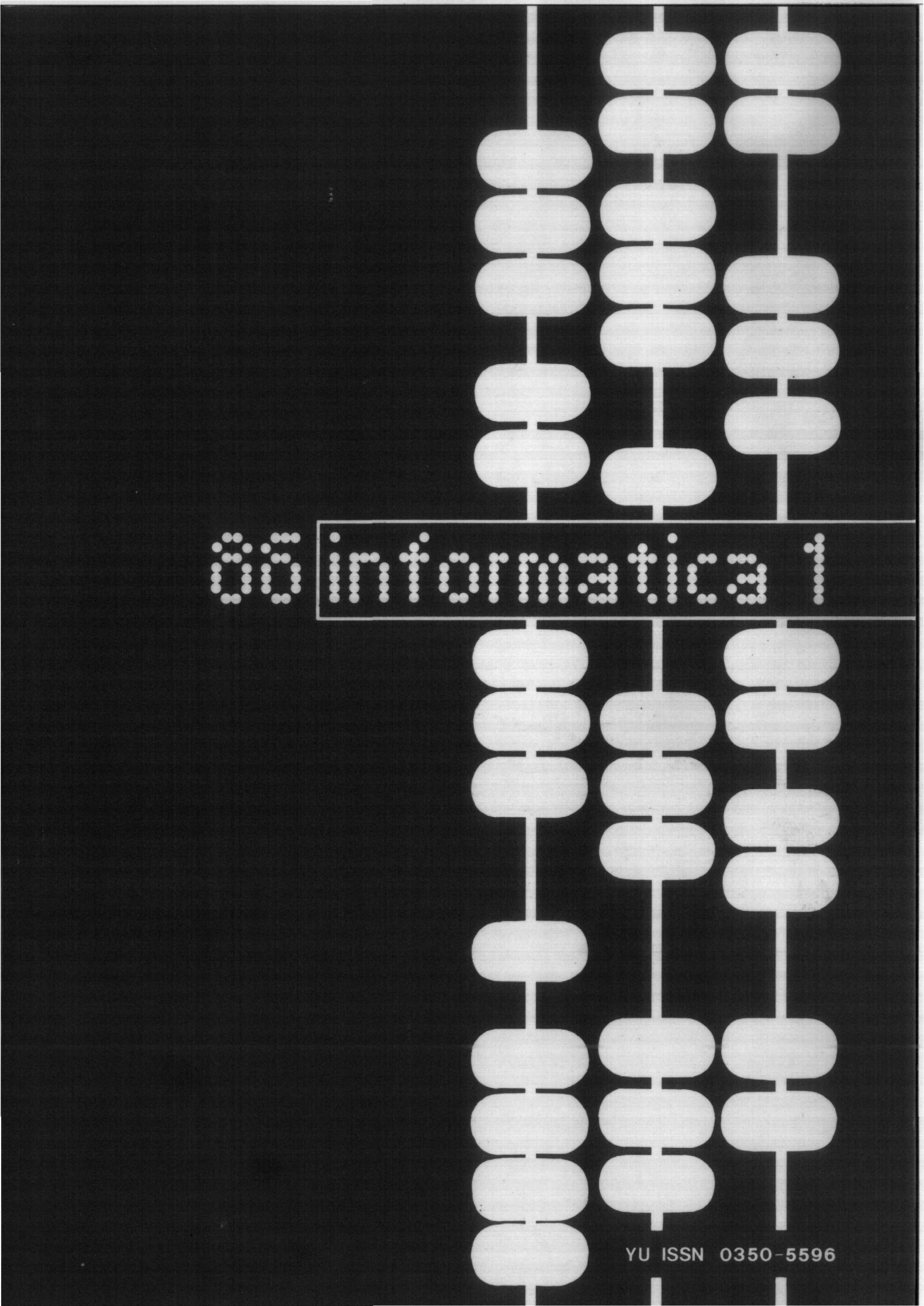
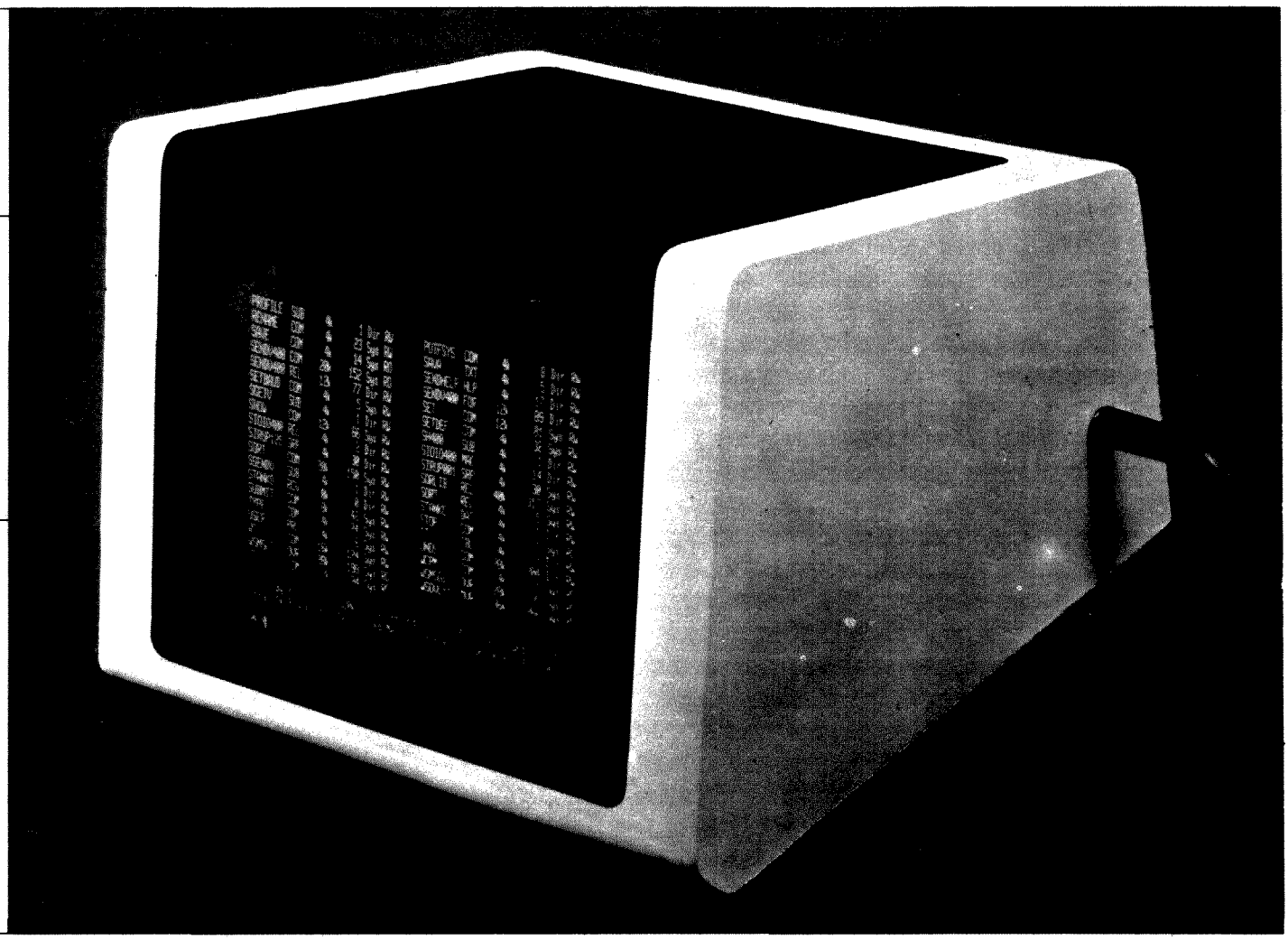


YU ISSN 0350-5596

SISTEM ZA ŠALTERSKO POSLOVANJE V BANKAH IN NA POŠTAH



računalniški sistemi delta®

Sistem za šaltersko poslovanje je sodobna računalniška oprema za delo v bankah in na poštah, opremljen z ustrezno programsko opremo.
Sistem omogoča samostojno ažurno poslovanje – od posameznih operativnih del na šalterjih do zajema podatkov za nadaljnjo obdelavo. Deluje lahko povsem samostojno ali v povezavi z glavnim računalnikom (prenos informacij je mogoč prek stalno najetih ali navadnih telefonskih linij). Delovanje sistema tudi ni odvisno od razpoložljivosti računalniških kapacitet glavnega računalnika.
Sistem nadomešča raznovrstno opremo, ki se uporablja pri šalterskem poslovanju – od klasičnih mehanografskih strojev, pisalnih strojev do kalkulatorjev in deloma mikročitalnikov.

Sistem za šaltersko poslovanje je savremena računarska oprema za rad u bankama i poštama, opremljen sa odgovarajućom programskom opremom.
Sistem omogućava samostalno ažurno poslovanje – od pojedinih operativnih poslova na šalterima do zahvata podataka za dalju obradu. Može da radi sasvim samostalno, ili da komunicira sa glavnim računarom (prenos informacija je moguć preko stalno iznajmljenih ili običnih telefonskih linija).
Rad sistema je takođe nezavisan od raspoložljivosti računarskih kapaciteta glavnog računara.
Sistem zamenjuje raznovrstnu opremu, koja se upotrebljava u šalterskom poslovanju – od klasičnih mehanografskih mašina, pisaćih mašina do kalkulatora i delimično čitača mikrofiševa.

informatica

JOURNAL OF COMPUTING AND INFORMATICS

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

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

PRINTED BY: Tiskarna Kresija, Ljubljana

DESIGN: Rasto Kirn

YU ISSN 0350-5596

VOLUME 10, 1986 – NO. 1

CONTENTS

informatica

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

YU ISSN 0350–5596

LETNIK 10, 1986 – ŠT. 1

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

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

LETNA NAROČNINA za delovne organizacije znaša 1900 din, za redne člane 490 din, za študente 190 din; posamezna številka 590 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

VSEBINA

JEDNA MOGUĆNOST PROŠIRENJA SKUPA UPRAVLJAČKIH NAREDBI RATFOR PREPROCESORA ZA FORTRAN

Kemenci Zoltan, Institut za mernu tehniku i upravljanje
Fakultet Tehničkih Nauka, Novi Sad
Turčić Tomislav, Rade Končar, "Industrijska elektronika"
Zagreb

UDK : 519.682.8

ABSTRAKT: U radu je opisana jedna vrsta unutrašnjeg potprograma koja postoji u nekim preprocesorima za FORTRAN, a nije je moguće realizovati standardnim upravljačkim naredbama RATFOR preprocesora. Dat je način opisivanja strukture unutrašnjeg potprograma odgovarajućim FORTRAN naredbama. Opisan je "hash" algoritam i dat je funkcijski potprogram za prevodjenje tekstualnog naziva unutrašnjeg potprograma u odgovarajući deo FORTRAN simbola.

AN APPROACH TO ENLARGE THE SET OF CONTROL STATEMENTS OF THE RATFOR PREPROCESSOR FOR FORTRAN

ABSTRACT: This paper describes an internal subroutine structure found in some preprocessors for FORTRAN, but not readily implementable using the standard RATFOR preprocessor control statements. A method for representing the internal subroutine structure by using FORTRAN statements is given. The hashing algorithm used is described, and a function subroutine for translating internal subroutine names to corresponding parts of FORTRAN symbols is given.

1. UVOD

U savremenim uslovima realizacije složenih softverskih sistema od izuzetnog su značaja ALATI koji stoje na raspolaganju timu ljudi koji rade taj posao. Problem izbora programskog jezika je još i danas aktuelan, jer ne postoji jezik kojeg svako dobro poznaje i postoji na svakom računaru. Upotrebom FORTRANa se dobija na univerzalnosti i portabilnosti sistema, dok su ograničenja u relativno skromnim kontrolnim strukturama ovog jezika. Kao moguće rešenje nameće se upotreba preprocesora koji FORTRAN tretira kao mašinski jezik, dok omogućava korisniku pisanje izvornih programskih modula u slobodnoj, modernoj formi, dozvoljava upotrebu struktuiranih upravljačkih naredbi i korišćenje biblioteka parametrizovanih izvornih modula. Ovo u mnogome olakšava "top-down" dizajn sistema, dobijanje programskog koda sa jasno kontrolisanim tokom i odličnim dokumentacionim mogućnostima.

RATFOR makro - preprocesor za FORTRAN sa upravljačkim naredbama preuzetim iz jezika C predstavlja odličan i poznat preprocesor. Medjutim, prilikom implementacije softverskih sistema pisanih u jeziku nekog drugog preprocesora može se javiti kontrolna struktura koja se ne može opisati kombinacijom postojećih upravljačkih naredbi, kao što je to slučaj sa unutrašnjim potprogramima čija je implementacija u RATFOR tema ovog rada.

2. UNUTRAŠNJI POTPROGRAMI

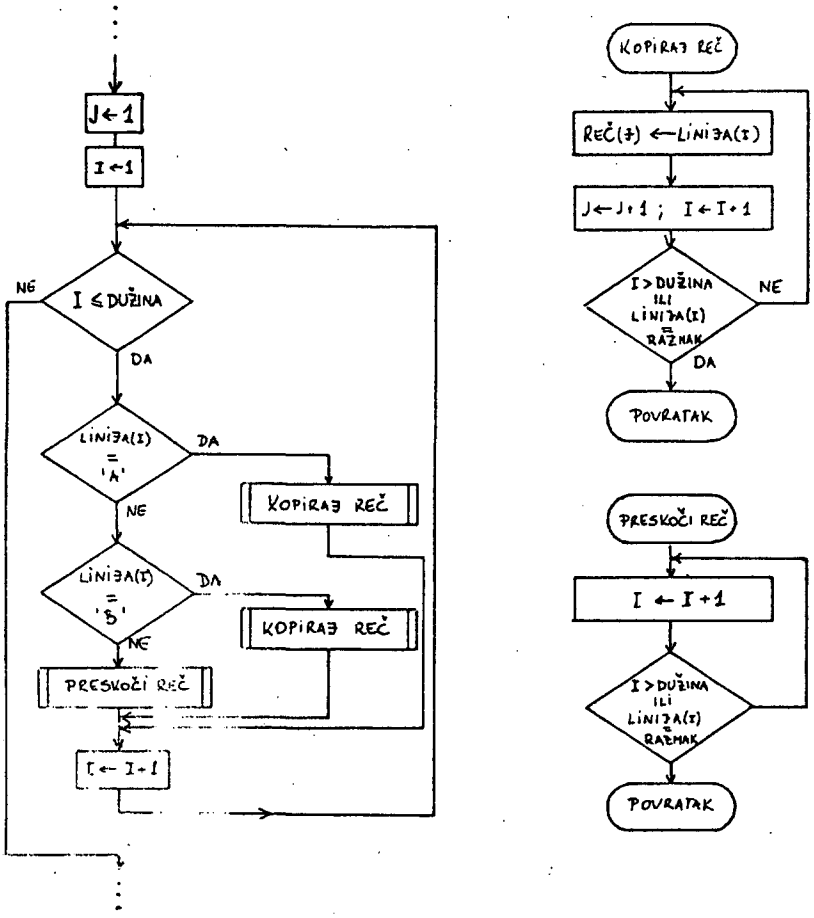
Većina preprocesora za FORTRAN poseduje upravljačke naredbe tipa IF - ELSE, REPEAT, WHILE, FOR, itd. samo neki od njih, medjutim, poseduju mogućnost definisanja skupa naredbi kao jedne celine, izvršavajući taj skup pomoću neke upravljačke naredbe tipa PERFORM (COBOL), odredjujući taj skup naredbi njegovim jedinstvenim tekstualnim nazivom. Ovakav skup naredbi naziva se unutrašnjim potprogramom ili procedurom (različitog tipa od procedure u Pascal-u ili PL/1). Definicija procedure važi samo za tekući programski modul u kojem se nalazi. Unutar procedure dostupne su sve promenljive koje su dostupne u istom programskom modulu. Ovakve procedure mogu biti vrlo korisne i mogu znatno doprineti razumljivosti i preglednosti osnovnog toka obrade programskog modula.

Kao ilustrativni primer na slici 1. dat je blok - dijagram algoritma za jedan deo hipotetičkog modula koji iz ulazne linije izdvaja reči koje počinju sa slovom A ili B. Odgovarajući kod koji realizuje datu obradu prikazan je na slici 2.

Primer je napisan u RATFOR-u kojem je ugradjena mogućnost obrade ovakvih unutrašnjih potprograma. U primeru se vidi da procedura "kopiraj reč" obuhvata nekoliko naredbi koje treba izvršiti na više mesta unutar datog modula a iz ekonomskih razloga ne želimo pisati u obliku spoljašnjeg potprograma opšteg ili funkcijskog tipa. Ovakav način pisanja znatno doprinosi razumljivosti i čitljivosti koda i smanjuje mogućnost pojavljivanja grešaka.

3. USLOVI I NAČIN REALIZACIJE: PREVODJENJE TEKSTUALNOG NAZIVA U FORTRANSKI SIMBOL

Da bi preprocesor mogao formirati odgovarajuće FORTRAN-ske upravljačke naredbe koje realizuju tok upravljanja procedure potrebno je prevesti tekstualni naziv procedure u odgovarajući simbol (ili njegov deo) koji će se koristiti prilikom formiranja kontrolnih tabela



Slika 1. - Blok dijagram algoritma

```
...
i = 1;  j = 1
WHILE ( i <=dužina) {
   IF ( linija(i) == "A" ){          #kopiraj reči koje
      PERFORM (Kopiraj reč)          #počinju sa A
      }
   ELSE IF (linija (i) == "B"){      #kopiraj reči koje
      PERFORM (Kopiraj reč)          #počinju sa B
      }
   ELSE {
      PERFORM (Preskoči reč)
      }
   i = i+1;  j = j+1                 #preskoči razmak
   }
...
RETURN                    #završetak glavnog dela modula
PROCEDURE (Kopiraj reč)       #definicija procedure 1
REPEAT {
   reč (j) = linija (i)
   i = i+1;  j = j+1
   }
UNTIL ( i > dužina | linija (i) == RAZMAK )
ENDPROCEDURE                  #kraj procedure
PROCEDURE (Preskoči reč)      #definicija procedure 2
REPEAT
   i = i+1
UNTIL (i > dužina | linija (i) == RAZMAK)
ENDROPROCEDURE                #kraj procedure 2
END                       #kraj modula
```

Slika 2. Kod u RATFOR-u

i promenljiva.

Zbog ograničenja jezika FORTRAN:

- maksimalna dužina simbola je 6 znakova,
- labele su brojevi sa maksimalnom dužinom od 5 znakova,
- nazivi promenljiva počinju slovnim znakom,

deo simbola koji odgovara tekstualnom nazivu procedure ograničen je na 4 znaka.

Očigledno je da:

- mogućin tekstualnih naziva ima više nego varijacija sa 4 znaka,
- nazivi mogu biti slični do te mere da se razlikuju u samo nekoliko znakova,
- dužina tekstualnog naziva je ograničena na neku realnu vrednost (napr.: 40 znakova),

pa se kao rešenje nameće odgovarajući "hash" algoritam koji tekstualni naziv pretvara u ceo broj u opsegu od 1000 do 9999 jer se:

- dodavanjem cifre (1, 3, 4, ...) ispred tog broja dobija labela koja se sa velikom verovatnoćom razlikuje od korisničkih labela. Ova labela se koristi za označavanje početka procedure;
- dodavanjem slova ispred tog broja dobija se celobrojna promenljiva koja se koristi u "assigned GOTO" naredbi na kraju procedure.

Na slici 3. prikazan je sistem prevodjenja upravljačkih naredbi vezanih za procedure. Broj koji se dobija "hash"-iranjem naziva procedure označen je sa /broj/:

```
RATFOR:                                  FORTRAN:
...                                              ...
PERFORM (tekstualni naziv)                       ASSIGN Lpovr TO K/broj/
                                                 GOTO 3/broj/
                                         Lpovr   CONTINUE
PROCEDURE (tekstualni naziv)             3/broj/ CONTINUE
...  #naredbe unutar procedure                   ...
ENDROPROCEDURE                                   GOTO K/broj/
```

Slika 3. Procedura u RATFOR-u i FORTRAN-u

```
INTEGER FUNCTION hash (a, n)
INTEGER     n                    #dužina ulaznog niza A
CHARACTER   a(n)                 #ulazni niz A
REAL        b(40)                #niz realnih brojeva B
REAL        r                    #radna promenljiva
n = MIN (40, n)                  #osigurajmo N
FOR (i=1; <=n; i=i+1)            #prenesi niz A u niz B
   b(i) = i_to_r(c_to_i(a(i)))   #sa pretvaranjem znakova u realne brojeve
FOR (i=3; i>n; i=i-1)            #za nizove kraće od 3 znaka
   b(i)=0                        #popuni nulama do b(3)
n = MAX (3, n)
FOR (i=3; i <=n; i=i+1)          #transformacija niza B
   b(i) = (b(i-2)/1231.+b(i-1)/1163.+b(i)/881.)%
          * 3275./9.
r=b(n) - i_to_r(INT(b(n)))       #0 <=r<1
hash = 1000 + MOD(INT(r*10000),9000)
#1000 < = hash <= 9999
RETURN
END
```

Slika 4. - Funkcijski podprogram HASH

NAPOMENA: - p1 = 1231, p2 = 1163, p3 = 881;

- i_to_r, c_to_i odgovarajuće funkcije konverzije različitih tipova podataka (integer - real, character - integer). Za različite FORTRAN prevodioce oni mogu imati različite oblike ili se mogu jednostavno izostaviti.

3. METODA PREVODJENJA TEKSTUALNOG NAZIVA U ODGOVARAJUĆI BROJ: "HASH" ALGORITAM

"Hash" algoritam korišćen za pretvaranje tekstualnog naziva u odgovarajući broj osniva se na deljenju ASCII kodova znakova velikim prostim brojevima i uzimanjem prvih nekoliko decimalnih cifara rezultata.

Ulazni niz znakova A dužine N <N =40) pretvara se u niz realnih brojeva, gde su b(i)=0 za N < i < 3. Niz B transformiše se za i=3,4,... N prema sledećoj formuli:

$$b(i) = \left(\frac{b(i-2)}{p1} + \frac{b(i-1)}{p2} + \frac{b(i)}{p3} * \frac{p1+p2+p3}{9}\right.$$

gde su p1, p2 i p3 veliki prosti brojevi.

Na kraju, uzimaju se 4 decimalne cifre poslednje formiranog realnog broja b(N) i podešavaju se u željeni opseg.

Ovu obradu realizuje funkcijski podprogram HASH prikazan na slici 4. Zbog preglednosti i rečitosti samog programa izostavljen je odgovarajući blok dijagram algoritma.

Testiranje ovog algoritma izvršeno je primenom na stotinak različitih naziva procedura nekoliko softverskih paketa. Ni u jednom slučaju se nije ponovio rezultat za različite nazive što je sasvim zadovoljavajuće za praktičnu primenu. Uz odgovarajuće provere i eventualne modifikacije algoritam se može primeniti i u drugim oblastima gde se ovakvi algoritmi redovito primenjuju.

4. ZAKLJUČAK

Opisani metod uvodjenja struktuiranih naredbi za rukovanje unutrašnjim potprogramima je poslužio kao osnova za implementiranje makro funkcije za "hash" - iranje raznih tekstualnin naziva u RATFOR preprocesor. Pomoću te i ostalih postojećih makro funkcija definisane su naredbe tipa PERFORM, PROCEDURE i ENDRO-PROCEDURE. Njihovo korišćenje prilikom pisanja programskih celina u znatnoj meri doprinosi mogućnosti "top-down" projektovanja i razumljivosti koda, dok sa druge strane, ne vezuje za odredjeni jezik, jer se i sam preprocesor a i generisani programski moduli oslanjaju na običan standardni FORTRAN.

Na kraju, autori se zahvaljuju prof. dr Danilu Obradoviću sa Instituta za mernu tehniku i upravljanje u Novom Sadu, na korisnim sugestijama datim u toku razvoja rada.

5. LITERATURA

1. B.W. Kernighan, P.J.Plauger: Software Tools. Addison - Wesley 1976.

2. J.Martin: Computer Data-base Organization. Prentice - Hall, Englewood Cliffs, N.J. 1975.

LOGIČNI MODELI RAČUNALNIŠKIH STRUKTUR II

MAKSIMILJAN GERKEŠ
TEHNIŠKA FAKULTETA, MARIBOR
VTO ELEKTROTEHNIKA, RAČUNALNIŠTVO IN INFORMATIKA
METALNA, MARIBOR
TOZD TOVARNA INVESTICIJSKE OPREME

UDK : 681.3.517.11/12

POVZETEK: Poljuben označen usmerjen graf z enim začetnim in končnim vozliščem, pri katerem se vse poti sklenejo med tema vozliščema, je preoblikovan v vase zaključeno selektorsko operacijo, tudi če vsebuje zanke s končno mnogo ponovitvami. Za izgradnjo modelov sekvenčnih in mikroprogramiranih strojev so izdelani postopki dekompozicije s pomočjo karakterističnih lastnosti, kar lahko vodi do realizacije, ki ima več sekvenčnih oz. mikroprogramiranih strojev kot začetni model. Definiran je posplošen model operatorja s selektorsko operacijo in ga je možno razgraditi do poljubnih detaljev. Izdelani so modeli konkretnih računalniških struktur.

ABSTRACT: LOGICAL MODELS FOR COMPUTER STRUCTURES II. Any labelled directed graph with single input and single output node in which all directed paths are terminated at the output node, can be modelled with a select operation in the self loop, even if loops with a finite number of repetitions are presented in the initial graph. For a model design of a sequential or microprogrammed maschine procedures were defined, that can lead to realizations with some sequential or microprogrammed maschines, altough initial model was single maschine model. Abstract operator with the ability to decompose it into any required detail is defined. Models for well known computer structures were built.

UVOD:

Poljuben označen usmerjen graf, ki ima eno začetno in eno končno vozlišče in v katerem vse usmerjene poti vodijo od začetnega h končnemu vozlišču lahko preoblikujemo v graf vase zaključene selektorske operacije. Takšno preoblikovanje usmerjenega grafa je možno tudi, če graf vsebuje operacije v zankah, ki imajo končno število ponovitev. Iz tega razloga lahko poljuben program, mikroprogram, ... realiziramo v obliki sekvenčnega stroja oz. mikroprogramsko krmiljenega stroja. Kadar pa vase zaključena selektorska operacija rabi kot izhodiščni model za snovanje izbrane računalniške strukture lahko s postopki dekompozicije preoblikujemo to operacijo v sestavljeno vase zaključeno selektorsko operacijo, katere realizacija lahko v splošnem zahteva izgradnjo več povezanih sekvenčnih strojev ali mikroprogramsko krmiljenih strojev.

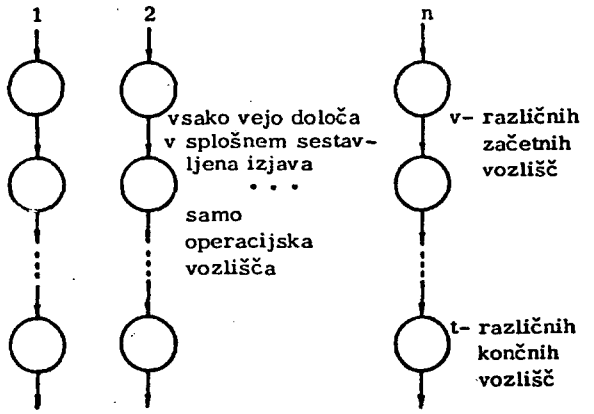
Pri dekompoziciji kompleksnih sekvenčnih strojev - kot so naprimer modeli procesorjev je smiselno uporabiti splošnejši pristop k dekompoziciji in sicer z uporabo karakterističnih lastnosti specifikacije, katerim podrejamo dekompozicijo izbrane strukture. Odstopanja od karakterističnih lastnosti pa obravnavamo kot izjeme, s katerimi korigiramo osnovno strukturo, dobljeno samo na podlagi upoštevanja karakterističnih lastnosti. Za snovanje operatorjev sekvenčnih oz. mikroprogramiranih strojev je uporabljen abstraktni pristop, ki omogoča definicijo posplošenega modela operatorja ter njegovo dekompozicijo do poljubnih detaljev. Predlagani so modeli konkretnih računalniških struktur, izgrajeni s pomočjo opisanih postopkov modeliranja od krmilnih enot preko pomnilnih struktur do operatorjev.

1. PRETVORBA GRAFA OPERACIJ V GRAF VASE ZAKLJUČENE SELEKTORSKE OPERACIJE

Kot iztočnico vzemimo označen usmerjen graf z enim začetnim in enim končnim vozliščem. Graf naj ne vsebuje operacij v zankah. Vse poti, ki izhajajo iz začetnega vozlišča morajo končati v končnem vozlišču. Vsem operacijskim vozliščem (Op_i, i = 1, 2, ..., n) pripišemo izjave Q_j, ki specificirajo naslednja operacijska vozlišča. Vsaki usmerjeni poti, ki povezuje začetno in končno vozlišče lahko tedaj priredimo graf samih sekvenčno povezanih operacij. Tak graf - sekvenčno operacijo pa lahko po [1] preoblikujemo v vase zaključeno selektorsko operacijo.

Vzemimo, da najdemo v označenem usmerjenem grafu G n-usmerjenih poti, ki vodijo od začetnega h končnemu vozlišču. Tedaj lahko narišemo n-sekvenčnih operacij, s katerimi ponazorimo prehode po grafu G v izbranih trenutkih. Začetna vozlišča teh grafov (sekvenčnih operacij) niso enaka, če je začetno vozlišče grafa selektorska operacija; enaka so, če je začetno vozlišče opera-

cijsko vozlišče. Podobno velja za končna vozlišča, da niso vsa enaka, če je graf zaključen s selektorsko operacijo, drugače so vsa vozlišča enaka. Na sliki 1.1 je podan splošen primer, če je število neenakih začetnih in končnih vozlišč različno. Poljubno operacijsko vozlišče Op_i je določeno z izjavo Q_i, ki jo pridružimo predhodnemu vozlišču Op_j in v bistvu določa usmerjeno vejo od Op_j k Op_i, ali s konjunkcijo $Q_i \wedge P_{ij} \wedge P_{kl} \wedge \ldots \wedge P_{rs}$, če vodi k vozlišču Op_i usmerjena pot preko več ali ene selektorskih operacij.

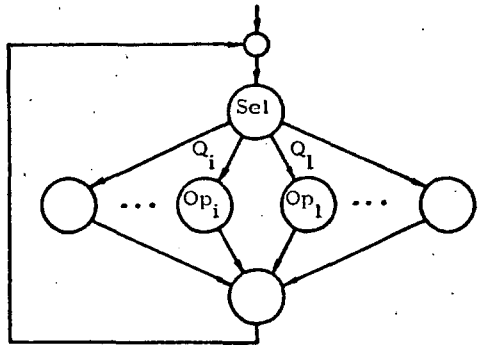


Slika 1.1: Ponazoritev usmerjenega grafa G z n-usmerjenimi grafi - sekvenčnimi operacijami

Po opisanem postopku pretvorbe se lahko nekatere operacije v vase zaključeni selektorski operaciji ponovijo tudi do n-krat, saj lahko isto vozlišče večkrat "prehodimo" po različnih usmerjenih poteh. Tedaj reduciramo veje, ki se v vase zaključeni selektorski operaciji ponavljajo takole:

$$Q_i \wedge Op_i \vee Q_i \wedge Op_i \vee \ldots \vee Q_i \wedge Op_i = Q_i \wedge Op_i \qquad (1.1)$$

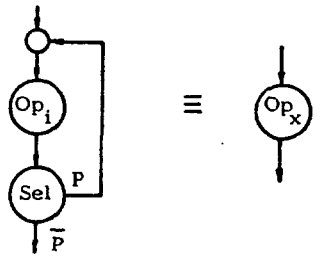
Operacije, ki se v vase zaključeni selektorski operaciji ponavljajo (1.1) nadomestimo z eno samo operacijo. Slika 1.2 podaja zgled tako preoblikovanega grafa G, ki ustreza modelu sekvenčnega stroja opisanega v [1] .



Slika 1.2: Usmerjen graf G preoblikovan v vase zaključeno selektorsko operacijo

1.1. Preoblikovanje operacij v zanki

Sedaj sprostimo pogoj, da usmerjen graf G ne sme vsebovati operacij v zanki. Obdržimo pa omejitev, da morajo imeti zanke končno mnogo ponovitev. Tvorimo nov graf, v katerem operacije v zanki ponazorimo kot vozlišča z enim vhodom in enim izhodom. Slika 1.3 podaja zgled takšne pretvorbe, kjer je lahko operacija Op_i tudi sestavljena operacija, ki vsebuje nove operacije v zanki.

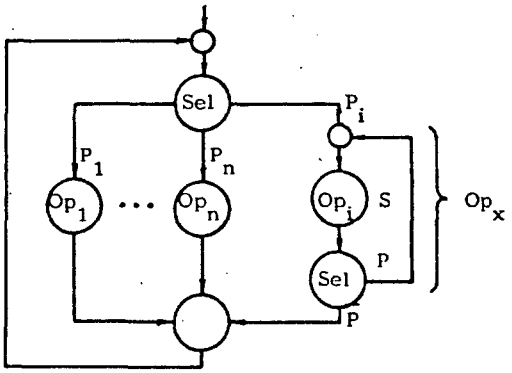


Slika 1.3: Pretvorba operacije v zanki v vozliščno operacijo

Tako dobljen graf sedaj ustreza pogojem, da ga lahko preoblikujemo v vase zaključeno selektorsko operacijo. Ugotovimo lahko, da vsaka operacija v zanki, če govorimo o njeni logični realizaciji v bistvu predstavlja avtonomen sekvenčni stroj. Pri logični realizaciji usmerjenega grafa, ki vsebuje več zank, bi tedaj morali izdelati še toliko sekvenčnih strojev, kolikor je operacij v zanki.

Takšen pristop pa je v splošnem preveč tog, zato skušamo operacije v zanki prevesti v koncept ene vase zaključene selektorske operacije oz. enega sekvenčnega stroja. Takoj pripomnimo, da končni cilj ni vedno ena sama vase zaključena selektorska operacija oz. en sekvenčni stroj, ampak da želimo imeti možnost, da sami odločamo o številu vase zaključenih selektorskih operacij oz. o organizaciji sistema.

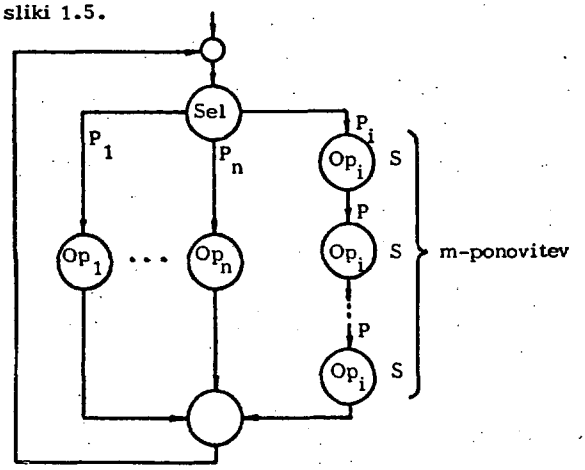
Vzemimo, da smo označen usmerjen graf G, ki vsebuje operacijo Op_i v zanki preoblikovali v vase zaključeno selektorsko operacijo, ki jo podaja slika 1.4.



Slika 1.4: Graf G preoblikovan v vase zaključeno selektorsko operacijo

Na sliki 1.4 so P_1, P_2, ..., P_n v splošnem sestavljene izjave oblike: Q_i, $Q_j \wedge P_1$, ...

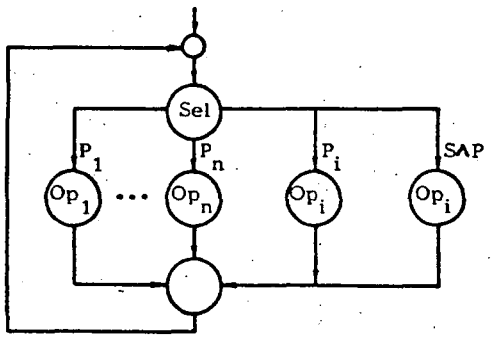
Sedaj pa opazujmo graf s slike 1.4 v trenutku, ko se operacija Op_i ponovi m-krat. Tedaj lahko narišemo graf na sliki 1.5.



Slika 1.5: Graf G v primeru, ko se operacija Op_i ponovi m-krat

Sekvenčno operacijo v grafu s slike 1.5 lahko po [1] preoblikujemo v vase zaključeno selektorsko operacijo. Graf na sliki 1.5 tedaj preide v graf na sliki 1.6. Pravkar opisano pretvorbo zančne operacije lahko verificiramo tudi s popolno indukcijo in s tem potrdimo pravilnost pretvorbe.

Za graf na sliki 1.6 izdelajmo še simbolični zapis:



Slika 1.6: Preoblikovana zančna operacija

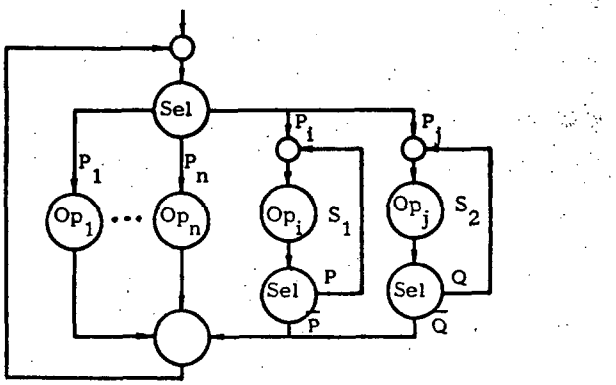
sel (S, P_j):

P_1 — Op_1
P_2 — Op_2
⋮
P_{i-1} — Op_{i-1}
P_{i+1} — Op_{i+1}
⋮
P_n — Op_n
P_i — Op_i
$S \wedge P$ — Op_i

kjer je za izjavo S (in tudi izjave Q_i, i=1, 2, ...) v praksi snovanja strojne opreme udomačen izraz stanje. Zančni operaciji Op_i smo izjavo S pripisali zato, da zagotovimo enoličnost selektorske operacije - samo ena izmed izjav P_1, P_2, ..., P_i, ... , P_n, P je lahko naenkrat pravilna.

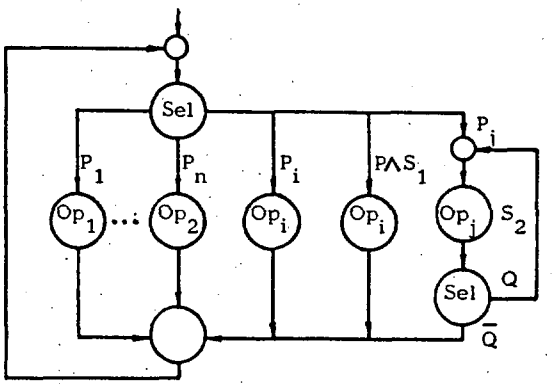
V ta namen vzemimo, da so izjave P_1, P_2,..., P_n izbrane kot konjunktivne kombinacije niza izjav p_{m-1}, p_{m-2},... ..., p_1, p_0 in velja $2^m - n \geq 1$.
Tedaj lahko za S izberemo iz preostalih konjunktivnih kombinacij iz niza p_{m-1}, p_{m-2},..., p_0.
Z zgornjim primerom smo pokazali, kako je mogoče z dodatno izjavo S ohraniti enoličnost selektorske operacije.

Če sta v usmerjenem grafu dve ali več zančnih operacij jih reduciramo postopoma - z nekaj izkušnjami pa lahko reduciramo tudi vse hkrati. Vzemimo primer, da sta nam pri preoblikovanju grafa G v vase zaključeno selektorsko operacijo preostali dve operaciji v zanki. Slika 1.7 podaja tak primer.



Slika 1.7: Preoblikovan graf G z dvema zančnima operacijama

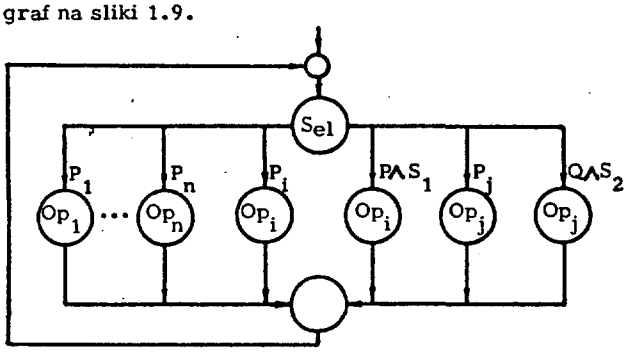
Najprej preoblikujemo zančno operacijo Op_i in pustimo zančno operacijo Op_j nedotaknjeno. Graf s slike 1.7 preide tedaj v graf na sliki 1.8.



Slika 1.8: Prvi korak preoblikovanja

Postopek ponovimo nad zančno operacijo Op_j in dobimo

graf na sliki 1.9.



Slika 1.9: Preoblikovan graf s slike 1.7

Zapišimo še selektorsko operacijo v simboličnem zapisu:

sel:

$P_1 \rightarrow Op_1$
$\vdots$
$P_{i-1} \rightarrow Op_{i-1}$
$P_{i+1} \rightarrow Op_{i+1}$
$\vdots$
$P_{j-1} \rightarrow Op_{j-1}$
$P_{j+1} \rightarrow Op_{j+1}$
$\vdots$
$P_n \rightarrow Op_n$
$P_i \rightarrow Op_i$
$S_1 \wedge P \rightarrow Op_i$
$P_j \rightarrow Op_j$
$S_2 \wedge Q \rightarrow Op_j$

Vzemimo, da so izjave $P_1, P_2, \ldots, P_i, \ldots, P_j, \ldots, P_n$ iz konjunktivnih kombinacij niza izjav:

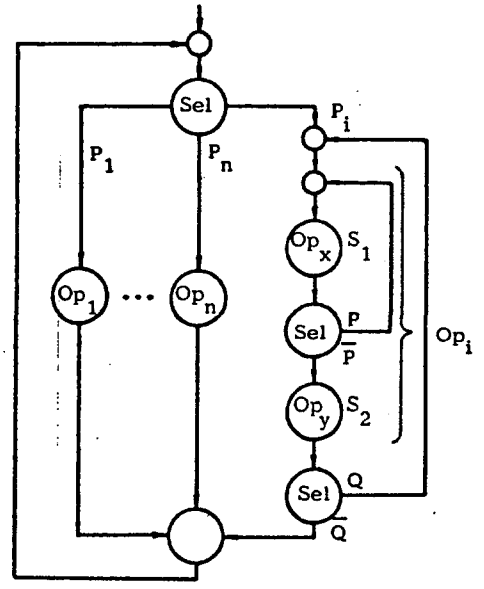
$P_{m-1}, P_{m-2}, \ldots, P_1, P_0$ in $2^m - n \geq 2$.

Tedaj lahko za S_1 in S_2 izberemo iz preostalih konjunktivnih kombinacij zgornjega niza.

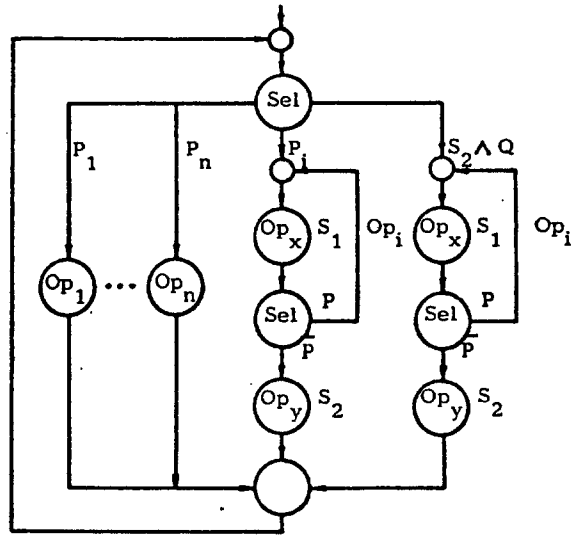
Obdelajmo še primer, ko zančna operacija v vase zaključeni selektorski operaciji tudi sama vsebuje zančno operacijo. Tako oblikovan graf G podaja slika 1.10.

Preoblikovanje grafa s slike 1.10 pričnemo tako, da si zanko določeno z izjavo Q zamislimo kot sestavljeno operacijo (Op_i). Tedaj preide graf v obliko na sliki 1.4, ki jo že znamo preoblikovati. Če opravimo postopek preoblikovanja, preide graf s slike 1.10 v obliko na sliki 1.11.

Ponovno si zamislimo zančni operaciji Op_x kot operaciji Op_i in dobimo graf, v katerem sta po dve sekvenčno povezani operaciji, ki ju lahko preoblikujemo po [1] v vase zaključeno selektorsko operacijo. S tem dobimo graf na sliki 1.12.

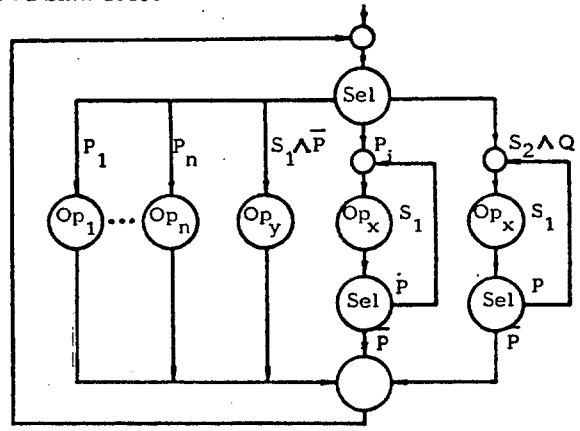


Slika 1.10: Graf G z vgnezdeno zančno operacijo

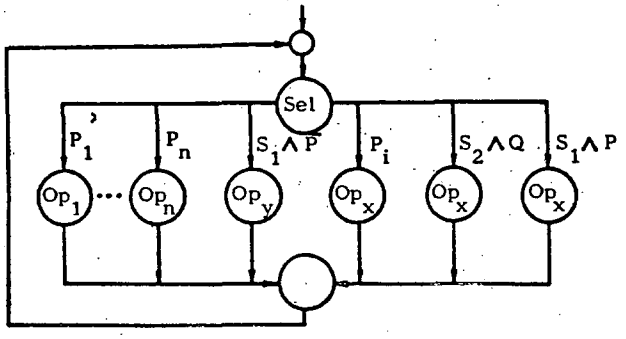


Slika 1.11: Prvi korak pri preoblikovanju vgnezdene zanke

Graf na sliki 1.12 pa je s stališča preoblikovanja zančnih operacij enak grafu na sliki 1.7, zato lahko narišemo kar končno obliko grafa vase zaključene selektorske operacije na sliki 1.13.



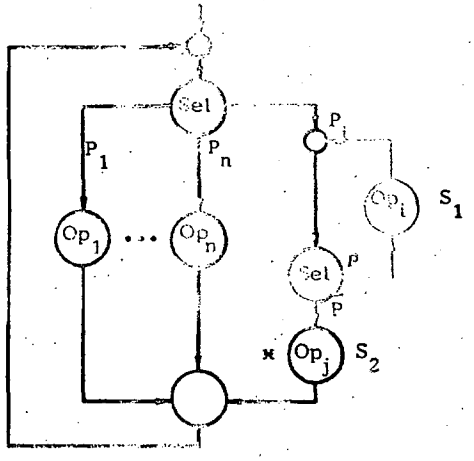
Slika 1.12: Drugi korak preoblikovanja vgnezdene zanke



Slika 1.13: Preoblikovan graf vgnezdene zanke

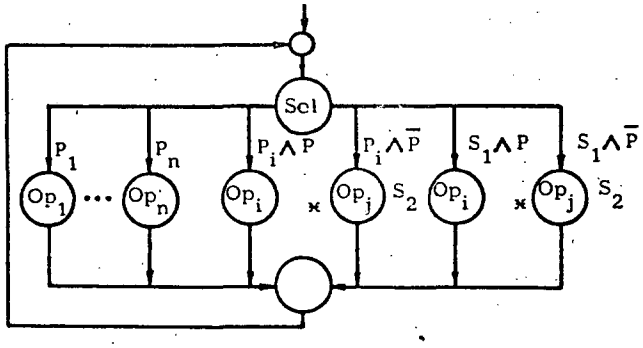
Posplošitev opisane transformacije na n-vgnezdenih zank je enostavna in lahko shajamo z doslej definiranimi transformacijami.

Opravimo pa še preoblikovanje grafa s slike 1.14.



Slika 1.14: Graf G z operacijo v povratni veji zanke

Grafu G smo kos s smiselno uporabo doslej opisanih transformacij. Zato lahko narišemo kar graf vase zaključene selektorske operacije na sliki 1.15.



Slika 1.15: Graf vase zaključene selektorske operacije za graf s slike 1.14

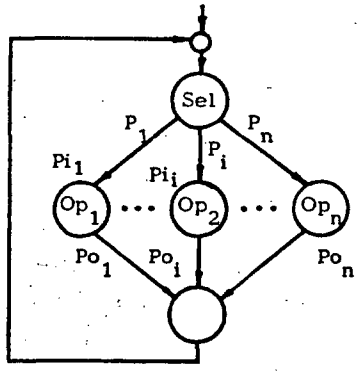
Z zvezdico označeno operacijo Op_j smo vrisali zato, ker lahko pogosto izvorni graf G preoblikujemo tako kot na sliki 1.14. Sicer bi morali vpeljati namesto vozlišča Op_j na sliki 1.15 vozlišče O_n, ki izvaja "prazno" operacijo.

Pri fizičnem snovanju moramo zagotoviti tudi inicializacijo operacij v naših grafih. To dosežemo s tem, da v vhodno vozlišče "pripeljemo" izjavo, ki izbere začetno vejo v vase zaključeni selektorski operaciji - sistem postavimo v začetno stanje.

2. DEKOMPOZICIJA VASE ZAKLJUČENE SELEKTORSKE OPERACIJE - SEKVENČNEGA STROJA; NAVZDOLNJE SNOVANJE

V tem razdelku se omejimo na dekompozicijo kompleksnih vase zaključenih selektorskih operacij - sekvenčnih strojev. Pri enostavnih sekvenčnih strojih lahko neposredno z uporabo transformacij definiranih v [1] in v 1. razdelku preoblikujemo graf sekvenčnega stroja v želeno obliko.

Izhajajmo iz vase zaključene selektorske operacije na sliki 2.1.



Slika 2.1: Izhodiščni graf za dekompozicijo

Dekompozicijo takšnega grafa lahko pričnemo, če imamo na voljo specifikacijo o operacijah, stanjih in pogojih, ki morajo biti izpolnjeni, da se operacije lahko izvedejo.

V splošnem lahko isti graf ob upoštevanju njegovega opisa (specifikacije) razgradimo na množico različnih načinov. Kako bo potekala razgradnja in kdaj bo postopek končan je odvisno od lastnosti operacij in zunanjih zahtev, ki smo jih postavili kot cilje snovanja.

Omejimo se na en sam primer dekompozicije sekvenčnega stroja, ki pa bo pretežno ilustriral karakteristike dekompozicije sekvenčnih strojev. Pri tem bomo uporabili poenostavljen zapis, saj bi bil formaliziran zapis preobsežen.

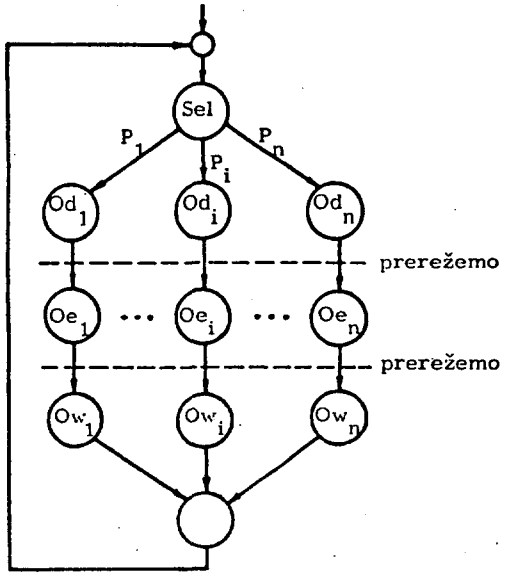
Izhajajmo iz naloge, da je potrebno izdelati procesor, ki izvaja instrukcije kompleksne računalniške arhitekture (CISC), katere opis obsega nekaj sto strani.
Že sam zapis selektorske operacije za tako instrukcijsko množico v smislu slike 2.1 bi bil toliko nepregleden, da iz njega ne bi kaj dosti razbrali. Zato uporabimo nekoli-

ko splošnejši pristop. Iz specifikacije naše instrukcijske množice ugotovimo, da je splošna zgradba instrukcije npr. takšna:

code src 1. rx src 2. rx ... src m.rx dst.wx (2.1)

V 2.1 je code ime operacije, iz katerega razberemo tudi tipe operandov, src j . rx so določila operandov ali kar operandi in dst . wx določilo destinacijskega operanda. Naša specifikacija dovoljuje, da lahko oblika 2.1 degenerira v code, v code z izvornimi operandi brez destinacijskega operanda ali code s samo destinacijskim operandom. Pri izvajanju vsake instrukcije se postavi še niz izjav, ki v splošnem lahko vplivajo na izbiro naslednje instrukcije za izvajanje. Značilna lastnost instrukcijske množice naj bo, da instrukcije i+1 ni možno pričeti dekodirati, dokler ni v celoti dekodirana instrukcija i, ali dokler ni v celoti dekodirana instrukcija i in izvedena zahtevana operacija, ki postavi niz izjav te instrukcije. To lastnost lahko ugotovimo iz imena operacije v instrukciji.

Ta opis nam zaenkrat zadošča, da lahko pričnemo z dekompozicijo. Izhajajmo iz slike 2.1 in predpostavimo, da imajo vse instrukcije splošno zgradbo (2.1). Tedaj lahko obdelavo poljubne instrukcije ponazorimo s tremi sekvenčno povezanimi operacijami in sicer z operacijo dekodiranja instrukcije Od_i, operacijo izvajanja Oe_i in operacijo vpisa destinacijskega operanda Ow_i. Eksplicitna definicija operacij nas ne zanima in jo lahko za konkretno instrukcijsko množico izpeljemo s pomočjo [1] in [2] . Model iz katerega smo izhajali na sliki 2.1 lahko sedaj preoblikujemo tako kot podaja slika 2.2.



