МАТЕМАТИЧКИ ИНСТИТУТ

ЗБОРНИК РАДОВА

НОВА СЕРИЈА
КЊИГА 5 (13)

РАЧУНАРСТВО
Број 2, 1993

COLLECTION OF PAPERS IN COMPUTING SCIENCE

Београд
1993

RAČUNARSTVO
Broj 2, 1993

YU ISSN 0353-8907

Periodična publikacija Matematičkog instituta, Knez Mihajlova 35,
11 001 Beograd

Collection of papers in Computer Science, a periodic publication of the Institute of Mathematics, Knez Mihajlova 35, 11 001 Belgrade, Yugoslavia

UDK 519-68

SADRŽAJ

Marica D. Prešić
Formalne gramatike 3

Cvetana Krstev
Algoritmi za sravnjivanje niskovnih obrazaca 25

Goran Gogić
Kurepina hipoteza o levom faktorijelu 41

Slaviša Prešić i Pavle Blagojević
Sverel—prološki program (objedinjavanje svih relacija) 47

Tanja Pulević
Paralelizacija rada generatora pseudo-slučajnih nizova bitova (GPSN) i uopštenog generatora pseudo-slučajnih brojeva (CFSR) 51

Vladimir Miličić
Propov model i generisanje narodnih pripovedaka pomoću računara 63

Boško Damjanović
Jedan algoritam za određivanje najvećeg zajedničkog delioca polinoma 67

Nedeljko Parezanović
Metodički aspekti opisa semantike programskih jezika 81

RAČUNARSTVO
Sveska 1, Broj 2, 1992

UDK 519.681

FORMALNE GRAMATIKE[1]

MARICA D. PREŠIĆ

Razvitak lingvisike, tu prvenstveno mislimo na tzv. opštu lingvistiku koja se bavi opštim jezičkim istraživanjima zajedničkim za sve jezike, poklapa se u glavnim crtama sa razvitkom (matematičke) logike: burno u antičko doba, sa visokim i vekovima neprevaziđenim dometima, lagano i veoma usporeno sve do XIX veka; u tom su periodu uglavnom razrađivane antičke ideje. XIX vek, međutim, nije bio vek opšte lingvistike, već vek posebnih lingvističkih istraživanja u koja spadaju komparativne gramatike, biološki naturalizam, "humboltizam", psihologizam i dr.

Bura nastaje ponovo u XX veku: nagli razvitak matematičke logike bio je priprema i osnova naglog razvitka opšte lingvistike, koji počinje 60tih godina [Istorijskim datumom se smatra objavljivanje knjige "Syntactic Structures" Noama Čomskog (Chomsky) godine 1957]. Oslanjajući se na tradicionalno gledište racionalističkih filozofa sedamnaestog i osamnaestog veka po kome "strukturu jezika određuje struktura ljudskog uma i da univerzalnost izvesnih osobina svojstvenih jeziku svedoči da je bar taj deo ljudske prirode zajednički svim članovima vrste", a budući da je na univerzitetu dobio solidno znanje kako iz lingvistike, tako i iz filozofije i matematike (posebno se to odnosi na fundamentaltalna dostignuća u oblastima rekurzivnih funkcija i teorije algoritama), Čomski, nastavljajući ideje svog učitelja, "blumfildovca" Zeliga Harisa, postavlja sebi za cilj zasnivanje sintakse u obliku takozvane formalne teorije.

Napomenimo odmah, da je semantika bila kod Čomskog potpuno zanemarena u prvim istraživanjima [što je donekle kasnije izmenjeno], nalazila se u podređenom položaju u odnosu na sintaksu i nije, po mišljenju Čomskog, u pravom smislu pripadala lingvistici.

Ukratko o formalnoj teoriji: Pojam, u današnjem obliku, dugujemo Hilbertu, a koreni su mu u pojmu aksiomatske teorije, koji potice iz antičkog doba [Euklid je za to najveći zaslužnik]. Formalnu teoriju određuju izvesni polazni znaci koji čine

[1]Rad finansiran od Fonda za nauku Republike Srbije preko Matematičkog instituta, projekat 0401A.

takozvani alfabet, skup izvesnih reči nad tim alfabetom koje se nazivaju formulama [za koje se pretpostavlja jedno potpuno određeno pravilo prepoznavanja], skup aksioma [to su neke formule izdvojene kao polazne], i najzad pravila izvođenja koja su oblika:

(1) Iz formula $F_1, F_2, \ldots, F_n$ se izvodi formula F

Pravila izvođenja su, ustvari, vrsta relacija među formulama. Formalne teorije su pravljene radi strogog zasnivanja najvažnijeg matematičkog pojma: teoreme. Aksiome su, ustvari, polazne teoreme, dok se sve druge druge teoreme dokazuju na osnovu aksioma primenom pravila izvođenja. Sam dokaz je jedan konačan niz formula koje su ili aksiome ili slede iz nekih prethodnih formula tog niza primenom izvesnog pravila izvođenja. U opštem slučaju dokazi teorema mogu biti veoma dugački i komplikovani, teorema može imati više različitih dokaza, a za datu formulu nije uvek lako, a najčeće je i nemoguće, odgovoriti da li ona jeste ili nije teorema neke formalne teorije. Međutim, za neke posebne vrste formalnih teorija rešavanje takvih pitanja je moguće i za to su u četvrtoj i petoj deceniji ovog veka razvijeni moćni logički aparati. Tu se prvenstveno misli na oblasti koje su se razvile u vezi sa naporima da se strogo zasnuje pojam algoritma (postupka), a to su: teorija rekurzivnih funkcija, Tjuringove mašine, kombinatorni sistemi, teorija algoritama. Mada su sve te teorije u određenom smislu međusobno ekvivalentne, Čomski se opredelio za kombinatorne sisteme [koje je Post razvio 1936. godine], budući da su oni bili najbliži idejama koje je imao u vezi sa strogim logičkim zasnivanjem sintakse prirodnih jezika.

Ukratko o kombinatornim sistemima: To je posebna vrsta formalne teorije nad konačnim alfabetom kod koje su sve reči (nad uočenim alfabetom) formule, konačno mnogo reči je uzeto za aksiome, dok su pravila izvođenja, a njih je takođe konačno mnogo, sva posebnog oblika:

(2) $$A_0X_1A_1X_2\ldots,X_rA_r \longrightarrow B_0X_{i_1}B_1X_{i_2}\ldots,X_{i_s}B_s$$

Tu su $A_0, A_1, \ldots, A_r, B_0, B_1, \ldots, B_s$ date, fiksirane reči, dok su $X_0, X_1, \ldots, X_r$ proizvoljne reči ($X_{i_1}, X_{i_2}, \ldots, X_{i_s}$ su neke od $X_1, X_2, \ldots, X_r$).

[Pravilo (2) je primer tzv. shema-pravila; budući da su $X_1, X_2, \ldots, X_r$ proizvoljne reči nad uočenim alfabetom, a njih je beskonačno mnogo, to je (2) zajednički zapis za beskonačno mnogo posebnih pravila izvođenja, otkuda i naziv shema-pravilo.]

Pravila prethodne vrste se, inače, nazivaju zamenama (ili produkcijama). [Umesto reči "Iz ... se izvodi ... " stoji strelica $\longrightarrow$ a samo pravilo $X \longrightarrow Y$ se čita "X se zamenjuje sa Y"]. Čest je slučaj da je polazni alfabet A kombinatornog sistema podeljen na dva međusobno razdvojena podskupa T (završni, terminalni) i N (pomoćni, neterminalni), pri čemu pomoćni simboli igraju, ustvari, ulogu promenljivih. Evo nekoliko jednostavnih primera kombinatornih sistema:

PRIMER 1.. *Alfabet je $\{a, b\}$, aksiome su #, a, b (# je oznaka za praznu reč), zamene su shema-zamene:*

$$X \longrightarrow aXa, \quad X \longrightarrow bXb \qquad (X \text{ je proizvoljna reč})$$

Neke teoreme su na primer:

#, a, b, aa, bb, aba, bab, abba, baba, ababa, abbba, babab, baaab, bbabb, abbabba.

Nije teško dokazati (indukcijom po dužini dokaza) da su sve teoreme simetrične reči (takozvani palindromi).

PRIMER 2.. *Alfabet je $\{a, b\}$, aksiome su ab, abab, ababab, abababab, zamena je: $aXbY \longrightarrow XbY$ (X, Y su ma koje reči, dakle ponovo shema-zamena). Teoreme su upravo ove reči: ba, baba, bababa, babababa. Recimo, teorema bababa ima tri dokaza i svi su oni dužine dva: U prvom imamo $X = \#$, $Y = abab$, jednom primenom zamene neposredno dobijamo bababa. U drugom dokazu $X = ba$, $Y = ab$, a u trećem $X = baba$, $Y = \#$ i opet je dovoljno, u oba slučaja, primeniti zamenu samo jedanput. Očigledno je takođe da se iz dobijene teoreme bababa ne može izvesti nijedna nova, budući da ta reč počinje sa b. Što se tiče teorema ba, baba, bababa, one imaju po redu: jedan, dva, četiri dokaza, a iz razloga sličnog onom maločas iz njih se ne može izvesti nijedna nova teorema pa je ukupnost svih teorema u ovom primeru upravo: ba, baba, bababa, babababa.*

Navedeni primer se može uopštiti uzimanjem za aksiome ma koliko (ali konačno mnogo) reči oblika $(ab)^n$ [X^n je uobičajena oznaka za stepen reči X dobijen dopisivanjem X n puta. Tu je n ma koji dati prirodan broj uključujući i nulu, pri čemu se X^0 definiše kao #.] Teoreme će tada biti sve odgovarajuće reči oblika $(ab)^n$ a svaka takva teorema će imati ukupno n dokaza (dužine dva).

PRIMER 3.. *Alfabet je $\{a, b, S, (,)\}$, završni simboli su a, b, (,), pomoćni simbol je S, aksioma je takođe S, a zamene su:*

$$XSY \longrightarrow X(S*S)Y, \qquad XSY \longrightarrow XaY, \qquad XSY \longrightarrow XbY$$
$$(X, Y \text{ su ma koje reči})$$

Neke teoreme su, na primer:

$$S,\ (S*S),\ (S*(S*S)),\ ((S*S)*(S*(S*S))),$$
$$(S*(a*b)),\ ((a*S)*(b*(S*a))),$$
$$a,\ b,\ (a*b),\ (b*a),\ (a*(a*b)),\ (b*(b*b)),\ ((a*b)*(b*(a*b)))$$

Teoreme u trećem redu su posebne vrste, takozvane završne, budući da u njima učestvuju jedino završni simboli. Kao što se vidi to su ustvari izrazi obrazovani od a, b, $*$ ($*$ je operacijski znak dužine dva). Indukcijom se dokazuje da su završne teoreme upravo svi takvi izrazi.

Kombinatorni sistem u prethodnom primeru je posebne vrste. Sve njegove zamene su oblika:

(3) $$XAY \longrightarrow XBY \qquad (X, Y \text{ su proizvoljne reči})$$

To je primer takozvanog polutuovskog sistema, pri čemu se zamene (3) obično zapisuju bez proizvoljnih reči X, Y, tj. u obliku:

(4) $$A \longrightarrow B$$

i podrazumeva se da **primena** takve zamene znači sledeće: U proizvoljnoj reči W (nad uočenim alfabetom) podreč A, i to bilo koje izabrano njeno pojavljivanje u W, zamenjuje se sa B. Tako, u prethodnom primeru teorema $(S * (S * S))$ se dokazuje na osnovu već dokazane teoreme $(S * S)$ zamenom drugog pojavljivanja slova S sa $(S * S)$, tj. primenom zamene $XSY \longrightarrow X(S*S)Y$, ili drugačije: primenom zamene $S \longrightarrow (S * S)$, ukoliko koristimo skraćeno pisanje (4) [X, Y su po redu: ($S*$, odnosno)].

PRIMER 4. *Polutuovski sistem je određen uslovima: Završni alfabet je $\{a, b, c\}$, pomoćni je $\{S, T\}$, aksioma je S, zamene su: $S \longrightarrow aT$, $S \longrightarrow c$, $T \longrightarrow Sb$. Uzastopnom primenom prve i treće zamene izvode se redom teoreme:*

$$S,\ aT,\ aSb,\ aaTb,\ aaSb,\ aaaTbb,$$
$$aaaaSbbb,\ aaaaTbbb,\ aaaaSbbbb,\ aaaaaTbbbb, \ldots$$

iz kojih, primenom pravila $S \longrightarrow c$ slede završne teoreme:

$$c,\ acb,\ aacbb,\ aaacbbb,\ aaaacbbbb, \qquad \textit{odnosno uopšte}$$
$$a^n c b^n \qquad (n = 0, 1, 2, 3, \ldots)$$

Najzad još jedan polutuovski sistem čije se teoreme ne opisuju tako očigledno kao što je to bio slučaj u prethodnim primerima.

PRIMER 5.. *Završni alfabet je $\{a, b, c\}$, aksioma je S (to je ujedno i jedini pomoćni simbol), zamene su: $S \longrightarrow aSb$, $S \longrightarrow c$, $aSb \longrightarrow bSa$.*

Polutuovski sistemi u trećem i četvrtom primeru bili su posebne vrste, koju preciziramo narednom definicijom.

DEFINICIJA 1. *Polutuovski sistem čiji je alfabet A pode-ljen na razdvojene podskupove T, N (završnih i pomoćnih simbola) i koji ima samo jednu aksiomu, obično je to S, je takozvana (formalna) gramatika ili gramatika Čomskog. Skup svih završnih teorema čini takozvani jezik generisan formalnom gramatikom.*

[Prethodna definicija gramatike je generativna, što znači da se sve njene teoreme izvode, generišu iz jedne jedine aksiome S. Postoji i drugačiji pristup

gramatikama, dualan prethodnom, tzv. analitički. Neka reč je teorema analitičke gramatike ukoliko se iz nje (iz te reči) može izvesti aksioma S.]

Najznačajnija vrsta formalnih gramatika su kontekstno slobodne gramatike (što predstavlja nakaradan ali odomaćen prevod engleskog naziva context-free grammars), kod kojih su sve zamene oblika:

(5) $$N \longrightarrow B \qquad (N \text{ je pomoćni simbol, } B \text{ je reč iz } A^*)$$

[A^* je uobičajena oznaka za skup svih reči nad alfabetom A, uključujući i praznu reč #.]

Napomenimo samo da pored kontekstno slobodnih gramatika postoji još nekoliko drugih vrsta gramatika (podelu je načinio sam Čomski), kao što su kontekstno osetljive, čije su zamene oblika:

(6) $$CND \longrightarrow CBD \qquad (N \text{ je pomoćni simbol, } C, B, D \text{ su iz } A^*, B \text{ nije } \#)$$

zatim regularne gramatike, normalne i druge. Primenu u prirodnim jezicima posebno su našle kontekstno slobodne gramatike.

[Pitanje da li su prirodni jezici kontekstno slobodni ili nisu još uvek je otvoreno. Sam Čomski je na početku svojih istraživanja izjavljivao da ne zna odgovor a kasnije je počeo da zastupa stanovište da je odgovor negativan, što je čitav niz godina postalo široko prihvaćeno. Poslednjih godina, međutim, to gledište je bitno poljuljano, jer se ispostavilo da su svi do sada objavljeni dokazi o kontekstnoj osetljivosti prirodnih jezika uglavnom vrlo diskutabilni a često i pogrešni. Kako sada stvari stoje još uvek nema valjanog dokaza ni da su prirodni jezici kontekstno slobodni niti da su kontekstno osetljivi.]

Pre nego što pređemo na detaljije izlaganje o primeni gramatika Čomskog na prirodne jezike, vratimo se ponovo, nakratko, na opšti slučaj kombinatornih sistema. U čemu je njihova važnost i značaj u matematici? Već smo pominjali teoriju rekurzivnih funkcija i teoriju algoritama. U istu grupu teorija spadaju i Tjuringove mašine, URM-mašine (unlimited register machine), kombinatorni sistemi. Sve su se te teorije razvile u vezi sa jednin jedinim ciljem: da se strogo, matematički zasnuje intuitivni pojam algoritma (postupka).

[Pod tim pojmom se podrazumevaju kako raznorazni tehnološki postupci, tako i , na primer, postupak kucanja teksta pisaćom mašinom, kao i matematički postupci množenja dva prirodna broja ili određivanja njihovog najvećeg zajedničkog delitelja. Glavna karakteristika svakog takvog postupka jeste da se on obavlja u konačno koraka i da se prelazak sa jednog na drugi korak obavlja po tačno utvrđenim pravilima—uputstvima, kojih takođe ima konačno mnogo.]

U svakoj od tih teorija centralni pojmovi su izračunljivost (funkcija je izračunljiva ukoliko postoji postupak kojim se za dati original, u oblasti definisanosti, određuje njegova slika), nabrojivost (skup je nabrojiv ukoliko postoji postupak nabrajanja njegovih elemenata), odlučivost (problem je odlučiv ukoliko postoji

postupak koji u svakom konkretnom slučaju problema daje odgovor **da** ili **ne**). Pokazalo se da su pristupi svakom od tih pojmova, razvijani uporedo u pomenutim teorijama, svi međusobno ekvivalentni, što predstavlja jednu od potvrda Čerčove teze po kojoj:

Izračunljivost, definisana na ma koji od međusobno ekvivalentnih načina, pokriva intuitivni pojam algoritma.

Pomenimo još da se teorijski razvitak pojma algoritma poklapa sa tehničkim dostignućima u oblasti elektronskih računskih mašina. Razlog za to je veoma prirodan: mašina može obavljati samo one radnje za koje postoji algoritmi.

Kontekstno slobodne gramatike, kojima se sada vraćamo, karakteriše posebna jednostavnost u postupku dokazivanja teorema. Budući da svaka takva gramatika ima samo jednu aksiomu, S na primer, to svaki njen dokaz počinje sa S. Dalje, sve njene zamene su oblika (5) čijom primenom se dužina teorema ne smanjuje (jer se slovo, tj. pomoćni znak N zamenjuje rečju B čija dužina je veća ili jednaka 1). Kako je svaka reč konačna, to u svakoj već dokazanoj teoremi može učestvovati najviše konačno mnogo pomoćnih znakova. Otuda se iz svake teoreme može izvesti najviše konačno mnogo novih teorema, a postupak njihovog izvođenja je potpuno određen: svaka primena po jedne zamene daje po jednu novu teoremu. Pored toga, ako je A data reč, onda postoji postupak utvrđivanja da li ona jeste ili nije teorema. Dosta je, naime, uočiti sve reči (nad alfabetom razmatrane gramatike) čija je dužina manja ili jednaka dužini reči A (a njih je konačno na broju budući da je alfabet konačan), i najzad među dokazima koji "idu" preko takvih reči (a i njih je konačno mnogo) pronaći jedan koji se završava sa A. Ukoliko takav dokaz postoji A jeste teorema, a ukoliko ne postoji A nije teorema. U skladu sa uobičajenom terminologijom napred izložene karakteristike pojma "teoremnosti" se iskazuju kratko: Skup teorema kontekstno slobodne gramatike je rekurzivan, a svojstvo "biti teorema" je odlučivo. To je razlog što su te gramatike podesne za rad na računskim mašinama. Evo kako izgleda jedan BASIC program kojim se za datu kontekstno slobodnu gramatiku generišu sve njene teoreme ili se za datu reč ispituje da li ona jeste ili nije teorema. Program je upitno-odgovorni, za promenljive se može uzeti ma koji početni komad ovog spiska

$$s,\ t,\ u,\ v,\ w,\ x,\ y,\ z$$

Ako se, recimo, opredelimo za tri promenljive, onda će to biti upravo: s, t, u. Sto se tiče završnog alfabeta, on se ne precizira posebno: svi znaci koji učestvuju u pravilima a koji nisu promenljive smatraju se završnim. Promenljiva s je uzeta za aksiomu. Pravila se zadaju, na odgovarajući upit, zadavanjem odgovarajućih desnih strana. Program teče ovako: Najpre se na programom određenim načinom uočava gramatika. U tu svrhu prvo se zadaje koliko promenljivih ima ta gramatika. Dalje se za svaku promenljivu navodi broj njenih zamena, kao i same te zamene. Tačnije, za svaku promenljivu se navode sve reči sa kojima se ta promenljiva sme zameniti. Najzad se odgovara na programom postavljeno pitanje da li se želi **nizanje** svih

teorema do željenog broja, ili se želi utvrđivanje da li data reč jeste **teorema**. Evo i samog programa [urađenog u saradnji sa S. B. Prešićem]:

```
 10 PRINT "Kakav zadatak uraditi:  N I Z A T I teoreme do zeljenog broja ili
 20 PRINT "za datu rec ispitati da li je T E O R E M A (gramatike Comskog
 30 PRINT "koja ce se naknadno zadati) ?"
 40 PRINT:PRINT
 50 PRINT "Ako treba NIZATI teoreme kucati rec NIZATI"
 60 INPUT "a u drugom slucaju kucati rec TEOREMA";PIT$
 70 IF PIT$<>"NIZATI" AND PIT$<>"TEOREMA" THEN PRINT:PRINT:PRINT " G R E S K A
 80 PRINT
 90 PRINT "Najpre cemo zadati gramatiku"
100 DIM VAR$(8),MM(50),D$(8,10),T$(1000),ZT$(100)
110 PRINT "Koliko promenljivih ?":INPUT M
120 REM promenljive su s,t, ... ,z; m<=8
130 FOR I=1 TO M:VAR$(I)=CHR$(114+I):NEXT I
140 FOR I=1 TO M:PRINT "Za promenljivu ";VAR$(I);" koliko zamena?":INPUT MM(I)
150 FOR J=1 TO MM(I):PRINT "Koja je ";J;"-zamena?  ":INPUT D$(I,J):NEXT J,I
160 IF PIT$="TEOREMA" THEN GOSUB 380:GOTO 680
170 T$(1)=VAR$(1)
180 INPUT "Do kog broja redom nizati teoreme";BROJ
190 PRINT " Teorema ";1;":",T$(1)
200 L=1:T=1
210 TT=T
220 FOR I=L TO TT
230 PAM=0
240 FOR K=1 TO LEN(T$(I))
250 S$=MID$(T$(I),K,1)
260 IF ASC(S$)<115 OR ASC(S$)>122 THEN GOTO 330
270 FOR J=1 TO MM(ASC(S$)-114)
280 PAM=1:T=T+1
290 T$(T)=MID$(T$(I),1,K-1)+D$(ASC(S$)-114,J)+MID$(T$(I),K+1,LEN(T$(I))-K)
300 PRINT " Teorema ";T;":",T$(T)
310 IF T>BROJ-1 THEN GOTO 370
320 NEXT J
330 NEXT K
340 NEXT I
350 KK=KK+1
360 L=TT+1:GOTO 210
370 GOTO 680
380 INPUT "Koju rec ispitivati ";W$
390 T$(1)=W$
400 L=1:T=1
410 TT=T
420 FOR I=L TO TT
```

```
430 FOR X=1 TO M:FOR Y=1 TO MM(X)
440 MER=LEN(D$(X,Y))
450 FOR Z=1 TO LEN(T$(I))-MER+1
460 S$=MID$(T$(I),Z,MER)
470 IF S$<>D$(X,Y) THEN GOTO 530
480 BR=BR+1
490 T$=MID$(T$(I),1,Z-1)+CHR$(X+114)+MID$(T$(I),Z+MER,LEN(T$(I))-Z-MER+1)
500 GOSUB 630
510 IF P=0 THEN T=T+1:T$(T)=T$
520 IF T$(T)="s" THEN GOTO 600
530 NEXT Z
540 NEXT Y
550 NEXT X
560 NEXT I
570 KK=KK+1
580 IF BR=0 THEN GOTO 610
590 BR=0:L=TT+1:GOTO 410
600 PRINT "Jeste teorema":GOTO 620
610 PRINT "nije teorema"
620 STOP
630 FOR II=1 TO T
640 IF T$=T$(II) THEN P=1:GOTO 670
650 NEXT II
660 P=0
670 RETURN
680 END
```

Glavna primena gramatika Čomskog jeste u istraživanjima o prirodnim jezicima, radi kojih su one, uostalom, i pravljene. Ta su se istraživanja nadovezala na već zaokrugljena znanja o gramatičkim kategorijama. Određivan i definisan stolećima, još od Platona i Aristotela, spisak gramatičkih kategorija je dobrim delom uobličen radovima Ajdukeviča 30-tih godina ovog veka. Kao najčešće i najznačajnije gramatičke kategorije ističu se sledeće:

CN	(kategorija zajedničkih imenica)
NP	(kategorija imeničkih izraza)
IV	(kategorija neprelaznih glagola)
TV	(kategorija prelaznih glagola)
ADJ	(kategorija prideva)
ADV	(kategorija priloga)
PRP	(kategorija predloga)
CON	(kategorija veznika)
S	(kategorija rečenica)

[Istina, ubrzo posle objavljivanja glavnih rezultata Čomskog u istraživanja

prirodnih jezika se uključuje čitav niz matematičara, posebno logičara, na čelu sa Montegjuom. Tada se ispostavilo da se spisak gramatičkih kategorija mora bitno menjati: neke su potpuno nestale, neke su se raspale na više novih kategorija, a pojavile su se i sasvim nove kategorije koje lingvisti nisu nikada bili u stanju da uoče. Ali o tom potom.]

Evo sada kako izgleda jedna gramatika Čomskog kojom se generišu jednostavne rečenice engleskog jezika sastavljene od jedne vlastite imenice i jednog neprelaznog glagola.

