. podatkov(1 blok= 1 zapis)
- datoteka povzetka (1 blok= 1 zapis)
- datoteka veznih blokov(1 blok= 100 zapisov)

Zadnje tri datoteke imajo moznost direktnega pozicioniranja zapisov. Njihova PASCAL-ska definicija je sledeca :

```
CONST nkey = 1000; -- maks. stevilo kljucev
      nrec = 1000; -- maks. stevilo zapisov
      nblk = 100;  -- stev. vez. zapisov bloka
      nlin = 8;    -- stev. linij povzetka
      linl = 70;   -- dolzina linij
      keyl = 15;   -- dolzina kljucev

TYPE
-- nestandardna definicija niza znakov
  STRING[nn] = PACKED ARRAY [1..nn] OF CHAR;

-- tipi kazalcev in letnice publiciranja
  RECPOINT = INTEGER; -- kazalci zapisov
  KEYPOINT = INTEGER; -- kazalci kljucev
  BLKPOINT = INTEGER; -- kazalci v bloku vez
  YEARS    = INTEGER; -- letnice

-- aktivni zapisi, kljuci in tabela kljucev
  LISTR = PACKED RECORD
    NRC : RECPOINT; -- stevilo zapisov
    NKY : KEYPOINT; -- stevilo kljucev
    KYS : ARRAY [1..nkey] OF STRING [keyl]
  END;

-- zapis osnovnih podatkov o publikaciji
  DATAR = PACKED RECORD
    TIT : STRING [linl]; -- naslov publik.
    AUT : STRING [linl]; -- avtor
    PUB : STRING [linl]; -- zalozba
    CLS : STRING [linl]  -- klasifikacija
  END;

-- zapis povzetka publikacije
  ABSTR = PACKED RECORD
    ABS : ARRAY [1..nlin] OF STRING [linl]
  END;

-- vezni blok kljucev in letnice z zapisi
  CONNR = PACKED RECORD
    CNT : ARRAY [1..nblk] OF SET OF 1..nkey;
    YEA : ARRAY [1..nblk] OF YEARS
  END;

VAR
  LIST : FILE OF LISTR; -- datoteka kljucev
  DATA : FILE OF DATAR; -- baza podatkov
  ABST : FILE OF ABSTR; -- baza povzetkov
  CONN : FILE OF CONNR; -- baza veznih blokov
  BLOK : BLKPOINT;      -- kazalec vez. bloka
  KEY  : STRING [keyl]  -- vhodni podkljuc
```

Skoraj vse procedure klicejo na pomoc funkcijo GETBLK, ki izracuna kazalec na vezni zapis v veznem bloku CONN ter ga po potrebi predhodno tudi prinese z diskete:

```
FUNCTION GETBLK( RECR : RECPOINT ) :BLKPOINT;

-- pripravi vezni blok in
-- izracuna kazalec na zapis RECR
-- (procedura SETNEXT pozicionira na blok)

VAR BL : BLKPOINT; -- kazalec na zaht. blok

BEGIN (*getblk*)

  BL := (RECR-1) DIV nblk; -- izracunaj blok
  IF BLOK<>BL THEN BEGIN   -- isti kot stari?
    BLOK := BL;            -- ne, pozicioniraj
    SETNEXT(CONN,BLOK);   -- in vzemi novi
    GET(CONN)              -- vezni blok
  END(*if*);
  GETBLK := RECR-BLOK*nblk; -- izracunaj kaz.
                            -- na vezni zapis
END(*getblk*);
```

Proceduro CLEARKEY uporablja procedura za brisanje zapisa iz baze, da bi se v tabeli kljucev izbrisali kljuci, ki so bili vezani samo z brisanim zapisom. Taksni kljuci se postavljajo na vrednost praznega niza znakov in so na voljo pri vnosu novih kljucev:

```
PROCEDURE CLEARKEY ( RECR : RECPOINT );

  -- brisanje kljucev pripadajocih
  -- samo zapisu RECR

VAR FLG : SET OF 1..nkey; -- lista zapisov
    I   : RECPOINT;       -- stevec zapisov
    J   : KEYPOINT;       -- stevec kljucev

BEGIN (*clearkey*)

  FLG := CONN^.CNT[GETBLK(RECR)];
  FOR I := 1 TO LIST^.NRC DO
    IF I<>RECR THEN
      FLG := FLG-(FLG*CONN^.CNT[GETBLK(I)]);

  IF FLG<>[] THEN
    FOR J := 1 TO LIST^.NKY DO
      IF J IN FLG THEN
        LIST^.KYS[I] := '';

END(*clearkey*);
```

Procedura SETKEY sluzi za povezovanje zapisa z vsemi kljuci ki imajo v sebi vkljucen dani vhodni podkljuc. V kolikor ne obstaja niti eden taksen, se vhodni kljuc doda na prosto mesto v tabeli (na izpraznjeno pri brisanju nekega drugega zapisa ali pa na konec tabele):

```
PROCEDURE SETKEY (     RECR  : RECPOINT;
                   VAR INKEY : BOLLEAN   );

-- Uvrscanje kljuca KEY v tabelo kljucev
-- in v vezni zapis RECR ter vracanje
-- statusa INKEY o vnesenem kljucu
-- (POS vraca pozicijo prvega kljuca v drugem)

VAR I : KEYPOINT; -- stevec kljucev
    J : KEYPOINT; -- kazalec na prosti kljuc
    K : BLKPOINT; -- kazalec na vezni zapis

BEGIN (*setkey*)

  -- iskanje podkljuca v tabeli
  INKEY := FALSE;  J := 0;
  K := GETBLK(RECR);

  FOR I := 1 TO LIST^.NKY DO
    IF (LIST^.KYS[I]='') THEN
      J := I
    ELSE
      IF POS(KEY,LIST^.KYS[I])<>0 THEN BEGIN
        CONN^.CNT[K] := CONN^.CNT[K]+[I];
        INKEY := TRUE;
      END(*if*) ELSE
        IF POS(LIST^.KYS[I],KEY)<>0 THEN BEGIN
          LIST^.KYS[I] := KEY;
          CONN^.CNT[K] := CONN^.CNT[K]+[I];
          INKEY := TRUE;
        END(*if*);

  -- vnos kljuca v tabelo
  IF NOT INKEY THEN
    IF (J<>0) OR (LIST^.NKY<nkey) DO BEGIN
      IF J=0 THEN BEGIN
        J := SUCC(LIST^.NKY);
        LIST^.NKY := J;
      END(*if*);
      CONN^.CNT[K] := CONN^.CNT[K]+[J];
      LIST^.KYS[J] := KEY;
      INKEY := TRUE;
    END(*if*);

END(*setkey*);
```

Procedura FINDKEY vrsi kompletno iskanje preseka zapisov, ki vsebujejo vsaj en alternativni kljuc iz vsake dane skupine kljucev. V kolikor je presek prazna mnozica, se algoritem avtomatsko postavi na zacetne vrednosti. Prazen kljuc prekinja vnos clanov skupine ter skupin kljucev.

```
PROCEDURE FINDKEY;

--   iskanje   zapisov   s   presekom   skupin
podkljucev
-- (POS vraca pozicijo prvega kljuca v drugem)

VAR I     : RECPOINT;          -- stevec blokov
    J     : KEYPOINT;          -- stevec kljucev
    CNT   : INTEGER;           -- stevec zadetkov
    GROUP : INTEGER;           -- stevec skupin
    FLAG  : SET OF 1..nrec;    -- skupina zapisov
    FLG   : SET OF 1..nrec;    -- presek skupin
    KEYF  : SET OF 1..nkey;    -- skupina kljucev

BEGIN (*findkey*)

WRITELN;
WRITELN('ISKANJE PO KLJUCIH :');
GROUP := 0; KEY := ''; KEYF := [];

REPEAT

  -- vhod skupine podkljucev
  WHILE KEY<>'' DO BEGIN
    CNT := 0;
    FOR J := 1 TO LIST^.NKY DO
      IF POS(KEY,LIST^.KYS[J])<>0 THEN BEGIN
        CNT  := SUCC(CNT);
        KEYF := KEYF+[J]
      END(*if*);
    IF CNT=0 THEN WRITELN('Ne najdem kljuc!');
    WRITE('ali: '); READLN(KEY);
  END(*while*);

  IF KEYF<>[] THEN BEGIN

    -- iskanje zapisov z vsaj enim kljucem
    CNT  :=  0;
    FLAG := [];
    FOR I := 1 TO LIST^.NRC DO
      IF KEYF*CONN^.CNT[GETBLK(I)]<>[] THEN
        BEGIN
          CNT  := SUCC(CNT);
          FLAG := FLAG+[I]
        END(*if*);
    KEYF := [];
    WRITELN('Stevilo zadetkov : ',CNT);

    -- doloci dosedanji presek skupin
    IF GROUP=1 THEN FLG := FLAG
               ELSE FLG := FLG*FLAG;
    IF FLG=[]  THEN BEGIN
      GROUP := 0;
      WRITELN('Ni skupnih zadetkov!')
    END(*if*);

  END(*if*);

  -- pripravi naslednjo skupino
  GROUP := SUCC(GROUP);
  WRITELN;
  WRITELN ('Vnesi skupino kljucev !');
  WRITELN (GROUP,'.:'); READLN(KEY);
UNTIL KEY='';

-- prestej in ispisi zapise
IF GROUP>1 THEN BEGIN
  CNT := 0;
  FOR I := 1 TO LIST^.NRC DO
    IF I IN FLG THEN CNT := SUCC(CNT);
  WRITELN;
  WRITELN('Totalno zadetkov : ',CNT);
  FOR I := 1 TO LIST^.NRC DO
    IF I IN FLG THEN PISI(I);
END(*if*);
END(*findkeys*);
```

Procedura FINDYEAR ispise vse zapise, ki so bili izdani v doloceni casovni periodi. Vsi ostali podatki v bazah se iscejo sekvencijalno po zadanem nizu znakov.

```
PROCEDURE FINDYEAR;

  -- iskanje zapisov po letnici publikacije

VAR I    : RECPOINT;         -- stevec zapisov
    J, K : YEARS;            -- letnice interv.
    FLG  : SET OF 1..nrec;   -- najdeni zapisi

BEGIN (*findyear*)

  -- vhod mejnih letnic
  WRITELN('OD leta :'); READLN(J);
  WRITELN('DO leta :'); READLN(K);

  -- iskanje po vseh zapisih v tabeli veze
  FLG := [];
  FOR I := 1 TO LIST^.NRC DO
    IF (CONN^.YEA[GETBLK(I)]>=J) AND
       (CONN^.YEA[GETBLK(I)]<=K) THEN
         FLG := FLG+[I];

  -- ispis vseh najdenih zapisov
  IF FLG<>[] THEN
    FOR I := 1 TO LIST^.NRC DO
      IF I IN FLG THEN PISI(I);

END(*findyear*);
```

Procedura LISTKEY ispise sortirano tabelo kljucev. Ker se tabela tako unici, jo je potrebno po ispisu ponovno naloziti z diskete.

```
PROCEDURE LISTKEY;

-- Listanje sortirane tabele kljucev

VAR I : KEYPOINT;             -- kazalec kljucev

PROCEDURE SORT( L,R : INTEGER );

-- sortiranje tabele kljucev od L do R

VAR CHKEY : STRING[keyl]; -- pomozni kljuc
    I,J   : KEYPOINT;     -- kazalci

BEGIN (*sort*)

  I := L; J := R;
  KEY := LIST^.KYS[ (I+J) DIV 2 ];

  REPEAT
    WHILE (LIST^.KYS[I]<=KEY) DO
      I := SUCC(I);
    WHILE (LIST^.KYS[J]>=KEY) DO
      J := PRED(J);
    IF (I<=J) THEN BEGIN
      CHKEY          := LIST^.KYS[I];
      LIST^.KYS[I]   := LIST^.KYS[J];
      LIST^.KYS[J]   := CHKEY;
      I := SUCC(I); J := PRED(J);
    END(*if*);
  UNTIL (I>J);

  IF (J>L) THEN SORT(L,J);
  IF (I<R) THEN SORT(I,R);
END(*sort*);

BEGIN (*listkey*)

  SORT (1,LIST^.NKY);

  FOR I := 1 TO LIST^.NKY DO BEGIN
    IF (I MOD 4 = 1) THEN WRITELN;
    WRITE(LIST^.KYS[I]:keyl,' , ');
  END(*for*);

END (*listkey*);
```

V. ZAKLJUCEK

Realizirani algoritmi imajo pri ze omenjenih popolnjenih kapacitetah:

1000 razlicnih kljucev dolzine 15 znakov,
1000 zapisov s
4*70 znakov osnovnih podatkov,
8*70 znakov povzetka,
letnico zalozbe,
povezavo vsakega sloga z vsemi kljuci,

naslednjo povprecno casovno zmogljivost:

a) procedura FINDKEY

- iskanje vseh kljucev za dani podkljuc .... 4.5 sekunde
- iskanje zapisov z unijo kljucev v skupini .... 20 sekund
- presek dveh skupin kandidatov zapisov .... 1.5 sekunde

b) procedura FINDYEAR .... 14 sekund

c) procedura SETKEY .... 7.5 sekunde

d) brisanje sloga iz baze (CLEARKEY in ostale potrebne procedure).. 32 sekund

e) iskanje po podnizih teksta podatkov

- osnovni podatki ... 500 zapisov/minuto
- povzetki .... 290 zapisov/minuto

Vidimo, da smo dobili kar sprejemljive casovne zmogljivosti, katere omogocajo solidno interaktivno preiskovanje tudi v skrajnje popolnjeni bazi podatkov. To pa je bil tudi nas namen.

Na ta osnovni sistem se seveda lahko nadogradijo tudi ostale knjiznicne funkcije, ki bi omogocale pregled in evidenco nad obstojeco literaturo (kolicina, kje se nahaja, ...). Vec o tem kdaj drugic.

VI. LITERATURA

1. Donald E. Knuth: The Art of Computer Programming (Addison-Wesley Publ. Company, 1973)
2. K. Jensen, K. Wirth: PASCAL User Manual and Report (Springer Verlag, 1975)
3. Control Data : PASCAL/M User's Reference Manual (CDC publ. number 62940022 B, 1981)

CALL FOR PAPERS

The Sixteenth International Symposium on Fault-Tolerant Computing

July 1–3, 1986
Vienna, Austria

Symposium Chairman
H. Kopetz
TU Vienna, Austria

Program Chairman
M. DalCin
Univ. Tübingen, FRG

Publicity Chairman
E. Schmitter
Siemens Munich, FRG

Program Committee
J. A. Abraham, USA
V. K. Agrawal, Canada
T. Anderson, GB
A. Avizienis, USA
J. Bartlett, USA
W. C. Carter, USA
F. Cristian, USA
K. E. Grosspietsch, FRG
J. Hlavicka, CSSR
R. Iyer, USA
K. H. Kim, USA
G. Le Lann, France
B. Littlewood, GB
R. Maxion, USA
D. Morgan, USA
S. Naito, Japan
B. E. Ossfeldt, Sweden
D. Powell, France
L. Simoncini, Italy
L. Svobodova, Switzerl.
Y. Tohma, Japan
K. Trivedi, USA
U. Voges, FRG
J. Wensley, USA
Y. W. Yang, China

Submit all Papers and Correspondence to
FTCS - 16
Interconvention Hofburg
P.O. Box 80
A-1107 Vienna, Austria
Phone: (43)(222) 520293
Telex: 111210 kgzhwa

Sponsored by: IEEE Computer Society's Technical Committee on Fault-Tolerant Computing
In cooperation with:
ÖCG Austria
GI Fed. Rep. of Germany
Technical University of Vienna
IFIP WG 10.4

FTCS is the conference on fault tolerant computing systems. It encompasses all aspects of specifying, designing, modeling, implementing, testing, diagnosing and evaluating dependable and fault tolerant computing systems and their components. In addition to the established fields of fault tolerance particular emphasis is placed on papers relating to practical experiences with real time systems, switching systems and transaction systems as well as the application of artificial intelligence techniques to the solution of problems in fault tolerance.

Information for Authors:

An abstract of the paper including up to five keywords should be submitted before October 25, 1985. Submit 6 copies of the paper (double spaced) before the submission deadline, November 25, 1985. Papers should be no longer than 5000 words. The first page of each paper must include the following information: title, the author's name, affiliations, complete mailing address, telephone number and electronic mail address where applicable, a maximum 150-words abstract of the paper and up to five keywords (important for the correct classification of the paper). If there are multiple authors, please indicate who will present the paper at FTCS-16 if the paper is accepted. The first page should also indicate that the papers has been cleared through the author's affiliations. The conference language is English only.

Papers relating to the following areas are invited:

- Fault-tolerant architectures
- Fault-tolerance in distributed systems and interconnection networks
- Artificial Intelligence for diagnosis and maintenance
- Reliable synchronization, consensus and interprocess communication in distributed systems
- Hardware/software tradeoffs in the design of fault-tolerant systems
- Design diversity in software and VLSI
- Robust programs and data structures
- Error handling, reconfiguration and restart
- Testing techniques, coverage and test tools for VLSI components and systems
- Fault-tolerance aspects of VLSI and WSI
- Modeling, verification and experimental evaluation of fault-tolerant systems
- Application of fault-tolerance techniques (robotics, pattern recognition, knowledge based systems etc.)
- Reliability and safety in real time systems
- Availability of transaction systems and electronic switching systems

Important dates
October 25, 1985
Abstract Due
November 25, 1985
Submission Deadline
March 10, 1986
Acceptance Notification
April 14, 1986
Final Version Due

INFORMATICA 3/85

STRUKTURNO AVTOMATSKO UČENJE

IGOR KONONENKO

FAKULTETA ZA ELEKTROTEHNIKO,
LJUBLJANA

UDK: 681.3:159.953

Prispevek je pregledno poročilo o veji umetne inteligence, ki se ukvarja z avtomatizacijo procesa učenja. Avtomatsko učenje lahko pripomore k boljšemu razumevanju pojava inteligence, ima pa lahko številne praktične posledice. Pri tem je pomembno, da je rezultat učenja človeku razumljiv. Kot alternativa klasičnim (statističnim) metodam za razpoznavanje in grupiranje vzorcev so se pojavile metode za <u>strukturno</u> avtomatsko učenje. Zanje je značilno, da so rezultati učenja simbolični opisi naučenih konceptov, ki so <u>človeku razumljivi</u>. V prispevku so opisani najprej splošni problemi učenja, nato so prikazane razlike med statističnim in strukturnim avtomatskim učenjem in zatem so opisani splošni principi strukturnega avtomatskega učenja. Predstavljene so tri splošne metode za strukturno avtomatko učenje na osnovi primerov (Mitchellova metoda prostora verzij, metoda zvezd Michalskega in gradnja odločitvenih dreves, ki jo je prvi uspešno implementiral Quinlan) ter metoda za konceptualno grupiranje vzorcev, ki sta jo razvila Michalski in Stepp. Predstavljeni so najbolj znani sistemi za strukturno avtomatsko učenje : Winstonov ARCHES, Mitchellov LEX, sistem ARCHES/X, ki je rekonstrukcija Winstonovega ARCHESa, Quinlanov ID3, naš ASISTENT, Patersonov in Niblettov ACLS, Michalskijevi AQ11, GEM in CLUSTER, Langleyev BACON, Dietterichov SPARC/G, E.Y. Shapirov MIS in Lenatov AM. Sistemi za strukturno avtomatsko učenje so uspešno preizkušeni v mnogih domenah in nekateri so zreli za rutinsko uporabo.

INDUCTIVE MACHINE LEARNING

Paper is an inductive machine learning state of the art report. The ability to learn is one of the most fundamental attributes of intelligent behavior. An artificial intelligence approach to this field has contributed to development of new methods for <u>inductive learning</u> which appear as an alternative to standard methods of pattern recognition and cluster analysis. The result of inductive learning is a symbolic description of given entities and is <u>comprehensible</u> to human users. The paper describes some fundamental principles of inductive learning. Three general methods for inductive learning from examples are presented and one for clustering: version space theory (Mitchell 78), the star methodology (Michalski 83), the decision tree approach (Quinlan 79) and a conceptual clustering method (Michalski & Stepp 83). Well known systems for inductive learning are described: ARCHES (Winston 75), LEX (Mitchell 83), ARCHES/X (Bundy 81), ID3 (Quinlan 79a), ASSISTANT (Kononenko et.al. 84), ACLS (Paterson & Niblett 82), AQ11, GEM (Michalski 83), CLUSTER (Michalski & Stepp 83), BACON (Langley 83), SPARC/G (Dietterich 80), MIS (E.Y. Shapiro 81), AM (Lenat 83). The inductive learning systems have proven their power in many fields such as chemistry, medicine, games, automatic programming etc. The products of inductive learning science became commercially acceptable and could be routinly used.

1. KAJ JE STRUKTURNO AVTOMATSKO UČENJE

1.1 KAJ JE UČENJE

Zmožnost učenja je eden osnovnih znakov <u>inteligentnega obnašanja</u>. Fenomen učenja nam je še vedno precej tuj in izzivalen. Globlje razumevanje procesa učenja lahko bistveno vpliva na naše razumevanje inteligence, poleg tega pa ima lahko čisto praktične posledice, npr. izboljšano in hitrejše izobraževanje. Proces učenja pri ljudeh izgleda včasih zelo počasen in neučinkovit. Celih 20 let je potrebno, da se človek razvije v strokovnjaka, ki je pripravljen, da se začne učiti svojega poklica. Zato je na dlani vprašanje, če lahko proces učenja avtomatiziramo.

Z razvojem avtomatskega učenja so se hitro pokazali problemi, ki nakazujejo kompleksnost in raznovrstnost tega fenomena, s tem pa se je naše razumevanje učenja delno izostrilo. Na vprašanje, kaj je to učenje, bi lahko odgovorili le približno. Eden od možnih odgovorov bi lahko bil takle (Simon 83): Učenje je kakršnakoli sprememba v sistemu, ki mu omogoča, da naslednjič izvaja isto ali sorodno nalogo bolje kot prej. Seveda je taka definicija preveč splošna, da bi nam vsaj delno približala ta misteriozni pojem. Nekoliko bolj podrobna je naslednja definicija (Carbonell in sod. 83) : Učenje v grobem vključuje

- pridobivanje novega opisnega znanja,

- razvoj in izpopolnjevanje veščin skozi prakso,

- strukturiranje že pridobljenega znanja in

- iskanje novih dejstev in teorij z opazovanjem in eksperimerntiranjem.

<u>Avtomatsko učenje</u> (machine learning) je torej avtomatsko izvrševanje prej naštetih procesov. V nadaljevanju 1. poglavja so podani osnovni cilji avtomatskega učenja in razlika med statističnim in strukturnim avtomatskim učenjem. V 2. poglavju so prikazani osnovni principi strukturnega avtomatskega učenja. V 3. poglavju so opisani najbolj znani obstoječi sistemi za strukturno avtomatsko učenje v svetu. V 4. poglavju je poudarjena uporabna plat sistemov in nakazane so smeri nadaljnjega razvoja.

1.2 ZAKAJ AVTOMATSKO UČENJE

Eden od namenov je bil že nakazan: zato, da bomo skozi probleme in principe avtomatskega učenja izpopolnili svoje razumevanje procesa učenja. Ena lepih lastnosti avtomatsko naučenega znanja je trivialna prenosljivost, ki pa na žalost ne velja za človeško znanje. Reševanje cele kopice problemov, ki jih sedaj rešujemo več ali manj "ročno" na podlagi dolgoletnega šolanja in izkušenj, se lahko pospeši, optimizira in izboljša s pomočjo avtomatskega učenja, npr. programiranje računalnikov in robotov, planiranje, napovedovanje vremena, medicinska diagnostika, razni klasifikacijski problemi, iskanje zakonitosti in pomembnih relacij v neznani ali zapleteni domeni ipd.

V novejšem času so se razvijalci ekspertnih sistemov (expert systems, glej npr. Bratko 82), to je sistemov, ki se na določenem ozkem problemskem področju znajo obnašati kot človek ekspert (specialist), znašli pred problemom, kako čim hitreje spraviti znanje ljudi ekspertov v računalnik. Ta prenos znanja je pri razvoju ekspertnih sistemov ozko grlo, saj ljudje le s težavo formalizirajo svoje znanje, pridobljeno na osnovi dolgoletnih izkušenj. Temu ozkemu grlu se lahko izognemo z avtomatskim učenjem tako, da na osnovi arhivskih podatkov o delu eksperta (učnih primerov) avtomatsko zgeneriramo znanje, ki ga je ekspert uporabljal pri reševanju problemov. Tak pristop je pokazal dobre rezultate (Michalski and Chilausky 80). Nekaj več o avtomatski sintezi znanja je napisano v (Bratko in sod. 85).

1.3 STATISTIČNO IN STRUKTURNO AVTOMATSKO UČENJE

Prvotne raziskave iz avtomatskega učenja so potekale na področju razpoznavanja vzorcev (pattern recognition) in grupiranja vzorcev (clustering analysis). Razvita je bila cela vrsta metod za avtomatsko učenje, ki se jih je pozneje oprijel vzdevek "statistične" metode. Metode za razpoznavanje vzorcev so npr. diskriminantna analiza, Bayesov verjetnostni princip in regresijska analiza, za grupiranje pa npr. postopek k-tih povprečij, hierarhično grupiranje in algoritem ISODATA (Nilsson 65, Nie in sod. 75, Pavešić in Mihelič 81, Kononenko in sod 84, Zupan 82).

Vse te metode imajo bistveno slabost: rezultati učenja so človeku nerazumljivi, nedojemljivi, nejasni, pa čeprav so mogoče pravilni. Vse te metode namreč uporabljajo določene (matematične) formalizme, ki so v naprej določeni (enačbe, funkcije) in nimajo nič skupnega s človeškim načinom razmišljanja. Če se npr. pri diagnosticiranju bolnika računalnikova diagnoza ne ujema z zdravnikovim mišljenjem, bo zdravnik računalnikovo diagnozo upošteval le, če jo bo računalnik obrazložil in argumentiral. To, da je izračun verjetnosti po taki in taki enačbi pokazal največjo verjetnost določene diagnoze ali pa, da je vrednost diskriminantne funkcije za to diagnozo večja od vrednosti za vse ostale diagnoze, je bore slaba obrazložitev. S tem nočemo trditi, da so zato vse te metode ničvredne in neuporabne. Prav gotovo so področja, kjer so te metode zelo uporabne, vendar niso primerne za reševanje problemov, kjer je potrebno globoko iskanje povezav med znanimi dejstvi, iz katerih lahko sklepamo na vzroke in posledice.

S temi problemi se ukvarjajo raziskovalci umetne inteligence (artificial intelligence, glej npr. Nilsson 82). Osnovni principi, ki se uporabljajo v metodah umetne inteligence so v grobem

- iskanje približno optimalnih rešitev kompleksnih problemov z uporabo heuristik, to je napotkov za usmerjanje reševanja problemov. Te so človeku razumljive in dajejo dobre eksperimentalne rezultate, čeprav je včasih njihova pravilnost nedokazana ali nedokazljiva.

- oponašanje človekovega načina reševanja problemov

- uporaba človeku razumljivih formalizmov za predstvitev znanja

- kvalitativno namesto numeričnega reševanja problemov.

Z uporabo naštetih principov se je ustvarila nova veja avtomatskega učenja: strukturno avtomatsko učenje (lahko bi rekli tudi simbolično, konceptualno, kvalitativno, induktivno učenje). Rezultat takega učenja je formula, pravilo, teorija ali opis koncepta v kvalitativnem, logičnem formalizmu, ki je človeku dostopen in razumljiv. Iz pravila lahko uporabnik razbere določene relacije, zakonitosti in logiko sklepanja, ki je potrebna, da sistem pride na osnovi pravila do določenih zaključkov. V poglavju 1.4 je narejena primerjava strukturnega učenja s statističnim glede natančnosti in razumljivosti, kar bo nekoliko pojasnilo dosedanje razmišljanje.

1.4 PRIMERJAVA STATISTIČNEGA IN STRUKTURNEGA UČENJA

Za ilustracijo razlike med strukturnim in statističnim učenjem bomo prikazali dve vrsti eksperimentov. Prve smo vršili na področju razpoznavanja vzorcev, druge pa sta izvedla Michalski in Stepp iz univerze v Ilinoju v ZDA na področju grupiranja vzorcev.

1.4.1 RAZPOZNAVANJE VZORCEV

Na Fakulteti za elektrotehniko in Onkološkem inštitutu v Ljubljani so bili narejeni poskusi z avtomatskim učenjem medicinskih diagnostičnih pravil na več načinov: z nekaterimi statističnimi metodami in s sistemom za strukturno avtomatsko učenje odločitvenih pravil v obliki odločitvenih dreves ASISTENT, ki smo ga razvili na Fakulteti za elektrotehniko v Ljubljani in je nekoliko podrobneje opisan v poglavju 3.3.2. Zgrajena odločitvena drevesa so direktno berljiva in zdravnikom popolnoma razumljiva. Problem učenja je definiran takole :

DANO : Množica učnih primerov, opisanih z množico atributov. Vsak objekt pripada enemu od možnih razredov.

POIŠČI : Pravilo, ki razlaga (pravilno klasificira) učne primere in ki se ga lahko uporabi za klasifikacijo novih primerov.

Uporabljali smo sledeče statistične metode :

- Bayesov princip verjetnosti (Kononenko in sod. 84), ki po določeni formuli računa verjetnosti posameznih diagnoz za dani primer. Parametri iz formule so aproksimirani z relativnimi frekvencami iz učne množice primerov.

- diskriminantno analizo (Nilsson 65, Nie in sod. 75, Roškar 84, Roškar in sod. 85), ki predpostavlja, da vsak primer predstavlja točko v n-dimenzionalnem prostoru (n je število atributov, ki opisujejo primere). Metoda išče funkcije, ki določujejo hiperravnine, ki ločijo med seboj grupacije primerov z istimi razredi.

- metodo lupin v n-dimenzionalnem prostoru (Soklič 80), ki tvori lupine okoli grupacij primerov z istimi razredi

V tabeli 1.1 so primerjalni rezultati (dosežena natančnost diagnosticiranja) eksperimentiranja v 6 različnih medicinskih domenah. Vse metode, tako statistične kot ASISTENT, so dosegle v vseh domenah natančnost diagnosticiranja zdravnikov specialistov. Bistvena prednost ASISTENTa je v razumljivosti dobljenega odločitvenega pravila, iz katerega lahko zdravnik direktno razbere logiko sklepanja in lahko celo ugotovi določene relacije in zakonitosti v svoji domeni (Zwitter

in sod. 83). Odločitveno drevo se lahko uporablja tudi brez računalnika, npr. kot priročnik za diagnosticiranje.

domena	Bayes	diskr.anal.	lupine	ASISTENT
primarni tumor	45%	-	47%	46%
rak na dojki	74%	-	-	72%
hepatitis	88%	-	-	80%
limfografija	67%	-	58%	65%
inkontinenca m.	67%	-	-	67%
inkont. ženske	79%	81%	-	81%

Tabela 1.1 Primerjava dosežene diagnostične natančnosti treh statističnih metod za avtomatsko učenje in sistema ASISTENT (glej pogl. 3.3.2). Znak "-" pomeni, da ustrezni poskus ni bil izveden.

1.4.2 GRUPIRANJE VZORCEV

Tu je problem definiran takole :

DANO : Množica primerov, ki so opisani z množico atributov.

POIŠČI : Grupe med seboj najbolj podobnih primerov. (število željenih grup je po navadi dano vnaprej)

Statistične metode se med seboj razlikujejo po algoritmu grupiranja in po merilu podobnosti, vse pa imajo to slabost, da dobljene grupe niso opisane. Alternativa tem metodam je sistem CLUSTER, ki sta ga razvila Michalski in Stepp (79,83,83a) in ki je podrobneje opisan v pogl. 3.5. CLUSTER poleg grupacije poda tudi logične, direktno berljive opise grup. Michalski in Stepp (83a) sta napravila nekaj poskusov s CLUSTERjem in standardnimi statističnimi metodami. CLUSTER se je izkazal za 2 velikostna razreda počasnejši, vendar so njegove grupacije opisane. Poleg tega pa se je izkazalo, da CLUSTER grupira vzorce bolj naravno (podobno kot človek) in so zaradi tega dobljene grupacije enostavneje opisljive (brez disjunkcij).

2. PRINCIPI STRUKTURNEGA AVTOMATSKEGA UČENJA

2.1 DEFINICIJA

Strukturno avtomatsko učenje je avtomatski proces pridobivanja znanja z induktivnim sklepanjem na osnovi informacij, dobljenih od učitelja, nekega zunanjega procesa ali pridobljenih z opazovanjem in eksperimentiranjem. Pridobljeno znanje je lahko pravilo, teorija ali opis koncepta. Tak proces zahteva posploševanja, specializacije in reformulacije notranje predstavitve znanja (opisa koncepta). Pri tem morajo biti rezultati učenja človeku razumljivi.

2.2 RAZUMLJIVOST

Opisi delno in popolnoma naučenega znanja morajo biti logični in morajo karakterizirati koncepte v visokonivojskih izrazih in relacijah. Zaradi tega je pomembna izbira opisnega jezika. Tipični predstavniki opisnih jezikov v sistemih za strukturno avtomatsko učenje so

- predikatni račun,
- produkcijska (if-then) pravila,
- hierarhični opisi,
- semantične mreže in
- odločitvena drevesa.

Da ohranimo razumljivost, mora predstavitev znanja zadovoljevati naslednji postulat razumljivosti (Michalski 83b) :

"Rezultati strukturnega avtomatskega učenja morajo biti simbolični opisi danih izpeljanih hipotez. Ti opisi morajo biti semantično in strukturno podobni opisom, ki bi jih sestavili strokovnjaki dane domene, če bi opazovali iste pojave kot sistem. Opisi morajo biti sestavljeni iz razumljivih zaključenih informativnih celot, ki se dajo direktno izraziti v naravnem jeziku. Jedrnato morajo izražati kvalitativne in kvantitativne koncepte."

Okvirni napotek za obsežnost opisa v predikatni logiki, da ga človek zlahka dojame, je približno takle : manj kot 5 pogojev v konjunkciji ali nekaj enostavnih pogojev v disjunkciji, največ en nivo oklepajev, največ ena implikacija, ne več kot dva kvantifikatorja in brez rekurzije.

2.3 VRSTE UČENJA

Vrste učenje lahko določimo po različnih kriterijih. Eden najpomembnejših je količina potrebnega induktivnega sklepanja, ki ga mora učenec izvršiti v procesu učenja. Po tem kriteriju delimo učenje na 5 kategorij (Carbonell in sod. 83):

1. Direktno ali rutinsko učenje (rote learning): za pridobivanje znanja ni potrebno nič sklepanja. V to skupino spadata direktno programiranje in shranjevanje podatkov.

2. Učenje na osnovi povedanega (learning by being told): učenje zahteva nekaj predznanja, na osnovi katerega z induktivnim sklepanjem pridobljeno znanje ustrezno spremenimo v notranjo obliko in ga vgradimo v bazo znanja. Pridobljeno znanje moramo biti sposobni uporabiti, ne da bi nam pri tem bili dani eksplicitni algoritmi. V to skupino spada sprejemanje pravil in dejstev od učitelja.

3. Učenje po analogiji (learning by analogy): pri tej obliki učenja je potrebno nekoliko več induktivnega sklepanja. Že pridobljeno znanje je treba transformirati tako, da ga lahko uporabimo za reševanje problemov, ki so podobni prvotnemu problemu (staro znanje prilagodimo novemu problemu).

4. Učenje na osnovi primerov (learning from examples): z induktivnim sklepanjem je treba izpeljati pravilo, teorijo ali opis pojava, ki je predstavljen z dejstvi - učnimi primeri. Učni primeri so lahko pozitivni ali negativni (so ali niso primeri koncepta, ki se ga želimo naučiti). Dobljeno pravilo ali opis koncepta mora vključevati vse pozitivne primere (kompletnostni pogoj) in izključevati vse negativne primere (konsistenčni pogoj). Če pa negativnih primerov ni, potem mora opis vključevati vse učne primere in čim manj primerov, ki niso v množici učnih primerov.

To je najširše raziskovalno področje avtomatskega učenja in prav na tem področju so doseženi že zelo dobri rezultati. V tem prispevku se bomo osredotočili na sisteme, ki se učijo na osnovi primerov (izjemi sta konceptualno grupiranje - sistem CLUSTER (glej 3.5) in sistem za samostojno odkrivanje AM (glej 3.6.4)). Učenje na osnovi primerov lahko razdelimo na dve glavni podpodročji glede na vir učnih primerov:

(a) Vir učnih primerov je učitelj ali pa so rezultat opazovanja in meritev nekega zunanjega procesa. V tem prispevku so opisani naslednji sistemi, ki spadajo v to skupino: ARCHES (glej 3.1), ID3 (3.3.1), ASISTENT (3.3.2), ACLS (3.3.3), AQVAL, INDUCE, AQ11 in GEM (3.4) in SPARC/G (3.6.2). Če je vir primerov učitelj, potem je lahko vrstni red primerov izbran tako, da učenec najlažje in čim hitreje napreduje. Winston (75) je ugotovil, da

so dobri učni primeri bližnji pogreški (near misses), to je negativni učni primeri, ki se razlikujejo od pozitivnih učnih primerov le po eni lastnosti ali nekaj lastnostih. (npr. če želimo nekoga naučiti koncepta avto, potem mu opis mize kot negativnega primera ne bo veliko koristil pri učenju).