Slika 2.2: Začetni korak dekompozicije

Odstopanja od idealizirane zgradbe instrukcij (2.1) bomo upoštevali kasneje. Takšen pristop nam omogoča, da se v določenih fazah snovanja osredotočimo na karakteristične lastnosti instrukcijske množice in prilagodimo izgradnjo modela tistim lastnostim arhitekture, ki so na določenem nivoju snovanja dominantne. Na ta način imamo ves čas pred seboj bistvo problema in se ne izgubljamo v detaljih. Seveda pa mora končni model naše arhitekture verno posnemati vse njene lastnosti. Lahko se zgodi, da pri takšnem pristopu "vidimo" tudi lastnosti, ki jih "ni" oz. tiste ki niso dovolj karakteristične, da bi jih upoštevali že na tekočem nivoju snovanja, ampak šele kasneje. Skrajna možna posledica je lahko, da je rezultat snovanja nezadovoljiv in je potrebno postopek snovanja ponoviti.

Da minimiziramo subjektivno komponento pri ugotavljanju karakterističnih lastnosti lahko koristimo razmeroma obsežen matematični aparat za analizo sistemov.

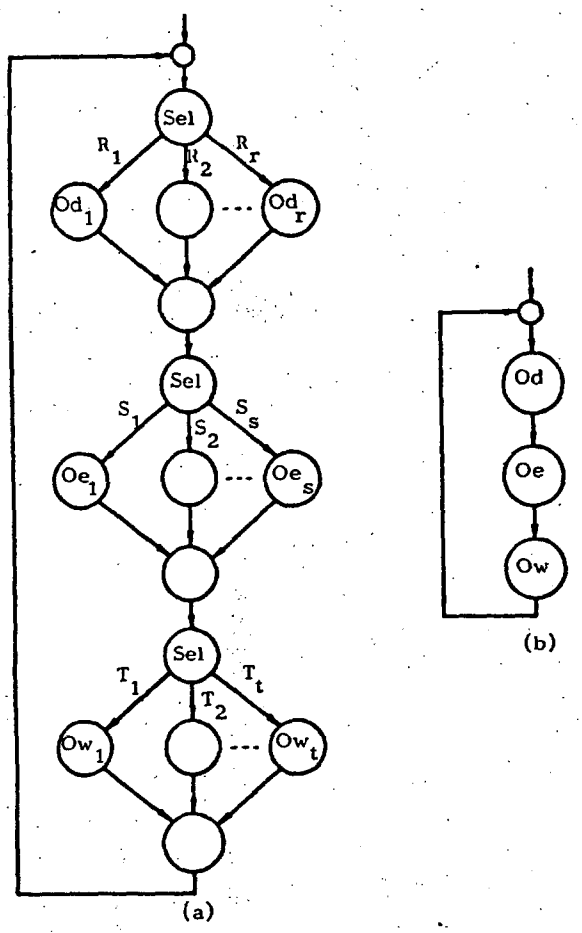
Graf s slike 2.2 lahko ponazorimo kot tri sekvenčno povezane vase zaključene selektorske operacije. Upoštevamo še, da lahko instrukcije, ki imajo enako operacijo Od_i, "enako" operacijo Oe_j (odvisno od organizacije operacijske enote) in enako operacijo Ow_l nadomestimo z eno samo operacijo, če ustrezno prilagodimo izjave $P_1,\ldots,P_n$ v vseh treh novih selektorskih operacijah. Tako dobimo krmilni model na sliki 2.3.

Na sliki 2.3 (b) je podan zgoščen graf za sliko 2.3 (a). 2.3 (b) nam bo omogočal dovolj zgoščen zapis, zato nadaljujemo snovanje s tako oblikovanim grafom. Poiskusimo sedaj združiti operaciji Od in Ow v eno samo sestavljeno operacijo. V ta namen definirajmo izjavo W-past, ki trdi, da je rezultat operacije Oe_j, j = 1,2,..., s določen. Ko postane izjava W-past pravilna se lahko sproži izvajanje operacije Ow.

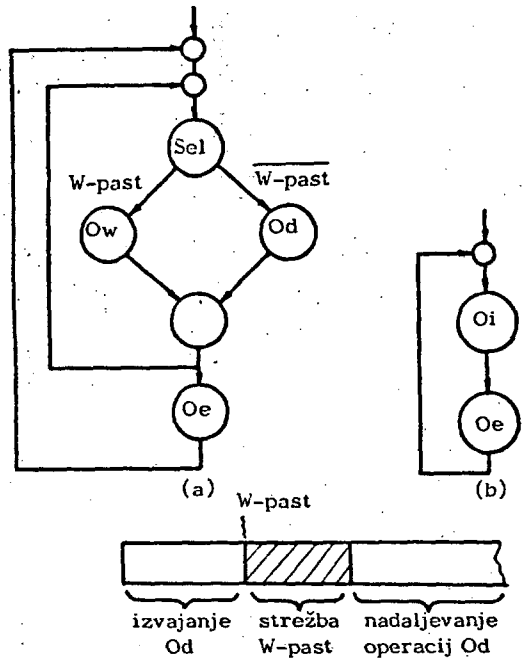
Graf s slike 2.3 (b) preide tako v obliko na sliki 2.4.

S tem smo dobili dve sekvenčno povezani sestavljeni operaciji Oi in Oe. Pri tem je Oi že v svoji zanki, kar pomeni, da jo bomo realizirali kot sekvenčni stroj. Tudi operacijo Oe bomo realizirali kot sekvenčni stroj, saj so operacije, definirane s kompleksno instrukcijsko množico toliko kompleksne, da zahtevajo pri logični realizaciji dodatno interpretacijo z enostavnejšimi operacijami.

Graf na sliki 2.4 (b) lahko preoblikujemo v paralelno operacijo, če dovolimo, da se ob izvajanju operacije Oe instrukcije i paralelno dekodira instrukcija i+1, torej izva-



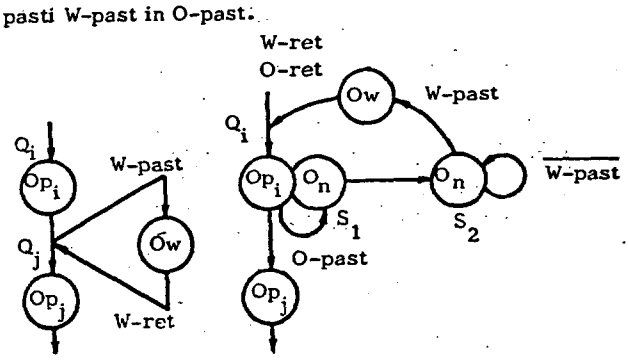
Slika 2.3: Ponazoritev s tremi sekvenčno povezanimi selektorskimi operacijami



Slika 2.4: Združitev operacij Ow in Od v sestavljeno operacijo Oi

janje operacije Oi. Takšna paralelna operacija je možna, če lahko po končanem dekodiranju instrukcije i pričnemo z dekodiranjem instrukcije i+1. Dekodiranje instrukcije i+1 se lahko izvede v celoti, če ni odvisnosti med izvornimi operandi instrukcije i+1 in destinacijskim operandom instrukcije i. Sicer je potrebno ob ugotovljeni odvisnosti med operandi, prekiniti izvajanje dekodiranja instrukcije i+1 ter počakati, da nastopi W-past, ustreči W-past, ter nadaljevati pri prekinjenem dekodiranju. Izjavo s katero ugotovimo odvisnost poimenujmo z O-past in jo definirajmo takole: O-past postane pravilna, če nastopi odvisnost med izvornimi operandi instrukcije i+1 in destinacijskim operandom instrukcije i, za katero se izvaja operacija Oe.

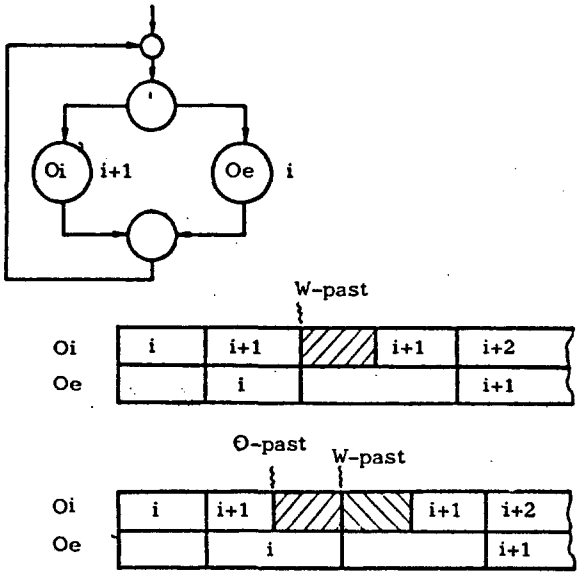
Na sliki 2.5 sta podana grafa, ki ponazarjata strežbo pasti W-past in O-past.



Sl. 2.5: Strežba W-pasti in O-pasti

Iz slike 2.5 razberemo, da se strežba W-pasti vrine med izvajanje operacij Op_i in Op_j (Od) in da se po končani strežbi izvaja operacija Op_j, ki je določena z izjavo Q_j, ki jo mora stroj pomniti, da lahko nadaljuje izvajanje. Pri strežbi O-pasti pa se operacija Op_i med izvajanjem spremeni v operacijo On, ki izvede "prazno" operacijo, nato nastopi čakanje na W-past in po končani strežbi W-pasti se ponovi izvajanje operacije Op_i (Od). Slika 2.6 podaja paralelno vase zaključeno operacijo Oi ' Oe. Uvedimo še termina za Oi in Oe za fizično izvedbo in sicer instrukcijski procesor za Oi ter operacijski procesor za Oe.

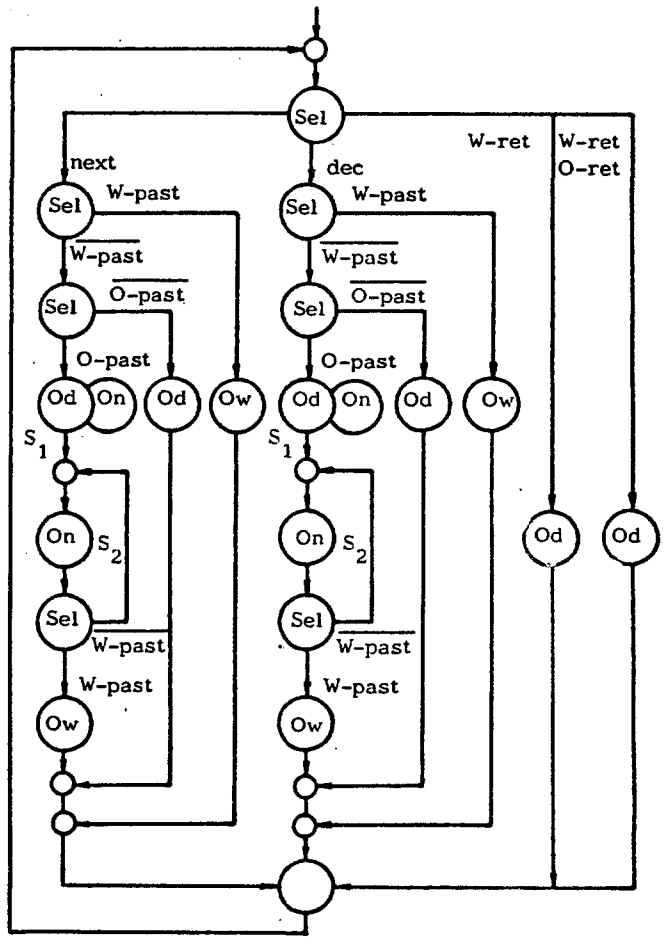
Sedaj imamo na voljo parametre za pričetek definicije krmilnih grafov za instrukcijski in operacijski procesor. Na sliki 2.7 je podan krmilni graf za instrukcijski procesor oz. za sestavljeno operacijo Oi s slike 2.6. Pri tem smo vpeljali še izjavi next in dec, ki omogočata - prva interpretacijo operacij dekodiranja in druga dekodiranje instrukcijskega niza. Nabor potrebnih krmilnih izjav s tem sicer še ni zaključen, vendar se bomo s takim krmilnim modelom zadovoljili in ga bomo prevedli na izhodiščno organizacijsko shemo instrukcijskega procesorja. Zato najprej preoblikujemo graf s slike 2.7 v graf vase zaključene selektorske operacije, tako kot jo podaja slika 2.8. Za preoblikovanje smo uporabili postopke



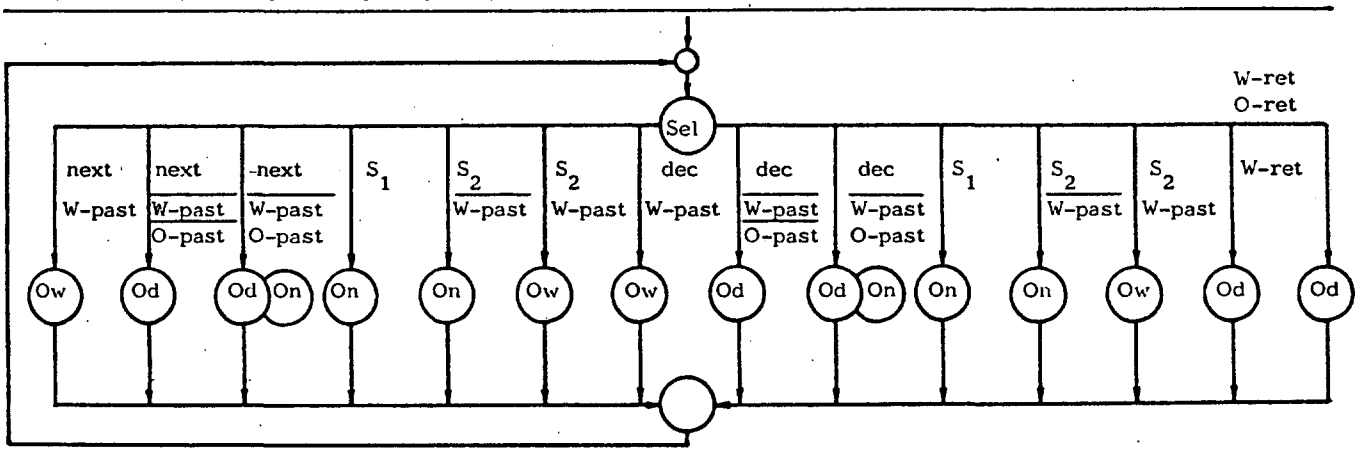
Sl. 2.6: Paralelni model izvajanja operacij

opisane v [1] in v 1. razdelku. V naslednjem koraku pa sliko 2.8 prevedemo v mikroprogramsko krmiljen model sekvenčnega stroja po sliki 2.9. Tako dobljeni model še ne izpolnjuje specifikacije naše kompleksne instrukcijske množice.

Zato specificiramo, da se instrukcije, katerih pričetek dekodiranja je odvisen od postavljenih izjav instrukcije v operacijskem procesorju, v celoti dekodirajo in izvedejo v instrukcijskem procesorju. Pri instrukcijah, ki nimajo destinacijskega operanda pa W-past sproži v in-

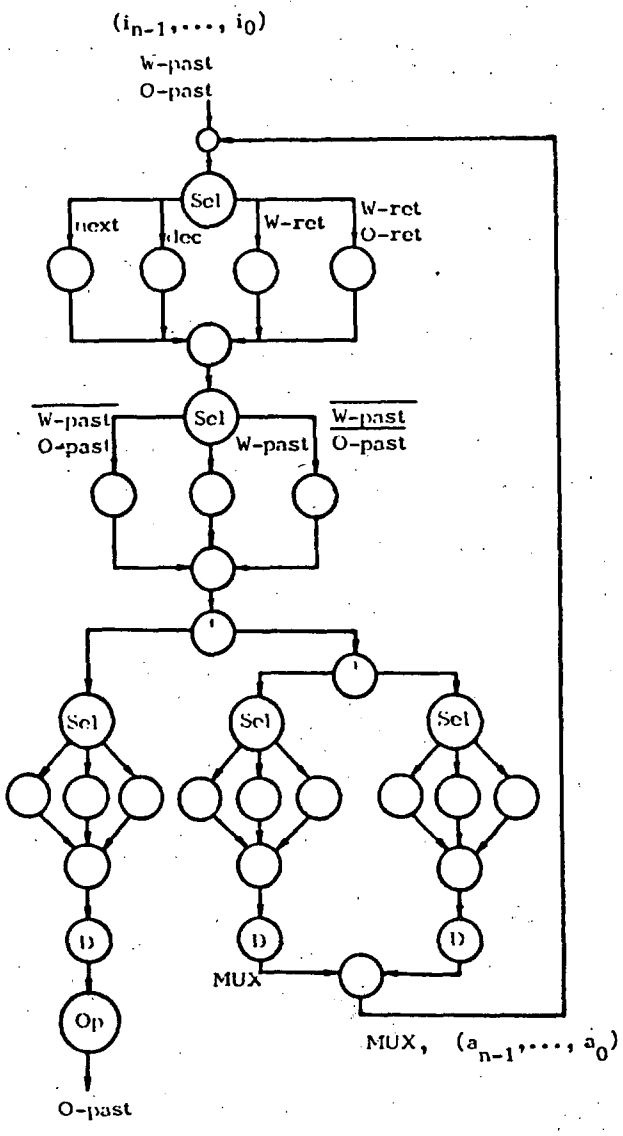


Slika 2.7: Začetni približek k realnemu krmilnemu modelu instrukcijskega procesorja

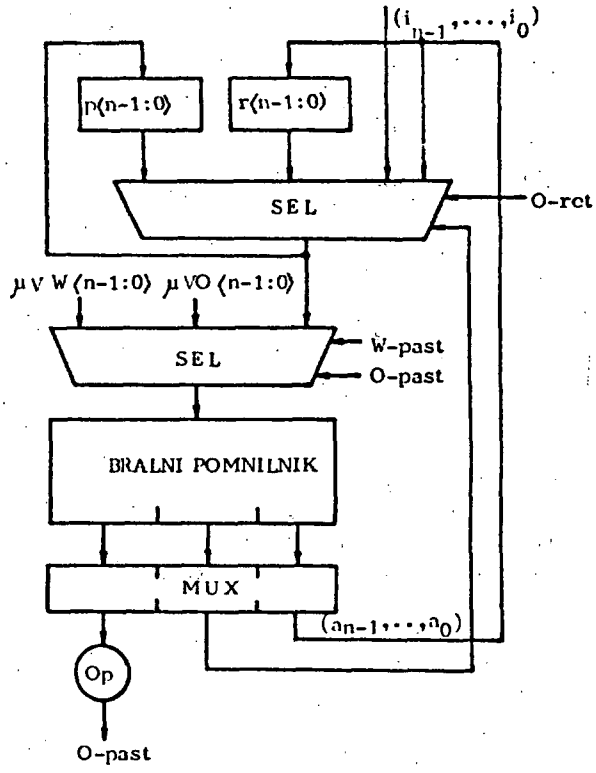


S_1 in S_2 lahko definiramo kot izjavi poljubno izbrani iz niza možnih konjunktivnih kombinacij izjav $a_{n-1}, \ldots, a_0$. next izjava določa niz $a_{n-1}, \ldots, a_0$ kot vir, ki določa naslednjo operacijo. Izjava dec določa niz $i_{n-1}, \ldots, i_0$ kot vir, ki določa operacije pri dekodiranju nove instrukcije. Izjava W-ret določa niz $r_{n-1}, \ldots, r_0$, kot vir, ki določa naslednjo operacijo ter W-ret ob aktivni izjavi O-ret niz $p_{n-1}, \ldots, p_o$ kot vir, ki določa novo operacijo. Izjava W-past izbira niz $\mu VW\langle n-1:0\rangle$ kot vir, ki določa naslenjo operacijo, ter izjava O-past niz $\mu V O\langle n-1:0\rangle$ kot vir, ki določa naslednjo operacijo. Definiramo še selektorsko operacijo MUX, ki specificira eno izmed izjav next, dec, W-ret.

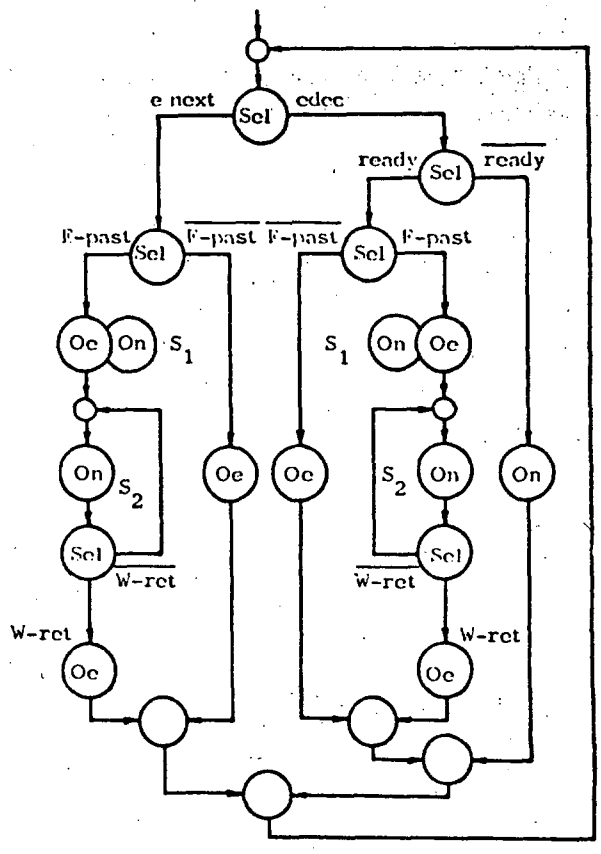
Slika 2.8: Vase zaključena selektorska operacija za 2.7



Slika 2.9: Mikroprogramsko krmiljen model instrukcijskega procesorja - začetni približek



Slika 2.10: Začetni približek k blokovni shemi instrukcijskega procesorja



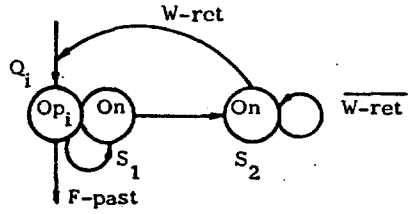
Slika 2.11: Začetni krmilni model operacijskega procesorja

strukcijskem procesorju izvajanje operacije On.

S tem zaključimo snovanje modela instrukcijskega procesorja; doslej dobljeno blokovno shemo podaja slika 2.10.

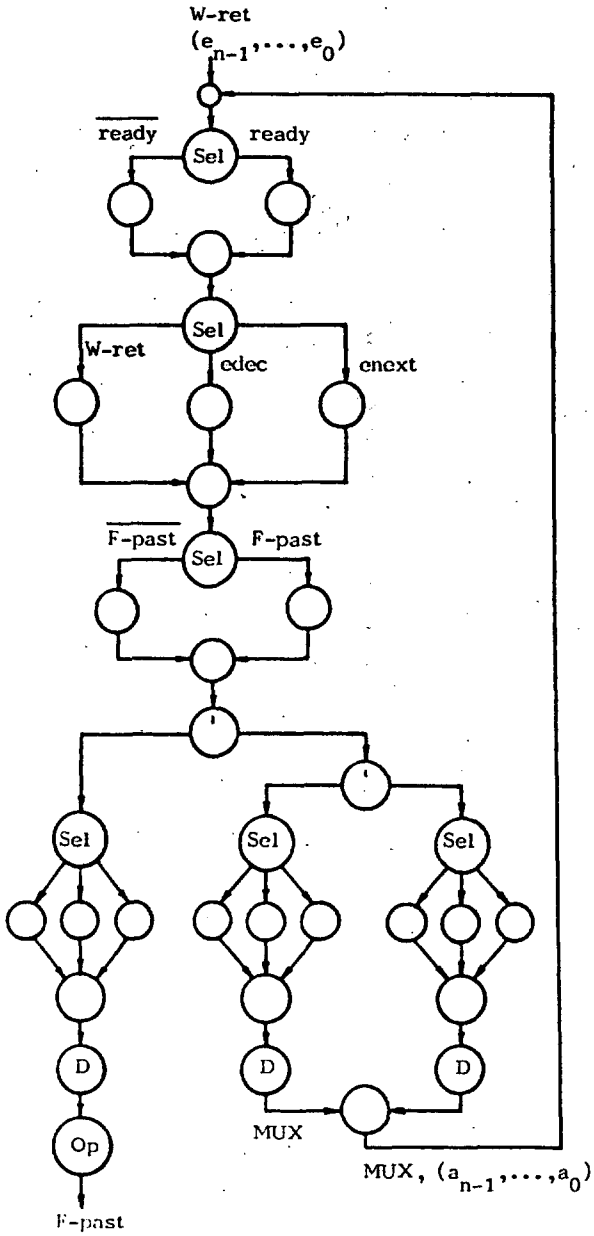
Podoben postopek opravimo pri snovanju začetnega modela operacijskega procesorja. Operacija Oe s slike 2.5 tako preide v sestavljeno operacijo, katere graf podaja slika 2.11. Pri tem je F-past izjava, ki pove, da se v operacijskem procesorju izvaja operacija, ki bo zahtevala strežbo W-pasti v instrukcijskem procesorju, hkrati pa v le-tem še teče strežba prejšnje W-pasti. Zato je potrebno počakati na zaključek strežbe W-pasti in nato ponoviti mikrooperacijo, ki je bila prekinjena ob strežbi W-pasti. Graf na sliki 2.12 podaja strežbo F-pasti. Izjava enext specificira izvajanje naslednje mikrooperacije, izjava edec pa pričetek izvajanja

novega mikroprograma, ki interpretira operacijo Oe_j, katero je določil instrukcijski procesor, če je tudi ready izjava, ki trdi da so vsi parametri za strežbo na voljo, pravilna.



Slika 2.12: Graf strežbe F-pasti

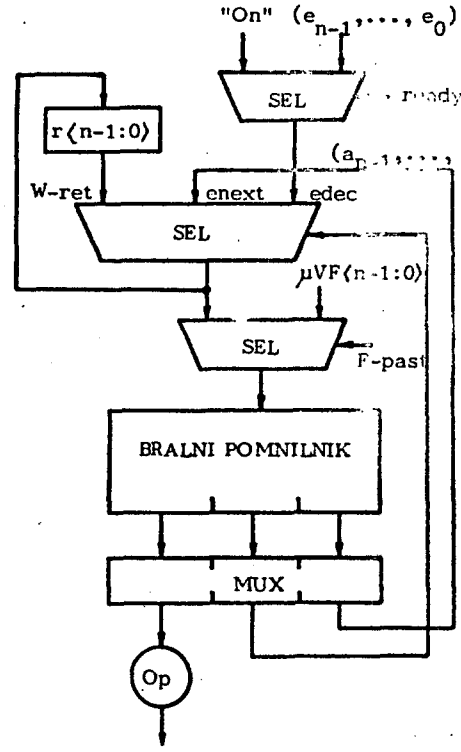
Graf s slike 2.11 po opisanih postopkih preoblikujemo v mikroprogramski model sekvenčnega stroja, katerega graf podaja slika 2.13. Začetno blokovno shemo operacijskega procesorja, dobljeno iz tega grafa, pa podaja slika 2.14.



Slika 2.13: Mikroprogramski model operacijskega procesorja - začetni približek

Doslej smo podali le pričetek dekompozicije sekvenčnega stroja (slika 2.1), s katerim smo pričeli razgradnjo. S smiselno uporabo doslej opisanih postopkov lahko krmilno strukturo obeh procesorjev razvijemo do potrebnih detaljev.

Čeprav smo se omejili na en sam primer dekompozicije, so v splošnem postopki razgradnje kompleksnih sekvenčnih strojev podobni doslej opisanim. Preostane nam še izgradnja modelov operacijskih enot za takšne stroje.

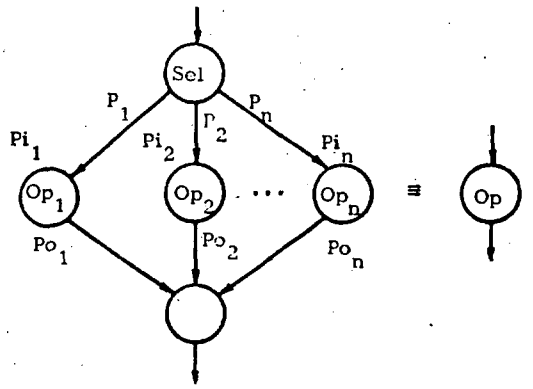


Slika 2.14: Blokovna shema operacijskega procesorja - začetni približek

3. MODELI OPERACIJSKIH ENOT

Predpostavimo, da smo za izbrano instrukcijsko množico razvili po postopkih [2] algoritme v simboličnem zapisu ali v poljubnem formalnem jeziku, ki opisujejo operacije določene z instrukcijsko množico na abstraktnem nivoju, primernem za logično realizacijo. Iz tako pripravljenih algoritmov lahko razberemo, katere operacije se izvajajo, katere komponente stanj moramo pomniti, katere komponente stanj dobimo iz instrukcijskega niza oz. zunanjega pomnilnika, katere izjave povzročijo vejitve itd. Skratka na voljo imamo vse podatke, da lahko v celoti realiziramo tako krmilno kot operacijsko strukturo sekvenčnega stroja oz. krmilne in operacijske strukture sekvenčnih strojev, odvisno od tega kako smo defi-

nirali organizacijo sistema. Načeloma lahko pričnemo z izgradnjo modela operacijske enote iz različnih izhodišč. V našem primeru pa definirajmo operacijsko enoto kot selektorsko operacijo, ki odvisno od pravilnosti izjav P_1, P_2,...,P_n izvede pripadajočo operacijo nad začetnim stanjem in producira končno stanje za to operacijo. Model operacijske enote lahko tedaj izgradimo iz selektorske operacije, katere graf podaja slika 3.1.
Izpolnjenost začetnih pogojev Pi_j, j = 1, 2, ..., n lahko ugotavljamo med izvajanjem operacij in sprožimo pasti, če ti pogoji niso izpolnjeni, ali pa smo ugotavljanje izpolnjenosti teh pogojev že vgradili v krmilne algoritme in jih ugotavljamo s pomočjo vejitev.

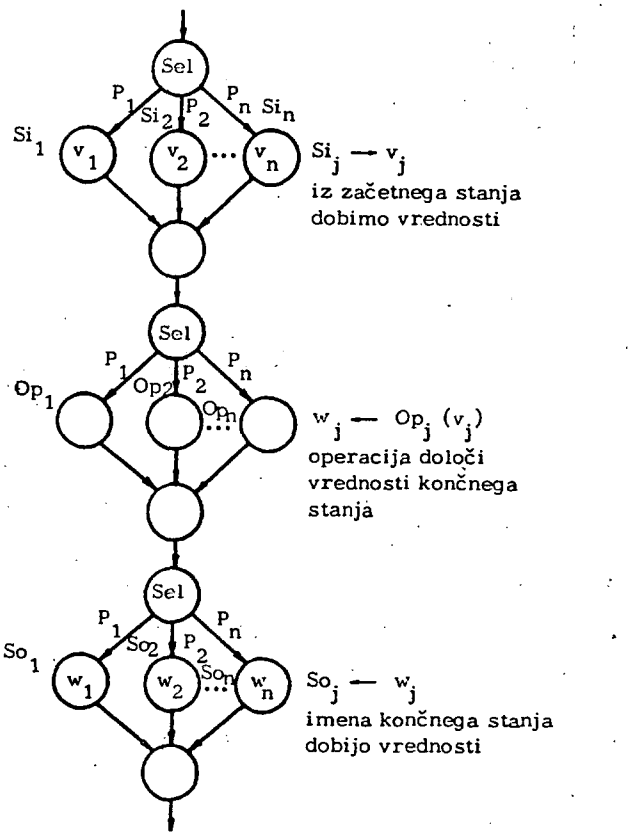


Slika 3.1: Začetni približek k modelu operacijske enote.

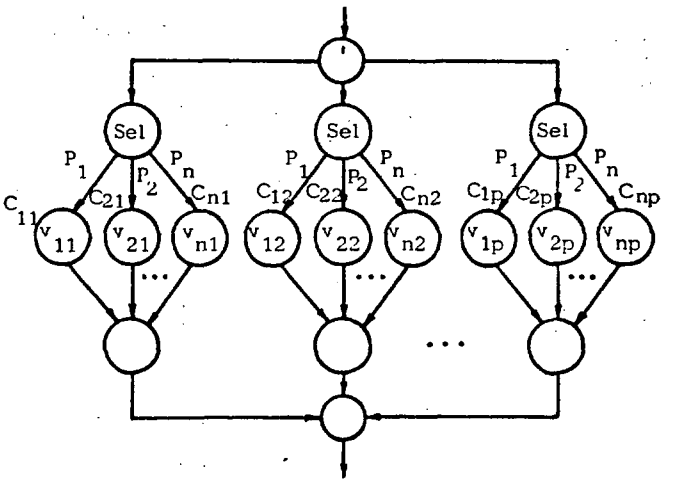
V praksi pogosto uporabljamo oba načina. V našem primeru bomo smatrali $Pi_j = 1$, j = 1, 2,..., n. V splošnem lahko rečemo, da glede na pravilno P_j, j = 1, 2,... ..., n najprej izberemo tej izjavi pripadajoče začetno stanje Si_j, nato aktiviramo izbrano operacijsko enoto, ki izvede operacijo Op_j, dobljene vrednosti pa priredimo imenom končnega stanja So_j. Graf s slike 3.1 preide tedaj v obliko na sliki 3.2.
V praksi so pogosto končna stanja nekaterih operacij ali njihove komponente hkrati tudi začetna stanja ali komponente operacij, ki se bodo šele izvršile. V ta namen predpostavimo, da imajo začetna stanja do p komponent in ustrezno preoblikujmo prvo selektorsko operacijo s slike 3.2 v paralelno selektorsko operacijo nad komponentami stanj po sliki 3.3.

Pri tem dovolimo, da so lahko komponente stanj prazne - $C_{jk} = 0$, $j \in \{1, 2,...,n\}$, $k \in \{1, 2,..., p\}$.

Za končna stanja So_j, j = 1, 2, ..., n predpostavimo, da imajo do r komponent. B_{jk}, $j \in \{1, 2, ..., n\}$, $k \in \{1, 2, ..., r\}$. Slika 3.4 podaja graf paralelne selektorske operacije, ki vrednosti komponent priredi imenom končnega stanja So_j. Tudi tukaj dovolimo, da se lahko izvedejo prazne prireditve $B_{jk} = 0$.

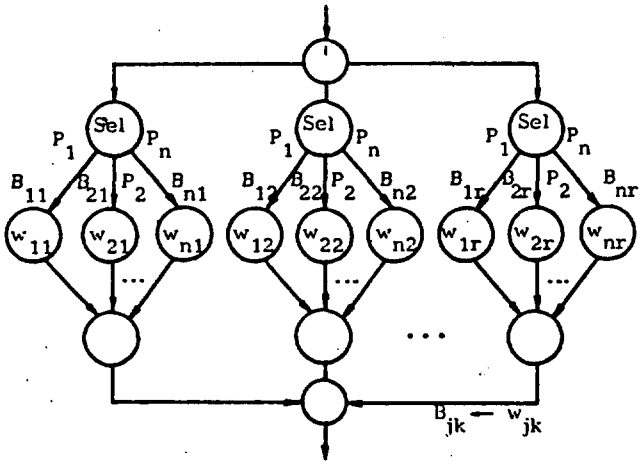


Slika 3.2: Model delovanja operacijske enote



Slika 3.3: Odjem vrednosti stanj Si_j po komponentah

Sedaj pa upoštevajmo, da so lahko končne komponente tudi začetne komponente novih operacij. V ta namen definirajmo tabelarično prireditev imen komponent C_{jk} in B_{jk} novim imenom C_1, C_2,..., C_m, tako da bo v tabeli vsaka komponenta, ki je hkrati začetna in končna imela eno samo ime. Nad preimenovanimi komponentami definirajmo novo paralelno selektorsko operacijo, pri kateri načeloma dovolimo, da lahko vsako vrednost komponente



Slika 3.4: Pripis vrednosti stanju So_j po komponentah stanja odjemamo ali priredimo (beremo oz. vpišemo).

Izjave R_i, i = 1, 2,..., p bodo določale odjem, izjave $\overline{R}_j$, j = 1, 2,..., r pa prireditev vrednosti. Izjave Q_{jk}, j = 1, 2,..., p oz r in k = 1, 2, ..., m bomo v konjunkciji z izjavami R_i oz. $\overline{R}_j$ uporabili za izbiro vozlišč katerim bomo odjemali vrednosti v_z, z = 1, 2,..., m oz. priredili vrednosti w_l, l = 1, 2,...,r. Za opis tako definirane sestavljene paralelne selektorske operacije uporabimo simbolični zapis.

$$
\begin{array}{lll}
\text{Sel:} & \text{Sel:} & \text{Sel:} \\
R_1 \text{ — sel:} & R_2 \text{ — sel:} & R_p \text{ — sel:} \\
\quad Q_{11} \text{ — } v_1 & \quad Q_{21} \text{ — } v_1 & \quad Q_{p1} \text{ — } v_1 \\
\quad Q_{12} \text{ — } v_2 & \quad Q_{22} \text{ — } v_2 & \quad Q_{p2} \text{ — } v_2 \\
\quad \vdots & \quad \vdots \quad \cdots & \quad \vdots \\
\quad Q_{1m} \text{ — } v_m & \quad Q_{2m} \text{ — } v_m & \quad Q_{pm} \text{ — } v_m
\end{array}
$$

$$
\begin{array}{ll}
\overline{R}_1 \text{ — sel:} & \overline{R}_r \text{ — sel:} \\
\quad Q_{11} \text{ — } (C_1) \text{ — } w_1 & \quad Q_{r1} \text{ — } (C_1) \text{ — } w_r \\
\quad Q_{12} \text{ — } (C_2) \text{ — } w_1 & \quad Q_{r2} \text{ — } (C_2) \text{ — } w_r \\
\quad \vdots \quad \cdots & \quad \vdots \\
\quad Q_{1m} \text{ — } (C_m) \text{ — } w_1 & \quad Q_{rm} \text{ — } (C_m) \text{ — } w_r
\end{array}
\qquad (3.1)
$$

Sestavljena paralelna selektorska operacija je definirana tako, da lahko "preberemo" poljubno permutacijo p vrednosti in "vpišemo" poljubno permutacijo do r vrednosti. Na ta način lahko dobimo vrednosti poljubnega začetnega stanja Si_j in priredimo vrednosti poljubnemu končnemu stanju So_j.

V praksi običajno ne potrebujemo tako splošne organizacije "pomnilnega prostora", zato lahko pri realizaciji konkretne naloge sestavljeno operacijo (3.1) primerno poenostavimo.

3.1. Model pomnilne celice

Izhajajmo iz selektorske operacije:

Sel (V):

$$
\begin{aligned}
V &\text{ — } P(Q, Q) \\
\overline{V} &\text{ — } P(Q, D),
\end{aligned}
\qquad (3.2)
$$

kjer je P relacija zamenjave vrednosti Q z vrednostjo Q ali vrednosti Q z vrednostjo D. (3.2) preoblikujemo v disjunktivno obliko:

$$V \wedge P(Q,Q) \vee \overline{V} \wedge P(Q, D) \qquad (3.3)$$

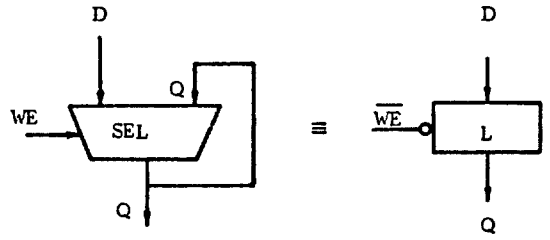
Modelirajmo V, Q, D z izjavami po [1] takole:
WE izjava, ki bo ekvivalentna izjavi V, $Q_{n-1},\ldots,Q_o$ niz izjav, s katerimi modeliramo Q, ter $D_{n-1},\ldots,D_o$ niz izjav, s katerimi modeliramo D. Relacijo zamenjave pa ponazorimo z logično ekvivalenco.

$$
\begin{aligned}
&WE \wedge (Q_{n-1} = Q_{n-1}) \vee \overline{WE} \wedge (Q_{n-1} = D_{n-1}) \\
&WE \wedge (Q_{n-2} = Q_{n-2}) \vee \overline{WE} \wedge (Q_{n-2} = D_{n-2}) \\
&\qquad\qquad \vdots \\
&WE \wedge (Q_o = Q_o) \quad \vee \overline{WE} \wedge (Q_o = D_o)
\end{aligned}
\qquad (3.4)
$$

Izraz (3.4) poenostavimo in dobimo:

$$
\begin{aligned}
Q_{n-1} &= Q_{n-1} \wedge WE \vee D_{n-1} \wedge \overline{WE} \\
Q_{n-2} &= Q_{n-2} \wedge WE \vee D_{n-2} \wedge \overline{WE} \\
&\vdots \\
Q_o &= Q_o \wedge WE \vee D_o \wedge \overline{WE}
\end{aligned}
\qquad (3.5)
$$

(3.5) imenujmo relacije pomnjenja,za pomnilno celico ki jo te relacije opisujejo pa uporabimo simbol na sliki 3.5.



Slika 3.5: Simbol pomnilne celice tipa zapah

3.2. Model pomnilnika s serijskim odjemom in serijsko prireditvijo

Da ne bi ponovno v celoti izvajali vseh relacij tako kot pri pomnilni celici, uporabimo nekoliko poenostavljen zapis. Z D označimo vrednost, ki je na vhodu pomnilnika, z D_n, $D_{n-1},\ldots, D_1$ označimo vrednosti v pomnilniku (pomnilnih celicah) z Q pa označimo vrednost na izhodu pomnilnika. Z A_i, i = 1, 2,..., n označimo poljubne

konjunktivne kombinacije niza izjav (a_{m-1}, a_{m-2},... ..., a_o), kjer velja $2^m \geq n$.

Z izjavo $\bar{R}$ pa bomo zahtevali prireditve vrednosti, z izjavo R pa odjem vrednosti.

Sedaj lahko definiramo selektorsko operacijo:

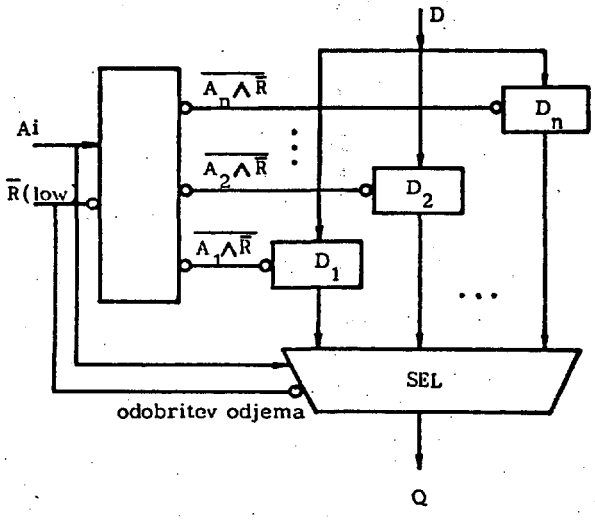
Sel:

A_1 — sel:
R — P (Q, D_1)
$\bar{R}$ — P (D_1, D)

A_2 — sel:
R — P (Q, D_2)
$\bar{R}$ — P (D_2, D) (3.6)
⋮

A_n — sel:
R — P (Q, D_n)
$\bar{R}$ — P (D_n, D)

Če upoštevamo model pomnilne celice in simbolični zapis (3.6), lahko kar narišemo eno izmed možnih blokovnih shem takega pomnilnika na sliki 3.6.



Slika 3.6: Blokovna shema serijsko organiziranega pomnilnika

3.3. Model paralelnega pomnilnika

Model serijskega pomnilnika posplošimo tako, da bo možno paralelno branje p vrednosti in paralelni vpis do r vrednosti ter s tem logično realizacijo sestavljene paralelne selektorske operacije (3.1). Predpostavimo $p \geq r$ in definirajmo ADR_{1i}, i = 1, 2,..., m,..., ADR_{ri},..., ADR_{pi}, ter so ADR_{1i},..., ADR_{pj}, i in j = 1, 2, ..., m, poljubne konjunkcije izjav a_{n-1},..., a_o ob pogoju da, če je i = j imamo opraviti z eno in isto konjunktivno kombinacijo izjav a_{n-1},..., a_o. D_r, D_{r-1},..., D_1 so vrednosti na vhodu paralelnega pomnilnika, Q_p, Q_{p-1},..., Q_1 so vrednosti na izhodu ter V_1, V_2,...,V_m vrednosti v pomnilnih celicah. R_1, R_2,..., R_p specificirajo branje iz paralelnega pomnilnika, izjave $\bar{R}_1$, $\bar{R}_2$,..., $\bar{R}_r$ pa vpis v paralelni pomnilnik.

Sestavljena paralelna operacija se tedaj glasi:

Sel:

R_1 — sel:
ADR_{11} — P(Q_1,V_1)
ADR_{12} — P(Q_1,V_2)
⋮
ADR_{1m} — P(Q_1,V_m)

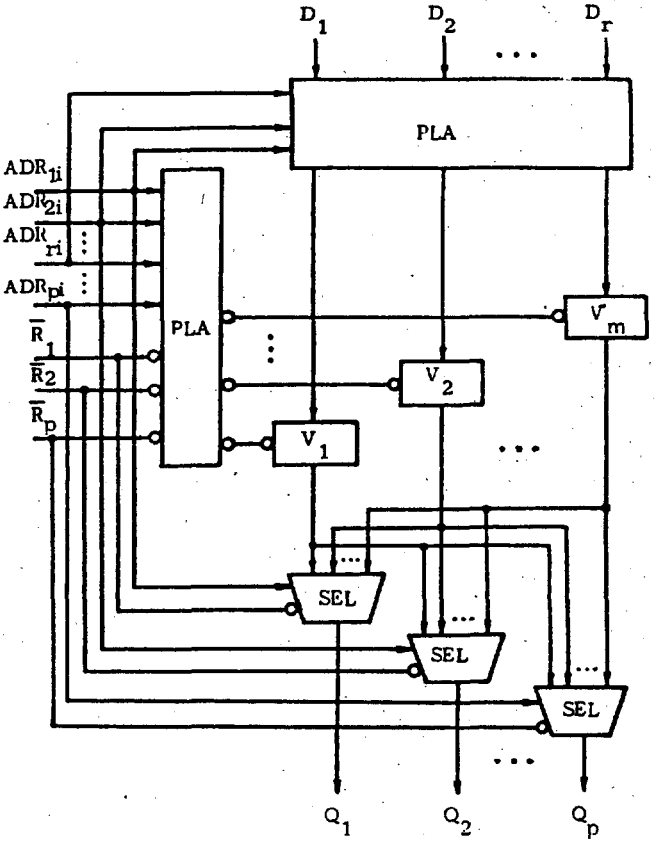
R_p — sel:
ADR_{p1} — P(Q_p,V_1)
ADR_{p2} — P(Q_p,V_2)
⋮
ADR_{pm} — P(Q_p, V_m)

$\bar{R}_1$ — sel:
ADR_{11} — P(V_1,D_1)
ADR_{12} — P(V_2,D_1)
⋮
ADR_{1m} — P(V_m,D_1)

$\bar{R}_r$ — sel:
ADR_{r1} — P(V_1,D_r)
ADR_{r2} — P(V_2,D_r)
⋮
ADR_{rm} — P(V_m,D_r)

Paralelna selektorska operacija, ki odloča o vpisu v pomnilne celice sicer ni enolična, vendar ni običajno, da bi istemu imenu komponente stanja skušali hkrati prirediti dve ali več vrednosti.

Eno izmed možnih blokovnih shem takega pomnilnika podaja slika 3.7.

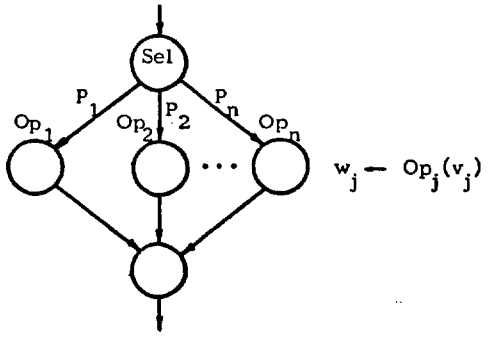


Slika 3.7: Blokovna shema paralelnega pomnilnika

Tak paralelni pomnilnik v praksi omogoča precej več kot realno potrebujemo. Zato ga lahko pri reševanju konkretne naloge primerno poenostavimo.

3.4. Modeli operatorjev

V tem razdelku bomo pod operatorjem razumeli predvsem enote, ki izvajajo operacije določene s selektorsko operacijo kot začetnim krmilnim modelom, ki ga lahko po potrebi razgradimo do primernega abstraktnega nivoja. V ta sklop sodi tudi selektorska operacija s slike 3.2, ki izvaja operacije $Op_j(v_j)$. Na sliki 3.8 je ponovno narisan graf selektorske operacije, iz katerega bomo razvili modele operatorjev.



Slika 3.8: Začetni približek k modelu operatorja

Selektorsko operacijo zapišimo v disjunktivni obliki:

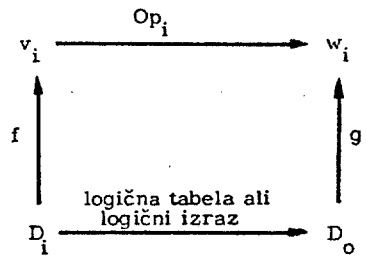
$$P_1 \wedge Op_1(v_1) \vee P_2 \wedge Op_2(v_2) \vee \ldots \vee P_n \wedge Op_n(v_n). \qquad (3.8)$$

$Op_j(v_j)$ pa pomeni vrednost w_j, ki jo dobimo kot rezultat operacije. (3.8) lahko tedaj zapišemo:

$$P_1 \wedge w_1 \vee P_2 \wedge w_2 \vee \ldots \vee P_n \wedge w_n \qquad (3.9)$$

Vendar (3.9) opisuje samo odjem vrednosti w_j iz izhoda operatorja, ne pa tudi izvajanje operacije. Zakaj sedaj nenadoma težave s selektorsko operacijo? Selektorska operacija, kot je definirana, je logični (krmilni) model kamor lahko zapišemo "karkoli" . Če sledimo pot i skozi selektorsko operacijo, lahko rečemo da, če je pravilna P_i potem se izvede operacija Op_i nad vrednostmi stanja Si_i in kot rezultat določi vrednosti končnega stanja So_i. Ostale poti v selektorski operaciji lahko v tem trenutku ignoriramo. Pri fizični izvedbi modela selektorske operacije pa je tako, da moramo definirati podatkovne poti, definirati logično izvedbo operatorjev, poskrbeti za krmiljenje celotnega operatorja in na koncu odvzeti pravilen rezultat.

V ta namen najprej definirajmo realizacijo operacije Op_i, i = 1, 2, ..., n, z operatorjem OP_i. V splošnem lahko operatorje realiziramo na dva načina; s pomočjo logičnih tabel ali s pomočjo logičnih izrazov. Slika 3.9



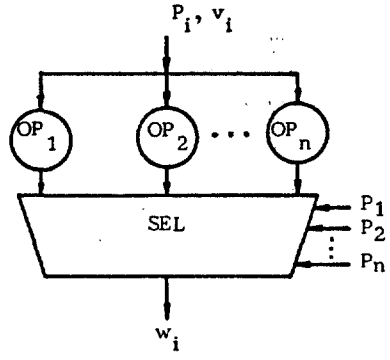
Slika 3.9: Definicija logičnega modela operatorja

podaja koncept po katerem operacijo Op_i prevedemo v njen logični model.

f in g sta surjektivni preslikavi, f^{-1} in g^{-1} pa sta v splošnem relaciji. D_i in D_o so nizi pravilnostnih vrednosti izjav, s katerimi smo modelirali vrednosti v_i in w_i glede na [1].

V splošnem lahko tedaj tudi operatorje ponazarjamo s selektorsko operacijo, saj lahko poljubno logično tabelo prevedemo v disjunktivno obliko prav tako pa tudi poljuben logični izraz.

Sedaj pa ponovno izhajajmo iz (3.8). Tedaj lahko glede na zgornja izvajanja narišemo naslednjo blokovno shemo operatorja za sliko 3.8 na sliki 3.10.



Slika 3.10: Začetni model operatorja

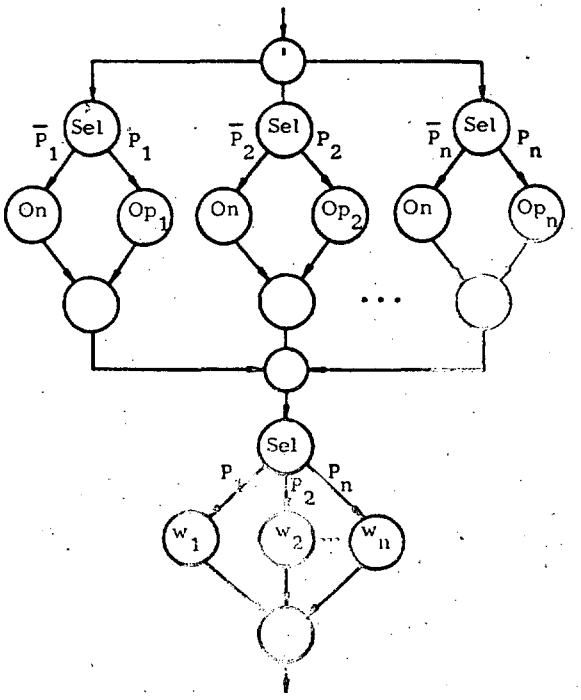
Kjer so $Op_1, Op_2, \ldots, Op_n$ v splošnem zopet selektorske operacije, ki jih lahko definiramo takole:

Sel:

$$\left.\begin{array}{l} P_i \wedge v_i^1 \rightarrow w_i^1 \\ P_i \wedge v_i^2 \rightarrow w_i^2 \\ \vdots \\ P_i \wedge v_i^x \rightarrow w_i^x \end{array}\right\} Op_i \qquad (3.10)$$

Z v_i^j smo označili različne vrednosti začetnih stanj definiranih z začetnim pogojem Pi_i (Si_i) in z w_i^j vrednosti končnih stanj So_i. V splošnem desna stran (3.10) ni enolična.

Sedaj pa definirajmo še nekatere enakovredne blokovne sheme operatorjev. Izhajajmo iz grafa na sliki 3.11.



