Croatian Professional Newsletter on Computing and Information Technology - CIT 1, 1993, 1, 83–90

Leksička analiza Aho-Korasickovim automatima u Prologu

Mirko Čubrilo

Fakultet organizacije i informatike, Varaždin, Hrvatska

Lexical Analysis with the Use of Aho-Korasick Automata in Prolog Language

This paper discusses implementation in Prolog and the use of Aho-Korasick automata in lexical analysis. The choice of any of the system approaches known so far depends, among other things, on the choice of the lexical analyser's implementation language. Although all algorithms referred to in the paper are procedural, the algorithm based on Aho-Korasick automata is suitable for an implementation in Prolog as a non-procedural language, since the predicates realizing finite automata can be implemented relatively easy. Besides, as a single-pass algorithm, it recognizes all key-words from the input word. The automaton constructed and the program it is implemented in are a part of a deductive mechanism preprocessor for the estimation of multiple valued dependences of relation data base model.

Ovaj se rad bavi implementacijom u Prologu i primjenom Aho-Korasickova automata u rješavanju problema leksičke analize.

Izbor bilo kojeg od danas poznatih sistemskih pristupa rješavanju problema leksičke analize ovisi među ostalim i o izboru jezika implementacije leksičkog analizatora. Iako su svi algoritmi koji se u radu spominju proceduralnog tipa, algoritam temeljen na Aho-Korasickovim automatima pogodan je za implementaciju u Prologu kao neproceduralnom jeziku, jer se predikati koji ostvaruju konačne automate dadu relativno lako implementirati. S druge strane algoritam je jednoprolazan i u tome jedinom prolazu raspoznaje sve ključne riječi koje se pojavljuju u zadanoj ulaznoj riječi.

Konstruirani automat i program u kojem je implementiran dio su pretprocesora za deduktivni mehanizam za račun višeznačnih ovisnosti relacijskog modela podataka.

Uvod

Leksička je analiza prvi korak u strukturiranju izvornog teksta (programskog koda, ...) u strukture koje se dalje mogu interpretirati (kao formule u odgovarajućem jeziku, ...). Jezgru svakoga leksičkog analizatora čini "modul" za raspoznavanje ključnih riječi u izvornom tekstu. Skup ključnih riječi koje u tijeku leksičke analize izvornog teksta treba raspoznati može se raspoznati i priručnim sredstvima jezika u kojem se leksički analizator implementira. Međutim, takav pristup obično ne zadovoljava s obzirom na utrošak resursa računala, što se posebno odnosi na vrijeme procesiranja i memoriju.

Među sistemskim pristupima izdvaja se nekoliko algoritama, kao algoritam Rabina i Karpa [6], Knutha, Morrisa i Pratta [7], Boyera i Moorea [2], Commentz-Walthera [3] te Ahoa i Korasicka [1]. Ahoov članak [1] daje detaljan prikaz svakoga od njih.

U ovom se radu bavim implementacijom i primjenom Aho-Korasickova algoritma u kontekstu Prologa kao jezika implementacije. Konstruirani automat i program u kojemu je implementiran dio su pretprocesora deduktivnog mehanizma za račune funkcijskih i višeznačnih ovisnosti, razvijenog u Prologu i osnovanog na pravilu rezolucije za račun sudova. Ulaz u deduktivni mehanizam čine skupovi disjunkta nastali preoblikom izvornih problema (logičkog) izvoda u spomenutim računima.

Izvorno se problem izvoda stvara kao tekstovna datoteka s popisom ovisnosti-premisa i ovisnosti--moguće-logičke-posljedice i zapisom u skladu sa sintaksom spomenutih računa.

Preoblikovanje izvornog zapisa problema izvoda u oblik prihvatljiv za deduktivni mehanizam odvija se u ovim koracima:

- od izvornog teksta iterativno se otkidaju dijelovi teksta koje dalje obrađuje leksički analizator;

- leksički analizator od svakog odlomka teksta koji potencijalno predstavlja funkcijsku ili višeznačnu ovisnost stvara listu (kao strukturu podataka u Prologu) leksema, tj. ireducibilnih (u kontekstu jezika spomenutih računa) semantičkih jedinica teksta;
- svaka tako dobivena lista leksema predstavlja ulaz u sintaktički analizator koji od nje (ako je to moguće) strukturira ovisnost odgovarajućeg tipa u oblik odgovarajućeg terma Prologa;
- svaka na opisani način strukturirana ovisnost preoblikuje se dalje u odgovarajući skup disjunkta.