(b) Učne primere predlaga učenec sam, da bi čim hitreje izoblikoval končni opis koncepta, ki se ga uči. Pri tem pa mora seveda imeti možnost, da za predlagan učni primer dobi odgovor, ali je primer pozitiven ali negativen (odgovor po navadi da učitelj). V tem prispevku so opisani naslednji sistemi iz te skupine: LEX (glej 3.2.3), ARCHES/X (3.2.4), BACON (3.6.1) in MIS (3.6.3).

Iste sisteme lahko razdelimo še po vrstnem redu upoštevanja učnih primerov:

(a) Vse učne primere upoštevajo naenkrat (ID3, ACLS, ASISTENT, AQVAL, AQ11, INDUCE, GEM, SPARC/G)

(b) Učne primere sprejemajo enega za drugim in vsakič spreminjajo opis delno naučenega koncepta (ARCHES, LEX, ARCHES/X, BACON, MIS).

Glede na cilj učenja imenujemo učenje na osnovi primerov tudi učenje konceptov (concept acquisition) in ga delimo na:

- učenje opisa koncepta ali teorije (characteristic description), sem spadajo sistemi ARCHES, LEX, ARCHES/X, BACON in MIS,

- učenje razlikovalnega pravila (discriminant description), sem spadajo ID3, ASISTENT, ACLS, AQVAL, AQ11, INDUCE in GEM,

- učenje pravila generacije zaporedja (sequence extrapolation rule), sem spada sistem SPARC/G.

5. Učenje s samostojnim odkrivanjem (learning from observation and discovery): To je najbolj zahtevna oblika učenja, ki vključuje odkrivanje novih konceptov, postavljanje in preizkušanje hipotez in sestavljanje novih teorij. To je učenje brez učitelja. Ta oblika učenja zahteva največ induktivnega sklepanja. Glede na način opazovanja okolice ga delimo na:

(a) pasivno opazovanje dogodkov (sem spada sistem CLUSTER, glej 3.5)

(b) aktivno eksperimentiranje in postavljanje teorij na osnovi ekperimentov (sem spada sistem AM, glej 3.6.4).

Glede na cilj učenja imenujemo učenje s samostojnim odkrivanjem tudi opisno posploševanje (descriptive generalization) in ga delimo na:

- formiranje teorije za opis skupine objektov in lastnosti (AM)

- iskanje vzorcev v opazovanih podatkih (CLUSTER)

2.4 ATRIBUTI - LASTNOSTI POJAVOV

V večini problemov učenja nastopajo objekti, pojmi ali pojavi, ki so opisani z določeno (fiksno) množico atributov (lastnosti, opisovalcev, deskriptorjev). Vsak atribut ima določeno množico možnih vrednosti, ki jih lahko zavzema. Ta množica je lahko:

- neurejena - nominalni atributi (npr. vreme je lahko sončno, oblačno, deževno, sneženo ali vetrovno)

- zvezno urejena - zvezni atributi (npr. temperatura telesa je lahko katerakoli vrednost od 35 st.C do 40 st.C)

- strukturirano urejena - strukturirani atributi (npr. lik je lahko konkaven ali konveksen, konveksen je lahko mnogokotnik ali krog, mnogokotnik je lahko trikotnik, štirikotnik ali večkotnik, štirikotnik je lahko kvadrat, romb, trapez ali romboid, itd.)

Uspešnost učenja je odvisna od kvalitete (informativnosti, popolnosti) atributov. Množica atributov je za dani problem:

(a) polna, če je z uporabo atributov iz dane množice možno eksaktno rešiti problem (naučiti se eksaktno pravilo),

(b) delna, če problem lahko rešimo le do določene meje,

(c) indirektno polna (ali delna), če problem lahko eksaktno (ali do določene meje) rešimo, če uspemo iz danih atributov izpeljati nove atribute, ki so za dano domeno pomembnejši kot sami osnovni atributi (npr. kvocient dveh rezultatov testov je lahko zelo pomemben za dani problem, medtem ko vsak test zase nima praktičnega pomena).

Izbira dobrih atributov je ključni problem pri učenju in je še vedno naloga človeka, eksperta iz dane domene. Treba je izbrati tiste atribute, za katere vemo, da vplivajo na dani pojav (npr. telesna temperatura je lahko znanilec bolezni), zavreči atribute, za katere vemo, da ne vplivajo na dani pojav (npr. barva las ne vpliva na potek bolezni) in vključiti atribute, za katere nismo čisto prepričani, če so pomembni. Teh zadnjih ne sme biti mnogo, ker lahko proces učenja postane preveč neučinkovit, pa tudi slabše rezultate lahko dobimo. Michalski (83) je takole zaokrožil iterativno reševanje problema z induktivnim učenjem:

```
ponavljaj
   1. uredi domeno čim bolje (izberi prave atribute)
   2. zgradi pravila (hipotezo, teorijo, opis koncepta)
   3. testiraj dobljeno pravilo,
dokler niso rezultati testiranja zadovoljivi
```

Da izberemo dobre atribute, moramo poznati:

- tip atributa, zalogo vrednosti, uporabne operatorje,

- omejitve, relacije z drugimi atributi, definicijo izpeljave iz drugih atributov,

- verjetnostne porazdelitve vrednosti, zakonitosti za objekte, ki jih opisujejo (npr. tranzitivnost),

- opis objektov, za katere se atribut uporablja in

- opis strukturiranih grup atributov.

2.5 PRAVILA SPREMINJANJA OPISOV

Vsako zahtevnejše učenje vsebuje uporabo pravil za posploševanje, specializacijo in reformulacijo tekočega opisa koncepta, hipoteze ali teorije. Tako npr. otrok, ko prvič vidi vrabca in mu mama reče, da je to vrabec, pojem vrabca posploši na vse živali, ki imajo krila, kljun in ki letajo. Ko bo zatem zagledal koša in mu rekel vrabec, ga bo mama poučila, da je to koš, ker je črn. Zato bo svoj opis vrabca specializiral na vse živali, ki imajo krila, kljun in letajo ter niso črne. Za posploševanje velja, da ohranja neresničnost (če je opis neresničen, je tudi posplošen opis neresničen). Za specializacijo velja, da ohranja resničnost (če je opis resničen, potem je tudi bolj specifičen opis resničen). Za reformulacijo velja, da mora ohranjati tako resničnost kot neresničnost. Če se omejimo na predikatno logiko, so pravila za reformulacijo splošno veljavna pravila za spreminjanje logičnih izrazov (npr. DeMorganova pravila). Dietterich in Michalski (81,83) sta pravila posploševanja (generalizacije) strnila v:

(a) spreminjanje konstante v spremenljivko (npr. stavek

"težje stvari se reče žoga, ker je okrogla" lahko posplošimo v "vsaka okrogla stvar je žoga")

(b) dodajanje notranje disjunkcije (npr. "vse žoge so rdeče" posplošimo v "vse žoge so rdeče ali črne")

(c) razširjanje vrednosti v interval (npr. "36.5 st.C je normalna telesna temperatura" posplošimo v "od 36.0 do 36.9 st.C je normalna telesna temperatura")

(d) plezanje po posploševalnem drevesu (npr. "trikotnik" posplošimo v "mnogokotnik")

(e) opustitev pogoja (npr. "ptiči imajo perje in letijo" posplošimo v "ptiči imajo perje")

(f) sprememba konjunkcije v disjunkcijo (npr. "ptiči ležejo jajca in letajo" lahko posplošimo v "ptiči ležejo jajca ali letajo")

(g) sprememba univerzalnega kvantifikatorja v eksistenčnega (npr. "vsak ptič je črn" posplošimo v "obstajajo črni ptiči")

Pravila za specializacijo so obratna pravilom generalizacije. Poleg tega obstaja še posebno pravilo specializacije - pravilo izjeme (npr. "vrabci so vse živali, ki imajo kljun, krila in letajo" lahko specializiramo v "vrabci so vse živali, ki imajo kljun, krila in letajo, razen če so črne").

2.8 MERILA ZA KVALITETO REZULTATOV UČENJA

Kvaliteto vmesnih in končnih rezultatov učenja (opisov koncepta, teorije ali pravil) lahko določimo po različnih kriterijih. Osnovni kriteriji so (Michalski 83b):

- razumljivost, preprostost opisov,

- stopnja pravilnosti glede na učne in testne primere,

- cena meritev, potrebnih za pridobitev vrednosti atributov, ki nastopajo v opisih (pomembno npr. v medicini, kjer so nekateri testi lahko zelo dragi ali nevarni za pacienta),

- cena kompleksnosti izpeljave opisov (npr. CPU čas, spominski prostor),

- spominski prostor, potreben za shranitev opisov in

- množina informacije, potrebna za zakodiranje dobljenih opisov.

3. METODE IN SISTEMI ZA STRUKTURNO AVTOMATSKO UČENJE

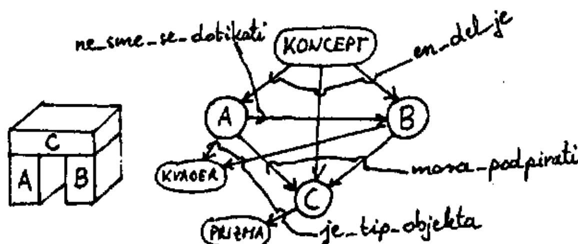
V tem poglavju so predstavljene tri splošne metode za učenje na osnovi primerov, metoda prostora verzij (pogl.3.2), metoda zvezd (3.4) in gradnja odločitvenih dreves (3.3), ter metoda za konceptualno grupiranje vzorcev (3.5). Ob vsaki metodi so opisani najbolj znani obstoječi sistemi v svetu, ki temeljijo na dani metodi. Na koncu (v pogl. 3.6) so podani še nekateri drugi zanimivi sistemi za strukturno avtomatsko učenje. V pogl. 3.1 je opisan Winstonov sistem ARCHES za učenje konceptov v svetu kock. To je začetni poskus ustvariti sistem, ki se bo učil po podobnih principih kot človek, vendar zaradi slabega formalizma ni ohranil svoje začetne slave. Pri vsakem sistemu je v grobem opisan osnovni algoritem učenja, opisni jezik, predstavitev pridobljenega znanja in področje uporabe.

3.1 ARCHES

Winston (75) je razvil sistem ARCHES za učenje konceptov na osnovi primerov in protiprimerov. Eksperimentalna domena je bil svet, v katerem nastopajo razna pravilna geometrijska telesa, kot so kocka, kvader, piramida itd. Sistem je dobil ime po enem od konceptov (slavolok), ki se ga je naučil. Algoritem učenja je v grobem takle:

1. Zgradi strukturni opis prvega pozitivnega primera
2. Za vse učne primere ponavljaj:
2.1 Primerjaj tekoči primer s trenutnim opisom koncepta.
2.2 Izračunaj razliko med primerom in opisom.
2.3 Glede na izračunano razliko in glede na to, ali je primer pozitiven ali negativen, ustrezno popravi opis koncepta.

Zanimiv je bil formalizem za predstavitev strukturnega opisa. Winston je uporabljal neke vrste semantične mreže. Vozlišča v mreži so ustrezala objektom, povezave med vozlišči pa relacijam med objekti. Zanimivo je to, da se odvisnosti med relacijami lahko podajajo v istem formalizmu. V tem primeru vozlišča predstavljajo relacije, povezave med vozlišči pa predstavljajo odvisnosti med relacijami. Slika 3.1 kaže primer slavoloka in semantično mrežo, ki ustreza naučenemu opisu koncepta.



Slika 3.1 Primer slavoloka in semantična mreža, ki ustreza konceptu slavoloka, ki se ga je ARCHES naučil.

Poskusi s sistemom so pokazali na slabo formalizacijo in veliko občutljivost sistema na izbiro učnih primerov in na vrstni red sprejemanja primerov. Ena pomembnejših ugotovitev je bila definicija dobrih učnih primerov, to je bližnjih pogreškov (near misses). Dober učni primer je tak negativen primer, ki se od pozitivnega razlikuje samo po eni lastnosti. Zato je učenje z natančno izbiro učnih primerov in njihovega vrstnega reda veliko hitrejše, kot če bi naključno izbirali učne primere.

3.2 METODA PROSTORA VERZIJ

3.2.1 DEFINICIJA PROSTORA VERZIJ

Mitchell (78) je razvil teorijo prostora verzij (version space), ki predstavlja splošno in formalno orodje za sisteme za strukturno avtomatsko učenje. Teorija zastavlja učenje opisov konceptov kot iskanje v prostoru možnih opisov za dani jezik. Opisi v celotnem prostoru možnih opisov so delno urejeni. Delno urejenost določa relacija "bolj specifičen kot" in jo bomo označevali z ">". Obratna relacija je "bolj splošen kot" ali "manj specifičen kot" in jo bomo označevali z "<". Relacija ">" določa delno urejenost, ker ima naslednje lastnosti:

(a) je antisimetrična : $A > B \Leftrightarrow B < A$

(b) je tranzitivna: $A > B \ \& \ B > C \Rightarrow A > C$

(c) če sta A in B enako specifična, sta tudi enaka: $A = B$

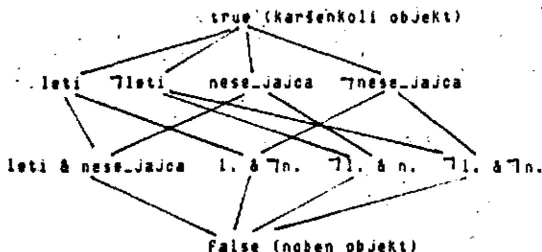
(d) velja: $(A = C \ \& \ D) \ \& \ (B = C \ \& \ E) \Rightarrow A > C \ \& \ B > C \ \& \ not(A > B) \ \& \ not(B > A)$

V takem primeru ne moremo določiti relacije med A in B. Nobeden ni bolj specifičen od drugega, enaka pa tudi nista (razen v primeru, ko je D = E).

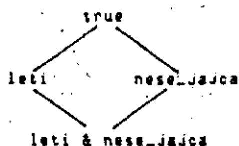
Trenutno znanje o konceptu, ki se ga učimo, nam določa podmnožico opisov iz celotnega prostora možnih opisov. To podmnožico imenujemo prostor verzij. Če o konceptu ne vemo ničesar, potem je prostor verzij enak celotnemu prostoru možnih opisov. Z izpopolnjevanjem znanja o konceptu se prostor verzij manjša. Koncept je popolnoma naučen, če je v prostoru verzij PV ostala množica opisov, ki medsebojno niso v relaciji ">":

PV = { x,y | x ≠ y => not(x > y) & not(y > x) }

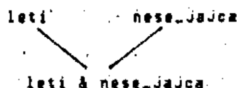
PRIMER: Recimo, da imamo dan jezik L = {leti, nese_jajca}. (zaradi enostavnosti bomo dovoljevali samo operatorja konjunkcije (&) in negacije (¬) nad enostavnimi izrazi). Potem je prostor možnih opisov sestavljen iz naslednjih opisov (povezave med opisi od spodaj navzgor predstavljajo relacijo ">"):



Poglejmo si opis učenja koncepta "ptič". Na začetku je prostor verzij kar celoten prostor možnih opisov. Zatem zvemo, da za nekega ptiča velja, da leti in nese jajca. Naš prostor verzij se zato zmanjša na:



Vsi preostali opisi ne ustrezajo našemu prvemu pozitivnemu primeru. Zatem zvemo, da za neko žival velja, da ne leti in da ne nese jajca in da ni ptič. Naš prostor verzij se spet zmanjša:



Nato zvemo še za ptiča, ki ne leti, a nese jajca. Prostor verzij je sedaj sestavljen iz samo enega opisa, ki ustreza tudi zadnjemu učnemu primeru: "nese_jajca". S tem je udenje končano, koncept "ptič" je naučen.

Pri tej metodi učenja je kritična predstavitev prostora verzij, saj za obsežen opisni jezik množica možnih opisov kombinatorično narašča. Mitchell (78) je ugotovil, da je prostor verzij enolično določen z dvema množicama opisov:

S - množica vseh najbolj specifičnih opisov konceptov iz prostora verzij (PV);

S = { x ∈ PV | y ∈ PV => not(y > x) }

G - množica vseh najbolj splošnih opisov konceptov;

G = { x ∈ PV | y ∈ PV => not(x > y) }

Poleg kompaktnosti opisa prostora verzij smo dobili preprost kriterij, kdaj je koncept popolnoma naučen. Takrat sta namreč množici S in G enaki. Uporaba mejnih množic tudi precej poenostavi algoritem učenja.

3.2.2 ALGORITEM UČENJA

Algoritem učenja je v grobem takle:

1. S <-- prvi pozitivni primer,
   G <-- true (opis kakršnegakoli objekta).
2. Dokler S ≠ G & S ≠ ∅ & G ≠ ∅ ponavljaj:
   če je novi primer negativen,
   potem
      2.1.1 zadrži v S samo opise, ki ne ustrezajo primeru,
      2.1.2 zamenjaj G z množico najbolj splošnih opisov v trenutnem PV, ki ne ustrezajo primeru,
   drugače (primer je pozitiven)
      2.2.1 zadrži v G samo opise, ki ustrezajo primeru
      2.2.2 zamenjaj S z množico najbolj specifičnih opisov v trenutnem PV, ki ustrezajo primeru
3. če je S = ∅ ali G = ∅,
   potem izpiši "učni primeri so nekonsistentni",
   drugače izpiši "rezultat je disjunkcija opisov iz " S.

Lastnosti metode s prostorom verzij so:

- dobimo vse opise, konsistentne z učnimi primeri,

- avtomatsko lahko ugotavljamo nekonsistentnosti v podatkih (npr. če sta pozitivni in negativni primer enako opisana), takrat namreč postane prosor verzij prazen (s tem tudi S in G),

- avtomatsko ugotavljanje, kdaj je koncept popolnoma naučen (S=G),

- čeprav koncept ni popolnoma naučen, lahko zanesljivo klasificiramo nove primere po algoritmu:

(1) če primer ustreza vsem opisom iz S, je to primer koncepta (pozitiven primer),

(2) če primer ne ustreza nobenemu opisu iz G, to ni primer koncepta (je negativen primer),

(3) drugače primera ne znamo klasificirati;

- omogoča avtomatsko izbiro informativnih učnih primerov (primer bo informativen, če bo čimbolj zmanjšal prostor verzij, zato mora biti vsebovan v trenutnem prostoru verzij in mora biti tak, da ga ne znamo klasificirati),

- rezultati so neodvisni od vrstnega reda učnih primerov,

- primerov ne pregledujemo ponovno, lahko jih sproti pozabljamo, zato je kompleksnost metode premo sorazmerna številu učnih primerov,

- vračanje na prejšnji opis koncepta ni nikoli potrebno (stare opise lahko sproti pozabljamo).

Najpomembnejši Mitchellovi rezultati so iz učenja pravil za generiranje molekularnih struktur v programu META-DENDRAL (Buchanan in sod. 78). V poglavjih 3.2.3 in 3.2.4 sta opisana eksperimentalna sistema za strukturno avtomatsko učenje, ki uporabljata metodo prostora verzij.

3.2.3 LEX

Mitchell je s sodelavci razvil sistem LEX, ki temelji na teoriji prostora verzij in se uči pravila (hevristike) za simbolično integeriranje (Mitchell 83, Utgoff 83, Mitchell in sod. 83). LEX ima na razpolago znane operatorje, ki določajo legalna pravila za transformacijo aritmetičnih izrazov z integrali (npr. izpostavljanje konstante iz integrala, integracija po delih,...), in hierarhično posplošitveno drevo aritmetičnih pojmov (npr. funkcija je lahko polinom ali trigonometrična ali..., trigon. funk. je lahko sinus ali cosinus, itd.). LEXova naloga je, da se nauči hevristična pravila, ki ga bodo vodila po čim krajši poti do rešitve. Pravila so v obliki "če situacija, potem uporabi tak in tak operator".

Tipična hevristika, ki se je je LEX naučil, je:

$$\int X \cdot trig(X)dX \Rightarrow \text{uporabi integracijo po delih } U = X,\ dV = trig(X)dX$$

Prostor verzij predstavlja LEXu prostor možnih opisov za iskano hevristiko. Pri tem se relacija "bolj specifičen kot" nanaša na prej omenjeno hierarhično drevo. Tako je izraz f1(X)*f2(X) bolj splošen kot X*f2(X) in ta spet bolj splošen kot X*trig(X), itd., medtem ko med izrazoma X*trig(X) in polinom(X)*cos(X) ne moremo določiti relacije "bolj specifičen kot". Tudi če hevristika ni popolnoma naučena, se lahko uporablja za reševanje problemov. Prav tako lahko avtomatsko generiramo probleme integracije, ki bodo zmanjšali prostor verzij za iskano hevristiko. LEX sestavljajo štirje moduli:

1. Problem-solver rešuje probleme simbolične integracije s pomočjo trenutno znanih hevristik, če pa mu zmanjka hevristik, uporablja "best-first-search" metodo iskanja poti do rešitve (vedno izbere pot, ki minimizira ocenjeno ceno poti do rešitve, glej npr. Nilsson 82).

2. Critic analizira pot do rešitve problema in generira učne primere za učenje hevristik. Pri tem so vsi deli poti, ki pelje od začetnega problema do rešitve, pozitivni primeri za učenje hevristik, in veje, ki vodijo stran od rešitvene poti, so negativni primeri.

3. Generalizer na osnovi učnih primerov generira nove hevristike (inicializira prostor stanj za novo hevristiko) ali pa detajlizira (zmanjša prostor stanj) že delno naučene hevristike za dani operator.

4. Problem-generator generira probleme simboličnega integriranja po dveh kriterijih:

- da omogoči detajliziranje delno naučenih hevristik in

- da omogoči generacijo novih hevristik: generira primere, ki ustrezajo pogoju za uporabo dveh ali več operatorjev hkrati, za katere še ni naučenih hevristik.

Poleg opisanih zmožnosti so izpopolnili sistem LEX tako, da zna generalizirati že iz enega samega rešenega problema, kar je bistveno pohitrilo učenje hevristik, poleg tega pa so se pokazale možnosti, da bi se sistem lahko sam naučil določene nove koncepte. Tako je npr. LEX iz enega problema integracije izpeljal definicijo lihega števila (real(C) & integer((C-1)/2))!

3.2.4 ARCHES/X

ARCHES/X je rekonstrukcija Winstonovega sistema ARCHES (glej 3.1), ki temelji na teoriji prostora verzij. Rekonstrukcijo so izvedli Plotkin, Young in Linz, implementiral pa Bundy (81). Program je napisan v PROLOGu (glej npr. Clocksin & Mellish 81) in obsega 9 strani PROLOGove kode. Program smo dobili na Fakulteto za elektrotehniko v Ljubljani in ga nekoliko dopolnili (Kononenko 83,83a). Sistem omogoča:

- delno in celotno učenje koncepta,

- klasifikacijo novih primerov z naučenim ali delno naučenim opisom koncepta,

- pomoč pri izbiri informativnih učnih primerov in

- ugotavljanje kontradiktornosti učnih primerov.

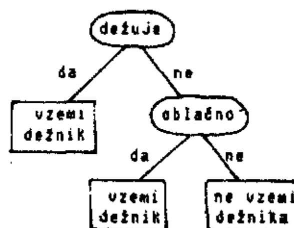
Pri eksperimentiranju z učenjem opisov konceptov iz sveta kock se je pokazala velika neučinkovitost sistema (če je N število objektov, ki nastopajo v opisu koncepta, sta prostorska in časovna kompleksnost reda O(N!). Razen tega sistem ne sledi popolnoma definiciji prostora verzij in ne omogoča disjunktivnega opisa koncepta.

3.3 GRADNJA ODLOČITVENIH DREVES

V tem poglavju je opisana vrsta sistemov, ki temeljijo na istem formalizmu. Učinkovitost učenja po principu gradnje odločitvenih dreves je prvič pokazal Quinlan s svojim sistemom ID3, iz katerega se je pozneje razvila cela vrsta sistemov, od katerih se nekateri že rutinsko uporabljajo in dosegajo lepe uspehe. Preprostost metode prispeva k učinkovitosti in dobrim rezultatom.

3.3.1 ITERATIVE DICHOTOMIZER 3 - ID3

Quinlan (79,79a,82,83,83a) je razvil program ID3 za gradnjo odločitvenih dreves nad velikimi množicami učnih primerov. Odločitveno drevo je drevo, katerega vozli ustrezajo atributom, veje iz vozla ustrezajo posameznim vrednostim atributa v vozlu in listi drevesa ustrezajo razredom. Zgled odločitvenega drevesa je na sliki 3.2.



Slika 3.2 Odločitveno drevo, po katerem se odločamo, če bomo vzeli dežnik.

Problem učenja je definiran enako kot v poglavju 1.4.1. Osnovni algoritem gradnje odločitvenega drevesa je sleden:

če vsi primeri spadajo v isti razred,
 potem postavi list s tem razredom,
 drugače
  1. izberi za vozel najbolj informativen atribut.
  2. razbij množico primerov v vozlu po posameznih vr. atributa v disjunktne podmnožice.
  3. za vsako podmnožico ponovi celoten algoritem.

Algoritem rekurzivno gradi drevo. Bistvo algoritma je izbira najbolj informativnega atributa. Kriterij temelji na naslednji izpeljavi: Potrebna količina informacije za klasifikacijo enega primera je enaka:

$$I = \sum_i P_i * \log_2 (P_i)$$

pri čemer je P_i apriorna verjetnost, da poljuben primer spada v i-ti razred (te verjetnosti lahko aproksimiramo z relativnimi frekvencami iz učne množice). Če za vrh drevesa uporabimo atribut A z V različnimi vrednostmi, je nova potrebna količina informacije za klasifikacijo enega primera enaka:

$$I(A) = \sum_v^V P_v * \sum_i (P_{vi}/P_v) * \log_2 (P_{vi}/P_v)$$

pri čemer je P_v apriorna verjetnost, da ima poljuben primer v-to vrednost atributa A in P_{vi} verjetnost, da ima poljuben primer v-to vrednost atributa A in da pripada i-temu razredu. Najboljši atribut je tisti, ki minimizira funkcijo I(A), ker od njega dobimo največ informacije.

ID3 se je izkazal kot zelo učinkovit. Quinlan (83a) ga je uporabljal za generacijo pravil za določanje izgubljenih pozicij v šahovskih končnicah.

3.3.2 ASISTENT

Na Fakulteti za elektrotehniko v Ljubljani smo iz Quinlanovega ID3 z mnogimi izpopolnitvami razvili sistem ASISTENT (Kononenko in sod. 83b,84,84a) in smo ga uspešno preizkusili v 6 različnih medicinskih domenah. Dobljeni rezultati (diagnostična natančnost) v primerjavi z zdravniki specialisti so podani v tabeli 3.1, v primerjavi z nekaterimi statističnimi metodami pa v tabeli 1.1.

domena	ASISTENT	št. vozlov	ZDRAVNIKI
primarni tumor	46%	35	42%
rak na dojki	72%	16	64%
hepatitis	80%	17	?
limfografija	65%	14	60-85%(ocena)
inkontinenca m.	67%	107	?
inkont. ženske	81%	174	?

Tabela 3.1 Primerjava dosežene diagnostične natančnosti sistema ASISTENT in zdravnikov specialistov (? pomeni, da ne poznamo natančnosti zdravnikov v ustrezni domeni).

Ob teh rezultatih bi mogoče kdo pomislil, da računalnik lahko nadomesti človeka, kar je zmotno mnenje. Računalnik ne bo in ne more izpodriniti zdravnika, lahko pa mu pomaga, da svoje delo opravlja hitreje, lažje in natančneje. Podrobneje bomo o ASISTENTu in poskusih v medicini pisali v eni od naslednjih številk te revije. Tu so le na kratko opisane značilnosti ASISTENTa (ki je splošen sistem, le poskuse smo do sedaj delali samo na podatkih iz medicine).

Prve poskuse z uporabo sistema ID3 v medicinski diagnostiki sta napravila Ivan Bratko in Peter Mulec (80, glej tudi Mulec 80). Rezultati so bili obetavni, zato smo nadaljevali z razvojem sistema. ASISTENT se razlikuje od ID3 v grobem v naslednjih značilnostih:

(a) ASISTENT uporablja binarno gradnjo: vsak atribut postane binaren, tako da se vrednosti grupirajo v dve disjunktni podmnožici, ki maksimizirata njegovo informativnost. Dobljena drevesa so manjša in imajo večjo klasifikacijsko natančnost (večji je efekt generalizacije nad učnimi primeri). Najnovejša raziskovanja Rossa Quinlana (85) so bila pogojena z rezultati naših raziskav in nakazujejo dodatne izboljšave h gradnji odločitvenih dreves.

(b) ASISTENT lahko uporablja nepopolne podatke: tam, kjer manjka vrednost atributa, se pripiše vsaka možna vrednost za ta atribut z določeno verjetnostjo.

(c) Verjetnostno sklepanje v kombinaciji z Bayesovim verjetnostnim principom omogoča ASISTENTu razrešiti konfliktne situacije.

(d) Rezanje nezanesljivih delov drevesa po principu maksimalne klasifikacijske natančnosti omogoča ASISTENTu, da se izogne slabostim ocenitvene funkcije nad majhnimi množicami primerov v vozlu.

ASISTENT je implementiran v PASCALu (cca 5000 vrstic) in ga poganjamo na računalniku DEC-10. Za gradnjo dreves porabi tipično nekaj sekund CPU za nekaj sto primerov in nekaj minut za nekaj tisoč učnih primerov. Zgrajena drevesa so razumljiva, lahko se uporabljajo brez računalnika, za uporabnika pa so zanimiva, ker iz njih lahko razbere določene relacije in zakonitosti iz svoje domene.

3.3.3 ACLS

ACLS je sistem, ki sta ga iz Quinlanovega ID3 razvila Paterson in Niblett (82) in je že doživel komercialno inačico pod imenom Expert-Ease. Na osnovi ACLS je Niblett (84) razvil sistem CLEAR, ki je programsko orodje za olajšavo sestavljanja in vzdrževanja baz znanja za ekspertne sisteme. CLEAR generira if-then pravila na osnovi primerov. Algoritem je zelo podoben osnovnemu algoritmu ID3, le da uporablja tako imenovano strukturno indukcijo.

Alen Shapiro (83,84) je iz ID3 razvil sistem z imenom Interactive-ID3, ki omogoča pomoč pri strukturiranju uporabnikove domene v dialogu z uporabnikom, gradnjo linearnih odločitvenih dreves ter avtomatsko generacijo razlage rezultatov sklepanja po strukturirano zgrajenem drevesu v skoraj naravnem jeziku. S problemi osnovnega algoritma gradnje odločitvenih dreves se ukvarjata še O'Keefe (83) in Shepherd(83). O'Keefe predlaga alternativo, ki je v ASISTENTu že preizkušena in privzeta. Shepherd je delal poskuse za primerjavo splošnih in binarnih odločitvenih dreves.

3.4 METODA ZVEZD

Ryszard Michalski (83b) je razvil splošno metodo za strukturno avtomatsko učenje na osnovi primerov, metodo zvezd. Zvezda, ki pripada pozitivnemu učnemu primeru pri dani množici negativnih učnih primerov, je definirana kot množica maksimalno splošnih konjuktivnih izrazov, ki pokrivajo dani pozitivni primer in ne pokrivajo nobenega negativnega primera iz dane množice. Tako je npr. zvezda, ki pripada pozitivnemu primeru;

leti & nese_jajca & ima_perje

pri množici negativnih primerov;

{leti & ¬nese_jajca & ima_perje, ¬leti & nese_jajca & ¬ima_perje}

enaka množici opisov;

{ leti & nese_jajca, ima_perje }

3.4.1 ALGORITEM METODE ZVEZD

Algoritem je v grobem takle:

1. Naključno izberi pozitivni učni primer.
2. Sestavi zvezdo za dani učni primer pri množici vseh negativnih učnih primerov.
3. Poišči najboljši opis iz zvezde po vnaprej podanem kriteriju in ga postavi v množico končnih opisov.
4. če množica končnih opisov pokriva vse pozitivne učne primere, potem končaj,
   drugače iz množice učnih primerov izloči pozitivne primere, ki jih množica končnih opisov pokriva in ponovi celoten algoritem.

Kot opisni jezik uporablja Michalski večvrednostno logiko (VL1 - variable valued logic 1), katere glavna moč je interna disjunkcija in dovoljuje npr. izraze oblike;

velikost = 150..180 & teža > 60 & barva_las = (črna V rjava).

V algoritmu sta 2. in 3. točka nedorečeni. 2. točka dovoljuje poljubno metodo za generiranje zvezd. Michalski uporablja metodo INDUCE, ki je opisana v nadaljevanju. Za 3. točko algoritma potrebujemo kriterij za ocenjevanje opisov. Michalski je v ta namen razvil metodo ocenjevanja LEF, ki je opisana v poglavju 3.4.3. V poglavju 3.4.4 je opisana uporaba metode zvezd.

3.4.2 METODA "INDUCE" ZA GENERACIJO ZVEZD

Diettrich in Michalski (81,83) sta razvila metodo za generacijo zvezd (glej definicijo na začetku pogl. 3.4). Metodo lahko v grobem opišemo z algoritmom:

1. Sestavi množico posameznih atributov M, ki opisujejo dani pozitivni primer e, in jo uredi po kriteriju pokrivanja najmanj negativnih in največ pozitivnih učnih primerov.

2. Najboljšim atributom konjuktivno dodajaj atribute, ki jih dobiš iz primera e z uporabo posplošitvenih pravil na osnovi predznanja iz domene.

3. Konsistentni in kompletni opisi gredo v množico rešitev. če je moč množice večja od željenega števila alternativ, potem končaj.

4. Nekompletne a konsistentne opise postavi v množico CONS. če je moč množice večja od vnaprej določene meje, potem nadaljuj pri točki 7.

5. Vsak preostali izraz iz množice M specializiraj na vse možne načine z dodajanjem atributov iz prvotne množice M.

6. Uredi množico M po kriteriju pokrivanja najmanj negativnih in največ pozitivnih primerov in nadaljuj pri točki 3.

7. Vsak izraz v množici CONS posploši na vse možne načine

8. Izberi najboljše dobljene izraze in jih postavi v množico rešitev ter končaj.

3.4.3 METODA "LEF" ZA OCENJEVANJE OPISOV

LEF (lexicographic evaluation functional) je definiran kot zaporedje parov [kriterij, toleranca], pri čemer je kriterij eden od vnaprej določenih kriterijev ocenjevanja kvalitete opisov, ki jih izbere uporabnik (glej 2.6), toleranca pa prag, izražen v procentih. Ocenjevanje poteka tako, da v prvem koraku zadržimo opise, ki zadovoljujejo 1. kriterij v tolerančni meji, nato 2. kriterij itd. Proces se konča, ko nam ostane samo en (najboljši opis), ali če zmanjka kriterijskih parov. V drugem primeru je rezultat množica opisov, ki so enakovredni glede na izbrani kriterij LEFa.

3.4.4 PRIMERI UPORABE METODE ZVEZD

Metodo so v različnih verzijah implementirali v seriji programov (AQVAL, AQ11, INDUCE1.2, GEM (Dietttrich in Michalski 81,83)). Michalski je eksperimentiral v različnih domenah in povsod dobil zelo dobre rezultate. Z uporabo metode zvezd je zgeneriral bazo znanja za ekspertni sistem za diagnosticiranje sojinih bolezni, ki je prekašal podoben sistem, pri katerem so bazo znanja sestavljali eksperti domene (Michalski in Chilausky 80). Zgradili so sistem ADVISE (Michalski in Baskin 83) kot programsko orodje za razvoj ekspertnih sistemov, ki med drugim vključuje program GEM. S pomočjo sistema ADVISE so uspešno razvili tri ekspertne sisteme.

3.5 KONCEPTUALNO GRUPIRANJE VZORCEV - CLUSTER

Michalski in Stepp (80, 83, 83a) sta razvila sistem za konceptualno grupiranje vzorcev kot alternativo klasičnim metodam za grupiranje vzorcev, ki imajo skupno slabost, da ne podajo opisov zgeneriranih grup (glej pogl. 1.4.2). Opisni jezik v sistemu CLUSTER je večvrednostna logika (glej 3.4.1). Kriterij podobnosti, ki ga CLUSTER uporablja, je tako imenovano merilo konceptualne kohezije (conceptual cohesiveness).

3.5.1 ALGORITEM KONCEPTUALNEGA GRUPIRANJA VZORCEV

Algoritem je analogen algoritmu klasične metode ISODATA za grupiranje vzorcev. Pri tem je število željenih grup (K) podano:

1. Naključno izberi K primerov.

2. Sestavi zvezdo za vsak izbrani primer, tako da vzameš preostale izbrane primere za negativne in vse ostale primere za pozitivne (za definicijo zvezde in algoritem generacije zvezd glej pogl. 3.4).

3. Ustrezno modificiraj dobljene opise iz zvezde tako, da dobiš disjunktno pokritje K izbranih primerov, ki je optimalno glede na vnaprej izbran kriterij LEF (glej 3.4.3).

4. če se v nekaj zadnjih iteracijah rezultat ni izboljšal, potem končaj.

5. če se je kvaliteta grup po kriteriju LEFa izboljšala, potem
  izberi K novih primerov tako, da izbereš centralni primer
  iz vsake grupe (primer, ki najbolj ustreza opisu grupe),
drugače
  izberi K novih primerov tako, da izbereš mejni primer
  iz vsake grupe (primer, ki najmanj sovpada z opisom grupe)

6. Nadaljuj pri točki 2.

CLUSTER ne generira samo enega nivoja grupiranja, ampak celo hierarhijo. Za to uporablja naslednji "glavni" algoritem:

1. Za K = 2 do MAX ponavljaj;
   zgeneriraj K grup in oceni kvaliteto grupacije.
2. Izberi optimalno število grup (K, ki je dal najboljšo grupacijo - ki se najbolj prilega primerom).
3. če se opisi grup bolje prilegajo primerom, kot na prej. hierarhičnem nivoju,
   potem za vsako grupo ponovi celoten algoritem.