Slika 3.11: Preoblikovana selektorska operacija

Ugotovimo lahko, da graf s slike 3.11 ustreza grafu s slike 3.8. Velja namreč:

$$OV\ldots VOVP_i \wedge Op_i(v_i) VOV\ldots VO \qquad (3.11)$$

in

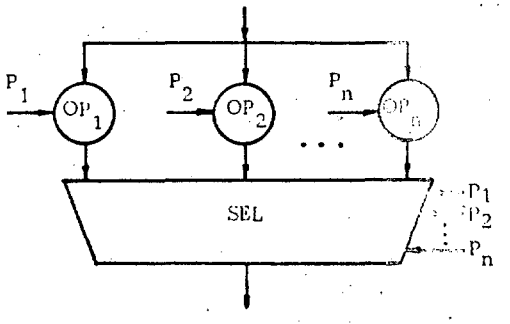
$$\begin{aligned}
&\overline{P}_1 \wedge O_n(v_i) \vee P_1 \wedge Op_1(v_1)\\
&\vdots\\
&\overline{P}_i \wedge O_n(v_i) \vee P_i \wedge Op_i(v_i) \qquad (3.12)\\
&\vdots\\
&\overline{P}_n \wedge O_n(v_n) \vee P_n \wedge Op_n(v_n)
\end{aligned}$$

Če je P_i pravilna tedaj zgornje relacije preidejo v obliko:

$$P_i \wedge Op_i(v_i)$$

kar je enako (3.11). Pri tem je oprecija O_n definirana tako, da velja $O_n(v_i) = 0$, $i = 1, 2, \ldots, n$.

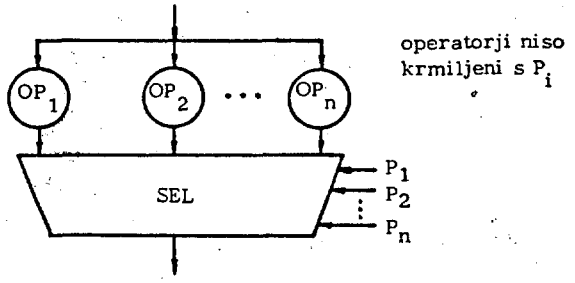
Blokovno shemo za ta primer podaja slika 3.12.



Slika 3.12: Enakovreden začetni model operatorja

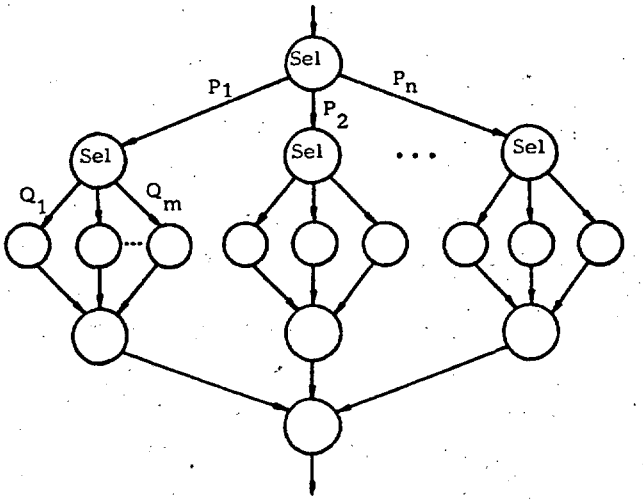
Iz razlogov, ki so snovalcem dobro znani lahko SEL operacijo s slike 3.12 pogosto nadomestimo z ALI operacijo ali s skupnim vodilom s tremi stanji.

V praksi se pogosto uporablja tudi poenostavljena blokovna shema operatorja s slike 3.12, ki jo podaja slika 3.13.



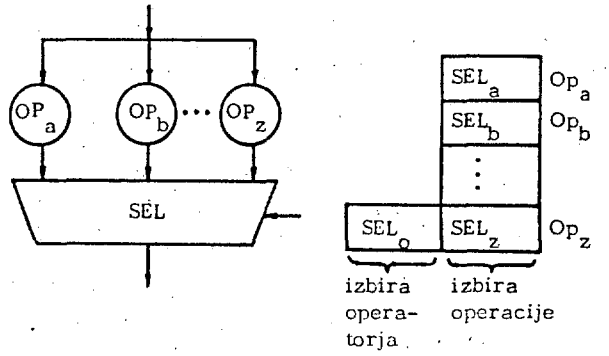
Slika 3.13: Poenostavljen model operatorja

Poimenujmo operator s slike 3.10 univerzalni operator. Sedaj pa izhajajmo iz krmilnega grafa na sliki 3.14.



Slika 3.14: Sestavljena selektorska operacija kot krmilni model

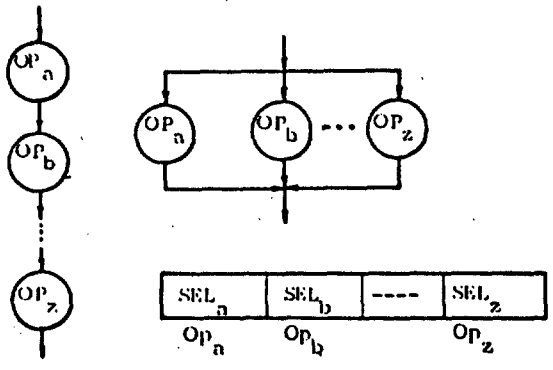
Vzemimo, da lahko notranje selektorske operacije realiziramo z operatorji OP_a, OP_b,..., OP_z. Tedaj lahko takoj narišemo blokovno shemo za celotno krmilno shemo s slike 3.14 na sliki 3.15.



Slika 3.15: Model operatorja in krmilna struktura za krmilni graf s slike 3.14

Na sliki 3.15 so SEL_i krmilni ukazi, do katerih pridemo s preoblikovanjem izjav $P_1,\ldots,P_n$, $Q_1,\ldots,Q_m,\ldots$ v nize ekvivalentnih izjav po postopku [1] . Pri snovanju mikroprogramiranih sistemov se je zanje udomačil izraz mikroukazi, za celotno polje pa izraz vertikalno funkcionalno polje.

V tem smislu sta na sliki 3.16 podani še blokovni shemi in struktura krmilnega polja za sekvenčno in paralelno vezane univerzalne operatorje.



Slika 3.16: Blokovni shemi in struktura krmilnega polja za sekvenčno in paralelno povezane operatorje

Snovanje operatorjev lahko tedaj pričnemo z izdelavo grafa, ki je v začetku selektorska operacija, ki jo po potrebi razgradimo v sestavljeno operacijo, sestavljeno iz selektorskih, sekvenčnih in paralelnih operacij. Tako dobljen graf pogosto imenujemo krmilni graf operatorja, saj določa precedenco operacij pri izvajanju. S pomočjo krmilnega grafa pa nato izdelamo blokovno shemo operatorja, ki izvede operacije določene z krmilnim grafom. Naslednji korak je logično snovanje, ki nas pripelje do logične sheme operatorja.

4. ZAKLJUČEK

Predlagani postopki snovanja in izgradnje logičnih modelov računalniških struktur so splošni in niso omejeni samo na snovanje mikroprogramiranih sistemov. Osnova postopkov so selektorska, sekvenčna in paralelna operacija, ki so lahko tudi vse zaključene. Z dodajanjem semantike lahko z njimi modeliramo najrazličnejše računalniške strukture na različnih abstraktnih nivojih.

Pokazali smo, kako lahko iste abstraktne strukture uporabljamo tako za snovanje krmilnih mehanizmov kot operatorjev, ki jih le-ti krmilijo. Izkazalo se je, da v splošnem ni mogoče postaviti ostre meje med krmilno in operacijsko strukturo izbrane računalniške strukture. Meja je s stališča zunanjega opazovalca (uporabnika, programerja itd.) določena z abstraktnim nivojem na katerem opazujemo obnašanje izbrane računalniške strukture. Od tod izhaja tudi programski model izbrane računalniške strukture za izbrani abstraktni nivo opazovanja. Po drugi strani pa lahko v splošnem isto računalniško strukturo opazujemo kot krmilno strukturo, operacijsko ali pomnilno strukturo. Ugotovili smo namreč, da v splošnem sestavljene strukture vsebujejo vse tri komponente. Od konteksta opazovanja je odvisno kako bomo "videli" takšno strukturo. Potrdimo to z zgledom: mikroprogramirana krmilna enota je brez dvoma krmilna struktura, saj določa operacije in njihovo zaporedje pri izvajanju. Hkrati pa je tudi pomnilna struktura, saj na adrese iz različnih virov odgovarja z različnimi podatki, ne glede na to kaj ti podatki predstavljajo. V njej lahko vidimo tudi operacijsko strukturo, posebej če jo opazujemo v kontekstu instrukcijski niz - mikroprogramirana krmilna enota - mikroinstrukcija - operacijska enota - izjave po končani operaciji, kjer je krmilna enota prva izmed dveh sekvenčno povezanih operatorjev (operacij).

Takšno menjavo kontekstov opazovanja pri snovanju pogosto uporabljamo, saj nam menjava konteksta prikaže problem v novi dimenziji in nas s tem navede na rešitve, do katerih bi lahko težko prišli z drugačnim kontekstom opazovanja.

5. LITERATURA

[1] M. Gerkeš: Logični modeli računalniških struktur, Informatica 3, str. 1 - 12, Ljubljana 1985.

[2] C. B. Jones: Software Design: A Rigorous Approach Prentice-Hall International, 1980.

[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, PORS 3, Visoka tehniška šola, Maribor, 1984.

Raziskavo je sofinancirala Raziskovalna skupnost Slovenije PoRS 03.

RASTERIZACIJA Z MIKRORAČUNALNIKOM

Barbara Lakner, France Dacar
Institut Jožef Stefan, Ljubljana

UDK : 681.3:514.1

V prispevku je opisana rasterizacija z mikroračunalnikom. Ker v mikroračunalniku ni dovolj prostora za predstavitev cele rastrske mreže, zgradimo najprej model slike, ki je sestavljen iz črt, vodoravnih tekstov in pik in s tehniko postopnega preiskovanja ravnine pregledamo model od leve proti desni, postopoma rasteriziramo in sproti izrisujemo sliko na grafični napravi.

SCAN-CONVERTING WITH A MICRO-COMPUTER

In the article scan-converting with a micro-computer is described. Because of the lack of memory to save the whole raster grid, we build a model of a picture. The model consists of lines, horizontal texts and dots. Then we sweep the model from left to right with the plane-sweep technique (or scan-line approach), gradually scan-convert and simultaneously draw the picture on a graphics device.

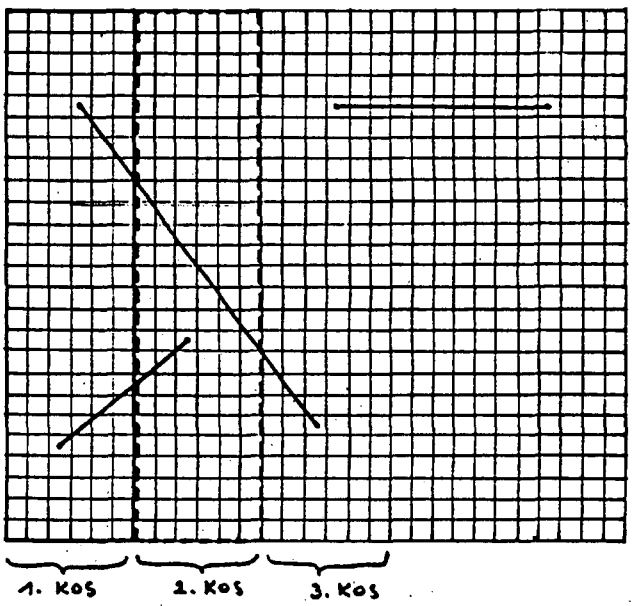
1. UVOD

Rasterizacija je predstavitev črtne slike s pikami (piksli), ki so razporejene na določeni dvodimenzionalni rastrski mreži. Osnovna naloga rasterizacije za črte je torej računanje koordinat pikslov, ki leže v neposredni bližini črte na tej mreži. Če imamo v računalniku dovolj prostora za predstavitev take mreže, nimamo nobenih posebnih težav: uporabimo enega izmed znanih algoritmov ([1], [2]), s katerim črtam priredimo ustrezne piksle na mreži in nato ob pregledovanju te mreže pošiljamo ustrezne ukaze v izbrano grafično napravo (npr. matrični printer).

V mikroračunalniku, npr. LSI-11 žal nimamo dovolj prostora za predstavitev cele rastrske mreže. Opisali bomo implementacijo preprostega grafičnega paketa na mikroračunalniku. Paket ima procedure za premikanje peresa s spuščeno in dvignjeno konico, pisanje tekstov v različnih smereh, nastavljanje različnih debelin peresa in nastavljanje koordinatnih sistemov. Paket ob klicih teh procedur gradi le model slike.

Model slike je sestavljen iz črt, pik in vodoravnih tekstov in je po potrebi, če je podatkov preveč, zapisan na datotekah. Ko elemente modela uredimo po levih x-koordinatah, se s preiskovalno premico lotimo postopnega pregledovanja modela od leve proti desni. Naenkrat obravnavamo le tiste elemente modela, ki so nabodeni na preiskovalno premico. Tem elementom priredimo piksle na mreži, ki leže v njihovi neposredni bližini in na preiskovalni premici. Nato preiskovalno premico premaknemo v levo za korak, ki ustreza širini piksla in ponovimo postopek prirejanja pikslov. Ko je preiskovalna premica prišla do desnega roba kosa rastrske mreže, kos mreže izrišemo, jo inicializiramo in nadaljujemo s preiskovanjem modela od leve proti desni (slika 1).

Tako smo pri rasterizaciji z mikroračunalnikom uporabili pristop, ki se v računalniški grafiki pojavlja z imenom "scan-line approach", v računalniški geometriji pa "plane-sweep paradigm".



Slika 1: Celotna rastrska mreža in posamezni kosi

2. OPIS ZMOGLJIVOSTI PAKETA

Opisani paket za rasterizacijo je namenjen risanju slik z matričnim printerjem in ima naslednje procedure in funkcije:

```
procedure StartPlot;
procedure StopPlot;

procedure PaperWidth(w: real);
procedure SetWindow(Wxmin, Wymin, Wxmax, Wymax:
real);
procedure SetViewPort(Vxmin, Vymin, Vxmax,
Vymax: real);
procedure SetIsotropicTransf(var Wxmin, Wymin,
Wxmax, Wymax: real);


procedure Move(x,y: real);
procedure Draw(x,y: real);

procedure PenWidth(d: integer);

procedure PlotText(s: string);
procedure TextHeight(h: real);
procedure TextDir(a: real);

function PenPos: point;
```

kjer je

```
    string = packed array[1..50] of char;
    point = record
              x,y: real;
            end;
```

S procedurama StartPlot oz. StopPlot začnemo oziroma končamo risanje.

Uporabnik določi svoj koordinatni sistem s proceduro SetWindow, s proceduro SetViewPort pa velikost pravokotnika na papirju, kamor rišemo sliko. Procedura SetIsotropicTransf nam omogoča, da popravimo dimenzije pravokotnika na papirju tako, da se pri preslikovanju iz uporabnikovega koordinatnega sistema v pravokotnik na papirju ohranjajo oblike, kar pomeni, da se krogi res preslikajo v kroge in ne v elipse. S proceduro PaperWidth lahko izsilimo risanje slike, ki je širša od printerskega papirja; slika se nariše v več kosih, ki jih pozneje zlepimo skupaj.

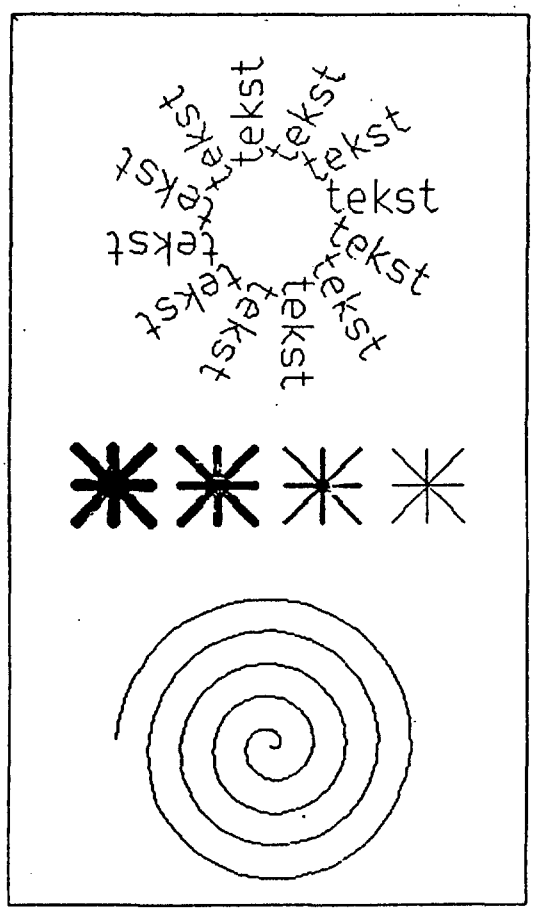
Gibanje peresa s spuščeno oziroma dvignjeno konico uravnavamo s proedurama Move in Draw, debelino peresa pa spreminjamo s proceduro PenWidth. Izbiramo lahko med štirimi različnimi debelinami. Funkcija PenPos nam pove trenutni položaj peresa.

Kot že ime pove, rišemo tekste s proceduro PlotText, njih višino nastavimo s proceduro TextHeight in kot, pod katerim jih pišemo, s proceduro textDir.

Naštete procedure in funkcije sodijo v takoimenovani železni repertoar vsakega grafičnega paketa in z njihovo pomočjo lahko že marsikaj narišemo (slika 2).

3. MODEL SLIKE

Model slike je zelo preprost: sestavljen je iz črt, vodoravnih tekstov in pik.To so osnovni gradniki slike. Model gradimo postopoma ob klicih posameznih procedur - delamo seznam osnovnih gradnikov. Sezname zapisujemo najprej



Slika 2: Slika, narisana z opisanim rasterizatorjem

v pomnilnik in če postane seznam predolg, ga prepišemo na datoteko in tja dopisujemo ostale elemente. Tako so modeli z malo elementi v pomnilniku in jih hitreje obdelamo, kot bi jih, če bi model že kar takoj začeli zapisovati na datoteke.

Ustavimo se še ob osnovnih gradnikih modela:

* **črte** so široke za debelino piksla in predstavljene s svojimi krajišči na mreži

* **vodoravni teksti** : ker so znaki (črke) sestavljeni iz črt, bi lahko vodoravne tekste takoj predelali v črte, vendar tega ne storimo, ker bi bil seznam s črtami zelo dolg. Tekst je predstavljen s pozicijo spodnjega levega kota, višino in nizom znakov.

* **pike** : črte, ki so široke za večkratno širino piksla, so sestavljene iz tankih črt. Tak sveženj tankih črt zaključimo na vsakem koncu s piko, da ustvarimo vtis risanja s peresom, ki ima okroglo konico. Pika je predstavljena s središčem in številko peresa, na katerega se nanaša.

4. POSTOPNA RASTERIZACIJA

4.1. Postopno preiskovanje ravnine

Postopno preiskovanje ravnine je metoda, ki se je v računalniški grafiki in geometriji že dodobra uveljavila [3]: uporablja se pri iskanju presečišč daljic, presekov ravninskih likov [4], presekov poliedrov [5] in barvanju poligonov ([1], [6]). Opisali bomo uporabo te metode pri rasterizaciji z mikroračunalnikom.

Rešitvam teh precej različnih problemov je skupno potovanje iskalne premice od leve proti desni (ali v kaki drugi izbrani smeri, npr. od zgoraj navzdol). Premica se ustavlja v karakterističnih točkah; ob vsakem postanku gledamo le objekte, ki so nabodeni nanjo in z njimi, v odvisnosti od problema, ki ga rešujemo, nekaj naredimo.

Seveda se seznam objektov, ki so nabodeni na premico, z gibanjem premice spreminja in ga moramo pri vsakem postanku premice popraviti. Udomačilo se je [4], da rečemo objektom, ki so nabodeni na preiskovalno premico, aktivni objekti, tistim, ki so povsem levo od nje, mrtvi objekti in objektom, ki so povsem desno od premice, speči objekti (slika 3).

Pri rasterizaciji se premica ustavi v središču vsakega piksla (pri tem mislimo na vodoravno vrstico pikslov) in objektom, ki so nabodeni na premico, priredimo piksle na mreži, ki leže v njihovi neposredni bližini in na preiskovalni premici (slika 4). Oglejmo si še zapis ustrezne procedure:

```
procedure PrintPicture;

begin (* PrintPict *)
    x := xmin; j := 0;
    while x <= xmax do begin
      j := j + 1;
      getActLines(x);
      SkanActLines(x,j);
      getActDots(x);
      SkanActDots(x,j);
      if (j=jmax) then
        begin PrintPart; j := 0; end;
      x := x + 1;
    end;
    if (j > 0) then PrintPart;
  end;
 end; (* PrintPict *)
```

4.2 Rasterizacija črt

4.2.1. Bresenhamov algoritem

Črte smo rasterizirali z inačico Bresenhamovega algoritma [2]. Ker izpeljava ni dolga, si jo oglejmo. Vzemimo, da rasteriziramo daljico s krajiščema (0,0) in (deltax, deltay). Naj bo

$$deltax > 0,\ deltay > 0 \text{ in } deltay <= deltax \quad (1)$$

Daljici prirejamo piksle s koordintami (x,y), kjer je

$$0 <= x <= deltax$$

in

$$deltay/deltax*x - 1/2 <= y < deltay/deltax*x + 1/2$$

Tako leže piksli, ki predstavljajo daljico, v pasu, ki je narisan na sliki 5. Gornjo neenakost preuredimo

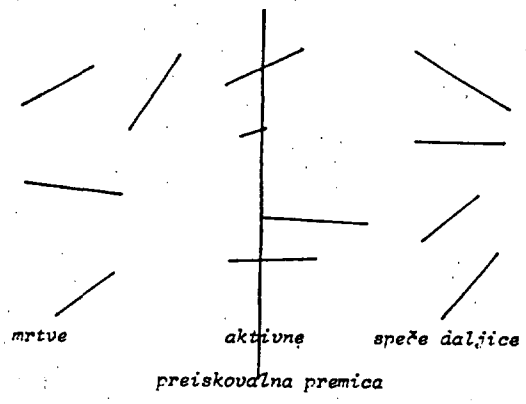
$$0 <= 2*deltax*y - 2*deltay*x + deltax < 2*deltax$$

in imamo

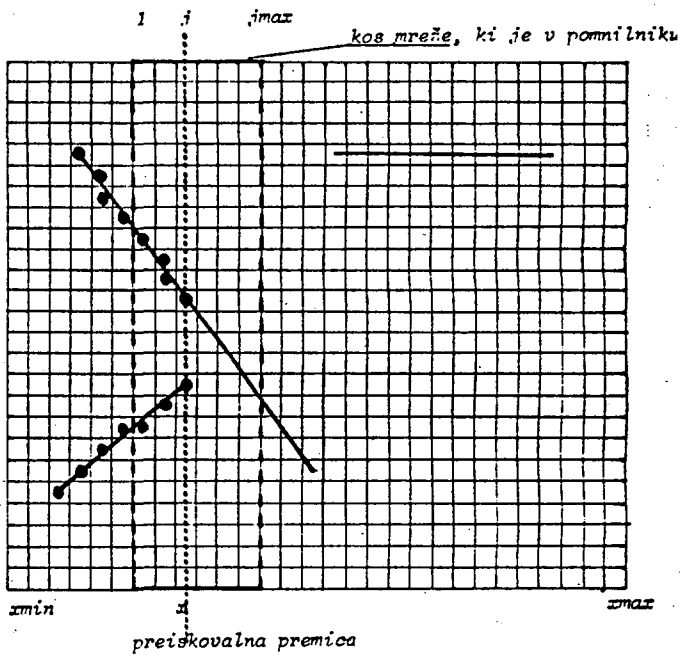
$$0 <= z(x,y) < 2*deltax,$$

kjer je

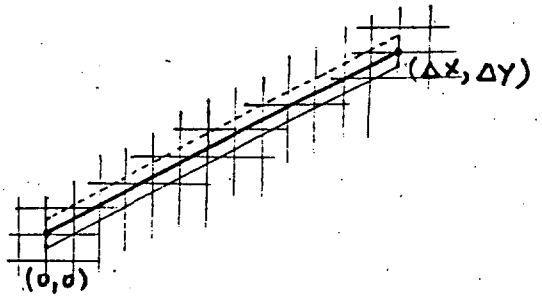
$$z(x,y) = 2*deltax*y - 2*deltay*x + deltax$$



Slika 3: Preiskovalna premica, mrtve, aktivne in speče daljice



Slika 4: Postopna rasterizacija



Slika 5: Pas, v katerem leže piksli, ki pripadajo narisani daljici

```
in je
    z(0,0) = deltax
    z(x+1,y) = z(x,y) - 2*deltay
    z(x,y+a) = z(x,y) + 2*deltax
```

S tem že lahko zapišemo algoritem za rasterizacijo daljice s krajiščema (x0,y0), (x1,y1), ki zadošča predpostavkam (1). Naj bo

```
    deltax := x1 - x0  in deltay := y1 - y0
```

```
z := deltax; y := y0;
PlotDot(x0,y0);
for x := 1 to deltax do begin
    z := z - 2*deltay;
    if z < 0 then begin
     z := z + 2*deltax;
     y := y + 1;
    end;
    PlotDot(x,y);
end;
```

Za ilustracijo navedimo še posplošitev gornjega algoritma za rasterizacijo poljubnih daljic:

```
procedure Line(x0,y0,x1,y1: integer);
var
  sgnx,sgny,absx,absy,sx,sy,a,b,n,i,z: integer;
begin

(*  inicializacija  *)
  if (x1-x0) < 0 then sgnx:=-1 else sgnx:=1;
  if (y1-y0) < 0 then sgny:=-1 else sgny:=1;
  absx := abs(x1-x0); absy := abs(y1-y0);
  if absx > absy then begin
     sx := sgnx; sy := 0;
     n := absx; b := 2*absy;
   end
   else begin
     sx := 0; sy := sgny;
     n := absy; b := 2*absx;
   end;
  a := 2*n; z := n;
  x := x0; y := y0; i := 0;

(*  rasterizacija  *)
  PlotDot(x,y);
  for i := 1 to n do begin
    z := z - b;
    if z<0 then begin x:=x+sgnx; y:=y+sgny end
           else begin x:=x+sx; y:=y+sy end;
    PlotDot(x,y);
  end;
 end; (*  Line  *)
```

4.2.2. Bresenhamov algoritem in postopno preiskovanje ravnine

Imejmo daljico (x0, y0), (x1, y1), kjer je x0 <= x1. Bresenhamov algoritem prireja daljici piksle koordinatami (x(i), y(i)), i = 1..n, kjer je x(i) <= x(i+1). Algoritem izračuna koordinate piksla (x(i+1), y(i+1)) iz koordinat svojega predhodnika (x(i), y(i)) in celih števil, ki so definirana znotraj procedure Line. Od tod že vidimo, da moremo daljici prirejati pripadajoče piksle postopoma, to je, najprej vse tiste piksle (x,y), kjer je x=x0, nato vse piksle, za katere je x = x0+1 in tako dalje, dokler ne pridemo do konca daljice, kjer je x=x1. In to je ravno tisto, kar delamo pri postopni rasterizaciji.

Aktivne daljice morajo torej nositi s sabo podatke o trenutnem stanju rasterizacije: x, y, b, sx, sy, n, i, sgnx, sgny in z (oznake se nanašajo na iste količine kot v procedure Line). Pred začetkom rasterizacije jih inicializiramo enako kot v proceduri Line. Navedimo zdaj še proceduro za postopno rasterizacijo

```
procedure Line1(var l: ddaData; j: integer);

  var xold: integer;
  begin
   with l do begin
    xold := x;  PlotDot(j,y);
    while (i <= n) and (x = xold) do begin
       z := z - b;
       if z < 0 then begin
                    z := z + 2*n; x := x + sgnx;
                    y := y + sgny
                 end
        else begin x:=x+sx; y:=y+sy end;
        i := i + 1;
        if (x = xold) then PlotDot(j,y);
    end;
   end;
 end; (*  Line1  *)
```

```
kjer je
   ddaData = record
        x,y,sx,sy,n,i,sgnx,sgny,z: integer
     end;
```

in je j lokalna x-koordinata pikslov na kosu rastrske mreže.

5. IMPLEMENTACIJA

Program za postopno rasterizacijo je napisan v Pascalu in teče na računalniku VAX 11/750, LSI-11 (PDP-11) in PMP-11. S komentarji vred obsega 1800 vrstic. Na mikroračunalnikih LSI-11 in PMP-11 je razbit na več kosov. Prirejen je za izrisovanje slik na printerjih la-120, Star Delta-10 in Facit 4542. Na sliko 2 smo pri računalniku LSI-11 morali čakati 100 sekund, na sliko 6 pa 237 sekund. Omenjeni sliki sta sestavljeni iz približno 600 oziroma 500 daljic. Sliko z malo daljicami (npr. 10) pa obdela v približno 15 sekundah.

6. LITERATURA

[1]: W. M. Newman, R. F. Sproull: Principles of Interactive Computer Graphics, Second Edition, McGraw-Hill Book Company, 1979

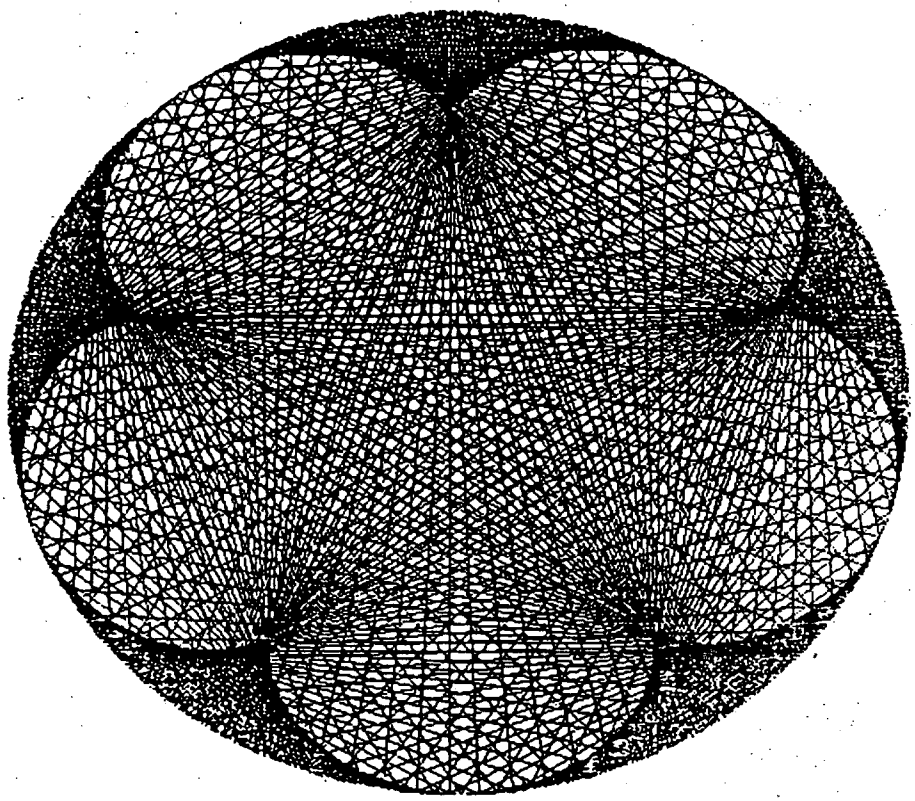
[2]: J. D. Foley, A. Van Dam: Fundamentals of Interactive Computer Graphics, Addison-Wesley Publishing Company, 1982

[3]: D. T. Lee, F. P. Preparata: Computational Geometry - A Survey, IEEE Transactions on computers, Vol. C-33, No. 12, Dec. 1984, 1072-1101

[4]: K. Mehlhorn: Multi-dimensional Searching and Computational Geometry (Daa Structures and Algorithms 3), Springer Verlag, Berlin 1984

[5]: M. Szivalasi-Nagy: An Algorithm for determing the intersection of Two Simple Polyhedra, Computer Graphics Forum 3(1984), 219-225

[6]: T. Pavlidis: Algorithms for Graphics and Image Processing, Springer Verlag, Berlin 1982



Slika 6

UPORABA VME VODILA PRI ROBOTSKIH KRMILNIH SISTEMIH

Ivan Verdenik, Laboratorij za robotiko
Fakulteta za elektrotehniko, Univerza Edvarda Kardelja, Ljubljana

UDK : 681.519.7

POVZETEK - Za učinkovito vodenje mehanskega manipulatorja po izbrani trajektoriji je potrebno poznavanje sil in momentov v vseh sklepih - dinamični model. Ker je matematični zapis zelo obsežen, je izračunavanje kompenzacijskih regulacijskih vhodov v realnem času z enim mikroprocesorjem neprimerno. Večprocesorski sistemi zahtevajo za pravilno, zanesljivo in učinkovito delovanje določena pravila, ki upoštevajo zahteve aparaturne in programske opreme. Eno izmed možnosti zgradbe večprocesorskega sistema nam daje standardno VME vodilo. To vodilo predstavlja skupino pravil in lastnosti za povezavo več modulov v procesor - multiprocesorski sistem. V delu so raziskane prednosti in slabosti uporabe VME vodila za gradnjo večprocesorskega krmilnika manipulatorja in karakteristične zahteve, ki jih postavljamo pri računanju dinamičnega modela.

APPLICATION OF VME BUS FOR ROBOT CONTROLLERS

ABSTRACT - For efficient control of mechanical manipulator through desired trajectory, forces and torques for driving actuators at joints must be known. Because of complexity of dynamic model, a real time computation with single microprocessor is not practical. Multiprocessor systems demand specific software and hardware related rules for adequate operation. One possibility for multiprocessor design is given by standard VME bus. This bus represents group of rules for interconnecting modules into system, where each module can contain its own processor. In this paper are reviewed advantages and drawbacks of FME based manipulator controller and characteristic requirements for dynamic model computations are given.

1. UVOD

Vodenje manipulatorja po neki trajektoriji predstavlja zahteven problem tako s stališča programske kot tudi aparaturne opreme. Poznati moramo mehanične lastnosti manipulatorja (kinematične in dinamične) in iz njih v realnem času izračunavati krmilne vhode v regulator. Izračun kinematike industrijskih manipulatorjev je v praksi že rešen problem, medtem ko izračun dinamičnega modela manipulatorja predstavlja še vedno izjemno zahtevno nalogo. Prvi del - samo formuliranje dinamičnih enačb sicer poteka po že ustaljenem postopku (Newton-Eulerjeva metoda), vendar so zaradi velike kompleksnosti enačb možnosti napak velike in njihovo odpravljanje zelo zamudno. Drugi del naloge - programiranje krmilnega sistema manipulatorja - je prav tako zelo zahteven postopek, saj so časovni kriteriji (čas izvajanja programa) izredno kritični.

Aparaturna oprema nam lahko delo pri razvoju programske opreme za računanje dinamičnega modela v realnem času olajša ali pa tudi oteži. Zato je zelo pomembno, kaj nam aparaturna oprema krmilnika manipulatorja nudi in kje nas omejuje. VME vodilo je eden izmed sistemov aparaturne opreme, ki se je izkazal za primernega za gradnjo oziroma razvoj krmilnika dinamično vodenega industrijskega manipulatorja.

2. DINAMIČNO KRMILJENJE MANIPULATORJA

Naloge, ki jih opravljajo industrijski manipulatorji, so določene s trajektorijo, ki jo opiše vrh prijemala. Za večino komercialno dosegljivih robotov je ta trajektorija programirana vnaprej, tako da mora krmilnik robota voditi aktuatorje v sklepih z dovolj velikimi navori oziroma silami za uspešno sledenje trajektoriji. V primeru velikih hitrosti gibanja ali ko je breme, ki ga manipulator prenaša, spremenljivo, pa nastopijo odstopanja od želene trajektorije.

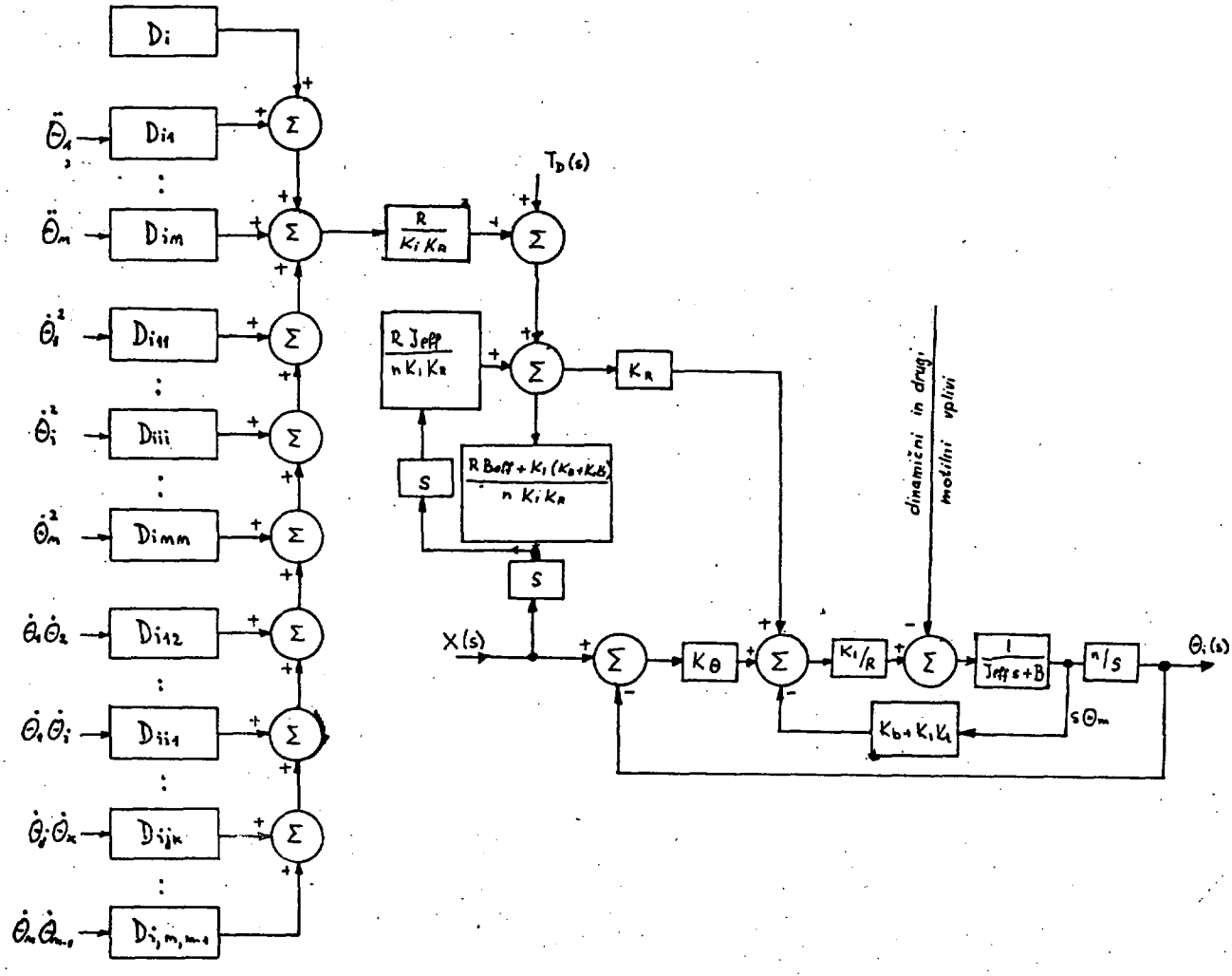
Vplivi centrifugalnih, Coriolisovih in gravitacijskih sil povzročijo omenjeno odstopanje od izračunane trajektorije. Da bi vpliv teh motilnih veličin odpravili, moramo izračunati njihovo velikost in jih kompenzirati. Običajno to napravimo s kompenzacijskimi vhodi v regulator /1/. Navor oziroma sila, s katero krmilimo posamezni sklep manipulatorja, je pri tem dana z enačbo:

$$\tau_i = \sum_{j=1}^{m} D_{ij}\,\ddot{\Theta}_j + J_{ai}\ddot{\Theta}_i + \sum_{j=1}^{m} D_{iii}\,(\dot{\Theta}_i)^2 + \sum_{j=1}^{k}\sum_{k=1}^{m} D_{ijk}\,\dot{\Theta}_j\,\dot{\Theta}_k + D_i$$

Ta enačba predstavlja inverzni dinamični model. Koeficiente D lahko razdelimo na pet glavnih skupin. Prva skupina (členi Dij.$\ddot{\theta}$j) predstavljajo medsebojne vplive vztrajnostnih momentov med segmenti manipulatorja, druga skupina (členi Jai.$\ddot{\theta}$i) vztrajnostne momente posameznih segmentov, tretja skupina (Diji($\dot{\theta}$j)^2) prispevek zaradi centrifugalnih pospeškov, členi (Dijk.$\dot{\theta}$i.$\dot{\theta}$j) Coriolisove prispevke in zadnja skupina (Di) prispevek zaradi sile teže. Na sl.1 je prikazana poenostavljena shema regulatorja ene prostostne stopnje manipulatorja s kompenzacijskimi dinamičnimi vhodi.

Na sl.1 predstavljajo θi notranje koordinate posameznih sklepov, $\dot{\theta}$i so pripadajoče hitrosti in $\ddot{\theta}$i pospeški. Po množenju s členi D in seštevanju dobimo potreben gonilni moment oziroma silo Td, ki predstavlja približno vrednost momenta ali sile trenja in bremena. Ostali del slike je klasičen regulator s konstantami, ki so odvisne od geometrije manipulatorja, vrste aktuatorja in načina prenosa z osi aktuatorja na sklep manipulatorja.

Izračun vseh členov Dijk v realnem času je vse prej kot enostaven postopek, saj so vsi ti



Slika 1

členi nelinearne funkcije, odvisne od geometrije in pozicije manipulatorja. Iskanje oziroma izračun vseh teh členov je inverzni dinamični problem. Za izračun tega modela je bilo razvitih in uporabljenih precej metod /3/, v zadnjem času pa kaže, da je Newton-Eulerjeva metoda najučinkovitejša numerična metoda za reševanje dinamike industrijskega manipulatorja v realnem času. V tabeli 1, ki jo podajamo po Hollerbachu /3/, so podana števila množenj in seštevanj, potrebna za izračun modela po različnih metodah.

metoda	št.množenj/deljenj	št.seštevanj/odštevanj
Lagrange (Uicker-Kahn)	66271	51548
rekurzivna Lagr. (Waters)	7051	5652
rekurzivna Lagr. (Hollerbach)	4388	3586
rekurzivna Lagr. (Hollerbach)	2195	1719
Newton-Euler	852	738

3. PARALELNO PROCESIRANJE PRI IZRAČUNAVANJU DINAMIČNEGA MODELA

Z Newton-Eulerjevo metodo je možen izračun dinamike v realnem času na srednje velikih miniračunalnikih, kot je npr. PDP 11 /2,3,5/. Če želimo opraviti ves izračun na mikroračunalniku, moramo poskrbeti za zmogljiv računski sistem. Jasno je, da paralelno procesiranje občutno poveča hitrost računanja. Z nižanjem cene procesorskih komponent je postala zamisel o večprocesorskem krmilniku manipulatorja uresničljiva. Luh in Lin /2/ sta predlagala paralelni način krmiljenja, kjer vsakemu sklepu manipulatorja pripada svoj procesor z lokalnim pomnilnikom za podatke in program ter globalni pomnilnik za spremenljivke, ki jih potrebuje več procesorjev. Problem razvrščanja nalog po procesorjih je na ta način poenostavljen, saj omogoča iskanje optimalnega razvrščanja. Boljši način razvrščanja nalog po procesorjih pa je tisti, kjer vsak procesor lahko opravi vsako nalogo. Pri tem načinu zaradi velikega števila nalog ne moremo najti optimuma, temveč se mu lahko z raznimi hevrističnimi metodami le približamo /4,6/. Na ta način se za manipulator s šestimi prostostnimi stopnjami in šestimi procesorji čas računanja zmanjša za 40 % glede na način, kjer je vsakemu aktuatorju prirejen svoj procesor. Ta prihranek časa gre na račun dejstva, da pri metodi, ko ima vsak sklep svoj procesor, prihaja do mrtvega časa, ko en procesor čaka na rezultate drugega. Ta zakasnitev je pri drugem načinu mnogo manjša.

Problem strukture robotskega pomnilnika je navezan na tip procesorskih modulov, ki jih uporabljamo, saj čas prenosa podatkov med

posameznimi procesorji glede na vsa potrebna množenja in druge aritmetične operacije, ni kritičen.

Čas računanja kinematike manipulatorja, to je pretvorbe iz kartezijevih v notranje koordinate, je zanemarljiv v primerjavi s časom računanja celotne dinamike. Morali pa bi ga upoštevati, če bi želeli zgraditi krmilnik s poenostavljenim dinamičnim modelom, torej z zanemarjanjem določenih dinamičnih parametrov sistema (običajno se zanemarja centrifugalni ali Coriolisov pospešek ali oba).

4. VME VODILO

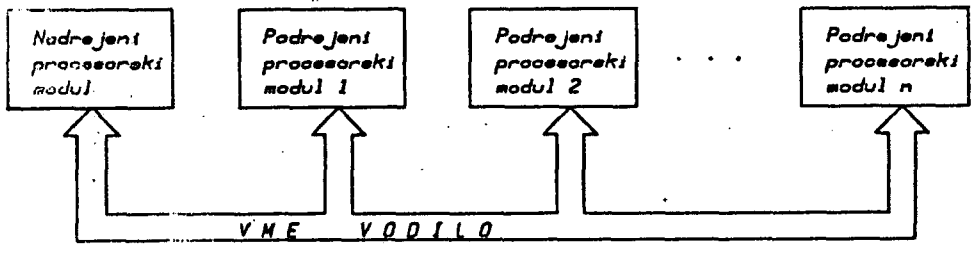
VME vodilo predstavlja vmesniški sistem za procesiranje, shranjevanje podatkov in za vodenje perifernih naprav v tesno vezani konfiguraciji. Tako VME vodilo omogoča:

- komunikacijo med napravami na vodilu brez motenj njihovih internih aktivnosti
- specificira električne in mehanske karakteristike sistema, ki so potrebne za razvoj in zanesljivo delovanje sistema
- specificira protokol in natančno definira odnose med napravami na vodilu
- predpisuje terminologijo in definicije za precizen opis sistemskega protokola
- dovoljuje širok spekter različnih naprav na vodilu, ne da bi se zmanjšala kompatibilnost sistema
- omogoča sistem, ki je omejen s sposobnostjo naprav na vodilu in ne z vodilom samim.

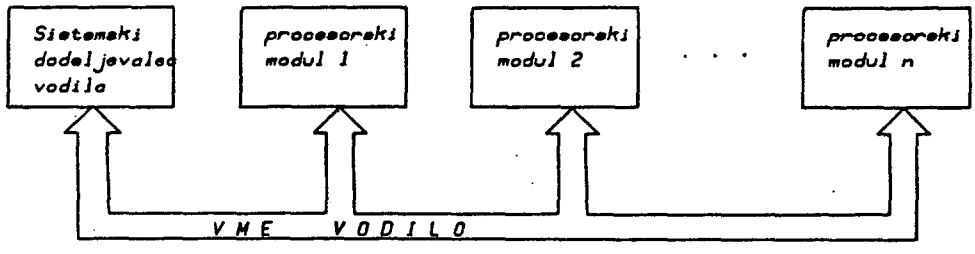
Glede na možnosti VME vodila in zahteve paralelnega procesiranja dinamičnega krmiljenja vidimo, da se ponujata dve alternativni izvedbi krmilnika manipulatorja. Pri prvi izvedbi (sl. 2) so vse procesorske enote podrejene centralni enoti, ki skrbi za prenos podatkov med moduli, za povezavo z regulatorskimi enotami in za komunikacijo z uporabnikom (operacijski sistem). Prednost te izvedbe je v tem, da sistem ne izgublja časa s stalnim razsojanjem, komu pripada vodilo ob vsakem trenutku ter prepusti celotno računanje posebej prirejenim procesorskim modulom.

V drugi izvedbi (sl.3) ima vsak procesor možnost postati nadrejena enota vodila (bus master). Tako se izognemo posebnemu centralnemu procesorju za koordinacijo posameznih procesnih modulov. To delo prepustimo modulom samim in sistemskemu dodeljevalcu vodila (arbitration module).

Porazdelitev pomnilnika po modulih vodila je odvisna od posamezne aplikacije. Globalni pomnilnik uporablja glavni sistemski procesor oziroma vsi procesorji za podatke, ki so rezultat ali vhodna veličina v določen podprogram. Lokalni pomnilnik pa uporablja vsak procesor za svoje lokalne spremenljivke. Pri tem lahko ugotovimo, da je zahtevana velikost pomnilnika majhna, medtem ko je čas dostopa bolj kritičen, saj direktno vpliva na hitrost delovanja posameznega procesorja.



Slika 2



Slika 3

Slabost VME vodila za gradnjo robotskega krmilnika je predvsem v tem, da gre za uporabo univerzalnih modulov v sistemu, ki je zelo specifičen. Vendar pa to pomanjkljivost odtehta razširjenost, cenenost in fleksibilnost VME standarda.

Poleg že naštetih prednosti VME vodila je izrednega pomena razširljivost sistema, saj lahko dodamo množico standardnih modulov (gibki disk, serijsko komunikacijsko enoto, analogne vhode/izhode...) in uporabimo že obstoječo programsko opremo (operacijski sistemi, prevajalniki...). S tem dosežemo, da je naš sistem vedno učinkovito prirejen posamezni aplikaciji, kar rezultira v optimalni ceni robotizacije. Pomembno pa je tudi poenostavljanje razvojnega dela, saj v času razvoja lahko uporabljamo module, s katerimi razvoj poteka hitreje, ko pa je projekt končan, opremimo sistem z najmanjšim potrebnim številom modulov.

5. ZAKLJUČEK

V delu smo podali pomembne zahteve za gradnjo večprocesorskega krmilnika dinamično vodenega industrijskega manipulatorja. Problemov izračuna kinematičnega modela nismo upoštevali, saj so glede na dinamični model enostavnejši. VME vodilo, kot zelo razširjen (tudi pri nas - Iskra Delta-Triglav) in zmogljiv računalniški sistem, se je pokazal kot najprimernejša osnova za razvoj kompleksnega robotskega krmilnika.

6. ZAHVALA

Raziskava je bila opravljena v Laboratoriju za robotiko na Fakulteti za elektrotehniko v Ljubljani. Delo sta financirali delno RSS in delno Iskra Delta. Zahvaljujem se prof.dr. A.Kralju za mentorstvo pri opravljanju naloge ter sodelavcem laboratorija za mnoge koristne pripombe in predloge.

7. LITERATURA

(1) J.Y.S.Luh, "Conventional Controller Design for Industrial Robots - A Tutorial, "IEEE Trans.Sys., Man, Cybern., vol. SMC-13, pp. 298-316, May/June 1983

(2) J.Y.S.Luh and C.S.Lin, "Scheduling of Parallel Computation for a Computer-Controlled Mechanical Manipulator",IEEE Trans. Sys., Man, Cybdrn., vol. SMC-12, pp. 214-234, March/April 1982.

(3) J.M.Hollerbach, "A Recursive Lagrangian Formulation of Manipulator Dynamics and a Comparative Study of Dynamics Formulation Complexity," IEEE Trans. Sys., Man, Cybern., vol. SMC-10, pp. 730-736, November 1980

(4) L.S.Gang, "Jedan postupak paralelnog procesiranja matematičkog modela manipulacionih robota", Zbornik, 4.jug.simpozij o uporabni robotiki, Vrnjačka Banja, pp. 42-55, maj 1985.

(5) J.Y.S.Luh, M.W.Walker and R.P.C.Paul, "On Line Computational scheme for Mechanical Manipulators", ASME Trans. Journal of Dynamic Systems, Measurement and Control, Vol. 102, No. 2, pp. 69-76, June 1980.

(6) N.Kasahira and S.Narita, "Load Distribution Among Real Time Control Computers Connected via Communication Media", pp. 194-199, Proceedings of 9th IFAC World Congress, Budapest, July 1984.

(7) R.P.Paul, "Robot Manipulators: Mathematics Programming and Control", MIT Press, Cambridge, 1981.

(8) P.Coiffet, "Modelling and Control", Kogan Page, London 1983.

(9) H.Kirrmann, "Events and Interrupts in Tightly Coupled Microprocessors", IEEE Micro, Vol.5, No.2, pp.53-66, Feb. 1985.

(10) W.Fischer, "IEEE P1014 - A Standard for the High-Performance VME Bus", IEEE Micro, Vol. 5, No.2, pp. 31-41, februar 1985

(11) "VME Bus Specification Mannual, Rev. B", VME bus Manufacturers Group, August 1982.

O P E R A C I J S K I S I S T E M V M / S P

J A N A J A M S E K

I N T E R T R A D E L J U B L J A N A

UDK : 681.3.06

POVZETEK: Prispevek govori o IBMovem operacijskem sistemu VM/SP. Glavna značilnost VM/SPja je simulacija več navideznih računalnikov, od katerih je vsak po funkcijah ekvivalenten realnemu računalniku, tako v smislu aparaturne kot programske opreme. V navideznih računalnikih lahko tečejo vglavnem isti operacijski sistemi kot na realnih računalnikih; interaktivni operacijski sistem CMS, ki je del VM/SPja, pa je namenjen posebej za navidezni računalnik.

ABSTRACT: The article represents IBM operating system VM/SP. The main characteristic of VM/SP is simulation of virtual computers (virtual machines), equivalents of real computers as far as hardware and software is concerned. In general in virtual machines run the same operating systems as in real ones; interactive operating system CMS, a part of VM/SP, is designed to run in a virtual machine.

1. UVOD

Zaradi vse večjih zahtev računalniških uporabnikov, se kažejo potrebe po boljšem izkoriščanju računalnika, kot pa ga nudita tehnika hkratnega izvajanja programov in sprotnega dela. VM/SP je operacijski sistem, ki združuje obe ti tehniki, in sicer tako, da nudi vsakemu uporabniku simulacijo računalnika. Tako lahko posamezni uporabnik izkoristi vse možnosti, ki jih računalnik nudi. Vsak simulirani računalnik ali navidezni računalnik uporablja svoj operacijski sistem, medtem ko sistemska sredstva, potrebna za delo vseh navideznih računalnikov, razdeljuje VM/SP. V računalniku z operacijskim sistemom VM/SP torej deluje hkrati več navideznih računalnikov, ki imajo lahko različne operacijske sisteme, opravljajo paketne ali sprotne obdelave ter uporabljajo hkratno izvajanje programov.

Okrajšave:

NR - Navidezni računalnik

VM/SP - Virtual Machines/System Product

CMS - Conversational Monitor System

CP - Control Program

DOS - Disk Operating System

V/I - Vhodno/izhodni

2. SPLOŠNO O VM/SP

2.1 Komponente VM/SP

Bistveni komponenti operacijskega sistema VM/SP sta CP (Control Program) in CMS (Conversational Monitor System). CP ali Krmilni program se pri začetni naložitvi operacijskega sistema napolni v pomnilnik in tam ostane ves čas delovanja VM/SP. Njegova naloga je, da omogoča obstoj in delovanje navideznih računalnikov ter čimbolj ekonomično porazdeljuje sistemska sredstva kot so procesorski čas, količina pomnilnika, sistemski

programi in datoteke med posameezne navidezne računalnike.