PRIMER 6. *Pomoćni simboli te gramatike su* NP , IV , S *(oznake kategorija imeničkih izraza, neprelaznih glagola i rečenica), aksioma je S, završni simboli su izvesne vlastite imenice, na primer: John, Mary, Bill, Peter, kao i izvesni neprelazni glagoli (treće lice jednine sadašnjeg vremena), recimo: runs, walks, talks, writes (glagol writes je uzet kao neprelazan, tj. sa značenjem* **baviti se pisanjem**, *tj.* **biti pisac**). *Zamene su:*

NP $\longrightarrow$ *John*, NP $\longrightarrow$ *Mary*, NP $\longrightarrow$ *Bill*, NP $\longrightarrow$ *Peter*
IV $\longrightarrow$ *runs*, IV $\longrightarrow$ *walks*, IV $\longrightarrow$ *talks*, IV $\longrightarrow$ *writes*
S $\longrightarrow$ NP VP

Kao završne teoreme dobijaju se rečenice:

John runs, Mary runs, Bill runs, Peter runs, John walks, i sl.,

ukupno 4x4=16 rečenica. Dokaz prve od njih izgleda:

S $\longrightarrow$	NP IV	(zadnje pravilo)
NP IV $\longrightarrow$	*John* IV	(primena prvog pravila)
John $\longrightarrow$	*runs*	(primena pravila IV $\longrightarrow$ *runs*)

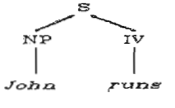
Taj se dokaz može skraćeno zapisati i u obliku ovakvog produženog niza:

S$\longrightarrow$ NP IV $\longrightarrow$ John IV $\longrightarrow$ John runs,

što je uobičajeno pisanje u ma kojoj gramatici Čomskog, a može se zgodno prikazati navedenim drvetom. Pored prethodnog, rečenica John runs očigledno ima i ovaj dokaz:

S$\longrightarrow$ NP IV $\longrightarrow$ NP runs$\longrightarrow$ John runs

kome takođe odgovara prikazano drvo.



Prethodna gramatika se može dalje dograđivati tako da se kao nove teoreme pojavljuju i rečenice u kojima učestvuju prelazni glagoli, složeniji imenički izrazi, pridevi, rečenični veznici i dr. Evo jedne takve gramatike:

PRIMER 7. *Njeni pomoćni simboli su:* CN , NP , IV , TV , ADJ , DET, CON , *S, aksioma je* S , *završni simboli su članovi narednih skupova:*

$A_1 = \{man, woman, apple, cat, letter, fish\}$
$A_2 = \{John, Mary, Bill, Peter, he_0, he_2, he_3, \ldots, he_{10}\}$
$A_3 = \{runs, walks, talks, writes\}$
$A_4 = \{eats, loves, seeks, reads, writes\}$
$A_5 = \{tall, sweet, long, black, old, odd\}$
$A_6 = \{the, a, every\}$
$A_7 = \{and, or\}$

[U imeničke izraze je uključeno i deset zamenica trećeg lica jednine muškog roda, nešto što nije baš uobičajeno u govornom jeziku ali je postalo svakodnevna praksa u današnjim matematičkim istraživanjima tih jezika. Uostalom, neophodnost uvođenja više zamenica se lako uočava u raznim jezičkim vratolomijama koje se pojavljuju u slučajevima kada se prirodno nametne potreba za više takvih ili nekih drugih imeničkih izraza iste vrste. Evo kako se tada obično dovijamo, kao u primeru:

> **"Petar je dobio pismo od jedne žene koje je donela druga žena, a ta druga žena je prijateljica treće žene koju je Petar upoznao prošle godine na letovanju."**

Pored toga kao gramatička kategorija pojavljuje se DET-kategorija određivača. To je jedna od potpuno novih i veoma značajnih kategorija koje lingvisti duguju matematičarima. Pomenimo još jednu značajnu kategoriju koju su uveli matematičari, a o kojoj će u daljem još biti reči. To je kategorija QU-kategorija kvantora, kojoj pripadaju izrazi kao:

the man, a man, every man, every woman, most men, both man and woman.]

Zamene su:

CN $\longrightarrow$ ma koja reč skupa A_1, NP $\longrightarrow$ ma koja reč skupa A_2
IV $\longrightarrow$ ma koja reč skupa A_3, TV $\longrightarrow$ ma koja reč skupa A_4
ADJ $\longrightarrow$ ma koja reč skupa A_5, CON $\longrightarrow$ ma koja reč skupa A_6

S $\longrightarrow$ NP IV, S $\longrightarrow$ S CON S, IV $\longrightarrow$ TV NP ,
CN $\longrightarrow$ ADJ CN , NP $\longrightarrow$ DET CN

U toj gramatici se mogu izvesti složene rečenice kakve su na primer:

John loves Mary, John loves every woman, Every man loves a woman, Peter writes a long letter, A woman eats a sweet apple, John runs and John eats a sweet apple, A man seeks a black cat and he runs,

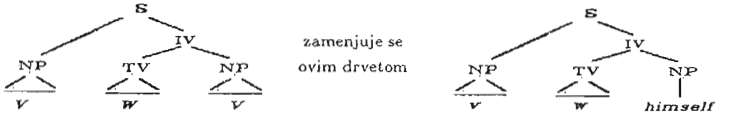
ali takođe i rečenice sintaksno potpuno ispravne ali neobičnog značenja, kao što su ove, na primer:

Peter writes a black apple, The black apple runs and every cat writes a long letter, The long odd letter reads every black cat, A tall black cat writes every old woman, The woman seeks a tall odd fish or he_1 *writes*

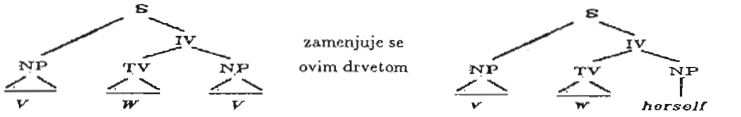
Takve rečenice, međutim, uopšte ne smetaju; one su neuobičajene u svakodnevnom pisanom i govornom jeziku ali su u poeziji sasvim moguće, dozvoljene, prihvatljive. Smetaju, međutim, rečenice:

John loves John, John loves he_3*, The tall woman loves the tall woman*

za koje se ne može reći samo da su neuobičajene već i da su gramatički neispravne. Kako izbeći takve rečenice? Jedan način je da se odgovarajuća gramatika obogati takozvanim **transformacijama**. Na primer, umesto rečenice *John loves John* treba da se pojavi rečenica *John loves himself.* Odgovarajuća trasformacija kojom se to postiže izgleda:



ukoliko je v muškog roda a u slučaju ženskog roda umesto prethodne usvajamo transformaciju:



[Kao što je uobičajeno trouglasti završeci drveta, kakav je recimo:  označavaju ispušteno poddrvo čiji je koren NP , u uočenom primeru.]

Kao što se uočava, tim transformacijama se jedno izvođenje (prikazano odgovarajućim drvetom) zamenjuje drugim izvođenjem. Pri tome su NP , IV ma koji imenički izraz, odnosno ma koji prelazani glagol (u ovom našem primeru NP je element skupa A_2, a IV je element skupa A_4). Transformacije se obično zadaju mnogo kraće, bez isticanja drveta izvođenja (ali tada je često neophodno uvesti više pomoćnih simbola.) Tako se prethodne dve transformacije obično zadaju u obliku:

NP TV NP $\longrightarrow$ NP TV *himself* (Ukoliko je NP muškog roda jednine)
NP TV NP $\longrightarrow$ NP TV *herself* (Ukoliko je NP ženskog roda jednine)

Slično se može uvesti transformacija kojom se rečenica *John loves* he_3 prevodi u rečenicu *John loves* him_3, koja je gramatički ispravna, ili transformacija kojom se ta ista rečenica prevodi u pasivni oblik: He_3 *is loved by John*, a takođe i transformacije kojima se od rečenice kakva je na primer:

Every man likes some woman

prelazi na izraze:

(7) *Every man such that that man likes some woman*

(8) *Some woman such that every man likes that woman*

Transformacije kojima se to postiže su dužine dva i izgledaju:

DET CN , DET CN TV NP ⟶ DET CN *such that that* CN TV NP
DET CN , NP TV DET CN ⟶ DET CN such that NP TV that CN

Odgovarajućim transformacijama se rečenice izvedene u nekoj gramatici Čomskog mogu prevesti u rečenice u pluralu, u nekom prošlom vremenu, u budućem vremenu, dalje u rečenice u kojima se sva pojavljivanja istog imeničkog izraza (izuzev prvog) zamenjuju odgovarajućom ličnom zamenicom i tako dalje.

Gramatike Čomskog u kojima se pored pravila zamene pojavljuje i izvestan broj transformacija nazivaju se, inače, **transformacionim** gramatikama; kao i kontekstno slobodne i one takođe potiču od Čomskog. Iz prethodnih primera se vidi da se transformacione gramatike dobiju iz običnih kontekstno slobodnih gramatika dodavanjem izvesnih transformacija, koje nisu ništa drugo do pravila izvođenja nad skupom svih reči uočene gramatike.

Očigledno je da su kontekstno slobodne gramatike bitno obogaćene uvođenjem transformacija. Međutim, ne izgleda baš prirodno da se, recimo, rečenica srpskohrvatskog jezika:

Ona je upravo otišla

dobija odgovarajućom transformacijom iz rečenice:

On je upravo otišao

[Ako ni zbog čega drugog ono zbog izbegavanja vekovne dominacije muškog roda. Ali ne vide se ni valjani razlozi, mislim na gramatičke, po kojima bi trebalo usvojiti da se prva rečenica dobija transformacijom druge.]

Slično se odnosi i na uzajamnu vezu rečenica u svakom od narednih parova:

Zoran je lep, Milica je lepa
Zoran svira, Zoran i Milica sviraju
Zoran spava, Zoranu se spava

Uvođenje transformacija kojima se povezuju rodovi, brojevi, padeži, vremena i razne druge promenljive odrednice kojima obiluju naročito jezici sa izrazitom fleksijom, može biti veoma komplikovano ili, što je još veća mana, veoma neprirodno. To je bio jedan od razloga što je stvoren još moćniji matematički aparat za istraživanje

prirodnih jezika. To su **gramatike metamorfoze** [potiču od Kolmeroera (Colmerauer) iz 1975. godine], kod kojih "svet" na kome se događaju jezička zbivanja nije više tako širok i neuhvatljiv kao kod kontekstno slobodnih i transformacionih gramatika [Kod njih je to, setimo se, skup svih reči nad datim alfabetom], već je uži, ali zato pravilniji i svojstveno bogatiji, pa se stoga njime lakše "hvataju" razna fleksivna svojstva. To je, naime jezik izraza, terma nad nekim operacijskim jezikom, takozvani **Erbranov svet** (Herbrand). Podsećamo da se Erbranov svet, označimo ga sa $Term[F]$, nad operacijskim jezikom F i fiksiranim skupom promenljivih, recimo $\{x_1, x_2, x_3, \dots\}$, uvodi ovako:

To je najmanji skup koji sadrži sve promenljive, sve operacijske znake dužine nula (tj. sve znake konstanata) iz F, i za svaki operacijski znak $f \in F$ dužine n sadrži reč $f(t_1, t_2, \dots, t_n)$, gde su t_1, t_2, ... , t_n ma koji termi iz $Term[F]$.

Gramatike metamorfoze se, tada, uvode definicijom:

DEFINICIJA 2. *Neka je F izvestan skup operacijskih znakova, za koji se zahteva da obavezno sadrži znake $\cdot$ (dužine dva) i nil (dužine nula) [nil, ustvari, ima ulogu prazne reči], i neka je $Term[F]$ odgovarajući Erbranov svet. Gramatika metamorfoze (nad F) je određena sa tri skupa (podskupa od $Term[F]$):*

T (skup završnih, terminalnih izraza)

N (skup pomoćnih, neterminalnih izraza)

A (skup polaznih izraza, tj.aksioma)

i izvesnim pravilima zamene nad skupom $(T \cup N)$. Pri tome su T i N međusobno disjunktni, dok je A podskup od N. Za zamene se zahteva da ne budu oblika $X \longrightarrow nil$.

Kao i kod prethodnih gramatika, teoreme su sve one reči, bliže svi oni izrazi iz $Term(F)$, koji slede iz aksioma, pri čemu i u ovom slučaju razlikujemo završne od nezavršnih teorema. Skup svih završnih teorema čini **jezik** generisan uočenom gramatikom metamorfoze.

PRIMER 8. *Skup F je $\{nil, zero, a, b, suite, bs, suc, \cdot\}$, pri čemu:*

$$red(nil) = 0, \quad red(zero) = 0, \quad red(a) = 0, \quad red(b) = 0$$
$$red(suite) = 1, \quad red(bs) = 1, \quad red(suc) = 1$$
$$red(\cdot) = 2$$

[Sa $red(f)$ smo označili red, tj. dužinu operacijskog znaka f.]

Skupovi T, A, N su sledeći:

$$T = \{a, b\}, \quad A = \{suite(x) : x \in Ter(F)\}, \quad N = A \cup \{bs(x) : x \in Ter(F)\}$$

Zamene su shema-zamene;

$$suite(x) \longrightarrow a \cdot suite(suc(x)),\ suite(x) \longrightarrow bs(x)$$
$$bs(suc(x)) \longrightarrow b \cdot bs(x),\ bs(zero) \longrightarrow nil$$

gde je x ma koji izraz iz $Term(F)$.

Jedna teorema (završna) je, na primer $a \cdot (b \cdot (b \cdot (b \cdot nil)))$ a jedan njen dokaz je:

$$\begin{aligned} suite(suc(suc(zero))) &\longrightarrow a \cdot suite(suc(suc(suc(zero)))) \\ &\longrightarrow a \cdot bs(suc(suc(suc(zero)))) \\ &\longrightarrow a \cdot (b \cdot bs(suc(suc(zero)))) \\ &\longrightarrow a \cdot (b \cdot (b \cdot bs(suc(zero)))) \\ &\longrightarrow a \cdot (b \cdot (b \cdot (b \cdot bs(zero)))) \\ &\longrightarrow a \cdot (b \cdot (b \cdot (b \cdot \mathrm{nil}))) \end{aligned}$$

Pri tome su redom primenjivane zamene: prva, druga, treća (tri puta uzastopce), i najzad četvrta. Nije teško dokazati da ta gramatika generiše jezik kome pripadaju reči:

$$a \cdot (b \cdot (b \cdot nil)), a \cdot (a \cdot (b \cdot (b \cdot (b \cdot nil)))),\quad a \cdot (a \cdot (a \cdot (b \cdot (b \cdot (b \cdot (b \cdot nil))))))$$

i uopšte reči koje su prozvod i a-ova, odnosno j b-ova i konstante nil, pri čemu su i, j ma koji prirodni brojevi i još je $j \geq i$, a proizvod je takozvani "desni", tj. sa zagradama grupisanim na desnu stranu. Posebna uloga koju imaju operacijski znaci $\cdot$ i nil u vezi je sa jednom važnom teoremom o algebarskim strukturama prema kojoj se svaka algebarska struktura (na proizvoljnom operacijskom jeziku) može izomorfno potopiti u strukturu sa jednom binarnom operacijom i izvesnim brojem konstanata. [Štaviše, ta binarna operacija može biti i asocijativna, a može imati i jedinični element.] Upravo tome služe operacijski znaci $\cdot$ i nil. Pomoću njih se ma koja operacija f dužine n može ovako izraziti

$$f(x_1, x_2, \ldots, x_n) = (f \cdot (x_1 \cdot (x_2 \ldots, (x_n \cdot nil) \ldots))) \tag{9}$$

Pri tome, znaci f sa leve i desne strane jednakosti, iako su označeni na isti način, imaju i različit smisao i različitu ulogu: sa leve strane je to operacijski znak dužine n koji se jednakošću (9) definiše, a sa desne strane je to znak konstante, koji za svaku novu operaciju mora takođe biti nov, do tada neupotrebljen.

Inače, uobičajeno je da se izraz oblika $(a_1 \cdot (a_2 \cdot (a_3 \ldots, (a_n \cdot nil) \ldots)))$ naziva **listom** i da se označava ovako: $[a_1, a_2, \ldots, a_n]$. Pri tome se prvi član a_1 naziva **glavom** a ostatak liste, tj. $[a_2, a_3, \ldots, a_n]$ je takozvani **rep**. Koristeći tu oznaku jednakost (9) se zapisuje u obliku:

$$f(x_1, x_2, \ldots, x_n) = [f, x_1, x_2, \ldots, x_n] \tag{10}$$

Pomenimo da je izražavanje operacija pomoću $\cdot$ i nil dato jednakostima (9), odnosno (10) ugrađeno u neke programske jezike, na primer u LISP i PROLOG.

Evo sada jednog primera koji se odnosi na prirodne jezike (i to ponovo na engleski) iz koga se jasno vidi kako se podesno izabranom gramatikom metamorfoze mogu "uhvatiti" razna flektivna svojstva kao što su rod i broj.

PRIMER 9. *Operacijski jezik čine znaci: s, cn, np, iv, tv, det, singular, plural, person, thing, musician, musicians, violin, plays, play, the i, naravno, znaci $\cdot$, nil, koji se podrazumevaju. Pri tome su s, cn, np, iv, tv, det, kao i svi termi u kojima oni učestvuju, pomoćni znaci a sve preostale reči su završni znaci. Dalje, np i cn su dužine dva, tv i iv su dužine jedan a svi ostali znaci su dužine nula, odnosno oni su konstante. Jedina aksioma je s dok su zamene :*

```
                  s ⟶ np(Number, person), iv(Number)
    np(Number, Type) ⟶ det, cn(Number, Type)
          iv(Number) ⟶ tv(Number), np(Number1, thing)
cn(singular, person) ⟶ [musician]
  cn(plural, person) ⟶ [musicians]
 cn(singular, thing) ⟶ [violin]
        tv(singular) ⟶ [plays]
          tv(plural) ⟶ [play]
                 det ⟶ [the]
```

U toj gramatici reči koje počinju velikim slovom su promenljive, dok su gramatičke kategorije, kao što se vidi, označene malim slovima. Najzad, zarez koji razdvaja terme na desnoj strani zamena ima ulogu konkatenacije (za liste). Usvojene oznake su u skladu sa uobičajenim oznakama u PROLOG-u, programskom jeziku koji je tesno povezan i razvijan uporedo sa gramatikama metamorfoze [i njegov tvorac je Kolmaroe], te je stoga i najpodesniji za prevođenje date gramatike metamorfoze u odgovarajući kompjuterski program. Gramatikom u ovom primeru generišu se rečenice:

The musician plays the violin, The musicians play the violin

koje se dobijaju u obliku lista:

[*the, musician, plays, the, violin*], [*the, musicians, play, the, violin*]

Prevođenje gramatike metamorfose u programski jezik PROLOG vrši se na standardan način, uvođenjem pomoćnih promenljivih. Tako imamo da se

```
a(Prom1) ⟶ [b],    a(Prom1) ⟶ b(Prom2),
a(Prom1) ⟶ b(Prom2),c(Prom3)
```

redom prevode u ove naredbe PROLOG-a:

```
a(Prom1,[b|U],U),  a(Prom1,U,V) :- b(Prom2,U,V),
a(Prom1,[b|U],U),  a(Prom1,U,V) :- b(Prom2,U,V),
                   a(Prom1,U,W) :- b(Prom2,U,V),  c(Prom3,U,W).
```

gde su sa *Prom1*, *Prom2*, *Prom3* označeni proizvoljni nizovi promenljivih. Na primer, zamenama gramatike iz prethodnog primera odgovara ovaj program u PROLOG-u:

```
np(Number,Type,U,W):-det(U,V),cn(Number,Type,V,W).
          s(U,W):-np(Number,person,U,V),iv(Number,V,W).
np(Number,Type,U,W):-det(U,V),cn(Number,Type,V,W).
     iv(Numer,U,W):-tv(Number,U,V),np(Number1,thing,V,W).
                    cn(singular,person,[musician|U],U).
                    cn(plural,person,[musician|U],U).
                    cn(singular,thing,[violin|U],U).
                    tv(singular,[plays|U],U).
                    tv(plural,[play|U],U).
                    det([the|U],U).
```

Naravno prethodna gramatika se jednostavno proširuje do gramatike kojom se generišu slične rečenice obrazovane od jednog imeničkog izraza, prelaznog glagola i još jednog imeničkog izraza; za to je dovoljno rečnik polaznih reči engleskog jezika proširiti novim rečima i dodati odgovarajuće zamene.

Gramatike metamorfoze su posebno dobro primenljive u izrazito flektivnim jezicima kakav je srpskohrvatski. Naravno flektivnost povlači za sobom uvođenje većeg broja parametara nego što je to slučaj u engleskom jeziku, te su stoga i odgovarajuće gramatike komplikovanije. Navodimo jedan program u PROLOG-u kojim se, polazeći od datog rečnika srpskohrvatskog jezika, generišu sve elementarne rečenice (na tom rečniku), tj. sve rečenice sagrađene primenom nekog glagola na odgovarajući broj imeničkih izraza koji su, uz to, u odgovarajućim padežnim oblicima. Program izgleda ovako:

```
imenica(jednina,muski,zivo)--->[muzicar,muzicara,muzicaru,muzicara,
                                muzicarom,muzicaru].

imenica(jednina,muski,zivo)--->[covek,coveka,coveku,coveka,covekom,
                                coveku].

imenica(jednina,zenski,zivo)--->[zena,zene,zeni,zenu,zenom,zeni].

imenica(jednina,zenski,nezivo)--->[violina,violine,violini,violinu,
                                   violinom,violini].

imenica(jednina,zenski,nezivo)--->[jabuka,jabuke,jabuci,jabuku,
                                   jabukom,jabuci].

padez_imenice(Broj,Rod,Vrsta,nom,[A|S],S):-imenica(Broj,Rod,Vrsta,
                                            [A,B,C,D,E,F|S],S).
```



```
padez_imenice(Broj,Rod,Vrsta,gen,[B|S],S):-imenica(Broj,Rod,Vrsta,
                                            [A,B,C,D,E,F|S],S).

padez_imenice(Broj,Rod,Vrsta,dat,[C|S],S):-imenica(Broj,Rod,Vrsta,
                                            [A,B,C,D,E,F|S],S).

padez_imenice(Broj,Rod,Vrsta,akuz,[D|S],S):-imenica(Broj,Rod,Vrsta,
                                            [A,B,C,D,E,F|S],S).

padez_imenice(Broj,Rod,Vrsta,inst,[E|S],S):-imenica(Broj,Rod,Vrsta,
                                            [A,B,C,D,E,F|S],S).

padez_imenice(Broj,Rod,Vrsta,lok,[F|S],S):-imenica(Broj,Rod,Vrsta,
                                            [A,B,C,D,E,F|S],S).

glagol(jednina,[zivo],[nom])--->[spava];
                                [pise];
                                [peva].

glagol(mnozina,[zivo],[nom])--->[spavaju];
                                [pisu];
                                [pevaju].

glagol(jednina,[zivo],[dat])--->[se_spava];
                                [se_zeva];
                                [se_putuje].

glagol(mnozina,[zivo],[dat])--->[se_spava];
                                [se_zeva].
                                [se_putuje];

glagol(jednina,[zivo,nezivo],[nom,akuz])--->[jede];
                                           [svira];
                                           [pise].

glagol(mnozina,[zivo,nezivo],[nom,akuz])--->[jedu];
                                           [sviraju];
                                           [pisu].

glagol(jednina,[zivo,zivo,nezivo],[nom,dat,akuz])--->[pise];
                                                    [daje];
                                                    [nosi].

glagol(mnozina,[zivo,zivo,nezivo],[nom,dat,akuz])--->[pisu];
                                                    [daju];
                                                    [nose].
```

```
glagol(jednina,[zivo,zivo],[nom,dat])--- >[pise];
                                         [se_divi];
                                         [se_okrece].

glagol(mnozina,[zivo,zivo],[nom,dat])--- >[pisu];
                                         [se_dive];
                                         [se_okrecu].

glagol(jednina,[zivo,nezivo],[nom,dat])--- >[se_divi];
                                           [se_okrece].

glalol(mnozina,[zivo,nezivo],[nom,dat])--- >[se_dive];
                                           [se_okrecu].

glagol(Broj,[Vrste|[ ]],[nom])--- >glagol(Broj,[Vrste|S],[nom,akuz]),
                                   padez_imenice(Broj1,Rod1,S,akuz).

glagol(Broj,[Vrste|[ ]],[nom])--- >glagol(Broj,[Vrste|S],[nom,dat]),
                                   padez_imenice(Broj1,Rod1,S,dat).

glagol(Broj,[Vrste|[ ]],[nom,akuz])--- >glagol(Broj,[Vste|S],[nom,dat,akuz]),
                                        padez_imenice(Broj1,Rod1,S,dat).

prod([A1,A2,A3,A4,A5,A6],[B1,B2,B3,B4,B5,B6],[C1,C2,C3,C4,C5,C6]):-
                         C1=[A1,B1],C2=[A2,B2],C3=[A3,B3],
                         C4=[A4,B4],C5=[A5,B5],C6=[A6,B6].

term(jednina,zenski,zivo)--- >[vera,vere,veri,veru,verom,veri].

term(jednina,muski,zivo)--- >[jova,jove,jovi,jovu,jovom,jovi].

term(Broj,Rod,Vrsta,[C1,C2,C3,C4,C5,C6|S],S):-
                    odredivac(Broj,Rod,Vrsta,[A1,A2,A3,A4,A5,A6|S],S),
                    imenica(Broj,Rod,Vrsta,[B1,B2,B3,B4,B5,B6|S],S),
                    prod([A1,A2,A3,A4,A5,A6],[B1,B2,B3,B4,B5,B6],
                        [C1,C2,C3,C4,C5,C6]).

append([ ],X,X).

append([X|Y],Z,U):-append(Y,Z,W),
                   U=[X|W].

padez_terma(Broj,Rod,Vrsta,nom,T,S):-term(Broj,Rod,Vrsta,
                                    [A,B,C,D,E,F|S],S),
                                    a(A,S,T).
```



```
padez_terma(Broj,Rod,Vrsta,gen,T,S):-term(Broj,Rod,Vrsta,
                                    [A,B,C,D,E,F|S],S),
                                    a(B,S,T).

padez_terma(Broj,Rod,Vrsta,dat,T,S):-term(Broj,Rod,Vrsta,
                                    [A,B,C,D,E,F|S],S),
                                    a(C,S,T).

padez_terma(Broj,Rod,Vrsta,akuz,T,S):-term(Broj,Rod,Vrsta,
                                     [A,B,C,D,E,F|S],S),
                                     append(D,S,T).

padez_terma(Broj,Rod,Vrsta,inst,T,S):-term(Broj,Rod,Vrsta,
                                     [A,B,C,D,E,F|S],S),
                                     a(E,S,T).

padez_terma(Broj,Rod,Vrsta,lok,T,S):-term(Broj,Rod,Vrsta,
                                    [A,B,C,D,E,F|S],S),
                                    a(F,S,T).

a(W,S,T):-ifthenelse((W=[U|V]),append(W,S,T),(T=[W|S])).

odredivac(jednina,muski,zivo)--- >[neki,nekog,nekom,nekog,nekim,nekom];
                                 [svaki,svakog,svakom,svakog,svakim,
                                 svakom].

odredivac(mnozina,muski,zivo)--- >[neki,nekih,nekim,neke,nekim,nekim];
                                 [svaki,svakih,svakim,svake,svakim,
                                 svakim].

odredivac(jednina,muski,nezivo)--- >[neki,nekog,nekom,neki,nekim,nekom];
                                   [svaki,svakog,svakom,svaki,svakim,
                                   svakom].

odredivac(mnozina,muski,nezivo)--- >[neki,nekih,nekim,neke,nekim,nekim];
                                   [svaki,svakih,svakim,svake,svakim,
                                   svakim].

odredivac(jednina,zenski,zivo)--- >[neka,neke,nekoj,neku,nekom,nekoj];
                                  [svaka,svake,svakoj,svaku,svakom,
                                  svakoj].

odredivac(mnozina,zenski,zivo)--- >[svake,svakih,svakim,svake,svakim,
                                  svakim];
                                  [neke,nekih,nekim,neke,nekim,nekim].
```

```
određivac(jednina,zenski,nezivo)--->[neka,neke,nekoj,neku,nekom,nekoj];
                                    [svaka,svake,svakoj,svaku,svakom,
                                    svakoj].

određivac(mnozina,zenski,nezivo)--->[neke,nekih,nekim,neke,nekim,nekim];
                                    [svake,svakih,svakim,svake,svakim,
                                    svakim].

padez_određivaca(Broj,Rod,Vrsta,nom,[A|S],S):-određivac(Broj,Rod,Vrsta,
                                              [A,B,C,D,E,F|S],S).

padez_određivaca(Broj,Rod,Vrsta,gen,[B|S],S):-određivac(Broj,Rod,Vrsta,
                                              [A,B,C,D,E,F|S],S).

padez_određivaca(Broj,Rod,Vrsta,dat,[C|S],S):-određivac(Broj,Rod,Vrsta,
                                              [A,B,C,D,E,F|S],S).

padez_određivaca(Broj,Rod,Vrsta,akuz,[D|S],S):-određivac(Broj,Rod,Vrsta,
                                              [A,B,C,D,E,F|S],S).

padez_određivaca(Broj,Rod,Vrsta,inst,[E|S],S):-određivac(Broj,Rod,Vrsta,
                                              [A,B,C,D,E,F|S],S).

padez_određivaca(Broj,Rod,Vrsta,lok,[F|S],S):-određivac(Broj,Rod,Vrsta,
                                              [A,B,C,D,E,F|S],S).

recenica--->padez_terma(Broj1,Rod1,Vrsta1,nom),
            glagol(Broj1,[Vrsta1],[nom]).

recenica--->padez_terma(Broj1,Rod1,Vrsta1,dat),
            glagol(Broj1,[Vrsta1],[dat]).

recenica--->padez_terma(Broj1,Rod1,Vrsta1,nom),
            glagol(Broj1,[Vrsta1,Vrsta2],[nom,akuz]),
            padez_terma(Broj2,Rod2,Vrsta2,akuz).

recenica--->padez_terma(Broj1,Rod1,Vrsta1,nom),
            glagol(Broj1,[Vrsta1,Vrsta2],[nom,dat]),
            padez_terma(Broj2,Rod2,Vrsta2,dat).
```



```
recenica--->padez_terma(Broj1,Rod1,Vrsta1,nom),
            glagol(Broj1,[Vrsta1,Vrsta2,Vrsta3],[nom,dat,akuz]),
            padez_terma(Broj2,Rod2,Vrsta2,dat),
            padez_terma(Broj3,Rod3,Vrsta3,akuz).
```