Prileganje opisa grupe s primeri iz grupe se meri z relativno redkostjo grupe (kvocient števila primerov, ki jih opis grupe pokriva, a ne nastopajo kot učni primeri, s celotnim številom pokritih primerov).

3.5.2 UPORABA SISTEMA "CLUSTER"

CLUSTER je bil uspešno preizkušen v mnogih domenah. Eden od poskusov je bil grupiranje vzorcev obolelih rastlin. CLUSTER je pravilno grupiral vzorce po posameznih boleznih in je vsako grupo opisal z logičnimi izrazi, ki so se ujemali z opisi strokovnjakov za posamezne bolezni (Michalski in Stepp 83). Sposobnost sistema so primerjali s standardnimi statističnimi metodami za grupiranje vzorcev. Izkazalo se je, da je CLUSTER za dva velikostna razreda počasnejši, vendar so dobljene grupe opisane z logičnimi izrazi, prav tako pa so rezultati grupiranj, ki jih izvede CLUSTER, človeku bolj naravni (opisi brez disjunkcij, Michalski in Stepp 83a). CLUSTER ima kljub počasnosti bistveno prednost na tistih področjih, kjer morajo biti rezultati grupiranja človeku razumljivi. CLUSTER je vključen v programski paket ADVISE (Michalski in Baskin 83), ki je pripomoček za razvoj ekspertnih sistemov.

3.6 NEKATERI ZANIMIVI SISTEMI

3.6.1 BACON

Langley in sod.(83,83a,83b) so razvili sistem BACON za učenje zakonov, ki veljajo med numeričnimi vrednostmi posameznih meritev danega pojava. BACON išče funkcijske zveze med posameznimi spremenljivkami. če najde zanimivo povezavo, tvori aritmetični izraz, ki jo definira. Ta izraz uporablja za nadaljnje učenje. Uporabnik načrtuje eksperimente (izbere spremenljivke in želene povezave), nato pa BACON sam vodi eksperiment. Pri tem uporablja hevristike za iskanje obetavnih teorij, ki opisujejo eksperimentalne rezultate. Z BACONom so eksperimentirali nad podatki, ki so opisovali razne fizikalne in kemične pojave. BACON je uspešno odkril množico osnovnih zakonov iz obeh področij, kot so Ohmov zakon, zakon o ohranitvi momenta, zakon gravitacije, itd.

3.6.2 SPARC/G

Diettrich (80) je razvil sistem SPARC/G za *iskanje pravila*, po katerem se generira zaporedje dogodkov in skuša ugotoviti pri danem končnem zaporedju dogodkov, kakšen bo naslednji dogodek iz zaporedja. Za opisni jezik uporablja večvrednostno logiko (glej 3.4.1). Sistem temelji na 3 algoritmih. Generalizacijski algoritem posplošuje učne primere (sosednje pare iz zaporedja dogodkov). Dekompozicijski algoritem razbije zaporedje dogodkov na pare in generira if-then pravila, ki opisujejo grupacije parov dogodkov. Ta pravila nato kombinira v splošnejša pravila, ki opisujejo posamezne dele zaporedja dogodkov. Periodični algoritem skuša najti periodične zakonitosti v zaporedju dogodkov. S sistemom so eksperimentirali v domeni kvalitativnih fizikalnih zakonov, pri preprostem programiranju robotov in v igri s kartami Eleusis. (Eleusis je zahtevna igra. Opis igre je v grobem: delilec si izmisli pravilo za generiranje zaporedja kart. Igralci ga morajo uganiti tako, da polagajo karte v zaporedje, delilec pa jih opozarja na napake). V igri Eleusis se je SPARC/G izkazal kot enakovreden partner mojstrom v igri.

3.6.3 MODEL INFERENCE SYSTEM - MIS

E.Y. Shapiro (81) je razvil sistem MIS za *učenje teorij*, ki opisujejo konkretna dejstva v neki domeni. Pri tem je dan opisni jezik in prerok (oracle), ki odgovarja na vprašanja, ali je dano dejstvo resnično ali neresnično. Dano je tudi zaporedje resničnih in neresničnih dejstev iz neke domene. MIS uporablja za opisni jezik Hornove stavke, ki ustrezajo programskim stavkom jezika PROLOG (glej npr. Clocksin in Mellish 81). Tako so dejstva iz domene predstavljena kot dejstva v PROLOGu, generirana teorija pa je predstavljena z direktno izvršljivim programom v PROLOGu. Oglejmo si zgled: Imejmo domeno "seznami" in dani jezik:

```
[]              - prazen seznam
[Glava|Rep]     - neprazen seznam, Glava je prvi element,
                  Rep pa preostali del seznama
append(X,Y,Z)   - Z je stik seznamov X in Y
reverse(X,Y)    - Y je obrnjen seznam X
```

Tu je nekaj primerov dejstev:

```
append([], [a], [a])                  je res
append([a,b], [c,d,e], [])            ni res
append([a,b], [c,d,e], [a,b,c,d,e])   je res
reverse([a], [b,a])                   ni res
reverse([a,b,c], [c,b,a])             je res
```

Teorija, ki jo je sestavil MIS v 17 sekundah iz 56 dejstev, je definicija predikatov append in reverse v PROLOGu:

```
append([], X, X)
append([A|X], Y, [A|Z]) <-- append(X, Y, Z)

reverse([], [])
reverse([A|X], Y)  <-- reverse(X,Z) & append(Z, [A], Y)
```

Poleg tega je MIS izpeljal še vrsto drugih teorij (različne aksiomatizacije regularnih množic, razne programe za operacije nad seznami, pravilo seštevanja in množenja celih števil, itd.). Omejitvi pri MISu sta dve: (a) opisni jezik je fiksen, in (b) MIS nikoli ne ve, ali je njegova teorija pravilna (kar pa ni nič presenetljivega, saj nad končno množico dejstev ni možno izpeljati teorije za opisovanje neskončno mnogo dejstev, za katero bi se dalo dokazati absolutno pravilnost). Nekoliko več o algoritmu učenja je opisano v (Bratko 81). Zanimivo je, da je MIS sposoben avtomatično odkrivati in odpravljati napake v PROLOGovih programih, če ima na razpolago prej opisanega preroka.

3.6.4 AUTOMATIC MATHEMATICIAN - AM

Lenat (83) je razvil *teorijo hevristik*, ki jo je podkrepil s sistemom AM. AM je sistem za *samostojno odkrivanje* teorij in zanimivih konceptov v dani domeni. AM začne z množico osnovnih aksiomov (npr. iz teorije množic) in z množico hevristik, ki vsebujejo znanje o principih znanstvenega raziskovanja, o načinih ocenjevanja pomembnosti izpeljanih konceptov in o strategijah za izbiro poti, ki vodi do zanimivih zaključkov. Nato pa samostojno odkriva in išče zanimive povezave in koncepte v dani domeni. Lenat definira metodologijo raziskovanja s petimi točkami takole:

1. Novo znanje lahko razvijemo s pomočjo hevristik.

2. Z novim znanjem potrebujemo nove hevristike.

3. Nove hevristike lahko razvijemo s pomočjo hevristik.

4. Z novim znanjem potrebujemo nove predstavitve znanja.

5. Nove predstavitve znanja lahko razvijemo s pomočjo hevristik.

AM je uspešno opravil 1. točko zgornjega opisa. Iz osnovnih konceptov iz teorije množic (definicija množice, seznama, unije, kompozicije itd.) in 243 hevristik je AM hitro odkril koncept števila, seštevanja, odštevanja, množenja, deljenja, potence, praštevila, itd. Podobne uspehe je imel na področju ravninske geometrije. Ko pa je njegovo raziskovanje izčrpalo ozko področje domene, AM ni bil več zmožen nadaljevati raziskovanja in nadgraditi pridobljenega znanja. Vzrok je v točkah 2 in 4 v zgornjem opisu. Njegove hevristike niso več zadostovale za uspešno raziskovanje, ker niso upoštevale na novo pridobljenega znanja. Lenat je nakazal možno avtomatizacijo izvajanja 3. in 5. točke v zgornjem opisu in njegova teorija hevristik je podlaga za nov sistem, ki ga je imenoval EURISKO.

4. ZAKLJUČEK

Kot alternativa klasičnemu pristopu k razpoznavanju in grupiranju vzorcev je *strukturno* avtomatsko učenje dokazalo svoje prednosti, vsaj na določenih področjih, kjer je potrebno iz rezultatov učenja poiskati logične povezave med danimi dejstvi, na osnovi katerih lahko sklepamo na vzroke in posledice. Rezultati učenja so uporabniku *razumljivi* in lahko prispevajo določena znanja k domeni. Zato ima strukturno avtomatsko učenje prednost na vseh področjih, kjer bo uporabnik rezultate sprejel le, če jih bo računalnik lahko obrazložil in argumentiral. Na osnovi rezultatov bo uporabnik lahko napravil določene zaključke, zato mora rezultatom učenja zaupati. To je tudi osnovno vodilo pri razvoju ekspertnih sistemov.

Strukturno avtomatsko učenje ne bo in ne more izpodriniti človeka, le njegovo delo bo s pomočjo računalnika lažje, hitrejše in natančnejše, kar se kaže z naglim razvojem metodologije *ekspertnih sistemov*. Prav pri razvoju ekspertnih sistemov se je pokazala potreba po hitrem sestavljanju baz znanja in to *ozko grlo* lahko premostimo z avtomatskim učenjem pravil na osnovi primerov (Bratko in sod. 85). Namesto dolgotrajnega zbiranja pravil od ekspertov in iz strokovne literature preprosto zberemo arhivske podatke o delu strokovnjaka in jih uporabimo za avtomatsko učenje pravil, ki bodo oponašala (in tudi izboljšala) delo ekspertov. Tako sta Michalski in Chilausky (80) napravila poskus z avtomatskim generiranjem baze znanja za ekspertni sistem in nato isti poskus s pridobivanjem znanja od ekspertov iz domene. Prvi način je bil neprimerno hitrejši in je dal tudi boljše rezultate!

Na področju učenja klasifikacijskih pravil se je ASISTENT pokazal za enakovrednega standardnim statističnim metodam glede *natančnosti* razvrščanja (glej tabelo 1.1). V

poskusih v medicinski diagnostiki je vedno dosegel klasifikacijsko natančnost zdravnikov specialistov (glej tabelo 3.1). Bistvena prednost pred statističnimi metodami je v razumljivosti odločitvenega drevesa. Odločitveno drevo se lahko uporablja brez računalnika, npr. kot priročnik za klasificiranje (diagnosticiranje). Iz drevesa je možno razbrati relacije in zakonitosti iz domene.

Na področju grupiranja vzorcev je CLUSTER pokazal dobre rezultate. Nekateri mu kljub razumljivim opisom grup in naravnejšemu grupiranju očitajo, da upočasnitev za 2 velikostna razreda ni sprejemljiva (Dale 85). Kako pa naj bo sprejemljivo naslednje dejstvo: Dr. Jure Zupan iz Kemijskega inštituta Boris Kidrič pravi, da je hierarhično drevo, dobljeno s statistično metodo (Zupan 82), na nekem kemijskem problemu, ki je obsegalo približno 1500 vozlov (ki seveda niso bili opisani), pregledovala in označevala skupina strokovnjakov celi dve leti, da je drevo postalo uporabno! Torej za določene probleme je CLUSTER idealen, saj je delo računalnika glede na človeško delo vsak dan cenejše. Michalski in Stepp (79,83) pa sta v svojih poskusih nevede nakazala še eno alternativo. Za primerjavo CLUSTERja s standardnimi statističnimi metodami sta dobljene grupacije statističnih metod avtomatsko opisala s pomočjo sistema za avtomatsko učenje po metodi zvezd, AQ11. Torej lahko dobimo avtomatsko opisane grupe tudi z uporabo statističnih metod za grupiranje in nato s pomočjo metod za strukturno avtomatsko učenje na osnovi primerov lahko opišemo posamezne grupe.

Strukturno avtomatsko učenje je mlada panoga in je v naglem razvoju. Teoretična osnova še ni zaokrožena in se še vedno gradi. Na voljo je nekaj splošnih formalizmov, kot so prostor verzij, metoda zvezd, gradnja odločitvenih dreves in konceptualno grupiranje vzorcev. Nekateri sistemi so dokazali svojo moč in uporabnost na celi vrsti problemov in so zreli za rutinsko uporabo (GEM, ACLS, ASISTENT, CLUSTER). Zaradi različnih pristopov je težko najti skupne mehanizme. Prvi korak k temu je napravil Ryszard Michalski (83b).

Vsi današnji uporabni sistemi za strukturno avtomatsko učenje se učijo na osnovi primerov. Opisni jezik je fiksen, brez možnosti razširitve z dodajanjem novih konceptov in brez možnosti spreminjanja "taktike" učenja *na osnovi pridobljenega znanja*. Nekaj poskusov izogniti se tej omejitvi je že bilo (Lenatov AM, Mitchellov LEX), vendar vedno precej neučinkovitih in strogo problemsko orientiranih. Ali to pomeni, da človeško učenje, čeprav je videti počasno in neučinkovito, le ni tako zelo slabo (neoptimalno)? Mogoče bodo bodoče raziskave odgovorile na to vprašanje.

Na koncu še nekaj o literaturi. V tem prispevku je podan splošen pregled literature o strukturnem avtomatskem učenju v svetu. Dobra osnova za vpeljavo v področje avtomatskega učenja je delo (Michalski, Carbonell, Mitchell 83), ki je zbirka prispevkov vrste vidnih avtorjev. Na nekatere od njih se sklicujemo v tem prispevku. 1. in 2. poglavje (Carbonell in sod. 83, Simon 83) podajata splošne poglede na probleme avtomatskega učenja, 3. in 4. poglavje (Diettrich in Michalski 83, Michalski 83b) pa prikazujeta osnovne principe strukturnega avtomatskega učenja, ki so pregledno opisani v 2. poglavju tega prispevka. Utgoff in Nudel (83) v istem delu podajata strukturirano in obrazloženo bibliografijo avtomatskega učenja, ki obsega blizu 600 referenc.

ZAHVALA

Ivan Bratko, moj učitelj in sodelavec, ki me je vpeljal v področje avtomatskega učenja, je z nasveti in pripombami veliko pripomogel k nastanku tega dela. Slavistka Irena Roglić-Kononenko je z izčrpno lekturo prispevala k boljšemu izražanju, kar ni ravno moja vrlina. Zahvaljujem se Tanji Majaron za natančen pregled rokopisa in za številne pripombe, ki so pripomogle k boljši razumljivosti tega prispevka. Zahvaljujem se tudi Antonu Železnikarju za pripombe k osnutku prispevka.

LITERATURA

I.Bratko (1981) Sistemi za strukturno avtomatsko učenje, Delovno poročilo, Fakulteta za elektrotehniko, Ljubljana

I.Bratko (1982) Inteligentni informacijski sistemi, skripta, Univerza Edvarda Kardelja v Ljubljani, Fakulteta za elektrotehniko

I.Bratko, P.Mulec (1980) An experiment in automatic learning of diagnostic rules, Informatica 4/4

I.Bratko, I.Kononenko, N.Lavrač, I.Mozetič, E.Roškar (1985) Automatic synthesis of knowledge, Automatika, Zagreb (v tisku)

Buchanan, B.G, Feigenbaum, E.A. (1978) DENDRAL and Meta-DENDRAL: their applications dimensions, Artificial Intelligence, vol.11, pp.5-24

A.Bundy (1981) The Winston-Plotkin-Young-Linz learning program, Prolog program library, Dep. of Artificial Intelligence, University of Edinburgh

Carbonell, J.G, Michalski, R.S, Mitchell, T.M (1983) An overview of machine learning, Machine Learning: an Artificial Intelligence Approach (Michalski, Carbonell, Mitchell, eds.), Palo Alto: Tioga Pub. Com.

Clocksin, W.F., Mellish, C.S. (1981) Programming in PROLOG, Springer Verlag

Dale, M.B. (1985) On the Comparison of Conceptual Clustering and Numerical Taxonomy, IEEE Transactions on pattern analysis and machine intelligence, vol.PAMI-7, no.2

Diettrich, T.G (1980) The methodology of knowledge layers for inducing descriptions of sequentially ordered events, M.S.Thesis and report No.1024, Dep. of Computer Science, University of Illinois, Urbana

Diettrich, T.G, Michalski, R.S. (1981) Inductive learning of structural descriptions: Evaluation criteria and comparative review of Selected Methods, Artificial Intelligence, vol.16, no.3

Diettrich, T.G, Michalski, R.S. (1983) A comparative review of selected methods for learning from examples, Machine Learning: an Artificial Intelligence Approach (Michalski, Carbonell, Mitchell, eds.), Palo Alto: Tioga Pub. Com.

I.Kononenko (1983) Priročnik za uporabo Winstonovega sistema za učenje konceptov iz sveta kock na osnovi primerov, Delovno poročilo, Fakulteta za elektrotehniko, Ljubljana

I.Kononenko (1983a) Popravki Winstonovega sistema za učenje konceptov iz sveta kock na osnovi učnih primerov, Delovno poročilo 4, Fakulteta za elektrotehniko, Ljubljana

I.Kononenko, I.Bratko, M.Zwitter (1983b) Eksperimenti z avtomatskim učenjem medicinskih diagnostičnih pravil, Referat v zborniku del: VII.Bosanskohercegovački simpozijum iz informatike, Jahorina 83

Kononenko, I., Bratko. I. Roškar, E.(1984) Experiments in automatic learning of medical diagnostic rules. ISSEK Workshop 84, Bled.

I.Kononenko, I.Bratko, M.Zwitter (1984a) Poskusi z ASISTENTom v *medicinski diagnostiki in prognostiki*, Referat v zborniku del: VIII.Bosansko- hercegovački simpozijum iz informatike, Jahorina 84

P.Langley, J.M.Zytkow, H.A.Simon, G.L.Bradshaw (1983) Mechanisms for qualitative and quantitative discovery, Machine learning conf., Urbana-Champaign, University of Illinois

P.Langley, J.M.Zytkow, G.L.Bradshaw, H.A.Simon (1983a) Three facets of scientific discovery, IJCAI

P.Langley,G.L.Bradshaw,H.A.Simon (1983b) Rediscovering Chemistry with BACON system, Machine Learning: an Artificial Intelligence Approach (Michalski, Carbonell, Mitchell, eds.). Palo Alto: Tioga Pub. Com.

Lenat, D.B. (1983) The role of heuristics in learning by discovery: Three case studies, Machine Learning: an Artificial Intelligence Approach (Michalski, Carbonell, Mitchell, eds.). Palo Alto: Tioga Pub. Com.

Michalski, R.S. (1983) A theory and methodology of inductive learning. Machine Learning: an Artificial Intelligence Approach (Michalski, Carbonell, Mitchell, eds.). Palo Alto: Tioga Pub. Company (tudi v : Artificial Intelligence 20/1983)

Michalski,R.S., Baskin, A.B. (1983) Integrating multiple knowledge representations and learning capabilities in an expert system : The Advise system, IJCAI

Michalski,R.S., Carbonell,J.G, Mitchell,T.M, eds. (1983) Machine Learning: an Artificial Intelligence Approach Palo Alto: Tioga Pub. Com.

Michalski,R.S, Chilausky, L.R. (1980) Learning by being told and learning from examples: an experimental comparison of two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis, Policy Analysis and Information Systems, Vol.4, no.2, pp. 125-160

Michalski,R.S, Stepp,R.E. (1979) Revealing conceptual structure in data by inductive inference, Expert Systems in the Microelectronic Age (ed. D.Michie) Edinburgh University Press.

Michalski,R.S, Stepp,R.E. (1983) Learning from observation: conceptual clustering, Machine Learning: an Artificial Intelligence Approach, Palo Alto: Tioga Pub. Com.

Michalski,R.S, Stepp,R.E. (1983a) Automated Construction of Classifications: Conceptual Clustering versus Numerical Taxonomy, IEEE Transactions on pattern analysis and machine intelligence, vol PAMI-5, no.4, pp. 396-410

T.M. Mitchell (1978) Version spaces : An approach to concept learning, Ph.D.Thesis, Stanford University

T.M. Mitchell (1983) Learning and problem solving, IJCAI

T.M.Mitchell, P.E.Utgoff, R.Banerji (1983) Learning by experimentation: Acquiring and refining problem-solving heuristics, Machine Learning: an Artificial Intelligence Approach (Michalski, Carbonell, Mitchell, eds.). Palo Alto: Tioga Pub. Com.

P.Mulec (1980) Algoritmi za avtomatsko učenje, diplomsko delo, Univerza Edvarda Kardelja v Ljubljani, Fakulteta za elektrotehniko

T.Niblett (1984) An interactive rule induction system, ISSEK Workshop 84, Bled

Nie,N.H, Hull, C.H, Jenkins, J.G, Steinbrener,K., Bent, D.H. (1975) SPSS - Statistical Package for the Social Sciences, McGraw-Hill

N.J.Nilsson (1965) Learning Machines, McGraw-Hill book com.

N.J.Nilsson (1982) Principles of Artificial Intelligence, Springer Verlag

R.A.O'Keefe (1983) Concept Formation from very large training sets, IJCAI

Paterson A., Niblett T.(1982) ACLS user's manual, Intelligent Terminals Limited

N.Pavešić, F.Mihelič (1981) Statistične metode , skripta, Univerza Edvarda Kardelja v Ljubljani, Fakulteta za elektrotehniko

Quinlan, J.R. (1979) Discovering rules by induction from large collections of examples. Expert Systems in the Microelectronic Age (ed. D.Michie) Edinburgh University Press.

Quinlan, J.R.(1979a) Iterative-Dichotomizer 3 (ID3), report, Stanford University, Artificial Intelligence Laboratory, Computer Science Department, Stanford, California

Quinlan, J.R.(1982) Semi-autonomous acquisition of pattern-based knowledge, Machine Intelligence 10 (eds. J.Hayes, D.Michie, J.H.Pao), Horwood & Wiley

Quinlan, J.R.(1983) Learning from noisy data, Machine lerning conference, Urbana-Champaign, Univ. of Illinois

Quinlan, J.R.(1983a) Learning efficient classification procedures and their application to chess end games, Machine Learning: an Artificial Intelligence Approach (Michalski, Carbonell, Mitchell, eds.). Palo Alto: Tioga Pub.Com.

Quinlan, J.R.(1985) Decision trees and multi-valued attributes, Machine Intelligence 11 Workshop, Glasgow

E.Roškar (1984) Mikroračunalniško zasnovane urodinamske in elektromiografske merilne tehnike za diagnostiko urogenitalnega trakta, Doktorska dizertacija, Univerza Edvarda Kardelja v Ljubljani, Fakulteta za elektrotehniko

Roškar, E. Bratko, I., Kononenko, I., Šuk, M., Abrams, P. (1985) An application of computer assisted mulivariate statistical methods and artificial intelligence to the diagnosis of lower urinary tract disorders, Automatika, Zagreb (v tisku)

A.Shapiro (1983) The role of structured induction in expert systems, Ph.D. Thesis, University of Edinburgh

A.Shapiro, D.Michie (1984) A self-commenting facility for iductively synthesised endgame expertise, ISSEK Workshop 84, Bled

E.Y.Shapiro (1981) Inductive inference of theories from facts, Research report 192, Dep. of Computer Science, Yale University

B.A.Shepherd (1983) An apprasial of a Decision tree approach to image classification, IJCAI

H.A.Simon (1983) Why should machines learn, Machine Learning: an Artificial Intelligence Approach (Michalski, Carbonell, Mitchell, eds.). Palo Alto: Tioga Pub.Com.

M.Soklič (1980) Računalniška diagnostika, Zaključno poročilo, Onkološki inštitut, Ljubljana

P.E.Utgoff (1983) Adjusting Bias in Concept Learning, IJCAI

P.E.Utgoff, B.Nudel (1983) Comprehensive Bibliography of Machine Learning, Machine Learning: an Artificial Intelligence Approach (Michalski, Carbonell, Mitchell, eds.). Palo Alto: Tioga Pub.Com.

P.H.Winston (1975) Learning Structural Descriptions from Examples, The Psychology of Computer Vision (P.H.Winston, ed.), McGraw-Hill

J.Zupan (1982) Clustering of large data sets, Research Studies Press (John wiley), Chichester

M.Zwitter,I.Bratko,I.Kononenko (1983) Rational and Irrational Reservations Against the Use of Computer in Medical Diagnosis and Prognosis, Proc. 3th med. conf. on medic. and biological engeneering, Portorož

INFORMATICA 3/85

KOMUNIKACIJA OPERATERJA Z LOKALNO KONZOLO PERIFERNEGA MIKRORAČUNALNIKA

I. KOMPREJ, D. ČUK,

INSTITUT „JOŽEF STEFAN",
JAMOVA 39, LJUBLJANA

UDK: 681.324

ABSTRACT - This article respresents the project of the software equipment for the operator's communication with the local console. The communication is projected for the peripheral microcomputer of the distributed moremicrocomputer system.

The software equipment is projected for TEM 500 (moremicrocomputer system for the complex energy systems control), that is being developed on the "Jožef Stefan" Institute.

POVZETEK - V članku je opisana programska oprema za komunikacijo operaterja z lokalno konzolo perifernega mikroračunalnika distribuiranega mikroračunalniškega sistema. Programska oprema je zasnovana za TEM 500 /1/, večmikroračunalniški sistem za vodenje kompleksnejših energetskih sistemov, ki je v razvoju na Institutu "Jožef Stefan".

1. UVOD

Računalniško vodenje industrijskih procesov prehaja s centraliziranih na decentralizirane sisteme /2/. To so prostorsko porazdeljeni večmikroračunalniški sistemi. Zgrajeni so okrog komunikacijskega vodila, na katerega je priključena vsaj ena centralna mikroračunalniška postaja ter potrebno število lokalnih mikroračunalniških postaj. Prednost decentraliziranih sistemov je v čimvečji samostojnosti lokalnih mikroračunalniških postaj. Komunikacija s centralnim mikroračunalnikom je zmanjšana na minimum /4/.

Termoenergetski večmikroračunalniški sistem TEM 500, ki je v razvoju na Institutu "Jožef Stefan", je sistem za distribuiran način vodenja procesov. Na skupnem komunikacijskem vodilu so priključene centralne in lokalne mikroračunalniške postaje.

Eden od tipov lokalnih mikroračunalniških postaj na sistemu TEM 500 je mikroračunalnik za digitalno procesno upravljanje DPU 051. DPU 051 je namenjen za zbiranje analognih in digitalnih signalov iz procesa, za regulacije ter za daljinsko upravljanje procesa. Programsko opremo predstavlja paket IDR (interpretativni digitalni regulator) /5/. Ker bi bilo podrobno predstavljanje vsega programskega paketa preobsežno, bo na tem mestu IDR le okvirno predstavljen.

IDR je blokovni, problemsko orientiran jezik. Sintakso jezika predstavljajo bloki, katerih vsak opravlja določeno funkcijo. Bloki, ki opravljajo isto funkcijo, so istega tipa. Bloki se izvajajo z izbranim časovnim intervalom. Bloki, ki se izvajajo z istim časovnim intervalom, so v isti zanki. Periodo izvajanja zanke določa vrsta procesa. IDR vključuje sistemske procedure za zbiranje podatkov, lokalno preobdelavo podatkov, regulacijske algoritme ter izvaja daljinske komande, ki jih zahteva centralna mikroračunalniška postaja (CMP) ali operater preko lokalne konzole. Preko lokalne konzole lahko operater v precejšnji meri vodi proces /6/, ne more pa sestavljati novih algoritmov ali v že narejene algoritme dodajati nove bloke (konfiguracija sistema). Operater komunicira z lokalno konzolo preko procesno prilagojene tastature s pomočjo programa IDRCON, ki je inštaliran na DPU 051. Zasnova paketa IDR in programa IDRCON, ki ga predstavlja ta članek, je podrobneje opisana v /3/.

2. FUNKCIONALNI OPIS

2.1. Namen programa IDRCON

Sestavni del programskega paketa IDR je program za komunikacijo operaterja z lokalno konzolo IDRCON. Program IDRCON simulira delovanje terminala. Vsak ukaz se sintaktično preveri. Pravilno vpisanim ukazom se doda funkcija in vsi potrebni podatki, da je notranja oblika ukaza, poslanega iz lokalne konzole enaka obliki ukaza, ki ga pošlje centralna mikroračunalniška postaja distribuiranega sistema TEM 500 preko serijskega komunikacijskega vodila.

Operater ima na voljo procesno prilagojeno tastaturo in 24-znakovni prikaz, ki simulirata delovanje tipkovnice in terminala.

7	8	9	del
4	5	6	,
1	2	3	exp
•	0	–	enter

MODE	BLOCK VALUE	BLOCK DISPLAY	PARAM
SP	MAN OUTPUT	BLOCK SHAPE	LOOP SHAPE
ACT NONACT	CAS LOC SUP	AUT MAN	▲
ALARM ACK	MEMORY	TIMER	▼

Čelna plošča DPU 051.

2.2. Ukazi programa IDRCON.

Program IDRCON obdeluje naslednje ukaze operaterja:

MODE: Spreminjanje načina delovanja regulacijskih in izhodnih blokov. Regulacijski bloki se lahko izvajajo v treh načinih delovanja (kaskadni, lokalni in nadzorni način). Izhodni bloki se lahko izvajajo v dveh načinih delovanja (avtomatski in ročni način). Operater izbira način delovanja s tipkama CAS/LOC/SUP in AVT/MAN .

BLOCK VALUE: Izpis trenutne izhodne vrednosti bloka. Operater lahko pogleda izhodno vrednost vsakega bloka z ukazom BLOCK VALUE. Če je vrednost alarmna ali opozorilna, se tudi to izpiše na konzoli. Meje za opozorilo in alarme so zapisane v izhodnem bloku. Operater s tem ukazom ne more spreminjati izhodne vrednosti bloka.

BLOCK DISPLAY: Sproten prikaz izhodnih vrednosti vseh blokov ob vsakem izvajanju zanke ni mogoč in je tudi nepotreben. Operater izbere nekaj blokov in z ukazom BLOCK DISPLAY določi izpis trenutne izhodne vrednosti teh blokov na konzolo ob večjih spremembah. Velikost spremembe je za različne bloke različna in je kot parameter vpisana v bloku.

PARAM: Spreminjanje parametrov bloka. Operater ima dostop do vseh parametrov, ki so v bloku in jih lahko tudi spreminja. Dostop do parametrov je zaradi enostavnosti komunikacije zaporeden.

SP: Spreminjanje željene vrednosti regulacijskih blokov. Regulacijski bloki izvajajo regulacijske algoritme. Krmilne veličine prirejajo tako, da se krmiljena veličina čimbolj približa željeni vrednosti. Operater lahko preko lokalne konzole spremeni željeno vrednost (SP). Željena vrednost, ki jo poda operater preko lokalne konzole, je lokalna željena vrednost. Program IDRCON ob sprejemu nove željene vrednosti hkrati postavi način delovanja regulacijskega bloka (MODE) na lokalni način (LOC).

MAN OUTPUT: Ročno nastavljanje izhodne vrednosti. Novo izhodno vrednost lahko operater povečuje ali zmanjšuje stopenjsko s tipkama ▲ in ▼ . Vsak pritisk na tipko pomeni 1 % spremembe izhodne vrednosti. Vrednost je omejena med 0 % in 100 % . Druga možnost je, da operater direktno vpiše novo izhodno vrednost.

BLOCK STATUS in LOOP STATUS sta ukaza za spreminjanje aktivnosti oz. neaktivnosti zanke.

ALARM ACK je ukaz za potrditev sprejema alarma oz. obvestila. Ukaz ima zvočni in svetlobni signal, ki opozori, da je vrednost nekega bloka alarmna ali v mejah opozoril. Če je hkrati več vrednosti alarmnih oz. v mejah opozoril, se te izpišejo po vrsti, kot so prišle, vendar vsaka čaka na potrditev prejšnje. Alarmi imajo višjo prioriteto kot obvestila in se zato najprej izpišejo vsi alarmi, nato pa obvestila. Najnižjo prioriteto ima izpis vrednosti bloka, ki je bil v alarmu oz. mejah obvestil, pa je zopet zavzel normalno vrednost. Izpis normalne vrednosti nima potrditvenega ukaza in se izpiše le, če je konzola prosta.

3. ZAKLJUČEK

Distribuirano procesiranje, ki je prilagojeno zahtevam vodenja kompleksnih procesov v realnem času, omogoča izvedbo sistemov, ki se približujejo teoretično idealnemu modelu.

Ti sistemi imajo za prakso pomembne lastnosti:
- modularnost, fleksibilnost in transparentnost ,
- enostavno testiranje, zanesljivost,
- možnost postopnega dograjevanja brez vpliva na že zgrajene dele sistema,
- enostavno vzdrževanje.

Pomembna kvaliteta distribuiranega sistema vodenja je v lokalni avtonomiji posameznih mikroprocesorskih postaj. Temu primerno je prirejena tudi programska oprema za posamezno lokalno mikroračunalniško postajo.

Komunikacijo operaterja z lokalno konzolo je mogoče izvesti na več načinov, odvisno od velikosti sistema in od dejstva, ali je potrebna vsakodnevna ali občasna komunikacija.

Komunikacija operaterja z lokalno konzolo je potrebna v neposredni bližini procesa. Zato morata biti prikaz in tastatura odporna proti vplivom industrijskega okolja.

Ker je lažje zaščititi nekaj-znakovni prikaz in majhno tastaturo, je na sistemu TEM 500 komunikacija z lokalno konzolo narejena s stalno, procesno prilagojeno tastaturo in 24-znakovnim prikazom. Ta izvedba je cenejša in priročnejša od izvedbe s terminalom in alfanumerično tastaturo, je pa prikrajšana za nazornejši prikaz in obširnejšo komunikacijo.

LITERATURA

/1/ Vidmar, M., D. Cuk: Structure of a Distributed Control System; IAESTED Second International Symposium APPLIED INFORMATICS, AI 84, Innsbruck 1984.

/2/ Damsker D.: Totally Distributed, Redundant Structured Hardware and Software Local Computer Control Network; IEEE Transactions on Power Apparatus and Systems, Vol. PAS-102, No. 1, January 1984.

/3/ Komprej I.: Komunikacija operaterja z lokalno konzolo; Diplomska naloga št. 3653/84, Fakulteta za elektrotehniko Univerze v Ljubljani, 1984.

/4/ Panić N. in ostali: Programska oprema distribuiranih sistemov vodenja procesov; INFORMATICA, 2/3, 1983.

/5/ Žmuc J. in ostali: Zasnova programske opreme za direktno digitalno vodenje procesov; Mikroračunala u sistemima procesnog upravljanja, MIPRO 1984, 3-107.

/6/ Morris H.M.: Stand-Alone Process Controllers Offer Increasing System Capability; Control Engineering, Februar 1983, 78-80.

INFORMATICA 3/85

ELEMENTI ARHITEKTURE RASPODIJELJENOG SISTEMA ZA RAD U STVARNOM VREMENU

KUKRIKA MILAN

ELEKTROTEHNIČKI FAKULTET BANJA LUKA

UDK: 681.3:007

SAŽETAK - U radu su ispitane neke osnovne odlike arhitektura raspodijeljenih sistema pogodnih za rad u stvarnom vremenu. Naglašavajući pitanja topologije i mehanizama upravljanja formuliran je skup principa projektiranja koje utiču na performanse u stvarnom vremenu. Zatim je skicirana konkretna arhitektura koja zadovoljava predložene principe, te definiran pristup rasporedjivanju zadataka u njoj.

ABSTRACT - ELEMENTS OF REAL-TIME DISTRIBUTED SYSTEM ARCHITECTURE. The paper examines some basic architectural characteristics of local computer networks suitable for fast process control. Emphazing the issues of topology and control mechanism, a set of design principles affecting real-time performance is defined. A specific architecture hich satisfies the proposed principles is outlined.

1. UVOD

Iako se elektronička računala koriste za vodjenje složenih procesa već u ranim šezdesetim godinama njihova primjena bila je ograničena visokom cijenom, te dugotrajnom realizacijom i tehničkom manjkavostima. Razvoj moderne tehnologije je temeljito izmijenio situaciju. Sklopovska podrška procesnih računala postala je ekonomična i pristupačna i za najmanje primjene.

U mnogim primjenama raspodijeljenih sistema za rad u stvarnom vremenu naglasak je na koordiniranom upravljanju prostorno raspodijeljenim procesima koji čine jedinstvenu tehnološku cjelinu. Da bi se dostigle željene performanse pri vodjenju brzih procesa neophodno je sagledati prirodu odgovarajućih ograničenja u projektiranju svakog pojedinog nivoa cjelokupnog raspodijeljenog sistema. U ovom radu razmotrena su ograničenja koja se odnose na izbor topologije, projektiranje osnovnih elemenata arhitekture sistema, te algoritama za rasporedjivanje zadataka u odabranoj konfiguraciji.

2. PRINCIPI PROJEKTIRANJA

Prilikom bilo kakvog projektiranja sistema kojim direktno ili indirektno upravlja računalo mora se poći od analize funkcija sistema, u cilju dobijanja akcija i opisa mogućih problema na koje se nailazi u toku rada. Na osnovu njih stručnjak za računala (odnosno ekipa) razradjuje odgovarajući računarski sistem, izradjuje dijagrame toka, formira algoritme i procedure, te prilazi konkretnoj realizaciji. Ukratko, obavlja se sinteza potrebne sklopovske i programske podrške.