CMS je sam zase operacijski sistem, predviden za delo v posameznem navideznem računalniku. Omogoča naprimer: prevajanje in izvajanje programov, pisanih v assemblerju ali drugih programskih jezikih, vnos podatkov, delo z datotekami itd.

Med ostalimi komponentami VM/SP omenimo naslednje:
RSCS (Remote Spooling Communications Subsystem) deluje v enem izmed NR in prenaša datoteke na terminale ali tiskalnike, oddaljeno priključene preko telefonske linije.
IPCS (Interactive Problem Control System) se izvaja pod kontrolo CMSa in ima naslednji nalogi: Standardizira poročanje o problemih ter tako identificira ponavljajoče se probleme ter omogoča pregleden prikaz oziroma izpis pomnilnika.
DMS (Display Management System): Izvaja se pod kontrolo CMSa in omogoča sprotno oblikovanje zaslonskih slik. Izgotovljene zaslonske slike se nato lahko uporabljajo v raznih aplikacijah pod CMSom.
DCF (Document Composition Facility): Tudi ta teče pod kontrolo CMSa, omogoča pa oblikovanje besedila ter izpis na različne izhodne enote, kot so tiskalnik, terminal in diski.

2.2 Navidezni računalnik

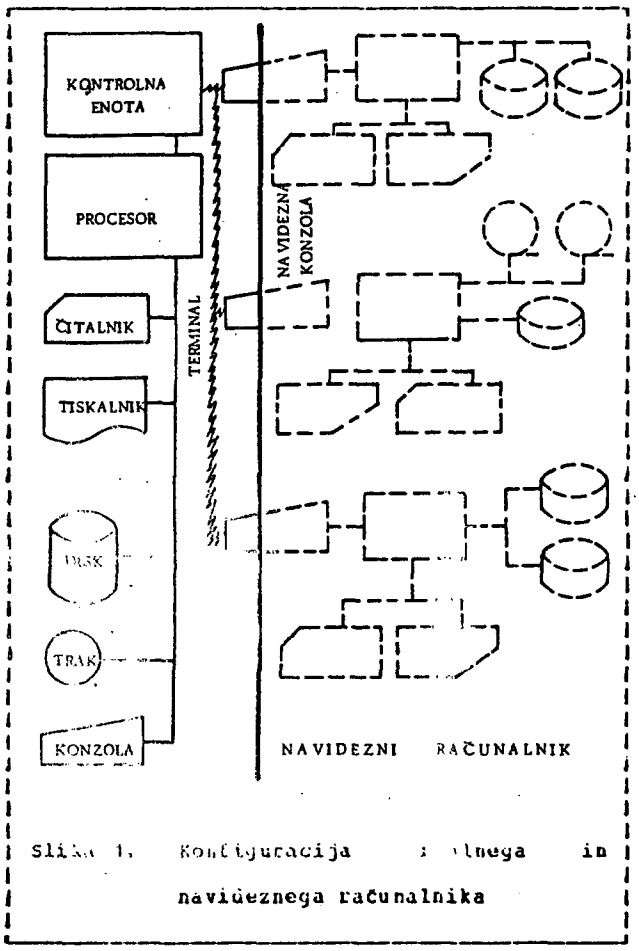
Kot že omenjeno, je bistvena značilnost VM/SPja to, da omogoča obstoj navideznega računalnika. Ta je po funkcijah enak realnemu računalniku, saj ima simulirano tako aparaturno kot programsko opremo. Pod nadzorstvom CPja lahko istočasno dela poljubno število NRov, odvisno od velikosti pomnilnika. Ni treba, da so vsi enaki in tudi njihovo število se lahko spreminja, posamezni NR se namreč lahko vključi ali izključi. Naprimer: V začetku sta vključena 2 NRa, od katerih ima vsak simuliran procesor, pomnilnik v velikosti 4 Mb, 8 diskov, 3 magnetne trakove, čitalnik, tiskalnik ter, kot programsko opremo, operacijski sistem DOS; obenem je vključeno 15 NRov, ki imajo simuliran procesor, pomnilnik v velikosti 512 Kb, 4 diske, čitalnik in tiskalnik, kot programsko opremo pa imajo CMS. V teku dela izključimo 1 NR z operacijskim sistemom DOS in vključimo 10 NRov z op. sistemom CMS.

Posameznemu NRu lahko sproti spreminjamo bodisi aparaturno, bodisi programsko opremo, npr. dodamo nekaj enot diskov, povečamo ali zmanjšamo pomnilnik, naložimo vanj drug operacijski sistem itd.

V realnem računalniku torej lahko pod VM/SPjem dela hkrati več NRov, po potrebi pa jih vključujemo in izključujemo. V VM/SPju obstaja poseben imenik, v katerem so vpisani vsi NRi, in sicer njihova imena, gesla in konfiguracija.

V vsakem NRu je simuliran t.i. navidezni procesor, ki izvaja instrukcije in sprejema prekinitve. Ker deluje v VM/SPju več NRov hkrati, je treba hkrati simulirati več navideznih procesorjev - to omogoča deljena izraba realnega procesorja (timesharing).

Ravno tako se v vsakem NRu simulira pomnilnik (navidezni), in sicer v velikosti od 8 Kb do 16 Mb; velikost je vpisana v prej omenjenem imeniku, v teku dela pa jo je mogoče spreminjati. Ker simulacijo pomnilnika omogoča princip stranjenja, lahko velikost navideznega pomnilnika preseže velikost realnega



Slika 1. Konfiguracija realnega in navideznega računalnika

pomnilnika. Primer: V računalniku s pomnilnikom, velikim 4 MB, je pod VM/SP-jem več NR, od katerih ima vsak navidezni pomnilnik velikosti 16 MB. Seveda pa velja: čim večji so navidezni pomnilniki, tem počasneje poteka delo na VM/SP-ju. Kontrolni program (CP) omogoča zaščito realnega pomnilnika, posameznemu NR namreč ni dosegljiv del pomnilnika, ki pripada drugemu NR; obstajajo pa posamezni odseki, ki si jih NR delijo med seboj.

Konzola NR ima 3 glavne funkcije:

1. V njej se izvaja začetno polnjenje operacijskega sistema, dajejo ukazi START in STOP za aktiviranje in deaktiviranje NR, STORE in DISPLAY za delo s simuliranimi registri, itd.

2. Prek navidezne konzole lahko uporabnik spreminja lastnosti NRa, npr. velikost pomnilnika, način dela navideznega procesorja in število vhodno/izhodnih enot.

3. Komunikacija z aplikacijo, ki se izvaja v enem ali več NRov, poteka z navidezne konzole.

Kot konzola NRa služi terminal, lahko pa NR deluje brez konzole, v ti. izključenem stanju (disconnect), ki ga povzroči uporabnik s posebnim ukazom.

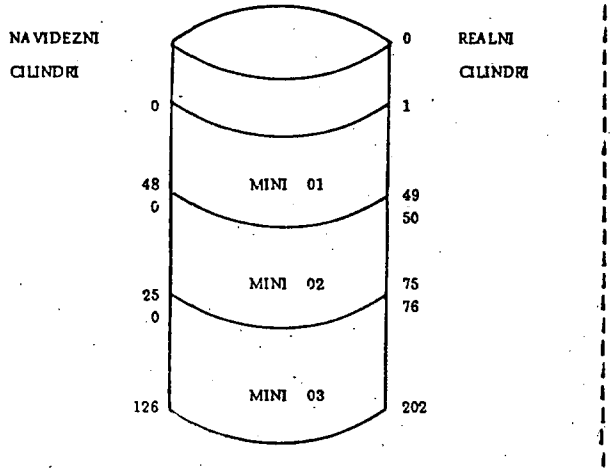
2.3 Navidezne vhodno/izhodne enote

V NRu je možno definirati navidezne vhodno/izhodne enote istega tipa kot so vhodno/izhodne enote pri realnem računalniku, in sicer pod kontrolo operacijskega sistema v NRu. Opisane so v imeniku, njihovo število in tip pa se lahko v teku dela spreminja. Vsaka navidezna V/I enota ima svoj naslov, ki se imenuje navidezni naslov. Ob zahtevi za čitanje ali pisanje, spremeni Krmilni program navidezni naslov v ustrezni realni naslov in napravi tudi druge potrebne spremembe, npr. preračuna lokacijo podatkov na disku.

Ločimo več tipov navideznih V/I enot:

Navidezni diski

Navidezni diski se imenujejo minidiski, ker navadno zavzemajo le del realnega diska. Lahko so različnih velikosti in tipov; uporabljajo jih razni operacijski sistemi. Slika prikazuje 3 minidiske na enem realnem disku, številke na levi pomenijo navidezne cilindre za posamezni minidisk, številke na desni pa realne cilindre. Posamezni minidisk lahko pripada enemu ali več NRom, po drugi strani pa je NR lahko brez



Slika 2. Realni disk in navidezni diski - minidiski

minidiskov, lahko pa ima enega ali več le-teh. Krmilni program vodi evidenco za vsak posamezni minidisk, in sicer na katerem realnem disku se nahaja, njegovo realno lokacijo (začetni in končni cilinder), kateremu NRu pripada in kateri je njegov navidezni naslov. Če minidisk pripada več NRom, ima lahko v vsakem izmed njih drugačen navidezni naslov. Minidisk zavzema npr. del realnega diska z naslovom 1C0, in sicer od cilindra 50 do 69. Pripada NRu A, v katerem ima naslov 191, obenem pripada tudi NRu B, kjer ima naslov 234 ter NRu C, kjer ima naslov 007. Če izda NR B instrukcijo za čitanje podatkov s 3. cilindra minidiska 234, jo Krmilni program priredi v instrukcijo za čitanje z diska 1C0, cilinder 53.

V VM/SPju obstajajo služnostni programi za označevanje in spreminjanje minidiskov, zaščita podatkov pa je dosežena na naslednji način: Lastnik minidiska (uporabnik NRa, ki mu minidisk pripada) navede, ali ga bo uporabljal samo za čitanje (read) ali za pisanje in čitanje (read-write) ali pa dovoli več NRom, da nanj pišejo (multi-write). Če želi kak drug uporabnik pristop do tega minidiska, mora poznati ustrezna gesla, pač glede na to, ali želi samo čitati ali čitati in pisati itd. Med delom je mogoče ustvariti začasne minidiske, ki trajajo le do izključitve NRa, zanje ima VM/SP predviden poseben prostor.

Navidezni čitalniki in tiskalniki

Vsak NR ima navadno svoj čitalnik, luknjalnik ter enega ali več tiskalnikov, vsi so seveda navidezni. Dejansko predstavlja navidezni čitalnik ali tiskalnik del vmesnega prostora na disku (spool). Po potrebi se posameznemu NRu za nekaj časa dodeli realni tiskalnik, na katerega lahko direktno izpisuje izhodne podatke. Kadar pa tiskalnik ne pripada nobenemu NRu, lahko sprejema podatke iz raznih navideznih tiskalnikov in jih po vrsti izpisuje. Ista pravila veljajao za realni čitalnik in luknjalnik.

NRi si lahko izmenjujejo podatke, uporabljajoč navidezne enote: NR A s posebnim ukazom usmeri svoj luknjalnik k čitalniku NRa B, s tem doseže, da se podatki, ki jih luknja, zapišejo v čitalnik NRa B.

Navidezni magnetni trakovi

NRu lahko dodelimo magnetni trak, ki postane začasno njegova last. NR mu lahko dodeli naslov, vsi zapisi nanj pa potekajo prek Krmilnega programa. Primer: NRu dodelimo magnetni trak z naslovom 340, ki ga ta spremeni v navidezni naslov 181. Krmilni program si zapomni realni in navidezni naslov traka ter podatek, kateremu NRu trak pripada. Vse instrukcije, s katerimi NR piše na trak 181, CP prevede v ustrezne instrukcije za trak 340. Zato je mogoče, da ima več NRov trakove z naslovi 181, ki pa pripadajo različnim realnim naslovom.

2.4 Sistemski imenik in vključitev NRa

Sistemski imenik

Za NR je značilna njegova definicija v sistemskem imeniku (Directory), tam ima vsak Nk opis, ki določa njegove lastnosti. Opis med drugim vsebuje:

- identifikacijo in geslo, potrebno za dostop do Nka
- začetno in največjo možno velikost pomnilnika NRa
- opis ukazov, ki so dovoljeni za uporabnika NRa
- navidezne tiskalnike in čitalnike
- minidiske z navideznimi naslovi
- povezave z minidiski drugih NRov
- parametre, ki vplivajo na prioriteto NRa pri dodeljevanju procesorskega časa in količine realnega pomnilnika

Slika 3 prikazuje opis v sistemskem imeniku.

```
USER alfa XXXXXXX 960K 2M G
 ACCOUNT 910030 alfa
 IPL CMS PARM AUTOCR
 CONSOLE 009 3215
 SPOOL 00C 2540 READER *
 SPOOL 00D 2540 PUNCH  A
 SPOOL 00E 1403 A
 LINK MAINT 190 190 RR
 LINK MAINT 19E 19E RR
MDISK 123 3350 527 002 CMSWRK M XX XX XX
MDISK 191 3350 246 005 CMSWRK M XX XX XX
```

Slika 3. Opis v sistemskem imeniku

Sistemski imenik je zapisan na sistemskem disku in je pod kontrolo CP. Ko želi uporabnik začeti delo z določenim NRom, se prijavi (izvede vlogon). CP poišče v sistemskem imeniku zahtevani Nk, rezervira ustrezno količino realnega pomnilnika in zgradi potrebne kontrolne bloke (zapis v pomnilniku, ki ga CP uporablja za nadzor nad NRom). Terminal, na katerem se je uporabnik prijavil, postane konzola NRa.

Z ustreznimi ukazi lahko uporabnik spreminja karakteristike NRa, zapisane v sistemskem imeniku, vendar te spremembe veljajo le do njegove odjave.

Prijavljanje

Kadar želi uporabnik delati z določenim NRom, se prijavi, v terminal vtipka identifikacijo NRa in takoj zatem geslo. CP oboje preveri, zgradi NR po opisu v imeniku in o tem prek terminala obvesti uporabnika, ki lahko nato s posebnim ukazom naloži v NR določen opracijski sistem.

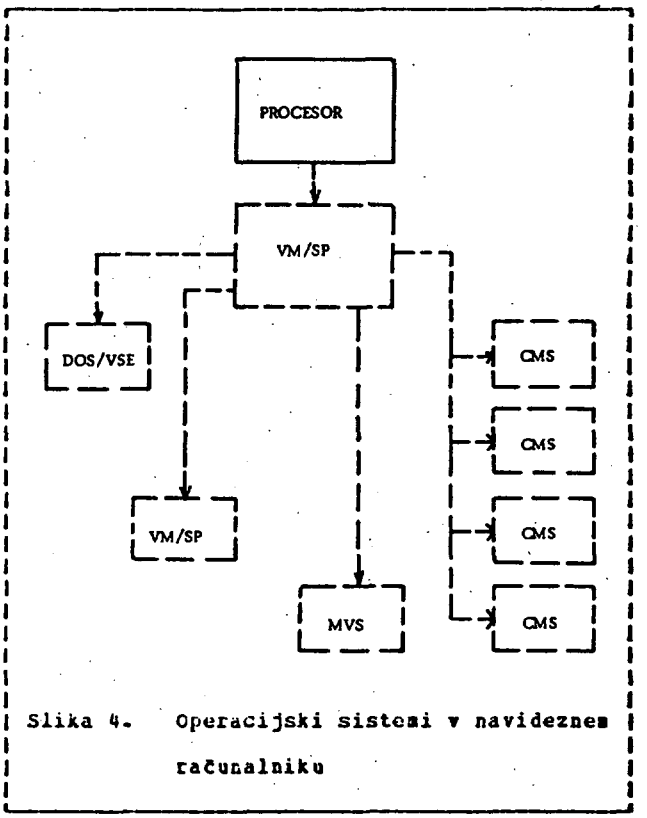
Kot že omenjeno, se lahko NR nahaja v izključenem stanju, kar pomeni, da nima konzole-terminala, njegovo delo pa ostaja nespremenjeno. Potem ko se je prijavil ter z ukazi naložil NRu določeno delo, se uporabnik po potrebi lahko izključi (disconnect) in uporabi terminal za druge namene.

2.5 Operacijski sistemi v navideznem računalniku

V VM/SPju delujejo NRi neodvisno od drugega, zato lahko naložimo v vsak NR lasten operacijski sistem, kot na primer:

- CMS
- DOS/VSE - Disk Operating System / Virtual Storage Extended
- MVS - Multiple Virtual Storage
- VM/SP

V posameznih NRih lahko delujejo tudi različne verzije ali pa kopije istega operacijskega sistema.



Slika 4. Operacijski sistemi v navideznem računalniku

Če se nahaja v NRu sistem, ki omogoča sprotno delo prek terminalov, potem en terminal deluje kot konzola NRka, medtem ko druge priključimo na NR s posebnim ukazom (DIAL).

Večina zgoraj navedenih operacijskih sistemov uporablja stranjenje. Če delujejo v NRu, pride do dvojnega stranjenja, namreč stranjenje op. sistema in stranjenje VM/SPja, vendar se to z določenimi parametri lahko prepreči in uporablja le enojno stranjenje.
Primer operacijskih sistemov v NRu:
pod VM/SP je 30 NR; v NR ALFA naložimo op. sistem DOS/VSE, ki se uporablja izključno za paketne obdelave; v NR BETA je DOS/VSE, namenjen za sprotno delo z aplikacijami; NR GAMA uporabljamo za preskušanje nove verzije DOS/VSE in NR DELTA za delo z novo verzijo VM/SP; ostali NR imajo operacijski sistem CMS, v njih se izvajajo aplikacije, podatke pa si izmenjujejo med seboj ter z NRi ALFA, BETA in GAMA.

2.6 Uporaba operacijskega sistema VM/SP

Navedimo nekaj načinov, kako se lahko izrabijo možnosti, ki jih nudi VM/SP.

- Sistemski programer lahko uporablja kopijo delujočega operacijskega sistema VM/SP v enem izmed NRov. Tam testira npr. nove naslove diskov, razmestitev datotek za stranjenje, sistemske programe, ki jih sam napiše, itd. V NRu tudi laže zasleduje napake na operacijskem sistemu, kot v realnem računalniku.
- Aplikativni programerji, ki pišejo programe za operacijski sistem DOS, imajo možnost svoje delo testirati v enem izmed NRov ter

ga po končanem testiranju prenesti v NR z DOSom.

- Tudi CMS omogoča razvoj novih aplikacij, npr. s področja linearnega programiranja, računalniške grafike itd.

- Uporabniki lahko v CMSu urejujejo datoteke, ki jih nato pošljejo na izvajanje v op. sistem DOS ali MVS.

3. KRMILNI PROGRAM - CP

Kot že omenjeno, ja CP del VM/SPja, ki omogoča obstoj navideznih računalnikov, in sicer tako, da nadzoruje procesorski čas, realni pomnilnik, V/I enote itd. ter jih porazdeljuje med posamezne NRe.

3.1 Navidezni procesor

Uporabniku NRa se zdi, da se procesorske instrukcije izvajajo v NRu, seveda pa je realni procesor ta, ki instrukcije dejansko izvaja. Obstoj več navideznih procesorjev omogoča tehnika časovnega porazdeljevanja (time-slicing), saj dobivajo NRi periodično na razpolago procesor, in sicer za zelo kratek časovni presledek, navadno nekaj milisekund. Kako pogosto in kako veliko DčasovnoD rezinoD bo NR dobil, odloča CP, glede na število terminalskih prekinitev, ki jih je NR povzročil v času izrabe predhodne časovne rezine. Če je bilo število veliko, bo NR dobil manjše DrezineD, a bolj pogosto, v nasprotnem primeru pa večje v redkejših časovnih razdobjih. Na ta način CP sproti porazdeljuje navidezne računalnike na spretno in paketno usmerjene. S posebnimi ukazi CPju lahko vplivamo na dodeljevanje procesorskega časa in s tem dajemo prednost enemu ali več NRom.

Poleg tega načina odločanja pa CP upošteva tudi uporabo pomnilnika in V/I operacije. Če NR, ki pričakuje svoj "časovno rezino", še nima na razpolago zadostne količine pomnilnika, ali če čaka na izvršitev V/I operacije, tudi za dodelitev DrezineD ne pride v poštev NRi, ki se potegujejo za sistemska sredstva, konkurirajo torej najprej za dodelitev pomnilnika, nato pa za procesorski čas.

3.2 Pomnilnik v navideznem računalniku

Vsak NR ima svoj simulirani pomnilnik - navidezni pomnilnik, katerega velikost je določena v sistemskem imeniku. Ker navidezne pomnilnike ustvarja in nadzoruje CP, so le-ti lahko večji od realnega pomnilnika.

Način, kako CP omogoča simulacijo pomnilnika, je na kratko naslednji: Realni pomnilnik je logično razdeljen na dele, velike 4 Kb, ki jih imenujemo okvirji (frames). Pomnilnik NRa pa je razdeljen na segmente (64 Kb) in znotraj segmentov na strani velikosti 4 Kb. Program, ki se zapiše v navidezni pomnilnik, se dejansko zapiše na disk. CP dinamično prevaja naslove navideznega pomnilnika v naslove realnega in prenese v realni pomnilnik samo tisto stran, ki je tranutno potrabna za izvajanje kake instrukcije. Kadar v realnem pomnilniku zmanjka razpoložljivih okvirjev, CP odloži trenutno nepotrebne strani na disk in s tem pridobi nekaj prostora. Ker CP pri prevajanju naslovov uporablja za vsak NR svoje tabele, omogoča s tem obstoj več navideznih pomnilnikov istočasno; v vsakem NR obstajajo tudi navidezni registri, tako naslovljivi kot interni.

Ker se strani stalno prenašajo iz pomnilnika na diske in obratno, je treba za diske, ki služijo stranjenju, izbrati tiste, ki omogočajo najhitrejše izvajanje vhodno/izhodnih operacij. Pri prenosu v pomnilnik strani posameznih NRov konkurirajo za razpoložljive okvirje, ki jih mora biti vedno nekaj na razpolago. Ko zaloga razpoložljivih okvirjev pade pod neko določeno mejo, CP izvrže trenutno neaktivne strani in jih, v primeru, da so bile medtem spremenjene, zapiše ponovno na disk.

Stranjenje (paging) se vrši na zahtevo CP (Demand paging). Stran se namreč ne prepiše z diska v pomnilnik, dokler ni tam potrebna za izvajanje instrukcij navideznega računalnika.

3.3 Delo z navideznimi vhodno/izhodnimi enotami

Ko se NR vključi, dobi avtomatično tiste vhodno-izhodne enote, ki so definirane v imeniku, lahko pa si med delom dodaja nove ali odvzema trenutno nepotrebne. Navidezne vhodno/izhodne enote so pod nadzorstvom operacijskega sistema v NRu. Te enote so lahko:

- povsem dodeljene NRu (dedicated)
- razdeljene med več NRov (naprimer minidisk, ki ga lahko čita več NRov)
- enote vmesnega prostora (navidezni čitalnik, tiskalnik itd.), ki predstavljajo dejansko del prostora na disku.

Kadar operacijski sistem krmili realni računalnik, se V/I operacije začnejo tako, da program pošlje operacijskemu sistemu zahtevo za instrukcijo SIO (Start Input Output) na določeni V/I enoti, ta jo izvede in ugotavlja eventualne napake na enoti. V NRu opravlja cp. sistem iste naloge s to razliko, da uporablja navidezni naslov enote in navidezno lokacijo v pomnilniku. Ko op. sistem izda instrukcijo SIO, CP ti dve količini prevede, ostale instrukcije, ki se nanašajo na V/I operacijo, pa shrani v posebnem področju. Vse strani, ki so potrebne za prenos podatkov, vnese CP v realni pomnilnik, kjer ostanejo, dokler ni V/I operacija izvršena. Naslov navidezne enote prevede v realno; če gre za minidisk, pa tudi ustrezno priredi pozicijo, kamor se podatki pišejo, oziroma od koder se čitajo. Če pride do napak na V/I enoti, CP ne prevzame odkrivanja in morebitnega popravljanja, razen najosnovnejših funkcij, pač pa to prepusti op. sistemu v NRu.

Navidezni diski ali minidiski lahko pripadajo več NRom, kar se doseže z vpisom v sistemski imenik ali pa s CP ukazom. Uporabnik, ki želi priti do minidiska kakega drugega NRa, navede, ali želi pristop za čitanje ali za čitanje in pisanje, poznati pa mora tudi ustrezna gesla. Pri vsakem posegu na tako pridobljen minidisk CP preveri, če poseg spada v okvir navedenih omejitev.

Obstajajo tudi začasni mnidiski. Uporabniku, ki zahteva tak minidisk, dodeli CP del za to namenjenega prostora, uporablja pa ga lahko do odjave.

Prevajanje naslovov pri navidezni V/I operaciji se poenostavi z uporabo posebne instrukcije DdiagnoseD. CMS, ki je namenjen za delo v NRu, uporablja skoraj vedno instrukcijo DdiagnoseD namesto standardne V/I operacije. V/I operacije, ki jih izvaja CP sam, npr. za prenos strani, niso podvržene prevajanju in se izvajajo direktno.

3.4 Delo z vmesnim pomnilnikom

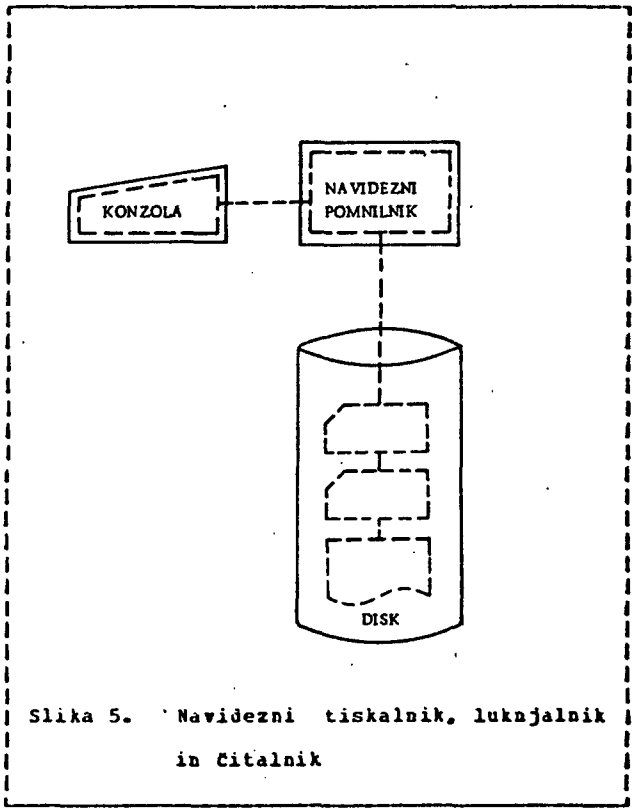
NAi si med seboj lahko delijo realne V/I enote. Podatki, ki jih posamezni NAi tiskajo, se namreč zapisujejo na navidezne tiskalnike (na vmesni pomnilnik) in se od tam po potrebi prenesejo na realni tiskalnik, ki na ta način služi več NAom. Navidezni tiskalniki, čitalniki in luknjalniki so pod nadzorstvom CP in kadar izstavi NA za tako enoto instrukcijo SIO, jo CP prilagodi na podoben način kot pri minidiskih.

Dodajanje novega vmesnega pomnilnika in prenašanje podatkov

Ko se vmesni pomnilnik na disku napolni, je delovanje navideznih V/I enot onemogočeno. Operater v tem primeru briše nekaj datotek z vmesnega pomnilnik ali pa navideznim enotam dodeli dodatno področje. Disk, na katerem se vmesni pomnilnik nahaja, pripada CPju, zato pri zapisovanju ne prihaja do prevajanja. Komunikacija med NAi je omogočena s prej omenjenim usmerjanjem navideznega tiskalnika v čitalnik ter prenašanjem podatkov na ta način.

Pri NAih, ki delajo v izključenem stanju, konzolnih sporočil navadno ne vidimo, vendar pa vmesni pomnilnik omogoča usmerjanje konzolnih zapisov direktno na disk, tako da jih kdaj pozneje lahko preberemo.

Pri komunikacijskem omrežju se vmesni pomnilnik uporablja na naslednji način: Daljinsko priključeni terminali služijo, tako kot lokalni, za konzole NIov, medtem ko daljinsko priključeni tiskalniki pripadajo posameznemu, zanje določenemu NAu. Podatki, ki jih NAi želijo poslati na daljinske priključke (bodisi drugim NAom, bodisi tiskalnikom), gredo najprej v vmesni pomnilnik omenjenega NAa; pro-



Slika 5. Navidezni tiskalnik, luknjalnik in čitalnik

gram, ki v njem teče, pa jih usmeri na želene lokacije.

4. OPERACIJSKI SISTEM CMS

CMS (Conversational Monitor System) je poleg Krmilnega Programa glavna komponenta VM/SP. Skupaj s CP uporablja tehniko časovnega porazdeljevanja in s tem omogoča sprotno delo. Za razliko od ostalih op. sistemov, ne more delovati samostojno v realnem računalniku, pač pa samo v NAu, in to zaradi izvajanja V/I operacij (Diagnose), ki potrebuje sodelovanje CP.

Glavne naloge CMSa so ustvarjanje, urejevanje in prenos datotek ter prevajanje, preskušanje in izvajanje programov.

Glede konfiguracije in lastnosti NKa CMS nima kakih posebnih zahtev. Ker ponavadi dela istočasno mnogo NKov s CMSom, se najbolj aktivni deli CMSa, kot je npr. jedro (Nucleus) nadzorovalnika, ne naložijo v vsak NR, pač pa si jih NKi med seboj delijo; s tem se prihrani prostor v realnem pomnilniku. Uporabniki CMSa imajo vsaj dva minidiska: Prvi (S-disk) služi za shranjevanje CMSovih sistemskih programov, zato morejo z njega samo čitati, drugi (A-disk) pa je namenjen za ustvarjanje in urejevanje datotek ter delo s programi.

4.1 Datoteke v CMSu

CMS razdeli A-disk v bloke stalne dolžine 800 ali 1600 KL. Tako formatiziran minidisk se nato uporablja za datoteke z zapisi stalne ali spremenljive dolžine. Tudi CMS datoteke na magnetnem traku imajo lahko poljubno dolge zapise stalne ali spremenljive dolžine.

Služnostni programi CMSa omogočajo formatiziranje minidiskov, pisanje in čitanje s trakov, kopiranje, preimenovanje in brisanje datotek, itd. Obstajajo posebni služnostni programi za obdelavo sistemskih bibliotek, ki vsebujejo makro-definicije ali programe.

Uporabnik CMSa ima poleg S-diska lahko do 25 minidiskov vsakemu je dodeljena ena črka. Ime datoteke v CMSu sestoji iz treh komponent:
filename - ime, ki ga določi uporabnik
filetype - ime, ki označuje tip datoteke
filemode - ime minidiska (ena črka)
Na primer: o datoteki z imenom TEST COBOL A vemo, da vsebuje program v cobolu, in da se nahaja na minidisku A. Če pri delu z datoteko ne navedemo vseh treh komponent imena, CMS upošteva pravila filetypa in iskanja po abecednem zaporedju minidiskov. Primer: Uporabnik želi prevesti z assemblerjem program v datoteki TEST1; v ta namen izda ukaz ASSEMBLE TEST1. CMS išče datoteko TEST1 ASSEMBLE, in sicer najprej na minidisku A, nato po vrstnem redu na B, P, S itd. Da se čas iskanja čimbolj zmanjša, se dodelijo najbolj uporabljanim minidiskom najnižje črke. Datoteke v CMSu navadno ne zavzemajo nepretrganega področja na minidisku, vendar pa ima vsak minidisk imenik datotek, ki vsebuje kazalce na verige zapisov, ki tvorijo posamezno datoteko. Vsebuje tudi informacije o formatu in dolžini zapisov ter o prostoru, ki ga posamezna datoteka zaseda.

CMS vsebuje pristopne metode kot so VSAM (Virtual Storage Access Method) ter metode za sekvenčni in direktni pristop.

4.2 Ukazi v CMSu

CMS ukazi služijo uporabniku za delo z datotekami in programi, na primer:
COPY - kopiranje datoteke
ERASE - brisanje
SORT - sortiranje zapisov datoteke
XEDIT - urejevanje
COBOL - prevajanje programov v cobolu
PASCAL - prevajanje programov v pascalu
LOAD - naložitev strojne kode v navidezni pomnilnik
START - pričetek izvajanja
itd.
CMS vzdržuje urejeno listo ukazov, zato prepozna ukaz tudi v skrajšani obliki, naprimer P za PASCAL. Uporabnik lahko določi omejitve okrajšav in si definira svoje besede (sinonime) za posamezne ukaze. Poljubnemu programu v strojni kodi, ki se nahaja na minidisku, lahko uporabnik dodeli ime in ga uporabi kot CMS ukaz, s katerim program izvede.

CMSov interpreter REXX omogoča povezovanje CMS-ukazov, pogojno izvajanje, izvajanje v zankah, itd. Poleg tega omogoča aritmetične in logične operacije, zapis števil z drsno vejico, interne in eksterne podprograme, večdimenzionalne spremenljivke in vgrajene funkcije.

5. IZBOLJŠAVA PERFORMANS

Ko govorimo o performansah računalnika, mislimo s tem na

- odzivni čas pri sprotnem delu
- čas, ki ga porabijo paketne obdelave,

izboljšanje performans pomeni izboljšanje obeh omenjenih časov. Idealni pristop k izboljšanju performans poteka tako, da si uporabnik računalnika (tu govorimo o realnem računalniku) postavi kot zahtevo določeno mejo odzivnega časa pri sprotnem in paketnem delu. Izboljšave se nato dosežejo z določenimi spremembami v razporeditvi programske opreme, s čim prikladnejšo razporeditvijo obdelav ter eventualno z nakupom dodatne računalniške opreme.

V VM/SPju dosežemo izboljšavo performans s spremembami na vhodno/izhodnem sistemu ter z boljšo izrabo procesorskega časa in pomnilnika. Najvažnejše so spremembe na vhodno/izhodnem sistemu, prenos podatkov na diske namreč zahteva zelo veliko časa v primerjavi z izvajanjem procesorskih instrukcij. Kanali so vhodno/izhodne enote, ki prevzamejo prenos podatkov in s tem razbremnijo procesor. Navadno dela en kanal z nekaj diski, in če je preveč zaseden, oziroma če so kanali neenakomerno obremenjeni, pride do zastojev. Zato je treba močno obremenjene datoteke porazdeliti čimbolj enakomerno glede na kanale. Tipičen primer take datoteke je prostor, kamor se izpisujejo strani navideznega pomnilnika. Navadno kreiramo več takih datotek in jih razporedimo na diske, priključene na različne kanale.

Izrabo procesorskega časa izboljšamo z vključitvijo posebne funkcije VMASSIST, ki najpogostejše operacije, npr. prevajanje naslovov minidiskov, prenese v mikrokodo. Če dela v navideznem računalniku operacijski sistem z velikim številom programov v skupni obdelavi, se doseže boljša izraba procesorskega časa s porazdelitvijo programov na več op. sistemov v različnih NR. S tem namreč dosežemo, da razpečevalnet funkcije VM/SPja, ki bolje izkoriščajo procesor, prevzamejo skupne obdelave programov.

Izrabo pomnilnika izboljšamo s tem, da omejimo velikosti navideznih pomnilnikov v posameznih NR. S tem tudi zmanjšamo zahteve po stranjenju. Čim več CMS sistemskih programov damo v področje, ki se istočasno deli med NR, saj s tem ravno tako prihranimo na stranjenju.

Če želimo dati prednost enemu izmed NRov, mu s posebnimi ukazi povečamo prioriteto pri razpečevanjuD in stranjenju. Seveda pa s tem upočasnimo delo ostalih NRov.

6. LITERATURA

- Performance Measurement Tools for VM - System Journal Reprint
- Tuning a Virtual Storage System - System Journal Reprint

S I S T E M Z A I N D U K T I V N O U Č E N J E A S I S T E N T

IGOR KONONENKO (1), IVAN BRATKO (1,2), EGIDIJA ROŠKAR (1)

(1) Fakulteta za elektrotehniko, Ljubljana
(2) Inštitut Jožef Stefan, Ljubljana

UDK : 681.3:159.953

POVZETEK

Iz Quinlanovega sistema ID3 za generiranje odločitvenih dreves na osnovi znanih primerov smo razvili sistem, ki smo ga imenovali ASISTENT. Osnovni algoritem smo izpopolnili tako, da omogoča uporabo nepopolnih podatkov, obravnavanje zveznih atributov, klasificiranje v kombinaciji z Bayesovim verjetnostnim principom, avtomatsko izbiro dobrih učnih primerov, enakovredno obravnavanje večvrednostnih atributov z binarno gradnjo in rezanje nezanesljivih delov odločitvenih dreves. Vse izpopolnitve imajo teoretično osnovo. Njihova pravilnost je potrjena z eksperimenti v 5 različnih medicinskih domenah. Odločitvena drevesa so manjša in s tem razumljivejša za uporabnika, hkrati pa so natančnejša pri klasifikaciji novih primerov. ASISTENT je v vseh medicinskih domenah dosegel razred natančnosti zdravnikov specialistov. Zgrajena odločitvena drevesa so razmeroma razumljiva in uporabnik lahko iz njih razbere določene zakonitosti iz svoje domene. Lahko se uporabljajo brez računalnika, npr. kot priročnik za diagnosticiranje. ASISTENT je implementiran v Pascalu in obsega pribl. 5000 vrstic izvorne kode. Trenutna implementacija omogoča hitro generacijo in testiranje pravil. Programska koda je komentirana in dokumentirana z navodili za uporabnika in z navodili za programerja. Sistem je zrel za rutinsko uporabo v poljubni ustrezno definirani problemski domeni.

AN INDUCTIVE LEARNING SYSTEM - ASSISTANT

ASSISTANT is an inductive learning system for constructing the decision trees from examples. It is derived from Quinlan's ID3 (Quinlan 79,79a,82). Extensions to ID3 include: multivalued attributes, continuous attributes, incompletely specified learning examples, binary construction, automatic selection of good training examples, tree-pruning with maximal classification precision principle and plausible classification in combination with statistical method, based on Bayesian principle. ASSISTANT was applied to a number of learning problems in medical diagnosis and prognosis: location of primary tumor, prognosis in breast cancer, lymphographic investigation, hepatitis and lower urinary tract dysfunctions. Some earlier experiments are described in (Kononenko et. al. 84, Roškar et. al. 85). A comparison with a statistical method based on the Bayesian principle is presented. By the comprehensibility criterion decision tree has many advantages (Zwitter et. al. 83). The diagnostic rules (decision trees) generated from examples, perform on new cases typically in the reliability range of human specialists. They can be used without the computer, simply printed on the paper. ASSISTANT can be used for automatic synthesis of the knowledge bases, which is the bottleneck in the development of expert systems (Bratko et. al. 85).

1. UVOD

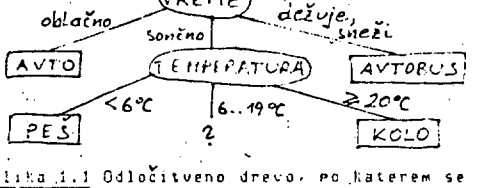
Kot alternativa standardnim statističnim metodam za razpoznavanje in grupiranje vzorcev se je pojavilo strukturno avtomatsko učenje, ki temelji na metodah umetne inteligence (glej npr. Nilsson 82). Bistvena prednost strukturnega (lahko bi rekli tudi induktivnega, simboličnega ali logičnega) avtomatskega učenja je v razumljivosti naučenih pravil, ki so simbolični opisi pojavov, teorij, objektov ali konceptov. Za osnovne principe strukturnega avtomatskega učenja glej (Kononenko 85). V tem prispevku nas bo zanimalo avtomatsko učenje odločitvenih pravil na osnovi primerov.

Problem induktivnega učenja odločitvenih pravil 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.

Sistem ASISTENT smo razvili iz Quinlanovega sistema ID3 (Quinlan 79,79a,82). Osnovna ideja sistema je gradnja odločitvenega drevesa. 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 1.1. Odločitveno drevo pravi : če je vreme oblačno, potem se odpeljem v službo z avtom, če je vreme deževno ali pa sneži, potem se odpeljem v službo z avtobusom, in če je vreme sončno, potem, če je temperatura zunaj pod +6 st C, grem v službo peš, če je temperatura zunaj nad + 19 st C, grem v službo s kolesom, drugače pa ne znam izbrati prevoznega sredstva.



Slika 1.1 Odločitveno drevo, po katerem se odločamo o izbiri prevoznega sredstva za v službo.

Osnovni algoritem gradnje odločitvenega drevesa je v grobem sledeč:

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 vrednostih atributa v disjunktne podmnožice
3. za vsako podmnožico ponovi celoten algoritem

Algoritem rekurzivno gradi drevo. Bistvo algoritma je izbira najbolj informativnega atributa, ki je podrobneje opisana v 2. poglavju. 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. Osnovni algoritem smo izpopolnili na več načinov. 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.

Eksperimenti v medicinskih domenah so pokazali na pravilnost izpopolnitev. ASISTENT je v vseh medicinskih domenah dosegel diagnostično natančnost zdravnikov specialistov. Zgrajena odločitvena drevesa se dajo direktno interpretirati v naravnem jeziku in so zlahka dojemljiva. Lahko se uporabljajo brez računalnika (izpisana na papir), npr. kot priročnik za diagnosticiranje. Za uporabnika so zanimiva, ker iz njih lahko razbere določene relacije in zakonitosti iz svoje domene. ASISTENT je splošen sistem, le poskuse smo do sedaj delali samo na podatkih iz medicine.

ASISTENT je implementiran v PASCALu (cca 5000 vrstic) in se 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.

V 2. poglavju so podane nekatere formalne izpeljave, ki so osnova izpopolnitvam v sistemu ASISTENT. V 3. poglavju so opisane izpopolnitve sistema ASISTENT. V 4. poglavju so prikazani eksperimenti z ASISTENTom v petih različnih medicinskih domenah. Implementacija sistema je opisana v poglavju 5.

2. TEORETIČNE OSNOVE

V tem poglavju so podane formalne izpeljave, ki so podlaga izpopolnitvam v sistemu ASISTENT. Najprej so podane lastnosti informacijske funkcije, ki jo je Quinlan uporabljal v sistemu ID3. V posl. 2.2 je podana izpeljava ocene verjetnosti klasifikacijske točnosti s sistemom ASISTENT. V posl. 2.3 je izpeljan Bayesov verjetnostni princip, katerega klasifikacijsko natančnost smo primerjali s sistemom ASISTENT (glej posl. 4.4). V posl. 2.4 je podan kriterij ocenjevanja klasifikacijske natančnosti, ki upošteva apriorne verjetnosti posameznih razredov.

2.1 LASTNOSTI INFORMACIJSKE FUNKCIJE

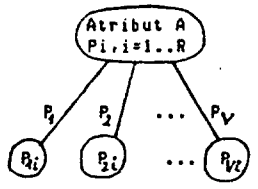
Bistvo algoritma za gradnjo odločitvenega drevesa je izbira najbolj informativnega atributa, ki v ID3 temelji na teoriji informacij (glej npr. Gvozdek 78). Kriterij temelji na naslednji izpeljavi: Potrebna količina informacije za klasifikacijo enega primera je enaka:

$$E = -\sum_i P_i * \log_2 (P_i)$$

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:

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

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. Verjetnosti, ki nastopajo v formulah, so ponazorjene na sliki 2.1.



V = število vrednosti atributa A

i = 1..R, R = število razredov

Slika 2.1 V korenu drevesa so verjetnosti razredov P_i, i=1..R, v v-tem nasledniku korena so verjetnosti razredov P_{vi}, i=1..R, P_v, v=1..V so apriorne verjetnosti posameznih vrednosti atributa A.

Oglejmo si nekaj lastnosti funkcij E in I. V nadaljevanju so podani nekateri izreki, ki so dokazani v (Kononenko 85a). Izreka 1 in 2 sta splošno znana in sta tu dodana zaradi kompletnosti.

2.1.1 EKSTREMNE VREDNOSTI

IZREK 1: Če je v danem univerzumu primerov z R različnimi razredi verjetnost nekega razreda r enaka 1, potem je potrebna količina informacije za klasifikacijo enega primera E enaka 0.

IZREK 2: Če imamo v danem univerzumu R možnih razredov, potem je največja možna potrebna količina informacije E za klasifikacijo enega primera enaka

$$E = \log_2 (1/R) \text{ in takrat velja } P_i = 1/R,\ i = 1..R.$$

2.1.2 INFORMATIVNOST ATRIBUTOV

DEFINICIJA: Informativnost atributa A definiramo kot razliko potrebne količine informacije za klasifikacijo enega primera pred in po uporabi atributa:

$$Inf(A) = E - I(A)$$

IZREK 3: Informativnost atributa je vedno večja ali enaka 0.

IZREK 4: Informativnost atributa je največ E, pri čemer mora biti število vrednosti V atributa A večje ali enako številu razredov R : $V \geq R$.

2.1.3 VEČVREDNOSTNI ATRIBUTI

IZREK 5: Imejmo atribut A z V vrednostmi. Dodajmo atributu A še eno vrednost tako, da prvo vrednost V_1 razbijemo na Z vrednosti V_0' in V_1' in imenujemo tako dobljeni atribut A'. Velja:

$Inf(A') \geq Inf(A)$ in

$$Inf(A') = Inf(A) \iff \forall r \in \{1..R\}: \frac{P_{V_i'r}}{P_{V_i'}} = \frac{P_{V_ir}}{P_{V_i}} = \frac{P_{V_0r}}{P_{V_0}}$$

2.1.4 PRINCIP NAJVEČJE KLASIFIKACIJSKE NATANČNOSTI

Če imamo v vozlu množico primerov, lahko verjetnosti razredov P_i v vozlu aproksimiramo z relativnimi frekvencami primerov. Tako dobljene verjetnosti P_i, i=1..R lahko uporabimo za klasifikacijo novih primerov tako, da vsak primer razvrstimo v razred M z največjo verjetnostjo P_M. Natančnost klasifikacije N_0 bo kar verjetnost pravilne klasifikacije P_M:

$$N_0 = P_M = \max_i P_i$$

Če izberemo za koren poddrevesa danega vozla atribut A z V vrednostmi, nam ta razbije množico primerov na V podmnožic. Ustrezne verjetnosti so narisane na sliki 2.1. Natančnost klasifikacije novih primerov po prej opisanem postopku bo sedaj:

$$N_1 = \sum_V^V P_V \max_i^R P_{Vi}$$

DEFINICIJA: Princip največje klasifikacijske natančnosti izbere za koren drevesa atribut, ki maksimizira N_1.

IZREK 6: Imejmo dva univerzuma U1 in U2, v obeh so objekti razdeljeni v R razredov. Trditvi

$$E(U1) > E(U2) \iff N_0(U1) < N_0(U2) \text{ in}$$

$$E(U1) = E(U2) \iff N_0(U1) = N_0(U2)$$

veljata vedno natanko takrat, ko je R=2.

2.2 OCENA KLASIFIKACIJSKE NATANČNOSTI

2.2.1 POLNA MNOŽICA ATRIBUTOV

Quinlan (83) je pokazal naslednje:

Vzemimo, da za dan univerzum, katerega objekti so opisani s končno množico atributov in pripadajo končno mnogo različnim razredom, eksistira eksaktno odločitveno drevo (drevo, ki pravilno klasificira vse objekte iz univerzuma). To pomeni, da je množica atributov, s katerimi so objekti opisani, polna. Recimo, da ima najmanjše eksaktno drevo L listov, in v vsakem listu Li je množica primerov Bi, i=1..L.

DEFINICIJA: Popoln sistem za učenje odločitvenih pravil za dani univerzum je sistem, ki na osnovi učnih primerov zgenerira odločitveno pravilo, ki pravilno klasificira vse primere iz množice Bi, i=1..L, če je bil med učnimi primeri vsaj en primer iz množice Bi.

Quinlan je pokazal, da če bi imeli tak popoln sistem, bi na osnovi učnih primerov dobili odločitveno pravilo, ki bi poljuben primer klasificiralo pravilno z verjetnostjo:

$$P = 1 - (L/(2.72\,N)) * (1 - \sum_r^R P_r^2) + d, \quad d > 0,$$

kjer je L število vozlov v najmanjšem eksaktnem drevesu, N je število učnih primerov, P_r je apriorna verjetnost, da naključno izbran primer pripada r-temu razredu in d je pozitivna konstanta napake. Če opustimo d, imamo spodnjo mejo verjetnosti pravilne klasifikacije poljubnega primera. V poskusih v šahovskih končnicah s polnimi množicami atributov je ID3 vedno presegel ocenjeno spodnjo mejo natančnosti klasificiranja novih primerov.

2.2.2 NEPOLNA MNOŽICA ATRIBUTOV

Imejmo sedaj univerzum z nepolno množico atributov, torej atributi ne zadostujejo za pravilno klasifikacijo vseh objektov. Denimo, da atributi zadostujejo za klasifikacijo MZ vseh primerov v univerzumu. Vzemimo, da imamo minimalno odločitveno drevo, ki zadošča za MZ klasifikacijsko natančnost in ima L listov. V vsakem listu je množica primerov Bi, i=1..L. Množica Bi je v splošnem sestavljena iz množic B_{ri}, r=1..R, pri čemer je B_{ri} množica primerov z r-tim razredom iz i-tega lista. Listu pripišemo razred, ki maksimizira moč množice B_{ri}. Če je več razredov z največjo močjo v listu, sistem naključno izbere med njimi en razred.

DEFINICIJA: Popoln sistem za učenje odločitvenih pravil v univerzumu z nepolno množico atributov je sistem, ki na osnovi učnih primerov zgenerira odločitveno pravilo, ki z MZ natančnostjo klasificira vse primere iz množice Bi, če je bil v učni množici vsaj en primer iz množice B_i.

Izpeljava verjetnosti pravilne klasifikacije poljubnega objekta je analogna izpeljavi v (Quinlan 83). Verjetnost, da med učnimi primeri ni objekta iz B_i, kateremu pripada objekt, ki ga želimo klasificirati, je:

$$P_n = \sum_1^L P(Bi)\cdot(1 - P(Bi))^N, \qquad N = \text{število učnih primerov}$$

Verjetnost, da med učnimi primeri je tak objekt, je enaka $1-P_n$. Verjetnost, da bo odločitveno pravilo naključno uganilo pravi razred objekta, je enaka:

$$P_{nak} = \sum_r^R P_r^2$$

Sedaj lahko izrazimo verjetnost, da bo poljuben primer pravilno klasificiran:

$$P = (1-P_n)\cdot M/100 + P_n \cdot P_{nak}$$

P_n doseže maksimum pri

$$\frac{dP_n}{dP(B_i)} = 0$$

to je pri P(Bi) = 1/(N+1). Tako je zgornja meja P_n:

$$P_n \leq \sum_i^L \frac{1}{N+1}\cdot\left(1-\frac{1}{N+1}\right)^N = \frac{L}{N}\left(\frac{N}{N+1}\right)^{N+1}$$

Ker velja:

$$\lim_{n\to\infty}\left(\frac{N}{N+1}\right)^N = \frac{1}{e},$$

je

$$P_n = \frac{L}{2.72\cdot N} - d, \quad d>0.$$

Končna ocena spodnje meje verjetnosti pravilne klasifikacije primera z odločitvenim pravilom je enaka:

$$P = \frac{M}{100} - \frac{L}{2.72\cdot N}\cdot\left(\frac{M}{100} - P_{nak}\right) + d', \quad d'>0.$$

Pri tem je N množica učnih primerov in d' je pozitivna klasifikacijska napaka. Za konkreten problem lahko L ocenimo s številom listov v dobljenem drevesu. Problematična je ocena parametra M.

2.3 BAYESOV VERJETNOSTNI PRINCIP

Problem klasifikacije novih primerov pri danih učnih primerih, ki je podan v posl. 1.4, lahko rešimo z Bayesovim verjetnostnim principom. Verjetnost razreda R pri danih vrednostih n atributov A* je enaka:

(če je P(R) apriorna verjetnost razreda R in P(A*) apriorna verjetnost danih vrednosti atributov in če označimo posamezne vrednosti atributov z Ai):

$$P(R|A*) = P(R) \cdot \frac{P(\bigwedge_i A_i \mid R)}{P(\bigwedge_i A_i)}$$

če predpostavimo medsebojno neodvisnost atributov, dobimo:

$$P(R|A*) = P(R) \cdot \frac{\prod_{i=1}^{n} P(A_i|R)}{\prod_{i=1}^{n} P(A_i)} = \frac{\prod_{i=1}^{n} P(A_i \& R)}{P(R)^{n-1} \cdot \prod_{i=1}^{n} P(A_i)}$$

Verjetnosti na desni strani enačbe lahko aproksimiramo z relativnimi frekvencami iz dane množice učnih primerov. Na ta način lahko klasificiramo poljuben nov primer. Primer klasificiramo v razred, ki maksimizira verjetnost P(R|A*), pri čemer so A* vrednosti posameznih atributov pri danem primeru.

2.4 KRITERIJI OCENJEVANJA KLASIFIKACIJSKE NATANČNOSTI

če je v dani množici primerov en razred zelo verjeten, potem lahko vedno dosežemo dobro natančnost klasificiranja. Preprosto klasificiramo vsak primer v razred, ki je najbolj verjeten. Tako klasificiranje nima pravega pomena. Zato želimo namesto preprostega seštevanja pravilnih odgovorov dobiti kriterij za ocenjevanje natančnosti klasificiranja, ki bo izpolnjeval naslednje pogoje:

a) Pravilno klasificiranje primera iz razreda, ki je manj verjeten, mora biti več vredno kot pravilno klasificiranje primera iz razreda, ki je bolj verjeten.

b) če so vsi razredi enako verjetni, mora biti kriterij enak kot kriterij seštevanja pravilnih odgovorov.

c) če je klasifikacija popolnoma naključna, tako da je verjetnost, da razvrstimo primer v razred r enaka 1/R, potem mora biti natančnost klasificiranja enaka 1/R (R je število razredov).

d) če pravilno klasificiramo vse primere, mora biti rezultat 100%.

e) če klasificiramo vse primere v en razred, mora biti natančnost klasifikacije enaka 1/R.