U radu je opisana implementacija fragmenta leksičkog analizatora kojemu je zadatak raspoznavanje ključnih riječi računa funkcijskih i višeznačnih ovisnosti.

Za razumijevanje daljnjeg izlaganja nije potrebno nikakvo poznavanje bilo računa relacijskog modela bilo pravila rezolucije za račun sudova. Sve što je o njima rečeno, rečeno je zbog postavljanja odabranoga testnog skupa ključnih riječi u širi kontekst problema leksičke analize. Za svrhe konstrukcije automata i implementacije algoritma u Prologu jednako bi dobro poslužio i bilo koji drugi skup ključnih riječi. Ipak, čitateljima koji će možda poželjeti dodatne informacije preporučujem knjige [4], [8] i [9].

Sadržaj ostatka teksta po odjeljcima je sljedeći:

U drugom odjeljku opisujem strukturu Aho-Korasickova automata prema članku Aho [1].

U trećem odjeljku izlažem detalje konstrukcije Aho-Korasickova automata za odabrani skup ključnih riječi. Kako je već rečeno, to je skup ključnih riječi računa višeznačnih i funkcijskih ovisnosti.

U četvrtom odjeljku dajem implementaciju automata i algoritma u jeziku PDC Prolog. Napominjem da program bez ikakvih izmjena radi i u Turbo Prologu jer PDC Prolog predstavlja nadgradnju Turbo Prologa, a u programu su upotrebljavani primitivi (ugrađeni predikati) iz njihove zajedničke jezgre.

U petom odjeljku izlažem rezultate primjene programa na zadanu ulaznu riječ. Praćenje programa obavljeno je interaktivno, a njegovi rezultati preusmjereni su u sistemsku datoteku PROLOG.LOG. Prikazani ulomci datoteke .LOG pokazuju pojedine faze raspoznavanja ključnih riječi u zadanoj ulaznoj riječi. U izvorni tekst datoteke .LOG umetnuti su komentari koji objašnjavaju rad algoritma.

U posljednjem, šestom odjeljku dani su prijedlozi za poboljšanje programa, primjenjivi ne samo na izloženi program već i na druge programe toga tipa i te namjene.

Struktura Aho-Korasickova automata

Aho-Korasickov automat čine sljedeće komponente:

1. Konačni skup Q unutarnjih stanja
2. Konačna ulazna abeceda Σ
3. Izravna funkcija prijelaza stanja g: $Q \times \Sigma \rightarrow Q \cup \{\text{neuspjeh}\}$
4. Povratna funkcija prijelaza stanja: h: $Q \rightarrow Q$
5. Početno stanje q_0
6. Skup završnih stanja Φ

Sljedeći program (u pseudo kodu Pascal) pokazuje kako taj automat u ulaznoj riječi $s = s_1 s_2 \ldots s_n$ pronalazi ključnu riječ iz zadanog skupa takvih riječi.

```
begin
    q = q0
    for j = 1 to n do
    begin
        while g(q,sj) = neuspjeh do q = h(q)
        q = g(q,sj)
        if q∈Φ then return "da"
        end
    return "ne"
end
```

Na početku je automat u stanju q_0 i procesira znak s_1 (prvi znak) ulazne riječi. Automat zatim ostvaruje niz prijelaza stanja. Ako je trenutno procesirani znak s_j, automat ostvaruje jedan ili više prijelaza stanja povratnom funkcijom prijelaza stanja, sve dok se ne postigne stanje q za koje je $g(q,s_j) \neq$ neuspjeh. Procesiranje završava dodatnim prijelazom u stanje što ga određuje izravna funkcija prijelaza stanja. Ako je to stanje završno stanje, onda automat prekida rad i daje odgovor "da". U protivnom procesira se sljedeći znak u riječi.

Izravna i povratna funkcija prijelaza stanja imaju sljedeća svojstva:

1. $g(q_0,a) \neq$ neuspjeh, za svako $a \in \Sigma$
2. Ako je $h(q) = q'$, onda je dubina od q' manja nego dubina od q, gdje je dubina stanja jednaka duljini najkraćeg niza izravnih prijelaza iz početnog stanja u zadano stanje.

Prvo svojstvo osigurava da ne bude prijelaza stanja iz početnog stanja uvjetovanih povratnom funkcijom prijelaza stanja.