Nekoliko objašnjenja u vezi sa programom: nazivi svih gramatičkih kategorija su prevedeni na srpskohrvatski. Imenice se zadaju kao uređene šestorke, tj. kao odgovarajuće liste, padežnih oblika (vokativ je izostavljen budući da on ne učestvuje u izgradnji rečenice, već ima isključivo ulogu dozivanja). Kategorija imenica zavisi od broja, roda i vrste (živo, neživo). Glagoli su dužine jedan, dva i tri i zavise od broja, vrsta i padeža. Pri tome su sada vrste i padeži liste čija je dužina jednaka gužini glagola a članovi tih lista su popunjeni odgovarajućim vrstama i padežima u kojima moraju biti imenički izrazi na koje se glagol primenjuje. Na primer glagol **dati** je dužine tri, i ima kao odrednice: jedninu, [zivo,zivo,nezivo],[nom,dat,akuz]. Od tog glagola i tri imenička izraza čije su vrste redom zivo,zivo,nezivo, koji su po redu u nominativu, dativu, akuzativu, i od kojih je prvi izraz u jednini obrazuje se elementarna rečenica kakva je na primer:

Neki čovek daje Jovi neku violinu

Slično imenicama, i određivači se zadaju kao liste odgovarajućih padežnih oblika. Kategoriji terma pripadaju vlastite imenice (zadane listama padežnih oblika) i imenički izrazi čija su pravila izgradnje data. Za imenice, određivače i terme data su odvojeno pravila izgradnje svih padežnih oblika (nominativa, genitiva, dativa, akuzativa, instrumentala i lokativa za koje koristimo skraćenice po redu: nom, gen, dat, akuz, inst, lok). Najzad, u program je uvršćena i definicija operacije append (dopisivanje lista) koja u nekim varijantama prologa nije ugrađena. Navedenim programom se generišu, recimo, rečenice:

Vera spava, Veri se spava, Svi ljudi pevaju, Svaki čovek se divi nekoj ženi, Jova svira violinu, Svaki muzičar se okreće nekoj violini, Svaki čovek nosi nekoj ženi neku jabuku, Svakoj ženi se jede neka jabuka, itd.

Naravno, proširenjem polaznog rečnika mogu se dobiti razne druge rečenice (elementarne), obrazovane od drugih glagola i drugih imeničkih izraza. Za to je dovoljno u prethodni program dodati odgovarajuće padežne oblike imenica, novih određivača, a za glagole navesti sve njegove odrednice (ukoliko glagol može imati više dužina, za svaku odabranu dužinu navode se odgovarajuće odrednice).

I pored širokih mogućnosti koje su se otvorile pojavom gramatika metamorfoze problemi koji stalno iskrsavaju u vezi sa strogim matematičkim istraživanjima prirodnih jezika su i dalje mnogobrojni. Glavni razlog je razuđenost i bogatstvo prirodnih jezika, kao i njihov neprekidni razvoj. Pomenimo samo kvantore (količinske) koje u ovom članku nismo ni dotakli. To je jedna potpuno nova kategorija koju su uveli matematičari (kvantori su na primer: svaki čovek, mnogi ljudi, obe žene, većina stvari i sl). U vezi sa kvantorima, i sa pitanjima izgradnje jednog strogog jezika koji neće patiti od nedostatka dvosmislenosti (što je, inače glavna

mana svih formalnih gramatika), Ričard Montegju (Montague) je početkom 70-tih godina zasnovao jedan potpuno nov pravac u matematičkim istraživanjima prirodnih jezika, poznat danas pod imenom Montegjuove gramatike. Njegova osnovna ideja je bila paralelna izgradnja kako sintakse, tako i semantike prirodnih jezika, pa je u vezi sa tim izgradio jednu novu logiku, takozvanu intenzionalnu, prilagođenu zahtevima prirodnih jezika. No, o tome u novom članku.

LITERATURA

[1] A. Colmerauer, *Les grammaires de metamorphose,* G.I.A., Université d' Aix-Marseillw, November 1975.
[2] Noam Chomsky, *Syntactic Structures,* Mouton, Hague 1957.
[3] M. Gross & A. Lentin, *Notions sur les grammaires formelles,* Gauthier–Villars, Paris 1967
[4] Milka Ivić, *Pravci u lingvistici,* Državna založba Slovenije, Ljubljana 1970
[5] Olga Mišeska Tomić, *Syntax and Syntaxes,* Savremena administracija, Beograd 1987
[6] R. Montague, *The Proper Treatment of Quantification in Ordinary English,* U knjizi J. Hintikka, J. Mooravcsik, and P. Suppes (eds.) Approaches to Natural Languages: Proceedings of the 1970 Stanford Workshop on Grammar and Semantics, D. Reidel Publishing Company Dordrecht 1973
[7] F. Pereira, & D. H. D. Warren, *Definite Clause Grammars for Language Analysis—a Survey of the Formalism and Comparison with Augmented Transition Networks,* Artificial Intelligence 13 (1980), pp. 231–278
[8] A. B. Tucker, *Programming Languages,* McGraw–Hill, New York, 1986

Matematički fakultet
Studentski trg 16
11000 Beograd

RAČUNARSTVO
Sveska 1, Broj 2, 1992

UDK 519.688

ALGORITMI ZA SRAVNJIVANJE NISKOVNIH OBRAZACA[1]

CVETANA KRSTEV

1 Uvod

Podaci koje treba računarski obrađivati se često ne mogu logički predstaviti pomoću nezavisnih zapisa koji se sastoje od malih prepoznatljivih delova. Ovakva vrsta podataka se najčešće zapisuje u obliku, obično veoma dugačke, niske.

Ovakva vrsta podataka je osnovna, na primer, kod takozvanih sistema za „obradu reči" koji pružaju obilje mogućnosti za manipulaciju tekstom na nekom prirodnom jeziku. U tom slučaju govorimo o tekstualnim niskama. Ovakve tekstualne niske nastaju nad azbukom koja sadrži slova, brojeve, interpunkcijske i specijalne znake. Tekstualne niske kojima manipulišu sistemi za obradu reči mogu biti veoma dugačke: tekst neke knjige može da sadrži i preko milion karaktera.

Poseban primer niski su binarne niske koje se tvore nad azbukom od samo dva simbola $\{0, 1\}$. Ovakve binarne niske se koriste u sistemima za računarsku grafiku koji sliku predstavljaju u obliku binarne niske: 0 predstavlja neosvetljen a 1 osvetljen piksel na ekranu ili nekom drugom izlaznom uređaju. Premda suštinskih razlika između tekstualnih i binarnih niski nema, veličina azbuke nad kojom se one tvore utiče na izbor najefikasnijeg algoritma za njihovo sravnjivanje.

Fundamentalna operacija nad niskama je sravnjivanje niskovnih obrazaca. Tako, na primer, svi sistemi za obradu reči obezbeđuju funkciju „pronađi" koja u tekstu pronalazi zadati niskovni obrazac. U bibliografskim sistemima je sravnjivanje niskovnih obrazaca uključeno u pretraživanje teksta pomoću ključnih reči. Sravnjivanje niskovnih obrazaca je esencijalno i u raznim leksičkim i lingvističkim analizama.

Sam niskovni obrazac može biti različitog oblika [1]: jedna ključna reč, konačan skup ključnih reči, regularni izraz ili može biti opisan na neki drugi proceduralni ili

[1]Rad finansiran od Fonda za nauku Republike Srbije preko Matematičkog instituta, projekat 0401A.

neproceduralni način (kakav omogućuje, na primer, SNOBOL-ski program [5]). U ovom radu biće opisani fundamentalni algoritmi za sravnjivanje jedne niske (ključne reči) sa tekstom.

Problem se može definisati na sledeći način: za dati tekst x dužine N i za dati obrazac (ključnu reč) p dužine M treba pronaći prvo pojavljivanje obrasca unutar teksta. Pretpostavljamo, takođe, da je $M > 0$, tj. da obrazac nije prazna niska. Većina algoritma za rešavanje ovog problema prolazi sekvencijalno kroz tekst pa se lako može uopštiti da pronađe sva pojavljivanja obrasca unutar teksta: kada se obrazac sravni, algoritam ponovo počinje sa skaniranjem teksta počev od pozicije u tekstu koja neposredno sledi početak poslednjeg sravnjenog obrasca.

2 Algoritam „grube sile"

Očigledan metod za sravnjivanje zadatog obrasca sa tekstom je isprobavanje redom svih pozicija u tekstu i utvrđivanje da li se obrazac na tim pozicijama podudara sa tekstom [8]. Funkcija *GrubaMetoda* na takav način pronalazi prvo pojavljivanje obrasca $p[1..M]$ u tekstu $x[1..N]$.

```
function GrubaMetoda: integer;
  var i, j: integer;
  begin
  i := 1; j := 1;
  repeat
    if x[i] = p[j]
          then begin i := i + 1; j := j + 1 end
          else begin i := i − j + 2; j := 1 end;
  until (j > M) or (i > N);
  if j > M then GrubaMetoda:= i − M
          else GrubaMetoda:= i
  end;
```

i i j su pokazivači tekućeg karaktera teksta, odnosno obrasca. Sve dok su tekući karakter teksta i tekući karakter obrasca jednaki, ovi pokazivači se uvećavaju. Ako se stigne do kraja obrasca ($j > M$), onda je obrazac pronađen u tekstu a *GrubaMetoda* dobija vrednost pozicije početka obrasca u tekstu. Ako se tekući karakter teksta i tekući karakter obrasca razlikuju, onda se obrazac na poziciji $i - j + 1$ ne može sravniti sa tekstom pa se obrazac pomera za jednu poziciju udesno u odnosu na tekst ($i = i - j + 2$) a j se postavlja na početak obrasca ($j = 1$). Ako se stigne do kraja teksta ($i > N$), obrazac u tekstu ne postoji a *GrubaMetoda* dobija vrednost $N + 1$.

U primenama ovog algoritma na tekst u prirodnom (ili programskom) jeziku, na primer u uređivačima teksta, unutrašnja petlja u kojoj se uvećavaju brojači i i j se retko izvršava. Pogledajmo sledeći primer: treba pronaći nisku PRAKSI u tekstu

JEDAN PRIMER KOJI POTVRDJUJE LINEARNOST METODE U PRAKSI

Pokazivač j će se uvećati samo tri puta: dva puta za PR iz PRIMER i jednom za P iz POTVRDJUJE. Međutim, razmotrimo slučaj kada je $p = a^{M-1}b$ a $x = a^N$. U tom slučaju se pokazivač j pomera $(m-1)n$ puta da bi se na kraju utvrdilo da se obrazac ne može sravniti sa tekstom.

Šema 1. prikazuje kako algoritam radi u slučaju kada je obrazac $p = abcabcacab$ a $x = babcbabcabcaabcabcabcacabc$.

	babcbabcabcaabcabcabcacabc	neuspeh	nastavak
1.	*abc*	$i = 5,\ j = 4$	$i = 3,\ j = 1$
2.	*abcabca*	$i = 13,\ j = 8$	$i = 7,\ j = 1$
3.	*abca*	$i = 13,\ j = 5$	$i = 10,\ j = 1$
4.	*a*	$i = 13,\ j = 2$	$i = 13,\ j = 1$
5.	*abcabca*	$i = 20,\ j = 8$	$i = 14,\ j = 1$
6.	*abcabcacab*		

Šema 1. Algoritam „grube sile"

Svaki red u ovoj šemi odgovara ulasku u unutrašnju petlju algoritma a svaki karakter u redu uvećanju pokazivača j. Ako pogledamo malo bliže šta se događa između 2. i 3. reda, videćemo da, pošto propadne pokušaj sravnjivanja obrasca na poziciji 6 u tekstu, isprobavaju se i pozicije 7 i 8 dok se ne dođe do delimičnog slaganja iz 3. reda na poziciji 9. To su, očigledno, nepotrebni pokušaji, jer nam delimično sravnjen obrazac iz 2. reda daje dovoljno informacija o tekstu na ovim pozicijama pa njih ne bi trebalo ni isprobavati (na poziciji 7 je b a na poziciji 8 je c). Pokušaj iz 4. reda je takođe nepotreban i mogao bi se izbeći. Podniska *abc* se ponavlja dva puta na početku obrasca. Ako je u 3. redu propao pokušaj sravnjivanja druge podniske *abc* obrasca, nema svrhe na istom mestu pokušavati da se sravni prva podniska *abc* obrasca. Cilj je, očigledno, da se izbegnu ovi nepotrebni pokušaji.

3 Knut-Moris-Pratov algoritam

Osnovna ideja Knut-Moris-Pratovog (KMP) algoritma sastoji se u sledećem [8]: kada se naide na neslaganje tekućeg karaktera teksta i tekućeg karaktera obrasca, „pogrešni pokušaj" se sastoji od karaktera koje unapred poznajemo, jer čine prefiks obrasca. Tu informaciju bi trebalo iskoristiti i ne vraćati pokazivač i preko već poznatih karaktera.

Razmotrimo ponovo primer iz prethodnog odeljka u kome se obrazac $p = abcabcacab$ sravnjivao sa tekstom $x = babcbabcabcaabcabcabcacabc$. Šema 2. prikazuje rad algoritma, koji poznavajući obrazac koji sravnjuje, posle svakog neslaganja preskače preko onih pozicija u tekstu koje ne mogu dovesti do slaganja.

Iz opisa rada ovog algoritma se vidi da algoritam izbegava sve pokušaje za koje je, iz delimično sravnjenog obrasca, unapred jasno da moraju biti neuspešni. Vidi

		neuspeh	nastavak
	babcbabcabcaabcabcabcacabc		
1.	*abc*	$i = 5,\ j = 4$	$i = 5,\ j = 0$
2.	*abcabca*	$i = 13,\ j = 8$	$i = 13,\ j = 5$
3.	*abca*	$i = 13,\ j = 5$	$i = 13,\ j = 1$
4.	*abcabca*	$i = 20,\ j = 8$	$i = 20,\ j = 5$
5.	*abcabcacab*		

Šema 2. Knut-Moris-Pratov algoritam

se, takođe, da se pokazivač teksta i nikada ne vraća unazad. Funkcija *KPMmetod* koristi ovakav algoritam da pronađe prvo pojavljivanje obrasca u tekstu.

```
function KMPmetod : integer;
  var i, j: integer;
  begin
  i := 1; j := 1;
  repeat
   if (j = 0) or (x[i] = p[j])
     then begin i := i + 1; j := j + 1 end
     else begin j := nast[j] end;
  until (j > M) or (i > N);
  if j > M then KMPmetod:= i − M
           else KMPmetod:= i;
  end;
```

Ovaj funkcijski potprogram formalizuje gornje ideje: kada se naiđe na neslaganje teksta sa j-tim karakterom obrasca, $nast[j]$ određuje sa kojim karakterom obrasca treba nastaviti sravnjivanje, ne pomerajući pokazivač teksta. Postavljanje pokazivača j na 0 znači da početak obrasca treba pomeriti iza tekućeg karakatera teksta a pokazivač teksta i uvećati. Dakle, u svakom prolasku kroz **repeat** petlju pomera se ili pokazivač teksta ili obrazac.

Da bi se dokazalo da je ovaj program korektan može se koristiti sledeća invarijanta: Neka je $k = i - j$, tj. neka je k pozicija u tekstu koja prethodi prvom karakteru tekuće pozicije obrasca [7]. Tada je

$$\forall l:\ 1 \le l < j\ x[k+l] = p[l] \wedge$$
$$\forall t:\ 0 \le t < k\ \exists s : 1 \le s \le M,\ x[t+s] \neq p[s].$$

Gornji program će, naravno, biti korektan samo ako se funkcija *nast* izračuna tako da gornja invarijanta važi kada se izvršava operacija $j := nast[j]$. Kada program izvršava operaciju $j := nast[j]$ znamo da je $j > 0$ i da je poslednjih j karaktera teksta, uključujući $x[i]$

$$p[1] \ldots p[j-1]y$$

gde je $y \neq p[j]$. Želimo da odredimo najmanji pomak obrasca za koji bi se ovi karakteri teksta mogli sravniti sa obrascem. Drugim rečima, $nast[j]$ treba da bude

najveće l manje od j takvo da je poslednjih l karaktera teksta

$$p[1] \ldots p[l-1]y$$

i $p[l] \neq p[j]$. Ako takvo l ne postoji, $nast[j]$ se postavlja na 0. Lako se dokazuje da, ako je $nast[j]$ definisano na ovaj način, gornja invarijanta programa važi. Prvi deo invarijante sledi iz definicije $nast[j]$. Treba još dokazati

$$\forall t: i - j \le t < i - nast[j]$$
$$x[t+1] \ldots x[i-1] \neq p[1] \ldots p[i-t-1]$$

Ako bi postojalo t iz $[i-j, i-nast[j])$ takvo da je $x[t+1] \ldots x[i-1] = p[1] \ldots p[i-t-1]$, onda bi bilo $nast[j] < i - t$, što je u suprotnosti sa definicijom $nast[j]$.

Ostaje još da se izračuna funkcija $nast[j]$. Problem bi se lakše rešio kada se u definiciji $nast[j]$ ne bi zahtevalo da bude $p[l] \neq p[j]$. Pogledajmo zato prvo kako se rešava lakši zadatak. Neka je $f[j]$ najveće l manje od j takvo da je $p[1] \ldots p[l-1] = p[j-l+1] \ldots p[j-1]$. Kako je uslov uvek ispunjen za $l = 1$, imamo da je za $j > 1$ uvek $f[j] \ge 1$. Neka je $f[1] = 0$.

Da bi se izračunala funkcija $f[j]$, možemo zamisliti da kopiju prvih $j-1$ karaktera obrasca pomeramo iznad same sebe s leva nadesno, počev od prvog karaktera kopije postavljene iznad drugog karaktera obrasca. Zaustavljamo se kada se podudare svi karakteri koji se preklapaju, ili kada ustanovimo da takvog podudaranja nema. Broj karaktera koji se preklapaju uvećan za jedan daje vrednost $f[j]$. Na primeru obrasca $p = abcabcacab$ dobijaju se sledeće vrednosti:

j	=	1	2	3	4	5	6	7	8	9	10
$p[j]$	=	*a*	*b*	*c*	*a*	*b*	*c*	*a*	*c*	*a*	*b*
$f[j]$	=	0	1	1	1	2	3	4	5	1	2

Lako se uočava da ako je $p[j] = p[f[j]]$, onda je $f[j+1] = f[j] + 1$. Zanimljivo je da se, ako je $p[j] \neq p[f[j]]$, za računanje $f[j+1]$ može primeniti potpuno isti algoritam kao i za sravnjivanje obrasca i teksta. Uočava se sličnost problema računanja $f[j]$ i invarijante algoritma sravnjivanja: algoritam računa najveće j manje od k za koje je $p[1] \ldots p[j-1] = x[k-j+1] \ldots x[k-j]$. S toga je i program za izračunavanje niza $f[j]$ u osnovi isti kao i algoritam sravnjivanja, samo što se obrazac p sravnjuje sa samim sobom.

```
procedure InitNast;
  var i, j: integer;
  begin
  i := 1; j := 0; f[1] := 0;
  repeat
    if (j = 0) or (p[i] = p[j])
      then begin
           i := i + 1; j := j + 1; f[i] := j
```

```
        end
    else begin j := nast[j] end;
until i > M;
end;
```

Procedura *InitNast* pretpostavlja da je funkcija *nast* već poznata. Ako se uporede definicije funkcija f i *nast* vidi se da je za $j > 1$

$$nast[j] = \begin{cases} f[j], & \text{ako je } p[j] \neq p[f[j]]; \\ nast[f[j]], & \text{ako je } p[j] = p[f[j]]. \end{cases}$$

Da bi se izračunala funkcija *nast*, proceduru *InitNast* treba preraditi, izostavljajući sasvim računanje funkcije f tako što iskaz $f[i] := j$ treba zameniti sa

```
if p[j] <> p[i] then nast[i] := j
                else nast[i] := nast[j];
```

3.1 Efikasnost

Efikasnost KMP algoritma, kao i ostalih algoritama za sravnjivanje niski utvrđuje se, s jedne strane, u odnosu na broj karaktera teksta koji se ispituju i, s druge strane, u odnosu na broj izvršenih naredbi programa (elementarnih koraka algoritma). KMP algoritam ispituje svaki karakter teksta tačno jednom dok ne sravni obrazac, odnosno utvrdi da obrazac u tekstu ne postoji. Procedura *KMPmetod* ima osobinu da se operacija $j := nast[j]$ ne izvršava češće od operacije $i := i+1$; šta više, ona se obično izvršava ređe jer se, u opštem slučaju, obrazac pomera udesno ređe od pokazivača teksta. Slično važi i za proceduru *InitNast*. Operacija $j := nast[j]$ uvek pomera gornju kopiju obrasca udesno pa se može izvršiti najviše M puta. Iz ovoga sledi da KMP algoritam pronalazi obrazac dužine M u tekstu dužine N u $O(N + M)$ jedinica vremena.