IZREK 7: če pri štetju pravilnih odgovorov namesto enk seštevamo izraze:

$\frac{1}{R \cdot Pr}$, R je število vseh razredov, Pr je apriorna verjetnost razreda r primera, ki smo ga pravilno klasificirali,

in je dobljeni rezultat klasificiranja enak :

$$Rez = \frac{1}{N} \sum_{r}^{R} \frac{Kr}{R \cdot Pr}$$, Kr je število pravilnih odgovorov iz r-tega razreda in N je št. razvrščenih primerov

potem dobljeni kriterij zadovoljuje pogoje a), b) in c). Pogoja d) in e) sta izpolnjena, če v množici testnih primerov veljajo apriorne verjetnosti razredov Pr, r=1..R.

3. ASISTENT

Sistem, ki smo ga razvili iz Quinlanovega ID3 (Quinlan 79), smo poimenovali ASISTENT. Osnovni algoritem sistema ID3 (glej pogl. 1) smo razširili in izpopolnili v naslednjih smereh:

1) V ASISTENTu je implementirana gradnja odločitvenih dreves brez iteracij.

2) ASISTENT lahko obravnava nepopolne podatke.

3) ASISTENT omogoča klasifikacijo dvomljivih primerov v kombinaciji s Bayesovim verjetnostnim principom.

4) ASISTENT omogoča avtomatsko izbiro dobrih učnih primerov.

5) ASISTENT sam avtomatično določa meje intervalov vrednosti zveznih atributov.

6) V ASISTENTu je rešena pomankljivost informacijske funkcije pri ocenjevanju večvrednostnih atributov z binarno gradnjo dreves.

7) ASISTENT omogoča avtomatsko rezanje nezanesljivih delov drevesa.

V eksperimentih v 5 različnih medicinskih domenah so razširitve prispevale k boljši diagnostični natančnosti in k večji razumljivosti generiranih diagnostičnih pravil v obliki odločitvenih dreves. V tem poglavju so opisane posamezne izboljšave. V 4. poglavju so opisani rezultati eksperimentov z ASISTENTom v primerjavi z osnovnim algoritmom in s statistično metodo, ki temelji na Bayesovem principu verjetnosti. Podana je tudi primerjava diagnostične natančnosti odločitvenih dreves in zdravnikov specialistov. V 5. poglavju je na kratko opisana implementacija sistema ASISTENT.

3.1 GRADNJA DREVESA BREZ ITERACIJ

Quinlan (82) je v svojih poskusih uporabljal naslednji iterativni algoritem za gradnjo dreves:

1. Iz množice učnih primerov naključno izberi *nekaj* primerov.
2. Ponavljaj
   2.1 Nad izbrano množico primerov zgradi drevo.
   2.2 Testiraj drevo s preostalimi primeri.
   2.3 K izbrani množici primerov dodaj *nekaj primerov*, ki jih dobljeno drevo ne klasificira pravilno,

   dokler drevo ne klasificira pravilno vseh *primerov*.

Ker se verjetnosti v informacijski funkciji (glej pogl. 2) aproksimirajo z relativnimi frekvencami iz množice učnih primerov, bo aproksimacija tem boljša, čim večja bo množica učnih primerov za gradnjo drevesa. če postavimo za učno množico celotno množico razpoložljivih primerov, bo aproksimacija najboljša. S tem se tudi izognemo iteracijam, saj se drevo zgradi samo enkrat.

Problem nastopi, če je hitri pomnilnik premajhen za celotno množico učnih primerov. V ASISTENTu je problem rešen tako, da se relativne frekvence primerov računajo ob branju datoteke s primeri toliko časa, dokler množica primerov v vozlu ni zadosti majhna, da jo lahko spravimo v hitri pomnilnik. Algoritem je zaradi tega za velike množice primerov (npr. nekaj sto tisoč primerov) počasnejši, vendar so dobljena drevesa manjša. Za manjše množice primerov (npr. nekaj tisoč primerov) pa je algoritem seveda hitrejši. Do istih zaključkov je prišel tudi O'Keefe (83).

3.2 OBRAVNAVANJE NEPOPOLNIH PODATKOV

V realnih problemih pogosto naletimo na nepopolne podatke, ko nekaterim primerom manjkajo vrednosti za določene atribute. Take primere je ID3 lahko upošteval samo, če je bila k vsakemu atributu dodana posebna vrednost, označena za neznano. Vendar ta rešitev ni zadovoljiva. Po nepotrebnem dobimo bolj razvejano in s tem večje drevo, pa tudi učna množica mora biti večja, če želimo imeti vse veje v drevesu izpolnjene (sicer dobimo kopico praznih listov, Quinlan jih imenuje listi NULL). če nov primer med klasifikacijo pade v prazen list, ga drevo ne zna klasificirati.

Druga rešitev, ki je implementirana v ASISTENTu, je ta, da se vsakemu primeru, ki mu manjka vrednost za določen atribut, pripišejo vse možne vrednosti za dani atribut, vsaka z določeno verjetnostjo. če dani primer spada v razred R, potem je verjetnost, da ima vrednost V atributa, za katerega mu manjka vrednost, enaka:

$$P(V|R) = \frac{P(V \& R)}{P(R)}$$

Verjetnosti na desni strani enačbe se lahko med gradnjo drevesa aproksimirajo z relativnimi frekvencami iz množice primerov v trenutno sledanem vozlu drevesa. Tako bo primer, ki nima podane vrednosti za izbrani atribut v vozlu, ustrezal vsem vejam iz vozla, vsaki z določeno verjetnostjo. Pri klasifikaciji novega primera, ki nima podane vrednosti atributa v vozlu, se upoštevajo samo apriorne verjetnosti vrednosti v vozlu (P(V)). V posl. 4.3.1 je podana primerjava eksperimentalnih rezultatov dobljenih z dvema opisanima načinoma obravnavanja nepopolnih podatkov.

3.3 KLASIFICIRANJE S POMOČJO BAYESOVEGA PRINCIPA

Odločitveno drevo je nezanesljivo, če nimamo polne množice atributov (množica atributov ne zadostuje za eksaktno klasifikacijo vseh primerov), če imamo premalo učnih primerov, je drevo še toliko slabše. Zanesljivost drevesa se izraža v številu primerov v listih drevesa. Če je v listu samo en učni primer, potem je Klasifikacija s tem listom nezanesljiva. Ekstremni primer je prazen list (NULL). Klasifikacija primerov s praznimi listi je nemogoča, z listi z malo primeri pa nezanesljiva.

V ASISTENTu je rešen problem praznih listov z Bayesovim verjetnostnim principom, ki je izpeljan v posl. 2.3. Med samo gradnjo se vsakemu listu pripišejo verjetnosti razredov, dobljene po Bayesovem verjetnostnem principu z upoštevanjem atributov, ki nastopajo od korena drevesa do danega lista. Če je list prazen, potem se mu pripiše razred z največjo izračunano verjetnostjo. Če se v neprazenm listu izračunane verjetnosti ujemajo s primeri v listu, je list zanesljivejši za klasifikacijo novih primerov.

3.4 IZBIRA DOBRIH UČNIH PRIMEROV

Če množica atributov ni polna (ne zadostuje za eksaktno klasifikacijo v dani problemski domeni), se v listih drevesa pojavlja več različnih razredov. Drugi vzrok za isto situacijo so lahko napake v učnih primerih (šum). Če pa je množica učnih primerov premajhna, bomo sicer v listih dobili samo po en razred, vendar je vzrok razločevanja vseh razredov med seboj z določenimi vrednostmi atributov zgolj naključje. Razmišljanje iz poglavja 2.2.2 pokaže, da je pri nepolnih množicah atributov dobro upoštevati samo dobre učne primere, ki bodo vodili k večji klasifikacijski natančnosti zgrajenega drevesa. Dobri učni primeri povzročijo tudi to, da se drevo ne razraste po nepotrebnem.

Princip za izbiro dobrih učnih primerov, ki smo ga uporabljali v poskusih z ASISTENTom v medicinskih domenah, je sledeč:

1. Nad dano množico primerov sestavi statistiko.
2. Klasificiraj vse primere z Bayesovim principom.
3. Pravilno klasificirani primeri so dobriza učenje

Drevesa, zgrajena samo nad dobrimi učnimi primeri, so bila precej manjša (zaradi tega razumljivejša) in tudi natančnejša pri klasifikaciji novih primerov (glej posl. 4.3.4).

3.5 OBRAVNAVANJE ZVEZNIH ATRIBUTOV

Pri osnovnem algoritmu gradnje dreves nastopi problem pri zveznih atributih. Ti atributi imajo lahko poljubne realne ali celoštevilčne vrednosti na določenem intervalu (npr. temperatura, pritisk, število bakterij na kubični cm, itd.). Če želimo take atribute uporabiti pri gradnji drevesa, je treba vrednosti grupirati v intervale tako, da ne bodo preozki (dobimo preveč možnih vrednosti, ki se ne razlikujejo mnogo med seboj) niti preširoki (izgubimo informacijsko vsebino atributa). Rešitev, da intervale določi človek strokovnjak na danem področju, ima dve slabosti:

1) Zahteva delo človeka strokovnjaka, ki ni vedno dostopen.

2) Dobljeni intervali so lahko pristranski, neobjektivni, površni ali neuporabni za gradnjo odločitvenega drevesa (npr. strokovnjak bo vrednosti atributa starost razbil na intervale po 10 let, čeprav pri svojem delu ne dela razlike med vrednostmi od 60 do 70 let in od 70 do 80 let).

Rešitev, ki je implementirana in preizkušena v ASISTENTu je ta, da vsak zvezni atribut postane binaren. Meja, ki razdeli interval možnih vrednosti na dva podintervala, se določi avtomatsko: to je meja, ki maksimizira atributovo informativnost. Atribut se lahko ponovno pojavi v poddrevesu, katerega prednik je sam, vendar z manjšim intervalom možnih vrednosti. Ista rešitev je privzeta v sistemu ACLS (Paterson in Niblett 82).

3.6 BINARNA GRADNJA

Ocenitvena funkcija daje prednosti atributom z več možnimi vrednostmi, kar neposredno sledi iz Izreka 5 (posl.2.1.3). Tako ima atribut A', dobljen iz nekega atributa A z naključnim razbijanjem ene vrednosti atributa A na več vrednosti, večjo informativnost kot originalni atribut A. Ta problem smo skušali rešiti z normiranjem informativnosti atributa na število potrebnih testiranj za pridobitev vrednosti atributa. Tako je normirana informativnost NI atributa A z V možnimi vrednostmi enaka:

$$NI(A) = \frac{Inf(A)}{\log_2 V}$$

Taka rešitev ni zadovoljiva, ker posamezne vrednosti atributa niso informacijsko enakovredne. Če npr. nekemu atributu z 2 možnima vrednostima dodamo še dve možni vrednosti, tako da razpolovimo vsako od prvotnih vrednosti, bomo dobili atribut s štirimi možnimi vrednostmi. Informativnost novega atributa bo ostala nespremenjena, normirana informativnost pa bo za polovico manjša od normirane informativnosti originalnega atributa. Vseeno pa je normirana ocenitvena funkcija dajala bolj natančna odločitvena drevesa (glej 4.3.2).

Problemu smo se izognili z uvedbo binarne gradnje. Vsak atribut postane binaren. Za zvezne atribute je postopek opisan v posl. 3.5. Pri diskretnih atributih se grupirajo vrednosti v dve disjunktni podmnožici tako, da maksimizirata atributovo informativnost. Šele ko so vsi atributi binarni, jih primerjamo med seboj po informativnosti. Tako zvezni kot diskretni atribut se lahko pojavi večkrat na isti poti od korena do lista drevesa. Na nižjih nivojih ima atribut manj možnih vrednosti kot na višjih nivojih. Pri tem je pomembna uvedba praznih listov (NULL). Tako se včasih iz vozla pojavi še tretja veja s praznim listom, ki ustreza vrednostim atributa v vozlu, katere nima noben primer v vozlu. Listi NULL nas opozorijo na pomankljivost učne množice ali pa na fiziološko ali logično nemogoče vrednosti.

Poleg tega, da se izognemo slabosti ocenitvene funkcije, so dobljena drevesa manjša, večji je efekt generalizacije nad učnimi primeri (glej posl. 4.3.5). Dobljena drevesa so tudi natančnejša pri klasifikaciji novih primerov. To je posledica slabosti informacijske funkcije, ker je nezanesljiva nad majhnimi množicami primerov (aproksimacija verjetnosti v formuli z relativnimi frekvencami je nezanesljiva). Nebinarna gradnja povzroči hitro razbitje učne množice, preden je izbran kak pomemben atribut. V vozlu ostane premalo primerov za zanesljivo oceno atributov. Binarna gradnja zadržuje razdelitev učne množice do nižjih nivojev v drevesu in tako omogoči tudi ostalim atributom, da so lahko izbrani za koren kakega poddrevesa. Več o nezanesljivosti informacijske funkcije nad majhnimi množicami primerov je opisano v posl. 3.7.

Binarna gradnja ima dve slabosti. Prva je neučinkovitost. Če želimo poiskati za vsak atribut najboljšo grupacijo možnih vrednosti, nam čas iskanja kombinatorično narašča s številom možnih vrednosti

atributa. Temu smo se v ASISTENTu delno izognili tako, da diskretne atribute ocenjujemo najprej s prej opisano normirano ocenitveno funkcijo, zatem pa nekaj najboljših skupaj z vsemi zveznimi atributi ocenjujemo z izčrpnim grupiranjem vrednosti. Iskanje učinkovitih hevrističnih algoritmov za grupiranje vrednosti atributov je predmet nadaljnjega dela.

Druga slabost binarne gradnje je nestrukturiranost dobljenih dreves. Tudi Shepherd (83) je poudaril nestrukturiranost binarnih dreves. Iz tega sledi navidezna slabša razumljivost dobljenih binarnih odločitvenih dreves. Pokazalo pa se je, da binarna drevesa oponašajo človekov način razmišljanja. Zdravniki so nam povedali, da sami ne morejo naenkrat držati v mislih vseh možnih vrednosti za atribute z več različnimi vrednostmi. Zato v mislih grupirajo podobne vrednosti, pozneje pa, ko želijo detajlizirati svoje razmišljanje, razbijejo grupe vrednosti na posamezne vrednosti atributa. Torej je binarno drevo razumljivo kljub navidezni slabi strukturiranosti. Poleg tega so dobljena drevesa nekajkrat manjša (glej posl. 4.3.5) in so že zaradi tega precej lažje razumljiva.

Quinlan (85) je predlagal drugačno normiranje ocenitvene funkcije: normiranje na potrebno količino informacije za pridobitev vrednosti danega atributa. Tako je normirana informativnost GNI atributa A, ki ima V možnih vrednosti enaka:

$$GNI(A) = \frac{Inf(A)}{-\sum_{v}^{V} Pv \log_2 Pv}, \qquad Pv \text{ je verjetnost } v\text{-te vrednosti atributa A.}$$

To normiranje je posplošitev normirane funkcije NI. Funkciji sta enaki, ko velja Pv = 1/V, v=1..V:

$$\sum_{v}^{V} (1/V) \log_2 (1/V) = -\log_2 (1/V) = \log_2 V$$

Funkcija GNI se je izkazala za boljšo pri atributih z majhnim številom vrednosti. Pri atributih z mnogo vrednostmi pa je bila binarna gradnja boljša. Slabost funkcije GNI je analogna slabosti funkcije NI. Če namreč ponovimo razmišljanje o atributu A z dvema možnima vrednostima, ki sa spremenimo v atribut A' s štirimi možnimi vrednostmi tako, da obe vrednosti razpolovimo, velja:

$$Inf(A) = Inf(A') \quad \text{in} \quad GNI(A) > GNI(A').$$

Druga trditev sledi iz prve:

$$GNI(A) = \frac{Inf(A)}{-p_1 \log_2 p_1 - p_2 \log_2 p_2} > GNI(A') = \frac{Inf(A')}{-p_1 \log_2 \frac{p_1}{2} - p_2 \log_2 \frac{p_2}{2}}$$

Torej, čeprav imata atributa enako informacijsko vsebino, je normirana ocena atributa A boljša od ocene atributa A', kar pomeni, da funkcija GNI daje prednost atributom z mnogo vrednostmi.

3.7 REZANJE NEZANESLJIVIH DELOV DREVESA

Verjetnosti v informacijski ocenitveni funkciji se aproksimirajo z relativnimi frekvencami primerov v trenutno sledanem vozlu med gradnjo drevesa. Zato je ocenitvena funkcija zanesljiva samo nad velikimi množicami učnih primerov v vozlu. Zaradi tega je gradnja drevesa na nižjih nivojih, kjer je malo primerov v vozlih, nezanesljiva.

Temu se izognemo z rezanjem nezanesljivih delov drevesa. V listih ostaja več razredov, kar pa je za nepolne množice atributov nujno (glej posl. 2.2.2). Porezano drevo je boljše od drevesa, ki je zgrajeno nad samo dobrimi učnimi primeri (glej posl. 3.4), ker opozarja uporabnika ne samo na najbolj verjetne razrede, ampak tudi na možne izjeme (npr. zdravnika opozori na vse možne diagnoze).

V ASISTENTu uporabljamo za rezanje princip maksimalne klasifikacijske pravilnosti. Izrek 6 (posl. 2.1.4) pravi, da je kriterij maksimalne klasifikacijske natančnosti zelo podoben informacijski funkciji. Vendar za uporabo pri rezanju potrebujemo nove primere za testiranje klasifikacijske natančnosti. Ker jih nimamo na razpolago, moramo eksperimentirati z učnimi primeri v vozlu. Princip rezanja je definiran takole:

1. Za vsak primer v vozlu drevesa ponovi:
   1.1 Izloči ga iz množice primerov v vozlu.
   1.2 Izberi najboljši atribut glede na preostalo množico.
   1.3 Za dani primer izračunaj klasifikacijske napake z in brez upoštevanja izbranega atributa.
   1.4 Prištej napake k vsotama napak z in brez upoštevanja izbranih atributov.
2. če je vsota napak brez upoštevanja atributov manjša ali enaka vsoti napak z upoštevanjem izbranih atributov, potem prenehaj gradnjo na tem mestu, sicer nadaljuj gradnjo poddrevesa.

Zaradi večje učinkovitosti je algoritem implementiran tako, da ponovitve pod 1. točko opravlja za določeno število naključno izbranih majhnih podmnožic primerov. Porezana drevesa so manjša, zaradi tega bolj razumljiva in tudi bolj natančna pri klasifikaciji novih primerov (glej posl. 4.3.6).

4. POSKUSI V MEDICINI

Sistem ASISTENT smo preizkusili v 5 različnih medicinskih domenah. Eksperimenti so potrdili pravilnost izpopolnitev, opisanih v 3 poglavju. Tu so opisana medicinska področja in podatki, ki smo jih uporabili v eksperimentih. Opisan je način eksperimentiranja in primerjava diagnostične natančnosti ASISTENTa z natančnostjo osnovnega algoritma ID3, z natančnostjo nekaterih statističnih metod in z natančnostjo zdravnikov specialistov.

4.1 OPIS MEDICINSKIH PODROČIJ IN PODATKOV

4.1.1 LOKALIZACIJA PRIMARNEGA TUMORJA

Pri pacientih z odkritimi metastazami je zdravljenje učinkovitejše, če je znana lokacija primarnega tumorja v telesu. Zdravniki ločijo med 22 različnimi lokacijami. Podatki o pacientu, na osnovi katerih zdravnik sklepa na lokacijo primarnega tumorja, so starost pacienta, spol, histološki tip karcinoma, stopnja diferenciacije in 14 možnih lokacij odkritih metastaz. Diagnostični problem je torej podan z 18 atributi in 22 možnimi razredi (diagnozami).

Iz Onkološkega inštituta v Ljubljani smo dobili podatke o 339 pacientih z znanimi lokacijami primarnih tumorjev. Podatki so bili nepopolni v smislu, da je za nekatere paciente manjkal podatek o histološkem tipu karcinoma in o stopnji diferenciacije. Na Onkološkem inštitutu so testirali 4 zdravnike specialiste onkologe in 4 interniste s tem diagnostičnim problemom (Zwitter 80). Internisti so v povprečju pravilno odgovorili v 32% primerov in onkologi v 42% primerov.

4.1.2 PROGNOSTIKA PONOVITVE RAKA NA DOJKI

Pri približno 30% pacientkah, ki jim je bil z operativnim posegom odstranjen rak na dojki, se bolezen ponovi v roku petih let. Za boljše zdravljenje po operaciji (obsevanje, zdravila) je potrebno za vsako bolnico napovedati verjetnost ponovitve bolezni. Na verjetnost ponovitve sklepajo zdravniki na osnovi podatkov o starosti bolnice, velikosti in lokacije tumorja, statusu menstruacijskega ciklusa in podatkih o povečanju bezgavk. Problem je podan z 11 atributi in dvema možnima razredoma.

Iz Onkološkega inštituta v Ljubljani smo dobili podatke za 286 operativnih bolnic, za katere je bil znan status bolezni pet let po operaciji. Podatki so bili precej pomanjkljivi. Na Onkološkem inštitutu v Ljubljani je bilo testiranih 5 zdravnikov specialistov s tem prognostičnim problemom. V povprečju so bili zdravniki točni v 64% primerov.

4.1.3 LIMFOGRAFIJA

Podatke o 150 pacientih Onkološkega inštituta v Ljubljani, ki so jih v poskusih uporabili Bratko in Mulec (80, glej tudi Mulec 80) ter Soklič (80) smo uporabili v naših eksperimentih. Za vsakega pacienta so znani podatki o starosti, spolu, nekateri podatki o bezgavkah in rezultati nekaterih laboratorijskih testov (skupaj 18 atributov). Zdravniki ločijo med 9 različnimi diagnozami.

V naših podatkih za eno diagnozo ni bilo nobenega primera, za dve diagnozi pa je bil samo po en primer. Tako je bil naš diagnostični problem definiran z 18 atributi in 6 možnimi diagnozami. Pozneje so nam zdravniki povedali, da sta si diagnozi "maligni limfom" in "sum na maligni limfom" tako podobni, da praktično ne delajo razlik med njima. Enako velja za diagnozi "metastaze" in "sum na metastaze". S tem se je število možnih razredov zreduciralo na 4. Pri tem diagnostičnem problemu ni bilo narejeno testiranje zdravnikov. Zdravnik specialist je ocenil, da so zdravniki začetniki pri tem diagnostičnem problemu približno 60% natančni in izkušeni zdravniki specialisti približno 85% natančni.

4.1.4 PROGNOSTIKA PREŽIVETJA PRI HEPATITISU

Dr. Gail Gons iz Carnegie-Mellon University (ZDA) je eksperimentirala s standardnimi statističnimi metodami s podatki o 155 pacientih, obolelih za kronični ali akutni hepatitis in z znanim preživetjem. Prognostični problem je podan z 19 atributi in dvema možnima razredoma (79% primerov pripada enemu razredu in 21% drugemu). Dr. Gonsova je s statističnimi metodami dosegla 80% ocenjeno prognostično natančnost (Diaconis & Efron 83). O natančnosti zdravnikov pri tem prognostičnem problemu nimamo podatkov.

4.1.5 DIAGNOSTIKA OKVAR SPODNJEGA URINALNEGA TRAKTA

Zdravniki ločijo med 8 različnimi okvarami spodnjega urinalnega trakta, torej je skupaj z "normalno" diagnozo možnih 9 različnih diagnoz. Diagnostični problem se razlikuje pri obeh spolih. Na diagnozo sklepajo na osnovi 19 simptomov in 25 znakov in laboratorijskih testov pri moških in 26 pri ženskah. Diagnostični problem je torej podan s 44 atributi pri moških in 45 atributi pri ženskah in z 9 možnimi diagnozami.

Iz Clinical Investigation Unit iz Ham Green Hospital v Bristolu (Velika Britanija) smo dobili podatke za 1843 moških in 3580 žensk, ki so se zdravili v tej bolnišnici, ki slovi po vodilih strokovnjakih na tem področju v svetu. Zdravniki sami ne vedo, koliko so natančni pri diagnosticiranju, ker končne diagnoze ni možno preveriti.

4.2 NAČINI EKSPERIMENTIRNJA IN OCENJEVANJA REZULTATOV

V vseh medicinskih domenah smo delali poskuse tako, da smo naključno izbrali 70% primerov za učenje in 30% preostalih primerov za testiranje zgrajenega drevesa. Vsak poskus smo ponovili štirikrat z naključno izbranimi učnimi in testnimi primeri, razen pri diagnostiki okvar spodnjega urinalnega trakta, ker je bila gradnja dreves predrasa glede na razpoložljiv računalniški čas. Natančnosti zgrajenih pravil so ocenjene s povprečji relativnih števil pravilnih klasificiranj testnih primerov od štirih poskusov.

Pri domenah, kjer se apriorne verjetnosti razredov močno razlikujejo (prognostika raka na dojki in prognostika preživetja pri hepatitisu), smo v rezultatih dodane ocene natančnosti, obtežene z obratnimi apriorhimi verjetnostmi razredov. Ta način ocenjevanja je izpeljan v poglavju 2.4.

Odločitveno drevo, izpisano na papirju, se lahko uporablja brez računalnika. Lahko se direktno interpretira v naravnem jeziku in je zdravniku razumljivo. Zato ima ASISTENT prednost pred statističnimi metodami. K razumljivosti prispeva tudi velikost odločitvenih dreves. Čim manjše je drevo, tem lažje je dojemljivo.

4.3 VPLIV IZBOLJŠAV ASISTENTa NA NATANČNOST DREVES

V poglavju 3 so opisane razlike med sistemoma ASISTENT in ID3. Tu so prikazani eksperimentalni rezultati v 5 različnih medicinskih domenah, ki potrjujejo pravilnost izboljšav v sistemu ASISTENT.

4.3.1 UPOŠTEVANJE NEPOPOLNIH PRIMEROV

V posl. 3.2 je opisan način obravnavanja nepopolnih podatkov v sistemu ID3 in v sistemu ASISTENT. V tabeli 4.1 so podani primerjalni rezultati med dvema različnima načinoma obravnavanja nepopolnih podatkov pri gradnji odločitvenih dreves v domeni "lokalizacija primarnega tumorja" (glej 4.1.1). Drevesa, dobljena z dodajanjem "neznane" vrednosti atributom, so bolj razvejana, večja in s tem tudi slabše razumljiva.

način gradnje	število vozlov	število listov	natančnost
s posebnimi vred.	298	164	37%
z obteževanjem	246	141	37%

Tabela 4.1 Rezultati gradnje drevesa v domeni "primarni tumor" z dvema različnima načinoma obravnavanja nepopolnih podatkov.

4.3.2 NORMIRANA OCENITVENA FUNKCIJA

V poglavju 3.6 so opisane slabosti informacijske ocenitvene funkcije in definirana je normirana ocenitvena funkcija. V tabeli 4.2 so podani rezultati gradnje dreves z navadno in z normirano ocenitveno funkcijo. Ker normirana ocenitvena funkcija bolje ocenjuje atribute, so dobljena drevesa manjša, zaradi tega razumljivejša in natančnejša pri klasifikaciji novih primerov.

ocenitvena fun.	število vozlov	število listov	natančnost
navadna	246	141	37%
normirana	238	135	41%

Tabela 4.2 Rezultati gradnje drevesa v domeni "primarni tumor" z normirano in nenormirano ocenitveno funkcijo.

4.3.3 OBRAVNAVANJE ZVEZNIH ATRIBUTOV

V posl. 3.5 je opisan način obravnavanja zveznih atributov v ASISTENTu. V tabeli 4.3 so podani rezultati eksperimentov v domeni "primarni tumor" z avtomatskim določevanjem mej zveznim atributom in z določevanjem mej zdravnikov specialistov. Ker so zdravniki možne vrednosti atributa starost razbili na 9 intervalov, se je atribut pojavil na vrhu drevesa, čeprav se je pozneje izkazalo, da je le malo pomemben. Avtomatska izbira mej za vrednosti zveznih atributov je izboljšala gradnjo. Dobljena drevesa so manjša in natančnejša.

izbira mej za zvezne atribute	število vozlov	število listov	natančnost
zdravniki	246	141	37%
avtomatsko	242	140	41%

Tabela 4.3 Rezultati gradnje drevesa v domeni "primarni tumor" z različnimi načini izbire mej za vrednosti zveznih atributov.

4.3.4 IZBIRA DOBRIH UČNIH PRIMEROV

V posl. 3.4 so definirani dobri učni primeri in opisan je algoritem za izbiro dobrih učnih primerov. V tabeli 4.4 so podani rezultati gradnje dreves nad celimi učnimi množicami in samo nad dobrimi učnimi primeri. Drevesa, dobljena samo iz dobrih učnih primerov, so precej manjša in natančnejša pri klasifikaciji novih primerov.

domena	učni primeri	št. vozlov	št. listov	natančnost
primarni tumor	vsi	242	140	41%
	samo dobri	100	50	44%
rak na dojki	vsi	190	133	67%
	samo dobri	34	22	72%
limfosrafija	vsi	90	40	75%
	samo dobri	37	22	79%

Tabela 4.4 Rezultati gradnje drevesa v 3 medicinskih domenah nad vsemi učnimi primeri in samo nad dobrimi učnimi primeri.

4.3.5 BINARNA GRADNJA

V poglavju 3.6 so opisani razlogi za binarno gradnjo dreves. Binarna gradnja je precej zmanjšala dobljena odločitvena drevesa, ki so zaradi tega lažje razumljiva. V tabeli 4.5 so podani rezultati binarne gradnje v primerjavi z nebinarno gradnjo.

domena	način gradnje	št.vozlov	št.listov	natančnost
primarni tumor	nebinarna	242	140	41%
	binarna	188	90	41%
rak na dojki	nebinarna	190	133	67%
	binarna	120	63	67%
limfosra-fija	nebinarna	90	40	75%
	binarna	38	22	76%
okvare sp.tr. pri moških	nebinarna	525	336	62%
	binarna	359	199	66%

Tabela 4.5 Rezultati binarne gradnje drevesa v 4 medicinskih domenah v primerjavi z nebinarno gradnjo.

domena	rezanje	št. vozlov	št. listov	natančnost
primarni tumor	da	35	18	46%
	ne	188	90	41%
rak na dojki	da	16	9	72%
	ne	120	63	67%
limfosra-fija	da	25	14	77%
	ne	38	22	76%
hepa-titis	da	17	9	80%
	ne	21	11	80%
okvare sp.tr. pri moških	da	107	58	67%
	ne	359	199	66%
okvare sp.tr. pri ženskah	da	174	92	81%
	ne	635	357	78%

Tabela 4.6 Rezultati gradnje drevesa v 5 medicinskih domenah z in brez rezanja nezanesljivih delov drevesa.

4.3.6 REZANJE

V poglavju 3.7 so opisani razlogi za rezanje nezanesljivih delov drevesa. Z rezanjem dobimo manjša drevesa, ki so bolj razumljiva, vendar nosijo v sebi vso razpoložljivo informacijo (opozorijo zdravnika na vse možne diagnoze). Dobljena drevesa so tudi bolj natančna pri klasifikaciji novih primerov, kar potrjuje, da je rezanje res rezanje nezanesljivih delov drevesa. V tabeli 4.6 so podani rezultati gradnje z rezanjem v primerjavi z gradnjo brez rezanja nezanesljivih delov drevesa.

4.4 PRIMERJAVA ASISTENTa S STATISTIČNIMI METODAMI

Na Fakulteti za elektrotehniko in Onkološkem inštitutu v Ljubljani so bili narejeni poskusi z avtomatskim učenjem medicinskih diagnostičnih pravil z nekaterimi statističnimi metodami. Uporabljene so bile sledeče statistične metode:

- Bayesov princip verjetnosti (glej posl. 2.3), ki po določeni formuli računa verjetnosti posameznih diagnoz za dani primer. Parametri iz formule so aproksimirani z relativnimi frekvencami iz učne množice primerov.

- diskriminantna analiza (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 skupacije primerov z istimi razredi.

- metoda lupin v n-dimenzionalnem prostoru (Soklič 80),ki tvori lupine okoli skupacij primerov z istimi razredi.

V tabeli 4.7 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 usotovi 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%(69%)	-	-	72%(63%)
hepatitis	88%(90%)	-	-	80%(71%)
limfosrafija	67%	-	58%	65%
okv.ur.tr.moški	67%	-	-	67%
okv.ur.tr.ženske	79%	81%	-	81%

Tabela 4.7 Primerjava dosežene diagnostične natančnosti treh statističnih metod za avtomatsko učenje in sistema ASISTENT. Znak "-" pomeni, da ustrezni poskus ni bil izveden. V oklepajih so dodane ocenjene natančnosti, obtežene z obratnimi apriornimi verjetnostmi (glej posl. 2.4). Pri limfosrafiji so za primerjavo z metodo lupin podani rezultati eksperimentov, kjer je možnih 6 različnih diagnoz (glej 4.1.3).

4.5 PRIMERJAVA ASISTENTa Z ZDRAVNIKI SPECIALISTI

Dobljeni rezultati (diagnostična natančnost) v primerjavi z zdravniki specialisti so podani v tabeli 4.8.

domena	ASISTENT	ZDRAVNIKI
primarni tumor	46%	42%
rak na dojki	72%(63%)	64%(63%)
limfosrafija	77%	85%-ocena

Tabela 4.8 Primerjava dosežene diagnostične natančnosti sistema ASISTENT in zdravnikov specialistov. V oklepajih so dodane ocene natančnosti, obtežene z obratnimi apriornimi verjetnostmi razredov (glej 2.4).

Ob teh rezultatih bi morebiti 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. ASISTENT je splošen sistem, le poskuse smo do sedaj delali samo na podatkih iz medicine.

5. IMPLEMENTACIJA

Podrobnosti implementacije so podane v (Kononenko 85a). Tu je opisana osnovna ideja implementacije in nekatere dodatne lastnosti sistema ASISTENT.

Največja teža algoritma za gradnjo odločitvenih dreves je v izbiri najbolj informativnega atributa. Pri binarni gradnji je naloga še zahtevnejša, ker je potrebno za vsak atribut poiskati dve disjunktni podmnožici vrednosti, ki maksimizirata atributovo informativnost. Za učinkovito opravljanje te naloge je Quinlan (79) uporabljal iterativni princip, opisan v posl. 3.1, tako da je vse primere, nad katerimi je gradil drevo (okno), hranil v hitrem pomnilniku.

Ideja implementacije v ASISTENTu je ta, da primerov ni potrebno držati v hitrem pomnilniku, ampak zadostuje samo statistika primerov po atributih, vrednostih atributov in po razredih. Pri tem se intervali možnih vrednosti za zvezne atribute razbijejo na dovolj majhne podintervale, katerih indeksi predstavljajo kodo vseh vrednosti v danem podintervalu. Tako je poraba pomnilniškega prostora neodvisna od števila primerov. Npr. če želimo hraniti v hitrem pomnilniku 200 primerov, opisanih z 20 atributi, potrebujemo približno 4K pomnilniškega prostora. če je možnih 10 razredov in največ 15 možnih vrednosti za en atribut, potem za statistiko poljubnega števila primerov iz iste problemske domene porabimo tudi pribl. 4K pomnilniškega prostora.

Statistiko sestavimo vedno za vsak vozel drevesa posebej. Primeri iz vozlov so na datoteki. Ko je primerov v vozlu zadosti malo, jih prenesemo v hitri pomnilnik, da pospešimo sestavljanje statistike. Enkrat urejena statistika za en vozel omogoča hitrejše izračunavanje informativnosti atributov, kot če jo delamo posebej za vsak atribut in za vsako možno razdelitev vrednosti atributa na dve disjunktni podmnožici.

Pri binarni gradnji je kritično grupiranje vrednosti diskretnih atributov, ki kombinatorično narašča s številom različnih vrednosti atributa. V trenutni implementaciji smo se temu delno izognili tako, da diskretne atribute ocenjuje najprej normirana ocenitvena funkcija (glej 3.6). Nekaj najboljših diskretnih atributov se zatem ocenjuje z izčrpno grupacijo vrednosti v dve disjunktni podmnožici na vse možne načine. Iskanje hitrih in dovolj dobrih hevrističnih algoritmov za grupacijo vrednosti diskretnih atributov je predmet nadaljnjega dela.

V sistemu ASISTENT je implementirana tudi statistična metoda, ki temelji na Bayesovem verjetnostnem principu (glej posl. 2.3). Verjetnosti iz formule se aproksimirajo iz statistike primerov v korenu drevesa. ASISTENT omogoča uporabo statistične metode same ali v kombinaciji z zgrajenim drevesom (glej Kononenko 85a). Pri testiranju ASISTENT izračuna absolutne in relativne rezultate testiranja ter za oboje tudi ocene, obtežene z obratno apriorno verjetnostjo (glej posl. 2.4). Absolutni rezultati so kar vsota pravilnih odgovorov. Ker pa je pri domenah z nepolnimi množicami atributov nujno, da se v nekaterih listih nahajajo primeri iz več različnih razredov (glej posl. 2.2.2), potem štetje eksaktnih odgovorov ne daje vedno prave slike. Odgovor je pravilen tudi, če je pravi razred podan z določeno verjetnostjo. Relativni rezultati so dejansko vsota verjetnosti pravilnih odgovorov.

ASISTENT je implementiran v Pascalu na računalniku DEC-10 (Računalniški center univerze v Ljubljani). Obsega pribl. 5000 vrstic izvorne kode. Sestavljen je modularno in omogoča hitro spreminjanje in razširjanje. Vsak večji podprogram je bil testiran s posebnim testnim podprogramom. Celoten sistem je preizkušen na mnogo trivialnih problemih in na 6 različnih problemih iz medicine. Izvajanje programa je relativno hitro (npr. za gradnjo drevesa nad nekaj sto primeri, opisanimi z 20 atributi s po 15 možnimi vrednostmi in z 10 možnimi razredi, porabi ASISTENT nekaj sekund CPU časa na računalniku DEC-10). Koda programa je komentirana in dokumentirana z navodili za uporabnika in z navodili za programerja (Kononenko 85a).

6. ZAKLJUČKI

6.1 UPORABNOST ASISTENTA

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 naslim razvojem metodologije ekspertnih sistemov (glej npr. Bratko 82). 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 na natančnost razvrščanja (glej tabelo 4.7). V poskusih v medicinski diagnostiki je vedno dosegel klasifikacijsko natančnost zdravnikov specialistov (glej tabelo 4.8). Bistvena prednost pred statističnimi metodami je v razumljivosti odločitvenega drevesa, ker se ga da direktno interpretirati v naravnem jeziku. Odločitveno drevo se lahko uporablja brez računalnika, npr. kot priročnik za klasificiranje (diagnosticiranje). Iz drevesa je možno razbrati relacije in zakonitosti iz domene.

Izpopolnitve ASISTENTa glede na osnovni algoritem sistema ID3 so izboljšale klasifikacijsko natančnost zgrajenih dreves. Zgrajena drevesa so tudi bistveno manjša, kar vpliva na dojemljivost pravil. Izpopolnitve imajo teoretične osnove in so potrjene z eksperimenti v 5 različnih področjih medicine. ASISTENT je splošen sistem, le poskuse smo do sedaj delali na področjih iz medicine. Implementacija omogoča preprosto uporabo sistema. Tako je za sestavitev klasifikacijskih pravil iz neke nove domene potrebno le nekaj človek dni za vnos in korekturo vhodnih podatkov in po nekaj minutah lahko z zgrajenim drevesom že klasificiramo neznane primere iz dane domene.

Nad podatki za napovedovanje vremena v Londonu, ki so bili objavljeni v (Naylor 84) je ASISTENT zgradil nekoliko manjše odločitveno drevo kot sistem ACLS (Paterson in Niblett 82). Razliko je povzročila večja natančnost razločevanja vrednosti zveznih atributov v sistemu ASISTENT. Poskus je pokazal na podobnost dveh sistemov, ki sta bila razvita iz istega sistema (ID3).

6.2 NATANČNOST ASISTENTa

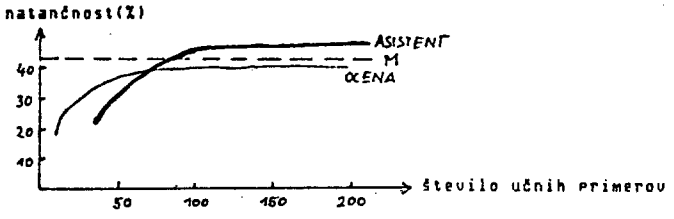
V posl. 2.2.2 je podana izpeljava ocene spodnje meje klasifikacijske natančnosti dobljenih pravil na osnovi učnih primerov s popolnim sistemom. V formuli je problematična ocena zgornje meje natančnosti (M), ki se lahko doseže z dano množico atributov pri danem klasifikacijskem problemu. Spodnjo mejo M lahko ocenimo z natančnostjo zdravnikov specialistov. V tabeli 6.1 je podana primerjava natančnosti ASISTENTa z ocenjeno spodnjo mejo natančnosti popolnega sistema.

domena	natančnost ASISTENTa	ocena spodnje meje natančnosti	upoštevani parametri M	L	N	> Pr
prim.tumor	46%	41.1%	42	18	237	0.111
rak na dojki	72%	63.9%	64	9	200	0.580
limfografija	77%	62.6%	85	14	105	0.356

Tabela 6.1 Primerjava natančnosti ASISTENTa in ocenjene spodnje meje natančnosti popolnega sistema (glej posl. 2.2.2). Meja M največje možne natančnosti razvrščanja pri dani množici atributov je ocenjena z natančnostjo zdravnikov specialistov.

Izkaže se, da ASISTENT ni presesel ocenjene spodnje meje klasifikacijske natančnosti samo pri limfografiji. Tu je potrebno opozoriti, da je natančnost zdravnikov specialistov nepreverjena ocena zdravnika specialista iz Onkološkega inštituta v Ljubljani. Iz tabele je razvidno, da ASISTENT generira dokaj zanesljiva odločitvena drevesa.

Druga zanimiva primerjava s popolnim sistemom je narejena pri gradnji dreves s spreminjanjem števila učnih primerov. Poskus smo naredili v domeni "primarni tumor" (glej 4.1.1). Slika 6.1 prikazuje rezultate poskusov z ASISTENTom in funkcijo verjetnosti pravilne klasifikacije v odvisnosti od števila učnih primerov. Obliki krivulj sta enaki, vendar zamaknjeni. Ocena spodnje meje klasifikacijske natančnosti je nezanesljiva pri majhnih množicah učnih primerov.



Slika 6.1 Primerjava natančnosti ASISTENTa z ocenjeno natančnostjo popolnega sistema pri spreminjanju števila učnih primerov v domeni "primarni tumor" (glej 4.1.1).

ZAHVALA

Slavistka Irena Roslić-Kononenko je z izčrpno lekturo prispevala k boljšemu izražanju. Zahvaljujemo se zdravniku Matjažu Zwittru iz Onkološkega inštituta v Ljubljani za eksperimentalne podatke in za strokovno pomoč pri eksperimentih v medicinskih domenah. Zdravnikom specialistom iz Clinical Investigation Unit iz Ham Green Hospital iz Bristola se zahvaljujemo za eksperimentalne podatke iz domene okvar spodnjega urinalnega trakta. Gail Gons iz Carnegie-Mellon University se zahvaljujemo za podatke iz prognostike preživetja pri hepatitisu. Milanu Sokliču iz Onkološkega inštituta v Ljubljani smo hvaležni za podatke iz prognostike ponovitve raka na dojki in iz limfografije.

LITERATURA

I.Bratko (1982) Intelisentni 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)

Diaconis, P., Efron, B. (1983) Computer-Intensive Methods in Statistics, Scientific American, vol 248

L.Gyergyek (1978) Statistične metode v teoriji sistemov in teorija o informacijah, skripta, Univerza Edvarda Kardelja, Fakulteta za elektrotehniko, Ljubljana

Kononenko, I., Bratko, I. Roškar, E.(1984) Experiments in automatic learning of medical diagnostic rules. ISSEK Workshop 84, Bled.

I.Kononenko (1985) Strukturno avtomatsko učenje, Informatika 2/85

I.Kononenko (1985a) Razvoj sistema za induktivno učenje ASISTENT, magistrsko delo, Univerza EDvarda Kardelja, Fakulteta za elektrotehniko, Ljubljana

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

P.Mulec (1980) Algoritmi za avtomatsko učenje, diplomsko delo, Univerza Edvarda Kardelja v Ljubljani, Fakulteta za elektrotehniko

Naylor, C. (1984) Discriminating experts, Practical Computing

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

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 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., Čuk, M., Abrams, P. (1985) An application of computer assisted multivariate statistical methods and artificial intelligence to the diagnosis of lower urinary tract disorders. Automatika, Zagreb (v tisku)

B.A.Shepherd (1983) An appraisal of a Decision tree approach to image classification, IJCAI

M.Soklič (1980) Računalniška diagnostika, Zaključno poročilo, Onkološki inštitut, Ljubljana

M.Zwitter (1980) Metastatični karcinom neznanega izvora, raziskovalna naloga, Onkološki inštitut v Ljubljani

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

EKSPERTNI SISTEM ZA POMOČ PRI VODENJU BANČNE LIKVIDNOSTI

Marko Bohanec, Matjaž Gams, Nada Lavrač

Institut Jožef Stefan, Jamova 39, Ljubljana

UDK : 681.3:159.953

POVZETEK: V prispevku je opisan računalniški ekspertni sistem za pomoč pri vodenju bančne likvidnosti. Uvodni del podaja kratek opis problematike vodenja bančne likvidnosti ter vlogo, funkcije in zasnovo ekspertnega sistema. V nadaljevanju je podrobno obdelan doslej realizirani del ekspertnega sistema, ki obravnava odločanje pri dnevnem vodenju likvidnosti. Opisani so vhodni podatki, izhodni rezultati, postopek obdelave, struktura sistema in primer uporabe. Podani so tudi koncepti, ki omogočajo nadgraditev sistema v smislu vključitve elementov planiranja likvidnosti.

AN EXPERT SYSTEM FOR BANK LIQUIDITY MANAGING: The paper presents a banking expert system for liquidity managing. The introductory part presents a short description of the problem area and the role, functions and design of the system. The main part of the paper describes the currently implemented part of the system which handles the bank liquidity managing in the daily decision-making. Input data, results, algorithms, structure of the system and an example of the system's use are given. The paper is concerned also with the concepts and elements, which will enable the system to manage the problems of liquidity planning.

1. UVOD

Ena od temeljnih zahtev za uspešnost bančnega poslovanja je **bančna likvidnost**. Pojem bančne likvidnosti označuje rokovno usklajenost obveznosti in terjatev ter oblikovanje ustreznih dodatnih rezerv sredstev, ki zagotavljajo plačilno sposobnost banke [1].

V sodelovanju z Ljubljansko banko - Gospodarsko banko Ljubljana, Ljubljansko banko - Stanovanjsko-komunalno banko in Visoko šolo za organizacijo dela Kranj razvijamo na Institutu "Jožef Stefan" računalniški **ekspertni sistem za pomoč pri vodenju bančne likvidnosti** [2].

Projekt sodi v okvir dolgoročnega sodelovanja na področju razvoja računalniških **ekspertnih sistemov v poslovnem planiranju in odločanju**, ki naj bi omogočali delo z nepopolnimi in nezanesljivimi podatki v hitro se spreminjajočih pogojih poslovanja. Zagotavljali naj bi tudi prilagodljivost in transparentnost računalniško podprtih postopkov ter s tem nudili podporo pri sprejemanju in izvajanju poslovnih odločitev na različnih področjih bančnega poslovanja.

Izdelavo sistema za pomoč pri vodenju bančne likvidnosti smo pričeli spomladi 1984. V prvi fazi, ki smo jo zaključili junija 1985, smo obdelali **dnevni** vidik vodenja bančne likvidnosti in ga računalniško realizirali v obliki prototipa **ekspertnega sistema za vodenje dnevne likvidnosti**.

2. PROBLEMATIKA VODENJA BANČNE LIKVIDNOSTI

Vodenje likvidnosti je odločitveni proces, ki ga v bankah vsakodnevno izvaja t.i. likvidnostna komisija. Ta na osnovi podatkov o razpoložljivih virih sredstev (npr. žiro račun banke, nakazila drugih bank) in podatkov o denarnih zahtevkih (t.i. dispozicijah, npr. krediti) izdela dnevni plan disponiranja (realizacije) sredstev, ki določa usmerjanje sredstev iz izbranih virov v izbrane dispozicije.

V splošnem mora likvidnostna komisija poiskati najugodnejšo varianto disponiranja sredstev. Ta mora biti legalna s stališča zakonov in drugih administrativnih omejitev, ki predpisujejo poslovanje banke, pokriti mora vse obvezne dispozicije, pri tem pa ohraniti likvidnost banke. Poleg tega mora biti varianta disponiranja tudi dohodkovno ugodna za banko (čim manj neplasiranih sredstev) in za njene članice.

Pomemben vidik vodenja bančne likvidnosti je tudi **planiranje likvidnosti** oziroma zagotavljanje likvidnosti banke v daljšem časovnem obdobju [1]. Likvidnostna komisija lahko s svojimi dnevnimi odločitvami vpliva na gibanje likvidnosti v naslednjih dneh. Odločitev, ki za dani dan morda ni najugodnejša, lahko vpliva na izboljšanje likvidnostnega stanja v prihodnosti in obratno. Na planiranje likvidnosti v veliki meri vplivajo lastnosti denarnih tokov, tedenska, mesečna in letna gibanja sredstev ter spremembe in drugi vplivi okolja (npr. spremembe obrestnih mer). Omeniti velja, da se mora likvidnostna komisija marsikdaj odločati na osnovi nenatančnih in negotovih podatkov o razpoložljivih virih in dispozicijah.

V vsakodnevni praksi poteka vodenje dnevne likvidnosti zadovoljivo, vendar pa ob tem le redko srečujemo elemente planiranja likvidnostnega poslovanja [1]. Obstoječi sistem vodenja likvidnosti sloni na značilnem "ekspertnem" znanju, kjer strokovnjaki večinoma vedo kako ukrepati, vendar to znanje ni formalizirano in prenosljivo. Tako znanje se tudi teže oplaja z zbiranjem informacij, ki so bistvene za planiranje likvidnostnega poslovanja. Navedeni lastnosti se pokažeta kot pomanjkljivosti zlasti v kritičnih likvidnostnih situacijah in ob odsotnosti katerega od pomembnejših strokovnjakov, ko se prisotni člani likvidnostne komisije le stežka odločajo, ker jim za kompetentno odločitev manjka argumentov.

Za zagotavljanje bančne likvidnosti je potrebno temeljito poznavanje gospodarskega stanja, gibanja realnih in finančnih tokov, učinkov ukrepov ekonomske in še posebej denarno-kreditne in devizne politike, razmer na denarnem in kreditnem tržišču, itd. Računalniški ekspertni sistem za vodenje likvidnosti nima namena in ne more nadomestiti bančnih strokovnjakov pri odločanju, lahko pa jim služi kot pomočnik oziroma nevtralni konzultant, ki skrbi za boljšo dokumentiranost zahtev po angažiranju in plasiranju sredstev, za predvidevanje določenih denarnih tokov na osnovi preteklih trendov in sprememb okolja ter za boljšo dokumentiranost in argumentiranost odločitev.

3. ZASNOVA EKSPERTNEGA SISTEMA ZA VODENJE BANČNE LIKVIDNOSTI

Navedimo **funkcije**, s katerimi bi lahko ekspertni sistem za vodenje bančne likvidnosti pomagal pri odločanju likvidnostne komisije:

- omogočal bi **preizkušanje različnih možnosti** usmerjanja razpoložljivih in ocenjenih prilivov sredstev v posamezne vrste plasmajev oziroma odlivov sredstev, ocenjeval bi likvidnostne, dohodkovne in druge učinke variant ter opravljal vsa potrebna računska opravila,

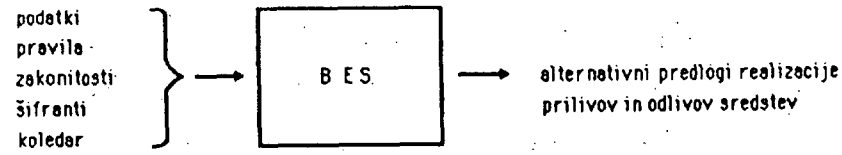
- omogočal bi **utemeljitev in razlago** predlaganih odločitev ter nazorno prikazovanje rezultatov,

- omogočal bi **sistematično vodenje** podatkov in informacij o razpoložljivih in predvidenih/planiranih prilivih in odlivih na daljši rok ter **formalizacijo** po možnosti vseh (tudi subjektivnih) kriterijev sprejemanja odločitev, kar bi zagotavljalo bolj sistematično in objektivno odločanje,

- zagotavljal bi enostavno **spreminjanje** sistema in podatkov ter s tem hitro **prilagajanje** na nove razmere,

- omogočal bi večjo **dokumentiranost** vodenja podatkov in odločanja, poleg tega pa bi bili v računalnik vpisani podatki primerni za vodenje raznih statistik,

- omogočal bi **prenosljivost in večjo kvaliteto** odločitvenega znanja, saj bi se to znanje formaliziralo v obliki vgrajenih pravil in drugih odločitvenih mehanizmov.



Slika 1: Zasnova bančnega ekspertnega sistema

Ekspertni sistem za vodenje bančne likvidnosti BES (Bančni ekspertni sistem) lahko poenostavljeno prikažemo s sliko 1.

Vhodni podatki obsegajo **podatke** o tekočih in planiranih virih in dispozicijah, **pravila** odločanja, **zakonitosti** poslovanja, **šifrante** virov, dispozicij, obrestnih mer, prioritet, itd., ter opis tipičnih denarnih tokov, vezanih na **koledar**.

Izhode iz sistema predstavljajo alternativni predlogi realizacije prilivov in odlivov sredstev. Predlogi omogočajo izbor alternative, ki naj zagotavlja:

- čimboljšo likvidnost,
- kreditno sposobnost,
- realizacijo prioritet,
- regularnost poslovanja,
- čimvečji dohodek.