Drugo svojstvo osigurava da ukupan broj prijelaza stanja ostvarenih povratnom funkcijom prijelaza, potrebnih za procesiranje ulazne riječi, bude manji od ukupna broja prijelaza generira-nih izravnom funkcijom prijelaza stanja.

Kako se za svaki znak u riječi ostvaruje točno jedan prijelaz stanja izravnom funkcijom prijelaza stanja, onda ukupan broj prijelaza nije veći od 2n.

Slijedi opis konstrukcije automata na osnovi zadanog skupa riječi $W = \{w_1, ..., w_k\}$.

Na osnovi skupa riječi konstruira se stablo u kojem svaki čvor predstavlja početni komad (prefiks) neke riječi iz *W*. Čvorovi u stablu predstavljaju stanja automata kojega je korijen početno stanje q_0, koje predstavlja prazan prefiks. Svaki čvor koji pripada potpunoj riječi predstavlja završno stanje.

1. Izravna je funkcija prijelaza stanja *g* definirana tako da čim bude $g(q,p_i) = q'$, onda q odgovara pre-fiksu $p_1p_2 \ldots p_{(i-1)}$, a q' prefiksu $p_1,p_2, ..., p_i$.
2. Za stanje q_0 stavlja se $g(q_0,a) = q_0$ za svako $a \in \Sigma$ za koje $g(q_0,a)$ nije bilo definirano u prethodnom koraku.
3. Stavlja se $g(q,a) =$ neuspjeh za sva q i a za koja $g(q,a)$ nije bilo definirano u prethodna dva koraka.

Gornja tri koraka definiraju izravnu funkciju prijelaza stanja. Uočimo da stanje q_0 ima svojstvo da je $g(q_0,a) \neq$ neuspjeh za svako $a \in \Sigma$.

Definicija: Funkcija povrata f: Q→Q na čvorovima stabla definira se na sljedeći način:

Ako su q_u i q_v stanja koja pripadaju prefiksima u i v nekih riječi iz *W*, onda je $f(q_u) = q_v$ ako i samo ako je v pravi sufiks maksimalne duljine od u koji je ujedno prefiks neke riječi u *W*.

Funkcija povrata može se izračunati pomoću tehnike prolaza stabla "najprije u širinu". To znači da se prije prolaza (obrade) čvorova dubine d+1 obrađuju svi čvorovi dubine d.

1. Stavlja se $f(q_0) = q_0$ i $f(q) = q_0$ za sva stanja q dubine 1.
2. Pretpostavimo da je f definirano za sva stanja dubine manje od $d \geq 2$ i da je $g(q,a) = q'$, gdje je q' stanje dubine d. Stavljamo $f(q') = g(r,a)$, gdje je r stanje koje računa sljedeći program:

```
r = f(q)
while g(r,a) = neuspjeh do r = f(r)
```

Uočimo da će petlja **while** uvijek okončati rad jer je dubina od **f**(r) uvijek manja nego dubina od r za sva stanja r osim nultog i $g(q_0,a) \neq$ neuspjeh.

Funkcija povrata i sama bi mogla poslužiti kao povratna funkcija prijelaza stanja. Međutim, ona katkad može proizvesti više prijelaza negoli je prijeko potrebno.

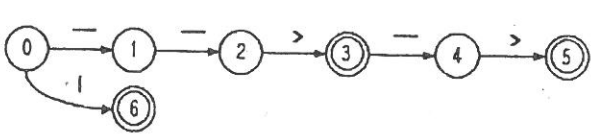
Povratna funkcija prijelaza *h* koja izbjegava nepotrebne prijelaze može se iz funkcije f konstruirati na sljedeći način:

1. $h(q_0) = q_0$
2. Pretpostavimo da je funkcija *h* definirana za sva stanja dubine manje od d i neka je q stanje dubine d. Ako je skup znakova za koje je u stanju f(q) vrijednost izravne funkcije prijelaza različita od "neuspjeh" podskup analognog skupa znakova za stanje q, onda je $h(q) = h(f(q))$. Inače je $h(q) = f(q)$.

Primjer konstrukcije Aho-Korasickova automata

Kao primjer konstrukcije Aho-Korasickova automata dajem konstrukciju automata koji raspoznaje skup riječi $W = \{-->, -->- >|\}$. Ovaj skup (sa zarezom i zagradama) predstavlja skup ključnih riječi računa višeznačnih i (funkcijskih ovisnosti). O razlozima izbora baš toga skupa ključnih riječi bilo je govora u uvodu.