Osim toga, Knut, Moris i Prat su dokazali [7] da je broj izvršavanja operacije $j := nast[j]$, pre nego što se pokazivač i pomeri, ograničen aproksimacijom funkcije $log_\phi M$, gde je ϕ zlatan presek ($(1 + \sqrt{5})/2 \approx 1.618$). Ova funkcija je i najbolja gornja granica. Da bi ovo dokazali koristili su Fibonačijeve niske koje se definišu na sledeći način:

$$\begin{aligned} \phi_1 &= b, \\ \phi_2 &= a, \\ \phi_n &= \phi_{n-1}\phi_{n-2}, \quad \text{za } n \geq 3 \end{aligned}$$

a koje su izgledale kao patološki obrazac za njihov algoritam. Oni su pokazali da se za Fibonačijeve niske, skanirajući jedan karakter teksta, operacija $j := nast[j]$ najviše izvršava približno $log_\phi M$ puta. Pokazali su, takođe, da su Fibonačijeve niske najgori slučaj, što znači da je $log_\phi M$ i gornja granica. Iz ovoga sledi zaključak

da ako se tekst učitava iz spoljašnje teke, između učitavanja dva susedna karaktera protekne samo $O(logM)$ jedinica vremena.

Sami autori Knut-Moris-Pratovog algoritma su uočili da njihov algoritam neće biti značajno efikasniji od algoritma „grube sile“ u većini realnih aplikacija. U takvim aplikacijama se, uglavnom, ne sravnjuje samoponavljajući obrazac sa veoma samoponavljajućim tekstom. Međutim, sa praktične strane, algoritam ima veoma korisnu osobinu. Algoritam sekvencijalno prolazi kroz ulaz i nikad se ne vraća unazad, što ga čini veoma pogodnim za korišćenje na tekstovima koji se učitavaju iz spoljašnjih teka. U takvim slučajevima, u algoritme koji ne isključuju vraćanje unazad mora da bude ugrađena složena tamponaža. Osim toga, postoje načini za značajno ubrzanje ovog algoritma kao i mogućnost njegovog proširenja na složenije obrasce.

3.2 Poboljšanja i proširenja

U uređivačima teksta obrasci su obično kratki, tako da je najefikasnije da se obrazac direktno prevede u mašinski kod koji implicitno sadrži funkciju *nast*. Na primer, sledeći program je ekvivalentan programu *KMPmetod* za obrazac $p = abcabcacab$, ali je mnogo efikasniji:

```
      i := 0;
 0 :  i := i + 1;
 1 :  if x[i] <>'a' then goto 0;   i := i + 1;
 2 :  if x[i] <>'b' then goto 1;   i := i + 1;
 3 :  if x[i] <>'c' then goto 1;   i := i + 1;
 4 :  if x[i] <>'a' then goto 0;   i := i + 1;
 5 :  if x[i] <>'b' then goto 1;   i := i + 1;
 6 :  if x[i] <>'c' then goto 1:   i := i + 1;
 7 :  if x[i] <>'a' then goto 0;   i := i + 1;
 8 :  if x[i] <>'c' then goto 5;   i := i + 1;
 9 :  if x[i] <>'a' then goto 0;   i := i + 1;
 10 : if x[i] <>'b' then goto 1;   i := i + 1;
      Metod := i - 8;
```

Obeležja u goto iskazima podudaraju se sa funkcijom *nast*. Šta više, procedura *InitNast* koja računa funkciju *nast* može se modifikovati tako proizvede ovakav program. Da bi se izbegla provera $i > N$ svaki put kada se pokazivač i pomeri, sam obrazac može se dopisati na kraj teksta, u $x[N + 1..N + M]$. (Ovo poboljšanje može se primeniti i u standardnoj proceduri *KMPmetod*).

Ovakav program može se opisati pomoću veoma jednostavnog modela, konačnog automata. Ovaj konačni automat se sastoji od stanja, koja su karakteri obrasca, i prelaza iz stanja u stanje. U svakom stanju moguća su dva prelaza: prelaz u slučaju uspeha sravnjivanja (kada su tekući karakteri obrasca i teksta jednaki) i prelaz u slučaju neuspeha sravnjivanja (kada oni nisu jednaki). U slučaju kada su ovi karakteri jednaki, pokazivač teksta se pomera.

Ovakvo tumačenje KMP algoritma omogućava njegovo uopštavanje na slučaj kada je obrazac konačan skup više ključnih reči, tj. kada se više ključnih reči paralelno sravnjuje: sravnjivanje je uspešno kada se u tekstu pronađe prvo pojavljivanje jedne od ključnih reči obrasca. Jedno moguće rešenje je da se za svaku ključnu reč formira funkcija *nast* i da se za njega održava pokazivač j. Ovakvo rešenje, međutim, za sravnjivanje obrasca od k ključnih reči čija je ukupna dužina M zahteva $O(kN + M)$ jedinica vremena. Korišćenjem istih ideja koje su u osnovi KMP algoritma, može se konstruisati algoritam koji sravnjuje obrazac od k ključnih reči u $O(N + M)$ jedinica vremena. U $O(M)$ jedinica vremena se prvo konstruiše automat za sravnjivanje obrasca koji se sastoji od sledećih komponenti [1]:

1. S, konačan skup stanja,
2. Σ, konačna ulazna azbuka,
3. $napr: S \times \Sigma \to S \cup \{otkaz\}$, funkcija prelaza napred,
4. $nast: S - \{s_0\} \to S$, funkcija prelaza nazad,
5. s_0, početno stanje,
6. F, skup završnih stanja.

Da bi se konstruisala funkcija prelaza napred, ključne reči se kombinuju u trai (engl. *trie*), čiji čvorovi predstavljaju prefikse jedne ili više ključnih reči, a grane specifikuju funkciju *napr* kao funkciju tekućeg stanja i tekućeg karaktera teksta. Funkcija prelaza nazad se konstruiše tako da zadovolji sledeći uslov: Ako stanja s i t predstavljaju prefikse u i v nekih ključnih reči, tada je $f[s] = t$ ako i samo ako je v najduži sufiks od u koji je ujedno i prefiks neke ključne reči. Da bi se izbegla nepotrebna vraćanja unazad, uvođenjem dodatnog uslova,

$$S_{f[s]} = \{s \in S \mid g(f[s], a) \neq \text{otkaz}\}$$
$$S_s = \{s \in S \mid g(s, a) \neq \text{otkaz}\}$$

$$nast[s] = \begin{cases} nast[s] = nast[f[s]], & S_{f[s]} \subset S_s \\ nast[s] = f[s], & \text{inače} \end{cases}$$

iz funkcije f može se konstruisati funkcija *nast*. Analogija sa funkcijama f i *nast* iz KMP algoritma je očigledna. Važno je da se, kao i u slučaju KMP algoritma, funkcije f i *nast* mogu konstruisati u $O(M)$ jedinica vremena. U slučaju obrasca $p = \{abcab, ababc, bcac, bbc\}$, automat bi se mogao predstaviti Tabelom 1.

Ako se automat, odnosno funkcije g i *nast* definišu iz obrasca na ovaj način, program za prepoznavanje obrasca je vrlo jednostavan. Celobrojni niz g ima vrednost -1 u slučaju otkaza, logički niz *kraj* ima vrednost **true** ako je stanje završno a vrednost niza *duž* je dubina stanja. Ovi nizovi se, kao i niz *nast*, formiraju u fazi konstruisanja automata.

		g				
čvor	prefiks	a	b	c	f	*nast*
0		1	9	0		
1	a	ot	2	ot	0	0
2	ab	3	ot	6	9	9
3	aba	ot	4	ot	1	0
4	$abab$	ot	ot	5	2	2
5	$ababc$	ot	ot	ot	6	6
6	abc	7	ot	ot	12	0
7	$abca$	ot	8	ot	13	13
8	$abcab$	ot	ot	ot	2	2
9	b	ot	10	12	0	0
10	bb	ot	ot	11	9	9
11	bbc	ot	ot	ot	12	12
12	bc	13	ot	ot	0	0
13	bca	ot	ot	14	1	1
14	$bcac$	ot	ot	ot	0	0

Tabela 1. Rad automata sa obrascem p.

```
function Automat: integer;
  var i, stanje: integer;
  begin
  i := 1; stanje := 0;
  repeat
    if g[stanje, x[i]] <> -1
      then begin
             i := i + 1; stanje := g[stanje, x[i]]
           end
      else stanje :=nast[stanje] end
  until (not kraj[stanje]) or (i > N);
  if kraj[stanje] then Automat:=i-duž[stanje]
                  else Automat:=i
  end;
```

Sličnost ove procedure sa funkcijom *KMPmetod* je očigledna: *stanje* zamenjuje pokazivač j, pomeranje pokazivača se zamenjuje funkcijom g a uslov $x[i] = p[j]$ uslovom da funkcija g za tekući karakter teksta i tekuće stanje ne otkazuje.

Ovaj algoritam za sravnjivanje obrasca je korišćen u sistemima za pretraživanje bibliografije u kojima su korisnici sistema zadavali ključne reči (i njihove bulovske kombinacije) po kojima su zahtevali pretraživanje [2]. Varijanta ovog algoritma je korišćena za identifikovanje funkcionalnih reči, prefiksa, sufiksa itd. u korpusu pisanih tekstova [9].

4 Boaje-Morov algoritam

R. S. Boyer i J. S. Moore su u [3] opisali algoritam za sravnjivanje niski, koji je, ako vraćanje unazad po tekstu nije problem, u velikom broju slučajeva značajno brži od do sada opisanih algoritama i obrazac sravnjuje u sublinearnom vremenu.

Osnovna ideja ovog algoritma je da se više informacija može dobiti ako se obrazac sravnjuje sa tekstom s desna ulevo, umesto s leva udesno. Zamislimo da je obrazac p dužine M postavljen iznad teksta x dužine N tako da je početak obrasca iznad prvog karaktera teksta. Prvo se upoređuju karakteri $p[M]$ i $x[M]$. Moguće su sledeće situacije:

a) Karakteri $p[M]$ i $x[M]$ nisu jednaki i karakter $kr = x[M]$ se ne pojavljuje nigde u obrascu. Tada nema potrebe da se početak obrasca postavlja na pozicije $2, 3, \ldots, M$, jer se ni na jednoj od tih pozicija obrazac ne može sravniti sa tekstom.

b) Karakteri $p[M]$ i $x[M]$ nisu jednaki a krajnje desno pojavljivanje karaktera $kr = x[M]$ u obrascu je udaljeno *skok* karaktera od kraja obrasca. Tada znamo da obrazac možemo odmah da pomerimo za *skok* mesta udesno.

Prema tome, ako tekući karakter teksta $kr = x[i]$ nije jednak poslednjem karakteru obrasca, može se preskočiti *skok* karaktera teksta, ne ispitujući uopšte preskočene karaktere. Veličina *skok* je funkcija karaktera kr i definiše se na sledeći način: Ako se karakter kr ne pojavljuje u obrascu, $skok[kr] = M$; inače je $skok[kr]$ razlika između dužine obrasca M i krajnje desne pozicije karaktera kr u obrascu.

c) Karakteri $p[M]$ i $x[M]$ su jednaki. Tada moraju da se uporede pretposlednji karakter obrasca i prethodni karakter teksta. Ako su i oni jednaki nastavlja se sa pomeranjem i pokazivača teksta i pokazivača obrasca sve dok se ili ne stigne do početka obrasca, čime je obrazac sravnjen sa tekstom, ili se naiđe na neslaganje karaktera $kr = x[M-j]$ i $p[M-j]$. U tom slučaju mogu se opet koristiti iste ideje kao u slučajevima a) i b): obrazac se pomera udesno za k pozicija da bi se izravnala dva pojavljivanja karaktera kr, pokazivač teksta se pomera za $k+j$ karaktera a pokazivač j se ponovo postavlja na kraj obrasca. Rastojanje k za koje treba pomeriti obrazac udesno zavisi od pozicije karaktera kr u obrascu. Ako je $skok[kr] < M-j$, tj. ako je tekući karakter obrasca ispred pozicije krajnje desnog pojavljivanja karaktera kr u obrascu, obrazac bi se morao pomeriti unazad da bi se izravnali karakteri $x[M-j]$ i $p[M-skok]$. To, naravno nije poželjno jer može predstavljati ulazak u beskonačnu petlju, te u ovom slučaju funkcija *skok* nije od koristi, pa se obrazac pomera za $k = 1$ karakter a pokazivač teksta za $M+1$ karaktera. Ako je, pak, $skok[kr] > M-j$, tj. ako je krajnje desno pojavljivanje karaktera kr u obrascu ispred tekućeg karaktera obrasca, obrazac se može pomeriti unapred za $k = skok[kr]-j$ karaktera a pokazivač teksta za $skok[kr]-j+j$ karaktera.

Na Šemi 3. je prikazano kako bismo, koristeći ovako opisan algoritam, pronašli nisku PRAKSI u tekstu. Iz ovog primera vidimo da je za pronalaženje obrasca dužine 6 karaktera u tekstu dužine 55 karaktera bilo potrebno ispitati samo 9 karaktera teksta (plus još šest da bi se ustanovilo sravnjivanje).

U funkciji *BMmetod* su neposredno primenjene ove neformalno izložene ideje. Niz *skok*, koji određuje pomeranje pokazivača teksta u slučaju razlikovanja tekućeg


```
JEDAN PRIMER KOJI POTVRDJUJE LINEARNOST METODE U PRAKSI
PRAKSI
      PRAKSI
         PRAKSI
            PRAKSI
               PRAKSI
                  PRAKSI
                     PRAKSI
                        PRAKSI
                           PRAKSI
                              PRAKSI
                                 PRAKSI
```

Šema 3. Sravnjivanje niske PRAKSI Boaje-Morovim algoritmom

karaktera obrasca i teksta, dobija se preprocesiranjem obrasca p. On je veličine ulazne azbuke i određuje ga procedura *InitSkok*.

```
function BMmetod: integer;
  var i, j, ind: integer;
  begin
  i := M; j := M;
  repeat
    if x[i] = p[j]
      then begin i := i - 1; j := j - 1 end
      else begin
             i := i + max(M - j + 1, skok[ord(x[i])]);
             j := M
           end
  until (j < 1) or (i > N);
  BMmetod:= i + 1
  end;

procedure InitSkok;
  var j: integer;
  begin
    for j := 0 to 127 do skok[j] := M;
    for j := 1 to M do skok[ord(p[j])] := M - j
  end;
```

BMmetod i *InitSkok* se zasnivaju na originalnoj ideji Boajea i Mora da se obrazac sravnjuje sa tekstom s desna ulevo. Oni su, međutim, predložili dalje poboljšanje svog algoritma koje se zasniva na sličnim idejama koje su ugrađene i u Knut-Moris-Pratov algoritam.

Opišimo ove ideje prvo neformalno. Neka je $M-j$ karaktera teksta x sravnjeno sa završnih $M-j$ karaktera obrasca p. Označimo ovu podnisku sa *subp*. Neka, takođe, ovom pojavljivanju *subp* u tekstu prethodi karakter kr koji se razlikuje od

karaktera koji u obrascu p prethodi sufiksu $subp$. Tada želimo, grubo govoreći, da pomerimo obrazac p udesno za onoliko karaktera koliko je potrebno da bi se poravnao najduži sufiks otkrivene podniske $subp$ sa njegovim krajnjim desnim pojavljivanjem u obrascu, kome ne prethodi isti karakter kao i njegovom završnom pojavljivanju.

U tom slučaju, dakle, želimo da pomerimo obrazac za k karaktera udesno, pri čemu k zavisi od pozicije u obrasu p krajnje desnog ponovnog pojavljivanja sufiksa obrasca p dužine $l \leq M - j$, kome ne prethodi isti karakter kao i sufiksu. Tada želimo da počnemo sa ispitivanjem karaktera teksta koji je poravnat sa poslednjim karakterom obrasca, što znači da se pokazivač teksta pomera za $k+M-j$ karaktera. Nazovimo ovo rastojanje $sskok$ a definišemo ga kao funkciju pozicije j u obrascu na kojoj je došlo do neslaganja.

U novoj verziji algoritma, pošto je $M - j$ karaktera obrasca p sravnjeno sa tekstom pre nego što je došlo do neslaganja, pomeramo pokazivač teksta za $1+M-j$ ili $skok[kr]$ ili $sskok[j]$ karaktera, u zavisnosti od toga koji od njih ga dalje pomera. Kako je $sskok[j]$ po definiciji $k+M-j$, a uvek je $k \geq 0$, sledi da je uvek $sskok[j] \geq 1+M-j$. Stoga se pokazivač teksta pomera za $max(skok[kr], sskok[j])$ karaktera.

Knut, Moris i Prat su dali preciznu definiciju finkcija $skok$ i $sskok$ i definisali su efikasan algoritam za izračunavanje funkcije $sskok$ [7].

Za svaki karakter kr iz azbuke je

$$skok[kr] = min\{k \mid k = M \vee (0 \leq k < M \wedge p[M-k] = kr)\},$$

a za svako j, $1 <= j < M$ je

$$sskok[j] = min\{k + M - j \mid k \geq 1 \wedge (k \geq j \vee p[j-k] \neq p[j]) \wedge ((k \geq i \vee p[i-k] = p[i]) \text{ za } j < i \leq M)\}$$

Da bismo primenili novu funkciju $sskok$, u proceduri $BMmetod$ treba samo promeniti iskaz kojim se pomera pokazivač teksta:

$$i := i + max(skok[ord(kr)], sskok[j]);$$

Procedura $InitSskok$ izračunava funkciju $sskok$ u $O(M)$ koraka.

```
procedure InitSskok;
  var i, j, k, l: integer;
  begin
  for k := 1 to M do sskok[k] := 2 * M - k;
  j := M; l := M + 1; f[M] := M + 1;
  repeat
    if (l > M) or (p[j] = p[l])
     then begin
            l := l - 1; j := j - 1
            f[j] := l
          end
     else begin
            sskok[l] := min(sskok[l], M - j);
            l := f[l]
          end
  until (j <= 0);
  for k := 1 to l do
    sskok[k] := min(sskok[k], M + l - k);
  end;
```


Na primer, za obrazac $p = badbacbacba$ dobijaju se sledeće vrednosti funkcija f i $sskok$:

j	=	1	2	3	4	5	6	7	8	9	10	11
$p[j]$	=	b	a	d	b	a	c	b	a	c	b	a
$f[j]$	=	10	11	6	7	8	9	10	11	11	11	12
$sskok[j]$	=	19	18	17	16	15	8	13	12	8	12	1

4.1 Efikasnost

Boaje i Mor su iscrpno testirali svoju implementaciju algoritma na računaru PDP-10 na tri razne vrste „tekstova" dužine 10000 karaktera: prvi je slučajna sekvencija 0 i 1, drugi slučajno izabrani tekst na Engleskom jeziku a treći slučajna sekvencija karaktera iz azbuke od 100 karaktera. Iz svakog od ovih tekstova je programski izabrano 300 obrazaca dužine M, za svako M od 1 do 14. Algoritam je zatim primenjen na sravnjivanje ovih obrazaca u tekstu, počev od neke slučajno izabrane pozicije u tekstu.

Za utvrđivanje efikasnosti algoritma korišćena su dva pokazatelja: jedan je odnos broja ispitanih karaktera teksta prema broju pređenih karaktera teksta dok se obrazac ne sravni (ili se ne stigne do kraja teksta) a drugi odnos broja izvršenih mašinskih instrukcija prema broju pređenih karaktera teksta.

Prvi pokazatelj, koji ne zavisi od konkretne implementacije algoritma, je pokazao da je broj ispitanih karaktera teksta po karakteru manji od 1, i smanjuje se sa povećanjem dužine obrasca. Tako je, na primer, za engleski obrazac dužine 5 prosečno ispitano 0,24 karaktera po karakteru, što znači da algoritam za svaki ispitani karakter teksta prelazi preko (približno) 4 karaktera. Za engleski obrazac dužine 14 ispituje se prosečno 0,11 karaktera. Ovi empirijski podaci su potvrdili da je algoritam sublinearan u odnosu na broj ispitanih karaktera teksta. Ovaj pokazatelj je korisno uporediti sa istim pokazateljem za KMP algoritam i za algoritam „grube sile": KPM algoritam ispituje tačno jedan karakter po pređenom karakteru, dok empirijski podaci za tekst na prirodnom jeziku pokazuju da algoritam „grube sile" ispituje približno 1,1 karakter po pređenom karakteru.

Drugi pokazatelj, koji zavisi od konkretne implementacije algoritma, je pokazao da je ukupan broj izvršenih instrukcija po pređenom karakteru manji ukoliko je dužina obrasca veća. Osim toga, ukoliko je azbuka dovoljno velika a obrazac dovoljno

dugačak, algoritam izvršava manje od jedne instrukcije po pređenom karakteru. Tako, na primer, za engleski tekst algoritam izvršava manje od jedne instrukcije za obrasce duže od 5 karaktera. Ovo praktično znači da ni jedan algoritam koji ispituje bar jedan karakter po pređenom karakteru u ovakvim slučajevima ne može biti brži od BM algoritma. KMP algoritam, implementiran na efikasan način opisan u tački 3.2, uzima svaki karakter teksta tačno jednom, ali se svaki karakter teksta može porediti sa više karaktera obrasca. Svako ovakvo poređenje zahteva bar tri instrukcije a broj ovakvih poređenja je, kao što je rečeno u tački 3.1, ograničen sa $log_\phi M$. Prema tome, za KMP algoritam, broj izvršenih instrukcija po pređenom karakteru je najmanje $3-p$, gde je p verovatnoća da je karakter teksta jednak tekućem karakteru obrasca. Empirijski rezultati pokazuju da efikasno kodiran algoritam „grube sile" zahteva prosečno 3,3 instrukcije po pređenom karakteru.

Ovi empirijski rezultati, kao i probabilistički model koji su Boaje i Mor izradili da bi analizirali prosečno ponašanje algoritma, su pokazali da je algoritam sublinearan u tipičnim slučajevima. Ponašanje algoritma u najgorem slučaju nije lako utvrditi: to potiče otuda što BM algoritam kad god pomeri obrazac udesno „zaboravlja" sve što je saznao o do tada sravnjenim karakterima. U [7] je pokazano da je, u slučaju kada se obrazac ne nalazi u tekstu, BM algoritam linearan i u najgorem slučaju i da je gornja granica ukupnog broj poređenja karaktera teksta sa karakterima obrasca $7N$. U [6] je pokazano da se ova gornja granica može smanjiti na $4N$.

LITERATURA

[1] Aho, A.V.: *Pattern Matching in Strings,* Formal Language Theory, Academic Press, 1980, pp. 325–347.

[2] Aho, A.V., Corasick, M.J.: *Efficient String Matching: An Aid to Bibliographic Search,* Communications of the ACM 18(6), June 1975, pp. 333–340.

[3] Boyer, R.S., Moore, J.S.: *A Fast String Searching Algorithm,* Communications of the ACM 20(10), October 1977, pp. 762–772.

[4] Davies, D.J.M.: *String Searching in Text Editors,* Software— Practice and Experience 12, 1982, pp. 709–717.

[5] Farber, D.J., Griswold, R.E., Polonsky, I.P.: *SNOBOL, A String Manipulation Language,* Journal of the ACM 11(2), January 1964, pp. 21–30.

[6] Guibas, L.J., Odlyzko, A.M.: *A new Proof of the Linearity of the Boyer-Moore String Searching Algorithm,* SIAM J. Computing 9(4), November 1980, pp. 672–682.

[7] Knuth, D.E., Morris, J.H., Pratt, V.R.:*Fast Pattern Matching in Strings,* SIAM J. Computing 6(2), June 1977, pp. 323–350.

[8] Sedgewick, R.: *Algorithms,* Addison-Wesley Publishing Company, 1983.

[9] Vitas, D.: *Prikaz jednog sistema za automatsku obradu teksta,* Zbornik radova sa Symp. Informatica 79, Bled, oktobar 1979.

Matematički fakultet
Studentski trg 16
11000 Beograd

RAČUNARSTVO
Sveska 1, Broj 2, 1992

UDK 519.688

KUREPINA HIPOTEZA O LEVOM FAKTORIJELU[1]

GORAN GOGIĆ

SAŽETAK. Na kongresu jugoslovenskih matematičara u Ohridu 1970. godine Đuro Kurepa je postavio sledeći problem: definišimo levi faktorijel kao sumu faktorijela svih nenegativnih celih brojeva manjih od datog broja; postavlja se pitanje šta može biti zajednički delilac levog i desnog faktorijela nekog broja. Kurepa postavlja hipotezu da je dvojka najveći zajednički delilac oba ova broja. Mada je problem veoma jednostavan, i na prvi pogled se čini da ne bi trebalo biti nekih poteškoća pri rešavanju, on do danas, 20 godina od nastanka, nije rešen.

1 Neki iskazi ekvivalentni Kurepinoj hipotezi

U cilju pojednostavljenja problema, pojavili su se mnogi iskazi ekvivalentni iskazu Kurepine hipoteze. Iskaz (E1) koji Kurepa navodi u [5] biće nam od velike pomoći pri proveri (KH) na računaru.

DEFINICIJA 1. *Levi faktorijel prirodnog broja n (u oznaci !n) definišemo kao*

$$!n = \sum_{i=0}^{n-1} i! = 0! + 1! + \cdots + (n-1)!.$$

DEFINICIJA 2. *Kurepina hipoteza je iskaz*

$$(KH) \qquad (\forall n > 1)\,\mathrm{NZD}(!n, n!) = 2$$

DEFINICIJA 3. *Neka je P skup prostih brojeva. Iskaz (E1) je*

$$(E1) \qquad (p \in P)(p > 2 \Rightarrow !p \not\equiv 0 \ (mod\, p))$$

TEOREMA 1. *Iskazi (KH) i (E1) su ekvivalentni.*

[1]Rad finansiran od Fonda za nauku Republike Srbije preko Matematičkog instituta, projekat 0401A.