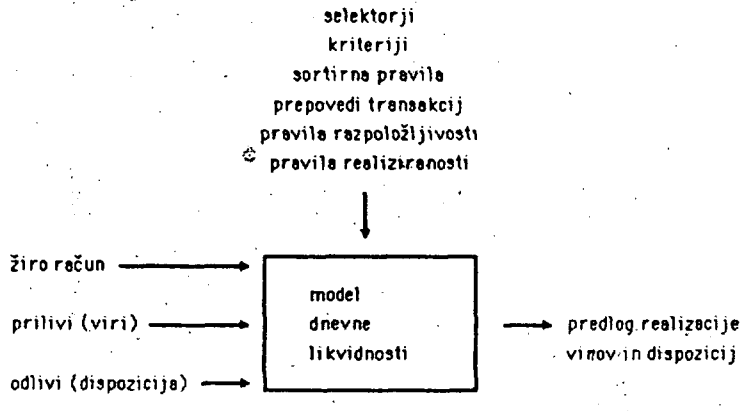
Sistem za vodenje dnevne likvidnosti imenujemo tisti del sistema BES, ki naj bi pomagal bančnim strokovnjakom pri tekočem poravnavanju vseh dospelih obveznosti, t.j. pri zagotavljanju tekoče (dnevne) plačilne sposobnosti banke. **Sistem planiranja likvidnosti** pa imenujemo podsistem, ki skrbi za pomoč pri kontinuiranem vodenju optimalne likvidnosti banke v daljšem časovnem obdobju.

4. EKSPERTNI SISTEM ZA VODENJE DNEVNE LIKVIDNOSTI

K izgradnji bančnega ekspertnega sistema za vodenje likvidnosti smo pristopili postopno. V prvi fazi smo začeli z izgradnjo sistema za vodenje dnevne likvidnosti. Ta faza je obsegala podrobno analizo posameznih elementov sistema, t.j. natančno opredelitev in razčlenitev posameznih vrst in oblik prilivov in odlivov sredstev, njihovih značilnosti, kriterijev, pogojev, omejitev in drugih dejavnikov pri odločanju, opredelitev želenih izhodov iz sistema, itd. O tem je bila opravljena analiza dejanskega stanja, zbrano je bilo ekspertno znanje ter definirani problemi, ki jih je potrebno rešiti pred pričetkom realizacije modela planiranja likvidnosti.

Kot rezultat prve faze smo izdelali **prototip ekspertnega sistema za vodenje dnevne likvidnosti** [2]. Jedro tega sistema je **model dnevne likvidnosti.** Ostale komponente sistema predstavljajo komunikacijski vmesnik in drugi pomožni programi, ki omogočajo uporabo modela na realnih podatkih.

Model dnevne likvidnosti iz vhodnih podatkov, znanja in pravil generira predlog realizacije virov in dispozicij. Prikazan je na sliki 2.



Slika 2: Model dnevne likvidnosti

4.1 Vhodni podatki modela dnevne likvidnosti

Vhodni podatki za model dnevne likvidnosti so naslednji:

A. Vhodne postavke

- razpoložljiva sredstva na **žiro računu** (stanje preteklega dne) ter razpoložljivi **viri** (prilivi sredstev) na dani dan, ki so razčlenjeni na:
  - ime vira,
  - oblika (tip) vira,
  - izvor sredstev,
  - znesek,
  - datum pričakovanja razpoložljivosti vira (predviden za uporabo v modelu planiranja likvidnosti; v modelu dnevne likvidnosti je to vedno obravnavani datum),
  - datum pričakovanja vrnitve vira (predviden za model planiranja likvidnosti in se v modelu dnevne likvidnosti ne uporablja),
  - obresti oz. stroški za izkoriščanje vira,
  - prioriteta vira,
  - število dni uporabnosti vira;
- **dispozicije** (zahteve po odlivih sredstev) na dani dan, ki so razčlenjene na:
  - ime dispozicije,
  - oblika (tip) dispozicije,
  - ponor sredstev,
  - znesek,
  - datum pričakovanja zahteve po dispoziciji (v modelu dnevne likvidnosti je to vedno obravnavani datum),
  - datum pričakovanja vrnitve dispozicije (v modelu dnevne likvidnosti se ta podatek ne uporablja),
  - obresti oz. dohodek od realizirane dispozicije,
  - prioriteta dispozicije,
  - število dni možnega zadrževanja dispozicije.

B. Pravila

Pravila predstavljajo **bazo znanja** sistema za vodenje dnevne likvidnosti. Z njimi je mogoče vplivati na delovanje sistema in s tem na rezultate, ki jih generira. Obstaja več tipov pravil:

- **Selektorji** razvrstijo posamezne vire/dispozicije v kategoriji "prepovedan" (ki naj se ne realizirajo) in "obvezen" (ki jih je nujno treba realizirati). Uporabnik lahko izbere dispozicije, ki se morajo ali ki se ne smejo realizirati. Določi lahko omejitve, ki se tičejo ekonomskih učinkov, vrednostnega obsega posameznih dispozicij, nosilcev oziroma uporabnikov, oblike (tipov), rokov, itd., ali pa tako, da eksplicitno navede določeni vir ali dispozicijo. Na tej osnovi program selekcionira vire in dispozicije tako, da izloči iz nadaljnje obdelave tiste, ki so se uvrstili v kategorijo prepovedani, obveznim virom in dispozicijam pa določi prvo prioriteto za realizacijo.

- **Kriteriji** razvrstijo vire in dispozicije, ki jih niso predhodno zajeli oziroma izločili selektorji, v prioritetne razrede. Kriteriji, ki lahko predstavljajo vsebinsko podobne elemente kot selektorji, so v bazi znanja urejeni po pomembnosti. Vsak kriterij razbije vire oz. dispozicije na dve množici: prvo, ki zadošča pogojem kriterija in se razporedi v dani prednostni razred, in drugo množico, ki jo bodo urejevali preostali kriteriji.

- **Sortirna pravila** omogočajo urejanje podatkov glede na relacijo "večji" ali "manjši". Običajno se ta relacija nanaša na prioritete postavk, v primerih enakih prioritet pa na obrestno mero (ceno) vira/dispozicije.

- **Prepovedi transakcij** določajo pogoje, pri katerih naj se transakcije virov v dispozicije ne izvršijo. Transakcijo lahko prepovemo eksplicitno z navedbo virov in dispozicij ali implicitno z navedbo njihovih lastnosti.

- **Pravila razpoložljivosti** določajo minimalni oz. maksimalni znesek sredstev (virov ali dispozicij), ki mora ostati nerealiziran (nerazporejen). Uporabo teh pravil predvidevamo v modelu planiranja likvidnosti za primere, ko je potrebno za neko planirano dispozicijo zagotoviti dovolj sredstev (virov) že nekaj dni pred njeno dejansko realizacijo.

- **Pravila realiziranosti** določajo minimalno oz. maksimalno vsoto sredstev (virov ali dispozicij), do katere se transakcije še lahko izvršujejo. Ta pravila so uporabna pri upoštevanju kreditnih limitov.

Pravila so zapisana kot dejstva programskega jezika prolog [3] in imajo naslednjo splošno obliko:

```
rul(Datum,Tip_pravila,Zahteve).
```

Z argumentom Datum je določen dan veljavnosti pravila. Če veljavnost pravila ni vezana na datum, uporabimo oznako '_'. Tip_pravila označuje enega od zgoraj naštetih tipov pravil. Pravilo velja za vse tiste vire/dispozicije, ki ustrezajo seznamu Zahteve.

Kot primer navedimo selektor, ki prepoveduje obravnavo dispozicij, katerih znesek je manjši od 50 tisoč dinarjev:

```
rul(_, prep(1), [disp,[znesek:[manjsi(50)]]]).
```

4.2 Postopek realizacije virov in dispozicij

Model dnevne likvidnosti na osnovi vhodnih postavk (virov in dispozicij) in definiranih pravil generira predlog realizacije virov in dispozicij, ki pove, katere vire sredstev velja izkoristiti in katere dispozicije realizirati s tako pridobljenimi sredstvi. Predlog realizacije virov in dispozicij je rezultat več procesov:

- **priprava podatkov:** branje vhodnih postavk in pravil;
- **urejanje vhodnih postavk:**
  - selekcija (na osnovi selektorjev) razporedi vire in dispozicije na prepovedane (ki se po tem po procesu izločijo iz obravnave) in obvezne (ki jih sistem mora realizirati),
  - razvrščanje preostalih postavk v prioritetne razrede (na osnovi kriterijev),
  - urejanje virov in dispozicij v posameznih razredih po prioriteti in ceni (le-ti sta lastnosti vhodnih postavk);
- **realizacija** pretakanja sredstev iz razpoložljivih virov v dispozicije, ob upoštevanju pravil in urejenosti postavk v razrede oz. v okviru posameznih razredov;
- **izpis rezultatov** realizacije.

4.3 Generiranje alternativnih predlogov realizacije

Sistem za dane vhodne podatke generira eno alternativo. Uporabnik lahko zahteva generiranje alternativnih predlogov, ki jih potem med seboj primerja. Za generiranje alternativnih rešitev obstaja več načinov:

Prvi način generiranja alternativnih rešitev je sprememba prioritetnega vrstnega reda virov in dispozicij. Začetni prioritetni vrstni red je določen s prioritetami, selektorji in kriteriji. Uporabnik ima možnost vsiliti sistemu drugačen vrstni red. V tem primeru model dnevne likvidnosti preskoči proces urejanja postavk in privzame vsiljeni vrstni red.

Drugi način generiranja alternativnih rešitev pa je ta, da uporabnik spremeni pravila, ki so vire in dispozicije selekcionirala ter razvrstila po prioriteti. Spremembo pravil je mogoče opraviti na dva načina: z izbiro celotnega novega nabora pravil (sistem dopušča več predefiniranih naborov pravil, uporabnih v različnih odločitvenih oz. likvidnostnih situacijah) ali pa samo z delnimi popravki trenutno veljavnega nabora pravil.

Primerjava alternativnih rešitev lahko uporabniku pomaga pri argumentiranem izboru prave rešitve. Vsaka rešitev je namreč opremljena tudi s podatkom o skupnem profitu realizacije, ki je v sistemu, v katerem ni elementov planiranja, pri odločanju edini merodajen (v kolikor so bili seveda realizirani vsi nujni zahtevki po odlivih sredstev).

4.4 Realizacija sistema za vodenje dnevne likvidnosti

Model dnevne likvidnosti smo realizirali v programskem jeziku prolog [3] (konkretno uporabljeni prolog je del sistema poplog, ki teče na računalniku VAX 750 na Institutu Jožef Stefan). Z uporabo prologa smo dosegli možnost enostavnega spreminjanja in prilagajanja programa novim zahtevam ter veliko učinkovitost programiranja. Kljub temu, da model opravlja vrsto razmeroma zahtevnih nalog, njegova velikost ne presega 1200 vrstic.

Pomožni programi za preoblikovanje podatkovnih datotek so realizirani v pascalu in skupaj obsegajo okrog 500 vrstic.

Zaradi prilagodljivosti sistema na morebitne spremembe vhodnih podatkov smo za vnos podatkov v prototipni verziji sistema uporabili sistemski urejevalnik EDT.

Za usklajeno delovanje navedenih komponent sistema skrbijo DCL komandne procedure [4], ki tudi vodijo celotno komunikacijo med uporabnikom in sistemom. Procedur je 24 in skupaj obsegajo 400 ukazov.

4.5 Primer uporabe sistema za vodenje dnevne likvidnosti

Oglejmo si poenostavljen primer vhodnih podatkov in rezultatov sistema za pomoč pri vodenju dnevne likvidnosti. Vzemimo, da smo na dan 14. november 1984 razpolagali z viri, prikazanimi v tabeli 1. Vir številka 1 je žiro račun (izvor sredstev je LB-GBL) v znesku 50.000.000 dinarjev. Tolikšen znesek na žiro računu je na razpolago samo ta dan. Ker sredstev ne vračamo, datum vrnitve ni naveden (0 0 0). Čas uporabe sredstev iz tega vira ni omejen.

Prilivov drugih sredstev ne pričakujemo, imamo pa možnost črpanja rezervnega sklada (prvi dan, zato ga lahko črpamo še 14 dni, vendar pa moramo črpani znesek vrniti že naslednji dan) in sredstev primarne emisije za kreditiranje izvoza.

Za isti dan naj bi pokrili zahtevke plačilnega prometa, likvidature, sektorja za poslovanje s tujino za nakazilo dinarske protivrednosti članicam za obvezno izločene devize za potrebe federacije in zahtevek za kreditiranje izvoza članic, kot to prikazuje tabela 2.

Na osnovi teh podatkov je sistem generiral predlog realizacije, prikazan v tabeli 3. Viri in dispozicije so izpisani po prioritetnem vrstnem redu, ki ga je generiral sistem. Pri razvrščanju so bile upoštevane samo prioritete posameznih postavk, medtem ko kriteriji in selektorji v tem primeru niso bili definirani (uporabljen je bil "prazen" nabor pravil, brez selektorjev in kriterijev).

Ker je bil skupni znesek virov manjši od zneska dispozicij, so ostali izvozni krediti delno nepokriti. Ob odločitvi za to varianto disponiranja bi bil tega dne dobiček banke (razlika med prihodkom od dispozicij in odhodkom za vire) 41.670 dinarjev.

Na osnovi teh rezultatov smo se odločili, da preizkusimo tudi varianto, ko v celoti realiziramo izvozne kredite, obveznosti po deviznem režimu pa zadržimo za en dan. To storimo tako, da s pravilom zahtevamo, da izvozni krediti postanejo obvezni ali pa, tako kot je prikazano v naslednjem primeru, izvoznim kreditom "vsilimo" večjo prioriteto od obveznosti po deviznem režimu. Rezultate prikazuje tabela 4.

V tem primeru je dobiček banke na ta dan skoraj trikrat večji, zato pa bo morala likvidnostna komisija obveznosti članicam po deviznem režimu naslednji dan obravnavati kot bistveno prioritetnejše oz. obvezne, ker jih ne bo več mogoče odlagati. Če sredstev ne bo dovolj, si jih bo potrebno izposoditi ne glede na njihovo ceno in druge pogoje. To pa so že dileme, ki jih bo pomagal reševati model planiranja likvidnosti, medtem ko jih pri modelu dnevne likvidnosti še vedno v celoti prepuščamo človeku.

5. ZASNOVA EKSPERTNEGA SISTEMA ZA PLANIRANJE LIKVIDNOSTI

To poglavje govori o zasnovi celotnega sistema za vodenje in planiranje likvidnosti, ki še ni realiziran, je pa že konceptualno zasnovan.

Sistem za vodenje dnevne likvidnosti kot tudi načrtovani sistem za planiranje likvidnosti smo zasnovali po metodologiji ekspertnih sistemov [5,6]. Komponente načrtovanega bančnega ekspertnega sistema BES so prikazane na sliki 3.

Uporabniški vmesnik bo skrbel za komunikacijo med uporabnikom in sistemom ter omogočal vnos podatkov, izbor tekoče uporabnih podatkov, izbor tekoče uporabnih pravil, izpis rezultatov, grafično predstavitev rezultatov in njihovo razlago.

ST.	IME	TIP	IZV	ZNES	DAT_PRI	DAT_VRN	OBR	PRI	D_UP
1	zr	zr	gb	50000	14 11 84	0 0 0	7	10	neom
2	rs	rs	gb	800000	14 11 84	15 11 84	0	20	14
6	nak_nb	reesk_kr_izv	nb	100000.	14 11 84	n n n	31	20	neom

Tabela 1: Primer vhodnih podatkov o virih

ST.	IME	TIP	PON	ZNES	DAT_PRI	DAT_VRN	OBR	PRI	D_ZD
1	plac_pr	plac_pr	gb	350000	14 11 84	0 0 0	0	220	1
2	likv	likv	gb	100000	14 11 84	0 0 0	0	100	0
3	ost_spt	dev_nb	cl	350000	14 11 84	0 0 0	0	460	1
4	izv_kr	ost_izv	cl	230000	14 11 84	n n n	33	640	neom

Tabela 2: Primer vhodnih podatkov o dispozicijah

VIRI

št.	pri.	razred	ime	tip	realiz.	nereal.
1	1	prosti	zr	zr	50000	0
6	2	prosti	nak_nb	reesk_kr_izv	100000	0
2	3	prosti	rs	rs	800000	0
Vsota:					950000	0

DISPOZICIJE

št.	pri.	razred	ime	tip	realiz.	nereal.
2	1	prosti	likv	likv	100000	0
1	2	prosti	plac_pr	plac_pr	350000	0
3	3	prosti	ost_spt	dev_nb	350000	0
4	4	prosti	izv_kr	ost_izv	150000	80000
Vsota:					950000	80000

Profit: 41.670

Tabela 3: Primer izhodnih rezultatov realizacije virov in dispozicij

VIRI

št.	pri.	razred	ime	tip	realiz.	nereal.
1	1	prosti	zr	zr	50000	0
6	2	prosti	nak_nb	reesk_kr_izv	100000	0
2	3	prosti	rs	rs	800000	0
Vsota:					950000	0

DISPOZICIJE

št.	pri.	razred	ime	tip	realiz.	nereal.
2	1	prosti	likv	likv	100000	0
1	2	prosti	plac_pr	plac_pr	350000	0
4	4	prosti	izv_kr	ost_izv	230000	0
3	3	prosti	ost_spt	dev_nb	270000	80000
Vsota:					950000	80000

Profit: 115.000

Tabela 4: Primer izhodnih rezultatov z vsiljeno prioriteto postavk

Mehanizmi sklepanja naj bi omogočili aktivno uporabo baze podatkov in baze znanja. Vsebovali naj bi več komponent, od katerih sta najpomembnejši **model dnevne likvidnosti** in **model planiranja likvidnosti**. Slednji je zasnovan kot "nadzornik", ki modelu dnevne likvidnosti prireja podatke in zastavlja cilje. Model izračunava prilive in odlive sredstev na osnovi že realiziranih tokov in pričakovanih dogodkov ter jih skuša na dani dan v modelu dnevne likvidnosti razporediti tako, da zagotovi optimalno vodenje likvidnosti za daljše časovno obdobje. Poleg nadzornega podsistema bodo mehanizmi sklepanja vsebovali tudi **podsistem za simulacijo dnevnih denarnih tokov** (za napovedovanje najbolj verjetnih posledic realizacije dispozicij danega dne na obseg in strukturo virov naslednjega dne) in **podsistem za statistično napovedovanje likvidnosti**. Statistično napovedovanje je koristno dopolnilo v primerih, ko postanejo podatki o planiranih virih in dispozicijah zaradi časovne odmaknjenosti izrazito nepopolni in s tem neprimerni za model dnevne likvidnosti.

Baza znanja bo vsebovala parametrizirano znanje o problematiki bančne likvidnosti. Znanje bo zapisano v obliki **pravil**, ki opisujejo zakonitosti bančnega poslovanja, kot so ekonomske in statistične zakonitosti, zakonitosti denarnih tokov, predpisi, pravila in izkušnje bančne prakse, itd. Baza znanja bo obsegala tudi **šifrante** za parametriziran opis imen in tipov postavk, obrestnih mer, prioritet postavk in terminskih obveznosti.

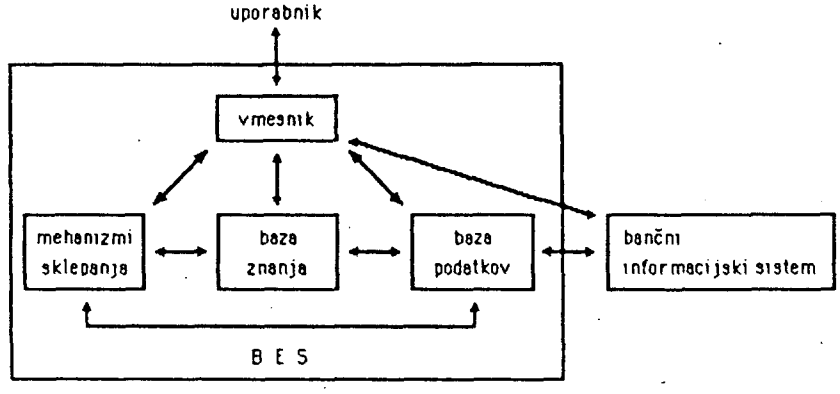
Bazo podatkov naj bi sestavljali podatki o tekočih in predvidenih virih/dispozicijah ter pravila, ki bi dajala uporabniku možnost interveniranja v delovanje programa, t.j. v izvajanje modela dnevne likvidnosti in modela planiranja likvidnosti. Uporabniku bi omogočala definiranje posebnih zahtev in pogojev, ki morajo veljati nad določenimi viri/dispozicijami. Tovrstna pravila smo uporabili in opisali že pri modelu dnevne likvidnosti. Zanje velja, da so prioritetnejša od pravil v bazi znanja in izrazito "lokalna", saj veljajo le za izbrane postavke določenega dne.

V viziji prihodnjega razvoja sistema predvidevamo **povezanost** sistema BES **z bančnim informacijskim sistemom**. Kot je razvidno iz slike 3, naj bi bila oba sistema povezana preko baze podatkov sistema BES, ki bi se polnila in ažurirala na osnovi podatkov bančnega informacijskega sistema.

6. ZAKLJUČEK

Realizirali smo prototip ekspertnega sistema za vodenje dnevne likvidnosti. Delo je bilo opravljeno z naslednjimi nameni:

- v kratkem času realizirati delujoči programski sistem, na katerem je možno preveriti, kako lahko uporaba takega sistema pripomore h kvaliteti odločitev likvidnostne komisije;
- realizirati sistem, ki ga je mogoče enostavno dopolnjevati z novimi spoznanji;
- preizkusiti sistem v praksi, ugotoviti njegove pomanjkljivosti ter ga nadalje razvijati na osnovi pridobljenih izkušenj.



Slika 3: Struktura bančnega ekspertnega sistema

Razvoj prototipa je pokazal, da je z uporabo sistema za vodenje dnevne likvidnosti možno generirati alternativne rešitve za prelivanje sredstev iz virov v dispozicije ter jih med seboj primerjati na osnovi dohodkovnih, kasneje pa tudi drugih kriterijev. S tem se uporabnik izogne ročnemu računanju dohodkovnega učinka izbrane alternative, prav tako pa mu sistem omogoča upoštevanje in dokumentiranost vseh zahtev po dispozicijah ter pričakovanih virih, na katere bi lahko uporabnik pozabil. Dokumentiranost teh zahtev ter predlagane rešitve, ki jih daje sistem, omogočajo argumentirano svetovanje tudi v primeru odsotnosti katerega od članov likvidnostne komisije, ki bi bil sicer nujno potreben pri sprejemanju in utemeljevanju odločitve v kritičnih likvidnostnih situacijah.

Z uporabniškega stališča se prednosti sistema kažejo predvsem v veliki prilagodljivosti sistema hitrim spremembam bančnega okolja in željam uporabnika. Uporabnik lahko spreminja tako podatke kot pravila. Z definiranjem različnih naborov pravil lahko prilagaja sistem različnim načinom delovanja. Manjše spremembe, vezane na posamezno postavko, so omogočene s spreminjanjem prioritet in vsiljevanjem uporabnikovega predloga razvrstitve postavk.

Z računalniškega stališča je sistem zanimiv predvsem zaradi prilagodljivosti, ki so jo omogočili koncepti logičnega programiranja in ekspertnih sistemov [5,6]. Odražajo se na implementacijskem (npr. enostavno spreminjanje oblike in pomena vhodnih podatkov) in uporabniškem nivoju (prilagodljivost, dosežena s spremembami pravil). Posebej velja poudariti tudi produktivnost programiranja v prologu. Relativno zahtevni in sposobni sistem je bil prototipno realiziran v izredno kratkem času in takoj uporaben v praksi.

Nadaljnji razvoj sistema bo obsegal izboljšave na osnovi pripomb uporabnikov in drugih izkušenj pri njegovi praktični uporabi, integracijo sistema v obstoječi bančni informacijski sistem in nadgraditev sistema z elementi planiranja likvidnosti [1,2].

ZAHVALA

Prvo fazo projekta sta financirali Ljubljanska banka - Gospodarska banka Ljubljana in Ljubljanska banka - Stanovanjsko-komunalna banka, delo pa je podprla tudi Raziskovalna skupnost Slovenije. Zahvaljujemo se Jasni Pipan za strokovno pomoč na področju bančne likvidnosti, Marku Grobelniku za njegov prispevek pri implementaciji sistema in sodelavcem Ivanu Bratku, Jelki Gorjup, Iztoku Lajovicu, Vladislavu Rajkoviču in Francu Žerdinu, ki so s svojim znanjem in izkušnjami odločilno pripomogli k doseženim rezultatom pri načrtovanju in realizaciji sistema.

LITERATURA

[1] Gorjup, J.: PROBLEMI IN PRISTOP K OBLIKOVANJU RAČUNALNIŠKO ZASNOVANEGA INFORMACIJSKEGA SISTEMA ZA POTREBE ZAGOTAVLJANJA PLAČILNE SPOSOBNOSTI TEMELJNE BANKE, XIV. posvetovanje o ekonomiki in organizaciji združenega dela, Portorož, 1985.

[2] N. Lavrač, M. Bohanec, M. Grobelnik, J. Pipan: EKSPERTNI SISTEM ZA PLANIRANJE BANČNE LIKVIDNOSTI, IJS delovno poročilo DP-3930, Institut Jožef Stefan, Ljubljana, 1985.

[3] Clocksin, W.F., Mellish, C.S.: PROGRAMMING IN PROLOG, Springer-Verlag, 1981.

[4] VAX/VMS GUIDE TO USING COMMAND PROCEDURES, Digital Equipment Corporation, 1982.

[5] Bratko, I.: INTELIGENTNI INFORMACIJSKI SISTEMI, Univerza Edvarda Kardelja, Fakulteta za elektrotehniko, Ljubljana, 1981.

[6] Bratko, I.: EXPERT SYSTEMS AND PROLOG, Supercomputer Systems Technology (ed. F. Sumner), Infotech State of the Art Report, Vol. 10, No. 6, 1982.

ANALITIČKI POSTUPCI U ODREDIVANJU NEPRODUKTIVNE OBRADE ZA JEDNU VRSTU PROCESNIH RAČUNARSKIH SUSTAVA

Nikola Bogunović, Institut "Ruđer Bošković", Zagreb

UDK : 681.3.012

SAŽETAK: U radu se razmatraju nekonzervativni procesni računarski sustavi s jednorazinskim prioritetnim rasporedivanjem grupa ulaznih događaja. Pokazuje se da, pri visokim ulaznim intenzitetima, računarski sustav troši značajan dio ukupnog vremena na neproduktivan rad oko pripreme i prihvata događaja, te reorijentacije na istu ili novu obradu. Za neke jednostavnije primjere procesnih računarskih sustava dati su postupci za određivanje globalnih radnih indeksa koji najvjernije opisuju aktivnost sustava i omogućuju ocjenu efikasnosti.

ANALYTICAL PROCEDURES OF ESTIMATION THE OVERHEAD PROCESSING TIME FOR A CLASS OF REAL-TIME COMPUTER SYSTEMS: In this paper nonconservative real-time computer systems with single level priority scheduling of input events are considered. It is shown that under high data arrival rates computer consumes a considerable portion of the total system time on nonproductive work such as event acquisition and set up conduct for the continuation of the processing. The procedures for estimating the global descriptor indexes, which most accurately characterize the performance of the system and enable measurement of the efficiency, are given for some prevalent models.

1. UVOD

Procesni, ugrađeni računarski sustavi djeluju u sredini s vlastitim dinamičkim karakteristikama, koje na taj način nameću ograničenja i na procese unutar računala. U takvim sustavima gotovo u pravilu susrećemo više asinhronih zadataka koji se u strukturi računala s jednim procesorom međusobno prekidaju kako bi poslužili ulazno-izlazne ili druge prioritetne zahtjeve. Prema sadašnjem stanju grade i organizacije takvih sustava može se uočiti da često usvojena pretpostavka o konzervativnosti predstavlja samo grubu aproksimaciju realnih sustava. Pri visokim intenzitetima dolazaka vanjskih događaja, računarski sustav troši značajan dio ukupnog vremena na neproduktivan rad oko suspendiranja procesa u toku, pripreme i prihvata novog ulaznog događaja, te reorijentacije na istu ili novu obradu. Razumljivo je da takav neproduktivan rad degradira osnovne parametre sustava (odzivno vrijeme ili propusnost, broj neobrađenih događaja u čekanju i sl.) U ovom radu istražit će se utjecaj neproduktivne obrade na efikasnost računarskih sustava koji prihvaćaju niz različitih događaja iz ulaznih jedinica koje su direktno vezane na tehnološki, mjerni ili neki drugi proces.

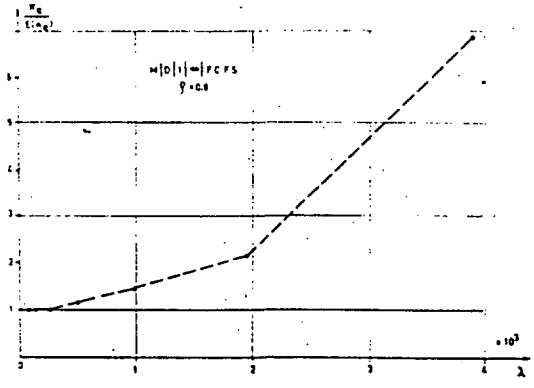
Razmatrani model ugrađenih računarskih sustava sadrži velik broj izvora događaja $\{A_{ij}\}$ koji su razdijeljeni u k prioritetnih grupa ($i=1,2,\ldots,k$) prema osnovnim fizikalnim veličinama koje predstavljaju. Općenito gledano, ulazni procesi pokazuju stohastička obilježja i najvjernije se mogu modelirati nezavisnim Poissonovim procesima /1/, /2/. U razmatranju procesa obrade često se može pretpostaviti homogena populacija pristiglih događaja. Razdiobe vremena obrade homogenih događaja imaju jednak oblik ali različite intenzitete po grupama, međutim točan oblik te razdiobe nije jednostavno odrediti. Zbog toga će se u nastavku analizirati sustav s općom razdiobom vremena obrade iako to unosi određene poteškoće u matematičko analitičke postupke. Prema standardnim oznakama u teoriji repova, predmet analize biti će sustavi koji se mogu opisati modelom M/G/1 s k grupa prioritetno raspoređenih izvora slučajnih događaja (i=1 ima najveći prioritet).

Nesmetano odvijanje nekoliko nezavisnih zadataka u računarskom sustavu zahtjeva da se ovi procesi mogu prekinuti i nastaviti kasnije s jednoznačnim konačnim rezultatom. Stanje sustava nakon k koraka možemo označiti višedimenzionalnim vektorom $\underline{s}_k$. Stanje sustava izraženo vektorom $\underline{s}_{k+1}$, nakon k+1 koraka, jednoznačno je određeno prethodnim stanjem $\underline{s}_k$. Uvjet za ovakvo ponašanje jest, da se stanje procesa između prekidanja i njegovog nastavka sačuva, da procesi u računarskom sustavu jedan drugom ne mijenjaju stanja, te da se svaki odvija na vlastitom skupu registara i memorijskih lokacija. Budući da sačuvanje i obnavljanje stanja zadataka traži konačno vrijeme, posebice u sus-

tavima s jednim kompletom registara, to je računarski sustav nekonzervativan.

2. UTJECAJ NEPRODUKTIVNE OBRADE NA GLOBALNE PARAMETRE SUSTAVA

Kvantitativan utjecaj neproduktivne obrade, zbog reorijentacije na novi zadatak možemo razmotriti na primjeru sustava s disciplinom odabiranja događaja u obradu prema redoslijedu dolaska (FCFS). Na slici 1, dati su rezultati mjerenja u M/D/1/FCFS sustavu.



Slika 1. $E(n_q)/\overline{n}_q$ u M/D/1FCFS sustavu

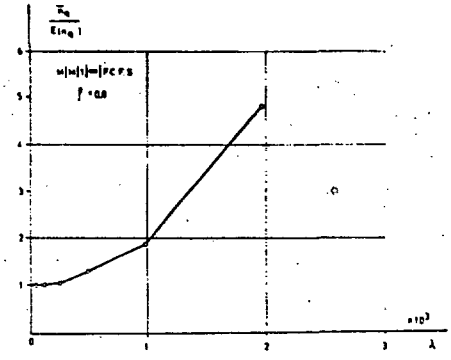
Na apscisi je dat intenzitet dolazaka događaja, a na ordinati omjer izmjerenog srednjeg broja neobradenih dogadaja u repu čekanja ($\overline{n}_q$) prema izračunatoj i izmjerenoj očekivanoj vrijednosti $E(n_q)$, za stacionaran sustav bez utjecaja neproduktivne obrade. Očekivanje $E(n_q)$ izračunato je za M/D/1/FCFS sustav iz Pollaczek-Khinchin izraza.

Eksperiment je izveden na računarskom sustavu DEC PDP-11/03L. Ulazni Poissonov proces ostvaren je pseudo-slučajnom binarnom sekvencom generiranom iz posmičnog registra s linearnim povratnim vezama. Dobivena diskretna razdioba poslužila je kao aproksimacija eksponencijalne kontinuirane razdiobe Poissonovog procesa /2/.

Srednje odzivno vrijeme, kao moment kontinuirane slučajne veličine, teško je mjerljiv direktno, već se može odrediti iz srednjeg broja neobradenih dogadaja u čekanju, koristeći teorem J.D.C.Littlea, koji predstavlja invarijantu u sustavima s repovima čekanja.

Na slici 1 opažamo da pri niskim intenzitetima $\overline{n}_q = E(n_q)$, što je i razumljivo jer neproduktivna obrada nije zamjetljiva. S povećanjem intenziteta računarski sustav troši proporcionalno sve više vremena na neproduktivan rad, te razlika $\overline{n}_q - E(n_q)$ postaje sve veća.

Na slici 2 prikazan je utjecaj neproduktivne obrade u sustavu M/M/1/FCFS, t.j. s eksponencijalnom razdiobom vremena izmedu pojave događaja i eksponencijalnom razdiobom vremena obrade. Opažamo strmi porast omjera izmjerene srednje vrijednosti broja događaja u repu čekanja i očekivane vrijednosti u stacionarnom sustavu bez utjecaja neproduktivne obrade.



Slika 2. $E(n_q)/\overline{n}_q$ M/M/1/FCFS sustavu

U oba mjerenja odabran je faktor iskorištenja sustava $\rho = 0.8$. To je umnožak intenziteta ulaznih događaja i očekivanog vremena obrade $E(x)$ i predstavlja ustvari srednji uneseni posao u sustav. Iz rezultata mjerenja zaključujemo da je utjecaj neproduktivne obrade doista značajan i u krajnjim točkama znatno veći nego što bi se moglo kompenzirati s teoretski uspješnijim disciplinama odabiranja događaja u obradu /3/.

3. NEPRODUKTIVNA OBRADA U M/G/1/FCFS SUSTAVU

Sustav s eksponencijalnom razdiobom između pojave ulaznih događaja, općom razdiobom vremena obrade i odabiranjem događaja u obradu prema redoslijedu dolaska, vrlo je detaljno istražen. Glavni parametri sustava dati su Pollaczek-Khinchin izrazima, od kojih se najčešće koristi očekivanje broja događaja u repu čekanja:

$$E(n_q) = \frac{\lambda^2 E(x^2)}{2(1-\rho)} \qquad (3.1)$$

$E(x)$ i (Ex^2) su momenti vremena obrade, a faktor iskorištenja $\rho = \lambda E(x)$. Prema teoremu Littlea, očekivano vrijeme u repu čekanja je:

$$E(w_q) = E(n_q)/\lambda = \frac{\lambda E(x^2)}{2(1-\rho)} \qquad (3.2)$$

Neproduktivnu obradu možemo definirati kao slučajnu veličinu o koja je opisana funkcijom razdiobe V(o) i funkcijom gustoće razdiobe v(o). U sustavima bez rasporedivanja ulaznih događaja u prioritetne grupe, neproduktivnu obradu možemo zamisliti kao dio vremena koje se dodaje vremenu obrade x. Iz ove pretpostavke slijedi da u analizi M/G/1/FCFS sustava s neproduktivnom obradom, ukupno vrijeme obrade x_o shvaćamo kao sumu nezavisnih kontinuiranih slučajnih veličina: $x_o = x + o$. Tada je očekivanje

i varijancija sume:

$$E(x_o) = E(x) + E(o) \qquad \sigma^2_{x_o} = \sigma^2_x + \sigma^2_o \qquad (3.3)$$

Na temelju izraza (3.3) odredimo $E(x_o)$ i $E(x_o^2)$ koje uvrstimo u (3.1) i (3.2) i time su najvažniji parametri za ocjenu aktivnosti M/G/1/FCFS sustava određeni.

Funkcija razdiobe i funkcija gustoće razdiobe složene slučajne veličine x_o date su konvolucijama:

$$B(x_o) = B(x) * V(o) \qquad b(x_o) = b(x) * v(o) \qquad (3.4)$$

gdje su B(x) i b(x) funkcije razdiobe i gustoće razdiobe vremena obrade x.

Neka je b(x) data eksponencijalom $b(x) = u \exp(-ux)$, a gustoća razdiobe neproduktivne obrade također eksponencijalom $v(o) = a \exp(-av)$, gdje su u i a odgovarajući intenziteti $u = 1/E(x)$, $a = 1/E(o)$. Jednostavno se može izračunati da je u tom slučaju gustoća razdiobe veličine x_o jednaka:

$$b(x_o) = \frac{u\,a}{a-u}\left(e^{-ux_o} - e^{-ax_o}\right)$$

Ako je b(x) data eksponencijalom, a v(o) uniformnom razdiobom $v(o) = 1/h$ za $0 \le o \le h$, $v(o) = 0$ za $o > h$, tada je gustoća razdiobe sume:

$$b(x_o) = \frac{1}{h}\left(1 - e^{-ux_o}\right) \qquad 0 \le x_o \le h$$

$$b(x_o) = \frac{1}{h}\, e^{-ux_o}\left(e^{uh} - 1\right) \qquad x_o > h$$

Ako je b(x) data eksponencijalom, a V(o) je deterministička t.j. $v(o) = \delta(o - \sigma)$, gdje je $\delta(o)$ Dirac delta funkcija, gustoća razdiobe sume ovih slučajnih veličina iznosi:

$$b(x_o) = u\, e^{-u(x_o - \sigma)}$$

Pri uniformnoj razdiobi veličina x i o:

$b(x) = 1/a \quad 0 \le x \le a, \qquad b(x) = 0 \quad x > a$

$v(o) = 1/b \quad 0 \le o \le b, \qquad v(x) = 0 \quad o > b$

slijedi:

$$b(x_o) = \frac{1}{ab}\left(x_o - (x_o - a)\, s(x_o - a) - (x_o - b)\right.$$

$$\left. s(x_o - b) + (x_o - a - b)\, s(x_o - a - b)\right)$$

gdje je $s(x - a)$ pomaknuta funkcija jediničnog skoka.

Uz obje determinističke razdiobe $b(x) = \delta(x - a)$, $v(o) = \delta(v - \sigma)$ slijedi trivijalno:

$$b(x_o) = \delta(x_o - a - \sigma)$$

Konačno za uniformnu i determinističku razdiobu slučajnih veličina x i o:

$b(x) = 1/h \quad 0 \le x \le h \qquad b(x) = 0 \quad x > h,$

$v(o) = \delta(o - \sigma)$

slijedi: $b(x_o) = 1/h(s(x_o - \sigma) - s(x_o - h - \sigma))$

Izraz se jednostavno računa uz upotrebu Laplaceove transformacije. $s(x_o - \sigma)$ je pomaknuta funkcija jediničnog skoka.

4. NEPRODUKTIVNA OBRADA U SUSTAVU M/G/1FCFS SA K GRUPA IZVORA ULAZNIH DOGAĐAJA

Razmotrimo računarski sustav s k grupa izvora slučajnih događaja na ulazu ali s disciplinom obrade prema redoslijedu dolaska (FCFS) bez obzira na pripadnost grupi. Pretpostavimo da postoji značajna neproduktivna obrada kao slučajna veličina o, samo kad se uzima u obradu događaj iz grupe različite od grupe prethodnog događaja. Pretpostavimo nadalje jednake intenzitete grupa $\lambda_i = \lambda/k$, $i = 1,2,\ldots,k$, te opće razdiobe vremena obrade $B_i(x) = G$. Svaki događaj traži vrijeme obrade:

$x_m = x$ s vjerojatnošću $1/r$ (pripada istoj grupi)

$x_m = x + o$ s vjerojatnošću $(1 - 1/r)$ (pripada različitoj grupi)

Potrebno je izračunati momente slučajne veličine x_m. Budući da su veličine x i o nezavisne slijedi:

$$E(x_m) = E(x)/r + E(x + o)(1 - 1/r) = E(x) + (1 - 1/r)\,E(o)$$

$$E(x_m^2) = E(x^2)/r + E(x + o)^2(1 - 1/r) = E(x^2) + (1 - 1/r)(2E(x)E(o) + E(o^2))$$

Faktor iskorištenja sustava se povećava zbog dodatnog vremena obrade o i iznosi:

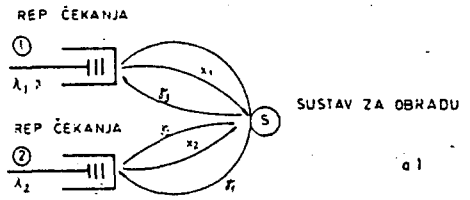
$$\varrho_m = \varrho/r + (1 - 1/r)\,E(x+o) = \varrho + (1 - 1/r)\,E(o)$$

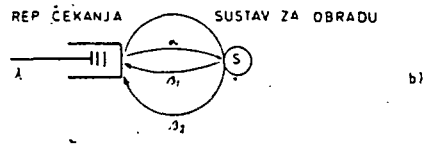
Pri tome je zbog nezavisnosti $E(x + o) = E(x) + E(o)$. Dobiveni izrazi mogu se uvrstiti u Pollaczek-Khinchin izraze (3.1) i (3.2). Razumljivo je da momente $E(x)$, $E(x^2)$, $E(o)$, $E(o^2)$ treba računati iz pretpostavljenih razdioba vremena obrade B(x) i vremena neproduktivne obrade.

5. NEPRODUKTIVNA OBRADA U PRIORITETNIM SUSTAVIMA M/G/1/PRI BEZ MOGUĆNOSTI PREKIDANJA

Razmotrimo stacionaran i ergodičan računarski sustav s k prioritetno raspoređenih grupa ulaznih događaja. Pri dolasku događaja iz prioritetnije grupe obrada događaja koja je u toku se ne prekida, već se prioritetniji događaj uzima u obradu tek po završetku prethodne obrade. U literaturi ne nalazimo razrađen postupak primjenljiv u općem slučaju za sustave s k ulaznih grupa. Analizirat će se jednostavniji primjer s dvije prioritetno raspoređene grupe izvora slučajnih događaja (k = 2) intenzite-

ta λ_1 i λ_2. Sustav je dat na slici 3a.





Slika 3. M/G/1/PRIORIT. sustav bez prekidanja

Neka je vrijeme neproduktivne obrade za reorijentaciju s grupe 1 na grupu 2 dato vremenom γ_1. Nakon reorijentacije, ako je rep čekanja grupe 2 prazan, ponovo nastupa reorijentacija s grupe 2 na 1 tokom vremena $\gamma_2 + \gamma_3$. Ako je rep čekanja grupe 2 neprazan, obrađuje se događaj u vremenu x_2 i ponovo se sustav reorijentira na grupu 1 u vremenu γ_3. Ako je rep čekanja grupe 1 neprazan, obrađuje se događaj u vremenu x_1 i ispita se stanje repa u vremenu γ_3. Pretpostavimo da su slučajne veličine x i γ nezavisne i date razdiobama B(x) i $R_\gamma(t)$ odnosno gustoćama b(x) i $r_\gamma(t)$.

U /4/ je predloženo da se ovakvi sustavi mogu riješiti analizom jednostavnijeg modela s jednim repom čekanja i neproduktivnim dijelom vremena obrade β_1 i β_2 (slika 3b). Zamislimo da sustav za obradu nakon vremena obrade α ispituje jedini rep u čekanju u vremenu β_1. Ako je rep prazan sustav miruje tijekom vremena β_2. Nakon toga, ako je rep neprazan, sustav prolazi ciklus $\alpha + \beta_1$. Definiramo vjerojatnost g_n da se u času ispitivanja u čekanju nalazimo n događaja.

Primjenjujući relaciju za prijelazne vjerojatnosti u M/G/1 sustavu /5/, na model dat na slici 3b, te budući da se rep čekanja ispituje u dva trenutka (nakon mirovanja i nakon obrade) slijedi:

$$g_n = g_0 \int_0^\infty \frac{(\lambda t)^n}{n!} e^{-\lambda t} r_{\beta_2}(t)\,dt + \sum_{i=1}^{n+1} g_i \int_0^\infty \frac{(\lambda t)^{n-i+1}}{(n-i+1)!} e^{-\lambda t} r_{\alpha+\beta_1}(t)\,dt \quad (5.1)$$

gdje je g_0 vjerojatnost da u času ispitivanja nalazimo prazan rep čekanja, a r(t) su gustoće razdioba kontinuiranih slučajnih veličina vremena. Slično, u trenutku završenog vremena obrade broj događaja u sustavu iznosi:

$$n_s = n_s g_0 + \sum_{i=1}^{n+1} g_i \int_0^\infty \frac{(\lambda t)^{n-i+1}}{(n-i+1)!} e^{-\lambda t} r_\alpha(t)\,dt \quad (5.2)$$

Ako načinimo z-transformaciju izraza (5.1) i (5.2) slijedi formula za transformaciju razdiobe vjerojatnosti broja događaja u sustavu datom na slici 3b:

$$Q(z) = \frac{g_0}{1-g_0} \frac{R^*_{\beta_2}(\lambda - \lambda z) - 1}{z - R^*_{\alpha+\beta_1}(\lambda - \lambda z)} R^*_\alpha(\lambda - \lambda z) \quad (5.3)$$

gdje su $R^*(s)$ Laplaceove transformacije gustoća razdioba vjerojatnosti r(t). Koristeći moment generirajuće svojstvo transformacija, iz (5.3) jednostavno se izračunaju očekivanja broja događaja u sustavu kao i vjerojatnost za prazan rep:

$$g_0/(1-g_0) = (1 - \lambda E(\alpha + \beta_1))/\lambda E(\beta_2) \quad (5.4)$$

Primjenjujući analogno razmatranje možemo analizirati model dat na slici 3a. Promatrajući neprioritetni rep uočavamo da je vremenski ciklus obrade jednog događaja iz repa 2 i ispitivanje repa 2, jednak poopćenom periodu zauzetosti repa 1. Taj se period sastoji iz dijelova ispitivanja (γ_1), obrade (x_2), reorijentacije na rep (γ_3), početne obrade događaja koji su stigli u rep 1 za vrijeme $\gamma_1 + x_2 + \gamma_3$, te obrade ostalih događaja u repu 1 od početka njegovog perioda zauzetosti. Prema ideji L. Takacsa i D.P.Gavera /5/, a zbog nezavisnosti vremenskih razmaka, slijedi za poopćene periode zauzetosti:

$$Y^*_{C21}(s) = R^*_{\gamma_1}(s + \lambda_1 - \lambda_1 Y^*(s))\, B^*_{x_2}(s + \lambda_1 - \lambda_1 Y^*(s)) \cdot R^*_{\gamma_3}(s + \lambda_1 - \lambda_1 Y^*(s)) \quad (5.5)$$

Pri tome je $Y^*_{C21}(s)$ Laplaceova transformacija gustoće razdiobe vremena ciklusa obrade događaja iz druge grupe, $B^*_{x_2}(s)$ je Laplaceova transformacija gustoće razdiobe vremena obrade događaja iz druge grupe, a $Y^*(s)$ je transformacija gustoće razdiobe standardnih perioda zauzetosti y koji, budući da se sastoje iz nezavisnih dijelova x_1 i γ_3 zadovoljavaju relaciju:

$$Y^*(s) = B^*_{x_1}(s + \lambda_1 - \lambda_1 Y^*(s))\, R^*_{\gamma_3}(s + \lambda_1 - \lambda_1 Y^*(s))$$

Na identičan način možemo razmotriti vremenski ciklus u slučaju praznog repa 2 i pisati za $Y^*_{C22}(s)$ izraz kao (5.5) uz supstituciju γ_2 umjesto x_2. Usporedbom modela sa slike 3a s modelom 3b, opažamo da u izrazu (5.3) umjesto $R^*_{\beta_2}$ možemo staviti Y^*_{C22}, umjesto $R^*_{\alpha+\beta_1}$ stavlja-

mo Y^*_{C21}, a umjesto R^*_{α} stavljamo $B^*_{x_2}$. Iz tako dobivene relacije možemo odrediti momente t.j. očekivani broj događaja u repu 2.

Promotrimo sada prioritetnu grupu događaja u modelu na slici 3a. Ciklus obrade sastoji se iz vremenskih razmaka $x_1 + \gamma_3$. Prema tome relacija analogna (5.5) glasi:

$$Y_{C11} = B^*_{x_1}(s) \cdot R^*_{\gamma_3}(s)$$

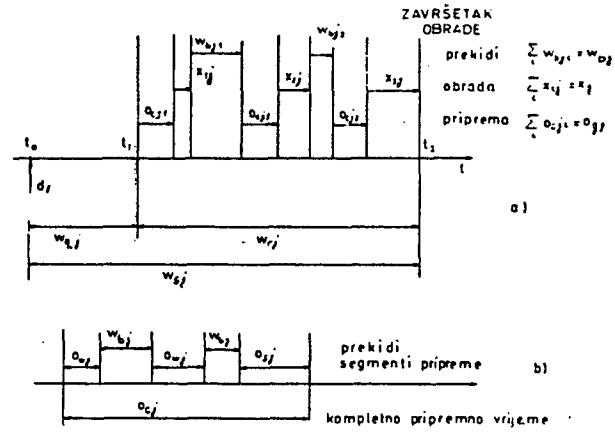
Ciklus praznog hoda nešto je kompliciraniji. Ako je rep 1 prazan vremenski razmak sastoji se iz nezavisnih vremenskih sekvenci γ_1, x_2 s vjerojatnošću $(1 - g_{20})$, γ_2 s vjerojatnošću g_{20}, te γ_3. Prema tome:

$$Y^*_{C12}(s) = R^*_{\gamma_1}(s)((1 - g_{20})B^*_{x_2}(s) + g_{20}R^*_{\gamma_2}(s)) \cdot R^*_{\gamma_3}(s) \qquad (5.6)$$

Usporedbom modela 3a i 3b opažamo da u relaciji (5.3) umjesto $R^*_{\beta_2}$ stavljamo Y^*_{C12}, umjesto $R^*_{\alpha+\beta_1}$ stavljamo Y^*_{C11}, a umjesto R^*_{α} stavljamo $B^*_{x_1}$. Iz dobivenog izraza, koristeći moment generirajuće svojstvo izračunamo očekivanje broja događaja u prioritetnom repu 1. Vjerojatnost g_{20} (prazan rep 2) izračunamo iz (5.4) tako da umjesto $E(\alpha + \beta_1)$ stavimo očekivanje perioda zauzetosti $E(y_{22})$ dobivenog moment generirajućim postupkom iz Y^*_{C22}, a umjesto $E(\beta_2)$ stavimo očekivanje $E(y_{21})$ dobiveno iz Y^*_{C21}. Intenzitet $\lambda = \lambda_2$. Prema /5/, transformacija razdiobe vjerojatnosti broja događaja $Q(z)$ i Laplaceova transformacija gustoće razdiobe vjerojatnosti vremena u M/G/1 sustavu $W^*_{w_s}(s)$, vezani su jednostavno $Q(z) = W^*_{w_s}(\lambda - \lambda z)$, te su time relacijom (5.3) određeni i vremenski parametri sustava.

6. NEPRODUKTIVNA OBRADA U PRIORITETNIM SUSTAVIMA M/G/1/PRI S PREKIDANJEM

Razmotrit će se model računarskog sustava s k-grupa izvora slučajnih događaja uz mogućnost prekida obrade i nastavka u točki prekida. Unutar pojedine prioritetne grupe sustav odabire događaje u obradu prema redoslijedu dolaska (FCFS). Prema slici 4a, u času t_o događaj d_j dolazi u sustav i priključuje se repu čekanja. U času t_1 događaj d_j prvi puta ulazi u obradu. Vrijeme čekanja u repu iznosi $w_{qj} = t_1 - t_o$. Prije svakog početka obrade, potrebno je kompletno pripremno neproduktivno vrijeme, koje za događaj d_j iznosi o_{cj}. Iza kompletnog pripremnog neproduktivnog vremena o_{cj1}, slijedi obrada (dio od x_j), pa mogući prekid, tokom kojeg je obrada blokirana kroz vremenski razmak w_{bj1}, zatim opet kompletno pripremno neproduktivno vrijeme o_{cj2}, te se tako ciklus ponavlja do trenutka t_2 kad



Slika 4. M/G/1/PRIORIT. sustav s prekidanjem

događaj konačno napušta obradu. Kompletno pripremno neproduktivno vrijeme o_{cji}, sastoji se prema slici 4b, od segmenta izgubljenog neproduktivnog vremena o_{wj}, segmenta blokiranog vremena w_{bj} i konačno segmenta uspješnog pripremnog vremena o_j, budući da pretpostavljamo mogućnost prekida i tokom neproduktivnog vremena pripreme obrade. Slučajna veličina o_j data je za pojedinu grupu izvora slučajnih događaja razdiobom $V_j(o)$, gustoćom razdiobe $v_j(o)$ i pridruženom transformacijom $V^*_{oj}(s)$.

Slučajni vremenski razmak od trenutka prvog ulaska događaja d_j u obradu do trenutka potpunog završetka obrade (rezidentno vrijeme) $w_{rj} = t_2 - t_1$ zauzima ključnu ulogu u analizi prioritetnih sustava s prekidanjem događaja u obradi. U sustavima bez prekidanja, evidentno je $w_{rj} = x_j$.

Opisani model sustava s neproduktivnom programskom podrškom može se dekomponirati na prekidne sustave bez neproduktivne programske podrške s nastavkom obrade u točki prekida (M/G/1/PRI-PRN) te na sustave s nastavkom obrade s početnim vremenom obrade (M/G/1/PRI-PRP).

Za sustav bez neproduktivne obrade i nastavkom obrade u točki prekida, rezidentno vrijeme događaja d_j sastoji se iz segmenta obrade x_{ij} te vremena blokade w_{bj}, odnosno iz totalnog vremena obrade (x_j) i segmenata vremena blokade w_{bj}. U sustavu s nastavkom u točki prekida, ukupno vrijeme obrade jednako je zahtjevanom vremenu obrade j-te grupe, tj. $x_{gj} = x_j$, pa su transformacije pridruženih gustoća razdioba identične.

U sustavu bez neproduktivne obrade i nastavkom obrade u točki prekida s identičnim, početnim, vremenom obrade izabranim iz $B_j(x)$, ukupno vrijeme obrade sadrži nekoliko izgubljenih (prekinutih) segmenata obrade x_{wj} i jedno

uspješno vrijeme obrade x_j. Rezidentno vrijeme u takvom sustavu sastoji se iz N zavisnih parova $x_{wj} + x_{bj}$ i jednog vremenskog razmaka uspješne obrade x_j. U literaturi /6/ nalazimo transformaciju gustoće razdiobe rezidentnog vremena $Y^*_{w_{rj}}(s)$.