Stablo koje predstavlja skup W izgleda ovako:



Skupovi iz definicije automata su sljedeći:

$\Sigma = \{-, >, |\}$

$Q = \{0, 1, 2, 3, 4, 5, 6\}$

$\Phi = \{3, 5, 6\}$

$W = \{-->, -->->, |\}$

Funkcije *g*, f i *h* zadane su sljedećom tablicom. Zapis unutarnjih stanja reduciran je do njihovih indeksa.

	g			f	h
	–	>	|		
0	1	0	6	0	0
1	2	neuspjeh	neuspjeh	0	0
2	neuspjeh	3	neuspjeh	1	1
3	4	neuspjeh	neuspjeh	0	0
4	neuspjeh	5	neuspjeh	1	1
5	neuspjeh	neuspjeh	neuspjeh	0	0
6	neuspjeh	neuspjeh	neuspjeh	0	0

Sada izlažem postupak računanja vrijednosti funkcije f prema opisanom algoritmu.

1. $f(q_0) = q_0$ (točka 1. definicije funkcije f)
2. Jedina dva stanja dubine 1 su q_1 i q_6 pa je $f(q_1)=q_0$ i $f(q_6)=q_0$
3. Jedino stanje dubine 2 je q_2 i $q_2=g(1,-)$. Prema točki 2) definicije funkcije f bit će $f(q_2)=g(r,-)$, gdje se stanje r računa prema ondje opisanom programu. Za ovaj slučaj to znači da je $r=f(q_1)=q_0$ početna vrijednost za r.
   Kako je $g(q_0,-)=1 \neq$ neuspjeh, to je $r=q_0$ i konačna vrijednost za r. Zbog toga je konačno $f(q_2)=g(0,-)=1$.
4. Ovdje je $f(q_2)=1$, $q_3=g(2,>)$, $f(q_3)=g(r,>)$, početna vrijednost za r je $f(q_2)=1$. Budući da je $g(1,>)$=neuspjeh, postaje $f(1)=0$ nova i ujedno konačna vrijednost za r pa je $f(q_3)=g(0,>) = 0$.
5. Na sličan se način dobivaju vrijednosti $f(q_4)=0$ i $f(q_5)=0$.

Lako je provjeriti da je funkcija *h* u ovom slučaju istovjetna funkciji **f**.

Implementacija Aho-Korasickova automata u PDC Prologu

U nastavku izlaganja dajem program u PDC Prologu kojim se implementira Aho-Korasickov automat konstruiran u prethodnom odjeljku, za raspoznavanje ključnih riječi jezika računa višeznačnih i funkcijskih ovisnosti. Program ujedinjuje implementaciju automata s neproceduralnom interpretacijom općega proceduralnog Aho-Korasickova algoritma iz drugog odjeljka. U izvorni programski kod umetnuti su komentari kojima se tumače značenja i definicije pojedinih predikata.

```
/*****************************************
Program D: \TOOLBOX \AHO-ALG1.PRO.
Program implementira Aho-Korasickov algoritam
(konacni automat) za raspoznavanje kljucnih
rijeci iz zadanog skupa rijeci.
*****************************************/
domains
    stanje = stanje(integer); neuspjeh
database
    dosegnuto(stanje)

/* Predikat dosegnuto je predikat interne baze
podataka. Klauzula na vrhu baze podataka pred-
stavlja trenutno stanje automata, dosegnuto ne-
uspijevanjem izravne funkcije prijelaza stanja. */

predicates
    akcija(stanje)
    g(stanje,char,stanje)
    h(stanje,stanje)
    kljucnaRijec(stanje,string)
    pocetno(stanje)
    procesiraj(stanje,string)
    procesirajString
    procesirajAutomatom(stanje,char)
    zavrsno(stanje)
clauses
    g(stanje(0),'-',stanje(1)).
    g(stanje(0),'>',stanje(0)).
    g(stanje(0),'|',stanje(6)).
    g(stanje(1),'-',stanje(2)).
    g(stanje(1),'>',neuspjeh).
    g(stanje(1),'|',neuspjeh).
    g(stanje(2),'-',neuspjeh).
    g(stanje(2),'>',stanje(3)).
    g(stanje(2),'|',neuspjeh).
    g(stanje(3),'-',stanje(4)).
    g(stanje(3),'>',neuspjeh).
    g(stanje(3),'|',neuspjeh).
```

```
g(stanje(4),'-',neuspjeh).
g(stanje(4),'>',stanje(5)).
g(stanje(4),'|',neuspjeh).
g(stanje(5),'-',neuspjeh).
g(stanje(5),'>',neuspjeh).
g(stanje(5),'|',neuspjeh).
g(stanje(5),'-',neuspjeh).
g(stanje(5),'>',neuspjeh).
g(stanje(5),'|',neuspjeh).
g(stanje(6),'-',neuspjeh).
g(stanje(6),'>',neuspjeh).
g(stanje(6),'|',neuspjeh).
```