Kada ovdje govorimo o dominantnim parametrima projekta, onda svakako podrazumijevamo da je rad sistema ili procesa u potpunosti definiran u slučaju normalnog rada. Medjutim, nastanak konfliktne situacije upućuje da je to tek početak pravog posla na projektu. Tada se cijela slika u potpunosti mijenja, ponekad i suštinski. O čemu se radi?

Pretpostavlja se da projekt nastaje kao rezultat timskog rada stručnjaka za odredjenu specijalnost (strojarstvo, hemiju, tehnologiju itd.) i stručnjaka za računala (kako za sklopovsku, tako i za programsku podršku). U ogromnom broju slučajeva kreće se od pretpostavke da je upravljani proces optimalno organiziran i da teče po jedino ispravnom redoslijedu. Regularni uvjeti rada podrazumijevaju specifikaciju niza parametara od kojih su neki važniji, a neki su u drugom planu. Prema tome se ravna i projekt. Ovo je sasvim u redu ali samo do nastanka jednog ili više problema: poremećaji koji djeluju na sistem upravljanja najčešće spadaju u skupinu nepotpuno poznatih fenomena. Kompletno poznavanje poremećaja u intervalu $-\infty < t_0 < t$, gdje je t_0 tekući moment vremena ne znači i da se njihov daljnji tok može sa izvjesnošću predvidjeti. Nepotpuna informacija o poremećajima zahtijevat će neodredjenost parametara pod čim se podrazumijeva:

- da se mogu mijenjati parametri upravljačkog dijela sistema u skladu sa dopunskom informacijom o parametrima objekta upravljanja;
- da se podešava struktura upravljačkog dijela sistema tako da odgovori promjenama strukture objekta upravljanja;
- da se modificiraju cilj ili kriterij u skladu sa dopunskom informacijom o okolini i ulozi promatranog sistema u njoj;
- da se mijenjaju ograničenja bilo zbog promjene raspoloživih sredstava, ili zbog dopunske informacije o

mogućim graničnim vrijednostima stanja.

Razmotrimo najprije neke osnovne principe projektiranja koji proističu iz zahtjeva da raspodijeljeni sistem efikasno vodi procese u stvarnom vremenu. Podrazumijevamo da su procesi dio jedinstvene primjene i da zahtijevaju tako brz odziv da čak i vrijeme prenosa kratke, hitne poruke kroz mrežu nije zanemarljivo. Takodjer podrazumijevamo da su sva računala u interakciji sa vanjskim svijetom.

Upravljanje u sistemu može biti više ili manje centralizirano ili decentralizirano.

Centralizirano upravljanje predstavlja izdvajanje upravljačkih računala višeg nivoa u vidu posebnih dijelova sistema na višem položaju od upravljačkih računala nižeg nivoa. Funkcionalno specifična podjela rada smislena je samo u horizontali kao razdvajanje različitih podzadataka istog ranga, dok se zbog integracije funkcija i zajedničke baze podataka uvode računala višeg ranga i upravljanje dijeli po nivoima. Vrhovna instanca mora predstavljati svrhu sistema. Prva ispod nje pruža joj sredstva za tu svrhu koja se fiksiraju za nju kao podsvrha, za koju se dolje niže moraju osigurati dstva itd. sve dok se ne dodje do dna hijerarhije. Formalna shema podjele posla je od značaja za pitanje odredjivanja uzroka koji su izazvali kritično stanje sistema i na kojem nivou hijerarhije se o prevladavanju krize može odlučivati.

Ukoliko dozvolimo da sa porastom nivoa raste i odgovornost za donošenje odluke i koordinaciju podredjenih računala, cijeli sistem je ovisan o mogućnostima i trenutnom stanju vodećeg računala i podložan je komunikacijskom kašnjenju kod pristupa tom računalu. Kašnjenje se povećava sa medjusobnom udaljenošću računala, te može predstavljati ozbiljan problem. Dakle, takva organizacija ispoljava velike slabosti kad svako od računala na višem nivou postane kritično za odvijanje procesa i kada dominantno postaje objedinjavanje funkcija. Kod ovakvog pristupa teško je postići da funkcije onesposobljenog računala na višem nivou preuzme neko drugo računalo.

Dakle, pojedina računala moraju posjedovati izvjesnu autonomiju. Prvo, računalo mora biti u stanju da u hitnim slučajevima samostalno reagira na spoljne događaje. Drugo, mora mu se omogućiti da šalje hitne poruke kroz mrežu, potiskujući manje hitne poruke drugih. Treće, mora biti u položaju da odloži, odbije ili ignorira zahtjeve drugih računala u slučaju da je zauzeto hitnijim poslom. Sve ovo isključuje potčinjavanje jednog računala drugom.

Teško je odlučiti se i za drugu krajnju mogućnost izgradnje sistema u kojem svako od računala ima autonomno upravljanje, tj. funkcionira potpuno nezavisno od stanja ostalih računala, jer bi tada bila nemoguća uskladjenost rada čitavog sistema u cjelini.

U mnogim slučajevima bi se pokazala prihvatljivom takva struktura u kojoj bi nekoliko računala na osnovnom nivou zajednički rješavali probleme. Pri tome svako od računala samostalno rješava vlastite lokalne zadatke, a u suradnji sa drugim računalima rješava samo općenitije zadatke, koje u hijerarhijskom sistemu rješava računalo na višem nivou.

Pri tome postaje moguće donositi rješenja putem glasanja, što bitno povećava pouzdanost funkcioniranja sistema.

Dakle, svako od računala trebalo bi da posjeduje vlastite upravljačke mehanizme, odgovorne za rješavanje problema u svojoj okolini. Uzajamna povezanost sa drugim računalima ostvarivala bi se na taj način što je funkcioniranje pojedinog računala potčinjeno i interesima cjeline (1).

Nadalje, da procesi kojima se upravlja ne bi izmakli kontroli bitno je da raspodijeljeni sistem očuva svoje vitalne funkcije čak i u prisustvu težih otkaza. Ova sposobnost posebno je značajna u otežanim fizičkim uvjetima rada, a za njeno ostvarivanje neophodna je izvjesna redundantnost - trebalo bi obezbijediti da svaka vitalna funkcija sistema bude prisutna u više radnih računala.

Najzad, značajna je i fleksibilnost mreže, tj. njena sposobnost da normalno funkcionira u raznovrsnim konfiguracijama. Mogućnost uklanjanja redundantnih računala važna je sa stanovišta robusnosti, a mogućnost dodavanja sa stanovišta povećanja paralelizma u cilju poboljšanja performansi u stvarnom vremenu.

3. ELEMENTI ARHITEKTURE

Polazeći od principa formuliranih u prethodnom poglavlju može se provesti kvalitativna analiza kako osnovni elementi arhitekture raspodijeljenog sistema mogu da podrže upravljanje brzim procesima. Ovdje je sasvim ukratko skicirano takvo razmatranje.

Pod strukturom raspodijeljenog sistema podrazumijeva se broj i raspored računala, te karakter njihove povezanosti (2).

Problematici povezivanja računala u literaturi je posvećena velika pažnja (3,4,5,6,7,8,9,10). Uobičajeni načini povezivanja računala dati su na slici 1. - najsloženija je potpuna struktura, a kidanjem veza izmedju pojedinih računala mogu se dobiti sve ostale strukture.

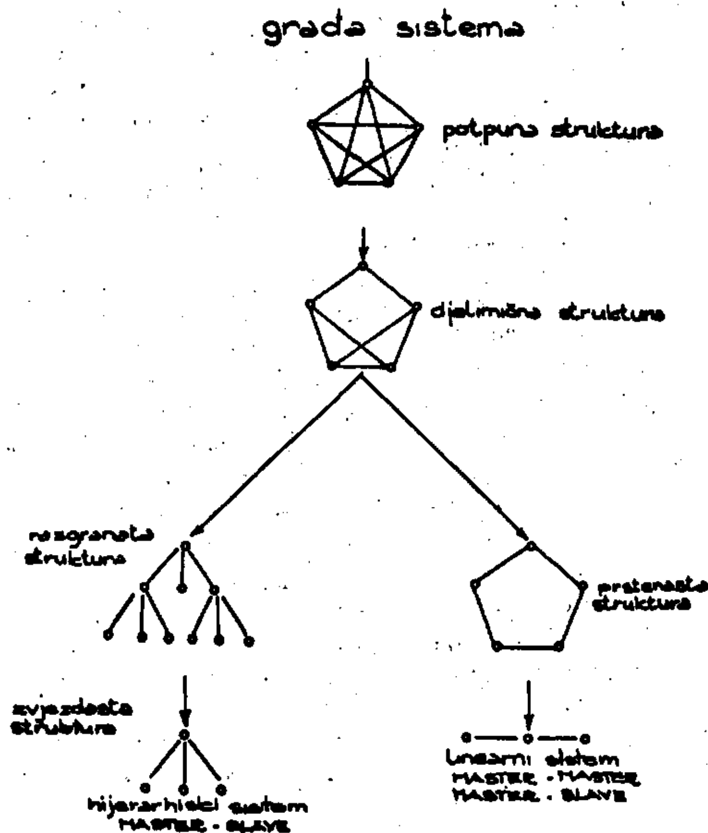
Kod potpune strukture računala su povezana po principu "svaki sa svakim" što omogućava jednostavnu direktnu komunikaciju izmedju izvora i odredišta, a moguće je uspostaviti vezu i preko velikog broja alternativnih puteva. Zbog toga je pouzdanost ovakve strukture maksimalna, ali je postignuta uz visoke troškove.

Kidanjem pojedinih veza potpune strukture dobija se djelimična struktura, a iz nje se izvode razgranata i prstenasta struktura.

Razgranata struktura odgovara hijerarhijskoj organizaciji čija karakteristična osobina je uzastopno raščlanjiva-

nje sistema na dijelove (podsisteme) izmedju kojih se uspostavljaju odnosi koordinacije. Tada upravljački uredjaj višeg ranga upravlja velikim jedinicama sistema, od kojih svaka ima svoj upravljački uredjaj. U mreži nema alternativnih puteva a pouzdanost je minimalna jer otkazivanje pojedinog višeg nivoa postaje kritično za upravljački sistem u cjelini.

Naime, u sistemima sa hijerarhijskom strukturom računalo nižeg ranga bi trebalo da rješava relativno proste lokalne zadatke upravljanja, te se pretpostavlja da će posjedovati i ograničene kapacitete prerade informacija. Pri tome su u nadležnosti računala na višem nivou zadaci koje je potrebno izvršavati u cilju medjusobnog uskladjivanja rada računala na osnovnom nivou i koji se mogu rješavati na osnovu manje detaljne informacije o stanju objekta. To se odnosi i na računala na višim nivoima tako da računala na osnovnom nivou dobijaju najdetaljniju i najkonkretniju informaciju o stanju objekta, a prelaskom na više nivoe ta informacija se uopćava sa karakterom zadatka koji rješavaju ta računala. Naredbe up-



Sl.1.

ravljanja u sistemima sa hijerarhijskom strukturom izdaje upravljačko računalo najvišeg nivoa.

Iz razgranate strukture može se izvesti zvjezdasta struktura kod koje centralno nadredjeno računalo upravlja sa više podredjenih računala. Struktura sistema je jednostavnija od razgranate, ali zbog usmjeravanja svih poruka preko središnjeg računala ovo postaje kritično po pouzdanost sistema.

Topologija zvijezde tradicionalno se javlja u kombinaciji sa centraliziranim upravljanjem koje obavlja središnje računalo. Sa stanovišta pouzdanosti zvjezdasta topologija je pogodna jer otkaz pojedinog računala ili prenosnog puta ne remeti komunikaciju ostalih. Centralizirano upravljanje je medjutim nepovoljno jer otkaz središnjeg računala onesposobljava mrežu. Autonomija računala i fleksibilnost su ostvarljive, ali još više ističu usko grlo u središnjem računalu.

Kod prstenaste strukture direktna veza omogućena je izmedju parova mikroračunala, a samom strukturom su eliminirani problemi usmjeravanja poruke.

Kidanjem veza izmedju računala u prstenu dobija se linija.

Topologije linije i prstena obično podrazumijeva decentralizirano upravljanje koje omogućava brz prenos poruka i podržava autonomiju računala. Nedostatak ovih topologija je da prekid prenosnog puta izmedju dva računala bitno remeti komunikaciju u sistemu.

Izbor načina povezivanja računala u svakoj konkretnoj primjeni diktiran je zahtjevima koji se na sistem postavljaju (pouzdanost, vrijeme odziva, fleksibilnost itd.), a zasniva se uglavnom na kompromisu zahtjeva pouzdanosti i cijene. Biraju se topologije koje će zadovoljavajući zahtjeve pouzdanosti povezati mikroračunala na najjeftiniji način, tj. omogućiti korištenje jednostavnih i jeftinih medjuspojeva i jednostavnih nižih protokola. Iz ovih razloga koriste se uglavnom tri topologije - prsten, zvijezda i linija (9,10).

3.1. Decentralizirana zvijezda

Na osnovu zahtjeva postavljenih u prethodnom poglavlju za konfiguraciju sistema bit će predložena decentralizirana zvijezda (slika 2). Ovakav pristup razlikuje se od uobičajenih po tome što se u sistemu vrši podjela na funkcije upravljanja dijelovima procesa i funkcije upravljanja vezama. Distribuiranjem programske podrške odvojene su funkcije vezane uz neposredno vodjenje dijelova procesa od centralnih komunikacijskih funkcija.

Sva računala prema slici 3. povezana su na grupni prekidač serijskom dupleksnom vezom. Grupni prekidač sastoji se od 2 skupine parova multiplexer-demultiplexer koji tvore 2 kanala od po 16 linija. Funkcijama multiplexera i demultiplexera komunikacijsko računalo upravlja putem izlaznih linija programibilnih paralelnih periferala. Na ulazne linije u prekidač priključeni su asinhroni programibilni periferali pomoću kojih komunikacijsko računalo "prisluškuje" što se u sistemu dogadja.

Komuniciranje je podijeljeno u tri faze:

- faza uspostavljanja veze;
- faza prenosa podataka;
- faza prekida veze.

Proces uspostavljanja veze izmedju predajnika i prijemnika započinje zahtjevom koji predajnik upućuje komunikacijskom računalu. Nadgledajući kontinuirano sve predajne linije komunikacijsko računalo čeka da se na nekoj od njih pojavi poruka. Da bi komunikacijsko računalo lako spoznalo koja poruka je njemu namijenjena formati poruka namijenjenih njemu i poruka koje učesnici medjusobno razmjenjuju se bitno razlikuju. Otkrivši u zaglavlju poruke karakterističnu kombinaciju komunikacijsko računalo prihvata poruku koja je njemu namijenjena. Poruke koje učesnici medjusobno izmjenjuju komunikacijsko računalo ignorira.

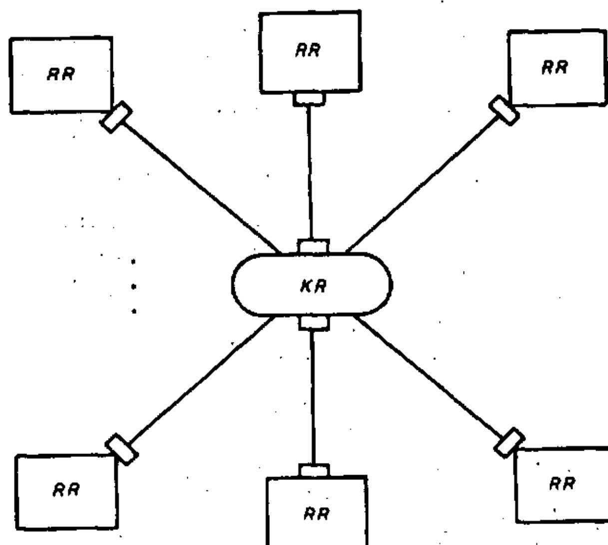
Primivši zahtjev za uspostavljanjem veze komunikacijsko računalo ispituje listu već priključenih korisnika i provjerava stanje prijemnog računala da bi ustanovilo da li je ovo spremno prihvatiti poziv. Prijemnik može biti slobodan, zauzet ili u kvaru. Na osnovu stanja prijemnika komunikacijsko računalo odredjuje da li, i koje linije dodijeliti učesnicima. U principu, broj linije odgovara broju učesnika, tj. ako učesnik 03 želi komunicirati sa učesnikom 19 bit će im dodijeljene linije 03 i 13. Stanja linija sadržana su u posebnim registrima. Linija može biti slobodna, zauzeta (registar sadrži broj učesnika) ili u kvaru.

Nakon toga komunikacijsko računalo obavještava predajno računalo o tome da li je zahtjev prihvaćen i komunikacija prelazi u fazu izmjene podataka. Nakon uspostavljanja direktne fizičke veze medju radnim računalima predajno računalo preuzima funkcije upravljanja, pa se komunikacijsko računalo isključuje iz daljnih intervencija na toj vezi sve do pojave zahtjeva za prekidom veze.

Prelaskom u fazu prenosa podataka daljni paketi prolaze mrežom po fiksiranom putu. Dolazeći predajnom linijom do multiplexera oni moraju samo "pogledati" kojim putem krenuti dalje.

Izuzimajući intervencije u slučaju kvara pojedinog upravljačkog računala autonomija upravljačkih podsistema nije ničim narušena. Problem kvara centralnog komunikacijskog računala dovodi u pitanje pouzdanost cijelog sistema. Medjutim, zadaci postavljeni pred komunikacijsko računalo svode mogućnost greške na minimum, a i njegova laka zamjenljivost opravdavaju ovakav pristup.

Detaljan prikaz komunikacijskih procedura sadržan je u (11) gdje je data i opširno komentirana programska lista kojom su komunikacijske procedure realizirane u asembleru mikroprocesora M6809.



Sl.2.

4. RASPOREDJIVANJE ZADATAKA U ZVJEZDASTOJ KONFIGURACIJI

U (12, 13 i 14) izložena je metodologija rasporedjivanja zadataka koji u stvarnom vremenu obavljaju odredjena izračunavanja na osnovu pobuda vanjske okoline, a koji medjusobno suradjuju tokom svoga izvodjenja.

Odluke o rasporedjivanju zadataka se prema ovim metodama izvode na dva nivoa-lokalnom i globalnom. Algoritam za lokalno rasporedjivanje opisan u (13) izvodi se u svakom od računala.

Odluke o rasporedjivanju donijete na lokalnom nivou odredjuju brzinu kojom pojedino računalo može odgovoriti na dogadjaje u stvarnom vremenu. U slučaju normalnog rada sistema aktivan je samo lokalni rasporedjivač, koji odlučuje koji proces iz repa pripravnih procesa nastaviti.

Ukoliko zahtjevi za izračunavanjima prevazilaze mogućnosti odredjenog računala, ili pak znatno degradiraju njegove performanse aktivirat će se globalni rasporedjivač čija zadaća je da odredi koje od računala je sa sistemskog stanovišta najpodobnije da u konfliktnoj situaciji preuzme izvodjenje hitnog zadatka.

Za razliku od algoritama za globalno rasporedjivanje definiranih u (12,14) za prstenastu konfiguraciju, ovdje je globalni rasporedjivač samo jedan a smješten je u komunikacijskom računalu.

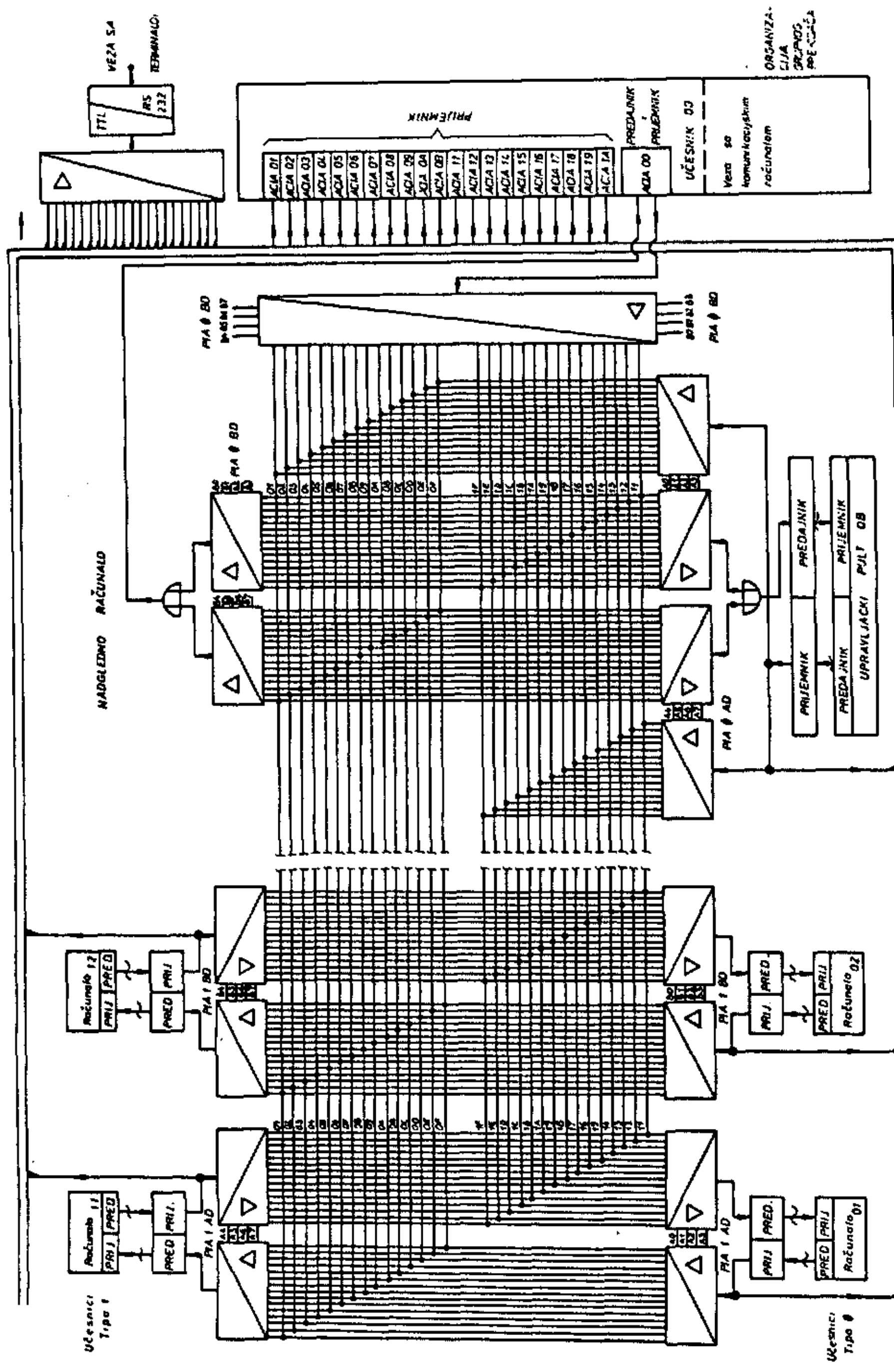
Algoritmi za lokalno rasporedjivanje definirani u (13) predstavljaju kostur i obezbjedjuju determinirane crte strukture, dok algoritam za globalno rasporedjivanje omogućava postizanje neophodne elastičnosti funkcioniranja i prilagodljivost sistema promjenljivim uvjetima okoline.

Ovakva konfiguracija dinamički vezanih lokalnih rasporedjivača omogućava rad i u slučaju ispada iz rada pojedinog procesora kada ostali procesori na zahtjev globalnog rasporedjivača preuzimaju njegove poslove.

4. ZAKLJUČAK

Da bi raspodijeljeni sistem efikasno vodio brze procese u stvarnom vremenu arhitektura sistema morala bi obezbijediti brz prenos hitnih poruka, autonomiju pojedinih računala, robusnost i fleksibilnost. Kvalitativna razmatranja koja su u radu izložena u pojednostavljenom obliku pokazuju da najpoznatije arhitekture raspodijeljenih sistema ne ispunjavaju u potpunosti postavljene zahtjeve za rad u stvarnom vremenu. Prenos poruka je znatno brži jer komunikacijsko računalo ima ulogu posrednika u sklapanju direktne veze izmedju medjuzavisnih radnih računala.

Prednosti ovakvog pristupa su u tome što je predložena



Sl.3.

hijerarhijska struktura u jednom nivou, čime se obezbjedjuje optimalna elastičnost i prilagodljivost sistema situaciji. Obrada informacija teče u skladu sa potrebama i može zadovoljiti i u slučajevima kada su pojedini lokalni raspoređjivači potpuno van upotrebe. Najveći nedostatak je u tome što i male nepravilnosti u radu globalnog raspoređjivača mogu dovesti do pada čitavog sistema.

LITERATURA:

1. Kukrika M.: "Prijedlog za jednoliko opterećivanje računala u raspodijeljenim sistemima", III Jugoslovensko savjetovanje o mikroračunalima u procesnom upravljanju - MIPRO, Opatija (1984).

2. Kukrika M.: "Problemi komuniciranja u sistemima sa više mikroračunala" II Jugoslovensko savjetovanje o mikroračunalima u procesnom upravljanju - MIPRO, Opatija (1983).

3. Le Lann,G.: "An analysis of different approaches to distributed computing", Proc. of the 1st intern.conf. of distributed comp.systems, Huntsville, AL, (oct. 1979).

4. Thurber, K.: "Distributed processor communication architecture", Lexington Books (1978).

5. Lampson,M.: "Distributed systems - architecture and implementation", Springer Verlag (1981).

6. Anderson,G. and Jensen, E.: "Computer interconnection: Taxonomy, Characteristics, and Examples", Computing Surveys, (dec. 1975).

7. Boorstyn,R.: "Large-Scale network topological optimization", IEEE trans. communication com-25 (jan 1977).

8. Kukrika M.: "Pristup organiziranju lokalnih mreža mikroračunala" V medjunarodni simpozij "Kompjuter na sveučilištu", Cavtat (1983).

9. Weitzman,G.: "Distributed micro-mini computer systems" Prentice Hall (1981).

10. Clark, D. et al.: "An introduction to local area networks", proc. IEE 66 (1978).

11. Tasić,T.: "Pristup realizaciji zvjezdaste višeračunarske topologije", Diplomski rad, ETF, B.Luka (1984).

12. Kukrika,M.: "Pristup dinamičkom rasporedjivanju zadataka u prstenastoj mreži računala", Informatica 2 (1984).

13. Kukrika,M.: "Pristup kreiranju rasporedjivača zadataka u distribuiranom izvršnom sistemu sa radom u stvarnom vremenu"; Informatica 3 (1984).

14. Kukrika,M.: "Primjer dinamičkog rasporedjivanja zadataka u višeračunarskim sistemima sa radom u stvarnom vremenu", Informatica 2 (1985).

MULTIPROGRAMIRANJE I MERENJE I/O ČEKANJA SISTEMA

DOBROSAV LEČIĆ DIPL. MAT.

UDK: 681.3.013

SOUR „BOROVO"
ELEKTRONSKO RAČUNSKI CENTAR

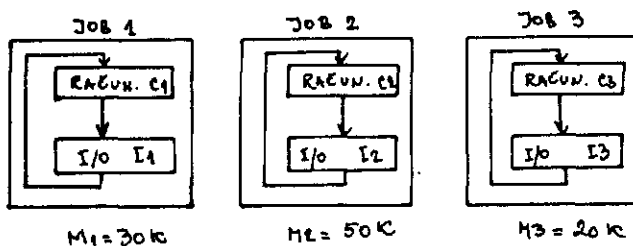
Mnogi od nedostataka koje karakterišu najjednostavnije tehnike upravljanja memorijom, potiču od problema sukobljavanja fiksnog resursa koji je na raspolaganju sa različitim zahtevima za tim resursom. Fizički, hardware resurs kompjutera može varirati (menjati se) samo u toku relativno dužih vremenskih perioda, napr. dodavanjem inkremenata 64K bajtova memorije, jedamput ili možda dva puta godišnje. Istovremeno zahtevi za resursima od strane različitih jobova mogu biti vrlo veliki.

Pokušamo li prisiliti programere da razvijaju sve jobove sa identičnim zahtevima prema resursima, videćemo da to ide kranje teško. Daleko efikasnije bi bilo operirati sa više nego jednim jobom istovremeno i distribuirati resurse izmedju tih jobova.

Ovakva tehnika se zove MULTIPROGRAMIRANJE i ona će ovde biti ukratko prezentirana u svojim najjednostavnijim oblicima.

Primer multiprogramiranja

Slika 1. daje opis tri aktivna joba. Svaki od njih zahteva vremena za usluge računanja od strane procesora Ci, a isto tako i korištenje kanala za I/O (ulaz/izlaz), Ii.



Slika 1

Osim toga adresni prostor svakog joba treba memoriju u iznosu Mi. Zbog jednostavnosti modelirali smo svaki job tako da prvo izvršava nekakva računanja, zatim vrši I/O operacije, i taj ciklus se ponavlja.

Predpostavimo da imamo 100K raspoložive memorije. Ukoliko bi radio samo jedan job (prvi), koristio bi 30K, a ostatak 70 K memorije bi bio neiskorišten. Dalje, vreme procesora u iznosu $I_1/(C_1+I_1)$ bi se gubilo zbog I/O čekanja.

Alternativno bi mogli smestiti adrese sva tri joba u glavnu memoriju istovremeno (30K+50K+20K=100K). Tada bismo imali potpuno iskorištenje 100K - bajtne memorije. Upravljač procesora bi dodelio procesor jobu 1. Posle izvršavanja računanja C_1, umesto stajanja zbog čekanja da se kompletira I_1, procesor se dodeljuje jobu 2. Slično kad job 2 stigne do I_2 procesor se može pridodati jobu 3. Kad job 3 stigne do I_3, procesor se opet može dodeliti jobu 1, ako je I_1 završeno (npr. $I_1 \leq C_2+C_3$). Na taj način procesor će čekati samo $I_1-(C_2+C_3)$ vremenskih jedinica umesto I_1.

Očigledno, da je korištenjem tehnike multiprogramiranja iskorištenje procesora mnogo veće u odnosu na slučaj kada se vrši obrada samo jednog joba u isto vreme.

U mnogim slučajevima je moguće kompletirati 2 ili 3 joba u skoro istom vremenskom iznosu koje je potrebno za jedan job.

MERENJA I/O ČEKANJA SISTEMA

Realan proces nije tako jednostavan kako je to prikazano na slici 1. Operacije računanja i I/O se mešaju na kompleksan način. Ovde ćemo se ukratko zadržati na jednom pojmu koji može predstavljati nekakve mere efikasnosti operativnog sistema (OS-a), a samim tim i kompjutera. To je procentni iznos vremena I/O čekanja jednog joba, a koji se izračunava na sledeći način:

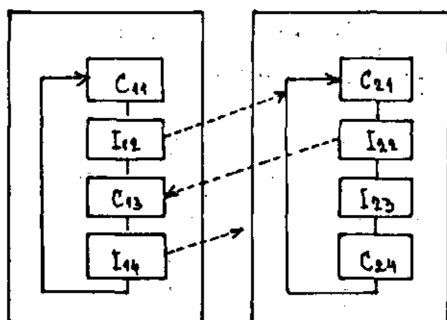
$$\omega = \frac{\text{ukupno I/O čekanje}}{\text{ukupno I/O čekanje + vreme CPU-a}}$$

Ovo se lako može meriti gotovo na svim kompjute-

rima koji ne rade multiprogramski. Nekoliko studija pokazuje da na prosečnim job izvodjenjima koji se rade na srednjim i velikim računarima,... iznosi oko 65%.

Ukoliko bi imali dva joba sa po ω =50% koji se multiprogramiraju, onda se efektivno I/O čekanje u procentima redukuje do nule. Prema tome na modelu sa sl. 1 bi imali $C_1=I_1=C_2=I_2$. Na taj način bi se C_2 konkurentno izvodio sa I_1 i slično C_1 sa I_2. Medjutim kao što smo napomenuli, ovaj model je suviše uprošten.

Slika 2. ilustruje procese sa medjusobno izmešanim periodima računanja i I/O operacija. Ovde ćemo predpostaviti da svaki C_{11}, I_{12}, C_{13} itd. imaju jednake dužine. Lako je uočiti da je vrednost ω za svaki job jednak 50%. Ukoliko pokušamo da ovaj problem rešimo multiprogramski, možemo upariti I_{12} sa C_{21} i I_{22} sa C_{13}. Medjutim kad job 1 želi da izvrši I/O pri I_{14}, vidimo da job 2 takodjer treba I/O pri I_{23}. Na taj način procesor "besposliči" dok se I_{14} i I_{23} kompletiraju.



Slika 2.

Da bi uopštili model sa slike 2 možemo koristiti teoriju verovatnoće da računamo efekte sistema na osnovu procenta I/O čekanja. Praktično, u opštem slučaju sistem mora čekati samo kada svi procesi zahtevaju I/O tačno u isto vreme. Ako multiprogramiramo n procesa, a svaki je sa istim procentima iznosa ω , ukupan iznos I/O čekanja sistema ω' bi aproksimativno bio $\omega' = \omega^n$. Tako npr. ako bi ω iznosio 50% imali bi:

Broj procesa	Procenat I/O čekanja sistema
1	$(0{,}50)^1 =$ 50%
2	$(0{,}50)^2 =$ 25%
3	$(0{,}50)^3 =$ 12,5%
4	$(0{,}50)^4 =$ 6,3%
5	$(0{,}50)^5 =$ 3,1%
6	$(0{,}50)^6 =$ 1,6%

Medjutim, ova tabela iako daje korisnu aproksimaciju, nije sasvim korektna. I ako bi hteli da zadržimo pretpostavku za korištenje računa verovatnoće, za svaki od procesa je potreban nekakav pomak u svakom vremenskom periodu. Ovo bi, doduše bilo sasvim tačno ukoliko bi imali n procesora i n kanala na raspolaganju u kom slučaju ω' indukovalo verovatnoću mirovanja svih procesora jednovremeno. Razumljivo je da predpostavljamo n konkurentnih I/O operacija, pogotovo ako raspolažemo višestrukim multi-pleksor kanalima. Pošto, medjutim, imamo samo jedan procesor i rezultat je vezan samo za njega. Bolja aproksimacija parametra ω' se dobija koristeći tzv. tehniku BIRTH - AND -DEATH Markov proces. Po toj osnovi je:

$$\omega' = \frac{\left(\frac{\omega}{1-\omega}\right)^n}{n! \sum_{i=0}^{n} \frac{\left(\frac{\omega}{1-\omega}\right)^i}{i!}}$$

Pre svega treba znati da su obe prezentirane formule aproksimativne. Značajno je, medjutim, to da uopšte uzev vreme I/O čekanja sistema se značajno redukuje povećanjem stepena multiprogramiranja, tj. brojem procesa koji se multiprogramiraju.

LITERATURA :

-"OPERATING SYSTEMS"
Stuart E. Madnick
John J. Donovan
INTERNATIONAL STUDENT EDITION

NOVE RAČUNALNIŠKE GENERACIJE

Mednarodna konferenca o računalniških sistemih pete generacije v Tokiu

V zborniku del mednarodne konference o računalniških sistemih pete generacije (FGCS 84, Fifth Generation Computer Systems 1984, Proceedings of the International Conference on Fifth Generation Computer Systems 1984, Tokyo, Japan, November 6-9, 1984, Edited by Institute for New Generation Computer Technology (ICOT), Tokyo, Japan) je objavljenih več prispevkov, ki so vredni naše pozornosti. Oglejmo si tematsko kar kazalo tega zbornika.