DOKAZ:

- $\neg(E1) \Rightarrow \neg(KH)$ Neka je $p \in P$ takav da je $p > 2$ i $!p \equiv 0 \pmod p$. Međutim, pošto je p prost, to znači da $p|!p$, a kako $p|p!$ onda je $\mathrm{NZD}(!p, p!) \geq p > 2$ i za $n = p$ dobijamo $\mathrm{NZD}(!n, n!) \neq 2$ što je negacija od (KH).
- $\neg(KH) \Rightarrow \neg(E1)$ Pošto je (KH) tačno za $n \leq 4$, to onda mora za neko $n > 4$ da važi $\mathrm{NZD}(!n, n!) = d$ i $d \neq 2$. Sada je

$$\begin{aligned} !n &= 0! + 1! + 2! + \cdots + (n-1)! \\ &= 10 + 4! + \cdots + (n-1)! \equiv 2 \pmod 4 \end{aligned}$$

a pošto $4|n!$ za $n > 4$, to onda $2|d$ i $4!d$ odakle je $d \neq 1$ a zbog $d \neq 2$ zaključujemo da d ima neki prost neparan delilac. Neka je $p > 2$ prost broj takav da $p|d$. Pošto $p|n! = 1 \cdot 2 \cdot \ldots \cdot n$ to je $p \leq n$. Međutim, $!n = !p + p! + (p+1)! + \cdots + (n-1)! \equiv !p \pmod p$ pa kako $p|!n$ to $p|!p$ što je u stvari negacija iskaza (E1). Ovim je teorema dokazana.

2 Opšta rekurentna formula

Pošto smo navedenom teoremom smanjili domen na kome treba ispitati tačnost (KH), ostaje da nađemo najlakši način za računanje vrednosti funkcije levog faktorijela.

TEOREMA 2. *Neka su funkcije* $\mathrm{V}: N \times N \to N$ *i* $\mathrm{W}: N \times N \to N$ *definisane na sledeći način:*

$$\mathrm{V}(n,k) = \prod_{i=0}^{k+1}(n-i) = n(n-1)\ldots(n-k+1) = n!/k!$$

$$\mathrm{W}(n,k) = \sum_{i=1}^{k} \mathrm{V}(n-i, k-i)$$

Neka su dalje dati prirodni brojevi k *i* n *pri čemu je* $k < n$ *i neka je* $l = [n/k]$ *i* r *takvo da je* $n = kl + r$, $0 \leq r < n$. *Ako je niz* S $(0 \leq i \leq l)$ *definisan na sledeći način:*

$$\begin{aligned} S_0 &= \mathrm{V}(n,k), \\ S_i &= S_{i-1} \cdot \mathrm{V}(n-ik, k) + \mathrm{W}(n-ik, k), \quad 1 \leq i < l, \\ S_l &= S_{l-1} \cdot r! + !r, \end{aligned}$$

tada je $S_l = !n$.

DOKAZ: Neka je $k \in N$ takvo da je $k < n$. Tada je

$$\begin{aligned} !n - !(n-k) &= \sum_{i=1}^{n}(n-i)! - \sum_{i=1}^{n-k}(n-k-i)! \\ &= \sum_{i=1}^{n}(n-i)! - \sum_{i=k+1}^{n}(n-i)! \\ &= \sum_{i=1}^{k}(n-i)!, \end{aligned}$$

a pošto je $(n-i)! = \mathrm{V}(n-i, k-i) \cdot (n-k)!$ to je onda

$$\begin{aligned} !n - !(n-k) &= \sum_{i=1}^{k}(n-i)! \\ &= \sum_{i=1}^{n}\mathrm{V}(n-i, k-i) \cdot (n-k)! \\ &= (n-k)! \sum_{i=1}^{k}\mathrm{V}(n-i, k-i) = (n-k)!\,\mathrm{W}(n,k) \end{aligned}$$

Sada izvodimo formulu

$$\begin{aligned} !n &= \sum_{i=0}^{l-1}(!(n-ik) - !(n-(i+1)k) + !(n-lk) \\ &= \sum_{i=0}^{l-1}(!(n-ik) - !(n-ik-k)) + !r \\ &= \sum_{i=0}^{l-1}(n-(i+1)k)!\,\mathrm{W}(n-(i+1)k, n-ik) + !r \end{aligned}$$

Uočimo još da za prirodne brojeve $a, b, A, B \in N$ takve da je $a > b$ važi

$$\begin{aligned} a! \cdot A + b! \cdot B &= b!(a(a-1)\ldots(b+1) \cdot A + B) \\ &= b!(\mathrm{V}(a, a-b) + B) \qquad \text{pa je sada} \\ !n &= (\ldots(\mathrm{W}(n,k) \cdot \mathrm{V}(n-k,k) + \mathrm{W}(n-k,k)) \cdot \mathrm{V}(n-2k,k) \\ &\quad + \mathrm{W}(n-2k,k)) \cdot \mathrm{V}(n-3k,k) + \cdots + \mathrm{W}(n-(l-1)k,k)) \cdot r! + !r \end{aligned}$$

pa sada navedeni niz ima tu osobinu da je $S_l = !n$.

3 Algoritam za proveru Kurepine hipoteze

Proveru Kurepine hipoteze izvodimo koristeći ranije izvedene relacije. Domen na kome vršimo ispitivanje je skup svih prostih brojeva, jer smo dokazali ekvivalentnost

(KH) sa iskazom (E1). Tako, ako (KH) važi za neke proste brojeve $p_1, p_2, \ldots, p_k$ onda (KH) važi za sve brojeve oblika $p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_k^{\alpha_k}$. U računanju levog faktorijela koristimo rekurentnu formulu iz prethodne teoreme za $k = 1$. Pošto je nama potrebna samo informacija o deljivosti broja $!p$ sa p, to je dovoljno sve operacije izvršavati u prstenu $Z = \{0, 1, \ldots, p-1\}$. Na taj način ćemo izbeći problem rada sa velikim brojevima, jer tada možemo izvršavati operacije hardverski implementirane u računaru. Paralelizam možemo ostvariti na više načina. Jedna mogućnost je da sve procesore angažujemo na izračunavanju vrednosti $!n \bmod n$. Tu se pojavljuje problem gubljenja vremena jer sekvencijalan algoritam za izračunavanje niza se ne može idealno paralelizovati. Efikasniji je drugi metod: svakom procesoru dodelimo po jedno n za koje treba ispitati (KH).

Algoritam KH1.

```
{Ulaz :   granice intervala [a,b] u kojem vršimo proveru}
{Izlaz:   informacija o tačnosti (KH)}
{Arhitektura: procesori P_0, P_1 , ... ,P_{k-1} }
01   DO in parallel 0 ≤ i ≤ k − 1
02       P_1 :  m = π(a) + 1 + i
03              WHILE p_m ≤ b DO
04              s = 1
05              FOR i = p_m − 1 DOWNTO 1 DO
06                  s = s ∗ i + 1 mod p_m
07              IF s = 0 THEN print:"KONTRAPRIMER"
08              m = m + i
```

Izračunajmo sada $T_1(n)$, vreme potrebno za proveru hipoteze u intervalu $[1..n]$. Pošto se FOR ciklus (05–06) izvršava u vremenu $O(p_m)$, to se primenom Stiltjesovog integrala može dobiti konačna formula:

$$\begin{aligned} T_1(n) &= (1/k) \sum p_i \\ &= (1/k) \int \ln x \, d\pi(x) \\ &= (1/k) x\pi(x) - (1/k) \int \pi(x) dx (1/k) x\pi(x) \\ &= \frac{x^2}{(k \ln x)} \end{aligned}$$

Dakle $T_1(n) = O(\frac{x^2}{k \ln x})$ gde je k broj procesora.

4 Realizacija algoritma na računaru

Navedeni algoritam je realizovan na transpjuterskoj ploči sa četiri transpjutera T800. Program je pisan u jeziku 3L Parallel FORTRAN za transpjutere. Posao je

obavljen tako što je prvo napravljena baza prostih brojeva manjih od 1000000, a zatim je svaki procesor ispitivao tačnost iskaza (E1) za odgovarajuću grupu prostih brojeva koja mu je dodeljena. U određenim vremenskim intervalima glavni procesor je skupljao informacije o dobijenim rezultatima svakog procesora. Konačan rezultat nije doneo nikakvu promenu u odnosu na dosadašnje rezultate. Za sve proste brojeve manje od milion iskaz (E1) odnosno (KH) je tačan. Preciznije rečeno, svi prirodni brojevi koji nemaju prost delilac veći od milion zadovoljavaju iskaz Kurepine hipoteze. Ranije provere Kurepine hipoteze su vršili Slavić za $n \leq 1000$, Wagstaff za $n \leq 50000$ i Mijajlović za $n \leq 310009$.

LITERATURA

[1] S. G. Akl, *The design and analysis of parallel algorithms,* Prentice Hall International, 1989.

[2] R. Guy *Unsolved problems in number theory,* Springer–Verlag, 1981.

[3] D. Kurepa, *On some new factorial propositions,* Mathematica Balkanica 4 (1974), 383–386.

[4] Ž. Mijajlović, *On some formulas involving !n and the verification of the !n-hypotesis by use of computers,* Publ. Inst. Math. 47 (1990), 24–32.

[5] D. Kurepa, *On the left factorial function,* Mathematica Balkanica 1 (1971), 147–153.

[6] M. J. Quinn, *Designing efficient algorithms for parallel computers,* McGraw–Hill, 1987.

[7] H. Riesel, *Prime numbers and computer methods for factorization,* Birkhauser, 1985.

Matematički institut
Knez Mihailova 35, P.P. 367
11001 Beograd

RAČUNARSTVO
Sveska 1, Broj 2, 1992

UDK 519.682

SVEREL—PROLOŠKI PROGRAM
(objedinjavanje svih relacija)[1]

SLAVIŠA PREŠIĆ I PAVLE BLAGOJEVIĆ

U ovom članku se izlaže jedan proloski program, i pritom u pisanju pojedinih delanaka se podrazumevaju liste. Takvu LISP-ovsku sintaksu koristi, na primer, MICRO-Prolog. Program je pisan upravo u toj verziji prologa.

Osnovna zamisao tog programa je sledeća:

- „Ulaz" je izvestan zadani proloski program P (to je ili tekući, radni program ili program u nekoj datoteci).
- „Izlaz", odnosno plod nakon primene SVEREL–programa je nov proloski program `sverel(P)`, koji ima tačno jednu relaciju imena „`sverel`". Ta relacija `sverel` u smislu koji objašnjavamo objedinjuje sve, svejedno kog broja, relacije polaznog programa P. Naime, u osnovi imamo ovakvu ekvivalenciju

```
(sverel rel a1 a2 ... an) <--> (rel a1 a2 ... an)
```

tj. Objekti a1, a2, ... ,an su u relaciji `rel` ako i samo ako Objekti `rel`, a1, a2, ... ,an su u relaciji `sverel`.

Primera radi, uočimo sledeći program P:

```
((a 1 2)) ((a 3 4)) ((a 5 6)) ((a 2 1))
((a 3 7)) ((a 6 5)) ((a 7 7)) ((b 9 99)) ((b 1 2))
((b _x _y) (a _x _y) (a _y _x))
```

Tada njegovim „sverelovanjem" nastaje ovaj program `sverel(P)`:

```
((sverel a 1 2)) ((sverel a 3 4)) ((sverel a 5 6))
((sverel a 2 1)) ((sverel a 3 7)) ((sverel a 6 5))
```

[1]Rad finansiran od Fonda za nauku Republike Srbije preko Matematičkog instituta, projekat 0401A.

```
((sverel a 7 7)) ((sverel b 9 99)) ((sverel b 1 2))
((sverel b _x _y) (sverel a _x _y) (sverel a _y _x))
```

U vezi sa programom P može se, na primer, postaviti pitanje

```
?((a _x _y) (PP _x _y))
```

i za _x, _y će se dobiti 1,2 tj. prva rešenja. Međutim, nemoguće je na prolog direktno prevesti ovakvo pitanje:

Naći relaciju `rel` *programa* P, *znajući da 1,2 jesu u toj relaciji.*

Naime, pokušaj oblika:

```
?((_rel 1 2) (PP _rel))
```

neće uspeti jer prolog uopšte zahteva da tokom razvoja prološkog algoritma svaka usput pojavljena relacija bude poznata.

Međutim sverelovanjem programa P to pitanje se, uz pomoć `sverel`–relacije može ovako postaviti:

```
?((sverel _rel 1 2) (PP _rel))
```

i kao odgovor će se dobiti a. Da smo, recimo pitali

```
?((sverel _rel 1 2) (PP _rel)FAIL)
```

u odgovoru bismo dobili a i b.

Kao što se već iz tog malog primera vidi, i ako se u prologu u načelu ne mogu postavljati pitanja sa nepoznatim relacijama, upotrebom transformacije sverelovanje pitanja se preobraćaju na nov oblik u kome se mogu uspešno postaviti.

Uočimo još jedan primer—primer faktorijela:

```
((fakt 0 1))
((fakt _n _m) (LESS 0 _n)
              (SUM 1 _nn _n)
              (fakt _nn _mm)
              (TIMES _n _mm _m))
```

gde su `LESS`, `SUM`, `TIMES` tri osnovne (kaže se i sistematske) relacije prologa:

```
(LESS a b) <-> a<b;     (SUM a b c) <-> c=a+b;     (TIMES a b c) <-> c=a·b
```

Pomenimo da relacije `SUM` i `TIMES` „su sposobne“ da uz poznavanje dva argumenta izračunaju treći. Recimo, ako se usput pojavi pojavi formula `(SUM 1 _nn`

6) onda po definiciji relacije `SUM` _nn je izračunljiv i _nn=5. Sverelovanjem tog programa nastaje sledeći program:

```
((sverel fakt 0 1))
((sverel fakt _n _m) (LESS 0 _n)
                     (SUM 1 _nn _n)
                     (sverel fakt _nn _mm)
                     (TIMES _n _mm _m))
```

Primetite, što je bitno, sverelovanje ne „hvata“ nijednu od sistemskih relacija.

Primera radi, u vezi sa prethodnim programom smemo postaviti i ovakvo pitanje:

```
?((sverel _rel 6 720) (PP _rel))
```

i kao odgovor ćemo dobiti: `fakt`.

U nastavku sledi `sverel`–program. Nekoliko reči o načinu korišćenja. Osnovni način je da se napravi datoteka imena `SVEREL.LOG`, koja onda po potrebi i želji služi kao „korisnički“ (*utility*) program. Shodno tome kad želimo u okviru prologa učitamo tu datoteku sa:

```
LOAD SVEREL
```

tada ako hoćemo da sverelujemo tekući program, tj. onaj sa kojim već neposredno radimo, postavimo pitanje

```
?((Sver Tekuci))
```

Međutim, ako hoćemo da sverelujemo neki drugi program u fajl imena `IME` onda postavimo ovakvo pitanje

```
?((Sver Ime))
```

Jedan određen primer: `?((Sver ''BETWEEN.LOG''))`.

Napomena:

Priloženi SVEREL–program kao svoje „službene“ reči koristi:

```
Sver, citaj, sve, sverel, ne_sis
```

što znači da tekući program, na koji hoćemo da upotrebimo taj `SVEREL` program ne sme među imenima svojih relacija da sadrži neko od tih. Dalje, ime `PRIV777.LOG` je korišćeno kao ime pomoćne datoteke.

LITERATURA

[1] K. L. Clark i F. G. McCabe *Micro-PROLOG: Programming in Logic*, Prentice-Hall, Englewood Cliffs, New Jersey, 1984.

```
(/* SVEREL PROGRAM )

((Sver Tekuci)
    (/* Ako hocemo tekuci program da sverelujemo)
    (KILL (Sver sve svefor dobro citaj))
    (SAVE "PRIV777.LOG")
    (LOAD "SVEREL.LOG")
    (Sver "PRIV777.LOG"))

(Sver _Ime) (/* Ako hocemo da sverelujemo program u datoteci Ime)
    (OPEN _Ime) (citaj _Ime) (CLOSE _Ime)
    (PP Sverelovanje zavrseno, postavite pitanje))

((citaj _X) (READ _X _Y) (sve _Y _Z) (ADDCL _Z) (citaj _X))
((citaj _X))

    ((svefor (NOT|_A) (NOT |_A1)) (svefor _A _A1))
    ((svefor (!|_A) (!|_A1)) (svefor _A _A1))
    ((svefor (_A _B) (_A _B1))
             (CON _A) (ON _A (ADDCL DELCL CL ?)) (sve _B _B1))
    ((svefor (OR _A _B) (OR _A1 _B1)) (sve _A _A1) (sve _B _B1))
    ((svefor (IF _A _B _C) (IF _A1 _B1 _C1))
             (svefor _A _A1) (sve _B _B1) (sve _C _C1))
    ((svefor (FORALL _A _B) (FORALL _A1 _B1))
             (sve _A _A1) (sve _B _B1))
    ((svefor (ISALL _A _B|_C) (ISALL _A _B|_C1)) (sve _C _C1))
    ((svefor (_A|_B) (sverel|(_A|_B)) (dobro _A))
    ((svefor _A _A))

    (dobro _A)
             (NOT ON _A (Sver citaj sve svefor dobro)) (CON _A) (NOT SYS _A))

    ((sve ((_A _B)) ((_A1 _B1)))
             (CON _A) (ON _A (ADDCL DELCL CL ?)) (sve _B _B1))
    ((sve (_X|_Y) (_P|_Q)) (svefor _X _P) (sve _Y _Q))
    (sve () ()))
```


RAČUNARSTVO
Sveska 1, Broj 2, 1992

UDK 519.683

PARALELIZACIJA RADA GENERATORA PSEUDO-SLUČAJNIH NIZOVA BITOVA (GPSN) I UOPŠTENOG GENERATORA PSEUDO-SLUČAJNIH BROJEVA (CFSR)

TANJA PULEVIĆ

SAŽETAK. U ovom članku se razmatra paralelizacija rada nekih generatora slučajnih brojeva. Tehnički, akcenat je stavljen na simulaciju dijeljenja memorijskog prostora kod transpjuterske ploče sa procesorima T800.

Prije izlaganja algoritama, nekih teorijskih osnova i razloga zbog kojih su generatori pseudo-slučajnih brojeva čiji rad paralelizujemo interesantni, upoznajmo se sa paketom potprograma Paralelnog Fortrana koji omogućava simulaciju dijeljenja memorijskog prostora.

1 O nitima u paralelnom Fortranu

Transpjuterska ploča sa procesorima T800 na kojoj su implementirani algoritmi iz ovog rada, nema realnu mogućnost dijeljenja memorijskog prostora među procesima. Paralelni Fortran koristi prednosti arhitekture transpjutera i dozvoljava kreiranje niti koje se konkurentno izvršavaju unutar jednog zadatka. Niti jednog zadatka izvršavaju se na jednom procesoru i mogu dijeliti memorijski prostor. Osim preko zajedničke memorije, niti mogu komunicirati jedna sa drugom i preko kanala. Potprogrami paketa `THREAD` (nit) omogućavaju programu napisanom na Paralelnom Fortranu da kreira niti unutar jednog zadatka. Svaki potprogram koji koristi paket `THREAD` treba da uvede datoteku paketa, koja se zove `THREAD.INC`, pomoću rečenice

```
INCLUDE 'THREAD.INC'
```

Opišimo neke potprograme i funkcije paketa `THREAD`, tj. ono što nam je potrebno za elementarne manipulacije sa nitima.

Kreiranje niti

Nit se može kreirati pomoću opšteg potprograma F77_THREAD_START ili pomoću funkcije F77_THREAD_CREATE.

Poziv

```
CALL F77_THREAD_START(SUB,WSARRAY,WSSIZE,FLAGS,NARGS,ARG1, ... ,ARGN)
```

znači da se kreira i počinje da se izvršava nova nit koja je bazirana na opštem potprogramu SUB. Ova nit koristi radni prostor koji počinje nizom WSARRAY. WSARRAY je tipa INTEGER. Veličina radnog prostora u bajtima je opisana sa WSSIZE. Argument FLAGS odgovara skupu atributa niti koja se kreira. Za sada, na raspolaganju je samo jedan atribut, proritet niti. Prioritet može biti "urgentan" i "nije urgentan". U datoteci THREAD.INC date su konstante koje opisuju atribut prioriteta. Te konstante su: F77_THREAD_URGENT i F77_THREAD_NOTURG. Ako nova nit treba da ima isti prioritet kao tekuća (ona u čijem sklopu se kreira nova nit) za argument FLAGS se uzima vrijednost F77_THREAD_PRIORITY() koja je tipa INTEGER. NARGS odgovara broju argumenata koje predajemo potprogramu SUB. ARG1, ... ,ARGN su argumenti koji se predaju potprogramu. Argumenti mogu biti bilo kojeg tipa samo treba obratiti pažnju da se promjenljive tipa CHARACTER predaju potprogramu pomoću dva argumenta i o tome treba voditi racuna kada se specificira NARG.

Drugi način da se kreira nit je pomoću logičke funkcije F77_THREAD_CREATE. Poziv I=F77_THREAD_CREATE(SUB,WSSIZE,NARGS,ARG1, ... ,ARGN) kreira i startuje potprogram SUB kao novu nit koja se izvršava sa istim prioritetom kao nit koja je kreira i radni prostor joj je veličine WSSIZE. Ostali argumenti imaju isto značenje kao kod F77_THREAD_START. Ako je nit kreirana, ova funkcija vraća vrijednost .TRUE., inače vraća vrijednost .FALSE..

Zaustavljanje niti

Nit završava svoje izvršavanje kada se SUB kao potprogram završi ili pomoću poziva CALL F77_THREAD_STOP.

Korišćenje RTL

Poziv CALL F77_THREAD_USE_RTL rezervise RTL (Run time library) za tekuću nit. Poziv CALL F77_THREAD_FREE_RTL oslobađa RTL.

Ovo bi bili osnovni pozivi.

2 O generatorima pseudo-slučajnih nizova bitova

U kriptografiji se sa posebnim interesovanjem izučavaju pseudo-slučajni nizovi bitova, kao i specijalni hardver namijenjen za njihovo generisanje. Generatori pseudo-slučajnih nizova bitova (u oznaci GPSN) su bazirani na pomjeračkim registrima. Oni proizvode i-ti bit u nizu na osnovu prethodnih m bitova na sljedeći način.

$$b_i = F(b_{i-1}, \ldots, b_{i-m})$$

za neku funkciju $F\colon \{0,1\}^m \mapsto \{0,1\}$

Obično se uzima linearna rekurentna veza

$$b_i = (d_1 b_{i-1} + \cdots + d_k b_{i-m}) \mod 2 \tag{1}$$

gdje su $d_1, \ldots, d_k$ binarne konstante. Sabiranje po modulu 2 i EOR (ekskluzivno ili) imaju istu istinitosnu tablicu, pa (1) može da se zapise kao

$$b_i = b_{i-l_1} \text{ EOR } b_{i-l_2} \ldots b_{i-l_k}$$

gdje je $d_{l_1} = \cdots = d_{l_k} = 1$ i ostali $d_l = 0$. Uvedimo oznaku $\dot{+}$ umjesto EOR. Maksimalna perioda ovakvog generatora je $2^m - 1$. U principu, zanimljivi su oni generatori koji imaju maksimalnu periodu. Računanje periode kod generatora tipa (1) i razmatranje pitanja dostizanja maksimalne periode, obavlja se pomoću metoda faktorizacije polinoma nad konačnim poljima.

Rekurentnoj vezi (1) pridružuje se polinom

$$P(x) = x^m + d_1 x^{m-1} + \cdots + d_m$$

Uobičajeno je da se razmatraju trinomi

$$1 + x^r + x^s \qquad \text{gdje je} \qquad 1 \le r < s$$

Ovom trinomu odgovara rekurentna veza

$$b_i = b_{i-s} \dot{+} b_{i-(s-r)} \tag{2}$$

Obrtanjem niza dobija se trinom

$$1 + x^{s-r} + x^s$$

čija odgovarajuca rekurentna veza izgleda

$$b_i = b_{i-s} \dot{+} b_{i-r} \tag{3}$$

(2) i (3) imaju istu periodu.

Ovdje nećemo razmatrati pitanje biranja parametara r, s i početnih bitova, jer nas interesuje paralelizacija rada generatora onda kada su svi parametri izabrani.

Navedimo neke parove (s, r) za koje GPSN daje maksimalnu periodu $2^s - 1$.

s	r
2	1
3	1, 2
⋮	⋮
17	3, 5, 6, 11, 12, 14
⋮	⋮
36	11, 25

Primijetimo da se od pseudo-slučajnih nizova bitova malim modifikacijama mogu dobiti pseudo slučajni brojevi iz uniformne raspodjele na intervalu (0,1). Postoji više načina za to. Jedan od najjednostavnijih je

$$U_i = 0.b_{i_t} b_{i_{t+1}} \ldots b_{i_{t+n}}$$

gdje su n i t prirodni brojevi takvi da je $0 < n < t$.

Generisanje brojeva iz uniformne raspodjele je, između ostalog, interesantno jer se pomoću njih mogu modelirati druge raspodjele.

2.1 Paralelizacija rada GSPN

Cilj je, dakle, da se konstruiše paralelni algoritam koji proizvodi bitove pomoću rekurentne veze

$$b_i = b_{i-s} \dot{+} b_{i-r}$$

gdje su dati s, r i prvih s bitova. Izložicemo jednu ideju za $s = 9$, $r = 5$ i nproc $= 4$ (nproc je broj procesora). Ideja se uz male izmjene može primijeniti za neke druge s, r i nproc.