Budući da u ovom slučaju ukupno vrijeme obrade x_{gj} nije jednako zahtjevanom vremenu obrade x_j, to se transformacija pridružena veličini x_{gj} dobije iz $Y^*_{w_{rj}}(s)$ uz supstituciju $Y^*_{w_{bj}}(s) = 1$; tj. izostavljanjem segmenata vremena blokiranja: $Y^*_{w_{bj}}(s)$ je transformacija gustoće razdiobe vremena blokiranja.

U literaturi /7/ također nalazimo Laplaceovu transformaciju razdiobe vremena u prekidnim sustavima $W^*_{w_{sj}}(s)$ izraženu preko rezidentnog vremena, i vremena blokiranja.

Razumljivo je da relacija za $W^*_{w_{sj}}(s)$ vrijedi za sve prekidne sustave (PRN, PRP) jer se razlučivanje odvija baš u rezidentnom vremenu odnosno pridruženoj transformaciji $Y^*_{w_{rj}}(s)$. Iz tih izraza koristeći svojstvo generiranja momenata izvedeno je očekivanje vremena u repu čekanja grupe j:

$$E(w_{qj}) = \frac{\lambda_j E(w_{rj}^2)}{2(1-\lambda_j E(w_{rj}))} + \frac{\lambda_a E(w_{bj}^2)}{2(1+\lambda_a E(w_{bj}))} = E(w_{sj}) - E(w_{rj}) \tag{6.1}$$

Relacija (6.1) daje nam najvažnije parametre u sustavu, ali izražene preko slučajnih veličina w_{rj} i w_{bj}. λ_a je intenzitet prekidanja t.j. $\lambda_a = \lambda_1 + \lambda_2 + \ldots + \lambda_{j-1}$.

U trenutku prekida (početak segmenta w_{bj}) obrade događaja d_j, u sustavu se nalazi samo jedan događaj višeg prioriteta (s indeksom $<j$). Promatrani segment w_{bj} sastoji se iz rezidentnog vremena prioritetnog događaja koji je uvjetovao prekid, te iz rezidentnih vremena događaja iz grupa j-1, j-2, ..., 1, koji su stigli u međuvremenu. Tako vrijeme blokiranja i rezidentno vrijeme ovise jedno o drugom rekurzivno (rezidentno vrijeme 1. grupe određuje period blokiranja 2. grupe itd.). Vrijeme blokiranja možemo izraziti kroz poopćene cikluse zauzetosti:

$$Y^*_{w_c}(s) = Y^*_{w_{bo}}(s + \lambda - \lambda Y^*_{w_b}(s)) \tag{6.2}$$

gdje je w_c ciklus blokiranja u kojem je prvi dio jednak vremenskom razmaku w_{bo}, a ostali dijelovi su normalni periodi blokiranja. Iz (6.2) slijede momenti:

$$E(w_c) = \frac{E(w_{bo})}{1-\lambda E(w_b)} \tag{6.3}$$

$$E(w_c^2) = \frac{E(w_{bo}^2)}{(1-\lambda E(w_b))^3} E(w_{bo}) + \frac{E(w_{bo}^2)}{(1-\lambda E(w_{bo})^2} \tag{6.4}$$

Period blokiranja događaja iz grupe j + 1 jednak je ciklusu zauzetosti događaja iz grupe j. Zbog dolazaka događaja iz grupe j, s vjerojatnošću $\lambda_j/(\lambda_a + \lambda_j)$, period blokiranja $w_{b,j+1}$ jednak je ciklusu zauzetosti događaja grupe j s početnim vremenom w_{rj} i normalnim periodima također w_{rj}. Zbog dolazaka događaja iz grupa 1 do j-1, s vjerojatnošću $\lambda_a/(\lambda_a + \lambda_j)$, period blokiranja $w_{b,j+1}$ jednak je ciklusu zauzetosti događaja grupe j s početnim vremenom w_{bj} i normalnim periodima w_{rj}. Koristeći (6.3) i (6.4) možemo pisati:

$$E(w_{bj+1}) = \frac{\lambda_j}{(\lambda_a+\lambda_j)} \frac{E(w_{rj})}{(1-\lambda_j E(w_{rj}))} + \frac{\lambda_a}{(\lambda_a+\lambda_j)} \frac{E(w_{bj})}{(1-\lambda_j E(w_{rj}))} \tag{6.5}$$

$$E(w_{b,j+1}^2) = \frac{\lambda_a E(w_{bj}^2)(1-\lambda_j E(w_{rj})) + \lambda_j E(w_{rj}^2)(1-\lambda_a E(w_{bj}))}{(\lambda_a+\lambda_j)(1-\lambda_j E(w_{rj}))^3}$$

Za grupu događaja j + 1 vrijedi (6.1) direktno. Uz supstituciju (6.5) i (6.6) slijedi rekurzivna formula:

$$E(w_{q,j+1}) = \frac{\lambda_{j+1} E(w_{r,j+1}^2)}{2(1-\lambda_{j+1} E(w_{r,j+1}))} + \frac{E(w_{qj})}{1-\lambda_j E(w_{qj})} \tag{6.7}$$

Analizirajući sliku 4a, uočavamo da se nakon prvog segmenta o_{cj1} promatrani sustav ponaša kao M/G/1PRI-PRN s intervalima blokiranja $w_{bji} + o_{cji}$, te zbog nezavisnosti slijedi za rezidentno vrijeme:

$$Y^*_{w_{rj}}(s) = Y^*_{o_{cj}}(s) \cdot B^*_j(s + \lambda_a - \lambda_a Y^*_{w_{bj}}(s)\, Y^*_{w_{cj}}(s))$$

Iz gornjeg izraza iskoristimo moment generirajuće svojstvo i izračunamo momente:

$$E(w_{rj}) = f_1(E(o_{cj}), E(w_{bj}), E(x_j)) \tag{6.8}$$

$$E(w_{rj}^2) = f_2(E(o_{cj}), E(o_{cj}^2), E(w_{bj}), E(w_{bj}^2), E(x_j), E(x_j^2)) \tag{6.9}$$

Prema slici 4b, kompletno neproduktivno vrijeme o_{cj} analogno je rezidentnom vremenu u sustavu s prekidima i nastavkom obrade s početnim vremenom. U /6/ nalazimo izraz za transformaciju rezidentnog vremena, koja s novim oznakama glasi:

$$Y^*_{o_{cj}}(s) = \int_0^\infty \frac{(s+\lambda_a) e^{-(s+\lambda_a)o}}{s + \lambda_a - \lambda_a Y^*_{w_{bj}}(s)(1-e^{-(s+\lambda_a)o})} v_j(o)\, do \tag{6.10}$$

Iz (6.10) izračunaju se momenti:

$$E(o_{cj}) = f_3(E(w_{bj}), E(o_{gj})) \tag{6.11}$$

$$E(o_{cj}^2) = f_4(E(w_{bj}), E(w_{bj}^2), E(o_{gj}), E(o_{gj}^2)) \tag{6.12}$$

Stavljajući u izraz (6.10) $Y^*_{w_{bj}}(s) = 1$, slijedi Laplaceova transformacija pridružena ukupnom vremenu neproduktivne obrade iz koje se mogu

izračunati momenti $E(o_{gj})$ i $E(o^2_{gj})$. Time su dati svi izrazi potrebni u iterativnom postupku odredivanja očekivanja vremena u sustavu i repu čekanja prema izrazu (6.1).

Iteracija započinje s prvom prioritetnom grupom ($j = 1$) za koju $\lambda_a = 0$, $w_{bj} = 0$, $w_{rj} = x_j$, te se $E(w_{qj})$ i $E(w_{sj})$ odreduju iz (6.1) direktno. Za ostale grupe, npr. $j+1$, za dati $E(w_{bj})$, $E(w^2_{bj})$, $E(w_{rj})$, $E(w^2_{rj})$, iz (6.5) i (6.6) slijedi $E(w_{b,j+1})$, $E(w^2_{b,j+1})$. Supstitucijom u (6.11) i (6.12) slijedi $E(o_{c,j+1})$ i $E(o^2_{c,j+1})$, jer $E(o_{g,j+1})$, $E(o^2_{g,j+1})$ slijedi nezavisno iz modificiranog izraza (6.10) stavljanjem $Y^*_{w_{bj}}(s) = 1$. Supstitucijom u (6.8) i (6.9) slijedi $E(w_{r,j+1})$, $E(w^2_{r,j+1})$, te konačno iz (6.1) slijede očekivanje vremena u sustavu ili repu čekanja grupe $j+1$.

7. ZAKLJUČAK

U ovom radu pokazano je i dokazano mjerenjem da često usvojena pretpostavka o konzervativnosti sustava nije ispravna. Pri visokim ulaznim intenzitetima događaja, neproduktivni dio vremena u sustavu ima dominantan utjecaj na odzivno vrijeme. Ako je povećanje vremena obrade jednako za sve događaje, bez obzira na pripadnost prioritetnoj grupi, razdioba ukupnog vremena obrade može se izračunati konvolucijskim teoremom. Ako je neproduktivni vremenski razmak vezan uz promjenu grupe pri obradi FCFS disciplinom, momenti složene slučajne veličine obrade mogu se odrediti uz poznavanje vjerojatnosti pripadnosti događaja istoj grupi. Ta je vjerojatnost proporcionalna udjelu grupe u ukupnom ulaznom intenzitetu. Proračun osnovnih stohastičkih parametara u prioritetnim sustavima bez prekidanja i s prekidanjem pokazao je da analitički pristup može dati općenito upotrbljive rezultate za relativno jednostavne modele. Nadene su transformacije razdioba vjerojatnosti nekih stohastičkih parametara i pokazano je kako se do momenata traženih slučajnih veličina može doći direktno ili iteracijskim postupkom.

REFERENCIJE:

/1/ L.Kleinrock: Queueing Systems, Vol.I: Theory, John Wiley & Sons, 1975.

/2/ N.Bogunović: Response Time Measurement of Real-Time Computer Systems with Priority Structures, 10th IMEKO World Congress Proceedings, Prag, ČSSR, 22-26.04.1985.

/3/ L.Schrage: Optimal Scheduling Discipline, Working paper, Graduate School of Business, Univ. of Chicago, 1974.

/4/ C.E.Skinner: A priority Queueing System with Server Walking Time, Operations Research, 14(1966) str. 279-285.

/5/ L.Kleinrock: Queueing Systems, Vol. II, John Wiley, 1975, 1976.

/6/ B.Avi Itzak: Preemptive Repeat Priority Queues as a Special Case of the Multipurpose Server Problem, Operations Research, 11 (1963) No.4, str 303-320.

/7/ R.W.Conway, W.L.Maxwell, L.W.Miller: Theory of Scheduling, Addison - Wesley, 1967.

MODEL DEDUKTIVNE BAZE PODATAKA IMPLEMENTIRAN U PROLOGU

MARIO RADOVAN, SVEUČILIŠTE RIJEKA, SET-PULA

UDK : 681.3.01

U članku je dat je prikaz (prijedlog) modela deduktivne baze podataka, implementiranog u Prologu. Dat je prijedlog načina kontrole (i tretmana) redundance, uslova integriteta i generirane redundance. Opisane su i primjerima ilustrirane osnovne (do sada implementirane) instrukcije za ažuriranje i komunikaciju sa bazom podataka. U predloženom modelu naglasak je dat na razvoj kooperativnog komunikacijskog sistema, sposobnog da korisnika čim potpunije obavještava o stanju sistema, kao i (mogućim) konsekvencama pojedinih akcija na sistemu.

A MODEL OF DEDUCTIVE DATABASE IMPLEMENTED IN PROLOG
The article presents a model of a deductive database, implemented in Prolog, which provides mechanisms for controlling redundancy and checking integrity constraints and generated redundancies. The query and update instructions (implemented so far) are described and illustrated by examples. The proposed model develops a cooperative communication system, capable of informing the user about the state of the system and the (possible) consequences of his actions.

1. UVOD

Kod relacijskog modela baze podataka, informacije (podatke) predstavljamo n-torkama, koje u terminima logike prvoga reda možemo smatrati temeljnim atomarnim formulama. Pojam deduktivne baze podataka odnosi se na proširenje relacijskog modela, dobiveno prihvaćanjem ne samo atomarnih formula (n-torki), već i (zatvorenih) neatomarnih formula slijedećeg oblika:

A1 * A2 * ... * An --> B;

gdje su A1, A2, ..., An i B atomarne formule, a '*' znak konjunkcije ili disjunkcije. Ovakve formule nazivamo definitnim (definite) klauzulama, a bazu podataka u kojoj nastupaju, definitnom deduktivnom bazom podataka. Bazu podataka pritom dijelimo na ekstenzionalni dio, u koji spadaju temeljne atomarne formule (tj. one kod kojih je n = 0, a konsekvens ne sadrži varijable), i intenzionalni dio, koji čine formule kod kojih je n > 0. Skup svih deducibilnih temeljnih atomarnih formula (dakle n-torki, u relacijskoj terminologiji), koje su ili eksplicitno prisutne u ekstenziji baze ili pak deducibilne pomoću pravila (formula iz intenzije), zvati ćemo ukupnom ekstenzijom baze podataka. Teoretska zasnova relacijskog modela baze podataka u terminima logike, data je npr. u /Gallaire,78/ i /Reiter,84/, a mogućnosti ekstenzije na deduktivni (logički) sistem razmatrane su u /Gallaire,83/, /Lloyd,83/ i /Gallaire,84/.

Programski jezik Prolog je za implementaciju deduktivne baze podataka posebno pogodan,jer su instrukcije jezika Prolog zapravo definitne klauzule, sa dodatkom kontrolnih funkcija (npr. "!"), sistemskih funkcija ("read", ...), i "Negacije kao neuspijeha". Stoga su informacije u deduktivnoj bazi podataka (date u obliku definitnih klauzula), ujedno instrukcije jezika Prolog.

Promatrajući deduktivnu bazu podataka kao teoriju (tj. kao konsistentan skup zatvorenih formula - rečenica), davanje odgovora na upit postavljen bazi, svodi se na njegovu logičku dedukciju iz informacija prisutnih u bazi. Prolog interpretor to čini na taj način, da negiranu formulu (implicitnu tvrnju) iz upita doda postojećim formulama u bazi, i pokuša (primjenom linearne rezolucije), dokazati da je teorija (tj. baza), tim dodavanjem postala nekonsistentna. Ukoliko u tome uspije, onda je implicitna tvrdnja iz upita (odnosno neka njena instanca), zaista deducibilna iz date baze podataka, te odgovor na upit (u kojem ne nastupaju varijable), glasi "DA", odnosno odgovor na upit (koji sadrži varijable), su one instance varijabli iz upita za koje teorija postaje nekonsistentnom. (Relacijskim terminima rečeno, te bi instance bile zapravo upitom tražene vrijednosti atributa.)

Temeljni problemi vezani za prološku implementaciju baze podataka razmatrani su u /Kowalski,79/, /Bowen,81/ i /Lloyd,82/. Relacijski sistemi implementirani u Prologu dati su u /Pereira,82/ i /Li,84/. Pritom Li Prolog upotrebljava prije svega kao jezik za pisanje interfejsa, kojim se omogućava, da se na istom DBM sistemu koristi više različitih relacijskih jezika. Mogućnosti poboljšanja kooperativnosti (/Kaplan,82/), kod relacijskog jezika QBE date su u /Neves,83/, a analiza prološke implementacije uslova integriteta (integrity constraints) kod relacijskog modela data je u /Williams,83/. Sistem za asimilaciju znanja (knowledge assimilation), kao osnova za e-

kspertne sisteme odnosno baze znanja (knowledge bases) potpuno implementiran u Prologu, dat je u /Miyachi,84/, ali baza nije deduktivna u ovdje definiranom smislu jer ne sadrži pravila (formule), već ostaje relacijskom. Rad na razvoju sistema, koji sadrži i pravila opisan je u /Kitakami,84/, stime što je u tom sistemu naglasak dat na razvoj mogućnosti induktivnog zaključivanja.

U modelu koji ovdje razvijamo naglasak je dat na kooperativnost sistema, pod čime podrazumijevamo sposobnost sistema, da sa korisnikom kkmunicira u jeziku za korisnika čim prikladnijem, te da daje obrazloženja uspješnih (i neuspješnih) pokušaja dedukcije tražene informacije, kao i upozorenja korisniku o mogućim jonsekvencama pojedinih akcija u sistemu. Zrikazane su osnovne konture sistema, naglašeni neki specifišni problemi, koje uvođenje pravila u bazu podataka donosi, te ilustriran rad dosad implementiranih funkcija.

2. OPIS MODELA

U ovom odjeljku dat je kratak opis modela i osnovnih (do sada implementiranih) instrukcija. Prološka implementacija deduktivne baze pogodna je za razvoj jezika sa širokim mogućnostima postavljanja upita (i komuniciranja uopće /Radovan,85/). Primjena datih instrukcija ilustrirana je primjerima u odjeljku (3).

Osnovne instrukcije sistema

`lfile(Ime_dat).`

Izlistava (doslovni) sadržaj datoteke "Ime_dat". Pravila pritom ostaju u izvornom obliku, a ukupna ekstenzija baze ne daje se eksplicitno.

`lext(Ime_dat).`

Izlistava ukupnu ekstenziju datoteke "Ime_dat". Pravila iz intenzije baze bivaju pritom "prevedena" u pripadne ekstenzije, pomoću njih deducibilne.

```
insert.
*: (Informacija).
```

Instrukcija "insert." je temeljna instrukcija sistema, te ju podrobnije opisujemo. Informacija, koja se unosi, može biti atomarna formula ("n-torka", odnosno temeljna instanca sheme relacije), ili pak pravilo (zatvorena formula), tipa

```
Consec :- Ant1 * Ant2 * ...* AntN.
```

gdje su Ant1, Ant2, ..., AntN literali (tj. atomarne formule ili negirane atomarne formule), a Consec atomarna formula. Dakle, pravila, kao elementi baze podataka (i ujedno Prolog instrukcije (klauzule)), čine širu klasu formula od klase definitnih klauzula (definiranih u odjeljku (1)), upravo dopuštanjem i negiranih atomarnih formula u pravilu. Pritom važi (meta)princip, da je cilj "not(A)" zadovoljen (istinit), onda kada cilj "A" nije zadovoljen (istinit) u datoj bazi podataka. Primjena negacije ograničena je na temeljne instance atoma (i formula, uopće), a to znači da u trenutku pozivanja cilja "not(A)", moraju (eventualne) varijable iz "A" biti već instancirane od strane cilju "not(A)" prethodećih ciljeva. Ovaj zahtjev izgleda (barem u operativnom smislu), ekvivalentnim Ullmanovom. zahtjevu po sigurnim (safe) formulama.

Kod unosa, najprije se provjerava redundantnost nove informacije. Informaciju smatramo redunrantnom, ukoliko je u trenutku unošenja već deducibilna (logički izvediva (slijedi)), iz baze podataka. U sistemu se to provjerava tako, da se u slućaju unošenja činjenica (n-torki), iste pokušaju najprije deducirati iz baze. U slučaju unošenja pravila (formula), pokuša se pokazati da ekstenzija antecedensa (deducibilna iz baze), nije veća od ekstenzije konsekvensa (deducibilne iz baze). Drugim riječima, pokušavamo pokazati, da iz baze nije moguće deducirati takvu instancu antecedensa, za koju nebi istodobno bila (već) deducibilna i odgovarajuća instanca konsekvensa. Ukoliko u tome uspijemo, to onda znači da je informacija koja se unosi redundantna (tj. da ne puvećava ukupne ekstenzije!). Sistem na to upozorava, te se prema zahtjevu korisnika, informaciju unosi ili ne unosi u sistem.
U slijedećem koraku provjerava se da li unošenje nove informacije dovodi do kršenja uslova integriteta baze podatka. Ograničenja, koja se uslovima integriteta baze postavljju, mogu se podijeliti na tri osnovne grupe:

- *ograničenje dopuštene vrijednosti* (tj. domene), za pojedine atribute u shemi relacije,
- strukturna ograničenja, u koja spadaju funkcijske ovisnosti,
- ograničenja na načine i pravo korištenja baze podataka.

Od navedenih, u modelu je razmatrana samo problematika kontrole i očuvanja funkcijske ovisnosti. Ujedno je pokazano koje sve probleme prisustvo pravila u bazi postavlja pred kontrolu integriteta baze. Uslovi integriteta predstavljeni su ovdje u "negativnom obliku", tj. ciljem, koji u bazi ne smije biti zadovoljen (deducibilan).

Ukoliko bi pak unošenjem nove informacije taj cilj postao zadovoljiv (deducibilan), onda se nova informacija ne prihvaća a korisnok obavještava.

Prihvaćena informacija unosi se u bazu, a zatim se vrši kontrola generirane redundantnosti. To se izvodi na taj način, da se svaku eksplicitno prisutnu informaciju iz baze redom (privremeno) odstrani, pa zatim istu pokuša deducirati. Ukoliko pokušaj dedukcije uspije, daje se obavijest o (stvorenoj) redundantnosti, i informacija zatim briše ili zadržava, prema zahtjevu korisnika. Kontrolom generirane redundantnosti, proces unosa (prihvaćene i (u odnosu na IC), valjane), informacije završava.

Navodimo slijedeće (potencijalne) razloge za zadržavanje redundantnih informacija u sistemu :

- eksplicitno prisustvo činjenice može ubrzati proces dedukcije,
- neke informacije, koje su u datom trenutku redundantne, mogu kasnijim ažuriranjima (bilo unosom bilo brisanjem drugih informacija), to prestati biti, te bi njihovo obavezno isključvanje (ili nedopuštanje unosa), nepotrebno

otežavalo posao ažuriranja baze,

- dopuštanje prisutnosti redundance može znatno olakšati rad u fazi razvijanja baze znanja i to upravo iz rzloga navedenih u gornjoj točci.

Valja međutim napomenuti, da prisustvo redundantnih informacija u bazi postavlja dodatne teškoće (ili bar zahtjeve) pred valjanu implementaciju funkcija agregacije, pout "suma", "prosjek" i sličnih.

```
delete.
*: (Informacija).
```

Naredbom "delete" brišemo informaciju datu kao argument naredbe. Informacija može biti činjenica ili pravilo; ukoliko se informacija ne nalazi u bazi podataka, ne poduzima se ništa, a korisnik o tome obavještava.

```
expl.
*: (Cilj).
```

Ovom instrukcijom tražimo od sistema odgovor, da li je (potencijalno složen) cilj "Cilj" deducibilan iz sistema, te ako jeste, tražimo objašnjenja, koji su (sve) mogući "putovi" njegova deduciranja. Program, kojim je navedena instrukcija implementirana, navodimo u cjelini, jer nam dato programsko rješenje izgleda jednostavnijim od programa za analogne instrukcije (predikate), kao što su "demo " i "deduce" (vidi npr. /Kowalski,79/, /Bowen,81/, /Kitakami,84/, /Miyachi,84/).

```
/* izvođenje i obrazlaganje dedukcije */
expl :- read(X), decomp(X,A),
      prove(A,L), nl, show(L), nl,
      write('Another explanation ?(y./n.)'),
            read(Ans), nl, next_one(Ans).
expl :- write('No (more) deduction(s)'),nl, !.

prove((A,B),L) :- prove(A, L1), prove(B, L2),
                  append(L1,L2,L).
prove((A;B),L) :-  prove(A,L); prove(B,L).
prove(not((A)), _) :- prove((A), _),!, fail.
prove(not(A),[not(A)]).
prove(A,[A from B|C]) :- clause(A,B),
                         prove(B,C).
prove(A, [is_true(A)]) :-
          functor(A, F, N),
          not(member(F,['=', not,';',true])),
          prolog_system_predicate(F, N),
          call(A).
prove(true,[]).
```

Slijedeće tri instrukcije omogućavaju nam da prije insertiranja neke informacije u sistem (tj. u bazu),provjerimo neka njezina svojstva i (buduće) učinke na bazu podataka.

```
check.
*: (Antec implies Consec).
```

Upitom tóga tipa tražimo od sistema da provjeri, vrijedi li (već) u bazi podataka promatrano pravilo tipa

"Antecedens implicira Konsekvens".

Ukoliko ne vrijedi, sistem navodi one instance antecedensa deducibilne iz baze, za koje iz baze nisu deducibilne odgovarajuće instance konsekvensa.

```
extinf.
*: (Informacija).
```

Ovom instrukcijom tražimo "ekstenziju informacije", tj. koje su sve instance sheme relacije deducibilne iz antecedensa razmatranog pravila. Uloliko je riječ o činjenici, onda je njena ekstenzija samo ona sama.

```
newext.
*: (Informacija).
```

Instrukcija slična gornjoj, stime da se njome provjerava koje su instance deducibilne samo iz "Informacije" (pravila), koje razmatramo, a bez primjene tog pravila ne bi uopće bile deducibilne. Drugim riječima tražimo koliko "proširenje" ukupne ekstenzije donosi primjena toga pravila. Ukoliko se radi o činjenici (a ne pravilu), onda je "newext" činjenice jednaka njoj samoj, ako ista nije već deducibilna iz baze, odnosno praznom skupu, ako jeste deducibilna.

```
all.
*: ((N-torke) such_that Uvjeti).
```

Ovaj tip upita analogan je upitima (SELECT-FROM-WHERE) iz relacijskog jezika SQL. Odgovor na upit je lista svih N-torki, koje zadovoljavaju Uvjete. Pritom u Uvjetima mogu nastupati logički operatori kao i funkcije agregacije. Činjenica, da je neka iformacija (instanca sheme u bazi), ponekad deducibilna na dva ili više načina (što možemo provjeriti pomoću naredbe "expl."), jeste razlogom da uobičajene prološke funkcije "bag_of" i "set_of" nisu direktno upotrebljive za implementaciju fukcija (instrukcija) tipa "all.", kojima zahtjevamo od sistema sve one "n-torke", za koje su ispunjeni uslovi iz upita. "Bag_of" funkcija bila bi neprikladna jer bi svaku informaciju (element ukupne extenzije baze), uzela u obzir (prilikom sumiranja i slično), onoliko puta na koliko je načina ta informacija deducibilna u sistemu! - što, naravno, u slučajevima gdje postoji redundanca u sistemu, ne bi davalo ispravne rezultate. S druge strane, primjena funkcije tipa "set_of", kojom se višestruko deducibilna informacij tretira samo jedanput, ne bi radila u slučaju kada je potrebno izvršiti npr. zbrajanje vrijednosti nekog atributa na skupu n_torki iz datoteke. Naime, u tom slučaju, svaka različita vrijednost atributa bila bi uzeta u obzir samo jednom, što, naravno, nije ispravno jer više različitih n-torki (instanci) može imati jednaku vrijednost promatranog atributa, a pribrojiti treba ipak sve (tj. svaku!). Taj smo problem ovdje riješili modifikacijom funkcije "set_of" - konkretno, procedurom "all _u(T,G,L)", koja ujedno čini osnov za implementaciju svih funkcija tipa "SELECT-FROM-WHERE". Suština modifikacije astoji se u tome, da se u klasičnoj "trojci" (N_torka, Cilj, Lista), Listu formira tako, da se najprije generiraju jedinstveni (tj. bez ponavljanja!), parovi (N_torka, Cilj), a zatim sve N_torke iz parova (među kojima može biti i više jednakih N_torki!), "pokupe" u Listu. Taj postupak (program slijedi), uspješno rješava oba gore navedena problema u vezi sa dupliciranjem.

```
/* instr. "all" i podr. fun. */
all :- read(X such_that Cond),
       decomp(Cond,Dcond),
       all_u(X,Decond,L), show(L), !.

all_u(T,G,_) :- find_tuples_u(T,G).
```

```
all_u(T,G,L) :- collect_them_u([],L),!.

find_tuples_u(T,G) :-
                    asserta(found(mark,mark)),
                    call(G),
                    ass(T,G), fail.

ass(T,G) :- found(T,G),!.
ass(T,G) :- asserta(found(T,G)),!.
collect_them_u(L,List) :-
                   get_next(X,Y),
                   collect_them_u([X|L],List).
collect_them_u(L,L).

get_next(X,Y) :- retract(found(X,Y)), !,
                 X \== mark.
```

3. OPIS BAZE I ILUSTRACIJA RADA

Rad sistema ilustrirajmo na sljedećem modelu (i početnom stanju) baze podataka.

```
 descript(pp, pp(_1, _2, _3),
   ic((pp(_1,_2,_3), pp(_1,_2,_4), _4 \==_3))).

 pp(a, d, 4).
 pp(a, c, 4).
 pp(b, c, 3).
 pp(X,Y,Z) :- qq(_1, _2, _3), not(rr(_1, _2)).

 descript(qq, qq(_1, _2, _3), no).

 qq(a, b, 6).
 qq(c, d, 8).
 qq(u, v, 8).
 qq(u, s, 5).
 qq(a, d, 4).

 descript(rr, rr(_1, _2), no).

 rr(b, a).
 rr(c, d).
 rr(a, d).
```

Baza se sastoji od tri datoteke, od kojih jedna sadrži pravilo. Pojedina datoteka opisana je klauzulom

```
descript(Ime_datoteke, Shema_relacije
                       ic(Uslovi_integriteta)).
```

Ovdje smo uslove integriteta dali samo za datoteku "pp" jer je za ilustraciju rada (i specifičnih problema kod deduktivne baze), to dovoljno. "(Argument)" u ic((Argument)) jeste konjuncija ciljeva, koja ne smije biti zadovoljiva u (odnosno, deducibilna iz) datoteci "pp", da bi u istoj važila (bila očuvana) funkcijska ovisnost (X,Y) --> Z.

Temeljne atomarne formule oblika

```
predikat(arg1, ..., argN).
```

nazivamo činjenicama (facts), i njihovo prisustvo u u bazi znači istinitost navedene instance sheme datoteke (relacije). Pravilo iz datoteke "pp"

```
pp(X,Y,Z) :- qq(X,Y,Z), not(rr(X,Y)).
```

kazuje, da je u bazi (točnije: datoteci "pp"),istinita svaka instanca sheme pp(X,Y,Z), za koju je istinita pripadna instanca sheme qq(X,Y,Z) datoteke "qq", a pritom nije istinita pripadna instanca sheme rr(X,Y) datoteke "rr".

Primjeri

Doslovni sadržaj datoteke "izlistavamo" pomoću instrukcije

```
 "lfile(Ime_datoteke)".
```

```
?- lfile(pp).
  pp(a, d, 4) :- true
  pp(a, c, 4) :- true
  pp(b, c, 3) :- true
  pp(_1, _2, _3) :- qq(_1, _2, _3) ,
                    not(rr(_1, _2))
yes

?- lfile(qq).
  qq(a, b, 6) :- true
  qq(c, d, 8) :- true
  qq(u, v, 8) :- true
  qq(u, s, 5) :- true
  qq(a, d, 4) :- true
yes

?- lfile(rr).
  rr(b, a) :- true
  rr(c, d) :- true
  rr(a, d) :- true
yes
```

Unos informacije vršimo naredbom "insert." Pokušajmo unijeti informaciju "pp(u,v,8)".

```
?- insert.
*: pp(u,v,8).
Information is redundant
Insert it ? (y./n.) *: n.
yes
```

Sistem nas obavještava da je ta informacija redundantna, tj. već deducibilna iz baze, te traži odgovor da li da ju prihvati ili ne. U ovom slučaju informaciju nismo prihvatili (odgovorivši "n."). Ukoliko želimo provjeriti je li informacija (u ovom slučaju: činjenica), koju smo željeli unijeti, zaista redundantna, izlistamo ukupnu ekstenziju odgovarajuće datoteke (tj. relacije). To činimo instrukcijom "lext(Ime_datoteke)".

```
?- lext(pp).
  pp(a, d, 4)
  pp(a, c, 4)
  pp(b, c, 3)
  pp(a, b, 6)
  pp(u, v, 8)
  pp(u, s, 5)
yes
```

Iz date ukupne ekstenzije datoteke "pp" očito je da je spomenuta informacija (kao ulazna), zaista redundantna jer je iz datoteke "pp" već deducibilna, na što ukazuje njeno prisustvo u ukupnoj ekstenziji datoteke "pp".

No, obzirom da ista nije prisutna u (eksplicitno datoj) ekstenziji datoteke "pp", slijedi da mora biti deducibilna pomoću intenzionalnog dijela datoteke "pp", tj. primjenom pravila. Objašnjenja o (svim mogućim) načinima njenog deduciranja možemo zatražiti pomoću instrukcije "expl.".

```
?- expl.
*: pp(u,v,8).

  pp(u, v, 8) follows from qq(u, v, 8)
                       and not(rr(u, v))
  qq(u, v, 8) is a fact in DB
  it is true that: NOT rr(u, v)

Another explanation ? (y./n.) *: y.

No (more) deduction(s)
yes
```

Dobiveno obrazloženje čitamo na slijedeći način: Da je "pp(u,v,8)" deducibilno (a time i "istinito u bazi"), slijedi iz toga što je deducibilno "qq(u,v,8)" i "not(rr(u,v))". "qq(u,v,8)" je deducibilno jer je to (eksplicitno prisutna) činjenica u bazi. "not(rr(u,v))"? slijedi iz toga što rr(u,v) nije deducibilno iz baze - a općenito važi (meta) princip, da je "not(Tvrdnja)" deducibilno (a samim time i istinito), onda kada "Tvrdnja" nije.

Pokušajmo unijeti u sistem informaciju "pp(u,v,9)".

```
?- insert.
 *: pp(u,v,9).
 Information would violates IC, because :
   pp(u, v, 9) and pp(u, v, 8) and 8 \== 9
 Insertion aborted
 yes
```

Sistem upozorava da bi unos informacije "pp(u,v,9)" prekršio uslove integriteta (za datoteku "pp"), jer bi tada informacije "pp(u,v,8)" i "pp(u,v,9)" spadale u (ukupnu) ekstenziju datoteke. Stoga informaciju "pp(u,v,9)" ne prihvaća.

No, slijedeći primjer pokazuje da uvjete integriteta možemo ("obilaznim putem" , ipak prekršiti.

```
 ?- insert.
 *: qq(u,v,9).
 Information inserted
 yes
```

Za datoteku "qq" nisu dati nikakvi uvjeti integriteta, te, obzirom da nije redundantna informacija "qq(u,v,9)" biva prihvaćena.

No, provjerimo sada jesu li i dalje ispunjeni zahtjevi integriteta za datoteku "pp". To činimo instrukcijom "valid(Ime_datoteke)".

```
 ?- valid(pp).
 File **pp** violates IC, because
   pp(u, v, 8) and pp(u, v, 9) and 9 \== 8
 are deducible from DB
 yes
```

Dakle, ažuriranjem jedne datoteke, možemo narušiti integritet druge. Jer, čak da su i bili dati uvjeti integriteta za datoteku "qq", ti ne bi morali zahtjevati upravo ona ograničenja, koja bi garantirala očuvanje integriteta u datoteci "pp". Očito, na posredno kršenje integriteta presudno utječe postojanje pravila (intenzije) u bazi podataka.

Zatražimo stoga, najprije obrazloženje odakle slijede "inkriminirane" informacije.

```
 ?- expl.
 *: pp(u,v,8) and pp(u,v,9).

    pp(u, v, 8) follows from qq(u, v, 8)
                             and not(rr(u, v))
    qq(u, v, 8) is a fact in DB
    it is true that: NOT rr(u, v)

    pp(u, v, 9  follows from qq(u, v, 9)
                            and not(rr(u, v))
    qq(u, v, 9) is a fact in DB
    it is true that: NOT rr(u, v)

 Another explanation ? (y./n.) *: y.

 No (more) deduction(s)
 yes
```

Očito dedukcija nedopustivih informacija u datoteci "pp" ide preko informacija "qq(u,v,8)" i "qq(u,v,9)" - koje u samoj datoteci "qq" ne smetaju - i nededucibilnosti informacije "rr(u,v)"! Stoga i postoji više načina da se datoteka "pp" vrati u valjano stanje. Najzanimljivijim izgleda slijedeći:

```
?- insert.
 *: rr(u,v).
 Information inserted
 yes

 ?- valid(pp).
 File pp is in accordance with IC
 yes
```

Dakle, insertiranjem neke informacije u datoteku možemo "povratiti integritet" nekoj drugoj datoteci. No, isto tako i brisanjem informacije, kako to slijedeći primjer pokazuje, integritet narušiti.

```
?- delete.
 *: rr(u,v).
 Informationd deleted
 yes

 ?- valid(pp).
 File **pp** violate IC, because
   pp(u, v, 8) and pp(u, v, 9) and 9 \== 8
 are deducible from DB
 yes
```

Izbrišimo konačno "pravog uzročnika" kršenja integriteta datoteke "pp".

```
?- delete.
 *: qq(u,v,9).
 Informationd deleted
 yes

 ?- valid(pp).
 File pp is in accordance with IC
 yes
```

Naravno, takvo nepouzdano djelovanje sistema nije prihvatljivo. Utoliko smo navedenim primjerima željeli prvestveno ukazati na neke od "popratnih efekata" uvođenja pravila u bazu, koji zaslužuju posebnu pažnju. Napomenimo samo, da bi jednostavno ali i najneefikasnije riješenje problema moglo bit, da se prilikom svakog (pojedinačnog!) ažuriranja bilo koje datoteke u bazi provjerava integritet svake pojedine datoteke u bazi.

Slijedeća instrukcija nam omogućava da prije pokušaja insertiranja nekog novog pravila (tj. zakona ili pak "znanja"), provjerimo (neke) od njegovih efekata na bazu podataka.

```
?- check.
 *: qq(X,Y,Z) implies pp(X,Y,Z).
 The rule does not hold because :
 qq(_1, _2, _3) is true but
 pp(_1, _2, _3) is NOT true
 for the following instances:

   qq(c, d, 8)
 yes
```

Dakle, promatrano pravilo (implikacija) u bazi ne važi (ili, u terminima teorije modela rečeno, ukupna ekstenzija baze (shvaćena kao struktura), nije model promatrane implikativne (i prećutno univerzalno zatvorene!) formule. To pak ujedno znači, da će njeno insertiranje u bazu (promatranu sada kao teoriju), povećati ukupnu ekstenziju baze (točnije: konsekvensa!), za instancu, koja odgovara navedenoj instanci antecedensa (tj. za "pp(c,d,8)"). Jednostavnije rečencloo, instrukcija "check." p kazala nam je da je iz baze deducibilna informacija "qq(c,d,8)", a da istodobno nije deducibilna informacija "pp(c,d,8)". To pak ujedno znači, da ako promatrano pravilo u bazu

mo Y^*_{C21}, a umjesto R^*_{α} stavljamo $B^*_{x_2}$. Iz tako dobivene relacije možemo odrediti momente t.j. očekivani broj događaja u repu 2.

Promotrimo sada prioritetnu grupu događaja u modelu na slici 3a. Ciklus obrade sastoji se iz vremenskih razmaka $x_1 + \gamma_3$. Prema tome relacija analogna (5.5) glasi:

$$Y_{C11} = B^*_{x_1}(s) \cdot R^*_{\gamma_3}(s)$$

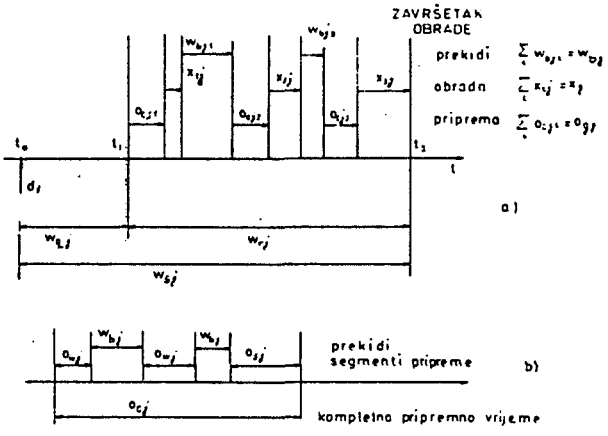
Ciklus praznog hoda nešto je kompliciraniji. Ako je rep 1 prazan vremenski razmak sastoji se iz nezavisnih vremenskih sekvenci γ_1, x_2 s vjerojatnošću $(1 - g_{20})$, γ_2 s vjerojatnošću g_{20}, te γ_3. Prema tome:

$$Y^*_{C12}(s) = R^*_{\gamma_1}(s)((1 - g_{20})B^*_{x_2}(s) + g_{20}R^*_{\gamma_2}(s)) \cdot R^*_{\gamma_3}(s) \qquad (5.6)$$

Usporedbom modela 3a i 3b opažamo da u relaciji (5.3) umjesto $R^*_{\beta_2}$ stavljamo Y^*_{C12}, umjesto $R^*_{\alpha+\beta_1}$ stavljamo Y^*_{C11}, a umjesto R^*_{α} stavljamo $B^*_{x_1}$. Iz dobivenog izraza, koristeći moment generirajuće svojstvo izračunamo očekivanje broja događaja u prioritetnom repu 1. Vjerojatnost g_{20} (prazan rep 2) izračunamo iz (5.4) tako da umjesto $E(\alpha + \beta_1)$ stavimo očekivanje pericda zauzetosti $E(y_{22})$ dobivenog moment generirajućim postupkom iz Y^*_{C22}, a umjesto $E(\beta_2)$ stavimo očekivanje $E(y_{21})$ dobiveno iz Y^*_{C21}. Intenzitet $\lambda = \lambda_2$. Prema /5/, transformacija razdiobe vjerojatnosti broja događaja $Q(z)$ i Laplaceova transformacija gustoće razdiobe vjerojatnosti vremena u M/G/1 sustavu $W^*_{w_s}(s)$, vezani su jednostavno $Q(z) = W^*_{w_s}(\lambda - \lambda z)$, te su time relacijom (5.3) određeni i vremenski parametri sustava.

6. NEPRODUKTIVNA OBRADA U PRIORITETNIM SUSTAVIMA M/G/1/PRI S PREKIDANJEM

Razmotrit će se model računarskog sustava s k-grupa izvora slučajnih događaja uz mogućnost prekida obrade i nastavka u točki prekida. Unutar pojedine prioritetne grupe sustav odabire događaje u obradu prema redoslijedu dolaska (FCFS). Prema slici 4a, u času t_o događaj d_j dolazi u sustav i priključuje se repu čekanja. U času t_1 događaj d_j prvi puta ulazi u obradu. Vrijeme čekanja u repu iznosi $w_{qj} = t_1 - t_o$. Prije svakog početka obrade, potrebno je kompletno pripremno neproduktivno vrijeme, koje za događaj d_j iznosi o_{cj}. Iza kompletnog pripremnog neproduktivnog vremena o_{cj1}, slijedi obrada (dio od x_j), pa mogući prekid, tokom kojeg je obrada blokirana kroz vremenski razmak w_{bj1}, zatim opet kompletno pripremno neproduktivno vrijeme o_{cj2}, te se tako ciklus ponavlja do trenutka t_2 kad



Slika 4. M/G/1/PRIORIT. sustav s prekidanjem

događaj konačno napušta obradu. Kompletno pripremno neproduktivno vrijeme o_{cji}, sastoji se prema slici 4b, od segmenta izgubljenog neproduktivnog vremena o_{wj}, segmenta blokiranog vremena w_{bj} i konačno segmenta uspješnog pripremnog vremena o_j, budući da pretpostavljamo mogućnost prekida i tokom neproduktivnog vremena pripreme obrade. Slučajna veličina o_j data je za pojedinu grupu izvora slučajnih događaja razdiobom $V_j(o)$, gustoćom razdiobe $v_j(o)$ i pridruženom transformacijom $V^*_{oj}(s)$.

Slučajni vremenski razmak od trenutka prvog ulaska događaja d_j u obradu do trenutka potpunog završetka obrade (rezidentno vrijeme) $w_{rj} = t_2 - t_1$ zauzima ključnu ulogu u analizi prioritetnih sustava s prekidanjem događaja u obradi. U sustavima bez prekidanja, evidentno je $w_{rj} = x_j$.

Opisani model sustava s neproduktivnom programskom podrškom može se dekomponirati na prekidne sustave bez neproduktivne programske podrške s nastavkom obrade u točki prekida (M/G/1/PRI-PRN) te na sustave s nastavkom obrade s početnim vremenom obrade (M/G/1/PRI-PRP).

Za sustav bez neproduktivne obrade i nastavkom obrade u točki prekida, rezidentno vrijeme događaja d_j sastoji se iz segmenta obrade x_{ij} te vremena blokade w_{bj}, odnosno iz totalnog vremena obrade (x_j) i segmenata vremena blokade w_{bj}. U sustavu s nastavkom u točki prekida, ukupno vrijeme obrade jednako je zahtjevanom vremenu obrade j-te grupe, tj. $x_{gj} = x_j$, pa su transformacije pridruženih gustoća razdioba identične.

U sustavu bez neproduktivne obrade i nastavkom obrade u točki prekida s identičnim, početnim, vremenom obrade izabranim iz $B_j(x)$, ukupno vrijeme obrade sadrži nekoliko izgubljenih (prekinutih) segmenata obrade x_{wj} i jedno

uspješno vrijeme obrade x_j. Rezidentno vrijeme u takvom sustavu sastoji se iz N zavisnih parova $x_{wj} + x_{bj}$ i jednog vremenskog razmaka uspješne obrade x_j. U literaturi /6/ nalazimo transformaciju gustoće razdiobe rezidentnog vremena $Y^*_{w_{rj}}(s)$.

Budući da u ovom slučaju ukupno vrijeme obrade x_{gj} nije jednako zahtjevanom vremenu obrade x_j, to se transformacija pridružena veličini x_{gj} dobije iz $Y^*_{w_{rj}}(s)$ uz supstituciju $Y^*_{w_{bj}}(s) = 1$; tj. izostavljanjem segmenata vremena blokiranja: $Y^*_{w_{bj}}(s)$ je transformacija gustoće razdiobe vremena blokiranja.

U literaturi /7/ također nalazimo Laplaceovu transformaciju razdiobe vremena u prekidnim sustavima $W^*_{w_{sj}}(s)$ izraženu preko rezidentnog vremena, i vremena blokiranja.

Razumljivo je da relacija za $W^*_{w_{sj}}(s)$ vrijedi za sve prekidne sustave (PRN, PRP) jer se razlučivanje odvija baš u rezidentnom vremenu odnosno pridruženoj transformaciji $Y^*_{w_{rj}}(s)$. Iz tih izraza koristeći svojstvo generiranja momenata izvedeno je očekivanje vremena u repu čekanja grupe j:

$$E(w_{qj}) = \frac{\lambda_j E(w_{rj}^2)}{2(1-\lambda_j E(w_{rj}))} + \frac{\lambda_a E(w_{bj}^2)}{2(1+\lambda_a E(w_{bj}))} = E(w_{sj}) - E(w_{rj}) \tag{6.1}$$

Relacija (6.1) daje nam najvažnije parametre u sustavu, ali izražene preko slučajnih veličina w_{rj} i w_{bj}. λ_a je intenzitet prekidanja t.j. $\lambda_a = \lambda_1 + \lambda_2 + \ldots + \lambda_{j-1}$.

U trenutku prekida (početak segmenta w_{bj}) obrade događaja d_j, u sustavu se nalazi samo jedan događaj višeg prioriteta (s indeksom $<j$). Promatrani segment w_{bj} sastoji se iz rezidentnog vremena prioritetnog događaja koji je uvjetovao prekid, te iz rezidentnih vremena događaja iz grupa j-1, j-2, ..., 1, koji su stigli u međuvremenu. Tako vrijeme blokiranja i rezidentno vrijeme ovise jedno o drugom rekurzivno (rezidentno vrijeme 1. grupe određuje period blokiranja 2. grupe itd.). Vrijeme blokiranja možemo izraziti kroz poopćene cikluse zauzetosti:

$$Y^*_{w_c}(s) = Y^*_{w_{bo}}(s + \lambda - \lambda Y^*_{w_b}(s)) \tag{6.2}$$

gdje je w_c ciklus blokiranja u kojem je prvi dio jednak vremenskom razmaku w_{bo}, a ostali dijelovi su normalni periodi blokiranja. Iz (6.2) slijede momenti:

$$E(w_c) = \frac{E(w_{bo})}{1 - \lambda E(w_b)} \tag{6.3}$$

$$E(w_c^2) = \frac{E(w_{bo}^2)}{(1-\lambda E(w_b))^3} E(w_{bo}) + \frac{E(w_{bo}^2)}{(1-\lambda E(w_{bo})^2} \tag{6.4}$$

Period blokiranja događaja iz grupe j + 1 jednak je ciklusu zauzetosti događaja iz grupe j. Zbog dolazaka događaja iz grupe j, s vjerojatnošću $\lambda_j/(\lambda_a + \lambda_j)$, period blokiranja $w_{b,j+1}$ jednak je ciklusu zauzetosti događaja grupe j s početnim vremenom w_{rj} i normalnim periodima također w_{rj}. Zbog dolazaka događaja iz grupa 1 do j-1, s vjerojatnošću $\lambda_a/(\lambda_a + \lambda_j)$, period blokiranja $w_{b,j+1}$ jednak je ciklusu zauzetosti događaja grupe j s početnim vremenom w_{bj} i normalnim periodima w_{rj}. Koristeći (6.3) i (6.4) možemo pisati:

$$E(w_{bj+1}) = \frac{\lambda_j}{(\lambda_a+\lambda_j)} \frac{E(w_{rj})}{(1-\lambda_j E(w_{rj}))} + \frac{\lambda_a}{(\lambda_a+\lambda_j)} \frac{E(w_{bj})}{(1-\lambda_j E(w_{rj}))} \tag{6.5}$$

$$E(w_{b,j+1}^2) = \frac{\lambda_a E(w_{bj}^2)(1-\lambda_j E(w_{rj})) + \lambda_j E(w_{rj}^2)(1-\lambda_a E(w_{bj}))}{(\lambda_a+\lambda_j)(1-\lambda_j E(w_{rj}))^3}$$

Za grupu događaja j + 1 vrijedi (6.1), direktno. Uz supstituciju (6.5) i (6.6) slijedi rekurzivna formula:

$$E(w_{q,j+1}) = \frac{\lambda_{j+1} E(w_{r,j+1}^2)}{2(1-\lambda_{j+1} E(w_{r,j+1}))} + \frac{E(w_{qj})}{1-\lambda_j E(w_{qj})} \tag{6.7}$$

Analizirajući sliku 4a, uočavamo da se nakon prvog segmenta o_{cj1} promatrani sustav ponaša kao M/G/1PRI-PRN s intervalima blokiranja $w_{bji} + o_{cji}$, te zbog nezavisnosti slijedi za rezidentno vrijeme:

$$Y^*_{w_{rj}}(s) = Y^*_{o_{cj}}(s) \cdot B^*_j(s + \lambda_a - \lambda_a Y^*_{w_{bj}}(s)\ Y^*_{w_{cj}}(s))$$

Iz gornjeg izraza iskoristimo moment generirajuće svojstvo i izračunamo momente:

$$E(w_{rj}) = f_1(E(o_{cj}), E(w_{bj}), E(x_j)) \tag{6.8}$$

$$E(w_{rj}^2) = f_2(E(o_{cj}), E(o_{cj}^2), E(w_{bj}), E(w_{bj}^2), E(x_j), E(x_j^2)) \tag{6.9}$$

Prema slici 4b, kompletno neproduktivno vrijeme o_{cj} analogno je rezidentnom vremenu u sustavu s prekidima i nastavkom obrade s početnim vremenom. U/6/ nalazimo izraz za transformaciju rezidentnog vremena, koja s novim oznakama glasi:

$$Y^*_{o_{cj}}(s) = \int_0^\infty \frac{(s+\lambda_a) e^{-(s+\lambda_a)o}}{s + \lambda_a - \lambda_a Y^*_{w_{bj}}(s)(1-e^{-(s+\lambda_a)o})} v_j(o)\, do \tag{6.10}$$

Iz (6.10) izračunaju se momenti:

$$E(o_{cj}) = f_3(E(w_{bj}), E(o_{gj})) \tag{6.11}$$

$$E(o_{cj}^2) = f_4(E(w_{bj}), E(w_{bj}^2), E(o_{gj}), E(o_{gj}^2)) \tag{6.12}$$

Stavljajući u izraz (6.10) $Y^*_{w_{bj}}(s) = 1$, slijedi Laplaceova transformacija pridružena ukupnom vremenu neproduktivne obrade iz koje se mogu

IMPLIKACIONI PROBLEM ZA FUNKCIONALNE ZAVISNOSTI I MEHANIČKO DOKAZIVANJE TEOREMA

MALEKOVIĆ MIRKO
CVVTŠ "GENERAL ARM. IVAN GOŠNJAK", ZAGREB

UDK : 681.3.01:519

Jedan od važnih problema u teoriji projektiranja relacionih baza podataka jeste specifikacija uvjeta koje mora zadovoljavati relaciona šema da bismo korektno modelirali razmatrani dio "svijeta". Od posebnog interesa su uvjeti nazvani zavisnostima. Optimalna specifikacija skupa zavisnosti vodi na implikacioni problem. U ovom radu, predlažemo metod za rješavanje implikacionog problema za funkcionalne zavisnosti. Metod je baziran na reprezentiranju funkcionalnih zavisnosti pomoću formula logike prvog reda i primjeni procedura dokazivanja koje su razvijene u teoriji mehaničkog dokazivanja teorema.

IMPLICATION PROBLEM FOR FUNCTIONAL DEPENDENCIES AND MECHANICAL THEOREM PROVING:
One of the important issues in the design theory of the relational database schemas is the specification of the constraints that the data must satisfy to model correctly the part of the world under consideration. Of particular interest are the costraints called data dependencies. In this work we have proposed a method for solving of implication problem for functional dependencies. The proposed method is based on representation of functional dependencies by formulas of first order logic and aplication of proof procedures which are developed in mechanical theorem proving.

1. Uvod

U teoriji projektiranja relacionih baza podataka ističu se dva problema. Jedan problem jeste karakterizacija dobre relacione šeme za reprezentaciju informacije. Rješenje ovog problema vodi na tzv. normalne forme. Drugi problem sastoji se u izboru dobrog skupa uvjeta integriteta. Ovdje, pod dobrim skupom uvjeta mislimo na skup koji je lako kontrolirati i održavati. Tretiranje, kako prvog tako i drugog problema, dovelo je do implikacionog problema za dani skup uvjeta. Do sada su poznata dva prilaza u rješavanju implikacionog problema. Jedan prilaz je baziran na konceptu formalnog sistema, [2], [4], [7], [8]. Drugi prilaz je semantičke prirode. Baziran je na chase procesu koji je razvijen u [1], [3], [6]. U ovom radu, prezentiramo novi prilaz rješavanju implikacionog problema za funkcionalne zavisnosti. Metod je baziran na reprezentiranju funkcionalne zavisnosti pomoću Skolemove standardne forme i primjeni procedura dokazivanja koje se baziraju na rezolucijskom principu. U izvjesnom smislu, predloženi metod je komplemetaran prilazu baziranom na konceptu formalmog sistema. Naime, točnost pravila formalnog sistema se može dokazati koristeći naš metod, što je i pokazano u sekciji 4. ovog članka. U daljem pretpostavljamo poznavanje teorije relacionih baza podataka na nivou [9], te elemenata rezolucijskog metoda. Detaljan opis procedura dokazivanja, koje se baziraju na rezolucijskom principu, imamo u [5].