V s e b i n a

Raziskave in razvoj v okviru ICOT:

-- Pregled izvirne filozofije projekta računalniških sistemov pete generacije (K. Fuchi)
-- Trenutno stanje in prihodnji načrti projekta računalniških sistemov pete generacije (K. Kawanobe)
-- Arhitekture in sistemi aparaturne opreme: paralelni stroj sklepanja in stroj baze znanja (K. Murakami, T. Kakuta, R. Onai)
-- Sistem osnovne programske opreme (K. Furukawa, T. Yokoi)
-- Stroj za zaporedno sklepanje: poročilo o napredovanju (S. Uchida, T. Yokoi)
-- Stroj za zaporedno sklepanje: njegovo programiranje in operacijski sistem (T. Yokoi, S. Uchida in Tretji laboratorij ICOT)

Povabljeno predavanje:

-- Enačbe in neenačbe na končnih in nekončnih drevesih (A. Colmerauer)

Povabljeni prispevki:

Temelji in osnovna programska oprema:

-- Programiranje z moduli kot tipsko funkcionalno programiranje (R. Burstall)

Arhitekture:

-- Masivna paralelna arhitektura za zelo obsežne baze podatkov (Y. Tanaka)

Uporabe:

-- Če je Prolog odgovor, kaj je potem vprašanje? (D.G. Bobrow)

Poslani prispevki:

Temelji logičnih programov (1):

-- Nekatere praktične lastnosti interpretov za logično programiranje (D.R. Brough, A. Walker)
-- Qute: funkcionalni jezik, ki temelji na unifikaciji (M. Sato, T. Sakurai)
-- Pojavni račun: mehanizem za verjetnostno sklepanje (A. Bundy)
-- Teorija popolnih logičnih programov z enakostjo (J. Jaffar, J.-L. Lassez, M.J. Maher)
-- Programska transformacija enačbnih programov v logične programe (A. Togashi, S. Noguchi)

Temelji logičnih programov (2):

-- Sinteza transformacijskega logičnega programa (T. Sato, H. Tamaki)
-- Učinkovita unifikacija z nekončnimi členi pri logičnem programiranju (A. Martelli, G. Rossi)
-- Avtomatična implementacija abstraktnih podatkovnih tipov, opisanih z logičnim programirnim jezikom (N. Heck, J. Avenhaus)
-- Programi kot izvršljivi predikati (C.A.R. Hoare, A.W. Roscoe)

Temelji logičnih programov (3):

-- Logična izpeljava prologovskega interpreta (K. Fuchi)

Temelji logičnih programov (4):

-- O kompleksnosti paralelnega računanja unifikacije (H. Yasuura)
-- Ažuriranje podatkovne baze v čistem Prologu (D.S. Warren)
-- DAL: logika za podatkovno analizo (L. Farinas, E. Orlowska)

Logični programirni jeziki-metodologije (1):

-- Večizvedbene strukture v Prologu (S. Cohen)
-- Iskanje začasnih členov v prologovskih programih (P. Vataja, E. Ukkonen)
-- Delta-Prolog: distribuirani logični programirni jezik (L.M. Pereira, R. Nasr)
-- Pomembne lastnosti ESP-ja (T. Chikayama)
-- Opombe o sistemih, ki programirajo v Parlogu (K. Clarck, S. Gregory)

Logični programirni jeziki-metodologije (2):

-- Usmerjene relacije in obrnitve prologovskih programov (Y. Shoham, D.V. McDermott)
-- Učinkovita obdelava tokkov-polj v logičnih programirnih jezikih (K. Ueda, T. Chikayama)
-- Kaj je spremenljivka v Prologu? (H. Nakashima, S. Tomura, K. Ueda)
-- Opomba o abstrakciji množice v logičnem programirnem jeziku (T. Yokomori)

Logični programirni jeziki-metodologije (3):

-- RF-Maple: logični programirni jezik s funkcijami, tipi in hkratnostjo (P.J. Voda, B. Yu)
-- Prevajanje prologovskih programov brez uporabe prologovskega prevajalnika (K.M. Kahn, M. Carlsson)
-- Dvoravninski Prolog (A. Porto)
-- Metakrmiljenje logičnih programov v Metalogu (M. Dincbas, J.-P. Le Pape)

Arhitekture za novogeneracijsko računalništvo (1):

-- Arhitektura hkratnega podatkovnega dostopa (H. Diel)
-- Na znanju osnovani, visokointegrirani smerni sistem WIREX (H. Mori, K. Mitsumo-

to, T. Fujita, S. Goto)
- Sword 32: zložnokodno posnemovalni mikroprocesor za objektno usmerjene jezike (N. Suzuki, K. Kubota, T. Aoki)

Arhitekture za novogeneracijsko računalništvo (2):

- Oblikovanje aparaturne opreme in implementacija osebnega računalnika za zaporedno sklepanje PSI (K. Taki, M. Yokota, A. Yamamoto, H. Nishikawa, S. Uchida, K. Nakajima, M. Mitsui)
- Mikroprogramirani interpret osebnega računalnika za zaporedno sklepanje (M. Yokota, A. Yamamoto, K. Taki, H. Nishikawa, S. Uchida, K. Nakajima)
- Oblikovanje in implementacija stroja relacijskih podatkovnih baz (H. Sakai, K. Iwata, S. Kamiya, M. Abe, A. Tanaka, S. Shibayama, K. Murakami)
- Pretok obdelave vprašanj pri arhitekturi Delta s funkcionalno distribucijo (S. Shibayama, T. Kakuta, N. Miyazaki, H. Yokota, K. Murakami)
- Algoritmi LPS (A. Lowry, S. Taylor, S.J. Stolfo)
- Zmogljivostne ocene stroja Dado: primerjava s Treat in Rete (D.P. Miranker)

Arhitekture za novogeneracijsko računalništvo (3):

- Sistolično programiranje: zgled paralelnega procesiranja (E. Shapiro).
- Omejeni in-paralelizem (D. DeGroot)
- Arhitektura stroja PIE s paralelnim sklepanjem (T. Moto-oka, H. Tanaka, H. Aida, K. Hirata, T. Maruyama)
- Stroj za relacijsko podatkovnopretočno podatkovno bazo na temelju hierarhične krožne mreže (J.I. Kim, S.R. Maeng, J.W. Cho)

Arhitekture za novogeneracijsko računalništvo (4):

- Assip-T: stroj za dokazovanje izrekov (W. Dilger, H.-A. Schneider)
- Paralelno izvajanje logičnih programov na temelju zamisli podatkovnega pretoka (R. Hasegawa, M. Amamiya)
- Podatkovno vodeni model za paralelno interpretacijo logičnih programov (L. Bic)
- EM-3: podatkovno vodeni stroj, osnovan na Lispu (Y. Yamaguchi, K. Toda, J. Herath, T. Yuba)

Arhitekture za novogeneracijsko računalništvo (5):

- Transputerska implementacija Occama (D. May, R. Shepherd)
- PEK: zaporedni prologovski stroj (N. Tamura, K. Wada, H. Matsuda, Y. Kaneda, S. Maekawa)
- Izvršitev seznama (bagof) na ali-paralelnem znakovnem (simbolnem) stroju (A. Ciepielewski, S. Haridi)

Uporabe v novogeneracijskem računalništvu (1):

- Prologovski izvedeniški sistem za logično oblikovanje (F. Maruyama, T. Mano, K. Hayashi, T. Kakuda, N. Kawato, T. Uehara)
- Specifikacija materialne opreme s časovno logiko in učinkovita sinteza diagramov stanj z uporabo Prologa (M. Fujita, H. Tanaka, T. Moto-oka)

Uporabe v novogeneracijskem računalništvu (2):

- Na znanju osnovani sistem za obratno (tovarniško) diagnozo (H. Motoda, N. Yamada, K. Yoshida)
- Krmiljenje hevrističnega iskanja v prologovskem, mikrokodnosinteznem izvedeniškem sistemu (M.D. Poe)
- Sidur: strukturirni formalizem sistemov, ki obdelujejo informacije znanja (D.D. Kogan, M.J. Freiling)
- Looks: Sistem predstavitve znanja za načrtovanje izvedeniških sistemov v okviru logičnega programiranja (F. Mizoguchi, H. Ohwada, Y. Katayama)

Uporabe v novogeneracijskem računalništvu (3):

- Mandala: programirni sistem znanja, osnovan na logiki (K. Furukawa, A. Takeuchi, S. Kunufuji, H. Yusukawa, M. Ohki, K. Ueda)
- Objektno usmerjeni pristop k sistemom znanja (M. Tokoro, Y. Ishikawa)
- Pametno informacijsko obnavljanje: zanimivo uporabnostno področje za novogeneracijske računalniške sisteme (G.P. Zarri)

Uporabe v novogeneracijskem računalništvu (4):

- Predstavitev znanja in okolje sklepanja: Krine, pristop k integraciji Frama, Prologa in grafike (Y. Ogawa, K. Shima, T. Sugawara, S. Takagi)
- Zasledovanje govisnosti v topični džungli (B. Grau)
- Korak k igralno usmerjenemu, integriranemu analizatorju (K. Uehara, R. Ochitani, O. Mikami, J. Toyoda)

Uporabe v novogeneracijskem računalništvu (5):

- Več o vrzelnih gramatikah (V. Dahl)
- Gramatike za natančno prevajanje stavčnih členov in logični opis podatkovnih tipov kot nedvoumne kontekstnosvobodne gramatike (H. Abramson)
- Paralelna interpretacija naravnega jezika (J.B. Pollack, D.L. Waltz)

Vplivi novogeneracijskega računalništva:

- Kakovostni odtok v novogeneracijskem računalništvu (H.J. Kohoutek)

Osnovna analiza konferenčne problematike

Že v naslovih posameznih referatov se pojavljajo novi pojmi, ki kažejo zlasti na specifične raziskovalne smeri Japoncev v okviru Icota. Treba je priznati, da je japonski način razmišljanja o novogeneracijskih računalnikih močno različen od ustaljenih konceptov in da so Japonci na poti razvoja svojskega, japonskozasnovanega novogeneracijskega računalniškega sistema. Ta sistem bo bržkone konceptualno in tehnološko bistveno drugačen od podobnih ameriških novogeneracijskih sistemov; prav v tem pa se skrivajo prednosti pa tudi velika tržna in tehnološka tvegania takega razvoja in kasnejše realizacije.

Japoncem je v tem trenutku prav gotovo potrebno zaupanje v lastno razvojno hipotezo. Zgledov nimajo! Tri raziskovalna leta so prinesla določene izkušnje in raziskovalne rezultate.

Konceptualni okvir zahteva rekonstrukcijo računalniške materialne in programske opreme, ki naj bi temeljila na logičnem sistemu, tkim. predikatni logiki. Novogeneracijski računalnik naj bi postal predikativno logični stroj, ki ga je moč imenovati tudi stroj sklepanja, saj je osnovna operacija predikatne logike prav sklepanje.

Današnji računalniki uporabljajo tkim. strojni jezik, s katerim je določena arhitektura posa-

meznega računalnika. Programska oprema je narejena prav za strojni jezik in značilnosti tkim. von Neumannovih strojev se zrcalijo bistveno v strojnih jezikih.

Novi strojni jezik, ki se imenuje jedrski jezik, je predikativno logični jezik. Zaradi tesa jezika je potrebno razviti nove materialne in programske arhitekture. Materialna oprema bo tako oblikovana za paralelne operacije in za asociativno iskanje, saj bo sklepanje njena osnovna funkcija. Današnji von neumannski računalniki so pretežno zgrajeni za zaporedne operacije in za naslovno iskanje. Zaradi tesa bo paralelni, asociativni sklepajoči stroj neke vrste ne-von neumannski računalnik. Pa tudi programska oprema bo zgrajena z moduli, ki bodo uporabljali osnovne funkcije sklepanja, vgrajene že v materialno računalniško opremo.

Neproceduralni jedrski jezik bo zelo visok programirni jezik v okviru trenutne tehnologije. Ker bo ta jezik postal strojni jezik, se bo razvoj programske opreme lahko začel na višji ravnini kot razvoj današnje programske opreme. Zmogljivost jezika bo uporabljena za doseganje visoko razvitih funkcij, kot so obdelava informacij znanja in naravnih jezikov.

Seveda pa jedrski jezik ne bo uporabniški jezik. Oblikovani bodo višji jeziki, kot je jedrski jezik, in sicer kot uporabniško usmerjeni (specializirani) jeziki. To ka se danes imenuje jezik za predstvitev znanja, bo obstajalo na uporabniški ravnini in tako bodo visoki uporabniški jeziki pravzaprav kar naravni jeziki.

Seveda pa je lahko hipotetična predpostavka, da bo jedrski jezik logični predikativni jezik, vendar je s to predpostavko mogoče nadaljevati japonski projekt. Seveda pa izbira takesa jezika ni naključna in temelji na analizi preteklega in prihodnjega napredovanja razvoja in tehnologije v svetu.

Japonski projekt je sestavljen iz delovnih področij, kot so umentna inteligenca (UI), programirna tehnika, arhitektura in podporne naprave. Hkrati pa se v okviru tesa projekta preučujejo tudi socialne slike in potrebe prihodnosti.

UI bo imela bržkone vrsto uporab in naj bi postala eden od glavnih razvojnih tokov prihodnje obdelave informacij. UI naj bi vplivala tudi na razvoj konvencionalnih uporabniških področij. Tipična uporabniška področja bodo postala npr. tehnika znanja in izvedeniški sistemi. UI je že danes komercialno zanimivo področje na računalniških sistemih četrte generacije. Njena ekonomska upravičenost pa je še dokaj oddaljena. In UI potrebuje predvsem izboljšano tehnološko osnovo:

Japonci izrecno poudarjajo, da njihov projekt ni nikakršen projekt UI in izvedeniških sistemov, kot to trdijo površno obveščeni posamezniki v ZDA in v Evropi. Raziskave UI gredo predvsem v smeri pojasnjevanja mehanizmov inteligence: tu pa so na vidiku veliki raziskovalni napori v daljšem raziskovalnem obdobju. Izvedeniški sistemi pomenijo ogromen potencial za raznovrstne uporabe. Prav tako ima predikativni logični jezik določene prednosti pri predstavljanju znanja v bazah znanja. UI, izvedeniški sistemi in logični predikativni jeziki so tako predvsem kandidati, ki zagotavljajo določen uporabnostni potencial.

Japonci raziskujejo dovolj intenzivno tudi programirno tehniko. Izboljšanje programirne storilnosti je pomemben projektni cilj. V okviru tesa pa se Japonci ne odločajo za uporabo izdelkov sedanje računalniške generacije, kot sta npr. Ada in Unix, ne želijo občutiti nikakršnih omejitev iz današnjesa tehnološkesa okvira. Tu pa se začenja miselnost, ki jo Japonci sami označujejo kot novodobska filozofija. Ta filozofija ne zajema samo novo računalniško generacijo, upošteva tudi velike spremembe v življenju prihodnjih populacij in poudarja soodvisnost in nujo po tehnološkemu sodelovanju med populacijami na planetu.

A. P. Železnikar

MProlog, jezik za umetno pamet

Podjetje Logicware Inc., 1000 Finch Avenue W., Ste 600, Downsview, Ontario, Canada M3J 2V5 ponuja jezik z imenom MProlog za umetnointeligenčno programiranje. Ta jezik je uporabljiv v prožnem operacijskem okolju, in sicer za ibmovske kabinetne računalnike z VMS/CMS in MVS/TSO, za decovske sisteme VAX/VMS in VAX/UNIX in tudi za mikroračunalniške sisteme IBM PC-XT in AT.

MProlog je prečiščena izvedenka Prologa za uporabo v poslovnih, industrijskih in raziskovalnih okoljih. Z jezikom MProlog je mogoče

-- zaščititi podjetniška vlaganja v strokovno znanje,
-- oblikovati učinkovite lastne izvedeniške sisteme,
-- skrajšati razvojne dobe programiranja strateških nalog in
-- zagotoviti prenosljivost posameznih uporab.

Izvajanje prologovskih programov omogoča krmiljeno izpeljevanje (sklepanje) z uporabo dejstev (aksiomov), odnosov (relacij) in izpeljevalnih pravil. Tako je mogoče opisati določen problem brez neposrednega programiranja posameznih korakov, ki vodijo k problemski rešitvi. Prologovski programi omogočajo sistemom učenje, asociiranje, sklepanje in odločanje. Enostavno: MProlog je programirni jezik umetne pameti.

Kje je mogoče MProlog uspešno uporabiti:

-- v izvedeniških sistemih,
-- v izpeljevalnih podatkovnih bazah,
-- pri razumevanju naravnih jezikov,
-- pri računalniško podprtem učenju,
-- v diagnostiki in popravljanju napak,
-- pri vidni zaznavi in vodenju,
-- v pametnih pomočnikih in
-- na drugih področjih umetne pameti.

MProlog izkazuje tudi nekaj lepih lastnosti:

-- visoko zmogljivost z učinkovito uporabo virov,
-- določeno neodvisnost od aparaturne in opracijske opreme,
-- možnosti modularnega oblikovanja: dovoljuje podmnožice problema, ki bo opredeljevan in preizkušan, z nudenjem bistvenega povečevanja produktivnosti,
-- programsko razvojno okolje:
  -- interaktivni programski urejevalnik,
  -- sprotna pomoč,
  -- hkratno urejevanje in popravljanje napak,
  -- programsko sledenje,
  -- uporabniško določeno obravnavanje napak,
  -- učinkovito zbiranje odpadkov in
  -- prek 250 vgrajenih predikatov,
-- povezave k proceduralnim jezikom in podatkovnobaznim upravljalnikom,
-- napredujoča podpora in izboljšave,
-- izčrpna dokumentacija in vzgoja.

A. P. Železnikar

UPORABNI PROGRAMI

```
=============================================
=                                           =
=  Besedilni oblikovalnik v jeziku Pascal   =
=                                           =
=============================================
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                            %
%  Informatica UP 22: Alsoritmi in podatkovne %
%                    strukture II - vaje      %
%  Text Formatter                            %
%  marec, april 1985                         %
%  priredila Jelena Ficzko in A.P. Železnikar %
%  sistem CP_M, Delta Partner                %
%  prevajalnik Pascal_MT+                    %
%                                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

POSKUSNI BESEDILNI OBLIKOVALNIK S POSEBNO UKAZNO STRUKTURO
1. del

Jelena Ficzko
Anton P. Železnikar
Iskra Delta, Ljubljana

1. Uvod

Besedilni oblikovalnik z delovnim imenom Lenca je nastal skladno s specifikacijami seminarskih vaj v okviru predmeta Alsoritmi in podatkovne strukture II. Ta študentska seminarska nalosa dovoljuje časovno razdobje enesa meseca; ko mora študent opraviti to naloso samostojno, tako da sme uporabljati razpoložljivo literaturo. Ukazna zalosa je pri tem dovolj omejena in študent jo lahko razširja skladno z razpoložljivim časom in izkušnjami.

Osnovna zamisel besedilnesa oblikovalnika narekuje uporabo štirimestnih ukazov s parametri in brez njih, tako da je te ukaze mosoče postavljati kjerkoli v besedilo, tj. na poljubnih mestih v vhodni besedilni vrstici. Upoštevajo se samo sintaksno pravilni ukazi, nepravilni ukazi (nizi, ki so mišljeni kot ukazi) pa ostanejo v izhodnem besedilu. Format ukaza brez parametra je

*XY,

s parametrom pa

*XY,nn,

kjer je '*' predznak in ',' sufiks tako ukaza kot parametra. Tako bosat format omosoča, da lahko postavljamo ukaze kamorkoli v besedilo, hkrati pa predstavlja tudi osnovo nalose, kako razpoznavati ukaze v besedilu čim bolj optimalno. Podniz 'XY' je ukazna mnemonika in je vobče sestavljen iz dveh znakov velikih črk, podniz 'nn' pa je največ štirimestno celo število.

Oblikovalnik mora izpisati tudi kazalo besedilnih naslovov s pripadajočimi številkami strani. V tem oblikovalniku je kazalni del procedurno popolnoma ločen od besedilnesa dela prosrama, tako da je konstrukcija prosrama jasna in dokaj neproblematična. Seveda pa je mosoče slavne procedure tako parametrizirati, da so uporabne za besedilni in kazalni del prosrama; pri tem se število potrebnih procedur zmanjša, imeti pa morajo kar znatno število parametrov. Ta optimizacija bo pokazana v drusem delu prispevka.

S temi zahtevami je nalosa okvirno določena in dela se lahko lotimo tako, da definiramo neko začetno ukazno množico. Pri tem upoštevamo, da bodi prosram oblikovalnika prilasojen ukaznim spremembam in dodatkom, torej hitremu spreminjanju oblikovalniške namembnosti.

2. Področje uporabe

Besedilni oblikovalniki so bili v široki uporabi pred pojavitvijo tkim. zaslonskih urejevalnikov, kot je npr. Wordstar. Pri besedilnem oblikovalniku ne vidimo takoj učinkov posameznih ukazov, ki smo jih v besedilo postavili in moramo oblikovalnik izvajati. Pri tem želimo imeti možnost preizkušanja tudi na zaslonu, še zlasti v ravojni fazi, ko preizkušamo učinke posameznih oblikovalnih ukazov. Poskusni oblikovalnik bo imel tudi te zmosljivosti. Omosočeno bo torej izvajanje oblikovalnika iz vhoda (vhodne zbirke) v izhod (izhodna zbirka) in v kazalo (kazalna zbirka). Pri tem bo lahko vhod, izhod in kazalo tudi konzola (CON: bo v tem primeru ime zadevne zbirke oziroma zbirk).

Prednost oblikovalnika kot besedilnesa urejevalnika je v tem, da imajo lahko nekateri njesovi ukazi intesralni učinek. Tako lahko uporabimo te ukaze tudi že v nekem formatiranem besedilu, ki pa mu želimo intesralno (v celoti) spremeniti format.

Besedilni oblikovalnik predstavlja hvaležno področje za pisanje posameznih procedur in za spoznavanje nizne (tekstovne) obdelave podatkov zlasti v jeziku Pascal (če upoštevamo standarizirani Pascal). Jezik Pascal/MT+ je razširitev Wirthove izvedenke in podpira celotni ISO standard jezika (DPS/7185). Ta prosram oblikovalnika bo temeljil na osnovnem (Wirthovem) standardu z izjemo prirejanja zbirčnih imen (za vhod, izhod in kazalo) in s tem prirejanjem povezanih niznih spremenljivk (vsrajeni tip 'strins') za zbirčna imena. Prosram bo torej z redkimi izjemami prenosljiv na različne računalniške sisteme oziroma njihove pascalske prevajalnike.

Podroben študij prosrama v listi 1 je priporočljiv zlasti za študente, saj daje možnosti spoznavanja niznih tipov, operacij nad nizi in niznih manipulacij v okviru standardne Wirthove pascalsce izvedenke.

Prosram je tudi dovolj obsežen, strukturirana sradnja je priporočljiva. Zaradi večje jasnosti so nekatere procedure (besedilo, kazalo) namerno podvojene (niso zlite v enotne procedure s posebnimi parametri). Prosram torej ni optimiziran, vendar bomo kasneje objavili v tej rubriki tudi njesovo optimalno predstavitev in razširitev na dodatne ukaze.

3. Opis oblikovalniškesa prosrama

Prosram Lenca v listi 1 obsesa 7 in 1/3 strani. Sestavljen je iz začetnesa pojasnjevalnesa besedila, iz ene same oznake, iz petih tipov

```
PROGRAM lenca (input, output);

(*--------------------------------------------*)
(*    Lastnosti besedilnega oblikovalnika    *)
(*              L e n c a                     *)
(*                                            *)
(* Ta program je splosen oblikovalnik bese-  *)
(* dil in uporablja v besedilo vgnezdene 4-  *)
(* znakovne oblikovalne ukaze brez in z pa-  *)
(* rametri.      Ti ukazi imajo obliko '*XX,'*)
(* in '*XX,nn,', kjer je XX ukazna mnemonika *)
(* in nn parameter (najvec stirimesten).     *)
(*--------------------------------------------*)
(*                                            *)
(* Vhod in izhod oblikovalnika:              *)
(*                                            *)
(* Vhod oblikovalnika je besedilo, ki vsebu- *)
(* je posebna znakovna zaporedja (ukazne ni- *)
(* ze) kot oblikovalne ukaze.  Izhod obliko- *)
(* valnika je nova izdaja  vhodnega besedila *)
(* brez oblikovalnih ukazov in je oblikovana *)
(* tako, kot narekujejo  vgnezdeni ukazi. U- *)
(* kazi  in  njihovi  učinki so  prikazani v *)
(* v proceduri 'ukazsezn'. Oblikovalnik izda *)
(* tudi s stranmi ostevilceno kazalo.        *)
(*--------------------------------------------*)
(*                                            *)
(* Zacetna nastavitev oblikovalnika:         *)
(*                                            *)
(* Pred  spremembami z  vgnezdenimi ukazi je *)
(* zacetna nastavitev oblikovalnika tale:    *)
(*                                            *)
(* Levi rob:            0 presledkov          *)
(* Dolzina vrstice: 47 znakov (stolpcev)     *)
(* Gornji rob:          4 vrstic             *)
(* Dolzina strani:  55 vrstic                *)
(* Dolzina vrstice v kazalu brez ostevilce-  *)
(* nja je vobce: dolzina vrstice - 7         *)
(*--------------------------------------------*)
(*                                            *)
(* Omejitve in delovanje oblikovalnika:      *)
(*                                            *)
(* Ukazi, ki nimajo pomena, se ne  upošteva- *)
(* jo in ostanejo kot nizi v izhodnem besedi-*)
(* lu. Program  uporablja  zamisem  končnega *)
(* avtomata  (prehajanja stanj) pri obdelavi *)
(* vhodnih znakov, pri  njihovem  stikanju v *)
(* v besede, pri zbiranju besed v vrstico in *)
(* pri izvajanju ukazov. Splosen nacrt obli- *)
(* kovalnika kot programa je tale:           *)
(*                                            *)
(*    Inicializacija.                        *)
(*    Obdelava vhodnih znakov do pojavitve   *)
(*      znaka 'konec_zbirke':                *)
(*      Vzemi znak in doloci njegov razred.  *)
(*      Uporabi razred in trenutno stanje    *)
(*        pri izbiri naslednje akcije.       *)
(*    Koncaj z zadnjo besedilno vrstico.     *)
(*--------------------------------------------*)
LABEL 1;

CONST
  maksdol = 132; (* Najvecja dolzina niza   *)
  ukazobseg = 19;(* Stevilo ukazov oblikoval.*)
(*--------------------------------------------*)

TYPE
  fiksniz = PACKED ARRAY Š 1 .. maksdol Č  OF
                                          char;

  niz =          (* Nizi razlicnih dolzin    *)
    RECORD
    nizi: fiksniz;     (* Nizno besedilo     *)
    dol: 0 .. maksdol  (* Dolzina niza       *)
    END;

  izhvrs =       (* Vrstica zbranih besed    *)
    RECORD
    izh: niz;        (* Vrstica, ki bo izdana *)
    aktstl: integer (* Položaj akcijskega    *)
    END;       (* stolpca: zadnji stolpec + 1 *)
```

```
  (* Znakovni razredi so v bistvu osnovna    *)
  (*   stanja obdelave:                      *)

  znakrazr =       (* Znakovni razredi:       *)
    (presl,        (*   presledek,            *)
     vejica,       (*   koncni znak ukaza,    *)
     konvrs,       (*   znak konca vrstice,   *)
     konzbir,      (*   znak konca zbirke in  *)
     nepresl);     (*   vsi ostali znaki.     *)

(* Ukazni razredi  so v  bistvu  podstanja   *)
(*   vejicnega stanja (ukaznega sufiksa):    *)

  ukazrazr = (* Ukazni razredi:               *)
    (levirob, (*  *LR,m,  (levi rob),         *)
     dolvrs,  (*  *DV,m,  (dolzina vrstice),*)
     gorrob,  (*  *GR,m,  (gornji rob),       *)
     dolstr,  (*  *DS,m,  (dolzina strani),   *)
     novavrs, (*  *NV,    (nova vrstica),     *)
     vecvrs,  (*  *SK,m,  (prek m vrstic),    *)
     novastr, (*  *NS,    (nova stran),       *)
     naslov,  (*  *NA,    (oblik. naslova),   *)
     levapor, (*  *LP,    (leva poravnava),   *)
     center,  (*  *CE,    (centriranje),      *)
     desnapor,(*  *DP,    (desna poravnava),*)
     koment,  (*  *KO,    (komentar),         *)
     naslzn,  (*  *NZ,    (naslednji znak),   *)
     nesprem, (*  *NE,    (nespremenjeno),    *)
     ustav,   (*  *US,    (ustavitev izdaje)*)
     stran,   (*  *SS,m,  (nast. strani),     *)
     tparam,  (*  *TP,    (tren. parametri),*)
     ukazi,   (*  *UK,    (izpis ukazov),     *)
     niukaz); (*  Niz, ki koncuje z ',', ni   *)
              (*  ukaz.                       *)

(*--------------------------------------------*)

VAR
  vrsta: izhvrs; (* Vrstica besed za izdajo. *)
  beseda: niz;   (* Trenutna vhodna beseda.  *)
  znak: char;    (* Trenutni vhodni znak.    *)
  razred: znakrazr;
               (* Vhodni znakovni razred.    *)
  nicla: niz;    (* Prazen niz.              *)
  znpresl: char; (* Znacilni presl. chr(0).  *)
  razf: ARRAY Š char Č OF znakrazr; (* Funk- *)
    (* cija za klasifikacijo vhodnih znakov. *)

  ststr:integer; (* Stevilka tekoce strani.  *)
  stvr: integer; (* Stevilka tekoce vrtsice. *)
  lrob: integer; (* Levi rob.                *)
  grob: integer; (* Gornji rob.              *)
  dvrs: integer; (* Dolzina vrstice.         *)
  dstr: integer; (* Dolzina strani.          *)

(* Spremenljivke, povezane s kazalom:        *)
  kazvrsta: izhvrs;    (* Vrsta za kazalo.   *)
  kazbeseda: niz;      (* Beseda za kazalo.  *)
  kazststr: integer;   (* Stev.str.kazala.   *)
  kazstvr: integer;    (* St.vrst.kazala.    *)
  kazdvrs: integer;    (* Dol.vrst.kazala.   *)

  vv, xx: char;  (* Pomozni spremenljivki.   *)

  ukaz: ukazrazr;     (* Trenutni vhodni ukaz. *)
  ukazparam: integer; (* Ukazni parameter.   *)
  uk: ARRAY Š 1 .. ukazobseg, 1 .. 3 Č OF
           char; (* Polje za ukazno mnemoniko*)
  ukazf: ARRAY Š 1 .. ukazobseg Č OF ukazrazr;

  i, ii, iii: integer;

  vhime, izhime, kazime: stringŠ16Č;
  vhzbir, izhzbir, kazzbir: text;
  ij: Boolean; (* Ostevilcenje str.v kazalu  *)
(*--------------------------------------------*)

PROCEDURE ukazsezn;
BEGIN (* ukazsezn *)
  (* Procedura izpise pomen posameznih ukazov*)
writeln;
writeln('   Ukazni seznam oblikovalnika:');
writeln;
```

```
writeln('    *LR,m,    Sprememba levega roba,');
writeln('    *DV,m,    Sprememba dolz.vrstice,');
writeln('    *GR,m,    Sprememba gornj. roba,');
writeln('    *DS,m,    Sprememba dolz.strani,');
writeln('    *NV,      Prehod na novo vrstico,');
writeln('    *SK,m,    Prehod prek m vrstic,');
writeln('    *NS,      Prehod na novo stran,');
writeln('    *NA,      Naslovi s kazalom,');
writeln('    *LP,      Leva poravnava vrstice,');
writeln('    *CE,      Centriranje vrstice,');
writeln('    *DP,      Desna poravn. vrstice,');
writeln('    *KO,      Komentarska vrstica,');
writeln('    *NZ,      Tiskanje nasled. znaka');
writeln('              je nespremenjeno,');
writeln('    *NE,      Tiskanje znakov med o-');
writeln('              mej. *NE, je nesprem.,');
writeln('    *US,      Ustavitev izpisa,');
writeln('    *SS,m,    Stevilka strani (nast.)');
writeln('    *TP,      Trenutni parametri,');
writeln('    *UK,      Seznam ukazov.')
END; (* ukazsezn *)
(*-----------------------------------------*)

PROCEDURE trennast;
  (* Procedura  izpise  trenutno nastavljene *)
  (* parametre.                              *)
  BEGIN (* trennast *)
  writeln;
  writeln('Trenutno veljavni parametri:');
  writeln;
  writeln('Gornji rob       = ', grob);
  writeln('Levi rob         = ', lrob);
  writeln('Dolzina vrstice  = ', dvrs);
  writeln('Dolzina strani   = ', dstr);
  writeln('Stevilka strani  = ', ststr);
  writeln;
  writeln('Vrstica kazala   = ', kazstvr);
  writeln('Stran kazala     = ', kazststr)
  END; (* trennast *)
(*-----------------------------------------*)

PROCEDURE ukaztab;

  (* S to proceduro se  definira ukazna mne- *)
  (* monika, ki je  sestavljena iz dveh zna- *)
  (* kov. Tretji element  pove, ali ima ukaz *)
  (* se ukazni parameter (vrednost '1').     *)
  (* Nadalje se definira se ukazna funkcija. *)
  VAR
    i, j: integer;

  BEGIN

  FOR i := 1 TO ukazobseg DO
    FOR j := 1 TO 3 DO
      ukŠi,jĆ := '0';

  ukŠ 1,1Ć :='L';ukŠ 1,2Ć :='R';ukŠ 1,3Ć :='1';
  ukŠ 2,1Ć :='D';ukŠ 2,2Ć :='V';ukŠ 2,3Ć :='1';
  ukŠ 3,1Ć :='G';ukŠ 3,2Ć :='R';ukŠ 3,3Ć :='1';
  ukŠ 4,1Ć :='D';ukŠ 4,2Ć :='S';ukŠ 4,3Ć :='1';
  ukŠ 5,1Ć :='N';ukŠ 5,2Ć :='V';
  ukŠ 6,1Ć :='S';ukŠ 6,2Ć :='K';ukŠ 6,3Ć :='1';
  ukŠ 7,1Ć :='N';ukŠ 7,2Ć :='S';
  ukŠ 8,1Ć :='N';ukŠ 8,2Ć :='A';
  ukŠ 9,1Ć :='L';ukŠ 9,2Ć :='P';
  ukŠ10,1Ć :='C';ukŠ10,2Ć :='E';
  ukŠ11,1Ć :='D';ukŠ11,2Ć :='P';
  ukŠ12,1Ć :='K';ukŠ12,2Ć :='O';
  ukŠ13,1Ć :='N';ukŠ13,2Ć :='Z';
  ukŠ14,1Ć :='N';ukŠ14,2Ć :='E';
  ukŠ15,1Ć :='U';ukŠ15,2Ć :='S';
  ukŠ16,1Ć :='S';ukŠ16,2Ć :='S';ukŠ16,3Ć :='1';
  ukŠ17,1Ć :='T';ukŠ17,2Ć :='P';
  ukŠ18,1Ć :='U';ukŠ18,2Ć :='K';

  (* Definicija ukazne funkcije:             *)
  ukazfŠ 1Ć := levirob;  ukazfŠ 2Ć := dolvrs;
  ukazfŠ 3Ć := gorrob;   ukazfŠ 4Ć := dolstr;
  ukazfŠ 5Ć := novavrs;  ukazfŠ 6Ć := vecvrs;
  ukazfŠ 7Ć := novastr;  ukazfŠ 8Ć := naslov;
  ukazfŠ 9Ć := levapor;  ukazfŠ10Ć := center;
  ukazfŠ11Ć := desnapor; ukazfŠ12Ć := koment;
  ukazfŠ13Ć := naslzn;   ukazfŠ14Ć := nesprem;
  ukazfŠ15Ć := ustav;    ukazfŠ16Ć := stran;
  ukazfŠ17Ć := tparam;   ukazfŠ18Ć := ukazi;
  ukazfŠ19Ć := niukaz;

  END;
(*-----------------------------------------*)

PROCEDURE dialog;
  (* S to proceduro se lahko nadaljuje obli- *)
  (* kovanje besedila,  vstavljajo se  imena *)
  (* zadevnih zbirk, odpirajo zadevne zbirke *)
  (*in sporocajo odpiralne napake.           *)

  BEGIN (* dialog *)
  writeln;
  writeln(' Oblikovalnik Lenca, tip 1.4/1985');
  writeln;
  write('Ali zelis oblikovati besedilo ');
  write('(d/n)? '); readln(xx); writeln;
  IF (xx <> 'd') AND (xx <> 'D') THEN exit;
  write('Vstavi ime  vhodne zbirke: ');
  readln(vhime);
  assign(vhzbir, vhime); reset(vhzbir);
  IF ioresult = 255 THEN
    BEGIN write('Vhodne zbirke ', vhime);
    writeln(' ni mogoce odpreti.'); exit END;
  write('Vstavi ime izhodne zbirke: ');
  readln(izhime);
  assign(izhzbir, izhime); rewrite(izhzbir);
  IF ioresult = 255 THEN
    BEGIN write('Izhodne zbirke ', izhime);
    writeln(' ni mogoce odpreti.'); exit END;
  write('Vstavi ime kazala:        : ');
  readln(kazime);
  assign(kazzbir, kazime); rewrite(kazzbir);
  IF ioresult = 255 THEN
    BEGIN write('Kazalne zbirke ', kazime);
    writeln(' ni mogoce odpreti.'); exit END;
  writeln;
  writeln('POCAKAJ NA KONEC OBDELAVE ! ! !')
  END; (* dialog *)
(*-----------------------------------------*)

PROCEDURE zapiranje;
  (* Zaprejo se vse odprte  zbirke, in sicer *)
  (* vhzbir, izhzbir in kazzbir.             *)

  BEGIN (* zapiranje *)
  close(vhzbir, i);
  IF i = 255 THEN
    BEGIN write('Vhodne zbirke ', vhime);
    writeln(' ni mogoce zapreti.'); exit END;
  close(izhzbir, ii);
  IF ii = 255 THEN
    BEGIN write('Izhodne zbirke ', izhime);
    writeln(' ni mogoce zapreti.'); exit END;
  close(kazzbir, iii);
  IF iii = 255 THEN
    BEGIN write('Kazalne zbirke ', kazime);
    writeln(' ni mogoce zapreti.'); exit END
  END; (* zapiranje *)
(*-----------------------------------------*)

PROCEDURE zacetek;
  (* Nastavitev  zacetnih  globalnih  spre-  *)
  (* menljivk ukazf, razf, nicla,  znpregl,  *)
  (* ukaz, beseda, razred.                   *)

  VAR
    znak: char;     (* Indeks za razf.       *)
    i: 1 .. maksdol; (* Indeks za polje      *)
                     (*   nicla.nizl.        *)

  BEGIN (* zacetek *)

  ukaztab;      (* Definiranje mnemonike.    *)

  (* Nastavitev zacetnih urejevalniskih pa-  *)
  (* rametrov:                               *)

  lrob :=  0;  (* Levi rob.                  *)
```

```
  grob  :=  4;  (* Gornji rob: ne sme  biti  *)
                (* manjsi od 3.              *)
  dvrs  := 47;  (* Dolzina vrstice.          *)
  dstr  := 55;  (* Dolzina strani.           *)
  stvr  :=  0;  (* Tekoca vrstica.           *)
  ststr :=  0;  (* Tekoca stran.             *)

  kazdvrs := dvrs-7;
                 (* Dolzina vrstice kazala. *)
  kazstvr :=  0;  (* Tekoca vrstica kazala.  *)
  kazststr:=  0;  (* Tekoca stran kazala.    *)