Paralelnu verziju rada GSPN

(4) $$b_i = b_{i-9} \dot{+} b_{i-5}$$

opišimo algoritmom SHIFT1. Fortranski kod za ovaj algoritam dat je u prilogu. Ideja za SHIFT1 se prirodno nameće rekurentnom vezom (4) i zahtijeva dijeljenje memorije među procesorima. To će se simulirati. Algoritam je jednostavan, njegovi zahtjevi za dijeljenje memorije se zadovoljavaju lako i zato je dobar kao edukativan primjer za početak rada sa paketom THREAD koji smo opisali u dijelu I.

```
!
!Konfiguraciona datoteka za zadatak shift1
!
!  shift1.cfg configuration file for task shift1
!
!  Hardware
!
processor host
processor t1
processor t2
processor t3
wire ?  host[0] t1[0]     !anonymous wire connecting PC to transputer
wire ?  t1[1] t2[0]
wire ?  t1[2] t3[0]
!
!  Task declarations indicating channel I/O ports
!       and memory requirements
!
task afserver ins=1 outs=1
task filter ins=2 outs=2 data=10K
task shift1 ins=5 outs=5 data=500k
!
!  Assign software tasks to physical processors
!
place afserver host
place shift1 t1
```



```
place filter t1
!
!  Set up the connections between the tasks.
!
connect ?  afserver[0] filter[0]
connect ?  filter[0] afserver[0]
connect ?  filter[1] shift1[1]
connect ?  shift1[1] filter[1]
```

```
      PROGRAM SHIFT1
************************************************************************
c     Vrsi se paralelizacija rada generatora slucajnih bitova.Simulira se
c     rad cetri procesa P1,P2,P3 i P4 koji dijele zajednicki prostor kroz
c     common zonu ZONA1.
c     Glavna nit P1 kreira niti P2,P3 i P4 koje imaju isti prioritet kao
c     ona.
c     niz bitova koji se generisu je niz DATA
************************************************************************
c     Uvodjenje datoteke paketa THREAD
      INCLUDE 'thread.inc'
c     Radni prostori niz2,niz3 i niz4 za niti P2,P3,P4
      INTEGER niz2(2500),niz3(2500),niz4(2500)
      INTEGER data
      EXTERNAL P2,P3,P4
c     Definisanje zajednicke zone
      COMMON/zona1/data(300),ip2,ip3,ip4
c        Prioritet tekuce niti koji kao argument treba predati ostalima
      iprio=f77_thread_priority()
      PRINT *,'duzina niza ?'
      READ *,iduz
c     postavljanje pocetnih bitova
      DO 130 i=1,9
130           data(i)=1

c     Kreiranje niti P2,P3 i P4
      CALL F77_THREAD_START(P2,niz2,2500*4,iprio,1,iduz)
      CALL F77_THREAD_START(P3,niz3,2500*4,iprio,1,iduz)
      CALL F77_THREAD_START(P4,niz4,2500*4,iprio,1,iduz)
      PRINT *,'kreirao sam'
c     Proces (nit) P1 radi svoj dio posla
      DO 122 i=10,iduz,4
      data(i)=mod(data(i-9)+data(i-5),2)
122   CONTINUE
c     Da li su ostali zavrsili posao?  ako jeste stampanje niza
```

```
900         IF((ip2+ip3+ip4).eq.3)THEN
                PRINT *,(data(i),i=1,iduz)
                STOP
            ELSE
                GO TO 900
            ENDIF
        END

c****************PROCES P2*****************************
        SUBROUTINE P2(KK)
        INTEGER data
        COMMON/zona1/data(300),ip2,ip3,ip4
        ip2=0
        DO 100 i=11,kk,4
            data(i)=mod(data(i-9)+data(i-5),2)
100     CONTINUE
        ip2=1
        END

c****************PROCES P3*****************************
        SUBROUTINE P3(kk)
        INTEGER data
        COMMON/zona1/data(300),ip2,ip3,ip4
        ip3=0
        DO 100 i=12,kk,4
            data(i)=mod(data(i-9)+data(i-5),2)
100     CONTINUE
        ip3=1
        END

c****************PROCES P4*****************************
        SUBROUTINE P4(kk)
        INTEGER data
        COMMON/zona1/data(300),ip2,ip3,ip4
        ip4=0
        DO 100 I=13,KK,4
            data(i)=mod(data(i-9)+data(i-5),2)
100     CONTINUE
        ip4=1
        END
```

Ideja:

Počevši sa 9 datih bitova $b_1, \ldots, b_9$ četiri procesa generišu po jedan bit i tako, u jednom taktu, nastavljaju niz za četiri bita. Npr. u prvom taktu proces $P1$ generiše b_{10}, $P2$ generiše b_{11}, $P3$ generiše b_{12}, $P4$ generiše b_{13}.

Vrijednosti koje su date za s, r i nproc su takve da je ovo izvodljivo, jer u jednom taktu svaki proces ima različito polje rada.

Primijetimo da poslije nekoliko taktova kada se "istroše" početne vrijednosti, za rad procesa $P1$ potrebni su samo bitovi koje generiše proces $P4$. Analogno, proces $P2$ zavisi od procesa $P1$, proces $P3$ zavisi od procesa $P2$ i proces $P4$ zavisi od procesa $P3$.

Pretpostavili smo idealnu situaciju, da svaki proces obavlja svaku operaciju u istom vremenskom intervalu. Bez ove pretpostavke algoritam SHIFT1 treba da se dopuni uvođenjem "čekanja" među procesima. Procesi bi trebalo da se čekaju po principu zavisnosti koju smo naveli, tj. proces $P1$ bi čekao proces $P4$, proces $P2$ bi čekao proces $P1$ itd ... Čekanje se može realizovati semaforskom tehnikom, indikatorima u zajedničkoj memoriji i slično.

Iz izloženog se nameće da je ocjena efikasnosti algoritma optimalna tj. jednaka $O(\text{nproc})$.

3 O generalizovanom generatoru pseudo-slučajnih brojeva baziranom na pomjeračkom registru (oznaka GFSR)

Konstruktori GFSRA-a, Lewis i Payne, ostaju u svijetu bitova i princip rada ovog generatora zasnivaju na pravljenju n-bitnih prirodnih brojeva. Preciznije iz pseudo-slučajnog niza bitova uzima se n nesusjednih bitova i dobija se n-bitni prirodni broj

$$X_i = b_i b_{i-j_2} \ldots b_{i-j_n} \tag{5}$$

Svaki bit od X_i zadovoljava zakon (2) pa se može formirati rekurentna veza $X_i = X_{i-s} + X_{i-(s-r)}$ Prvih s početnih vrijednosti ne moraju da zadovoljavaju vezu (5). Perioda niza niza (X_i) zavisi od početnih vrijednosti.

Trivijalan način da se od ovakvog niza dobije niz pseudo-slučajnih brojeva iz uniformne raspodlele na intervalu (0,1) jeste

$$U_i = 2^{-n} X_i$$

Primjer za GFSR $s = 7$, $r = 1$, $n = 3$ i početna matrica (seed) je:

$$\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{matrix}$$

Ovaj generator daje niz cija je perioda jednaka 127:

0 1 2 3 4 5 6 1 3 1 7 1 3 7 2 2 6 6 2 4 ...

Algoritam koji je paralelna verzija rada GFSR-a i čiji fortranski kod je dat u prilogu, naziva se SHIFT2. Algoritam je ilustrovan na gornjem primjeru.

```
! konfiguraciona datoteka za shift2
! shft2.cfg configuration file for task shift2
!
! Hardware
!
processor host
processor t1
processor t2
processor t3
wire ?  host[0] t1[0]     !anonymous wire connecting PC to transputer
wire ?  t1[1] t2[0]
wire ?  t1[2] t3[0]
!
! Task declarations indicating channel I/O ports
!      and memory requirements
!
task afserver ins=1 outs=1
task filter ins=2 outs=2 data=10K
task shi21 ins=5 outs=5 data=500k
!
! Assign software tasks to physical processors
!
place afserver host
place shi21 t1
place filter t1
!
! Set up connections between the tasks.
!
connect ?  afserver[0] filter[0]
connect ?  filter[0] afserver[0]
connect ?  filter[1] shi21[1]
connect ?  shi21[1] filter[1]
```

```
      PROGRAM SHIFT2
************************************************************************
C     Vrsi paralelizaciju rada uopstenog generatora slucajnih brojeva, pomocu
c     3 - bitnih prirodnih brojeva.  Simulira se rad tri procesa P1, P2 i P3.
c     Proces (nit) P1 kreira P2 i P3 koji su istog prioriteta kao i ona.
c     Matrica mat sadrzi 3-bitne prirodne brojeve
c     P1 formira prirodne brojeve od vrsta matrice mat koja je u djeljivoj
```



```
c     memoriji koju smo simulirali preko common zone zona1
c     Napomena :  pomjeranja se mogu parametrizovati.  Dakle, vrijednosti
c     za s i r se mogu ucitavati i prenijeti nitima kao ulazni argumenti.
c     Ovdje to nije uradjeno nego program radi za s=7 i r=1

**************PROCES P1 ***************************************************
c     Proces P1 kreira ostale, radi na prvoj koloni, formira prirodne brojeve
c     i stampa.
C     Uvodjenje datoteke paketa THREAD
      INCLUDE 'thread.inc'
c     Radni prostori za P2 i P3
      INTEGER niz2(2500),niz3(2500)
      EXTERNAL P2,P3
      DIMENSION niiz(3000)
c     Definisanje djeljive memorije
          COMMON/zona1/mat(300,3),ipp2,ipp3,ipp4
c     Pocetne vrste matrice
      mat(1,1)=0
      mat(1,2)=0
      mat(1,3)=0
      mat(2,1)=0
      mat(2,2)=0
      mat(2,3)=1
      mat(3,1)=0
      mat(3,2)=1
      mat(3,3)=0
      mat(4,1)=0
      mat(4,2)=1
      mat(4,3)=1
      mat(5,1)=1
      mat(5,2)=0
      mat(5,3)=0
      mat(6,1)=1
      mat(6,2)=0
      mat(6,3)=1
      mat(7,1)=1
      mat(7,2)=1
      mat(7,3)=0

      PRINT *,'koliko vrsta ?'
      READ *,ivrs
c     Definisanje prioriteta tekuce niti
      IPRIO=F77_THREAD_PRIORITY()
      PRINT *,((mat(I,J),J=1,3),I=1,7)
c     Kreiranje niti P2 i P3
```

```
      CALL F77_THREAD_START(P2,NIZ2,2500*4,IPRIO,1,IVRS)
      CALL F77_THREAD_START(P3,NIZ3,2500*4,IPRIO,1,IVRS)

      PRINT *,'Kreirao sam'
c     P1 obavlja svoj dio posla, radi na prvoj koloni
      DO 111 i=8,ivrs
111       mat(i,1)=mod(mat(i-7,1)+mat(i-6,1),2)
c     Da li su P2 i P3 zavrsili posao?  Ako jesu onda formiram prirodne brojeve
c     i stampam ih (zbog provjere)
9000      IF(ipp2+ipp3.eq.2)THEN
                DO 112 i=1,ivrs
                   niiz(i)=mat(i,1)*4+mat(i,2)*2+mat(i,3)
                   PRINT *,niiz(i)
112             CONTINUE
              ELSE
                GO TO 9000
          ENDIF
      END

**********PROCES P2 RADI DRUGU KOLONU**********

      SUBROUTINE P2(KK)
      COMMON/zona1/mat(300,3),ipp2,ipp3
      ipp2=0
      DO 22 i=8,kk
         mat(i,2)=mod(mat(i-7,2)+mat(i-6,2),2)
22          CONTINUE
      ipp2=1
      END

**********PROCES P3 RADI TRECU KOLONU**********

      SUBROUTINE P3(KK)
      COMMON/zona1/mat(300,3),ipp2,ipp3
      ipp3=0
      DO 22 i=8,kk
         mat(i,3)=mod(mat(i-7,3)+mat(i-6,3),2)
22          CONTINUE
      ipp3=1
      END
```

3.1 Paralelizacija rada generatora GFSR

Dakle, treba paralelizovati generisanje niza n-bitnih prirodnih brojeva po vezi $X_i = X_{i-s} + X_{i-(s-r)}$. Algoritam SHIFT2 podrazumijeva da je nproc $= n$.

Ideja:

Izračunavanja po modulu 2 se vrše po koordinatama. Prvi procesor sabira bitove prve kolone po modulu 2, ..., n-ti procesor sabira bitove n-te kolone po modulu 2. Prilikom generisanja bitova procesi nemaju potrebe da se čekaju jer jedan proces radi stalno na jednoj koloni, kojoj ne pristupaju ostali procesi. Iz izloženog proizilazi ocjena efikasnosti rada algoritma: optimalna je i jednaka O(nproc).

Pretpostavka o n procesa može se zamijeniti sa nproc $= k$ gdje je $k < n$. Tada bi svaki proces radio na jednom bloku kolona. Blokovi su veličine $[n/k]$, eventualno neki blok bi bio veći. Ideja ostaje ista.

Ovaj rad je nastao zahvaljujući uslovima koji su mi obezbijedili MATEMATIČKI INSTITUT—BEOGRAD i MATEMATIČKI FAKULTET—BEOGRAD.

LITERATURA

[1] Brian D. Ripley, *Stochastic Simulation*, J Wiley & Sons, Canada, 1987.

Prirodno-matematički fakultet
Cetinjski put b. b.
81000 Titograd

RAČUNARSTVO
Sveska 1, Broj 2, 1992

UDK 519.688

PROPOV MODEL I GENERISANJE NARODNIH PRIPOVEDAKA POMOĆU RAČUNARA

VLADIMIR MILIČIĆ

SAŽETAK. U radu je prikazan pokušaj postavljanja na računar Propovog modela strukture i procesa ruskih bajki posredstvom semiografije i verovatnosne mreže.

1 Uvod

Vladimir Prop, jedan od mnogih briljantnih ruskih folklorista svoga vremena, objavio je **Morfologiju bajki** 1928. godine [1]. Ta knjiga predstavlja vrhunsko dostignuće ruske folklorističke škole, koju je predvodio u devetnaestom veku drugi veliki folklorista Veselovski, a koja je inače bogata vrlo značajnim istraživačkim rezultatima. Propova **Morfologija** je rezultat kombinacija uticaja, pre svega, ruske folkloristike i ruskog formalizma i strukturalizma, finskih i nemačkih ideja iz te oblasti istraživanja, kao i Propovog genija za sistematizaciju i apstrakciju folklornog narativnog materijala.

Morfologija je uspešan napor apstrahovanja trideset i jedne osnovne narativne funkcije i preko sedam bazičnih dejstvujućih karaktera (prema Propu, *dramatis personae*) ruskih bajki. Tu knjigu danas moramo uvrstiti među druga polazna (?) intelektualna dostignuća koja reprezentuju rusku naklonost i darovitost za teoretska istraživanja, apstraktnu delatnost i generalizaciju, kao što su, na primer, da spomenem samo dva dela, sistem elemenata Mendeljejeva i sistem glume Stanislavskog.

2 Šta predstavlja Propov model?

Ako se zapitamo šta zapravo predstavlja Propova **Morfologija**, mogli bismo da odgovorimo da **ona ocrtava, na formalan i sistematski način, deo uglavnom nesvesnog znanja tipičnog ruskog pripovedača ruskih bajki**. Drugim rečima, ona utelovljuje deo narativne kompetencije ruskog folklornog pripovedača na sličan način na koji neke od modernih gramatika predstavljaju deo jezičke kompetencije govornika maternjeg jezika. Propova knjiga u stvari daje sistem znanja kojim može da se rukuje i kojim svako može da se koristi ne samo sa namerom da razume deo unutrašnjeg mehanizma ruskih bajki već takođe i da koristeći taj sistem znanja stvori narodne priče ne samo ruskog tipa.

Model koji je Prop predstavio u svojoj knjizi podstakao je pokušaje pisanja sličnih knjiga a takođe i vrlo intenzivna istraživanja strukture priča uopšte kao i ispitivanja u vezi sa održavanjem pamćenja pripovedaka i njihovih delova i funkcija kod slušalaca tokom raznih vremenskih perioda. Objavljena je bogata literatura povodom ove Propove knjige, naročito po njenom prevođenju na engleski i francuski jezik pre tridesetak godina.

3 Rad na postavljanju Propovog modela na računar

Jedan od vodećih kanadskih etnologa-folklorista, Pjer Maranda—često sa svojom suprugom Eli koja je istog poziva—već duže vremena proučava probleme naratologije, naročito se koncentrišući na ruske modele Propa i njegovog nastavljača Meletinskog. Među drugim značajnim doprinosima na polju folkloristike, Pjer Maranda je objavio i kratki članak "Semiografija i veštačka inteligencija" u **International Semiotic Spectrum**, šestomesečnoj publikaciji Torontskog semiotičkog kruga (broj 4, jun 1985, pp. 1–3). Da bi transformisao Propov model za potrebe računara, autor navodi da se koristio semiografijom—jednom od grana semiotike—teorijom okvira (engl. *frame theory*) iz oblasti proučavanja veštačke inteligencije. "**Semiografija je**", piše on, "tehnika opisa značenja i njegove konstrucije ... Kao tehnika "grafiranja", ona se oslanja na verovatnosne mreže (engl. *probabilistic networks*). Teorija okvira je oruđe za simulaciju "mentalnog stanja" (engl. *frame of mind*) u kome treba da bude računar da bi dešifrovao i generisao značenja svojstvena nacrtu" (p.1). Koristio sam Marandin grafički model (vidi sliku)—koji on naziva **Verovatnosna mreža reprezentacije Propovih dramatskih funkcija koje čine strukturu ruskih bajki**—kao osnovu za računarski program koji generiše narodne pripovetke. Maranda objašnjava:

> "Ova mreža je (...) semiografija Propove analize 100 ruskih folklornih pripovedaka. Kao "okvir", ona parametrizuje narativni prostor unutar folklora. Ona je "reprezentacija znanja"; znanja potrebnog i dovoljnog za prosečnog pripovedača da funkcioniše na zadovoljavajući način. Njeni čvorovi (engl. *nodes*) su "žlebovi" (engl. *slots*) koji obrazuju različita narativna stanja kojima pripovedač može da poveže—prema Propovoj terminologiji, različite narativne "funkcije". Dati lanac žlebova koje pripovedač

> poveznje skicira (engl. *map*) jednu od stotine pripovedaka zbirke. Primetite da svi ulančeni nizovi nisu iste verovatnoće. Oni sa visokom verovatnoćom strukturiraju stereotipe, ... (dok) oni sa niskom verovatnoćom podrazumevaju stvaralački i maštovit rad. (p. 1)"

(Slova grafikona na slici od **A** do **W**—sa indeksom ili bez njega—označavaju pojedine narativne funkcije u engleskom prevodu Propove knjige [2]. Brojevi upisani rukom dezigniraju odgovarajuće funkcije mog računarskog programa, tj. onog dela bez prvih sedam funkcija. Zaokruženi brojevi—procenti čvorova predstavljaju verovatnoće zadatih funkcija u sto ruskih bajki koje je Prop analizirao, dok brojevi—procenti na lukovima između dva čvora obeležavaju verovatnoće ostvarenja veza među čvorovima.)

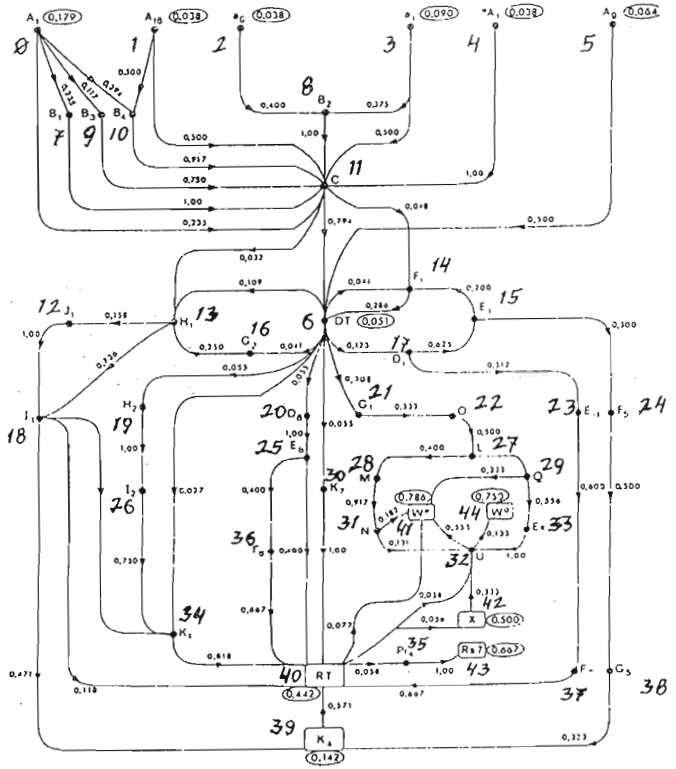
4 Istraživanje i rezultati

Već krajem šezdesetih godina želeo sam da napravim računarski program koji generiše ruske bajke ali sve što sam tada uradio bilo je vrlo primitivno u poređenju sa mogućnostima koje pruža Marandina verovatnosna mreža. Ona modelira sve Propove bazične funkcije (nazovimo ih tipovima) izuzev **uvodne situacije**, zatim prvih sedam funkcija i **XXIX**-te funkcije, transfiguracije. Mreža, međutim, uključuje—prema Propovom obeležavanju u engleskom prevodu—negativne funkcije **E i F** (junakovu reakciju odnosno dobijanje magičnog instrumenta odn. moći), i funkciju **X**, koja—prema Propu—predstavlja nejasne elemente i forme ili, prema Marandinim rečima:

> "šum" (kao, na primer, pripovedačev trenutni zaborav (engl. *memory blank*), mutnu logiku (engl. *fuzzy logic*), ili druge nedostatke kodiranja); priča može da se završi tu (verovatnoća je p= 0.5 da **N** bude "ponor" (engl. *sink*) ili da bude prepona koja sasvim preuzima pažnju) ako je pripovedač nekompetentan, ako se razboleo, itd. ili pripovedanje može da se nastavi (p= 0.333) ako je publika tolerantna." (p.3)

Pored već spomenute dvadeset i tri pojavnica-funkcije (engl. *type*) i tri različnica-funcije (engl. *token*) ($\mathbf{E}_{neg}$, $\mathbf{F}_{neg}$ i X), mreža takođe uključuje i osamnaest varijanata—različnica, što sve u svemu čini četrdeset i četiri funkcije.

Za potrebe svog nastavnog predmeta **Struktura Pripovetke** analizirao sam dosta priča raznih folklornih tradicija (ruske, srpske, makedonske, irske, engleske, iranske) kao i seriju "umetničkih" pripovedaka i zaključio da Propov model—sa izvesnim manjim izuzecima i promenama—može da objasni njihovu morfologiju. Uzevši sve navedeno u obzir, rešio sam da napravim računarski program za generisanje raznih verzija jedne iste priče. Prop-Marandin model sadrži osam glavnih puteva (engl. *paths*) od kojih svaki ima nekoliko dodatnih pod-puteva. Pošto Maranda nije koristio za svoju verovatnosnu mrežu prvih sedam Propovih funkcija, dodao sam ih kao zaseban niz povezan verovatnoćama (koje sam dodelio na osnovu svoje



Probabilistic Network Representation of Propp's Dramatic Functions Constituting the Dynamic Structure of Russian Folktales

intuicije) sa ostalim funkcijama. Na taj način je moj računarski model odn. program sadržavao pedeset i jednu funkciju. (Propovih varijantnih funkcija, različnice, ima preko šezdeset i pet.) Primera radi, evo nekih od najdužih primera svake od osam glavnih staza izraženih u brojevima. Prvih sedam funkcija (I, II, III, IV, V, VI, VII) nisu uključene u priloženi graf a ni u ove primere (premda se, u najpotpunijoj

formi pojavljuje svih sedam):

1.	0	10	11	6	16	13	12	18	34	40	32	44;	
2.	2	8	11	6	19	26	34	40	35	43;			
3.	4	11	6	17	15	14	6	34	40	42	32	41;	
4.	5	6	20	25	36	40	32	41;					
5.	3	8	11	6	30	40	35	43;					
6.	1	10	11	6	21	22	27	28	31	32	44;		
7.	0	9	11	6	17	23	37	40	32	41;			
8.	0	7	11	14	6	17	15	24	38	39	40	35	43.

Ovaj program sam već koristio u radu sa studentima i tokom raznih eksperimenata sam utvrdio da program ima ogromne generativne mogućnosti, a njegove teoretske posledice su takođe vrlo značajne. Prvo sam dao grupi od sedam svojih studenata po nekoliko funkcija sa njihovim definicijama i tražio da napišu odgovarajući tekst za svaku od njih. Prethodno smo se dogovorili oko imena karaktera kao i o opštoj ideji sadržaja buduće priče. Kada je tekst bio napisan i unesen u računar, program je primenjen na njega. Rezultati su bili mnogobrojne varijante pripovetke—koje je program nasumice (engl. *randomly*) generisao prema ugrađenim verovatnoćama modela—duž osam glavnih puteva koje sadrži mreža. Pošto smo zajedno pročitali i diskutovali neke od varijanata, uočili smo da bi među piscima sadržaja pojedinih funkcija moralo postojati više saradnje ili, pak, da bi sadržaj svake funkcije trebalo da piše ista osoba kako bi varijante priče bile tečnije, a prelazak sa funkcije na funkciju logičniji. Pošto smo to uradili, kontinuitet pričanja je suštinski popravljen. Međutim, u mnogim varijantama bilo je potrebno dodati nešto ili, pak, rukom ispraviti da bi bi se tok priče prirodnije razvijao. U nameri da rešim neke od datih problema—tako da bi se pisac što je moguće manje mešao u računarske rezultate—treći cilkus pisanja sadržaja funkcija bio je usmeren na funkcije svakog od osam glavnih puteva pojedinačno i nekih od njihovih varijanata. Takav prilaz se pokazao najuspešnijim s tačke gledišta kvaliteta generisanih pripovedaka. Glavni problem koji je preostao bio je neposredni rezultat nasumičnog generisanja varijanata priče sa različitim brojem funkcija uključenih u pojedinačne puteve: nasumična redukcija nekih funkcija u mnogim slučajevima je imala negativne posledice za logiku i kontinuitet razvoja radnje date priče-varijante. Kao posledica, one varijante pripovedaka na putevima sa većim ili maksimalnim brojem funkcija mogle su da stoje same za sebe bez većih izmena nasuprot onim pripovetkama sa manjim projem puteva nad kojima je bilo potrebno izvršiti veće ručne intervencije.