/* Klauzule kojima je prikazana izravna funkcija prijelaza stanja */

```
h(stanje(0),stanje(0)).
h(stanje(1),stanje(0)).
h(stanje(2),stanje(1)).
h(stanje(3),stanje(0)).
h(stanje(4),stanje(1)).
h(stanje(5),stanje(0)).
h(stanje(6),stanje(0)).
```

/* Klauzule kojima je prikazana povratna funkcija prijelaza stanja) */

```
zavrsno(stanje(3)).
zavrsno(stanje(5)).
zavrsno(stanje(6)).
```

/* Popis završnih stanja */

```
pocetno(stanje(0)).  /* Stanje "0" je
početno stanje */

procesirajString:-
    write("Upisi string! "),
    readln(String),
    pocetno(stanje(Stanje)),
    asserta(dosegnuto(stanje(Stanje))),
    procesiraj(stanje(Stanje),String).
```

/* String koji se procesira učitava se pomoću tastature predikatom **readln**. Zatim se utvrđuje početno stanje i ono se posprema na vrh unutarnje baze podataka kao trenutno dosegnuto stanje. Procesiranje preuzima predikat **procesiraj** */

```
procesiraj(-,"").
```

/* Ako je dio stringa koji se procesira prazan string, onda je procesiranje završeno */

```
procesiraj(stanje(Stanje),String):-
    retract(dosegnuto(stanje(Stanje))),
    asserta(dosegnuto(stanje(Stanje))),
    frontchar(String,PrednjiZnak,Ostatak),
    procesirajAutomatom(stanje(Stanje),
    PrednjiZnak),
    procesiraj(stanje(NovoStanje),Ostatak).
```

/* Trenutno dosegnuto stanje dobavlja se s vrha unutarnje baze podataka predikatom **retract**. Ta je operacija destruktivna pa se baza odmah obnavlja predikatom **asserta**. Nakon toga od dijela stringa koji je ostao za procesiranje otkida se prednji znak koji se procesira automatom, nakon čega se procesira ostatak stringa */

```
procesirajAutomatom(stanje(Stanje),
PrednjiZnak):-
    g(stanje(Stanje),PrednjiZnak,neuspjeh),
    h(stanje(Stanje),stanje(NovoStanje)),
    asserta(dosegnuto(stanje(NovoStanje))),
    procesirajAutomatom(stanje
    (NovoStanje),PrednjiZnak).
```

/* Prva klauzula predikata **procesirajAutomatom** ostvaruje niz prijelaza stanja povratnom funkcijom prijelaza. Naime ova klauzula predikata uspijeva (rekurzivno) sve dok uspijeva prva klauzula iz tijela njene definicije, tj. sve dok je vrijednost izravne funkcije prijelaza stanja jednaka vrijednosti "neuspjeh". Svaki put kad se to ustanovi ostvaruje se prijelaz povratnom funkcijom prijelaza i ažurira se vrh baze dosegnutih stanja. */

```
procesirajAutomatom(stanje(Stanje),Pred-
njiZnak):-
    retract(dosegnuto(stanje(Stanje))),
    g(stanje(Stanje),PrednjiZnak,stanje
    (NovoStanje)),
    asserta(dosegnuto(stanje(NovoSta-
    nje))),
    akcija(stanje(NovoStanje)).
```