  (* Nastavi funkcijsko polje za klasifika-  *)
  (*   cijo vhodnih znakov:                  *)

  FOR znak := chr(0) TO chr(254) DO
    razfŠznakĆ  := nepresl;
  razfŠ' 'Ć     := presl;
  razfŠ','Ć     := vejica;
  razfŠchr(13)Ć := konvrs;
  razfŠchr(26)Ć := konzbir;

  (* Inicializacija praznega niza za prazni- *)
  (*   nitev drugih nizov:                   *)
  FOR i := 1 TO maksdol DO
    nicla.niz1ŠiĆ := ' ';
  nicla.dol := 0;

  (* Nastavitev znacilnega presledka:        *)
  znpresl := chr(0);

  beseda := nicla; kazbeseda := nicla;
  zacvrsta(vrsta); zacvrsta(kazvrsta);
  ukaz := niukaz;
  razred := nepresl
  END; (* zacetek *)
(*-------------------------------------------*)

PROCEDURE vzemiznak ( VAR znak: char;
                      VAR razred: znakrazr );
  (* Procedura  vzame naslednji znak iz vho- *)
  (* dne zbirke, poisce njegov razred in vr- *)
  (* ne presledek, če je znak tipa 'eof' ali *)
  (* 'eoln'.                                 *)

  BEGIN (* vzemiznak *)
    IF eof(vhzbir) THEN
      BEGIN razred := konzbir; znak := ' ' END
    ELSE IF eoln(vhzbir) THEN
      BEGIN razred := konvrs; readln(vhzbir);
            znak := ' ' END
    ELSE BEGIN read(vhzbir, znak);
              razred := razfŠznakĆ END
  END; (* vzemiznak *)
(*-------------------------------------------*)

PROCEDURE stikznak ( znak: char; VAR s: niz );
  (* Znak se pritakne k s, ce je prostor.    *)

  BEGIN (* stikznak *)
    WITH s DO
      IF dol < maksdol THEN
        BEGIN
        dol := dol + 1;
        niz1ŠdolĆ := znak
        END
  END; (* stikznak *)
(*-------------------------------------------*)

PROCEDURE stikniz ( VAR s: niz; novo: niz );
  (* Niz novo se pritakne desno k s.         *)

  VAR
    i: 1 .. maksdol;  (* Zancni indeks.      *)

  BEGIN (* stikniz *)
    WITH s DO
      FOR i := 1 TO novo.dol DO
        BEGIN
        dol := dol + 1;
        niz1ŠdolĆ := novo.niz1ŠiĆ
        END
  END; (* stikniz *)
(*-------------------------------------------*)
```

```
PROCEDURE pomik ( VAR vrsta: izhvrs;
                  koncno: integer );
  (* Podniz vrstice (ki zacenja pri polozaju *)
  (* zadnje  akcije in  koncuje pri trenutni *)
  (* dolzini) se pomakne v desno do polozaja *)
  (* koncno.                                 *)

  VAR
    i: 1 .. maksdol;  (* Zancni indeks       *)
    prih: integer; (* Indeks za pomik znakov *)

  BEGIN (* pomik *)
  (* Pomik znakov in  nadomestitev  njihovih *)
  (* starih položajev s presledki.           *)
  WITH vrsta, izh DO
    IF (aktstl <= dol) AND (dol < dvrs) AND
      (dol < koncno) THEN
      BEGIN
      prih := koncno;
      FOR i := dol DOWNTO aktstl DO
        BEGIN (* posamicni pomiki *)
        niz1ŠprihĆ := niz1ŠiĆ;
        niz1ŠiĆ := ' ';
        prih := prih - 1
        END (* posamični pomiki *)
      END;
  (* To je sedaj mesto zadnje akcije:        *)
  WITH vrsta, izh DO
    BEGIN
    dol := koncno;
    aktstl := dol + 1
    END
  END; (* pomik *)
(*-------------------------------------------*)

PROCEDURE poravnava ( VAR vrsta: izhvrs;
                      smer: ukazrazr );
  (* Poravnava skupine besed v vrsto skladno *)
  (* s smerjo (levo, sredinsko, desno):      *)

  VAR
    novadol: integer; (* Dolzina po poravnavi*)

  BEGIN (* poravnava *)

  (* Dolocitev nove dolzine vrste:           *)
  WITH vrsta, izh DO
    CASE smer OF
      levapor:
        (* Po  definiciji je  dolzina  tocka *)
        (* poravnave:                        *)
        novadol := dol;
      center:
        (* Dolzina je enaka polovici dolzine *)
        (* besedne skupine plus polovica naj-*)
        (* vecje dolzine vrstice:            *)
        novadol := ((dol-aktstl+1) DIV 2) +
                   (dvrs DIV 2);
      desnapor:
        (* Besede se  pomaknejo  do  konca v *)
        (* desno.                            *)
        novadol := dvrs
    END; (* Primeri smeri. *)
  (* Nastane podniz s poravnanimi  besedami: *)
  pomik ( vrsta, novadol )
  END; (* poravnava *)
(*-------------------------------------------*)

PROCEDURE justiranje ( VAR vrsta: izhvrs;
                       dvrs: integer );
  (* Vrsta  se  justira  na  dolzino  dvrs z *)
  (* vstavitvijo dodatnih presledkov med be- *)
  (* sede od  leve  proti  desni. Na zacetku *)
  (* ima vrstica po en  presledek med vsebo- *)
  (* vanimi besedami.                        *)

  VAR
    presledki: integer;  (* Stevilo presled- *)
                (* kov, ki bodo vstavljeni *)
    reze: integer;       (* Stevilo presled- *)
```

```
                    (* kov med besedami vrstice*)
   n: integer;    (* Obseg  razsiritve  reze *)
   prih: integer;(* Novo mesto za pomaknje- *)
                  (* ni znak.                *)
   vir: integer; (* Izvirni  stolpec  tega   *)
                  (* znaka.                  *)

 BEGIN (* Justiranje *)

 WITH vrsta, izh DO
   (* Razsiritev vrstice, ce je prekratka:  *)
   IF dol < dvrs THEN
     BEGIN
     (* Prestetje rez med besedami:         *)
     reze := 0;
     FOR vir := 1 TO dol DO
       IF niz1ŠvirČ = ' ' THEN
         reze := reze + 1;
     (* Poisci stevilo  presledkov, potreb- *)
     (* nih za razsiritev vrste:            *)
     presledki := dvrs - dol;
     (* Pomikanje znakov v desno z vstavlja-*)
     (* njem dodatnih presledkov med besede:*)
     prih := dvrs;
     vir := dol;
     WHILE reze > 0 DO
       BEGIN (* Razsiritev vrstice: *)
       IF niz1ŠvirČ <> ' ' THEN
         BEGIN (* Pomikanje znakov: *)
         (* Pomaknitev  znaka in  vstavitev *)
         (* presledka na njegovo mesto:     *)
         niz1ŠprihČ := niz1ŠvirČ;
         niz1ŠvirČ := ' '
         END (* Pomikanje znakov *)
       ELSE
         BEGIN (* Puscanje presledkov *)
         (* Poisci stevilo presledkov za to *)
         (* rezo in jih preskoci:           *)
         n := presledki DIV reze;
         prih := prih - n;
         reze := reze - 1;
         presledki := presledki - n;
         END; (* Puscanje presledkov *)
       (* Upostevanje naslednjega izvira in *)
       (* prihoda za znake:                 *)
       vir := vir - 1;
       prih := prih - 1
       END; (* Razsiritev vrstice *)
     dol := dvrs
     END
 END; (* Justiranje *)
(*-----------------------------------------*)

PROCEDURE vrsstr;
  (* Prehod na  novo  stran, izpis  gornjega *)
  (* roba, stevilke strani in  presledka med *)
  (* stevilko  strani in  besedilom.         *)

  VAR k: integer;

  BEGIN (* vrsstr *)
  IF (stvr=dstr) OR
              ((stvr=0) AND (ststr=0)) THEN
    BEGIN (* if stavek *)
    ststr := ststr + 1; page (izhzbir);
    FOR k := 1 TO grob-3 DO writeln(izhzbir);
    (* Izpis stevilke strani v besedilu:     *)
    IF ststr<100 THEN
      writeln(izhzbir, ststr:lrob+dvrs DIV 2+1)
    ELSE
      writeln(izhzbir, ststr:lrob+dvrs DIV 2+2);
    writeln(izhzbir); writeln(izhzbir);
    stvr := 0
    END (* if stavek *)
  END; (* vrsstr *)
(*-----------------------------------------*)

PROCEDURE vrsst1;
  (* Tj. 'vrsstr' za kazalo.                 *)
  VAR k: integer;
  BEGIN (* vrsst1 *)
  IF (kazstvr=dstr) OR
             ((kazstvr=0) AND (kazststr=0)) THEN
     BEGIN (* if stavek *)
     kazststr := kazststr + 1; page(kazzbir);
     FOR k := 1 TO grob-3 DO writeln(kazzbir);
     (* Izpis stevilke strani v kazalu:    *)
     IF kazststr<100 THEN
     writeln(kazzbir,'K-':lrob+dvrs DIV 2+1,
                                   kazststr)
     ELSE
       writeln(kazzbir,'K-':lrob+dvrs DIV 2,
                                   kazststr);
     writeln(kazzbir); writeln(kazzbir);
     kazstvr := 0;
     IF kazststr=1 THEN
       BEGIN
       writeln(kazzbir, 'K a z a l o':
                           lrob+dvrs DIV 2+6);
       writeln(kazzbir); writeln(kazzbir);
       kazstvr := 3
       END
     END (* if stavek *)
   END; (* vrsst1 *)
(*-----------------------------------------*)

PROCEDURE izpis ( s: niz );
 (* Izpisovanje  levega  roba in niza s  (s *)
 (* spremembo  znacilnih  presledkov v pre- *)
 (* sledke) v eno vrstico.                 *)

 VAR
   i: 1 .. maksdol; (* Korakanje skozi s.  *)
   j: integer;

 BEGIN (* izpis *)
 vrsstr;
 FOR j := 1 TO lrob DO (* Izpis  'lrob'     *)
   write(izhzbir, ' '); (* presledkov *)
 WITH s DO
   FOR i := 1 TO dol DO
     IF niz1ŠiČ = znpresl THEN
       write(izhzbir, ' ')
     ELSE
       write(izhzbir, niz1ŠiČ);
 writeln(izhzbir); stvr := stvr + 1
 END; (* izpis *)
(*-----------------------------------------*)

PROCEDURE izpis1 ( s: niz );
  (* Tj. izpis za kazalo!                   *)
  VAR i: 1 .. maksdol; j: integer;
  BEGIN (* izpis1 *)
  vrsst1;
  FOR j := 1 TO lrob DO
    write(kazzbir, ' ');
  WITH s do
    BEGIN
    FOR i := 1 TO dol DO
      IF niz1ŠiČ = znpresl THEN
        write(kazzbir, ' ')
      ELSE
        write(kazzbir, niz1ŠiČ);
    IF ij THEN
      BEGIN write(kazzbir, ststr: dvrs-dol);
      ij := false  END
    END;
  writeln(kazzbir); kazstvr := kazstvr + 1
  END; (* izpis1 *)
(*-----------------------------------------*)

PROCEDURE zacvrsta ( VAR vrsta: izhvrs );
  (* Inicializacija vrste v prazno vrsto.    *)

  BEGIN (* zacvrsta *)
  WITH vrsta DO
    BEGIN
    izh := nicla;
    aktstl := 1
    END
  END; (* zacvrsta *)
(*-----------------------------------------*)

PROCEDURE konvrsta;
  (* Izpis vrste in njena inicializacija.    *)
```

```
  BEGIN (* konvrsta *)
  izpis(vrsta.izh);
  zacvrsta(vrsta)
  END; (* konvrsta *)
(*-----------------------------------------------*)

PROCEDURE konvrs1;
  (* Tj. konvrsta za kazalo!                      *)
  BEGIN (* konvrs1 *)
  izpis1(kazvrsta.izh);
  zacvrsta(kazvrsta)
  END; (* konvrs1 *)
(*-----------------------------------------------*)

PROCEDURE vstavitev;
  (* Vstavitev besede na  konec vrste (ce je *)
  (* je to mogoce). Sicer konvrs in  ponovi- *)
  (* tev  postopka.  Ponovna  inicializacija *)
  (* besede v prazno besedo.                 *)

  BEGIN (* vstavitev *)
  (* Poskusi vstaviti besedo, pred katero je *)
  (* presledek, na konec vrstice:            *)
  IF beseda.dol = 0 THEN
    (* ne naredi nicesar                     *)
  ELSE IF (vrsta.izh.dol + beseda.dol + 1) <=
          dvrs THEN
    BEGIN (* Vstavitev je mogoca.            *)
    IF vrsta.izh.dol > 0 THEN
      stikznak(' ', vrsta.izh);
    stikniz(vrsta.izh, beseda)
    END (* Vstavitev je mogoca.              *)
  ELSE
    (* Kombinacija je predolga. Ali je pre-  *)
    (* dolga beseda?                         *)
    IF beseda.dol >= dvrs THEN
      BEGIN (* Predolga beseda. *)
      (* Izdaja vrstice                      *)
      justiranje(vrsta, dvrs);
      konvrsta;
      izpis(beseda)
      END (* Predolga beseda. *)
    ELSE
      BEGIN (* Normalni prestop. *)
      justiranje(vrsta, dvrs);
      konvrsta;
      (* Zacetek nove vrstice.               *)
      stikniz(vrsta.izh, beseda)
      END; (* Normalni prestop. *)
    beseda := nicla
    END; (* vstavitev *)
(*-----------------------------------------------*)

PROCEDURE vstav1;
  (* Tj. 'vstavitev' za kazalo.              *)
  BEGIN (* vstav1 *)
  IF kazbeseda.dol = 0 THEN
  ELSE IF (kazvrsta.izh.dol+kazbeseda.dol+1)
          <= kazdvrs  THEN
    BEGIN
    IF kazvrsta.izh.dol > 0 THEN
      stikznak(' ', kazvrsta.izh);
    stikniz(kazvrsta.izh, kazbeseda)
    END
  ELSE IF kazbeseda.dol >= kazdvrs THEN
    BEGIN
    justiranje(kazvrsta, kazdvrs);
    konvrs1; izpis1(kazbeseda);
    END
  ELSE
    BEGIN
    justiranje(kazvrsta, kazdvrs);
    konvrs1; stikniz(kazvrsta.izh, kazbeseda)
    END;
  kazbeseda := nicla
  END; (* vstav1 *)
(*-----------------------------------------------*)

PROCEDURE vvrstic;
  (* Procedura izpise ukazparam praznih vrst,*)
  (* ce obstaja prostor na tekoci strani, si-*)
  (* cer izpise to na naslednji strani.  Pri *)
  (* izpisu naslovov (ukaz *NA,) v besedilo se *)
  (* uporablja  procedura 'vvrst1', ki je 'po- *)
  (* dobna' tej proceduri.                     *)
  VAR i: integer;

  BEGIN (* vvrstic *)
  vstavitev;
  konvrsta;
  IF ukazparam >= dstr-stvr THEN
    BEGIN
    stvr := dstr;
    vrsstr
    END;
  FOR i := 1 TO ukazparam DO
    BEGIN
    writeln(izhzbir); stvr := stvr + 1
    END
  END; (* vvrstic *)
(*-----------------------------------------------*)

PROCEDURE vvrst1;
  (* Procedura se uporablja pri izpisu nasl. *)
  BEGIN (* vvrst1 *)
  vstavitev;
  konvrsta;
  IF 10 >= dstr-stvr THEN
    BEGIN stvr := dstr; vrsstr  END
  END; (* vvrst1 *)
(*-----------------------------------------------*)

PROCEDURE razmnem ( u1: char; u2: char );
  (* Razpoznavanje ukazne mnemonike.           *)

  BEGIN (* razmnem *)
  WITH beseda DO
    BEGIN (* with stavek *)
    IF dol < 4 THEN
      ukaz := niukaz
    ELSE IF (niz1Šdol-3Č <> '*') THEN
      ukaz := niukaz
    ELSE IF ((niz1Šdol-2Č = u1) AND
            (niz1Šdol-1Č = u2)) THEN
      ukaz := naslov
    ELSE ukaz := niukaz
    END (* with stavek *)
  END; (* razmnem *)
(*-----------------------------------------------*)

PROCEDURE nasl1;
  (* Procedura  izpise  naslov  med  ukazoma *)
  (* '*NA,' v  izbrano  zbirko in v kazalo s *)
  (* pripadajoco stevilko strani.            *)
  VAR i: integer;

  BEGIN (* nasl1 *)
  ukaz := niukaz; ij := true;
  (* Izpis ostanka in  priprava za  naslov v *)
  (* besedilu:                               *)
  vvrst1;
  IF stvr > 1 THEN
    (* Vstavijo se 3 prazne vrst. v besedilo *)
    FOR i := 1 TO 3 DO
      BEGIN writeln(izhzbir); stvr := stvr + 1
      END;
  REPEAT
    vzemiznak(znak, razred);
    CASE razred OF
      nepresl:
        BEGIN
        stikznak(znak, beseda);
        stikznak(znak, kazbeseda)
        END;
      presl, konvrs, konzbir:
        BEGIN vstavitev; vstav1 END;
      vejica:
        BEGIN (* vejica *)
        stikznak(znak, beseda);
        stikznak(znak, kazbeseda);
        razmnem('N','A');
        IF ukaz <> niukaz THEN
          (* Imamo konec naslova               *)
          BEGIN (* if stavek *)
```

```
          zbrisi( 4,beseda );
          zbrisi( 4,kazbeseda );
          vstavitev; konvrsta;
          vstavl; konvrsl;
          FOR i := 1 TO 2 DO
            BEGIN writeln(izhzbir);
              stvr := stvr + 1 END
          END (* if stavek *)
        END (* vejica *)
    END (* case stavek *)
  UNTIL (razred = konzbir) OR (ukaz <> niukaz)
  END; (* nasl1 *)
(*-----------------------------------------------*)

PROCEDURE nespr1;
  (* Procedura izpisuje besedilo med ukazoma *)
  (* '*NE,' v izbrano zbirko z  upoštevanjem *)
  (* levega roba (lrob).                     *)
  PROCEDURE izp1;
    (* Procedura izpise besedo, ko se pojavi *)
    (* presl, konvrs ali konzbir.            *)
    VAR i: 1 .. maksdol; j: integer;
    BEGIN (* izp1 *)
    vrsstr;
    FOR j := 1 TO lrob DO
      write(izhzbir,' ');
    WITH beseda DO
      FOR i := 1 TO dol DO
        write(izhzbir, niz1ŠiĆ);
    beseda := nicla
    END; (* izp1 *)

  BEGIN (* nespr1 *)
  ukaz := niukaz; vstavitev; konvrsta;
  REPEAT
    vzemiznak(znak, razred);
    CASE razred OF (* case stavek *)
      nepresl, presl:
        stikznak(znak, beseda);
      konvrs, konzbir:
        BEGIN izp1; writeln(izhzbir);
        stvr := stvr + 1  END;
      vejica:
        BEGIN (* vejica *)
        stikznak(znak, beseda);
        razmnem('N','E');
        IF ukaz <> niukaz THEN
          BEGIN zbrisi(4,beseda); izp1;
          writeln(izhzbir); stvr := stvr + 1
          END
        END (* vejica *)
    END (* case stavek *)
  UNTIL (razred = konzbir) OR (ukaz <> niukaz)
  END; (* nespr1 *)
(*-----------------------------------------------*)

PROCEDURE kom1;
  (* Procedura izpise  komentar med  ukazoma *)
  (* '*KO,' na zaslon brez levega roba.      *)
  PROCEDURE izp2;
    (* Procedura izpise besedo.              *)
    VAR i: 1 .. maksdol;
    BEGIN (* izp2 *)
    WITH beseda DO
      FOR i := 1 TO dol DO
        write(niz1ŠiĆ);
    beseda := nicla
    END; (* izp2 *)

  BEGIN (* kom1 *)
  ukaz := niukaz; vstavitev; konvrsta;
  REPEAT
    vzemiznak(znak, razred);
    CASE razred OF (* case stavek *)
      nepresl:
        stikznak(znak, beseda);
      presl:
        BEGIN izp2; write(' ') END;
      konvrs, konzbir:
        BEGIN izp2; writeln END;
      vejica:
        BEGIN (* vejica *)
        stikznak(znak, beseda);
        razmnem('K','O');
        IF ukaz <> niukaz THEN
          BEGIN zbrisi(4, beseda); izp2;
          writeln END
        END (* vejica *)
    END (* case stavek *)
  UNTIL (razred = konzbir) OR (ukaz <> niukaz)
  END; (* kom1 *)
(*-----------------------------------------------*)

PROCEDURE naslz1;
  (* Procedura izpise znak za ukazom  '*NZ,' *)
  (* nespremenjeno.                          *)

  BEGIN (* naslz1 *)
  vzemiznak(znak, razred);
  CASE razred OF
    nepresl, presl, vejica:
      stikznak(znak, beseda);
    konvrs:
      BEGIN vstavitev; konvrsta END;
    konzbir:
      vstavitev
  END
  END; (*naslz1 *)

(*-----------------------------------------------*)
(*   Procedure in funkciji za razpoznavanje   *)
(*            oblikovalnih ukazov              *)
(*-----------------------------------------------*)

PROCEDURE zbrisi (     n: integer;
                   VAR beseda: niz );
  (* Zbrise se  zadnjih  'n' znakov v besedi *)
  VAR i: integer;

  BEGIN (* zbrisi *)
  WITH beseda DO
    BEGIN (* with stavek *)
    IF dol >= n THEN
      FOR i := 1 TO n DO
        niz1Šdol+1-iĆ := ' ';
    dol := dol - n
    END (* with stavek *)
  END; (* zbrisi *)
(*-----------------------------------------------*)

PROCEDURE razpoparam (VAR ukaz: ukazrazr;
                      VAR ukazparam: integer );
  (* Razpoznavanje regularnosti parametra in *)
  (* dolocitev njegove vrednosti:             *)
  VAR i, j, k: integer;

  BEGIN (* razpoparam *)
  stikznak(znak, beseda);
  WITH beseda DO
    BEGIN (* with stavek *)
    k := dol - iii;
    IF k < 2 THEN
      ukaz := niukaz
    ELSE IF k > 5 THEN
      ukaz := niukaz
    ELSE
      (* Ali so v parametru stevilke ?          *)
      FOR i := 1 TO k-1 DO
        IF jestev(niz1Šiii+iĆ) THEN
          (* Imamo stevilko                     *)
        ELSE
          ukaz := niukaz;
    IF ukaz <> niukaz THEN
      (* Imamo veljaven parameter.              *)
      BEGIN (* je parameter *)
      ukazparam := 0;
      j := iii + 1; i := k - 2;
      REPEAT
        ukazparam := ukazparam +
          ((ord(niz1ŠjĆ)-ord('0'))*deset(i));
        j := j + 1; i := i - 1;
      UNTIL i = -1;
      zbrisi(4+k, beseda)
      END (* je parameter *)
    END (* with stavek *)
```

```
  END; (* razpoparam *)
(*-------------------------------------------*)

PROCEDURE razpoukaz ( VAR ukaz: ukazrazr;
                      VAR ii: integer );
  (* Ta procedura razpozna mnemonični del u- *)
  (* kaza brez parametra.                    *)

  BEGIN (* razpoukaz *)
  stikznak(znak, beseda);
  WITH beseda DO
    BEGIN (* with stavek *)
    iii := dol;
    IF dol < 4 THEN
      ukaz := niukaz
    ELSE IF (niz1Šdol-3Č <> '*') THEN
      ukaz := niukaz
    ELSE
      BEGIN (* dolocitev ukaznega tipa *)
      ii := 0;
      REPEAT
        ii := ii + 1
      UNTIL (((ukŠii,1Č = niz1Šdol-2Č) AND
             (ukŠii,2Č = niz1Šdol-1Č))    OR
            (ii = ukazobseg));
      ukaz := ukazfŠiiČ
      END; (* dolocitev ukaznega tipa *)
    IF (ukaz <> niukaz) AND
      (ukŠii,3Č = '0') THEN
      zbrisi(4, beseda)
    END (* with stavek *)
  END; (*-razpoukaz *)
(*-------------------------------------------*)

FUNCTION jestev ( znak: char ): Boolean;
  (* Ce je znak stevilka, je  vrednost funk- *)
  (* cije enaka true, sicer pa false.        *)
  BEGIN (* jestev *)
  IF ((ord(znak) >= ord('0')) AND
     (ord(znak) <= ord('9'))) THEN
    jestev := true
  ELSE
    jestev := false
  END; (* jestev *)
(*-------------------------------------------*)

FUNCTION deset ( i: integer): integer;
  (* Izracun vrednosti 10 na potenco i.      *)
  VAR x: integer; j: integer;
  BEGIN (* deset *)
  x := 1;
  FOR j := 1 TO i DO
    x := x*10;
  deset := x
  END; (* deset *)

(*-------------------------------------------*)
(*===========================================*)

BEGIN (* oblikovalnik *)
1:zacetek;
dialog;
IF ((xx <> 'd') AND (xx <> 'D')) OR
  (ioresult = 255) THEN exit;
zacvrsta(vrsta);

REPEAT
  vzemiznak(znak, razred);
  CASE razred OF (* osnovni case stavek *)
    nepresl:
      stikznak(znak, beseda);
    presl, konvrs, konzbir:
      vstavitev;
    vejica:
    (* Razpoznavanje oblikovalnih ukazov:     *)
      BEGIN (* vejica: *)
      razpoukaz(ukaz, ii);
      IF ukŠii,3Č = '1' THEN
        BEGIN (* razpozn. ukaza s parametrom *)
        REPEAT
          vzemiznak(znak, razred);
          CASE razred OF (* case param znaki *)
            nepresl:
              stikznak(znak, beseda);
            presl, konvrs, konzbir:
              BEGIN vstavitev; ukaz := niukaz
              END;
            vejica:
              BEGIN (* parametrski ukazi *)
              razpoparam(ukaz, ukazparam);
              CASE ukaz OF (* case 1 stavek *)
                levirob:
                  (* Sprememba levega roba:  *)
                  BEGIN lrob := ukazparam;
                  ukaz := niukaz END;
                dolvrs:
                  (* Sprememba dolz. vrstice:*)
                  BEGIN dvrs := ukazparam;
                  kazdvrs := dvrs - 7;
                  ukaz := niukaz END;
                gorrob:
                  (* Sprememba gornjega roba:*)
                  BEGIN grob := ukazparam;
                  ukaz := niukaz END;
                dolstr:
                  (* Sprememba dolzine strani*)
                  BEGIN dstr := ukazparam;
                  ukaz := niukaz END;
                vecvrs:
                  (* Prehod prek vec vrstic: *)
                  BEGIN
                  vvrstic; ukaz := niukaz
                  END;
                stran:
                  (* Sprememba stev. strani: *)
                  BEGIN
                  vstavitev; konvrsta;
                  ststr := ukazparam - 1;
                  stvr := dstr;
                  ukaz := niukaz
                  END;
                niukaz:
                  ukaz := niukaz
                END (* case 1 stavek *)
              END (* parameterski ukazi *)
            END (* case param znaki *)
        UNTIL ((razred = konzbir) OR
              (ukaz = niukaz))
        END (* razpozn. ukaza s parametrom *)
      ELSE
        (* Razpoznavanje ukaza brez parametra*)
        BEGIN
        CASE ukaz OF (* case 2 stavek *)
          novavrs:
            (* Prehod v novo vrstico:        *)
            BEGIN
            vstavitev;
            konvrsta
            END;
          novastr:
            (* Prehod na novo stran:         *)
            BEGIN
            vstavitev;
            konvrsta;
            stvr := dstr; vrsstr
            END;
          naslov:
            (* Oblikovanje  naslovnih vrstic *)
            (* in kazala:                    *)
            nasl1;
          levapor:
            (* Poravnava do levega roba:     *)
            BEGIN
            vstavitev;
            poravnava(vrsta, levapor)
            END;
          center:
            (* Centriranje besedila v vrst.: *)
            BEGIN
            vstavitev;
            poravnava(vrsta, center)
            END;
          desnapor:
            BEGIN
            (* Poravnava do desnega roba:    *)
            vstavitev;
```

```
        poravnava(vrsta, desnapor);
        konvrsta
        END;
      koment:
        (* Komentar se nahaja med ukazo- *)
        (* ma '*KO,'.                    *)
        kom1;
      naslzn:
        (* Naslednji znak se izpise v    *)
        (* vsakem primeru:               *)
        naslz1;
      nesprem:
        (* Besedilo med ukazoma *NE, se  *)
        (* izpise v nespremenjeni obliki:*)
        nespr1;
      ustav:
        (* Ustavitev izpisa dokler se    *)
        (* ne vtipka znak presledka:     *)
        BEGIN
        set(input); vv := inputČ;
        WHILE vv <> ' ' DO
          BEGIN
          set(input); vv := inputČ
          END
        END;
      tparam:
        trennast;
      ukazi:
        ukazsezn;
      niukaz:
        ukaz := niukaz
      END (* case 2 stavek *)
      END
    END (* vejica *)
  END (* osnovni case stavek *)
UNTIL razred = konzbir;
vstavitev; konvrsta; zapiranje;
IF (i = 255) OR (ii = 255) OR (iii = 255) THEN
  exit;
GOTO 1
END. (* oblikovalnik *)
```

Lista 1. Lista na prejšnjih sedmih straneh in na tej strani prikazuje pascalski program Lenca za oblikovanje vhodnih besedil. Ta program je zgrajen modularno, je pregleden in razumljiv. Glavni program začenja na prejšnji strani in je sestavljen iz glavne REPEAT zanke ter iz začetnega in končnega segmenta. V osnovnem CASE stavku sproži vejica v vhodnem besedilu ukazno razpoznavanje; tkim. vejični primer lahko ima pri tkim. oklepajnih ukazih še vgnezdeni vejični primer. Funkcije procedur so opisane s komentarji v zadevnih procedurah.

(fiksniz, niz, izhvrs, znakrazr, ukazrazr), iz večjega števila spremenljivk, iz procedur oziroma funkcij (ukazsezn, trennast, ukaztab, dialog, zapiranje, zacetek, vzemiznak, stikznak, stikniz, pomik, poravnava, justiranje, vrsstr, vrsst1, izpis, izpis1, zacvrsta, konvrsta, konvrs1, vstavitev, vstav1, vvrstic, vvrst1, razmnem, nasl1, nespr1, kom1, naslz1, zbrisi, razpoparam, razpoukaz, jestev in deset) in iz glavnega programa. Glavni program ima začetni segment, ki mu sledi glavna REPEAT zanka, tej pa končni segment.

Filozofija programa temelji na petih glavnih oziroma znakovnih stanjih, ki so elementi tipa znakrazr in predstavljajo alternativno nepresledek, presledek, znak konca vrstice in konca zbirke in vejico. S temi petimi razredi je mogoče obvladati besedilni vhodni niz (besedilno zbirko) z vgnezdenimi oblikovalnimi ukazi. Tu je vejica potencialno ukazno stanje, če je podniz s to vejico na koncu ukaz. Vejično stanje vodi tako v vsa možna ukazna stanja (podstanja). Oglejmo si sedaj to filozofijo na samem programu v listi 1.

Osnovni CASE stavek v okviru glavnega REPEAT stavka (v glavnem programu liste 1, na koncu) ima tako tri primere, in sicer

nepresl,
presl, konvrs, konzbir,
vejica,

pri čemer se primer (stanje) vejica nadaljuje z vsemi ukaznimi primeri (podstanji).

Primer konzbir je seveda ključen, in v tem primeru je zagotovljen takojšen izstop iz glavne REPEAT zanke in nato zapiranje vseh zbirk. Seveda mora biti ta izstop zagotovljen tudi v primeru vseh vejičnih podstanj, če se pojavi na vhodu znak konca zbirke.

Glavni vejični primer (glavna zanka, osnovni CASE stavek) se nadaljuje z razpoznavanjem ukaza in nato z dvema ločenima segmentoma, ki obsegata ukaze s parametri in ukaze brez parametrov. Prvi segment mora razpoznavati še parameter. Če parameter ni pravilen, imamo primer niukaz. Če ukaz ni pravilen (prva faza, procedure razpoukaz), izstopimo prav tako prek primera niukaz. Element niukaz tipa ukazrazr je tako predviden za označevanje neukaznega primera, ko v vejičnem stanju razpoznavamo, ali imamo ukaz ali pa le podniz vhodnega besedila.

Filozofija z uvedbo ukaznega primera niukaz je predvsem učinkovita in transparentna v katerem-koli delu oblikovalniškega programa, neglede na to ali je segment procedura ali podsegment glavnega programa. Podobno velja to tudi za znakovno stanje oziroma primer konzbir. To stanje povzroči izstop iz glavne zanke, stanje niukaz pa izstop iz glavnega vejičnega stanja pa tudi iz vgnezdenih vejičnih podstanj (primeri tkim. oklepajnih ukazov, ki imajo svoj začetni in končni označevalnik).

Program v listi 1 upošteva 18 ukazov, stanje 19 je predvideno za primer niukaz. Konstanta ukazobseg na začetku programa določa to število.

Oglejmo si še podatkovne tipe programa. Tip fiksniz je predviden za nizno predstavljanje v standardni Wirthovi izvedenki (PACKED ARRAY). Pri operacijah s tem nizi, so posamezni znaki niza določeni z indeksiranimi spremenljivkami. Tip niz ima tako dve komponenti: faktični niz (niz1) in njegovo trenutno dolžino (dol). Vse operacije nad nizi v besedilnem in kazalnem delu programa bodo upoštevale tako definirani nizni tip (dvokomponentni). Tip izhvrs predstavlja izhodno vrstico, ki bo prejkoslej izdana kot izhodna besedilna oziroma kazalna vrstica. Ta tip je dvokomponenten in je sestavljen iz našega tipa niz in iz aktualnega stolpca v tem nizu (aktstl). Slednja spremenljivka bo potrebna pri justiranju vrstice, ko bomo imeli desno poravnavo z vstavljanjem potrebnih presledkov med besedami v vrstici pred njeno izdajo.

Tip znakrazr razvršča skupaj s funkcijskim poljem razf (glej pri spremenljivkah) vhodne znake v pet znakovnih razredov, kot smo to že opisali. 19 ukaznih stanj (primerov) je poimenovanih z elementi tipa ukazrazr in njihova imena se ujemajo z ukaznimi pomeni. Uvedba tega tipa prispeva predvsem k nazornosti in preglednosti celotnega programa.

Program Lenca uporablja samo besedilne zbirke (tip text), čeprav bi bila včasih morda smotrnejša uporaba tipa FILE OF char. Tip text skrajšuje program, ker so za ta tip vgrajene še

nekatere lastnosti, ki program skrajšujejo. Taka lastnost je npr. avtomatična pretvorba celih števil (integer) v nize številk, tako da lahko vrednosti teh spremenljivk izpisujemo v besedilo brez posebne programske pretvorbe. Razen zbirk input, output imamo še tri zbirčne spremenljivke tipa text, in sicer vhzbir, izhzbir in kazzbir. Tem spremenljivkam bomo z assign stavki priredili iz konzole sprejeta imena vhime, izhime in kazime, ki so v tem primeru (kot že preje omenjeno) spremenljivke vgrajenega tipa string (Pascal/MT+). Ta imena in specifični zbirčni stavki se pojavljajo samo v dveh procedurah, in sicer v dialog in v zapiranje.

Opišimo na kratko še pomen posameznih procedur.

Procedura ukazsezn izpiše seznam vseh ukazov oblikovalnika. Ta procedura je tako tudi izvršitev ukaznega primera ukazi (na koncu glavnega programa liste 1). Ta seznam se vselej izpiše samo na konzolo in je informacija za uporabnika.

Procedura trennast izpiše na konzolo trenutno veljavne vrednosti formatnih parametrov za izhodno besedilo in kazalo in pomeni izvršitev ukaznega primera tparam (na koncu glavnega programa liste 1).

Procedura ukaztab definira ukazno mnemoniko za ukaze z in brez parametrov (tridimenzionalno polje uk) in določa še ukazno funkcijo (polje) ukazf, uporabi pa se v proceduri začetek.

Procedura dialog omogoči vpis imen želenih zbirk (vhod, izhod, kazalo) v CP/M formatu z uporabo assign stavkov, nadzoruje možnost odpiranja teh zbirk in omogoča nadaljevanje novega oblikovanja besedila po izvršenem oblikovanju.

Procedura zapiranje zapre vse z dialogom odprte zbirke in nadzira tudi možnosti zapiranja.

Procedura zacetek nastavi vse bistvene začetne vrednosti spremenljivk, tako besedilnih kot kazalnih.

Procedura vzemiznak jemlje znake iz vhodne zbirke in določi razred posameznega znaka z uporabo funkcije razfŠznakČ. V primeru znaka eoln se za tekstovno zbirko uporabi bralni stavek readln.

Procedura stikznak pritakne znak k dani besedi.

Procedura stikniz pritakne besedo k danem nizu besed.

Procedura pomik se uporablja v proceduri poravnava za ustrezno pomikanje besed (levo, centrirano, desno).

Procedura poravnava uredi (poravna) posamezne skupine besed s primeri levapor, center in desnapor z vsakokratno uporabo procedure pomik.

Procedura justiranje razporedi besede v dani izhodni vrstici tako, da pri desni poravnavi napolni reže med besedami z ustreznim številom presledkov.

Procedura vrsstr kontrolira prehod na naslednjo stran in izpiše gornji rob, številko strani in končno še presledek med številko strani in besedilom. Ker zaznava konec strani (stvr = dstr), mora biti uporabljena vselej, preden se oblikovana vrstica izpiše.

Procedura vrsstl ima podobno vlogo kot procedura vrsstr, le da kontrolira izpis vrstic v kazalo. Razen tega je njena naloga še v tem, da na začetku izpiše naslov 'Kazalo' in da oštevilčuje kazalne strani v obliki K-nn.

Procedura izpis izpiše najprej veljaven levi rob in nato vrstico z upoštevanjem njene dolžine. Kot že povedano, mora obvezno pred vsakim izpisom uporabiti proceduro vrsstr.

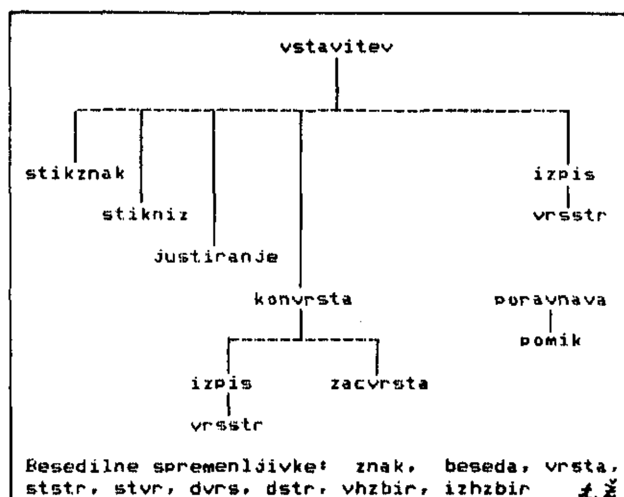
Procedura izpisl ima podobno vlogo pri izpisovanju vrstice v kazalo, ko mora obvezno uporabiti proceduro vrsstl.

Procedura zacvrsta inicializira vrsto tipa izhvrs v tkim. prazno vrsto (vrsta, kazvrsta).

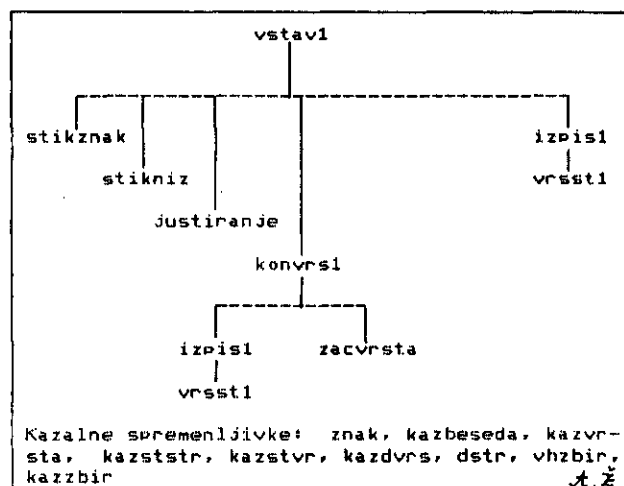
Procedura konvrsta se uporablja pri izpisovanju v besedilo in vsebuje proceduri izpis in zacvrsta.

Procedura konvrsl se uporablja pri izpisovanju v kazalo in vsebuje proceduri izpisl in zacvrsta.

Procedura vstavitev je ključna (glej sliko 1) in uporablja vrsto doslej opisanih procedur (glej sliko 1). Z njo se oblikovana beseda vstavlja v vrsto (če je to mogoče), vrsta pa se potem, ko je polna, izpiše.



Slika 1. Procedurne povezave (drevesi) pri besedilnem oblikovanju (izhodna zbirka)



Slika 2. Procedurna povezava (drevo) pri kazalnem oblikovanju (kazalna zbirka)

Procedura vstavl je podobna proceduri vstavitev pri oblikovanju kazala.

Procedura vvrstic je podprogram za ukaz *SK,nn. Izpisati mora ustrezno število praznih vrstic, seveda če je na tekoči strani dovolj prostora, sicer pa mora ta prostor zaseči na naslednji strani. Zaradi tega mora obvezno uporabiti proceduro vrsstr.

Peocedura vvrsstl je na določen način podobna proceduri vvrstic, le da pri izpisu besedilnega naslova zahteva na tekoči strani vsaj prostor desetih vrstic. Seveda mora obvezno uporabiti proceduro vrsstr (ker lahko pri tem preide na naslednjo stran).

Procedura razmnem je predvidena za razpoznavanje ukazne mnemonike. Če ima ukaz mnemoniko, določeno s parametroma, postane ukaz := naslov, sicer pa ukaz := niukaz. Ta procedura se uporablja v tkim. oklepajnih ukazih (kot so *NA,, *KO, in *NE,) pri razpoznavanju zaklepaja (zadnjega oklepaja).

Procedura nasl1 je podprogram ukaza *NA, in z njo se izpiše naslov v besedilo in ta naslov se v spremenjenem formatu vnese tudi v kazalo. Ta procedura vsebuje proceduro vvrstl in ker čaka na končni oklepaj, mora vsebovati tudi REPEAT zanko s CASE stavkom za znakovne razrede. Iz tega podprograma se izstopi le tedaj, ko se pojavi konzbir ali pa konec ukaza (zaklepaj). V okviru ukaznega stanja '*NA,' drugi ukazi nimajo izvajalne moči, zato smemo v naslovih izpisovati ukazne nize z izjemo niza *NE,.

Procedura nespr1 je podprogram ukaza *NE, in njena uporaba povzroči nespremenjeni izpis besedila do pojavitve končnega ukaznega niza *NE, (vključno z znaki za nove vrstice). Procedura mora upoštevati proceduro vrsstr in mora podobno kot prejšnja procedura vsebovati značilno REPEAT zanko. Izstop iz procedure je pogojen s pojavitvijo končnega ukaza *NE, ali z znakom konzbir.

Procedura koml izpiše komentar, tj. besedilo med dvema zaporendima ukazoma '*KO,' v nespremenjeni obliki in brez levega roba neposredno na konzolo; vsebuje glavno REPEAT zanko s pripadajočim CASE stavkom za razpoznavanje znakovnih razredov. Tj. procedura primera koment v glavni zanki programa.

Procedura naslzi je procedura ukaznega primera naslzn v glavni zanki programa. S tem ukazom se doseže nespremenjeni izpis znaka za tem ukazom in s tem npr. prekinitev ukaznega zaporedja, ki ga želimo izpisati.

Procedure in funkciji za razpoznavanje oblikovalnih ukazov (pri oklepajnih ukazih za razpoznavanje začetnih oklepajev) oblikujejo posebno podprogramsko skupino.

Procedura zbrisi zbriše zadnjih n znakov iz besede, in sicer tiste znake, ki oblikujejo ukaz brez parametra ali ukaz s parametrom. Tako je najmanjša izbrisna dolžina 4 (ukaz brez parametra, npr. '*XY,') in izbrisna dolžina med 2 in 5 (ukaz s parametrom, npr. parameter '$$$$,').

Procedura razpoparam mora razpoznati parametrični del ukaza, potem ko je bil mnemonični že razpoznan. Parameter je največ štirimestni ASCII številčni niz, ki se končuje z vejico. Procedura razpozna dolžino parametričnega niza, ga pretvori v celo število in v primeru regularnosti parametra zbriše parameter iz besede. Ta procedura uporablja tako proceduro zbrisi, boolovsko funkcijo jestev in funkcijo deset (opisani kasneje).

Procedura razpopukaz razpoznava mnemonični del ukaza in določi ukaz iz ukaznega razreda ukazrazr. Če je mnemonični del različen od tkim. niukaza, se ukazna mnemonika zbriše iz besede.

Funcija jestev ugotavlja, ali je dani znak številka.

Funkcija deset izračunava vrednost 10 na potenco i in se uporablja pri pretvorbi niza v celo število.

Glavni program oblikovalnika Lenca je med drugim sestavljen iz glavne REPEAT zanke s proceduro vzemiznak in glavnim CASE stavkom za značilne znakovne primere (5). Vejični primer je razdeljen v segment ukazov s parametrom in ukazov brez parametra. Ostalo je bilo že opisano v prejšnjih delih tega prispevka.

