MOGUĆNOSTI PROCEDURALNOG PROGRAMIRANJA U PROGRAMSKOM JEZIKU PROLOG

INFORMATICA 3/89

Keywords: procedural programming, Prolog, matrix multiplication

Iztok Z. Race,
Mašinski fakultet, Beograd

SAŽETAK

Programski jezik PROLOG kao deklarativni programski jezik se bitno razlikuje od proceduralnih programskih jezika, jer prvenstveno opisuje posmatrani problem, a ne način kako se rešava neki problem. U skladu sa tako koncipiranim programskim jezikom, bitno se razlikuju programi pisani u PROLOG-jeziku od programa pisanih u proceduralnim jezicima. Zbog toga se mora govoriti i o drugačijem načinu razmišljanja programera prilikom izrade programa u programskom jeziku PROLOG. U radu je dato moguće rešenje za neke uobičajene programske strukture i strukture podataka iz proceduralnih jezika, te ilustracija mogućnosti proceduralnog programiranja u PROLOG-jeziku na primeru množenja matrica.

1. PROLOG KAO DEKLARATIVNI PROGRAMSKI JEZIK

Poslednjih godina, naglim razvojem veštačke inteligencije i njenom primenom u raznorodnim ljudskim aktivnostima, za programski jezik PROLOG kao oruđe u ovoj oblasti, izuzetno raste interesovanje.

PROLOG-jezik je nastao iz ideje da se predikatska logika posmatra kao programski jezik. Zbog toga, program u ovom jeziku opisuje relacije između pojedinih objekata, i pravila na osnovu kojih se iz postojećih relacija mogu dokazati nove. To je bitna razlika koja odvaja ovaj jezik od proceduralnih jezika kao što su npr. FORTRAN, BASIC ili PASCAL. U proceduralnim jezicima se program sastoji iz niza koraka u kojima se opisuje kako da računar izvrši određena izračunavanja. U PROLOG-jeziku je program opis problema koji se posmatra, dok se sam način na koji se problem rešava direktno ne opisuje. Drugim rečima, proceduralni programski jezici opisuju **kako** se rešava pojedini problem, dok deklarativni programski jezici opisuju šta je posmatrani problem.

PROLOG-jezik se od proceduralnih jezika razlikuje u nizu osobina. Promenljive u PROLOG-jeziku dobijaju vrednost samo u toku pokušaja zadovoljenja određene rečenice PROLOG-jezika. Njihova vrednost je trenutna i vezana je za trajanje razmatranja date rečenice. Na taj način promenljiva ne predstavlja striktno određenu memorijsku lokaciju. Sa druge strane, proceduralni jezici imaju velik broj sličnih programskih struktura ili struktura podataka koje ne postoje u PROLOG-jeziku poput if-then-else strukture, petlji ili nizova. Nasuprot tome PROLOG-jezik ima svoje osnovne mehanizme koji ga izdvajaju od ostalih jezika kao što su automatski backtracking i cut-predikat. Ovi mehanizmi omogućavaju da se pojedini problemi u programskom jeziku PROLOG reše na drugačiji i bolji način, što sa svoje strane može predstavljati teškoću prilikom pisanja programa u ovom jeziku, pogotovu ako se prilikom programiranja koriste načini razmišljanja iz drugih programskih jezika.

Ponekad je, u kompleksnijim problemima koji se rešavaju u PROLOG-jeziku, potrebno izvršiti određena izračunavanja, za koja su proceduralni jezici daleko efikasniji. Od načina na koji se problem postavi u PROLOG-jeziku vrlo često zavisi i brzina, a ponekad i ukupan uspeh datog rešenja. U radu se ukazuje na neke od mogućih pristupa tom problemu.

2. PREDSTAVLJANJE NEKIH STRUKTURA PODATAKA U PROGRAMSKOM JEZIKU PROLOG

Promenljive. U mnogim programskim jezicima promenljive predstavljaju određene memorijske lokacije, čiji sadržaji ostaju nepromenjeni sve dok se ne izračuna neka druga vrednost i memoriše kao novi sadržaj posmatrane memorijske promenljive. Za razliku od tog pristupa, promenljive u programskom jeziku PROLOG imaju naglašeno lokalan karakter, tj. one su aktuelne samo u okviru rečenice u kojoj se spominju. Takođe je karakteristično da svaki pokušaj izmene vrednosti već postavljene memorijske promenljive, u ovom programskom jeziku završava neuspehom.

Zbog ekstremne lokalnosti promenljivih u PROLOG-jeziku, prenos parametara se vrši na jedan od sledeća dva načina:

- funkcijskim pozivom

- upisivanjem vrednosti promenljive kao činjenice.

Primer funkcijskog poziva u PROLOG-jeziku bi bio:

```
a1(P1,P2,...).
a1(P1,P2,...):-...,a2(P1,P2,...),...
```

čime se vrednosti promenljivih P1,P2,... u predikatu a1, postavljaju na vrednosti promenljivih u predikatu a2, pa se na taj način i obavlja prenos parametara.