/* Nakon što prva klauzula predikata **procesiraj-Automatom** ne uspije, poziva se ova klauzula. Očitava se trenutno dosegnuto stanje iz baze dosegnutih stanja (koje se za vrijeme uspijevanja prve klauzule predikata moglo promijeniti i nekoliko puta) i ostvaruje se (za trenutno procesirani PrednjiZnak) jedinstveni prijelaz izravnom funkcijom

prijelaza stanja. Daljnje akcije poduzimaju se u ovisnosti o dosegnutom stanju.

```
akcija(stanje(NovoStanje)):-
    zavrsno(stanje(NovoStanje)),
    kljucnaRijec(stanje(NovoStanje),Rijec),
    write("Prepoznata je kljucna rijec ",Rijec,
        " u zavrsnom stanju ",NovoStanje).

/* Ako je to stanje završno stanje, daje se poruka
o prepoznatoj ključnoj riječi. */

    akcija(-).

/* U protivnom predikat akcija bezuvjetno uspijeva
i procesiranje preuzima predikat procesiraj */

    kljucnaRijec(stanje(3),"-->").
    kljucnaRijec(stanje(5),"-->->").
    kljucnaRijec(stanje(6),"|").

/* Klauzule kojima su definirane ključne riječi */
goal

    procesirajString.
```

Primjer primjene programa

U ovom odjeljku dajem odlomke datoteke AHO–ALG1.LOG s rezultatima programa u primjeni na ulaznu riječ i "--->->-". Automat će raspoznati ključne riječi "-->" i "-->->" kao podriječi u zadanoj ulaznoj riječi. Datoteka je nastala praćenjem izvršavanja programa prema naredbi **shorttrace** prevodiocu Prologa. Ona je na mjestima skraćena i njen je sadržaj je komentiran. Komentari upozoravaju na pojedine faze raspoznavanja ključnih riječi u zadanoj ulaznoj riječi.

```
Compiling D:\TOOLBOX\AHO–ALG1.PRO to
memory
708 WARNING: The variable is not bound in this
clause. (F10=ok, Esc=abort)
akcija g h kljucnarijec pocetno procesiraj
procesirajstring procesirajautomatom
zavrsno
Execute programCALL:–PROLOG–Goal()
CALL:procesirajstring()
write("Upisi string! ")
Upisi string! CALL:readln(–)
--->->-
RETURN:readln("--->->-") /* Obavljen unos stringa */
CALL:pocetno(stanje(–))
RETURN:pocetno(stanje(0))
assert(dosegnuto(stanje(0)))
CALL:procesiraj(stanje(0),"--->->-") /* Početak procesiranja */
REDO:procesiraj(stanje(0),"--->->-")
retract(dosegnuto(stanje(0)))
assert(dosegnuto(stanje(0)))
CALL:frontchar("--->->-",–,–)
RETURN:frontchar("--->->-",'-',"-->->-") /* Otkinut PrednjZnak */
CALL:procesirajautomatom(stanje(0),'-') /* Počinje njegovo procesiranje automatom*/
CALL:g(stanje(0),'-',neuspjeh)
FAIL:g(stanje(0),'-',neuspjeh) /* Vrijednost funkcije g različita od "neuspjeh" */
REDO:procesirajautomatom(stanje(0),'-')

/* Poziva se druga klauzula predikata procesiraj-
Automatom */

retract(dosegnuto(stanje(0)))
CALL:g(stanje(0),'-',stanje(–))
RETURN:*g(stanje(0),'-',stanje(1))
assert(dosegnuto(stanje(1)))
CALL:akcija(stanje(1))
CALL:zavrsno(stanje(1))
FAIL:zavrsno(stanje(1)) /* Dosegnuto stanje nije završno stanje */
REDO:akcija(stanje(1))
RETURN:akcija(stanje(1)) /* Predikat akcija bezuvjetno uspijeva */
CALL:procesiraj(stanje(–),"-->->-")