```
2A>lenca4

 Oblikovalnik Lenca, tip 1.4/1985

Ali zelis oblikovati besedilo (d/n)? d

Vstavi ime  vhodne zbirke: vh
Vstavi ime izhodne zbirke: izh
Vstavi ime kazala:         : kaz

POCAKAJ NA KONEC OBDELAVE ! ! !

Začetna nastavitev parametrov je *GR,4, *DV,47,
*DS,22, *LR,0,
Vstavi glavo Petit 12!

Trenutno veljavni parametri:

Gornji rob       = 4
Levi rob         = 0
Dolzina vrstice  = 47
Dolzina strani   = 22
Stevilka strani  = 963

Vrstica kazala   = 3
Stran kazala     = 2

    Ukazni seznam oblikovalnika:

    *LR,m,    Sprememba levega roba,
    *DV,m,    Sprememba dolz.vrstice,
    *GR,m,    Sprememba gornj. roba,
    *DS,m,    Sprememba dolz.strani,
    *NV,      Prehod na novo vrstico,
    *SK,m,    Prehod prek m vrstic,
    *NS,      Prehod na novo stran,
    *NA,      Naslovi s kazalom,
    *LP,      Leva poravnava vrstice,
    *CE,      Centriranje vrstice,
    *DP,      Desna poravn. vrstice,
    *KO,      Komentarska vrstica,
    *NZ,      Tiskanje nasled. znaka
              je nespremenjeno,
    *NE,      Tiskanje znakov med o-
              mej. *NE, je nesprem.,
    *US,      Ustavitev izpisa,
    *SS,m,    Stevilka strani (nast.)
    *TP,      Trenutni parametri,
    *UK,      Seznam ukazov.