5 Zaključak

Nekoliko interesantnih nalaza je proisteklo iz eksperimentisanja sa ovim programom. Pre svega, bilo je zanimljivo shvatiti da je sa neznatnim promenama sadržaja funkcija i promenom imena dejstvujućih karaktera, moguće generisati osam različitih priča sa masom njihovih varijanata. Ali najvažniji i sasvim neočekivani rezultat korišćenja ovog programa je činjenica da najkraća varijanta bilo koje generisane priče sadrži tri funkcije, te tako **teorijski i praktično definiše strukturu i sadržaj najkraće pripovetke zahvaljujući nasumičnim verovatnoćama ugrađenim u Prop-Marandin model**. Takva, "najkraća", priča u potpunoj je saglasnosti sa teoretskim tvrdnjama i definicijama koje su predložili teoretičari književnosti.

Zahvalnost

Zahvaljujem se profesoru Saimu Uralu sa Odseka za računarstvo našeg univerizteta na realizaciji programa u jeziku **Modula 2**.

LITERATURA

[1] Владимир Пропп: *Морфология сказки*, 2. издание, Москва, 1969.
[2] Vladimir Propp: *Morphology of the Folktale*, Second Edition Revised, University of Texas Press, 1968.

Ključne reči: Ruska bajka, narativna funkcija, generisanje, semiografija, teorija okvira, verovatnosne mreže.

Western Washington University
Bellingham, Washington, 98225
USA

RAČUNARSTVO
Sveska 1, Broj 2, 1992

UDK 519.688

JEDAN ALGORITAM ZA ODREĐIVANJE NAJVEĆEG ZAJEDNIČKOG DELIOCA POLINOMA[1]

BOŠKO DAMJANOVIĆ

SAŽETAK. U članku je izložen i na programskom jeziku BASIC realizovan algoritam za određivanje najvećeg zajedničkog delioca od n polinoma sa celobrojnim koeficijentima, koji je zasnovan ne na definiciji najvećeg zajedničkog delioca već na njegovim svojstvima. Navedenim programom NZD od n polinoma određuje se direktno bez primene uobičajenih iteracija i deljenja polinoma sa polinomom. Navodi se i primer koji ilustruje algoritam.

1 Uvod

Euklidov algoritam za određivanje najvećeg zajedničkog delioca (NZD) dva polinoma $f(x)$ i $g(x)$, gde stepen polinoma $f(x)$ nije manji od stepena polinoma $g(x)$, sastoji sastoji u određivanju sledećeg niza polinoma:

$$\begin{aligned} f &= q \cdot q_1 + r_1, \\ g &= r_1 \cdot q_2 + r_2, \\ r_1 &= r_2 \cdot q_3 + r_3, \\ &\cdots \\ r_{m-1} &= r_m \cdot q_{m+1}, \end{aligned}$$

gde je $\operatorname{step} g > \operatorname{step} r_1 > \cdots > \operatorname{step} r_m$. Lako je ustanoviti da je sa polinomom r_m deljiv svaki polinom navedenog niza polinoma f, g, $r_1, \ldots, r_{m-1}$, pa to specijalno važi i za polinome f i g. Na taj način polinom r_m je delilac polinoma f i g. Osim toga i bilo koji drugi polinom koji se od polinoma r_m razlikuje do na konstantni činilac takođe je delilac polinoma f i g. Isto to važi i za polinome sa kojim je r_m deljiv. Na taj način, NZD dva polinoma je onaj delilac ta dva polinoma koji

[1] Rad finansiran od Fonda za nauku Republike Srbije preko Matematičkog instituta, projekat 0401A.

je deljiv svim drugim njihovim zajedničkim deliocima i određen je sa tačnošću do konstantnog činioca.

Budući da je $\mathrm{NZD}(f_1, f_2, f_3) = \mathrm{NZD}(\mathrm{NZD}(f_1, f_2), f_3)$ to navedena procedura može biti proširena na nalaženje NZD od više polinoma. Međutim, upotreba Euklidovog algoritma na upravo prikazan način u slučaju velikog broja polinoma velikog stepena može biti prilično rogobatan posao. Zato ćemo navesti jedan drugi algoritam zasnovan ne na definiciji NZD polinoma već na njegovim svojstvima.

2 Svojstva NZD polinoma

Neka su data dva polinoma:

$$\begin{aligned} f(x) &= a_1 + a_2x + a_3x^2 + \ldots + a_{n+1}x^n \quad \text{i} \\ g(x) &= b_1 + b_2x + b_3x^2 + \ldots + b_{m+1}x^m \end{aligned}$$

sa celobrojnim koeficijentima i sa $a_{n+1} \neq 0$, $b_{m+1} \neq 0$ i $m \leq n$.

Ako je $d(x)$ jedan od NZD polinoma $f(x)$ i $g(x)$ tada je:

a) $d(x) = \mathrm{NZD}(k_1 f(x), k_2 g(x))$, $\quad k_1, k_2 \in \mathbf{Z} \setminus \{0\}$,

b) $d(x) = \mathrm{NZD}(f(x), g(x) + kf(x))$, $\quad k \in \mathbf{Z}$,

c) Ako je $a_1 = 0$ i $b_1 \neq 0$ tada je $d(x) = \mathrm{NZD}(g(x), f(x)/x)$,

d) Ako je $a_1 = b_1 = 0$ tada $d(x)$ sadrži faktor x, tj. $d(x) = x \cdot \mathrm{NZD}(f(x)/x, g(x)/x)$, gde je sa $\mathbf{Z}$ označen skup svih celih brojeva.

3 Opis metode

Algoritam nalaženja NZD polinoma $f(x)$ i $g(x)$ biće zasnovan samo na tim svojstvima. Na dati par polinoma $f(x)$ i $g(x)$ primenjivaće se navedena svojstva od NZD dva polinoma da bi se dobili drugi parovi polinoma koji imaju isti NZD kao i početni par. Sistematski primenjujući ta svojstva i birajući na odgovarajući način faktore k, k_1 i k_2 dobićemo niz parova polinoma sa nerastućim zbirom njihovih stepena (zbog svojstava *c*) i *d*)) koji će završiti parom polinoma $(d(x), 0)$ među kojima je drugi identički jednak nuli. Dobijeni, ne identički jednak nuli polinom $d(x)$ je traženi NZD tih polinoma.

Radi lakšeg izračunavanja NZD od data dva polinoma $f(x)$ i $g(x)$ predstavljaćemo ih u obliku tabele

$$T = \begin{vmatrix} b_1 & b_2 & \ldots & b_{m+1} & 0 & \ldots & 0 \\ a_1 & a_2 & \ldots & a_{m+1} & a_{m+2} & \ldots & a_{n+1} \end{vmatrix}$$

dimenzije $2 \times (n+1)$ u kojoj prvoj vrsti odgovara polinom nižeg stepena.

Algoritam određivanja NZD polinoma f i g započinjemo uklanjanjem svih vodećih nul kolona iz tabele T sve dok se ne dobije tabela T_1 u čijoj prvoj koloni je bar jedan element različit od nule. Ako su prvih k kolona tabele T, nul-kolone a

$(k+1)$-va kolona to nije, onda je stepen x^k faktor od $\mathrm{NZD}(f(x), g(x))$ i ostaje da se, prema svojstvu *d*), odredi NZD od polinoma koji odgovaraju tabeli T_1.

Prema svojstvu *c*) mogu se ukloniti sve vodeće nule u prvoj ili drugoj vrsti posmatrane tablele T_1, ako one tamo postoje, i postići, prema svojstvu *a*), da prvi koeficijenti u njima budu međusobno jednaki. Na primer, tabela

$$\begin{vmatrix} 3 & 3 & 2 & 1 & 0 & 1 & 5 & 6 \\ 0 & 0 & 4 & 6 & 0 & 4 & 18 & 0 \end{vmatrix}$$

se može, posle uklanjanja vodećih nula iz druge vrste, skraćivanja njenih elemenata sa njihovim NZD i množenjem elemenata prve vrste sa 2 a druge sa 3, zameniti sa tabelom

$$\begin{vmatrix} 6 & 6 & 4 & 2 & 0 & 2 & 10 & 12 \\ 6 & 9 & 0 & 6 & 27 & 0 & 0 & 0 \end{vmatrix}$$

Na taj način, pomoću svojstava *a*) i *c*) možemo postići da sve vrste nove tabele T_2 počinju istim elementom. $\mathrm{NZD}(f(x), g(x)$ biće sada jednak proizvodu stepena x^k i NZD polinoma predstavljenih tabelom T_2. U slučaju da je polinom koji odgovara drugoj vrsti dobijene tabele manjeg stepena od polinoma pridruženog prvoj vrsti te tabele izvršiće se zamena vrsta u tabeli. Na taj način će se postići da prvoj vrsti svake sledeće tabele odgovara polinom manjeg stepena.

Svojstvo *b*) se sada može upotrebiti da bi se prvi član u drugoj vrsti dobijene tabele promenio u nulu. Potrebno je samo drugu vrstu zameniti razlikom prve i druge vrste. Pomoću svojstva *c*) možemo ponovo ukloniti sve vodeće nule u drugoj vrsti novodobijene tabele, ako te nule tamo postoje a zatim pomoću svostva *a*) postići da prvi koeficijenti u vrstama budu međusobno jednaki. Ovi postupci će se ponavljati sve dok druga vrsta ne bude sadržavala samo nule. Na taj način, $\mathrm{NZD}(f(x), g(x))$ biće jednak proizvodu stepena x^k i polinoma koji odgovara ne nula vrsti dobijene tabele.

Korišćenjem činjenice da je $\mathrm{NZD}(f_1, f_2, f_3) = \mathrm{NZD}(\mathrm{NZD}(f_1, f_2), f_3)$ dobili bi iterativan algoritam za nalaženje NZD skupa od s polinoma $f_1, f_2, \ldots, f_s$. Međutim, mnogo efikasniji algoritam može se dobiti radeći slično razmotrenom algoritmu za NZD dva polinoma. Umesto uzastopnih iteracija određivanja NZD od dva polinoma može se raditi istovremeno sa celim skupom od s polinoma predstavljenih u tabeli sa s vrsta.

4 Ilustracija algoritma

Neka su dati polinomi

$$\begin{aligned} p_1(x) &= x^2 + 5x^3 - 5x^4 - x^5, \\ p_2(x) &= x - 3x^2 + 3x^3 - x^4, \\ p_3(x) &= x - x^4, \\ p_4(x) &= x - x^2. \end{aligned}$$

Odredimo najveći zajednički delilac ovih polinoma.

Označimo sa $d_T(x)$ traženi NZD i pridružimo datim polinomima p_1, p_2, p_3 i p_4 tabelu

$$T = \begin{vmatrix} 0 & 0 & 1 & 5 & -5 & -1 \\ 0 & 1 & -3 & 3 & -1 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \end{vmatrix}$$

dimenzije 4×6 sastavljenu od njihovih koeficijenata. Prema svojstvu *d)* imali bi da je $d_T(x) = x \cdot d_{T_1}(x)$, gde je d_{T_1} NZD od polinoma koji odgovaraju tabeli:

$$T_1 = \begin{vmatrix} 1 & -1 & 0 & 0 & 0 \\ 1 & -3 & 3 & -1 & 0 \\ 1 & 0 & 0 & -1 & 0 \\ 0 & 1 & 5 & -5 & -1 \end{vmatrix}$$

dobijenoj iz tabele T prvo uklanjanjem nul kolona, a zatim zamenom prve i poslednje vrste kako bi polinom najmanjeg stepena odgovarao prvoj vrsti tabele T. Prema svojstvu *c)* možemo, ne menjajući $d_{T_1}(x)$, polinom koji odgovara četvrtoj vrsti tabele podeliti sa x, budući da sa x nisu deljivi ostali polinomi koji odgovaraju toj tabeli te x nije faktor u NZD $d_{T_1}(x)$, i na taj način dobiti tabelu

$$T_2 = \begin{vmatrix} 1 & -1 & 0 & 0 \\ 1 & -3 & 3 & -1 \\ 1 & 0 & 0 & -1 \\ 1 & 5 & -5 & -1 \end{vmatrix}$$

u kojoj su poslednje dve nul kolone izostavljene. Prema svojstvu *b)* možemo, ne menjajući $d_{T_1}(x)$, postići da svi elementi prve kolone tabele T_2, počevši od druge vrste budu jednaki nuli i tako doći do tabele

$$T_3 = \begin{vmatrix} 1 & -1 & 0 & 0 \\ 0 & -2 & 3 & -1 \\ 0 & 1 & 0 & -1 \\ 0 & 6 & -5 & -1 \end{vmatrix}$$

a od nje, prema svojstvu *c)* i uklanjanjem poslednje nul kolone, do tabele

$$T_4 = \begin{vmatrix} 1 & -1 & 0 \\ -2 & 3 & -1 \\ 1 & 0 & -1 \\ 6 & -5 & -1 \end{vmatrix}$$

Odavde se pomoću svojstava *a)* i *b)* dolazi do tabele

$$T_5 = \begin{vmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ 0 & 1 & -1 \\ 0 & 1 & -1 \end{vmatrix}$$

Korišćenjem svojstva *c)* i izostavljanjem poslednje nul kolone, sada se odmah dobija tabela

$$T_6 = \begin{vmatrix} 1 & -1 \\ 1 & -1 \\ 1 & -1 \\ 1 & -1 \end{vmatrix}$$

odnosno, prema svojstvu *b)* i uklanjanjem poslednjih nul vrsta—tabela

$$T_7 = \begin{vmatrix} 1 & -1 \end{vmatrix}$$

kojoj odgovara samo jedan polinom $d_{T_1}(x) = 1 - x$.

Na taj način NZD datih polinoma je $D_T(x) = x \cdot d_{T_1}(x) = x(1-x) = x - x^2$.

5 Programska realizacija

Na osnovu svega rečenog sada možemo napisati program kojim će se odrediti NZD zadatih polinoma.

```
10  REM NZD POLINOMA
20  INPUT "BROJ POLINOMA N=";N
30  INPUT "NAJVECI STEPEN OD
        SVIH POLINOMA Q=";Q
40  LET M=Q+1
50  DIM A(N,M) : DIM S(N)
60  FOR I=1 TO N
70  INPUT "STEPEN POLINOMA P"
        (I);"=";S(I)
80  IF M>S(I) THEN LET M=S(I):
                LET W=I
90  PRINT "P";I;"=";
100 FOR J=1 TO S(I)+1
110 INPUT "A(";(I);",";(J-1);
        ")=";A(I,J)
120 IF A(I,J)=0 THEN GO TO 150
130 IF A(I,J)>0 THEN PRINT "+";
140 PRINT A(I,J);"X^";J-1
150 NEXT J
```

```
160  PRINT
170  NEXT I
180  IF W=1 THEN GO TO 210
190  GO SUB 680
200  GO SUB 1070
210  LET J=1
220  FOR I=1 TO N
230  IF A(J,I)=0 THEN NEXT I:
                      LET J=J+1;
                      GO TO 220
240  LET K=J-1
250  IF K=0 THEN LET X=0:
                  GO TO 280
260  GO SUB 1400
270  LET X=Q+1
280  GO SUB 560
290  GO SUB 400
300  GO SUB 1170
310  LET I=2
320  GO SUB 1260
330  IF Y=Q+1 THEN GO SUB 790
340  LET I=I+1
350  IF I<=N THEN GO TO 320
360  GO SUB 1070
370  IF N<>1 THEN GO TO 270
380  GO SUB 880
390  STOP

400  REM BROJ NULA KOJIM
         ZAVRSAVA PRVA VRSTA
410  LET T=0
420  FOR J=Q+1 TO 1 STEP -1
430  IF A(I,J)<>0 THEN GO TO 460
440  LET T=T+1
450  NEXT J
460  RETURN

470  REM UKLANJANJE VODECIH
         NUL KOLONA
480  FOR R=P TO Q+1
490  LET A(I,R-P+1)=A(I,R)/F
500  NEXT R
510  FOR R=Q-P+3 TO Q+1
520  LET A(I,R)=0
```



```
530  NEXT R
540  LET S(I)=S(I)-P+1
550  RETURN

560  REM KOLONA PRVOG NE NULA
         ELEMENTA U SVAKOJ VRSTI
570  FOR I=1 TO N
580  LET P=1
590  IF A(I,P)=0 THEN LET P=P+1
                       GO TO 590
600  GO SUB 1500
610  IF P*F=1 THEN GO TO 660
620  GO SUB 470
630  IF P=1 THEN GO TO 660
640  IF M>S(I) THENLET M=S(I):
                    LET W=1
650  IF X>P-1 THEN LET X=P-1
660  NEXT I
670  RETURN

680  REM POLINOM NAJNIZEG STEPENA
         U PRVU VRSTU
690  FOR J=1 TO Q+1
700  LET T=A(I,J)
710  LET A(1,J)=A(W,J)
720  LET A(W,J)=T
730  NEXT J
740  LET T=S(1)
750  LET S(1)=S(W)
760  LET S(W)=T
770  LET W=1
780  RETURN

790  REM UKLANJANJE NUL POLINOMA
800  FOR G=I TO N-1
810  FOR H=1 TO Q+1
820  LET A(G,H)=A(G+1,H)
830  NEXT H
840  NEXT G
850  LET I=I-1
860  LET N=N-1
870  RETURN
```

```
880  REM STAMPANJE NZD POLINOMA
890  PRINT ' "NZD=";
900  FOR J=1 TO M+1
910  IF A(I,J)=0 THEN GO TO 990
920  IF A(I,J)>0 THEN PRINT "+";
930  IF A(I,J)=1 AND K+J-1<>0
     THEN GO TO 970
940  IF A(I,J)=-1 AND K+J-1<>0
     THEN PRINT "-";:
          GO TO 970
950  PRINT A(I,J);
960  IF K+J-1=0 THEN GO TO 990
970  IF K+J-1=1 THEN PRINT "X";:
                     GO TO 990
980  PRINT "X^";K+J-1;
990  NEXT J
1000 RETURN

1010 REM NZD DVA BROJA
1020 IF F=0 THEN LET F=E:
                 GO TO 1060
1030 LET D=INT (E/F)
1040 LET Z=E-D*F
1050 IF Z<>0 THEN LET E=F:
                  LET F=Z:
                  GO TO 1030
1060 RETURN

1070 REM MEDJUREZULTAT
1080 PRINT
1090 FOR G=1 TO N
1100 FOR H=1 TO Q+1
1110 IF A(G,H)>=0 THEN
        PRINT " ";
1120 PRINT A(G,H);"  ";
1130 NEXT H
1140 PRINT
1150 NEXT G
1160 RETURN

1170 REM TABELA BEZ POSLEDNJIH
        NUL KOLONA
1180 IF X<>T THEN LET X=T
1190 LET Q=Q-X
```



```
1200 IF X=0 THEN GO TO 1220
1210 GO SUB 1070
1220 IF W=1 THEN GO TO 1250
1230 GO SUB 680
1240 GO SUB 1070
1250 RETURN

1260 REM ANULIRANJE DELA PRVE
         KOLONE POCEV OD VRSTE 2
1270 LET B=A(1,1)
1280 LET C=A(1,1)
1290 LET E= ABS (B)
1300 LET F= ABS (C)
1310 GO SUB 1010
1320 LET Y=0
1330 FOR J=1 TO Q+1
1340 LET U=A(I,J)*B
1350 LET V=A(I,J)*C
1360 LET A(I,J)=(U-V)/F
1370 IF A(I,J)=0 THEN LET Y=Y+1
1380 NEXT J
1390 RETURN

1400 REM TABELA BEZ
         VODECIH NUL KOLONA
1410 LET P=K+1
1420 FOR I=1 TO N
1430 GO SUB 1500
1440 GO SUB 470
1450 NEXT I
1460 LET Q=Q-P+1
1470 LET M=M-P+1
1480 GO SUB 1070
1490 RETURN

1500 REM NZD VRSTE I
1510 LET F=ABS(A(I,P))
1520 FOR J=P+1 TO Q+1
1530 LET E=ABS (A(I,J))
1540 IF E<>0 THEN GO SUB 1010
1550 NEXT J
1560 RETURN
```

U navedenom programu su najveći i najmanji stepen od svih stepena s_i, $i = 1, 2, \ldots, n$, n datih polinoma p_i, $i = 1, 1, \ldots, n$ označeni redom sa q i m. Redni broj polinoma stepena m označen je sa w. Budući da ćemo datim polinomima uređenim po rastućim stepenima nepoznate pridružiti odgovarajuću tabelu dimenzije $n \times (q+1)$ njihovih koeficijenata, za koje pretpostavimo da su celi brojevi, to će u vrsti w te kolone počevši od kolone $m+2$ biti same nule. Broj prvih nul kolona te tabele u programu je označen sa k.

Ako je $k \neq 0$ pozvaće se potprogram PP-1400 u kojem će se sa p označiti redni broj prve ne nula kolone a zatim odrediti nova tabela u kojoj neće biti prvih k tj. p-1 nul kolona stare tabele, a elementi svake vrste biće skraćeni sa najvećim zajedničkim deliocem f elemenata te vrste, određenim u potprogramu PP-1500. Kako je novodobijena tabela uža za prvih p-1 kolona u odnosu na prethodnu to će njenoj najdužoj vrsti odgovarati polinom stepena $q-(p-1)$ koji će se opet označiti sa q tj. broj kolona nove tabele će biti jednak $q-p$. Naime, uklanjanju prvih $p-1$ kolona tabele odgovara deljenje svih polinoma sa x^{p-1}. Iz istog razloga će najniži stepen polinoma koji odgovara novodobijenoj tabeli biti $m-(p-1)$.

U potprogramu PP-560 će se za svako i, $i = 1, 2, \ldots, n$, pre pomeranja elemenata vrsta za odgovarajući broj $p-1$ kolona nalevo, gde je p broj kolone prvog od nule različitog elementa u i-toj vrsti, pozivati i potprogram PP-1500 za određivanje najvećeg zajedničkog delioca f svih elemenata vrste i počev od njenog prvog različitog od nule elementa $a_{i,p}$. Potprogram PP-470 se neće pozvati, odnosno pomeranje elemenata i-te vrste se neće vršiti ako je taj delilac jednak 1 i, u isto vreme, prvi nenula element i-te vrste u prvoj koloni. U suprotnom slučaju će se u potprogramu PP-470 posle pomeranja elemenata i-te vrste za $p-1$ kolona nalevo, stepen s_i polinoma koji odgovara i-toj vrsti tabele da smanji za $p-1$ i ponovo označi sa s_i. U potprogramu PP-560 određuje se i prvi najniži stepen m od svim stepena s_i, $i = 1, 2, \ldots, n$, polinoma koji odgovaraju vrstama posmatrane tabele kao i redni broj w tog polinoma.

Da bi izračunali koliko u posmatranoj tabeli ima poslednjih uzastopnih nul kolona koje možemo ukloniti i tako odgovarajuću tabelu skratiti, izračunaćemo potprogramom PP-560 minimalan broj x vodećih nula u vrsti među svim vrstama tabele i potprogramom PP-400 broj t nula kojim završava prva vrsta tabele budući da se u njoj nalaze koeficijenti polinoma najnižeg stepena pa ona završava sa najvećim brojem t nula. Na taj način ako prva vrsta završava nenula elementom, onda će t biti jednako nuli. Isto tako je minimalan broj x vodećih nula u vrsti jednak nuli ako tabela ne počinje nul kolonom tj. ako je za neko i, $i = 1, 2, \ldots, n$, $p = 1$, gde je p redni broj prve kolone tabele sa ne nula elementom u i-toj vrsti. U suprotnom slučaju znamo samo da x nije veće od broja kolona $q+1$ posmatrane tabele. Da bi i u tom slučaju odredili minimalan broj x početnih nula, izračunaćemo u potprogramu PP-560 za svaku vrstu i, broj $p-1$ njenih vodećih nula i na kraju za x uzeti minimalnu vrednost između svih izračunatih vrednosti za $p-1$. Budući da će se višestrukim pozivanjem potprograma PP-470 ukloniti vodeće nule iz svake vrste tabele i na njihova mesta upisati redom ostali elementi te vrste, (podeljeni sa najvećim zajedničkim deliocem f elemenata te vrste, određenim u potprogramu

PP-1500), a na ranija mesta tih nenula elemenata upisaće se nule, to će se broj poslednjih nula u tim vrstama povećati za odgovarajući broj $p-1$ nula.

Na taj način poslednjih $\min\{x, t\}$ kolona posmatrane tabele biće nul kolone pa će se potprogramom PP-1170 one iz nje ukloniti tako što će se najveći stepen q, od svih posmatranoj tabeli odgovarajućih polinoma, da smanji za $\min\{x, t\}$ i ponovo označi sa q.