/* Počinje procesiranje ostatka ulazne riječi . Odlomak datoteke u kojem su rezultati procesiranja idućih dvaju znakova "-" izostavljam */

CALL:procesiraj(stanje(–),">->-")
REDO:procesiraj(stanje(–),">->-")
retract(dosegnuto(stanje(2)))
assert(dosegnuto(stanje(2)))
CALL:frontchar(">->-",–,–)
RETURN:frontchar(">->-",'>',"->-")
CALL:procesirajautomatom(stanje(2),'>')
CALL:g(stanje(2),'>',neuspjeh)
```

```
FAIL:g(stanje(2),'>',neuspjeh)
REDO:procesirajautomatom(stanje(2),'>')
retract(dosegnuto(stanje(2)))
CALL:g(stanje(2),'>',stanje(–))
REDO:g(stanje(2),'>',stanje(–))
RETURN:*g(stanje(2),'>',stanje(3))
/* Rezultat primjene izravne funkcije prijelaza na znaku ">" */

assert(dosegnuto(stanje(3)))
CALL:akcija(stanje(3))
CALL:zavrsno(stanje(3))
RETURN:*zavrsno(stanje(3)) /* Dosegnuto je stanje završno stanje */
CALL:kljucnarijec(stanje(3),–)
RETURN:*kljucnarijec(stanje(3),"-->")
write("Prepoznata je kljucna rijec ")
Prepoznata je kljucna rijec write("-->")
-->write(" u zavrsnom stanju ")
u zavrsnom stanju write(3) /* Ispis poruke o prepoznatoj ključnoj riječi */
3RETURN:*akcija(stanje(3))
CALL:procesiraj(stanje(–),"->-") /* Procesiranje se nastavlja */

/* Ispušten odlomak datoteke */

assert(dosegnuto(stanje(5)))
CALL:akcija(stanje(5))
REDO:zavrsno(stanje(5))
RETURN:*zavrsno(stanje(5)) /* Stanje 5 prepoznato kao završno */
REDO:kljucnarijec(stanje(5),–)
RETURN:*kljucnarijec(stanje(5),"-->->")
write("Prepoznata je kljucna rijec ")
Prepoznata je kljucna rijec write("-->->")
-->->write(" u zavrsnom stanju ")
 u zavrsnom stanju write(5) /* Ispis poruke o prepoznatoj drugoj ključnoj riječi */
RETURN:*akcija(stanje(5)) /* Procesiranje se nastavlja */
CALL:procesiraj(stanje(–),"-")
REDO:procesiraj(stanje(–),"-")
retract(dosegnuto(stanje(5)))
assert(dosegnuto(stanje(5)))
CALL:frontchar("-",–,–)
RETURN:frontchar("-",'-',"")
CALL:procesirajautomatom(stanje(5),'-')
CALL:g(stanje(5),'-',neuspjeh)
RETURN:*g(stanje(5),'-',neuspjeh)

/* Prva klauzula predikata procesirajAutomatom uspijeva */

CALL:h(stanje(5),stanje(–)) /* Poziva se povratna funkcija prijelaza stanja */
RETURN:*h(stanje(5),stanje(0)) /* Iz stanja 5 vrši se povrat na stanje 0 */
assert(dosegnuto(stanje(0)))
CALL:procesirajautomatom(stanje(0),'-') /* Procesira se posljednji znak */
retract(dosegnuto(stanje(0)))
CALL:g(stanje(0),'-',stanje(–))
RETURN:*g(stanje(0),'-',stanje(1)) /* Obavljen prijelaz izravnom funkcijom prijelaza */
assert(dosegnuto(stanje(1)))
CALL:akcija(stanje(1))
CALL:zavrsno(stanje(1))
FAIL:zavrsno(stanje(1)) /* Dosegnuto stanje nije završno stanje */
REDO:akcija(stanje(1))
RETURN:akcija(stanje(1)) /* Predikat akcija bezuvjetno uspijeva */
CALL:procesiraj(stanje(–),"")

/* Dio stringa koji je ostao za procesiranje prazan je */

RETURN:*procesiraj(stanje(–),"")

/* Predikat procesiraj bezuvjetno uspijeva */

RETURN:–PROLOG–Goal() /* Program uspješno okončava rad */
```