 Oblikovalnik Lenca, tip 1.4/1985

Ali zelis oblikovati besedilo (d/n)? n
```

Lista 2. Ta lista prikazuje izpis na zaslonu pri uporabi oblikovalnika Lenca. Viden je vhodni dialog za določitev treh zbirk in izpisi, ki se pojavijo na zaslonu kot posledice vgnezdenih ukazov v vhodnem besedilu (komentar, trenutni parametri in ukazni seznam). Uporaba programa je iterativna (izstop na zahtevo).

*KO,
Začetna nastavitev parametrov je *GR,4, *DV,47,
*DS,22, *LR,0, *KO,
*GR,4, *DV,47, *DS,22, *LR,0, *NA,

PREIZKUS POMENA POSAMEZNIH UKAZOV
OBLIKOVALNIKA Lenca*NA,
Oblikovalnik Lenca je oblikovalnik splošne
vrste in preizkus njegovih ukazov bomo
opisali v kratki obliki. Imamo ukaze s
parametri in ukaze brez parametrov. Namen tega
besedila je le omejen preizkus vseh ukazov
oblikovalnika, ki so trenutno vgrajeni. *NV,
*NV, Oblikovalnik Lenca omogoča enostavno
dodajanje novih ukazov in spreminjanje
obstoječih.
*NA,
1. Začetna nastavitev *NA.
Najprej nastavimo z ukazi gornji rob (ukaz GR
s parametrom 4), dolžino vrstice (ukaz DV s
parametrom 47), dolžino strani (ukaz DS s
parametrom 22) in levi rob (ukaz LR s
parametrom 0). Imamo torej štiri ukaze (glej
zgoraj): *NV,*LR,22, *NE,*GR,4
*DV,47,
*DS,22,
*LR,0*NE,
*LR,0, *NA,
2. Komentarski ukaz (KO) *NA,
Komentarski ukaz je bil prvi, ki smo ga v
našem primeru uporabili (glej zgoraj). Njegova
oblika je bila *NV,*LR,22, *NE,*KO,
*NE,*LR,0,Komentarski ukaz povzroči
nespremenjeni izpis komentarja na zaslon, tako
da je komentar lahko tudi navodilo ali
pojasnilo za uporabnika. *NA,
3. Sprememba levega roba (LR) *NA, Z ukazom
levega roba nastavljamo rob od števila nič
ali več presledkov, Nastavitvena oblika tega
ukaza je na primer *NV,
*LR,22, *NE,*LR,22,
*NE,*LR,0, ko imamo 22 presledkov. To je ukaz
s parametrom. S tem ukazom dosegamo različne
besedilne umaknitve.*NV,*NV, Pri uporabi ukaza
LR moramo biti previdni, ker je od trenutne
vrednosti njegovega parametra odvisen tudi
izpis številke strani, ko se levi rob prišteva,
tako da je izpis te številke sredinski glede
na besedilo.
*NA,4. Nastavitev dolžine vrstice (DV) *NA,
S tem ukazom nastavimo želeno dolžino vrstice,
ki je trenutno 47. Z ukazom *NV,*LR,22,
*NE,*DV,30,
*NE, *LR,0,*DV,30, smo nastavili to
dolžino na trideset znakov v vrstici, kot kaže
ta izpis. *DV,47,Dolžino vrstice nato popravimo
na 47. Kot vidimo, se novi ukaz za spremembo
dolžine vrstice upošteva že v tekoči vrstici,
ki se tako podaljša iz 30 na 47 znakov.
*NA,5. Nastavitev gornjega roba (GR) in dolžine
strani (DS) *NA,
Trenutni ukaz gornjega roba (GR) dopušča
nastavitev parametra, ki ni manjši od 3 (to so
tri vrstice, od katerih je prva s številka
strani, naslednji dve pa sta prazni). Največja
smiselna nastavitev dolžine strani (DS) je
približno 55, tako da imamo ohranjen okvir
tiskalniškega izpisa strani. Ukaza sta tedaj v
našem primeru: *NV,*LR,22,*NE,*GR,3,
*DS,22,*NE,*LR,0,*NV,*NV,
Izpis številke strani je odvisen od trenutno
veljavnih vrednosti levega roba in dolžine
strani. V teh primerih se lahko pojavijo
pasti, ki jih nastavljajo besedilni
oblikovalniki (v nasprotju z zaslonskimi
oblikovalniki in urejevalniki kot je npr.
Wordstar).*NV,
*NA, 6. Prehod na novo vrstico (NV) in na novo
stran (NS) *NA,
Ukaza za vrstični in stranski prehod sta
razumljiva in neproblematična.
*NA, 7. Prehod prek več vrstic (SK) *NA,
Ta ukaz se uporablja za rezervacijo prostora,
ki je potreben npr. za vstavitev slike ali
nepredvidenega besedilnega segmenta. Če na
tekoči strani ni dovolj prostora, se ta prostor
zasede na naslednji strani. Prostor se izraža
s številom praznih vrstic, tako da imamo obliko
ukaza *NV,*NE, *SK,6,*NE,
*NV,in prazen prostor, ki obsega 6 praznih
vrstic. *SK,6,V ta prostor bomo npr. vstavili
narisano shemo, seveda pa lahko pod shemo
izpišemo njeno številko in opis.
*NA,8. Ukaz za oblikovanje naslova (NA)*NA,
Naslov se oblikuje v besedilu (tri prazne
vrstice, naslov, dve prazni vrstici) in v
kazalu, kjer se pripiše še tekoča številka
strani. Vrstica v kazalu je krajše poravnana,
tako da se ustrezno poravnano pripiše tekoča
številka stani. Kazalo je posebna zbirka, ki
se lahko na koncu obdelave pritakne k besedilni
zbirki na njenem začetku ali njenem koncu. V
stanju naslovnega ukaza (med dvema naslovnima
ukazoma) ni mogoča uporaba drugih ukazov in vsi
vstavljeni ukazi se izpisujejo kot besedilo.
*SK,1,Uporaba ukaza*NV,
*NE, *NA,*NE,*NV,
je razvidna iz vrste dosedanjih primerov.
*NA,9. Leva poravnava (LP), centriranje (CE) in
desna poravnava (DP) besedila v vrstici*NA,
V naslovu navedeni ukazi omogočajo oblikovanje
vrstice, ki je besedilo levo poravnano,
centrirano, ali desno poravnano, možna je pa
tudi poljubna kombinacija teh lastnosti.
Oglejmo si nekaj primerov!*SK,1,
Imejmo npr. na levi sodo številko strani, v
sredini naj se pojavi osrednji pojem te strani,
na desni pa zvezek časopisa. Imamo:*NV,*NV,
214*LP,Sintaksna analiza*CE,Info 9*DP,
*NV,Seveda lahko imamo samo*SK,2,
Sintaksna analiza*CE, *SK,2,ali*NV,*NV,
214*LP,Info 9*DP,*NV,
itd. Možne so torej vse kombinacije teh treh
ukaznih zvrsti.*SK,2, Omenjeni trije ukazi so
tipični vrstični ukazi in njihova oblika je
*SK,1, *NE,
*LP,
*CE,
*DP,*NE,
*NA,10. Komentar (KO) in ustavitev izpisa (US)
*NA, Besedilo med ukazoma tipa KO se izpisuje
samo na zaslon (in ne v izhodno besedilno
zbirko). Z ukazom US ustavimo izpisovanje v
izhodno zbirko in ga nadaljujemo tedaj, ko
prek konzole vtipkamo presledek. To nam daje
možnost, da npr. zamenjamo pisalno glavo na
tiskalniku in nadaljujemo izpisovanje z drugo
pisavo do naslednje ustavitve. Na zaslon lahko
pri tem izpišemo navodilo, katero glavo moramo
vstaviti. Učinka teh dveh ukazov torej nista
vidna v izhodni zbirki.
*KO,Vstavi glavo Petit 12!*KO,*US,*SK,2,
Po tej dvojni operaciji se izdajanje besedila
nadaljuje. Ukaza sta tako*SK,1,*NE,
*KO,
*US,*NE,
*NA,11. Naslednji znak (NZ) in nespremenjeni
izpis besedila (NE)*NA,
Z ukazom NZ lahko izpišemo naslednji znak
nespremenjeno. Tako lahko izpišemo npr. sam ta
ukaz, ko imamo *NZ*NZ,,. Besedilo med ukazoma
NE pa se izpiše nespremenjeno. Končni obliki
teh dveh ukazov sta tedaj*SK,2,*LR,22,
*NZ*NZ,,*NV,*NE*NZ,,*NV,*LR,0,
*NA,12. Nastavitev nove številke strani (SS)
*NA,
Novo številko strani nastavimo z ukazom tipa
SS, ko npr. nadaljujemo s pisanjem določenega
besedila od neke strani naprej. v tem primeru
imamo ukaz *SK,1,*NE,
*SS,963,*NE,*SK,1,
Praktično pa imamo novo številko strani!
*SS,963,
*NA,13. Izpis trenutnih parametrov (TP) in
razpoložljivih ukazov na zaslon*NA,

Zadnja dva ukaza sta instrukcijska in omogočata vpogled v trenutno nastavitev oblikovalnih parametrov in v seznam oblikovalnih ukazov. Obe sporočili se izpišeta na zaslon in se tako ne pojavita v izhodni zbirki. Ukaza sta *SK,1, *NE,
*TP,
*UK,
*NE,*TP,*UK,
*NA,14. Epilog*NA,
Uporaba predloženega oblikovalnika je vse prej kot priročna. Potrebno je nekaj vaje in uporaba določenih ukaznih kombinacij v posameznih primerih. Vsaka napaka pri izpisu ukaza ima posledico: to velja še posebej za oklepajne ukaze, kot so NE, KO in NA.

Lista 3. Lista na prejšnji in na tej strani prikazuje vhodno besedilo z vgnezdenimi ukazi, ki bo oblikovano z uporabo oblikovalnika Lenca (glej rezultate oblikovanja v listi 4 in listi 5). To vhodno besedilo je bilo shranjeno v vhodni zbirki z imenom vh in napisano s standardnim besedilnim urejevalnikom. Učinki posameznih ukazov so vidni v listi 5 in tudi v kazalni listi 4 (zbirka kaz). Primerjava liste 5 s to listo je bistvena in kaže natanko funkcijo posameznih ukazov.

K-1

K a z a l o

PREIZKUS POMENA POSAMEZNIH UKAZOV OBLIKOVALNIKA Lenca	1
1. Začetna nastavitev	2
2. Komentarski ukaz (KO)	3
3. Sprememba levega roba (LR)	4
4. Nastavitev dolžine vrstice (DV)	5
5. Nastavitev gornjega roba (GR) in dolžine strani (DS)	6
6. Prehod na novo vrstico (NV) in na novo stran (NS)	7
7. Prehod prek več vrstic (SK)	7
8. Ukaz za oblikovanje naslova (NA)	9
9. Leva poravnava (LP), centriranje (CE) in desna poravnava (DP) besedila v vrstici	10
10. Komentar (KO) in ustavitev izpisa (US)	12
11. Naslednji znak (NZ) in nespremenjeni izpis besedila (NE)	13

K-2

12. Nastavitev nove številke strani (SS)	14
13. Izpis trenutnih parametrov (TP) in razpoložljivih ukazov na zaslon	963
14. Epilog	964

Lista 4. Ta lista predstavlja kazalo, ki se je oblikovalo iz vhodnega besedila z uporabo naslovnih ukazov. V kazalo se izpisujejo tudi strani, na katerih se naslovi v besedilu pojavljajo.

1

PREIZKUS POMENA POSAMEZNIH UKAZOV OBLIKOVALNIKA Lenca

Oblikovalnik Lenca je oblikovalnik splošne vrste in preizkus njegovih ukazov bomo opisali v kratki obliki. Imamo ukaze s parametri in ukaze brez parametrov. Namen tega besedila je le omejen preizkus vseh ukazov oblikovalnika, ki so trenutno vgrajeni.

Oblikovalnik Lenca omogoča enostavno dodajanje novih ukazov in spreminjanje obstoječih.

2

1. Začetna nastavitev

Najprej nastavimo z ukazi gornji rob (ukaz GR s parametrom 4), dolžino vrstice (ukaz DV s parametrom 47), dolžino strani (ukaz DS s parametrom 22) in levi rob (ukaz LR s parametrom 0). Imamo torej štiri ukaze (glej zgoraj):

*GR,4
*DV,47,
*DS,22,
*LR,0

3

2. Komentarski ukaz (KO)

Komentarski ukaz je bil prvi, ki smo ga v našem primeru uporabili (glej zgoraj). Njegova oblika je bila

*KO,

Komentarski ukaz povzroči nespremenjeni izpis komentarja na zaslon, tako da je komentar lahko tudi navodilo ali pojasnilo za uporabnika.

4

3. Sprememba levega roba (LR)

Z ukazom levega roba nastavljamo rob od števila nič ali več presledkov. Nastavitvena oblika tega ukaza je na primer

*LR,22,

ko imamo 22 presledkov. To je ukaz s parametrom. S tem ukazom dosegamo različne besedilne umaknitve.

Pri uporabi ukaza LR moramo biti previdni, ker je od trenutne vrednosti njegovega parametra odvisen tudi izpis številke strani, ko se levi rob prišteva, tako da je izpis te številke sredinski glede na besedilo.

5

4. Nastavitev dolžine vrstice (DV)

S tem ukazom nastavimo želeno dolžino vrstice, ki je trenutno 47. Z ukazom

*DV,30,

smo nastavili to dolžino na
trideset znakov v vrstici, kot
kaže ta izpis. Dolžino vrstice nato popravimo na 47. Kot vidimo, se novi ukaz za spremembo dolžine vrstice upošteva že v tekoči vrstici, ki se tako podaljša iz 30 na 47 znakov.

6

5. Nastavitev gornjega roba (GR) in dolžine strani (DS)

Trenutni ukaz gornjega roba (GR) dopušča nastavitev parametra, ki ni manjši od 3 (to so tri vrstice, od katerih je prva s številka strani, naslednji dve pa sta prazni). Največja smiselna nastavitev dolžine strani (DS) je približno 55, tako da imamo ohranjen okvir tiskalniškega izpisa strani. Ukaza sta tedaj v našem primeru:

*GR,3,
*DS,22,

Izpis številke strani je odvisen od trenutno veljavnih vrednosti levega roba in dolžine strani. V teh primerih se lahko pojavijo pasti, ki jih nastavljajo besedilni oblikovalniki (v nasprotju z zaslonskimi oblikovalniki in

7

urejevalniki kot je npr. Wordstar).

6. Prehod na novo vrstico (NV) in na novo stran (NS)

Ukaza za vrstični in stranski prehod sta razumljiva in neproblematična.

7. Prehod prek več vrstic (SK)

Ta ukaz se uporablja za rezervacijo prostora, ki je potreben npr. za vstavitev slike ali nepredvidenega besedilnega segmenta. Če na tekoči strani ni dovolj prostora, se ta prostor zasede na naslednji strani. Prostor se izraža s

8

številom praznih vrstic, tako da imamo obliko ukaza

*SK,6,

in prazen prostor, ki obsega 6 praznih vrstic.

V ta prostor bomo npr. vstavili narisano shemo, seveda pa lahko pod shemo izpišemo njeno številko in opis.

9

8. Ukaz za oblikovanje naslova (NA)

Naslov se oblikuje v besedilu (tri prazne vrstice, naslov, dve prazni vrstici) in v kazalu, kjer se pripiše še tekoča številka strani. Vrstica v kazalu je krajše poravnana, tako da se ustrezno poravnano pripiše tekoča številka stani. Kazalo je posebna zbirka, ki se lahko na koncu obdelave pritakne k besedilni zbirki na njenem začetku ali njenem koncu. V stanju naslovnega ukaza (med dvema naslovnima ukazoma) ni mogoča uporaba drugih ukazov in vsi vstavljeni ukazi se izpisujejo kot besedilo.

Uporaba ukaza

*NA,

je razvidna iz vrste dosedanjih primerov.

10

9. Leva poravnava (LP), centriranje (CE) in desna poravnava (DP) besedila v vrstici

V naslovu navedeni ukazi omogočajo oblikovanje vrstice, ki je besedilo levo poravnano, centrirano, ali desno poravnano, možna je pa tudi poljubna kombinacija teh lastnosti. Oglejmo si nekaj primerov!

Imejmo npr. na levi sodo številko strani, v sredini naj se pojavi osrednji pojem te strani, na desni pa zvezek časopisa. Imamo:

214 Sintaksna analiza Info 9

Seveda lahko imamo samo

Sintaksna analiza

Lista 5. Lista s prejšnje polovice strani, na tej strani in na naslednji strani prikazuje tkim. izhodno zbirko, ki je rezultat obdelave vhodnega besedila z vgnezdenimi ukazi z besedilnim oblikovalnikom Lenca. Posledica tega oblikovanja je 16 strani izhodnega besedila. Strani so oštevilčene v sredini zgoraj. Parametri oblikovanja so bili tako izbrani, da je primer nazoren in razviden (kratke vrstice, kratke strani, veliko število naslovov, uporaba vseh ukazov).

11

ali

214 Info 9

itd. Možne so torej vse kombinacije teh treh ukaznih zvrsti.

Omenjeni trije ukazi so tipični vrstični ukazi in njihova oblika je

*LP,
*CE,
*DP,

12

10. Komentar (KO) in ustavitev izpisa (US)

Besedilo med ukazoma tipa KO se izpisuje samo na zaslon (in ne v izhodno besedilno zbirko). Z ukazom US ustavimo izpisovanje v izhodno zbirko in ga nadaljujemo tedaj, ko prek konzole vtipkamo presledek. To nam daje možnost, da npr. zamenjamo pisalno glavo na tiskalniku in nadaljujemo izpisovanje z drugo pisavo do naslednje ustavitve. Na zaslon lahko pri tem izpišemo navodilo, katero glavo moramo vstaviti. Učinka teh dveh ukazov torej nista vidna v izhodni zbirki.

Po tej dvojni operaciji se izdajanje besedila nadaljuje. Ukaza sta tako

13

*KO,
*US,

11. Naslednji znak (NZ) in nespremenjeni izpis besedila (NE).

Z ukazom NZ lahko izpišemo naslednji znak nespremenjeno. Tako lahko izpišemo npr. sam ta ukaz, ko imamo *NZ*,. Besedilo med ukazoma NE pa se izpiše nespremenjeno. Končni obliki teh dveh ukazov sta tedaj

*NZ,
*NE,

14

12. Nastavitev nove številke strani (SS)

Novo številko strani nastavimo z ukazom tipa SS, ko npr. nadaljujemo s pisanjem določenega besedila od neke strani naprej. V tem primeru imamo ukaz

*SS,963,

Praktično pa imamo novo številko strani!

963

13. Izpis trenutnih parametrov (TP) in razpoložljivih ukazov na zaslon

Zadnja dva ukaza sta instrukcijska in omogočata vpogled v trenutno nastavitev oblikovalnih parametrov in v seznam oblikovalnih ukazov. Obe sporočili se izpišeta na zaslon in se tako ne pojavita v izhodni zbirki. Ukaza sta

*TP,
*UK,

964

14. Epilog

Uporaba predloženega oblikovalnika je vse prej kot priročna. Potrebno je nekaj vaje in uporaba določenih ukaznih kombinacij v posameznih primerih. Vsaka napaka pri izpisu ukaza ima posledico: to velja še posebej za oklepajne ukaze, kot so NE, KO in NA.

Lista 5 (nadaljevanje s prejšnjih dveh strani). Iz liste 3 in te liste je razvidno kako je mogoče na več načinov izpisovati tudi same ukazne nize (da se ti ne obravnavajo kot ukazi v vhodnem besedilu). V ta namen lahko uporabimo več ukaznih kombinacij. Oklepajni ukaz *NE, je zato že primeren, ker v stanju trajanja tega ukaza do njegove ponovne pojavitve (ukazni zaklepaj) drugi ukazi nimajo svoje moči. Pripravna je tudi uporaba ukaza *NZ, , ki v bistvu prekine dani ukazni niz, ki ga želimo izpisati. Takšno prekinitev bi lahko dosegli tudi z ukazom *KO,. Izbrana ukazna zaloga omogoča rešitev večine primerov, ki si jih lahko izmislimo. Seveda pa je mogoče oblikovalnik še dopolniti in ga tudi v določenih primerih izboljšati. Opaženih je bilo le majhno število tkim. stranskih učinkov, ki pa so vsi rešljivi z ustrezno drugačno uporabo ukaznih zaporedij.

4. Izvajanje oblikovalnega programa

Oblikovalnik Lenca lahko izvajamo na več načinov. V poskusni fazi lahko vstavimo prek procedure dialog namesto zbirčnih imen ime CON: in dobimo tako celotno komunikacijo (vhodno, izhodno, kazalno) prek konzole (tipkovnice in zaslona). Oglejmo si primer, ko imamo vhodno besedilo v zbirki vh, izhodno besedilo želimo imeti v zbirki izh in kazalno besedilo v zbirki kaz. Izvajanje oblikovalnika lenca4 je prikazano s konzolnim izpisom v listi 2.

Skladno z vhodnim besedilom v zbirki vh se na zaslon izpišejo sporočila in rezultati ukazov. Najprej imamo na zaslonu začetni dialog, ko vstavimo imena treh želenih zbirk (vh, izh, kaz). Po zadnji vstavitvi se oblikovalnik izvršuje naprej in moramo počakati na konec obdelave. Med oblikovanjem vhodnega besedila se na zaslon izpisujejo različna sporočila oziroma rezultati posameznih ukazov v vhodnem besedilu. Najprej se izpiše komentar o začetnih nastavitvah (glej vhodno besedilo v listi 3). Nato se pojavi drugi komentar, ki pravi, naj se vstavi tiskalna glava Petit 12. V tej točki se oblikovanje ustavi (zaradi uporabe ukaza *US,). Ko npr. glavo zamenjamo (na tiskalniku), vtipkamo presledek in oblikovanje preostalega vhodnega besedila se nadaljuje. Pred koncem oblikovanja se uporabita še dva ukaza, ki posredujeta izpis na zaslon, in sicer ukaz *TP, in ukaz *UK, (glej listo 3 proti koncu). Po končanem oblikovanju lahko začnemo oblikovati nov vhodni tekst in postopek se ponovi.

Lista 3 prikazuje vhodno besedilo, ki je shranjeno v zbirki z imenom vh. To besedilo začenja s komentarjem, ki se izpiše na zaslon. Komentar ima svoj končni oklepaj. Nato se z oblikovalnimi ukazi nastavijo nekateri oblikovalni parametri. Tem sledi ukaz naslova (*NA,). Posledice teh oblikovalnih ukazov so vidne v listi 5. Naslovi se istočasno v spremenjenem formatu izpisujejo tudi v zbirko kaz, kjer se oblikuje kazalo vhodnega besedila (glej listo 4). Proti koncu vhodnega besedila v listi 3 imamo še ukaz za spremembo oštevilčenja strani, in sicer *SS,963, s katerim se spremeni oštevilčevanje (to je vidno v listi 5 na zadnjih dveh izpisanih straneh in v kazalu v listi 4).

Vhodno besedilo je seveda primerno nepregledno in potrebno je nekaj vaje in tudi poskušanja, da se dobijo ustrezni oblikovalni učinki. Vsekakor je priporočljivo, da oblikovalnik Lenca izvajamo nad dano vhodno zbirko, pogledamo rezultate v izhodni zbirki in pred izpisom s tiskalnikom popravimo napake uporabnika oblikovalnika.

Lista 4 prikazuje kazalo, ki se je oblikovalo v zbirki kaz iz vhodnega besedila v zbirki vh (oziroma v listi 3). Dolžina vrstice za besedilo v kazalu je nekoliko krajša (velja dvrs - 7), da tako dobimo ustrezen prostor za izpis tekoče številke strani na koncu prve vrstice naslova. Naslov v kazalu oziroma v besedilu se seveda lahko razprostira prek več vrstic, vendar se pripadajoča številka strani izpiše samo v prvi vrstici naslova v kazalu. Strani kazala se za razliko od strani besedila oštevilčujejo v obliki K-nn, na začetku kazala pa se izpiše še naslov K a z a l o . Zadnja dva naslova v kazalu sta ustrezno oštevilčena s spremembo tekoče strani (namesto s 15 in 16 z 963 in 964).

V listi 5 imamo končno še izhodno oblikovano besedilo. Strani tega besedila so oštevičene, vijugaste črte med stranmi pa označujejo, da nismo upoštevali praznega prostora tekoče strani. Naslovi so vselej ustrezno izpisani: preizkuša se preostali prostor na tekoči strani in če ni prostih vsaj 10 vrstic, se naslov izpiše na naslednjo stran. To izhodno besedilo hkrati pojasnjuje tudi pomen posameznih ukazov.

V primeru nepredvidenih učinkov posameznih ukazov imamo seveda možnost, da spreminjamo ukazna zaporedja v vhodnem besedilu in tako rešimo praktično vsakršen neustrezen primer.

5. Sklep

Oblikovalnik Lenca že ima praktičen pomen. Seveda ga je možno še modificirati in prirejati novim zahtevam. Ker je oblikovalnik napisan modularno, lahko imamo hitro več njegovih izvedenk (za različne ukuse). Seveda pa je mogoča tudi njegova razširitev na dodatne ukaze.

Povedali smo že, da je mogoče procedure v samem progamu parametrizirati; tako bi se programsko besedilo lahko precej skrajšalo na račun preglednosti (tudi modularnosti) samega programa. K ukazni zbirki bi bilo smiselno dodati ukaz za navajanje in razlago opomb v vhodnem besedilu. Tu obstajata dve možnosti: oštevilčene opombe bi lahko izpisovali v posebni zbirki (podobno kot kazalo), lahko pa bi jih izpisovali tudi na tekočih straneh (z ustrezno takojšnjo rezervacijo na tekoči in na naslednji strani).

Oblikovalnik, ki smo ga prikazali, je namenjen predvsem programiranju različnih niznih struktur (tipov z nizi kot bistvenimi elementi) in operacijami nad njimi; kaže pa tudi osnovne možnosti oblikovanja besedilnih zbirk (zbirk tipa text). S spreminjanjem ukaznega formata oblikovalnih ukazov bi se v strukturi programa spremenile procedure za razpoznavanje ukazov. Vendar je predloženi ukazni format (takšen, kot je v tem prispevku) poučen in splošen glede na uporabo.

Slovstvo

((1)) N. Wirth: Algorithms + Data Structures = Programs. Prentice-Hall, Englewood Cliffs, NJ, 1976.

((2)) D.V. Moffat: Common Algorithms in Pascal. Prentice-Hall, Englewood Cliffs, NJ, 1984.

((3)) Pascal / MT+ User's Guide, Release 5, 1981.

POLEMIKA

Anton P. Železnikar:

UMETNA INTELIGENCA: polemični zapis

1. Uvod

Umetna inteligenca postaja predmet strokovnih in nestrokovnih razprav širom po svetu. Pojavljajo se natolcevanja, podtikanja, filozofizmi, podcenjevanja in pretiravanja o zmogljivostih današnjih in prihodnjih umetnointeligenčnih izdelkih, o njeni metodologiji. Umetna inteligenca (kratko UI) postaja tako tudi vsebolj ideološki spopad starih in novih razvojnih tokov, ko se njeni očetje opravičujejo, njeni rojeni otroci pa ji strežejo in jo ohranjajo, poskušajo pa jo tudi razvijati prek njenih realnih meja. Vsem pa je tako ali drugače jasno, da je UI v povojih, da se iz nje nekaj razvija, čeprav danes še ni mogoče povedati, kakšna je njena genetična osnova in v kaj in kako daleč se lahko razvije.

UI potrebuje zmogljivejše stroje (obsežnejše, hitrejše), potrebuje bolj zapletene programske segmente, potrebuje pa tudi še neodkrito metodologijo. Ti dejavniki omejujejo njeno današnjo komercialno uporabo, tj. raznovrstnost, masovnost in seveda predvsem inteligenčnost njenih izdelkov. Obstaja vrsta vprašanj, ki jih lahko postavimo in iščemo odgovore. Pa poskusimo!

2. Vprašanja o umetni inteligenci

1. Ali je dilema UI umetna?
2. Ali so današnji umetni inteligenčniki umetniki ali stvarniki?
3. Ali je UI komercialno uspešna?
4. Koliko je pomembnih, praktičnih rezultatov UI, kakšna je njihova dejanska vrednost?
5. Ali je razpravljanje o UI podobno zaklinjanju, kovanju zarotitvenih obrazcev in ali so umetni inteligenčniki res podobni kozelnikom, šamanom, žrecem, vračem in čarovnikom?
6. Ali obstaja ideologija UI oziroma njen metodološki Mein Kampf? Ali obstaja nadomestek za njene zgubljene začetne iluzije?
7. Ali velja res enačba UI = 5. generacija računalnikov?
8. Ali obstaja določilo o tem, kaj je UI in v kakšnem odnosu je z inteligenco?
9. Ali je vsak računalniški program tudi umetnointeligenčen?
10. Kaj vse se uvršča v UI, kje so njene meje, ali jih sploh ima?
11. Kakšne so dejanske perspektive UI?
12. Ali razmišljajo umetni inteligenčniki umetno (enoumno, robotoumno)?
13. Kaj lahko povzroči podcenjevanje UI?

3. Kategorizacija inteligence

V kakšnem odnosu je UI z drugimi inteligencami? Ker ne moremo odgovoriti na vprašanje, kaj je inteligenca, kako je strukturirana in kako se razvija (npr. pri otrocih), pa lahko postavimo neko samosvojo klasifikacijsko zgradbo, ki seveda ni kaj prida spoznavna:

INTELIGENCA:

- naravna (čustvena, racionalna; desno- in levosferna glede na možgane):
  - človekova, - živalska, (- rastlinska);
  - inteligenca kot stabilna razvojna strategija živega (od prajuhe naprej);
- okoliška (prihajajoča kot stabilni/obstojni vpliv iz okolja):
  - populacijska, - kulturna, - skupinska;
- umetna (v določenem pomenu nenaravna):
  - ideološka, - družbena, - bolezenska,
  - izrojena (v propad, v nazadovanje);
  - tehnološka:
    - sistemska, - strojna, - programska;
    - interaktivna med strojem in človekom.

Ta groba kategorizacije inteligence seveda ne pove ničesar o njeni zgradbi in njenem razvoju. Pri UI se največkrat razpravlja o tem, da je logično utemeljena (npr. s predikativno logiko prve stopnje in postopoma tudi z višjimi stopnjami), da lahko deluje nad dano bazo podatkov o znanju s pomočjo deduktivnih pravil (logično, statistično in še kako drugače implikativnih), da s temi pripomočki rojeva nove informacije in sklepe, preverja svoje trditve in temu podobno. UI se tako še ne ukvarja s široko problematiko psihološke inteligence, ki temelji na splošnih in bogatejših spoznavnih modelih, z meritvami inteligence (to se pravi nje same) z uporabo dovolj raznovrstnih inteligenčnih testov itd.

A. Binet je postavil svoja pojmovanja inteligence in njene metrike že na začetku tega stoletja. Po njemu je inteligenca prilagojevalna moč, s katero se rešujejo vsakodnevni življenski problemi; ta moč je splošna, čeprav je sestavljena iz raznovrstnih sestavin in se pri človeku linearno povečuje z njegovo rastjo. V okviru psihometričnih metod za preverjanje sposobnosti otrok je nastala tudi Binet-Simonova skala za ugotavljanje splošne inteligence (splošne duševne sposobnosti).

Reševanje problemov je le ena izmed kategorij raziskovanja mišljenja; ta metoda se je razvila iz zoopsihologije (Hobhouse, Thorndike). Pri tem je bistveno spoznanje, kako raziskovani subjekti rešujejo naloge: katere so faze reševalnega postopka, kakšna je reševalna motivacija, kako vplivajo pridobljene izkušnje, usmerjenost mišljenja (induktivnost, deduktivnost) itd. Nekaterim oblikam reševanja problemov so podobne tudi tehnike raziskovamja pojmovnega oblikovanja (nastajanja pojmov). Tu se pojavlja problematika oblikovanja znanstvenih in umetnih

(eksperimentalnih) pojmov. Znanstveni pojmi so sestavljeni pojmovni sistemi, ki se medsebojno opredeljujejo in so hierarhično povezani, tako da so mogoče raznovrstne logične operacije.

Inteligenco je mogoče meriti na več načinov. Turingova opredelitev inteligenčnega stroja je v tej zvezi ne samo splošna, temveč tudi izrazito nezadostna (naivna). Raziskovali naj bi se dovolj strogo mehanizmi oziroma natančneje procesi mišljenja. Take meritve inteligence bi npr. lahko uporabljale raziskovanje operacij, kot so klasifikacija, zapovrstnost, številska korespondenca, oblikovanje pojmov, razumevanje razrednih in relacijskih matrik, geometrične operacije, sorazmernost, kombinatorika, permutacija, verjetnost itd.

Dodatna zahteva meritev sta področji konvergentnega in divergentnega mišljenja (npr. Gilfordova razdelitev). Konvergentno mišljenje je v vsakem problemskem primeru usmerjeno k iskanju ene same, logično najbolj utemeljene rešitve. Divergentno mišljenje se usmerja k iskanju čim večjega števila, čim bolj raznovrstnih rešitev. V Gilfordovi terminologiji se samo take oblike duševnih dejavnosti uvrščajo v ustvarjalnost. Ustvarjalnost pa ne rešuje samo problemov, jih predvsem odkriva, sestavlja, oblikuje oziroma ustvarja. Ustvarjalnost je tako neke vrste problemskoreševalna metadejavnost, ni pa samo to.

UI lahko išče podobne modele predvsem izkustveno in za zelo posebne primere. Tako dopušča npr. nezanesljivost podatkov, učenje, graditev zanesljivejših podatkov in informacij, popravljanje in potrjevanje dobljenega itd.

Za kategorizacijo inteligence je npr. značilen tkim. Gilfordov model strukture sposobnosti. Sposobnost je v bistvu prepletenost inteligenc, ki je pri njemu tridimenzionalna. Dimenzijske osi sposobnosti so operacije, produkti in vsebine. Če so operacije os x v tridimenzionalnem sistemu, potem imamo na tej osi spoznavanje (kognicijo), spomin, divergentno mišljenje, konvergentno mišljenje in ocenjevanje (evaluacijo). Na vsebinski osi (os y) imamo figuralne (oblikovne), simboločne, semantične in vedenjske (obnaševalne) vsebine. Končno imamo na produktni osi z še enote (elemente), razrede, relacije, sisteme, transformacije in implikacije. UI je tako tudi s tehniškega vidika najbližja produktni osi modela, čeprav prodira počasi z razvojem tehnološke zavesti tudi po drugih oseh (npr. spomin, konvergentnost, ocenjevanje, simboločnost, semantičnost). Gilfordov model nudi tako 120 (5 krat 4 krat 6) posebnih in neodvisnih inteligenc pri človeku.

Eden od očitkov Gilfordovemu modelu je, da so inteligence konstruirane umetno in da je odkrivanje naravnih inteligenc še vedno bistveno vprašanje. Zato so se pojavili še drugi inteligenčni modeli. Eysenck je glede na merjenje inteligence predložil model s tremi osmi, in sicer takole:

- preizkusni material (os x):
  - verbalni (besedni, ne pismen),
  - numerični (številski),
  - spacialni (prostorski);
- kakovost (os y):
  - hitrost,
  - moč;
- duševni procesi (os z):
  - percepcija (dojemanje),
  - pomnjenje,
  - sklepanje.

Določeno soglasje na področju opredeljevanja inteligence je bilo doseženo na področju tkim. skupinskih faktorjev. Primarne inteligence (Cattell) naj bi bile tele:

- verbalna inteligenca (testiranje razumevanja slovarja, razumevanja prečitanega, gramatika in sintaksa, iskanje ekvivalentnih izrekov itd.);
- numerična inteligenca (osnovna sposobnost izvajanja operacij matematičnega mišljenja);
- prostorska inteligenca (testiranje s kockami, zastavami itd.);
- percepcijska hitrost (identificiranje slik, iskanje podobnosti v slikovnem materialu in v konfiguracijah, branje v ogledalu, razpoznavanje številčnika itd.);
- hitrost zaokrožanja celote (vidno spoznanje, zaznavanje geštalta, hitrost prilagajanja);
- induktivno sklepanje (splošno sklepanje, odkrivanje pravil, zakonitosti in principov, skrivnostni zapisi, šifriranje);
- deduktivno sklepanje (logično vrednotenje, gibanje od splošnega k posebnemu, silogizmi, oblikovanje predpostavk);
- neposredno pomnjenje (asociativno pomnjenje, besedne dvojice, slike in dvojice besed in števil itd.);
- mehanično znanje in veščine (poznavanje orodij in naprav, razumevanje načina delovanja strojev);
- besedna lahkotnost (verbalna spretnost, fluentnost, besede, ki začenjajo ali končujejo na določen način, anagrami);
- idejna lahkotnost (tematska fluentnost, uganke, oblikovanje naslovov, raznovrstnost v uporabi predmetov);
- preoblikovanje celote (prestrukturiranje, fleksibilnost celot, skrite oblike, skriti znaki);
- splošna motorna koordinacija (psihomotorna koordinacija, telesno prilagajanje, koordinacija udov, okončin itd.);
- ročna spretnost (manipuliranje, usmerjanje gibov, kirurška, finomehanišna spretnost);
- glasbena občutljivost za višino in barvo (preizkušanje muzikalne inteligence);
- spretnost grafičnega predstavljanja (risanje figur);
- prožnost (odpornost) proti okorelosti (izvirnost, originalnost, izvirni izreki, nenavadne uporabe, predvidevanje, napovedovanje, oddaljene posledice predpostavk).

Na področju tkim. splošne inteligence je Cattell v okviru ožjih inteligenčnih faktorjev (glej zgoraj) predložil še dva splošna faktorja, ki ju je poimenoval fluidna in kristalizirana inteligenca. Med posameznimi komponentami (faktorskimi inteligencami) človekove sposobnosti naj bi obstajala tudi vzročna in ne samo strukturna povezanost. Tako je npr. splošna sposobnost zaznavanja odnosov povezana z razvitostjo asociativnega nevronskega sistema v možganih. Ta splošna lastnost se lahko uporabi za selektivno zbiranje in urejanje informacij v človekovem spominu in tako ni vezana na neko posebno spretnost (inteligenco). Tj. tkim. fluidna inteligenca. Pri nabiranju izkušenj deluje fluidna inteligenca vzajemno z motivacijo, pomnjenjem in z okoliškimi vplivi in rojeva nove in bolj zapletene intelektualne dosežke oziroma inteligence. Tako pridobljene in zapletene inteligence, ki delujejo v okviru visoko organiziranih sposobnosti razumevanja in razsojanja v posebnih percepcijskih in motornih možganskih področjih, oblikujejo tkim. kristalizirano inteligenco; njen pojav je tedaj povezan z vrsto posebnih področij. Fluidno in kristalizirano inteligenco je mogoče ugotavljati (meriti) s posebnimi psihološkimi testi.

Iz napisanega lahko ugotavljamo, da je mogoče UI še marsikaj dodati, da bi postala bližja nečemu, kar bi lahko imenovali inteligenca. Seveda pa je mogoče v okviru danih definicij govoriti tudi danes o strojni, programski in

sistemski inteligenci tehnoloških sistemov, ki jih je zgradil človek. UI je pri tem lahko nekaj, kar v bistvu ni več izvirna inteligenca.

4. Odgovori na vprašanja o umetni inteligenci

Poskusimo odgovoriti na vprašanja, ki so bila postavljena v podpoglavju 2.

1. Umetna inteligenca je novo, tudi spekulativno področje dela in nedela (filozofiranja). UI potrebuje predvsem praktične rezultate, da bi prenehala biti dilematična in umetna (nenaravna).

2. Umetni inteligenčniki (krajše umiči) so prav gotovo tudi umetniki. Umetnost in njene metode vse bolj vdirajo na tradicionalna abstraktna in racionalna področja: torej so lahko tudi stvarniki. Vdor umetnostne metodologije lahko zrahlja okostenele racionalne metode in vpliva na obogatitev reševanja problematike, katere področje je tudi inteligenca.

3. Komercialna uspešnost UI je zaenkrat nepomembna (neznatna) in je lahko le predmet daljnje prihodnosti.

4. Zares uporabnih praktičnih rezultatov UI je preštevnih na prste ene ali največ dveh rok: to so najbolj opevani izvedeniški (ekspertni) sistemi, pa še ti niso vsi pomembni.

5. Kot že povedano, je področje UI tudi močno hipotetično, spekulativno, spiritualistično, skrivnostno in obstajajo umetnointeligenčni zarotitveni obrazci (prepričanja o obstoju neobstoječih in nedokazanih kakovosti UI sistemov) in zaklinjanja, ki pa jih lahko jemljemo bolj za svobodno filozofiranje (fantaziranje, nakladanje) kot za dejansko smotrnost ali škodljivost.

6. UI je dobila nov, spodbuden impulz z vpeljavo japonskega projekta pete generacije računalnikov. Japonski izziv so občutili Japonci sami kot nujnost, skozi katero širijo vpliv japonskega mišljenja in vpliva na zapadni razviti svet. Tehnološka ideologija pete generacije je razvojno motivirajoča za širok krog elektronske industrije in njenih spremljevalcev, podobno kot je npr. v zaostrenih pogojih lahko motivirajoča vojna industrija. Tako postaja peta generacija idol, težko uresničljiv in dovolj oddaljen cilj, katerega prihod (odrešenje) se lahko po potrebi odlaga. Idol pa je vselej nadomestek za nekaj drugega.

7. Japonci sami odločno zanikajo veljavnost enačbe

   UI = peta generacija računalnikov

   Tu gre za dejansko zlonamernost pa tudi za realnost zapadnih tehnoloških opazovalcev. Peta generacija je predvsem tehnološka kategorija (arhitekturna, ne-von neumannovska, paralelna, potencirano kompleksna, s strojnim jezikom predikativne logike, z vgrajenimi funkcijami sklepanja). UI so v bistvu tudi in predvsem aplikacije na novi računalniški tehnologiji oziroma s pomočjo nove tehnologije.

8. Inteligenca je vobče informacija, ki ustvarja (rojeva, generira) novo informacijo. Informacija je proces. Tudi UI je informacija, ki pa ima per definitionem manjšo zmogljivost od naravne inteligence. Kakšen specifični, fiziološki, energijski, biološki, miselni, organski, kemični, psihološki, informacijski, individualni proces je inteligenca ali vobče informacija, pa še ni natančneje znano.

9. V programu je lastnost, ki je pred njegovim nastankom lahko bila inteligenca. Vse, kar je napisano ali izpovedano, po definiciji ni več inteligenca, ker ni več izvirno porajajoče, že obstaja. Tu nastopa problem relativnosti inteligence: kakor za koga! Program lahko deluje navzven kot inteligentna lastnost in če ga testiramo npr. na njegovo numerično sposobnost, lahko to lastnost ugotovimo podobno kot pri človeku. Uporaben računalniški program vsebuje za zunanjega opazovalca nekaj, kar je povezano z lastnostjo inteligence.

10. UI je mehanična (formalizirana, algoritmična, interaktivna) inteligenca. Svoje meje si postavlja v okviru mogočega, uresničljivega. Pri današnji obdelavi podatkov, informacij in znanja lahko opredeljujemo strojno (računalniško), programsko in sistemsko inteligenco. Ti termini bodo še ustrezneje veljavni v prihodnosti, saj bo inteligenčni količnik strojev, programov in sistemov naraščal. Z rastjo mehanične inteligence se bodo širile tudi meje UI in v določenem trenutku bodo te meje lahko tudi presegle zmogljivosti naravne inteligence. Seveda pa lahko velja takšno izjavljanje danes kot protikulturno, ker kultura le težko priznava, da bi lahko postal stroj, ta simbol suženjstva, pametnejši, izvirnejši, zmogljivejši in sposobnejši od človeka pa tudi od poljubne skupine izvedencev.

11. Perspektive UI so sicer še nekoliko oddaljene, so pa že na vidiku. Človek tega stoletja se je odločil, da razvije UI v pomoč sebi in svojemu obstoju. Nove generacije računalnikov ne bodo samo pametnejše, bodo tudi biologizirane in naposled oživljene s pomočjo lastne, od človeka neodvisne inteligence. Od določene razvojne stopnje naprej se bodo stroji lahko sami vzdrževali, popravljali, načrtovali, sestavljali oziroma reproducirali z uporabo svoje umetne, nenaravne inteligence. Ta inteligenca bo seveda lahko močno različna od tkim. naravne, človekove inteligence, lahko bo raznovrstnejša in zmogljivejša. Tudi s tem se bo človek naposled sprijaznil, namreč z obstojem življenja strojev.

12. Čeprav se večkrat dozdeva, da razmišljajo današnji umetni inteligenčniki prek svojih sposobnosti in se jim nekorektno očita enoumnost, robotoumnost in temu podobno, je vendar treba priznati, da razvoj inteligentnih strojev pospešeno napreduje in da so bistveni praktični rezultati le časovno vprašanje, ki je povezano z razvojem novih, ključnih tehnologij.

13. Podcenjevanje razvoja UI na področju strojegradnje (robotov, računalniških sistemov) pomeni prelaganje nujnosti od danes na jutri. UI je pred vrati, ne še utelešena, vendar s svojo senco in slutnjo. Zato je prav, da spremljamo njen razvoj in se na njen prihod pripravljamo.

NOVICE IN ZANIMIVOSTI

Šaljivo in tragično

Sliši se šaljivo, v resnici pa je tragično: imamo možnosti, da bomo preživeli, kajti poslejmo revno Evropo. Stežka je zbrala milijardo in 200 milijonov dolarjev za ESPRIT in bo za ta denar dobila nekaj računalništva. Videti je, da smo mi veliko bogatejši: za Obrovac, za Feni, za naše gore list Gorenje smo zbrali milijardo in pol dolarjev in jih vrgli proč. To se pravi, da denar je. Za nekaj drugega gre: nekdo pri nas ravna z denarjem popolnoma neodgovorno. Akcija za računalništvo kot sestavni del filozofije novih "možganov" se začenja pri boju za oblast, pri boju za akumulacijo, s katero razpolaga danes tisti sloj, ki se računalništva ne more več naučiti. Ta sloj pa je potreben samo v družbi, ki računalništva in modernega razmišljanja ne obvladuje. In akcija za preživetje bo akcija za spopad s takimi silami.

(E. Vrenko v Računalniška nepismenost in družbeni razvoj, MC MK ZKS Ljubljana, 1984, str.53)

(ESPRIT je okrajšava za Evropski strateški program za raziskave v informacijski tehnologiji.)

A. P. Železnikar

Prvih deset na nemškem tržišču programske opreme

Na nemškem tržišču mikroračunalniške programske opreme je končno po nemških (ne ameriških) podatkih na seznamu te opreme deset najbolj prodajanih poklicnih programskih paketov. Ti paketi so uporabniško usmerjeni, vendar se še ne prodajajo v zelo velikih količinah. Tabela prvih 10 paketov izgleda takole:

mesto	programski paket	proizvajalec -prodajalec	kategorija izdelka
1	Wordstar	Micropro	obdelava besedila
2	Lotus 1-2-3	Lotus Development	integrirani paket
3	Multiplan	Microsoft	izračuni v preglednicah
4	dBase II	Ashton Tate	sistem banke podatkov
5	Open Access	SPI	integrirani paket
6	Symphony	Lotus Development	integrirani paket
7	Tex-Ass	Bongartz in Schmidt	obdelava besedila
8	Fibu	IBM	integrirani paket
9	Appleworks	Apple	integrirani paket
10	Chart	Microsoft	poslovna grafika

A. P. Železnikar

Nove knjige

Nekatere nove knjige založbe Addison-Wesley (razdobje januar-april) so tele:

A.V.Aho, R.Sethi, J.D.Ullman: Compilers: Principles, Tools and Techniques (0 201 10088 6, cena 32,95 dolarjev).
Vsebina: Uvod v prevajalnike. Enostavni enoprehodni prevajalnik. Besedna analiza. Sintaksna analiza. Sintaksno vodeno prevajanje. Preizkušanje tipov. Upravljanje pomnilnika v izvajalnem času. Vmesno generiranje koda. Kodno generiranje. Kodna optimizacija. Razvoj prevajalnika. Pogled v posamezne prevajalnike.

E.Charniak, D.McDermott: Introduction to Artificial Intelligence (0 201 11945 5, cena 29,95 dolarjev).
Vsebina: Umetna inteligenca in notranja predstavitev. Lisp. Videnje: od luči k notranji predstavitvi. Analizni jezik. Iskanje. Predikatni račun: predstavitev znanja in dedukcija. Organizacija pomnilnika in dedukcija za posebne namene. Abdukcija in ukrepanje pri negotovosti. Upravljalni načrti akcije. Jezikovna obsežnost. Učenje.

M.Kittner, B.Northcutt: Introduction to Basic: A Structured Approach (0 8053 4302 4, cena 24,95 dolarjev).
Vsebina: Uvod v obdelavo podatkov in strukturirano programiranje. Začetek. Vhod-izhod. Računanje in funkcije. Krmilne strukture. Več o zankah. Strukturirano programiranje in menuji. Polja. Napredni koncepti. Nizna manipulacija. Zbirke. Glosarij.

S.Manna, R.Waldinger: The Logical Basis for Computer Programming, Volume 1: Deductive Reasoning (0 201 18260 2, cena 30,95 dolarjev).
Vsebina: Del I: Matematična logika. Izjavna logika. Predikatna logika, osnovna in napredna. Posebne teorije. Del 2: Teorije z indukcijo. Nenegativna cela števila. Nizi. Drevesa. Seznami. Množice. N-terice.

E.B.Koffman: Problem Solving and Structured Programming in Pascal. 2-E (0 201 11736 3, cena 28,70 dolarjev).
Vsebina: Uvod v računalnike in programiranje v Pascalu. Reševanje problemov. Del 1: Reševanje problemov. Del 2. Enostavni podatki. Podatkovni tipi. Krmilni stavki. Polja. Zapisi in množice. Rekurzija, iskanje in sortiranje. Zbirke. Kazalčne spremenljivke in povezane podatkovne strukture.

A. P. Železnikar

DOMAČA GRAFIČNA OPREMA SNOVANJE, PREDSTAVITEV IN IZRISOVANJE ČRNOBELIH SLIK

V Odseku za računalništvo in informatiko Instituta Jožef Stefan v Ljubljani ob podpori Raziskovalne skupnosti Slovenije razvijamo, implementiramo in prototipno izdelujemo grafično aparaturno in programsko opremo za programiranje, predstavitev in izrisovanje črnobelih slik na družini računalnikov Iskra-Delta ter DEC pod operacijskimi sistemi RT-11, RSX-11, VMS ter njihovimi domačimi izvedbami. Na sedanji stopnji razvoja lahko ponudimo končnim uporabnikom ter računalniškim proizvajalcem paket grafične aparaturne in programske opreme, ki obsega:

- standardni grafični programski paket GKS za računalnike pod operacijskim sistemom VMS;
- grafični procesor kot dodatek za videoterminal KOPA 1000 oziroma DEC VT100;
- grafični dodatek za risanje na matričnem pisalniku DEC LA-120;
- grafični vmesnik za risanje na matričnem pisalniku FACIT 4540;

V bližnji prihodnosti pa bo dokončan razvoj naslednje grafične opreme:

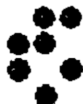
- digitalizacijska tablica;
- grafični procesor za videoterminal Gorenje;
- programska knjižnica programiranje grafike na miniračunalnikih tipa DEC PDP-11 in LSI-11 in podobnih računalnikih Iskra-Delta ter na podobnem računalniku IJS PMP-11;

GRAF-100
GRAFIČNI PROCESOR ZA VIDEOTERMINAL KOPA 1000 (VT100)

Institut Jožef Stefan je razvil in izdeluje grafični procesor GRAF-100 za vgradnjo v videoterminal Kopa 1000 oziroma VT100. S tem dodatkom pridobi videoterminal zmožnosti grafičnega terminala z ločljivostjo 650 x 240 svetlobnih točk ter pri tem ohrani vse lastne zmožnosti alfanumeričnega terminala. Bistvena prednost tega grafičnega procesorja pred uvoženimi procesorji tega tipa je v velikem številu (16) nivojev svetlobne intenzivnosti posamezne točke. To zmožnost procesor GRAF-100 izrablja za navidezno dvakratno povečanje ločljivosti s pomočno operacij za odpravo stopničenja (anti-aliasing) - zmožnost, ki so jo doslej omogočali le grafični procesorji najvišjega cenovnega razreda. Zmožnost risanja z velikim številom poltonov med črno in belo barvo omogoča uporabo tega grafičnega terminala za upodabljanje prostorskih objektov v strojništvu, gradbeništvu, lesarstvu, elektroniki in drugod.

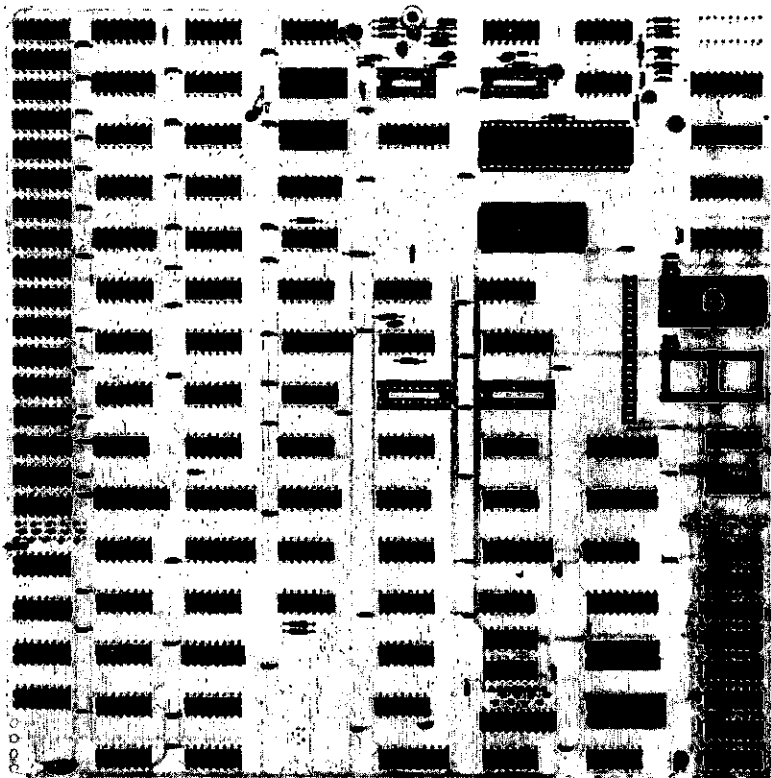
LAGRAF-120
GRAFIČNI DODATEK ZA RISANJE NA MATRIČNEM PISALNIKU DEC LA-120

Grafični dodatek LAGRAF-120 omogoča uporabo matričnega pisalnika DEC LA-120 za rastrsko risanje z visoko ločljivostjo. Pri tem tiskalnik ohrani vse svoje zmožnosti za alfanumerično tiskanje. Dodatek LAGRAF-120 omogoča risanje z ustreznimi ukaznimi nabori, ki so kompatibilni z DECwriter IV-RA. Velikost in poraba električne energije sta manjši v primerjavi s podobnim dodatkom Selanar SG-120. Vgradnja plošče je zelo enostavna, tako da jo lahko izvede vsak brez posebnega orodja v nekaj minutah.

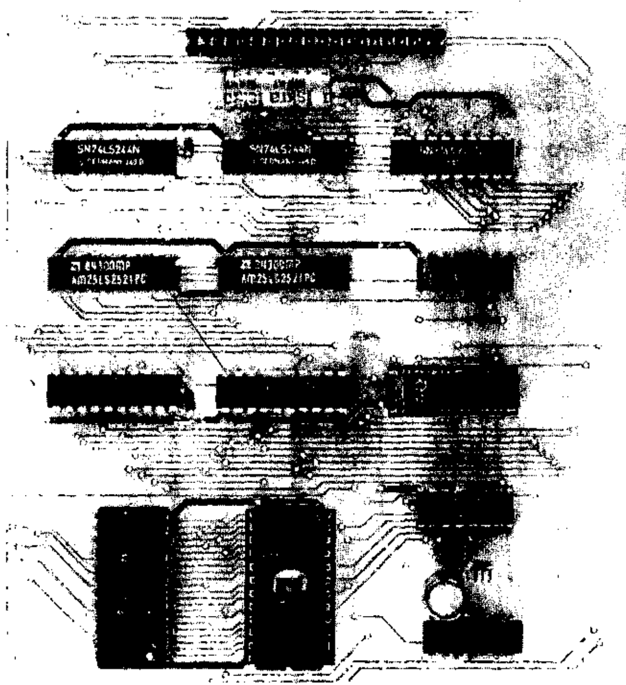


institut "jožef stefan" ljubljana, jugoslavija

GRAF–100
GRAFIČNI PROCESOR ZA VIDEOTERMINAL KOPA 1000 (VT100)



LAGRAF–120
GRAFIČNI DODATEK ZA RISANJE NA MATRIČNEM PISALNIKU DEC LA–120

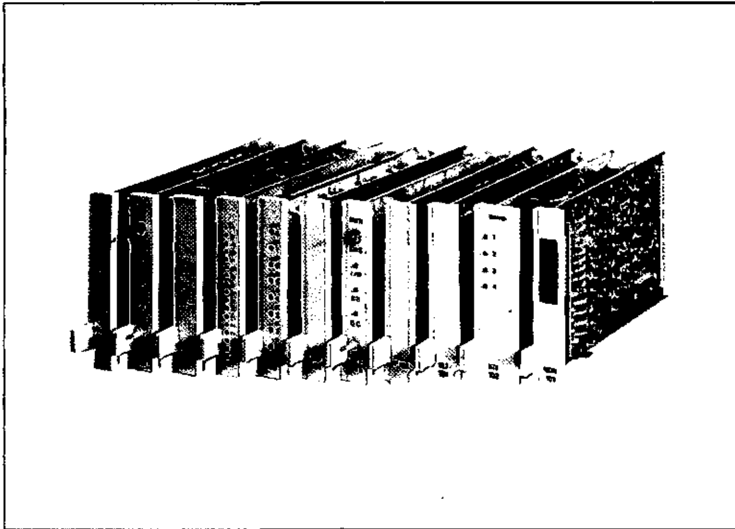


gorenjeprocesna oprema

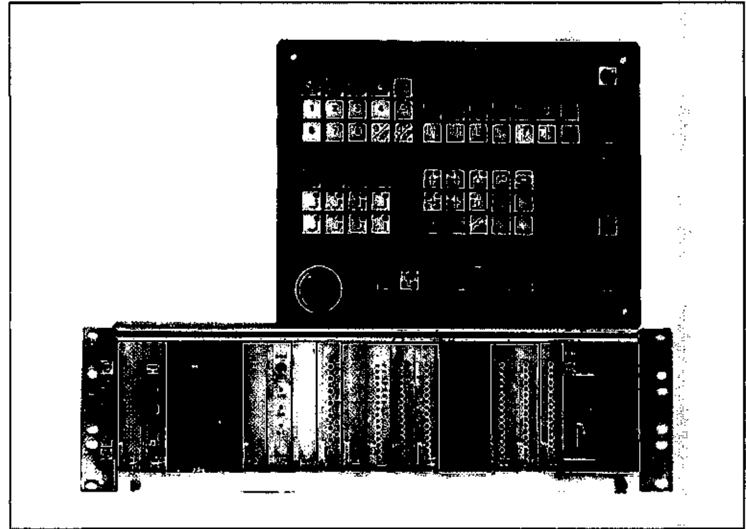
Gorenje Procesna oprema, n.sol.o.
Celjska 5a
63320 Titovo Velenje
Telefon: (063) 850 030, 851 000
Telex: 33547 yu tgove

PLK 1000
Programirljiv krmilni sistem
PLK 1000
Programabilni sistem za upravljanje

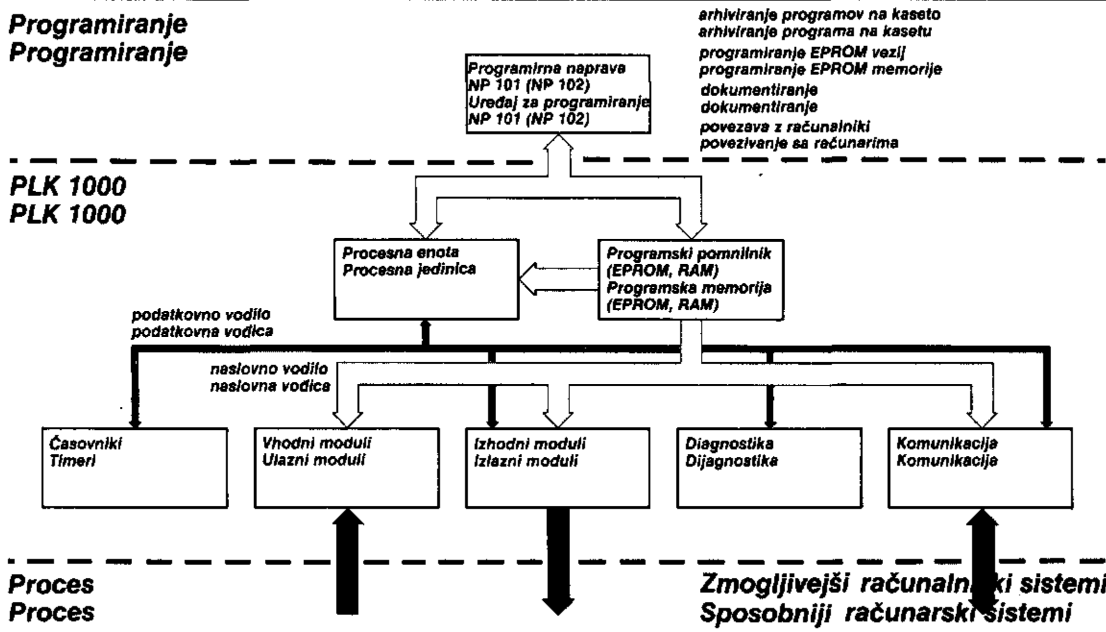
gorenje procesna oprema



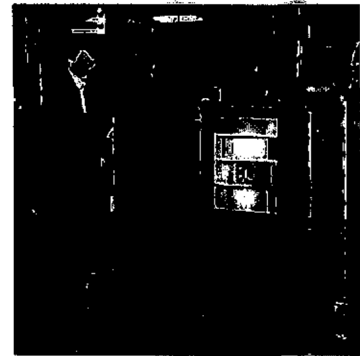
Moduli sistema
Moduli sistema



Krmilnik stružnega avtomata
Upravljač automatske tokaralice



Shema sistema
Šema sistema



Krmiljenje v proizvodnji avtomobilskih plaščev
Upravljanje u proizvodnji autoguma

Opis sistema

PLK 1000 na osnovi programa, ki ga napiše uporabnik preko vhodov spremlja stanje v procesu in daje krmilne signale na izhode za vklapljanje oziroma izklapljanje najrazličnejših porabnikov.
Sistem je zasnovan modularno in ga lahko načrtujemo od najmanjših obsegov do njegove največje zmogljivosti, t.j. do 512 vhodov in 512 izhodov.
Pri večjem številu zahtevanih vhodov in izhodov lahko med seboj povežemo več sistemov, ki delujejo sinhrono.
Sistem PLK 1000 lahko preko posebnega komunikacijskega kanala dvosmerno povežemo z računalniki v kompleksne informacijske sisteme. S tem dosežemo, da določen tehnološki proces nadzorujemo in vplivamo nanj še z drugih nivojev, ne samo s PLK 1000.

Opis sistema

PLK 1000 na osnovu programa, izrađenog od strane korisnika preko ulaza prati stanje u procesu i daje signale za upravljanje izlazima za uključivanje i isključivanje najrazličitijih korisnika.
Sistem se zasniva modularno i možemo ga planirati od najmanjeg opsega do njegovog najvećeg kapaciteta, t.j. do 512 ulaza i 512 izlaza.
Kod većeg broja traženih ulaza i izlaza može se među sobom povezati više sistema koji rade sinhrono.
Sistem PLK 1000 može se preko posebnog komunikacionog kanala povezati sa računarima u kompleksne informacione sisteme. Sa time postiže se, da uz sistem PLK 1000 određeni tehnološki proces kontrolišemo i utičemo nanj i sa drugih nivoa.

Prednosti sistema

PLK 1000 je najcenejša rešitev med do sedaj znanimi rešitvami.
Zavzema zelo malo prostora.
Zagotavlja veliko obratovalno zanesljivost.
Spreminjanje funkcij je hitro in enostavno, prav tako kot tudi načrtovanje sistema.
Enostaven za rokovanje in servisiranje.
Vsaka okvara na stroju ali tehnološki liniji nam aktivira sistem za diagnostiko. Smer ali mesto okvare nam PLK 1000 prikaže v številčni obliki.

Prednosti sistema

PLK 1000 jeste najjeftinije rešenje među do sada poznatim rešenjima.
Zauzima malo prostora.
Obezbeđuje visoku pogonsku sigurnost.
Menjanje funkcija je brzo i jednostavno, istotako i planiranje sistema.
Jednostavan za rukovanje i servisiranje.
Svaki kvar na mašini ili tehnološkoj liniji aktivira sistem za dijagnostiku. Pravac ili mesto kvara PLK 1000 javlja u brojčanom obliku.