2. Implikacioni problem za funkcionalne zavisnosti.

U ovoj sekciji uvodimo pojam funkcionalne zavisnosti za danu relacionu šemu, a onda karakteriziramo implikacioni problem.

Definicija Neka je $R(A_1,..,A_n)$ relaciona šema nad skupom atributa $U=\{A_1,..,A_n\}$, $X,Y \subseteq U$. Izraz oblika $X \longrightarrow Y$ zovemo funkcionalna zavisnost; govorimo još da X funkcionalno određuje Y.

Neka je u danom trenutku vremena relaciona šema R predstavljena relacijom r . Relaciju r zovemo primjerom od R , a elemente od r zovemo tiplovi. Konvencionalno, r se reprezentira tabelom; redovi tabele su tiplovi, a stupci su imenovani atributima iz U .

Definicija Kažemo da $X \longrightarrow Y$ vrijedi (ili da je zadovoljeno) u r ako i samo ako

$$\forall t_1,t_2 \in r\,(t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]).$$

Iskazana definicija znači da $X \longrightarrow Y$ vrijedi u r ako i samo ako za bilo koja dva tipla t_1, t_2 iz relacije r vrijedi: iz jednakosti tiplova t_1,t_2 na atributima iz X slijedi jednakost istih tiplova na atributima iz Y.

Definicija Kažemo da $X \longrightarrow Y$ vrijedi u relacionoj šemi R ako i samo ako vrijedi u svakom njenom primjeru.

Sa FD(U) označimo skup svih funkcionalnih zavisnosti nad U tj. $FD(U)=\{X \longrightarrow Y / X,Y \subseteq U\}$. Neka je sada $C \subseteq FD(U)$, $f \in FD(U)$.

Definicija Za relaciju r kažemo da je model od f ako i samo ako je f zadovoljeno u r. Dalje, r je model od C ako i samo ako je r model svakog člana iz C.

Sada možemo uvesti pojam logičke implikacije za funkcionalne zavisnosti.

Definicija Kažemo da C logički implicira f ako i samo ako svaki model od C jeste model od f. Pisat ćemo $C \models f$ ili $\frac{C}{f}$ zavisno od pogodnosti.

Za dane $C \subseteq FD(U)$, $f \in FD(U)$, implikacioni problem jeste pitanje da li $C \models f$.
Uvedimo skup $C^* = \{f \in FD(U) / C \models f\}$. C^* je zatvarač od C. Implikacioni problem jeste zapravo pitanje da li je $f \in C^*$. Opišimo, ukratko, važnost rješavanja navedenog problema.
Bilo koji skup $C_1 \subseteq FD(U)$, takav da je $C_1^* = C^*$, C_1 je pokrivač od C, ima isti skup modela M kao i C. Ako sa M(C) označimo skup modela za skup uvjeta C, navedeno može se pisati kao $M(C_1)=M(C)$. Očigledno, da u izboru pokrivača trebamo preferirati pokrivač sa reduciranim brojem elemenata. Sada, kada smo izabrali adekvatan skup uvjeta $C \subseteq FD(U)$, možemo biti pred problemom, da li novi uvjet $f \in FD(U)$ treba dodati skupu C. Ovo vodi na implikacioni problem. Naime, trebamo ispitati nezavisnost f od C. Pri tome imamo slijedeću definiciju.

Definicija Za f kažemo da je nezavisno od C ako i samo ako $C \not\models f$ i $C \not\models \sim f$. Ovdje, $C \not\models f$ znači da nije $C \models f$, a $\sim f$ znači da nije f.

U slučaju nezavisnosti, f treba dodati skupu C. Ako $C \models f$, onda je nekorisno da dodajemo f u C jer bismo dobili redundantan skup $C \cup \{f\}$ tj. vrijedilo bi $M(C)=M(C \cup \{f\})$. Ako $C \models \sim f$, onda je $C \cup \{f\}$ kontradiktoran skup uvjeta, pa je operacija dodavanja zabranjena.
U teoriji funkcionalnih zavisnosti su razvijeni formalni sistemi gdje je jedan od glavnih zadataka pokazati točnost i kompletnost formalnog sistema. Dokaz točnosti formalnog sistema vodi na implikacioni problem. Cijela sekcija 4. bit će posvećena ispitivanju točnosti formalnog sistema predloženog u [9].

3. Reprezentacija funkcionalne zavisnosti pomoću Skolemove standardne formule

Neka je $R(A_1,..,A_n)$ relaciona šema, r primjer od R i $X,Y \subseteq \{A_1,..,A_n\}$. Rekli smo da funkcionalna zavisnost $X \longrightarrow Y$ vrijedi u r ako i samo ako

$$\forall t_1,t_2 \in r\,(t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]).$$

Ako uvedemo predikat E_X, gdje $E_X(t_1,t_2)$ ima intendirano značenje: tiplovi t_1 i t_2 su jednaki na skupu atributa X tj. $t_1[X] = t_2[X]$, funkcionalnoj zavisnosti možemo dati oblik:

(1) $X \longrightarrow Y: \forall t_1 \forall t_2 [E_X(t_1,t_2) \Rightarrow E_Y(t_1,t_2)]$.

Prema tome, formulu (1) interpretiramo na skupu tiplova (relaciji) r. Ponovimo, da je interpretacija r model za (1) ako (1) vrijedi u r, te da (1) vrijedi u R ako je svaki primjer od R model od (1)

Primjer 1.

Neka je zadana relaciona šema R(A,B,C), koja je u danom trenutku vremena predstavljena relacijom r :

r	A	B	C
a	1	o	2
b	1	o	o
c	1	o	1

U tabeli, a,b,c su oznake za tiplove relacije r. Lako uočavamo da je r model za $A \longrightarrow B$, dok nije model za $\{A,B\} \longrightarrow C$. Da r nije model za $\{A,B\} \longrightarrow C$ slijedi iz činjenice da je $E_{\{A,B\}}(a,b)$ i $\sim E_{\{C\}}(a,b)$.
U skladu sa uobičajenom notacijom u teoriji baza podataka, jednočlan skup {A} pisat ćemo kao A , a uniju $X \cup Y$ skupova X i Y atributa kao XY.
Napišimo sada Skolemovu standardnu formu za $X \longrightarrow Y$. Iz (1) lako dibijemo da je tražena forma

(2) $SSF(X \longrightarrow Y): \sim E_X(t_1,t_2) \vee E_Y(t_1,t_2)$.

kspertne sisteme odnosno baze znanja (knowledge bases) potpuno implementiran u Prologu, dat je u /Miyachi,84/, ali baza nije deduktivna u ovdje definiranom smislu jer ne sadrži pravila (formule), već ostaje relacijskom. Rad na razvoju sistema, koji sadrži i pravila opisan je u /Kitakami,84/, stime što je u tom sistemu naglasak dat na razvoj mogućnosti induktivnog zaključivanja.

U modelu koji ovdje razvijamo naglasak je dat na kooperativnost sistema, pod čime podrazumijevamo sposobnost sistema, da sa korisnikom kkmunicira u jeziku za korisnika čim prikladnijem, te da daje obrazloženja uspješnih (i neuspješnih) pokušaja dedukcije tražene informacije, kao i upozorenja korisniku o mogućim jonsekvencama pojedinih akcija u sistemu. Žrikazane su osnovne konture sistema, naglašeni neki specifišni problemi, koje uvođenje pravila u bazu podataka donosi, te ilustriran rad dosad implementiranih funkcija.

2. OPIS MODELA

U ovom odjeljku dat je kratak opis modela i osnovnih (do sada implementiranih) instrukcija. Prološka implementacija deduktivne baze pogodna je za razvoj jezika sa širokim mogućnostima postavljanja upita (i komuniciranja uopće /Radovan,85/). Primjena datih instrukcija ilustrirana je primjerima u odjeljku (3).

Osnovne instrukcije sistema

lfile(Ime_dat).

Izlistava (doslovni) sadržaj datoteke "Ime_dat". Pravila pritom ostaju u izvornom obliku, a ukupna ekstenzija baze ne daje se eksplicitno.

lext(Ime_dat).

Izlistava ukupnu ekstenziju datoteke "Ime_dat". Pravila iz intenzije baze bivaju pritom "prevedena" u pripadne ekstenzije, pomoću njih deducibilne.

insert.
*: (Informacija).

Instrukcija "insert." je temeljna instrukcija sistema, te ju podrobnije opisujemo. Informacija, koja se unosi, može biti atomarna formula ("n-torka", odnosno temeljna instanca sheme relacije), ili pak pravilo (zatvorena formula), tipa

Consec :- Ant1 * Ant2 * ...* AntN.

gdje su Ant1, Ant2, ..., AntN literali (tj. atomarne formule ili negirane atomarne formule), a Consec atomarna formula. Dakle, pravila, kao elementi baze podataka (i ujedno Prolog instrukcije (klauzule)), čine širu klasu formula od klase definitnih klauzula (definiranih u odjeljku (1)), upravo dopuštanjem i negiranih atomarnih formula u pravilu. Pritom važi (meta)princip, da je cilj "not(A)" zadovoljen (istinit), onda kada cilj "A" nije zadovoljen (istinit) u datoj bazi podataka. Primjena negacije ograničena je na temeljne instance atoma (i formula, uopće), a to znači da u trenutku pozivanja cilja "not(A)", moraju (eventualne) varijable iz "A" biti već instancirane od strane cilju "not(A)" prethodećih ciljeva. Ovaj zahtjev izgleda (barem u operativnom smislu), ekvivalentnim Ullmanovom zahtjevu po sigurnim (safe) formulama.

Kod unosa, najprije se provjerava redundantnost nove informacije. Informaciju smatramo redunrantnom, ukoliko je u trenutku unošenja već deducibilna (logički izvediva (slijedi)), iz baze podataka. U sistemu se to provjerava tako, da se u slućaju unošenja činjenica (n-torki), iste pokušaju najprije deducirati iz baze. U slučaju unošenja pravila (formula), pokuša se pokazati da ekstenzija antecedensa (deducibilna iz baze), nije veća od ekstenzije konsekvensa (deducibilne iz baze). Drugim riječima, pokušavamo pokazati, da iz baze nije moguće deducirati takvu instancu antecedensa, za koju nebi istodobno bila (već) deducibilna i odgovarajuća instanca konsekvensa. Ukoliko u tome uspijemo, to onda znači da je informacija koja se unosi redundantna (tj. da ne puvećava ukupne ekstenzije!). Sistem na to upozorava, te se prema zahtjevu korisnika, informaciju unosi ili ne unosi u sistem.
U slijedećem koraku provjerava se da li unošenje nove informacije dovodi do kršenja uslova integriteta baze podatka. Ograničenja, koja se uslovima integriteta baze postavljju, mogu se podijeliti na tri osnovne grupe:

- ograničenje dopuštene vrijednosti (tj. domene), za pojedine atribute u shemi relacije,
- strukturna ograničenja, u koja spadaju funkcijske ovisnosti,
- ograničenja na načine i pravo korištenja baze podataka.

Od navedenih, u modelu je razmatrana samo problematika kontrole i očuvanja funkcijske ovisnosti. Ujedno je pokazano koje sve probleme prisustvo pravila u bazi postavlja pred kontrolu integriteta baze. Uslovi integriteta predstavljeni su ovdje u "negativnom obliku", tj. ciljem, koji u bazi ne smije biti zadovoljen (deducibilan).

Ukoliko bi pak unošenjem nove informacije taj cilj postao zadovoljiv (deducibilan), onda se nova informacija ne prihvaća a korisnok obavještava.

Prihvaćena informacija unosi se u bazu, a zatim se vrši kontrola generirane redundantnosti. To se izvodi na taj način, da se svaku eksplicitno prisutnu informaciju iz baze redom (privremeno) odstrani, pa zatim istu pokuša deducirati. Ukoliko pokušaj dedukcije uspije, daje se obavijest o (stvorenoj) redundantnosti, i informacija zatim briše ili zadržava, prema zahtjevu korisnika. Kontrolom generirane redundantnosti, proces unosa (prihvaćene i (u odnosu na IC), valjane), informacije završava.

Navodimo slijedeće (potencijalne) razloge za zadržavanje redundantnih informacija u sistemu :

- eksplicitno prisustvo činjenice može ubrzati proces dedukcije,
- neke informacije, koje su u datom trenutku redundantne, mogu kasnijim ažuriranjima (bilo unosom bilo brisanjem drugih informacija), to prestati biti, te bi njihovo obavezno isključivanje (ili nedopuštanje unosa), nepotrebno

otežavalo posao ažuriranja baze,

- dopuštanje prisutnosti redundance može znatno olakšati rad u fazi razvijanja baze znanja i to upravo iz rzloga navedenih u gornjoj točci.

Valja međutim napomenuti, da prisustvo redundantnih informacija u bazi postavlja dodatne teškoće (ili bar zahtjeve) pred valjanu implementaciju funkcija agregacije, pout "suma", "prosjek" i sličnih.

```
delete.
*: (Informacija).
```

Naredbom "delete" brišemo informaciju datu kao argument naredbe. Informacija može biti činjenica ili pravilo; ukoliko se informacija ne nalazi u bazi podataka, ne poduzima se ništa, a korisnik o tome obavještava.

```
expl.
*: (Cilj).
```

Ovom instrukcijom tražimo od sistema odgovor, da li je (potencijalno složen) cilj "Cilj" deducibilan iz sistema, te ako jeste, tražimo objašnjenja, koji su (sve) mogući "putovi" njegova deduciranja. Program, kojim je navedena instrukcija implementirana, navodimo u cjelini, jer nam dato programsko rješenje izgleda jednostavnijim od programa za analogne instrukcije (predikate), kao što su "demo " i "deduce" (vidi npr. /Kowalski,79/, /Bowen,81/, /Kitakami,84/, /Miyachi,84/).

```
/* izvođenje i obrazlaganje dedukcije */
expl :- read(X), decomp(X,A),
        prove(A,L), nl, show(L), nl,
        write('Another explanation ?(y./n.)'),
              read(Ans), nl, next_one(Ans).
expl :- write('No (more) deduction(s)'),nl, !.

prove((A,B),L) :- prove(A, L1), prove(B, L2),
                  append(L1,L2,L).
prove((A;B),L) :-  prove(A,L);  prove(B,L).
prove(not((A)), _) :- prove((A), _),!, fail.
prove(not(A),[not(A)]).
prove(A,[A from B|C]) :- clause(A,B),
                         prove(B,C).
prove(A, [is_true(A)]) :-
          functor(A, F, N),
          not(member(F,['',', not,';',true])),
          prolog_system_predicate(F, N),
          call(A).
prove(true,[]).
```

Slijedeće tri instrukcije omogućavaju nam da prije insertiranja neke informacije u sistem (tj. u bazu),provjerimo neka njezina svojstva i (buduće) učinke na bazu podataka.

```
check.
*: (Antec implies Consec).
```

Upitom toga tipa tražimo od sistema da provjeri, vrijedi li (već) u bazi podataka promatrano pravilo tipa

"Antecedens implicira Konsekvens".

Ukoliko ne vrijedi, sistem navodi one instance antecedensa deducibilne iz baze, za koje iz baze nisu deducibilne odgovarajuće instance konsekvensa.

```
extinf.
*: (Informacija).
```

Ovom instrukcijom tražimo "ekstenziju informacije", tj. koje su sve instance sheme relacije deducibilne iz antecedensa razmatranog pravila. Uloliko je riječ o činjenici, onda je njena ekstenzija samo ona sama.

```
newext.
*: (Informacija).
```

Instrukcija slična gornjoj, stime da se njome provjerava koje su instance deducibilne samo iz "Informacije" (pravila), koje razmatramo, a bez primjene tog pravila ne bi uopće bile deducibilne. Drugim riječima tražimo koliko "proširenje" ukupne ekstenzije donosi primjena toga pravila. Ukoliko se radi o činjenici (a ne pravilu), onda je "newext" činjenice jednaka njoj samoj, ako ista nije već deducibilna iz baze, odnosno praznom skupu, ako jeste deducibilna.

```
all.
*: ((N-torke) such_that Uvjeti).
```

Ovaj tip upita analogan je upitima (SELECT-FROM-WHERE) iz relacijskog jezika SQL. Odgovor na upit je lista svih N-torki, koje zadovoljavaju Uvjete. Pritom u Uvjetima mogu nastupati logički operatori kao i funkcije agregacije. Činjenica, da je neka iformacija (instanca sheme u bazi), ponekad deducibilna na dva ili više načina (što možemo provjeriti pomoću naredbe "expl."), jeste razlogom da uobičajene proloske funkcije "bag_of" i "set_of" nisu direktno upotrebljive za implementaciju fukcija (instrukcija) tipa "all.", kojima zahtjevamo od sistema sve one "n-torke", za koje su ispunjeni uslovi iz upita. "Bag_of" funkcija bila bi neprikladna jer bi svaku informaciju (element ukupne extenzije baze), uzela u obzir (prilikom sumiranja i slično), onoliko puta na koliko je načina ta informacija deducibilna u sistemu! - što, naravno, u slučajevima gdje postoji redundanca u sistemu, ne bi davalo ispravne rezultate. S druge strane, primjena funkcije tipa "set_of", kojom se višestruko deducibilna informacij tretira samo jedanput, ne bi radila u slučaju kada je potrebno izvršiti npr. zbrajanje vrijednosti nekog atributa na skupu n_torki iz datoteke. Naime, u tom slučaju, svaka različita vrijednost atributa bila bi uzeta u obzir samo jednom, što, naravno, nije ispravno jer više različitih n-torki (instanci) može imati jednaku vrijednost promatranog atributa, a pribrojiti treba ipak sve (tj. svaku!). Taj smo problem ovdje riješili modifikacijom funkcije "set_of" - konkretno, procedurom "all _u(T,G,L)", koja ujedno čini osnov za implementaciju svih funkcija tipa "SELECT-FROM-WHERE". Suština modifikacije astoji se u tome, da se u klasičnoj "trojci" (N_torka, Cilj, Lista), Listu formira tako, da se najprije generiraju jedinstveni (tj. bez ponavljanja!), parovi (N_torka, Cilj), a zatim sve N_torke iz parova (među kojima može biti i više jednakih N_torki!), "pokupe" u Listu. Taj postupak (program slijedi), uspješno rješava oba gore navedena problema u vezi sa dupliciranjem.

```
/* instr. "all" i podr. fun. */
all :- read(X such_that Cond),
       decomp(Cond,Dcond),
       all_u(X,Decond,L), show(L), !.

all_u(T,G,_) :- find_tuples_u(T,G).
```

MIKROPROLOG

MATJAŽ GAMS*, TATJANA ZRIMEC**
*INSTITUT „JOŽEF STEFAN, LJUBLJANA"
**FAKULTETA ZA ELEKTROTEHNIKO, LJUBLJANA

UDK : 681.3.06:519.682

Podana je primerjava mikroprologa s prologom in izkušnje pri poučevanju. Mikroprolog hitro postaja ena najbolj razširjenih modernih variant prologa, zato je podana ocena sprememb in izboljšav glede na bolj ustaljene inačice. Poglavitni namen mikroprologa pa je vzgoja, zato so posebej analizirane vzgojne poante tega jezika, zlasti izkušnje pridobljene s poučevanjem najmlajših učencev in pri izobraževanju odraslih.

A comparison is given between MICRO-PROLOG and PROLOG and especially experiences with teaching MICRO-PROLOG. MICRO-PROLOG is one of the modern wide spread variants of PROLOG so it was interesting to evaluate improvements and changes according to more standard PROLOGs. Since the main purpose of MICRO-PROLOG is education special care was devoted to experiences with teaching youngest pupils and adults thus evaluating full spectrum of learning characteristics.

1. Uvod - zgodovina razvoja

Idejna zasnova logike kot programskega jezika se pojavlja že okrog leta 1970 /1,2/, vendar pot od ideje do realizacije ni bila enostavna in premočrtna. Že ob prvih implementacijah pa se je pojavilo prvo veliko razpotje. Ena prvih implementacij je bil PLANNER (Hewitt, M.I.T.), ki pa je po velikih in neizpolnjenih pričakovanjih kmalu neslavno propadel. Verjetno je to eden poglavitnih razlogov, da v ZDA še danes prevladuje lisp. Druga učinkovitejša in enostavnejša implementacija je bil PROLOG (PROgramming in LOGic) (Colmerauer, Marseille,1972). Ta veja se je razširila v množico bolj ali manj podobnih inačic. Mikroprolog je ena novejših inačic in verjetno tudi najbolj razširjena, saj obstaja v nekoliko prilagojeni verziji celo za hišne računalnike tipa sinclair ZX ali commodore 64. Prav tako ga dobimo na raznih operacijskih sistemih, recimo CPM/80, CPM/86 ali MSDOS/PCDOS /3/. Med avtorji mikroprologa omenimo predvsem Kowalskega, Ennalsa in McCaba /3,4,5/. Posebej zanimiv je Robert Kowalski,saj je položil temelje logičnega programiranja /1,2/. F.G.McCabe je pravzaprav glavni implementator mikroprologa, J.R. Ennals pa se je ukvarjal predvsem z uporabo mikroprologa v pedagoške namene, v okviru projekta "Logic as a Computer Language for Children". Večina dela je potekala na Imperial College v Veliki Britaniji od leta 1980 dalje.

Prolog kot implementacija logičnega programiranja je osnova japonske 5. generacije računalnikov in poleg Japonske doživlja velik vzpon predvsem v Evropi. Prolog je precej razširjen zlasti v Sloveniji in tudi drugod v Jugoslaviji. Veliko zaslug pri uvajanju tega jezika gre na račun I. Bratka /6,7/. Prolog je že nekaj let redni učni predmet na katedri za računalništvo in informatiko Fakultete za elektrotehniko Ljubljana, večino prevajalnikov za prolog pa smo nakupili ali dobili preko Instituta Jožef Stefan, kjer dokončujemo prevajalnik za prolog v pascalu.

2. Nekaj enostavnih primerov

V nasprotju s klasičnim programiranjem, ki zahteva "strog" in dobro premišljen pristop ter točno določen potek izvajanja /8/, omogoča mikroprolog programiranje vzorčno vodenih sistemov (pattern-directed systems). Taki sistemi temeljijo na samostojnih kosih informacij (pravila ali dejstva), ki veljajo za določeno problemsko področje. Interpreter sam izvaja pravila tako, da primerja, če se med podatki v podatkovni bazi pojavi ustrezen vzorec. Ravno ta zamisel, da lahko v sistem dodajamo (ali pa brišemo ali spreminjamo) pravila ali dejstva neodvisno od ostalega sistema, močno olajša programiranje še zlasti za najmlajše učence, saj je ob pravilnosti vseh vloženih pravil zajamčeno tudi pravilno delovanje celotnega sistema.

V mikroprologu običajno programiramo tako, da
1. definiramo dejstva, objekte in relacije med njimi
2. definiramo pravila, s katerimi določimo zakonitosti problemskega prostora
3. postavljamo razna vprašanja o vpisanih dejstvih in pravilih ter zaključkih sestavljenih pravil.

Tako način programiranja kot komunikacija sama ste bliže opisovanju v naravnem jeziku kot pri ostalih programskih jezikih. Na nekaj enostavnih uvodnih primerih si oglejmo izražanje v slovenščini, prologu in mikroprologu, oziroma

dodatku SIMPLE, ki ga dobimo hkrati s kaseto z mikroprologom za mikroračunalnik sinclair ZX.

```
Slovenščina: Jaka je oče od Andreja.
Prolog     : oce(jaka, andrej).
Mikroprolog: Jaka oce-od Andrej
       ali oce(Jaka Andrej)
       ali oce(jaka andrej)
```

Komentar: Pri prologu se konstante začnejo z malo začetnico, zato pišemo npr. "andrej". V mikroprologu lahko pišemo konstante z veliko ali malo. V *osnovnem prologu imamo prefiksno* notacijo, v mikroprologu (programu SIMPLE - glej dodatek 1) pa lahko izbiramo med prefiksno in infiksno notacijo za enega ali dva argumenta. Za več argumentov lahko izbiramo med prefiksno notacijo in infiksno z več argumenti združenimi v seznam.

```
Slovenščina: Janez ljubi Majdo.
Prolog     : ljubi(janez, majda).
Mikroprolog: Janez ljubi Majda
```

Komentar: V obeh prologih razumljivo ne moremo sklanjati, spregati itd., še zlasti ne v slovenščini. Zato moramo povsod pisati popolnoma enake oblike besed.

```
Slovenščina: Janez prodaja krompir.
             Janez je zelje.
             Janez prodaja paradiznik.
             Janez je krompir.
Prolog     : prodaja(janez, krompir).
             je(janez, zelje).
             prodaja(janez, paradiznik).
             je(janez, krompir).
Mikroprolog: Janez prodaja krompir
             Janez je zelje
             Janez prodaja paradiznik
             Janez je krompir

Slovenščina: Ali Janez je in prodaja zelje?
Prolog     : je(janez, zelje),
             prodaja(janez, zelje).
Odgovor    : YES
Mikroprolog: is( Janez je zelje and
                 Janez prodaja zelje)
Odgovor    : YES
```

Komentar: Konjunkcije (AND) delamo v prologu z vejicami, v SIMPLu z "and". Disjunkcije (OR) običajno delamo z več samostojnimi stavki.

```
Slovenščina: Kdo je in prodaja krompir?
Prolog     : je(X, krompir),
             prodaja(X, krompir).
Odgovor    : X = janez
Mikroprolog: which(X: X je krompir and
                      X prodaja krompir)
Odgovor    : Janez
```

Komentar: Spremenljivke so vezane znotraj enega stavka, to pomeni, da mora X znotraj stavka imeti natanko isto vrednost, dva X-a v dveh stavkih pa nimata neposredne povezave preko imena. Spremenljivke v mikroprologu so x, X, y, Y, z, Z, lahko pa jim dodamo še številko, npr.: x11.

```
Slovenščina: Janez pripoveduje:" Moja mama je
             rodila otroka, pa mi ni ne brat
             ne sestra. Kdo je to?"
Prolog     : mati(mama, janez).
             kdo(X) :- mati(mama, X),
                       not brat(janez, X),
                       not sestra(X, janez).
Odgovor    : X = janez
Mikroprolog: moja-mama je-mati-od Janez
             which(To je X:
                   moja-mama je-mati-od X and
                   not Janez je-brat-od X and
                   not X    je-sestra-od Janez.
Odgovor    : To je Janez.
```

Komentar: Pri takem preslikovanju iz naravnega jezika v katerikoli prolog je seveda treba upoštevati, da je v prologu zelo malo predprogramiranega znanja v obliki podprogramov. Zato moramo človeško znanje (angl. *common sense*) zakodirati za vsak primer posebej, npr. v zgornjem primeru moramo definirati relaciji "brat" in "sestra", čeprav program tokrat pravilno deluje tudi brez tega. V teh primerih smo prvič srečali negacijo - konstrukt "not".

3. Primerjava med prologom in mikroprologom

Eno zanimivih vprašanj o mikroprologu (programu SIMPLE) je, v kolikšni meri so dodatki in spremembe boljše kot v standardnih verzijah prologa. To bi nam povedalo, v kolikšni meri lahko pričakujemo nadaljne spremembe v razvoju logičnega programiranja. Odgovor na to je precej opurtunističen, saj spremembe le malenkostno spreminjajo osnovni koncept. Določeni dodatki kot možnost infiksnega zapisa so verjetno smiselni tudi za resno programiranje, za učenje pa so po mnenju avtorjev izredno koristni. Drugi dodatki kot možnost pisanja konstant z veliko začetnico so ugodni, vendar potegnejo za seboj težave pri pisanju spremenljivk (ni mnemoničnih spremenljivk - glej /8/). Tudi pri pisanju numeričnih izrazov na prvi pogled pridobimo v obrnljivosti in imamo tako eno izjemo manj, saj lahko numerični izraz obravnavamo kot običajno relacijo in lahko iščemo katerokoli spremenljivko. V običajnih prologih to ni mogoče, saj lahko računamo le v eno smer, npr. Y is X + 5, v mikroprologu pa bi zapisali SUM(X 5 Y) in bi lahko računali X npr. s SUM(X 5 7) ali Y s SUM(2 5 Y). Nekatere druge spremembe kot pisanje presledkov namesto vejic, pisanje "and" za konjunkcije itd. pa so bolj ali manj oblikovne narave.

Omenimo še nekatere dodatne lastnosti kot ukaze za delo z nizi, solidno realno aritmetiko (vsaj za jezike umetne inteligence), ukaze za delo z datotekami...

Čeprav so določene oblikovne razlike, pa lahko v mikroprologu naredimo praktično vse operacije kot v prologu, tako lahko uporabljamo tudi najmočnejše programske prijeme iz običajnega prologa celo na najmanjših mikroračunalnikih. Žal pa nekateri moduli skoraj ne pridejo v poštev na najmanjših računalnikih, recimo ukazi za sledenje izvajanju, saj na najmanjših mikroračunalnikih hitro zmanjka pomnilniškega prostora.

Tehnične karakteristike mikroprologa so prav tako med najboljšimi glede na ostale inačice. V enem stavku bi jih našteli z: *izredno majhen in hiter prevajalnik z nekaj opcijami* (osnovni mikroprolog, standardni prolog, SIMPLE...), modularna struktura, izdelana komunikacija s perifernimi napravami.

Modularno programiranje je močno izraženo. Večji program razbijemo na logično zaključene manjše dele in jih deklariramo za module. Med izvajanjem lahko izvajamo samo nekaj modulov. Ko potrebujemo kakšen modul, ki ga nimamo v centralnem pomnilniku, ga naložimo z ustrezne periferne enote (diskete, diska, ...) in ga ponovno odložimo, ko ga ne potrebujemo več. Tako lahko izvajamo precej obsežnejše programe, kot pa nam to omogoča pri mikroračunalnikih precej omejeni centralni pomnilnik.

Tudi komunikacija s perifernimi napravami je zelo učinkovita in enostavna, saj preko vmesnika RS232 dosegamo naprave od tiskalnikov do gibkih ali trdih diskov.

Literatura o mikroprologu je dokaj solidna, ponekod pa je nepotrebno prilagojena mlajšim učencem. Oglejmo si primer iz /4/ s strani 129:

```
(x y) sums-to z if SUM(x y z)
(x ! y) sums-to z if y sums-to X and SUM(X x z)
```

če dobro premislimo, bi bilo namesto prvega stavka bolje: () sums-to 0
saj je krajše, bolj univerzalno in bolj "čisto" logično razmišljanje. Take drobne ohlapnosti so nepomembne pri programiranju malih primerov, pri zahtevnejših programih pa so eden poglavitnih vzrokov za nezanesljivo in nepravilno delovanje, saj podobno kot gornji primer puščajo prostor za nepredvidene situacije (npr. napaka v primeru iskanja vsote enega elementa).

V celoti gledano pa so zbirke nalog med boljšimi.

4. Mikroprolog kot šolski jezik

Mikroprolog je zanimiv zlasti kot šolski jezik. Naštejmo osnovne značilnosti (prednosti) mikroprologa kot učnega jezika:

- perspektivnost

Zgodovina računalništva /8/ nas uči, da gre razvoj programskih jezikov v smeri približevanja ljudem. Od strojnega jezika preko zbirnega in visokih algoritmičnih jezikov pridemo do deklarativnih jezikov tipa prolog. V japonskem projektu 5. generacije računalnikov pa pride do revolucionarne spremembe, kjer je strojni jezik kar inačica proloqa, imenovan KLO (Kernel Language 0). Nad njim srečamo druge inačice, npr. ESP na nivoju makrozbirnika. Torej lahko pričakujemo, da bodo jeziki prihodnosti če že ne inačica proloqa, pa vsaj precej bolj podobni prologu kot npr. fortranu. Kakšen smisel je učenje že zdaj zastarelih jezikov (tipa fortran ali cobol), ki bodo še precej bolj zastareli (če ne cjlo izumrli), ko bodo učenci dokončali študij?

- razširjenost

Današnji pogoj za uporabnost programskega jezika je razširjenost na vseh, tudi najmanjših mikroračunalnikih. Mikroprolog je ne samo dosegljiv na hišnih računalnikih, ampak ima celo posebej prilagojene dodatke za začetnike, npr. program SIMPLE /3/.

- splošnost

V nasprotju z učenjem specializiranih področij znotraj računalništva naj bi bila osrednja tema učenja z mikrologom učenje LOGIČNEGA RAZMIŠLJANJA. Programski jeziki se menjajo, prihajajo programski paketi in sistemi, ki sploh ne zahtevajo znanja programiranja na nivoju programskih jezikov, vsako smiselno človeško opravilo pa je vezano na logiko. Z mikrologom se lahko učimo programirati kot s programskim jezikom, prav lahko pa se učimo tudi druge predmete, npr. matematiko, zgodovino itd. /4/. Ne glede na to, iz kakšnega področja so vaje, pa vedno podpirajo razvijanje logičnega mišljenja.

5. Kratek pregled učne snovi za najmlajše učence

5.1. vaje po učni snovi

Poglejmo nekaj vaj v mikroprologu (po vzoru iz /4/) najprej po učni tematiki:

MATEMATIKA:

```
x ima-povprečje y if x ima-vsoto z and
                     x dolg X and
                     deljeno(z X y)
```

Kar preberemo kot: povprečje od x je y, če je vsota elementov iz x enaka z in je x sestavljen iz X elementov in y = z/X.

Podprograme "ima-vsoto", "dolg" in "deljeno" moramo sami sprogramirati:

```
() ima-vsoto 0
(x ! y) ima-vsoto z if y ima-vsoto z1 and
                       SUM(z1 x z)
```

Komentar:
Vsota praznega seznama je 0.
Vsota seznama s prvim elementom x in ostankom seznama y je z pri čemer je y vsota z1 in z = z1 + x.

```
() dolg 0
(x ! y) dolg z if y dolg z1 and
                  SUM(z1 1 z)

deljeno(x y z) if TIMES(y z x)
```

Operacije nad množicami:

```
x presek (y z) if x isall(X: X ON y and X ON z)
x razlika (y z) if x
                   isall(X: X ON y and not X ON z)
x član X if x ON X
```

Komentar: Npr. presek: množica x je presek množic y in z, če je sestavljena iz elementov, ki so v y in z hkrati. Tu smo srečali konstrukt "isall", ki zgradi seznam iz elementov, ki ustrezajo logičnemu pogoju v oklepajih.

ZGODOVINA:

```
(začetek druge svetovne vojne) datum 1941
Informbiro datum 1948
(konec druge svetovne vojne) datum 1945
which(x se je zgodil leta 1941: x datum 1941)
(začetek druge svetovne vojne) se je zgodil
  leta 1941

which( Informbiro je bil leta X: Informbiro
  datum X)
Informbiro je bil leta 1948
```

```
Hitler je nacist
nacist hoce nadvlado
nacist hoce vojno
Učenec skuša odgovoriti na vprašanje, kaj hoče
Hitler:
which(x: Hitler hoce x)
No (more) answers
(prvo vprašanje ni rodilo sadov)

which(x: Hitler je x)
nacist
which(Hitler hoce x: nacist hoce x)
Hitler hoce nadvlado
Hitler hoce vojno
```

Z gradnjo obsežnih baz, ki vsebujejo veliko med seboj povezanih podatkov, lahko učenci povezujejo informacije med seboj in se tako učijo spoznavati povezave, ne samo gola dejstva. Poleg tega je enostavno pisati programe za simulacijo zgodovinskih dogajanj /4/. Ti programi so prav tako primerni za učenje zemljepisa, sociologije, spoznavanje narave, učenje kemije ali arheologije /9/.

UČENJE JEZIKOV

Za šalo prepišimo mikroprolog v slovenščino:

```
dodaj(x)   if add(x)
kateri(x)  if which(x)
en(x)      if one(x)
vsota(x y z) if SUM(x y z)
x element y  if x ON y
dodaj(Janez ljubi Micka)
kateri(x: x ljubi Micka)
Janez
```

Poskusimo še s prevajanjem med jeziki:

```
I          je jaz
you        je ti
sun        je sonce
see        je vidim
hollidays  je počitnice
sea        je morje
and        je in
()       angl-sl ()
(x ! X) angl-sl (y ! Y) if x je y and
                           X angl-sl Y

which(x: x angl-sl (jaz vidim sonce))
I see sun.

which(x: (see hollidays and sun and sea)
         angl-sl x)
vidim počitnice in sonce in morje
```

Komentar: zaradi bogastva slovničnih oblik v slovanskih ali germanskih jezikih takšni primerčki niso tako uspešni kot npr. v angleščini ali francoščini.

Seznam vaj iz računalniškega vrtca je v dodatku 2.

Po mnenju avtorjev je ena večjih prednosti poučevanja mikroprologa ravno v obsežni zbirki zanimivih vaj, ki so hkrati zanimive in poučne. Tako se učimo postavljati vprašanja (query) v detektivskih nalogah, npr. iščemo Jacka Razparača v podatkih o osumljencih, izbiramo miss sveta izmed kandidatk itd. Po drugi strani pa je zaporedje učne snovi v /4/ nekoliko prepočasi tempirano za sposobnejše učence, saj prepočasi pride do praktičnega programiranja.

Ena večjih pomanjkljivosti predvsem za mlajše učence je pomanjkanje specializiranih konstruktov za risanje. Tako v basicu kot v logoju večina začetnega poučevanja temelji na grafiki, v mikroprologu pa kljub podatkom iz literature nismo uspeli dobiti posebne grafike na sinclairju ZX in učnih materialov zanjo. Seveda pa lahko sami napišemo svoje grafične rutine s pomočjo izpisovanja krmilnih znakov.

6. Izkušnje s poučevanjem

V /4/ Ennals predstavlja šolske materiale za britanski projekt (največ v obliki seznama vaj). Učenje je potekalo v običajnih šolskih razredih z enim računalnikom in velikim zaslonom. V razredu je bilo 26 učencev v starosti 10-11 let. Pouk je potekal v okviru rednega pouka in je trajal 3 uri 20 minut tedensko. Učenci so imeli popoldne možnost učenja na šolskih računalnikih.

V Jugoslaviji je potekalo učenje mikroprologa na Institutu "Jožef Stefan" v okviru izobraževanja zaposlenih na Institutu in v okviru Računalniškega vrtca za učence v starosti 10-14 let /10/. Pouk štirinajstih mladih učencev je potekal v 18 učnih sklopih vsakič po dve uri in 15 minut, izobraževanje petnajstih zaposlenih pa v 8 sklopih po 2 uri. Učenje je potekalo v računalniških učilnicah z računalniki sinclair ZX z 48 K in monitorjem na vsaki mizi.

Težave pri poučevanju:
Težave pri praktičnem poučevanju so izvirale iz več vzrokov. Nekaj težav je izviralo iz opreme, tako npr. je bilo tipkanje na računalnikih sinclair z gumijastimi tipkami izredno zamudno. Pri najmlajših učencih je bilo opaziti tudi probleme z motiviranostjo, saj bi se včasih rajši igrali kakšne igrice. Pri izobraževanju računalniško podkovanih kadrov je bilo opazno razumevanje tudi na globjem proceduralnem nivoju, medtem ko so učenci v računalniškem vrtcu ostali na nivoju deklarativno-intuitivnega razumevanja rekurzije in nekako niso uspeli preskočiti tega praga.

Rezultati testiranja:
Na predlog dr. M. Ribariča (idejni pobudnik in vodja celotnega projekta računalniških vrtcev) smo poskusili ugotoviti, ali je osnovni cilj učenja z mikroprologom uspel ali ne. Ker je osnovni cilj učenje logičnega razmišljanja, smo izbrali naloge - logične uganke - iz /11/ in primerjali rezultate testov ob začetku in koncu tečaja iz mikroprologa v računalniškem vrtcu. Učenci so reševali pismene teste načeloma v skupinah po dva, tako kot je običajen način dela v vrtcu. Obakrat so imeli reševalci na voljo 25 minut. V tabelah vidimo rezultate prvega in drugega testiranja. Številke na levem robu pomenijo številke nalog iz /11/, številke na zgornjem robu zaporedno število izdelka, enice znotraj tabele pa uspešno rešene naloge.

Rezultati prvega testiranja:

	1	2	3	4	5	6	7	8
26	1	1	1		1	1	1	1
27								
28	1					1		
34						1	1	
37				1		1	1	
41								

3. izdelki niso dobili nobene točke
Skupno 14 rešenih nalog, 11 izdelkov.
Število rešenih nalog na izdelek: 14/11 = 1.27.

Rezultati drugega testiranja:

	1	2	3	4	5	6	7	8	9
29	1	1	1	1	1	1	1	1	
30		1				1		1	
32									
36			1	1	1	1	1		1
38	1	1			1			1	1
39									

Skupno 22 rešenih nalog, 9 izdelkov.
Število rešenih nalog na izdelek: 22/9 = 2.44.

Rezultati testiranja nakazujejo pozitiven vpliv učenja mikroprologa na logično razmišljanje, saj je število uspešno rešenih nalog na izdelek narastlo z 1.27 na 2.44. Vendar ti rezultati nimajo neoporečne znanstvene teže, saj je izstopilo nekaj učencev, ki je imelo največ težav, tudi zasedba dvojic ni bila enaka med prvim in drugim testiranjem. Zaradi tega imajo rezultati le informativno vrednost. V obeh testiranjih so bile izbrane približno enako težke naloge, tako da je bila ena očitno pretežka, za ostale pa smo upali, da jih bodo lahko rešili. Glede na dosežene rezultate pa so bile nekatere naloge le pretežke, zato bi bilo za večjo objektivnost rezultatov smiselno ponoviti testiranje z večjim številom lažjih nalog. Učenci so se večinoma pritoževali, da so bile drugič naloge težje kot prvič. Kljub vsemu pa testi kažejo na to, da obstaja močan pozitiven učinek učenja mikroprologa na logično razmišljanje, saj se je po vsem sodeč močno dvignilo povprečje teh učencev.

Poglejmo si število dvojic učencev, ki so rešili po 4, 3, 2 in 1 nalogo v obeh merjenjih:

Nalog	Prvič	Drugič
4	1	0
3	1	4
2	1	5
1	5	0
0	3	0

7. Diskusija

Osebne izkušnje: Pri prehodu s prologa na mikroprolog so določene težave, čez nekaj časa pa se pokaže, da je svoboda izražanja nekoliko večja v mikroprologu, oblika zapisa pa nekoliko enostavnejša. Tudi poučevanje v mikroprologu je bolj učinkovito. Kljub pogostim trditvam, da je prolog enostaven za učenje, se avtorja ne strinjata povsem s tem. Učenje prologa je predvsem učinkovito in hitro, to pomeni, da lahko učenci kmalu začnejo pisati zahtevne programe. Pred tem pa morajo narediti zahteven miselen preskok. Ta preskok je dokaj enostaven za tehnično izobražene kadre, npr. matematike ali za usmeritve, ki podpirajo logično razmišljanje. Pri drugih izobrazbah pa se pokažejo težave. Posebno rekurzija in proceduralni pomen prologa delajo veliko težav tudi računalniško izobraženim kadrom, ki so navajeni programirati v starejših jezikih brez rekurzije (cobol, fortran). Z mikroprologom je učenje potekalo opazno hitreje in bolj uspešno.

Mikroprolog kot programski jezik: S stališča profesionalnega programiranja mikroprolog ne prinaša velikih sprememb glede na ustaljene inačice kljub temu, da ima nekaj dodatkov, ki nekoliko polepšajo logično celovitost jezika. Ti dodatki so včasih celo nekoliko dolgovezni s stališča izurjenega programerja, zato pa so veliko bolj koristni za učenje programiranja, saj ne samo olajšujejo učenje, ampak omogočajo še nekoliko višji - bolj deklarativni nivo in s tem dajejo možnost večjega poudarka učenju logičnega razmišljanja. Druga misel, ki se ponuja, pa je, da je osnovni mehanizem prologa zelo močno orodje, ki ga inačice niti v peti generaciji ne bodo bistveno spremenile vsaj z logičnega stališča. Tehnično-uporabniško gledano pa je mikroprolog veliko bliže jezikom za profesionalno programiranje, saj ima izdelano komunikacijo z zunanjimi napravami, omogoča modularno programiranje, prevajalnik sam pa je med najmanjšimi in najhitrejšimi.

Mikroprolog kot učni jezik: Mikroprolog je verjetno še nekoliko primernejši za učenje glede na ostale inačice, še posebej pokaže prednosti pri poučevanju učencev z ne pretirano predizobrazbo. Celo primerjava s programskim jezikom logo /4/ pokaže kar nekaj prednosti za učence, ki so stari nad 10 let. Le za mlajše učence pod 10 let je mikroprolog pretežak. Poglavitne prednosti učenja z mikroprologom so razvijanje logičnega mišljenja, razvijanje kreativnega in samostojnega razmišljanja in učenja programerskih principov pete generacije računalnikov.

8. Literatura

1. Kowalski R.: Predicate logic as programming language, Proc. IFIP 74 Conf., North-Holland 1974

2. Kowalski R.: Algorithm = Logic + Control, CACM, Vol. 22, No.7, 572-595, 1979

3. Clark K.L., McCabe F.G., Ennals J.R.: A Micro-PROLOG Primer, Logic Programming Associates, 1983

4. Ennals R.: Beginning Micro-PROLOG, Ellis Horwood Ltd., London, 1904

5. McCabe F.G., Clark K.L., Steel B.D.: Micro-PROLOG Programmers Reference Manual, Logic Programming Associates, 1984

6. Bratko I., Gams M.: Prolog: osnove in principi strukturiranja podatkov, Informatica 4/1980, str. 40-46, 1980

7. Bratko I.: Expert Systems and Prolog, Pergamon Infotech State of the Art Report: Supercomputer Systems Technology, F. Sumners (ed.) Series 10, No. 6, Pergamon Infotech Ltd., 1982

8. Gams M.: Osnove dobrega programiranja, Cankarjeva založba, 1985

9. Ennals R.: Micro-PROLOG across the Curriculum, Collected papers 1982-1984, Research Report DoC 84/17, Imperial College, 1984

10. Gams M.: Mikroprolog, seminarski materiali, str. 50, 1985

11. Smullyan R.M.: Alica v deželi ugank, Državna založba Slovenije, Ljubljana 1984

DODATEK 1

4. Seznam osnovnih ukazov za delo s programom "SIMPLE"

Opomba: Nekateri ukazi so vezani na sinclair ZX:

"caps shift" "Z" "break space" istočasno (prekinitev izvajanja)
"caps shift" "a" istočasno (no scrool)

accept -za vpisovanje več stavkov z isto relacijo.
Primer:
```
accept ima-rad
ima-rad.(Lojze Mojca)
ima-rad.(bbb ccc)
ima-rad.end
```

add -za dodajanje stavkov bazo podatkov
Primer:
```
add(Jure ima-rad Meri)
```
(doda stavek v bazo)
```
add 2 (Mira ima-rad Miha)
```
(doda stavek na tretje mesto med stavke relacije "ima-rad")

delete -zbriše stavek dane relacije z ustrezno zaporedno številko
Primer:
```
delete ima-rad 1
```

edit -editiranje in popravljanje stavkov programa.
Splošna oblika:
```
edit ime-relacije zaporedna-številka-stavka
```
Primer:
```
edit ima-rad 1
1 (Lojze ima-rad Mojca)
```
Zdaj se lahko premikamo po stavku s tipkami/puščicami in ga sproti popravljamo.

kill -uniči vse stavke za dano relacijo ali cel program.
Primer:
```
kill ima-rad
```
(zbriše vse stavke relacije ima-rad)
```
kill all
```
(zbriše ves program)

list -izpiše vse stavke dane relacije ali cel program.
Primer:
```
list ima-rad
```

(izpiše vse stavke relacije "ima-rad")
list all
(izpiše cel program)

which -išče vse možne odgovore.
all -sinonim za "which".
one -daje samo en odgovor naenkrat.
is -preveri pravilnost in odgovarja z "YES" ali "NO".
isall -grajenje seznamov.
Primer:
which(x:x isall (y:y ima-rad Mezi))
(izpiše se seznam vseh, ki imajo radi Mezija)
not -negiranje pogojev.
load -naloži program v računalnik z kasetofona. Splošna oblika:
load ime-programa
save -shrani celotni program iz računalnika na kasetofon. Splošna oblika:
save ime-programa
reserved -za iskanje rezerviranih besed.
Primer:
which(x:x reserved)
(izpiše vse rezervirane besede)
defined -če hočemo izvedeti, katere relacije so definirane.
Primer:
which(x:x defined)
(izpiše imena vseh relacij, ki so definirane v našem programu)
ON -članstvo v seznamu. Je izpolnjen, ko je objekt član seznama.
Primer:
is(5 ON (1 2 3 4 5))
YES
CONS -doda objekt na prvo mesto v danem seznamu.
Primer:
which(x:CONS(3 (4 5 6)))
(3 4 5 6)
No (more) answers
APPEND -združi dva seznama v tretjega.
Primer:
which(x:APPEND((1 2 3 4)(5 6 7 8) x))
(1 2 3 4 5 6 7 8)
No (more) answers

DODATEK 2

Tu so zbirno navedene vaje tečaja iz mikroprologa, ki je potekal v okviru računalniškega vrtca /10/:

0. seznanjanje z računalnikom, seznanjanje z osnovnimi ukazi tipa "add", "list","accept", "delete", "kill", "edit"
1. enostavni stavki, opisovanje podatkov/stanj (ukaz "add", vaje na nivoju "Pavle ima-rad Meri", vaje na nivoju opisovanja sedežnega reda učencev v učilnici)
2. postavljanje vprašanj (ukaz "is", enostavne vaje)
3. pretvarjanje stavkov iz slovenščine v mikroprolog (trditve, vprašanja, vaje v pretvarjanju drugih jezikov)
4. uporaba spremenljivk in ukaza "which" (vaje na nivoju enostavnih družinskih relacij)
5. sestavljena vprašanja s konstruktom "is" (tvorjenje logičnih konjunkcij z uporabo konstrukta "and")
6. sestavljena vprašanja s konstruktom "which" (tvorjenje logičnih konjunkcij z uporabo konstrukta "and")
7. vaje - izpraševanje o vpisanih podatkih - kaj je nad, višje..
8. enostavna aritmetika (LESS, SUM, TIMES in uporaba teh konstruktov za računanje)
9. splošna oblika stavka "which"
10. pravila (A if B and C and ...)
11. pravila z uporabo spremenljivk (vaje, npr. detektivske naloge tipa iskanje Jacka Razparača)
12. predstavitev enostavnih pravil z diagrami/mrežami (vaje iz družinskih relacij, npr. Jože je-oče-od Lojze)
13. predstavitev pravil s spremenljivkami v obliki semantičnih mrež (vaje iz družinskih relacij, npr. x je-oče-od y if y je-sin-od x)
14. prehod na zapletene oblike pravil, ki se med seboj povezujejo (npr. stric je brat od očeta ali ...)
15. aritmetične in logične relacije (npr. kdo je višji ali starejši od koga)
16. seznami (npr. pretvarjanje iz slovenščine v mikroprolog)
17. dostop do elementov seznama (uporaba vzorcev in operatorja za ločevanje glave od seznama)
18. vaje s seznami
19. konstrukt "one"
20. vaje s seznami in bazami podatkov (npr. izbiranje miss sveta, iskanje asociacij, zamenjevanje besed ...)
21. rekurzija (vaje na nivoju družinskih relacij - npr. "prednik")
22. članstvo v seznamu (konstrukt "ON" in njegova definicija)
23. vaje s seznami (programi za dolžino seznamov, združevanje seznamov)
24. negacija (uporaba konstrukta "not")
25. enomestne relacije (npr. Nika bere)
26. grajenje seznamov (uporaba konstrukta "isall")
27. naloge s seznami (seštevanje elementov seznamov, množenje elementov seznamov)
28. nekoliko težje naloge s seznami (kupovanje najprimernejšega avtomobila , iskanje raznih povprečij)
29. grajenje verig (npr. iskanje verig potomcev v družinskih drevesih)
30. vaje (npr. potapljanje ladjic, ugibanje skritih informacij, iskanje "kdo je najvišji" itd.)
31. sistemski ukazi

SVEUČILIŠNI RAČUNSKI CENTAR
UNIVERSITY COMPUTING CENTRE

POZIV NA SUDJELOVANJE

VIII MEĐUNARODNI SIMPOZIJ **"KOMPJUTER NA SVEUČILIŠTU"**
SVIBANJ 12. – 15. 1986.

Mjesto:
DUBROVNIK/CAVTAT, HOTEL CROATIA

Organizator:
SVEUČILIŠNI RAČUNSKI CENTAR, ZAGREB

Teme:

- INFORMATIČKA IZOBRAZBA
- RAČUNSKI SISTEMI I MREŽE RAČUNALA
- OSOBNA RAČUNALA
- SOFTVERSKO INŽINJERSTVO
- INFORMACIJSKI SISTEMI I BAZE PODATAKA
- STATISTIKA I STATISTIČKI SOFTVER
- ANALIZA PODATAKA
- MODELIRANJE, SIMULACIJA I OPTIMIZACIJA
- DIZAJN I PROIZVODNJA POMOĆU RAČUNALA (CAD/CAM)
- UMJETNA INTELIGENCIJA I EKSPERTNI SISTEMI
- PRIMJENA INFORMATIČKIH SREDSTAVA I METODA U PRIRODNIM I DRUŠTVENIM ZNANOSTIMA
- DRUŠTVENI I PRAVNI ASPEKTI INFORMATIKE

Rok za sažetke:
15. PROSINAC, 1985.

Rok za radove:
15. OŽUJAK, 1986.

Obavijest o prihvaćanju:
1. VELJAČA, 1986.

Struktura Simpozija:
PREDAVANJA, POSTER SEKCIJE, PREZENTACIJE HARDVERA I SOFTVERA, PANEL DISKUSIJE, IZLOŽBE

Informacije:
Branka Radić
Sekretarica Simpozija

SVEUČILIŠNI RAČUNSKI CENTAR, 41000 Zagreb, Engelsova b.b., Jugoslavija,
Tel.: 041/510 099, Tlx.: 21871