U slučaju da je u potprogramu PP-560 došlo do pomeranja nalevo elemenata neke vrste tj. ako je $x \neq 0$, onda će se u potprogramu PP-1170 prvo pomoću potprograma PP-1070 odštampati dobijena tabela, a zatim ako koeficijenti polinoma najnižeg stepena odgovaraju vrsti $w \neq 1$ pozvati potprogram PP-680 i izvršiti međusobna zamena odgovarajućih elemenata prve vrste i vrste w, vodeći računa da se posle toga stepeni s_i i s_w koji odgovaraju polinomima tih vrsta takođe međusobno zamenjuju i da promenljiva w postaje jednaka jedan.

Sada se pristupa potprogramu PP-1260 pomoću kog će se anulirati u dobijenoj tabeli svi elementi prve kolone počev od druge vrste. Anuliranje će se izvesti tako što će se za svaku vrstu i, $i = 1, 2, \ldots, n$, prvo u potprogramu PP-1010 da odredi najveći zajednički delilac f elemenata b i c, prvih u prvoj i i-toj vrsti tabele, respektivno, a zatim svaki element i-te vrste zameniti razlikom odgovarajućih proizvoda elemenata i-te vrste sa b/f i prve vrste sa c/f.

Paralelno sa određivanjem u potprogramu PP-1260 novih elemenata i-te vrste izračunaće se i broj y nula u toj vrsti. Ako je taj broj jednak broju $q+1$ kolona posmatrane tabele, onda će se pomoću potprograma PP-790 da smanji broj n vrsta te tabele i izvrši pomeranje svih vrsta počev od $i+1$-ve za jedno mesto naviše čime se ranija i-ta vrsta zamenjuje sa $i+1$-vom itd.

LITERATURA

[1] D. E. Knuth, *The Art of Computer Programming—Seminumerical Algorithms* (Volume 2), Addison–Wesley, MA, 1969

[2] Đ Kurepa, *Viša algebra*, I deo, Zavod za izdavanje udžbenika, Beograd, 1971

RAČUNARSTVO
Sveska 1, Broj 2, 1992

UDK 519.682

METODIČKI ASPEKTI OPISA SEMANTIKE PROGRAMSKIH JEZIKA[1]

NEDELJKO PAREZANOVIĆ

SAŽETAK. Opis semantike programskih jezika predstavlja neophodan korak u izučavanju i nastavi programskih jezika. Medjutim, složenost notacije koja se koristi u ove svrhe čini istu teško prihvatljivom u nastavi i učenju programskih jezika. U ovom članku je pokazan jedan jednostavniji prilaz u stvaranju semantičkih modela za potrebe nastave i učenja programskih jezika.

1 Uvod

Poznavanje sintakse jezika je samo jedan deo u izučavanju programskih jezika. Drugi deo se odnosi na poznavanje značenja svake sintaksne jedinice jezika. Nauka koja izučava značenje jezičkih konstrukcija zove se *semantika*. Dok je formalni, a to znači precizan, opis sintakse relativno jednostavan, formalni opis semantike je znatno složeniji problem [1]. Medjutim, interes za formalni opis semantike je isto tako veliki kao i za formalni opis sintakse. Cilj formalnog opisa semantike bio bi da se isključe sve nepreciznosti koje može da proizvede primena prirodnog jezika u opisu semantike programskog jezika. Medjutim, s obzirom na složenost formalnog opisa semantike jezika, u knjigama i priručnicima se, najčešće, koristi prirodan jezik za opis semantike programskih jezika [3].

Mi ćemo ukratko prikazati prilaze formalnom opisu semantike, a detaljnije ćemo izložiti neke prilaze u stvaranju semantičkih modela u nastavi programskih jezika. Dakle, naš cilj nije da stvorimo konzistentan aparat za opis semantike, već da ukažemo na neke mogućnosti stvaranja korektnih semantičkih modela o složenim naredbama programskih jezika. Ovo se naročito odnosi na naredbe kojima se zadaju programske strukture.

[1]Rad finansiran od Fonda za nauku Republike Srbije preko Matematičkog instituta, projekat 0401A.

2 Formalni opis semantike

Nepostojanje šire prihvaćene notacije za opis semantike programskih jezika najbolje ilustruje težinu problema. Medjutim, postoji nekoliko prilaza u rešavanju ovog problema. To su sledeći prilazi:

- operacijska semantika,
- aksiomatska semantika i
- denotacijska semantika.

Sva tri prilaza su isuviše složena da bi mogli biti prihvaćeni u nastavi programskih jezika. Mi ćemo ih ukratko prikazati da bismo uočili suštinu prilaza i mogućnost delimičnog korišćenja ideja u opisu semantike programskih jezika. U ovom smislu posebnu pažnju, po našem mišljenju, zaslužuje operacijska semantika.

2.1 Operacijska semantika

Da bismo objasnili ideju na kojoj se zasniva operacijska semantika, posmatrajmo mašinski jezik i računar. *Mašinski jezik* je skup binarnih reči za koje postoji interpretacija u računaru. Tako, za operaciju sabiranja postoji, u mašinskom jeziku, binarna reč kojom se definiše ova operacija i adrese registara u kojima se nalaze brojevi koje treba sabrati. Dakle, pre izvršenja ove operacije postoje odredjena stanja registara u računaru koja će biti, ovom instrukcijom, promenjena. To, kako će ova stanja biti promenjena, predstavlja, zapravo, semantiku ove konstrukcije mašinskog jezika. Tako se računar javlja kao interpretator konstrukata mašinskog jezika. Definisanjem svih promena stanja, koje pojedini konstrukti mašinskog jezika proizvode na računaru, možemo u potpunosti, precizno definisati značenje svakog od njih, a to znači, možemo definisati semantiku mašinskog jezika.

Ovu ideju možemo proširiti na programske jezike, tako, što ćemo za svaki programski jezik definisati realnu ili apstraktnu mašinu, na kojoj ćemo pratiti promene stanja koje proizvode pojedini konstrukti programskog jezika. Ovako definisana mašina biće interpretator odredjenog programskog jezika. Naravno, za ovo nije dobro izabrati realnu mašinu, jer bi semantičke definicije bile vezane za jednu konkretnu mašinu, a to znači i za sve specifičnosti odredjene arhitekture računara. Zato je izbor apstraktne mašine pogodniji. Pokazaćemo ovo na jednom primeru. Posmatrajmo brojački programski ciklus u Pascal-jeziku:

for i := pocetak to kraj do

⋮

Značenje ovakvog zapisa ne može biti lako razumljivo. Stvarni mehanizam izvršavanja ciklusa je skriven za korisnika. Naravno, oni koji poznaju razne zapise programskih ciklusa mogu lako pretpostaviti značenje gore navedenog zapisa. Medjutim, i

za njih će biti teško da odgovore, šta će se dogoditi ako je vrednost promenljive *pocetak* veća od vrednosti promenljive *kraj*?

Zamislimo, sada, da naša apstraktna mašina raspolaže sa instrukcijama dodele, uslovnog i bezuslovnog prelaska. Tada bi se gornji ciklus mogao zapisati instrukcijama apstraktne mašine u obliku:

```
        i := pocetak
ciklus: if i >kraj then goto izlaz
                ⋮
        i := i + 1
        goto ciklus
izlaz:  ...
```

Sada je jasno značenje programskog ciklusa, pa i to da se telo ciklusa neće izvršiti ako je vrednost promenljive *pocetak* veća od vrednosti promenljive *kraj*.

2.2 Aksiomatska semantika

Aksiomatska semantika je zasnovana na matematičkoj logici. Osnovni motiv za ovaj prilaz u opisu semantike jezika jeste razvoj metoda za proveru korektnosti programa. Za svaku jezičku konstrukciju definišu se logički izrazi u kojima se zadaju ograničenja za promenljive koje se javljaju u odgovarajućoj jezičkoj konstrukciji. Ovi izrazi se zovu *tvrdjenja* (engl. assertions). Logički izraz u kome se definišu ograničenja promenljivih pre izvršavanja jezičke konstrukcije zove se *preuslov* (engl. precondition), a onaj posle izvršavanja zove se *postuslov* (engl. postcondition). Tako, za naredbu dodele

$$y := 3 \cdot x + 7$$

ako je postuslov $y > 31$, onda je preuslov $x > 8$. Naravno, korektni preuslovi biće i $x > 10$, $x > 25$, $x > 3000$ itd. Medju svim ovim preuslovima postoji jedan koji daje najmanja ograničenja na vrenost promenljive x, a za koju će postuslov biti tačan. U našem primeru to je preuslov $x > 8$. Ovakav preuslov se zove *najslabiji preuslov*.

U aksiomatskoj semantici je uobičajena notacija za konstrukt S u jeziku sa preuslovom P i postuslovom Q u obliku

$$\{P\}S\{Q\}$$

Funkcija kojom se za zadati jezički konstrukt S i postuslov Q definiše najslabiji preuslov P zove se *predikatski transformator* (T), tj.

$$P = T(S, Q)$$

Pokazaćemo proces odredjivanja najslabijeg preuslova za slučaj naredbe dodele. Uzmimo ranije navedenu naredbu

$$y := 3 \cdot x + 7$$

i postuslov $y > 31$. Zamenjujući promenljivu y u postuslovu sa izrazom u naredbi dodele, dobijamo

$$3 \cdot x + 7 > 31$$

Odavde sledi

$$x > 8$$

Prema tome, za zadatu naredbu dodele $\alpha := \beta$, gde su α–promenljiva, β–izraz, predikatski transformator se dobija

$$T(\alpha := \beta, Q) = Q$$

tj. u uobičajenoj notaciji

$$\{Q_{\alpha \to \beta}\}\alpha := \beta\{Q\}$$

Što znači, za naredbu dodele, najslabiji preuslov se dobija kada se u postuslovu promenljiva α zameni izrazom β.

Ako bismo za svaki konstrukt jezika odredili postuslov i najslabiji preuslov, tada bismo mogli odrediti postuslov i preuslov za ceo program. Dakle, polazeći od postuslova poslednjeg konstrukta u programu, a to je i postuslov programa, tada vraćajući se prema početku programa, odredili bismo i preuslov prvog konstrukta programa, a to je preuslov programa. Na ovaj način, možemo tvrditi, da za ograničenja zadata u preuslovu programa, biće tačno tvrdjenje koje se javlja u postuslovu programa. Osnovna teškoća u ovom prilazu jeste odredjivanje postuslova i najslabijih preuslova za svaki konstrukt jezika.

Bez obzira na sve teškoće primene aksiomatske semantike u nastavi i učenju programskih jezika, ostaje činjenica da bi provera korektnosti programa u procesu njegovog razvoja bila dragocena. Kada bi ovakav prilaz bio manje složen i praktičniji, to bi u velikoj meri izmenilo način nastave i učenja programiranja. Programiranje bi u potpunosti imalo apstraktni karakter, a praktičan rad na računaru ne bi se odnosio na proveru korektnosti programa, već samo na eksploataciju programa. Naravno, korektnost programa ne možemo dokazati njegovim izvršavanjem na računaru, ali možemo, proverom programa putem izvršavanja, učiniti da program sa manjom ili većom verovatnoćom bude korektan.

2.3 Denotacijska semantika

Denotacijska semantika se sastoji u pridruživanju značenja sintaksnim definicijama jezika. Po ovom svojstvu je i nazvana, jer na engleskom *denote* znači *označiti*. U ovom pristupu se uz sintaksnu definiciju označava i njeno značenje. Pokazaćemo ovo na primeru binarnog broja. Sintaksna definicija binarnog broja može biti zapisana u obliku:

binarni broj :
- 0
- 1
- binarni broj 0
- binarni broj 1

Svaka sintaksna definicija se tretira kao objekat na koji se može primeniti funkcija koja taj objekat preslikava u matematički objekat koji definiše značenje. Značenje binarnog broja uzimamo u obliku njegove vrednosti u dekadnom brojčanom sistemu, a funkciju koja vrši ovo preslikavanje označimo sa N, tada je

$$N(0) = 0$$
$$N(1) = 1$$
$$N(\underline{\text{binarni broj}}\ 0) = 2 \cdot N(\underline{\text{binarni broj}})$$
$$N(\underline{\text{binarni broj}}\ 1) = 2 \cdot N(\underline{\text{binarni broj}}) + 1$$

gde su argumenti funkcije N sintaksne definicije koje se javljaju u sintaksnoj definiciji binarnog broja. Tako se dobija da zapis binarnog broja 101 ima sledeće značenje

$$N(101) = 2 \cdot N(10) + 1 = 2 \cdot [2 \cdot N(1)] + 1 = 5$$

Denotacijska semantika ima veliku primenu u konstrukciji programa za prevodjenje. Ovaj prilaz omogućava konstrukciju sintaksno-vodjenih prevodilaca. Jezički konstrukti za koje je denotacijska semantika složena, po pravilu, predstavljaju složene konstrukcije i za korisnike računara, pa ih treba izbegavati pri projektovanju programskih jezika [2].

3 Semantički modeli u nastavi i učenju

Primena formalnih opisa semantike programskih jezika u nastavi i učenju programskih jezika nije opravdana. Teškoće su višestruke. Primena formalnih opisa semantike zahteva matematičko znanje i apstraktno razmišljanje koje se ne može očekivati od većine korisnika programskih jezika. Učenje programskih jezika bilo bi neopravdano opterećeno formalnim aparatom i složenim opisom semantike. Medjutim, ostaje činjenica da semantiku svake konstrukcije programskog jezika korisnik mora dobro razumeti i stvoriti korektan misaoni model o aktivnostima koje svaka konstrukcija jezika proizvodi u računaru. Ovakve predstave o semantici jezičkih konstrukcija zvaćemo *semantički modeli*. Dakle, naš cilj neće biti izgradnja aparata za formalni opis semantike programskih jezika, već samo da ukažemo kako se za pojedine konstrukcije jezika mogu dati razumljive, lako prihvatljive i tačne semantičke predstave.

Značenje velikikog broja jezičkih konstrukcija u programskim jezicima može se jasno opisati prirodnim jezikom i na osnovu ovakvog opisa korisnici jezika mogu izgraditi misaone modele koji će im omogućiti ispravnu primenu tih konstrukcija. Teškoće u ovakvom opisu nastaju kada sintaksa konstrukcije ne sadrži jasno sve elemente značajne za značenje konstrukcije, već je to na neki implicitan način sadržano u konstrukciji. Ovakve konstrukcije zahtevaju uvodjenje nekih prezentacija koje će omogućiti korisniku lako razumevanje i pamćenje značenja konstrukcije. U ovom smislu mi ćemo koristiti sledeće prilaze:

- svodjenje na poznate operacije i
- vizuelizaciju toka upravljanja.

Dakle, ukazaćemo na mogućnosti stvaranja korektnih semantičkih modela, bez korišćenja formalnog opisa semantike.

3.1 Svodjenje na poznate operacije

Ovaj prilaz se u osnovi zasniva na operacijskoj semantici. Izabraćemo nekoliko naredbi čije su semantike jednostavne i lako razumljive za korisnika, a pomoću kojih se može izraziti aktivnost koja se definiše složenijim naredbama, čije semantike želimo da opišemo. Ovako odabrane naredbe imaju ulogu apstraktne mašine u operacijskoj semantici.

Kako se preko naredbi obrade i uslovnog prelaska mogu izraziti ostale složenije kontrolne naredbe, to ćemo za objašnjenje složenijih naredbi izabrati sledeće naredbe:

naredba dodele:
    promenljiva ::= izraz

naredba uslovnog prelaska:
    if logički izraz goto obeležje

naredba bezuslovnog prelaska:
    goto obeležje

Značenje ovih naredbi je lako razumljivo. Prva naredba, vrši izračunavanje vrednosti izraza i izračunatu vrednost dodeljuje promenljivoj. Druga naredba, vrši prelazak na navedeno obeležje ako je vrednost logičkog izraza tačna i treća naredba, vrši bezuslovni prelazak na navedeno obeležje. Ovde smo, pored uslovnog, uključili i bezuslovni prelazak, jer će omogućiti jednostavniji prikaz složenih naredbi.

Iskoristimo, sada, gornje naredbe za opis sintakse programskog ciklusa u jezicima FORTRAN IV i FORTRAN 77. U oba ova jezika sintaksa naredbe za programski ciklus može se zapisati u obliku:

```
DO obeležje promenljiva = početak, kraj [, korak]
```

Manje je značajna razlika, da u FORTRAN-u IV namesto ciklusnih parametara početak, kraj i korak mogu stajati celi neoznačeni brojevi ili celobrojne promenljive, dok u FORTRAN-u 77 to mogu biti celobrojni ili realni izrazi. Medjutim, koliko je semantika skrivena za korisnika najbolje pokazuje značenje gornje naredbe. Ovo ćemo objasniti na konkretnoj naredbi

```
      DO 100 I = POCETAK, KRAJ, KORAK
        :
100   CONTINUE
```


Semantika ove naredbe u FORTRAN-u IV može se opisati na sledeći način:

```
      I = POCETAK
99      .
        .
        .
      I = I + KORAK
      IF (I .LE. KRAJ) GOTO 99
100   CONTINUE
```

Dakle, ciklus se mora izvršiti najmanje jedanput i parametri ciklusa KORAK i KRAJ ne smeju se menjati u telu ciklusa. Pogledajmo sada značenje istog ciklusa u FORTRAN-u 77:

```
      brojač = (KRAJ - POCETAK)/KORAK + 1
      korak = KORAK
      I = POCETAK
99    IF (brojač .GT. 0) GOTO 98
        :
      I = I + korak
      brojač = brojač - 1
      GOTO 99
100   CONTINUE
98      .
```

U ovom slučaju, ciklus se može ne izvršiti, a ciklusni parametri se mogu i menjati unutar ciklusa, ali to neće uticati na izvršavanje ciklusa, jer se vrednosti parametara koriste za odredjivanje sistemskih promenljivih (korak i brojač) koje se koriste unutar ciklusa. Ovaj primer najbolje ilustruje neophodnost korišćenja dodatnih sredstava za precizan opis semantike.

3.2 Vizuelizacija toka upravljanja

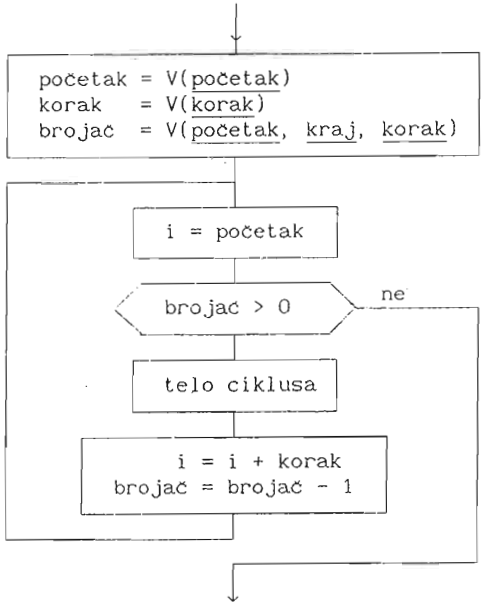
Vizuelizacija je moćno sredstvo u mnogim oblastima istraživanja, a naročito nastave. To je grafičko predstavljanje odnosa i procesa u cilju lakšeg stavaranja misaonih modela o objektu koji je predmet vizuelizacije. Kako grafički prikazi, po pravilu, nose konciznu i jasnu informaciju to su posebno pogodni za korišćenje u nastavi. Mi ćemo ovaj prilaz koristiti za opis semantike konstrukcija programskih jezika. Razlikovaćemo dva pristupa:

- vizuelizacija pomoću tokovnika i
- vizuelizacija nad sintaksnim definicijama.

Pod tokovnikom podrazumevamo grafički prikaz algoritma (engl. flowchart), što je uobičajen prilaz u programiranju.

Vizuelizacija pomoću tokovnika

U ovom slučaju koristimo tokovnike da prikažemo način izvršavanja odredjene konstrukcije u jeziku. Za ovo je najbolje koristiti svodjenje na poznate operacije, a zatim prikaz u obliku tokovnika. Tako, već naveden primer programskog ciklusa



Sl. 1. Tokovnik DO-naredbe u FORTRAN-u 77

u FORTRAN-u 77 mogli bismo prikazati u obliku tokovnika (sl 1). Iz tokovnika vidimo neka važna semantička svojstva naredbe DO, koja se ne mogu ni nazreti iz sitaksne definicije ove naredbe:

- Parametri naredbe se koriste za izračunavanje vrednosti koje se dodeljuju sistemskim promenljivim (početak, korak i brojač) pre ulaska u ciklus. Na taj način je obezbedjeno izračunavanje vrednosti parametara jedanputa, a ne više puta unutar ciklusa.
- U kontroli programskog ciklusa ne učestvuju parametri ciklusa već samo sistemska promenljiva—brojač, čijom vrednošću je odredjen broj ponavljanja ciklusa.
- Uslov za izlazak iz ciklusa se ispituje pre tela ciklusa, što znači da se ciklus može izvršiti nulaputa.

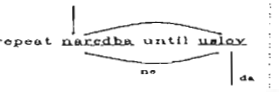
- Po izlasku iz ciklusa ciklusna promenljiva ima vrednost za koju je poslednji put izvršen ciklus uvećanu za korak. Ako se ciklus ne izvrši onda je to početna vrednost.

Očigledno, vizuelizacija svodjenja na poznate operacije, pomoću tokovnika, može doprineti bržem razumevanju i trajnijem pamćenju semantike nekih naredbi.

Vizuelizacija nad sintaksnim definicijama

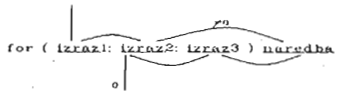
Korisnici programskih jezika moraju naučiti sintaksu jezika. Vezivanje semantike za sintaksne definicije ima višestruke koristi u učenju programskih jezika. To će omogućiti lakše pamćenje sintakse, jer se semantički modeli lakše i duže pamte od strane korisnika. S druge strane, to će omogućiti da sintaksa konstrukcije doprinosi podsećanju korisnika na semantička svojstva konstrukcije. Naš predlog vizuelizacije nad sintaksnim definicijama jezika, omogućiće praćenje toka upravljanja, pri izvršavanju odredjene jezičke konstrukcije na računaru. Tok upravljanja biće definisan usmerenim linijama koje povezuju pojedine sintaksne jedinice u složenim sintaksnim definicijama. Najmanje jedna usmerena linija biće ulazna i najmanje jedna, biće izlazna. Ulazna linija prenosi upravljanje sa prethodne naredbe programa, a izlazna predaje upravljanje sledećoj naredbi programa. Linija će biti neoznačena kada se upravljanje bezuslovno prenosi, odnosno, biće označena ako je prenos upravljanja uslovljen. Pokazaćemo ovo na nekim primerima. Naredba while-do u Pascal-u može se prikazati na sledeći način:

To je pretest ciklus, pa može doći do neizvršavanja ciklusa. Izlazak iz ciklusa se vrši ako izlazni uslov nije zadovoljen. A sada, pogledajmo naredbu repeat-until u Pascal-u:



To je posttest ciklus, pa mora doći do najmanje jednog izvršavanja ciklusa. Izlazak iz ciklusa se vrši ako je izlazni uslov zadovoljen.

Programski ciklus u C-jeziku, koji se obrazuje naredbom for, posebno ima neobična svojstva. Koristeći vizuelizaciju sintaksnih definicija, možemo ga prikazati na sledeći način:
gde naredba može biti obična, složena ili prazna naredba. Ciklus se izvršava ako je vrednost izraz 2 različita od nule. Kako je ovo pretest to može doći do neizvršavanja ciklusa, a to znači da se neće izvršiti ni izraz 3.



4 Zaključak

U nastavi programskih jezika mora se obratiti ozbiljna pažnja preciznom opisu sintakse i semantike jezika. Dok se formalni opis sintakse može, relativno, lako sprovesti [4], to se formalni opis semantike ne može lako uvesti u nastavu programskih jezika, zbog složenosti prilaza i notacije. U ovoj situaciji treba tražiti pogodne forme stvaranja korektnih semantičkih modela jezičkih konstrukcija, koje će omogućiti brzo i jednostavno razumevanje i pamćenje semantike jezika. U ovom radu se predlažu dva prilaza:

- svodjenje na poznate operacije i
- vizuelizacija nad sintaksnim definicijama.

Prvi prilaz je zasnovan na ideji operacijske semantike, a drugi prilaz na grafičkom prikazu toka upravljanja. U ovom drugom slučaju, moguće je koristiti tokovnike i prikazivanje toka upravljanja povezivanjem usmerenim linijama odgovarajućih sintaksnih jedinica u složenim sintaksnim definicijama.

LITERATURA

[1] Ž. Mijajlović: *Semantika programskih jezika*, Računarstvo u nauci i obrazovanju, Broj 1/4, 1987.

[2] R. Sebesta: *Concepts of Programming Languages*, The Benjamin/Cummings Pub. Com., 1989.

[3] R. Sethi: *Programming Languages*, Addison-Wesley Publishing Company, 1989.

[4] N. Parezanović: *Metodički aspekti opisa sintakse programskih jezika*, Računarstvo, Sveska 1, Broj 1, str. 101–111, 1991.

ABSTRACT. A description of the semantics of programming languages is the necessary step in studying and teaching of programming languages. However, the complexity of the notation used for this purpose, makes it hard to be accepted in teaching and learning of programming languages. This paper presents another simple approach for the use of semantical models in teaching and learning of programming languages.

Matematički fakultet
Studentski trg 16
11000 Beograd

U prvom broju časopisa u članku "Algebarski sistemi" autora Steva Šegana i Miloša Miljkovića greškom je izostavljen deo programa. Čitaoci mogu da se obrate autorima ukoliko su zainteresovani za ovaj program.