Daljnje mogućnosti razvoja programa

Na kraju upozoravam na neka moguća poboljšanja i proširenja predložene implementacije Aho-Korasickova automata i njegova algoritma.

1. Iz priloženoga se primjera vidi da neke ključne riječi mogu biti podriječi drugih ključnih riječi. Tako je riječ "-->" podriječ riječi "-->->". Kako se program za leksičku analizu mora odlu-čiti za jednu od njih, potrebno je na skupu takvih riječi zadati linearni uređaj koji će omogućiti ko-načnu odluku.

2. Baze znanja funkcija *g* i *h* mogu za opsežnije skupove međusobno različitih ključnih riječi biti toliko velike da vremenski znatno opterećuju pretraživanje pri ujednačavanju (unificiranju) klauzula za vrijeme rješavanja problema.

Ako se, na primjer, pri ujednačavanju klauzula traži vrijednost *g*(stanje(5),'−', neuspjeh) funkcije g, onda će se u tijeku pretraživanja prije pronalaženja te vrijednosti provjeriti sve vrijednosti funkcije *g* za sva stanja od 1 do 4.

Pretraživanje se može skratiti na više načina.

Funkciji *h* svojstvena je neinjektivnost. To znači da će ona za dvije ili više različitih vrijednosti argu-menta poprimiti istu vrijednost. Konkretno, u izloženom primjeru vidi se da je

h(stanje(2),stanje(1)) i

h(stanje(4),stanje(1)),

dok je u svima preostalim slučajevima h(stanje(Stanje),stanje(0)). Zbog toga se funkcija *h* kao predikat može skraćeno definirati ovako:

h(stanje(2),stanje(1)).

h(stanje(4),stanje(1)).

h(−,stanje(0)).

Za funkciju *g* može se za svaku vrijednost stanja--argumenta definirati po jedna klauzula nove funkcije (predikata) na ovaj način:

*g*1(stanje(0)):- *g*(stanje(0),−,−).

.

.

.

*g*1(stanje(6)):- *g*(stanje(6),−,−).

Da bi se pretraživanje reduciralo, potrebno je u tijelima klauzula kojima se definira predikat **procesirajAutomatom** svaki nastup predikata *g* zamijeniti predikatom g_1. Tada će se u svakom pokušaju da se zadovolji taj predikat pokušati zadovoljiti predikat g_1 za zadanu (vezanu) vrijednost prve varijable. Definicija funkcije g_1 ograničit će pretraživanje baze znanja za funkciju *g* samo na tu vrijednost argumenta.

Literatura:

[1] A. V. Aho: Algorithms for Finding Patterns in Strings, u J. van Leeuven(ed.): Handbook of Theoretical Computer Science, vol. A, Algorithms and Complexity, Elsevier, 1990, 1992, str. 255 - 300.

[2] R. S. Boyer, J. S. Moore: A fast string searching algorithm, Comm. ACM 20(10), 1972, str. 62--72.

[3] B. Commentz-Walter: A string matching algorithm fast on the average, u H.A. Maurer(ed): Proc 6th Internat. Coll. on Automata, Languages and Programming, Springer, Berlin, 1979, str. 118-132.

[4] M. Čubrilo: Matematička logika za ekspertne sisteme, IBI - 15, Informator, Zagreb, 1989.

[5] PDC Prolog User's Guide, Prolog Development Center, Copenhagen, 1990.

[6] R. M. Karp, M. O. Rabin: Efficient randomized pattern-matching algorithms, IBM J. Res. Develop. 31(2), 1978, str. 249-260.

[7] D. E. Knuth, J. H. Morris, V. R. Prat: Fast pattern matching in strings, SIAM J. Comput. 6(2), 1977, str. 323-350.

[8] M. Radovan: Programiranje u PROLOGu, Informator, Zagreb, 1987.

[9] S. Tkalac: Relacijski model podataka, IBI-13, Informator, Zagreb, 1988.

Address for correspondence:
Mirko Čubrilo
Fakultet organizacije i informatike
Pavlinska 2
42000 Varaždin
tel.: +38 42 51-199
fax: +38 42 55-594

Primljeno: 30. listopada 1992.
Prihvaćeno: 8. prosinca 1992.

Rođen sam 30. 9. 1953. g. u Josipovcu kod Osijeka. Osnovnu školu i gimnaziju završio sam u Gospiću. Diplomirao sam 1979. g. na smjeru Teorijska matematika na Matematičkom odjelu Prirodoslovno-matematičkog fakulteta u Zagre-bu i stekao zvanje diplomiranog inženjera matematike. Iste sam godine upisao postdiplomski studij matematike koji sam završio 1984. g. i srekao akademsko zvanje magistra prirodnih znanosti iz područja matematike. Doktorat iz područja računarskih znanosti postigao sam 1992. godine na Elektrotehničkom fakultetu u Zagrebu.

Imam četrnaest godina radnog iskustva. Osam godina radio sam na poslovima nastavnika matematike na srednjoj školi, a posljednjih šest godina radim kao asistent na Fakultetu organizacije i informatike u Varaždinu.

Na Fakultetu sam, u okviru nastavne djelatnosti, za posljednje tri školske godine samostalno držao kompletnu nastavu iz kolegija *Struktura i organizacija podataka*, *Oblikovanje baze podataka* i *Teorija klasifikacije*.

Objavio sam više stručnih i znanstvenih radova, među kojima i monografiju Matematička logika za ekspertne sisteme (1989) u informatičkoj biblioteci (IBI) izdavača "Informator" iz Zagreba.

Glavni sam istraživač na projektu pod nazivom *Specifikacija i implementacija nekih programskih sučelja* (projekt br. 1-02-130) Ministarstva znanosti, informatike i tehnologije.