Upisivanje vrednosti promenljive kao činjenice u bazu podataka PROLOG-jezika se vrši pomoću sistemskog predikata asserta ili assertz. Tako bi npr. rečenica:

```
assertz(vrednost(a2,1000)).
```

u bazu podataka (na njen kraj) upisala PROLOG-činjenicu vrednost(a2,1000). Na taj način se, formiranjem ovakve strukture u bazi podataka PROLOG-jezika, može predstaviti promenljiva a2 kojoj je pridružena vrednost 1000, odnosno relacija **vrednost** koja se uspostavlja između atoma (imena promenljive) a2 i atoma (sadržaja promenljive) 1000. Promena ove vrednosti, na recimo 500, se vrši, pomoću sistemskih predikata retract i assertz, odnosno tako što se prvo obriše stara vrednost, a zatim upiše nova, odnosno, u PROLOG-jeziku:

```
retract(vrednost(a2,_)),
assertz(vrednost(a2,500)).
```

To, dinamički gledano odgovara činjenici da atomi a2 i 1000 više nisu u relaciji vrednost, i da je ta relacija uspostavljena između atoma a2 i 500. Tako se pomoću relacija može predstaviti izmena sadržaja memorijske promenljive u proceduralnim jezicima. Pri tome se mora naglasiti, da za razliku od situacije kod proceduralnih jezika, atom a2 (označava ime promenljive) može biti u relaciji sa više atoma koji označavaju vrednosti istovremeno, što može korisno poslužiti pri rešavanju nekih problema.

Nizovi. U PROLOG - jeziku nema nizova za razliku od mnogih drugih jezika. Zato se, ukoliko je potrebno određene podatke strukturirati u niz, oni unose u bazu podataka PROLOG-jezika slično promenljivama:

```
assertz(vrednost(a(1,2),550)).
```

Strukturiranje se može izvršiti i na druge slične načine.

Pokazivači. Pokazivači se u PROLOG - jeziku realizuju na taj način što se doda još jedno polje u postojeću strukturu promenljive koje ukazuje na neku drugu promenljivu. Tako strukturisana promenljiva ima oblik:

```
a(indexm,...,indexp)
```

gde promenljiva u nizu a sa indeksom indexm ima pokazivač koji pokazuje na promenljivu sa indeksom indexp.

Stekovi. Stekovi u programskom jeziku PROLOG, uglavnom se predstavljaju ili kao liste, ili kao niz činjenica.

Stek se predstavlja pomoću liste na taj način što se oformi struktura

```
stek([vrh,...,dno])
```

gde su elementi liste, redom elementi u steku, od elementa na vrhu steka (označenog sa vrh) do elementa na dnu steka (označenog sa dno). Dodavanje novog elementa na stek se vrši zadovoljenjem predikata push, koji može imati sledeći oblik:

```
push(Element,Stek,[Element|Stek]).
```

dok se izbacivanje elementa sa vrha steka može realizovati na sledeći način u PROLOG-jeziku:

```
pop(_,[],[]):-write('Stack underflow').
pop(Element,[Element|Stek],Stek).
```

Prva rečenica kojom je opisan predikat pop odgovara stanju kada se pokušava da izbaci element iz praznog steka, dok druga rečenica odgovara izbacivanju elementa u regularnom slučaju.

Stek se može predstaviti i kao niz činjenica u PROLOG-jeziku, oblika:

```
stek(vrh).
.........
stek(elementm).
.........
stek(dno).
```

Pri tome se ubacivanje elemenata u stek vrši zadovoljavanjem predikata push koji ima sledeći oblik:

```
push(Element,Stek):-
     Struktura=..[Stek,Element],
     asserta(Struktura).
```

Zadovoljavanjem predikata pop se vrši izbacivanje elemenata iz steka:

```
pop(Element,Stek):-
     Struktura=..[Stek,Element],
     pop1(Struktura).

pop1(Struktura):-
     retract(Struktura),!.
pop1(Struktura):-
     write('Stack underflow').
```

Zadovoljavanje predikata pop se vrši na taj način što se prvo zadovolji univ-predikat i tako formira struktura čiji je funktor sadržaj promenljive Stek (koja za vrednost ima ime steka) a jedini argument nepostavljena promenljiva Element. Zatim se zadovoljava predikat pop1. Ukoliko postoji u bazi podataka PROLOG-jezika struktura sa gore određenim funktorom (prva takva na vrhu) se izbacuje iz baze (i iz steka). Time se, ujedno i postavlja promenljiva Element u strukturi i predikatu pop. U slučaju da u bazi podataka takva struktura ne postoji, javlja se odgovarajuća poruka.

Niska (queue). Niske se mogu implementirati u PROLOG-jeziku kao liste, pa se

formira sledeća struktura:

```
niska([početak,...,kraj])
```

međutim takvo rešenje ima svojih loših strana, koje se uočavaju prilikom umetanja novih elemenata na kraj niske. Da bi postavili element iza elementa koji je na kraju liste, moramo zadovoljiti predikat insert koji može imati sledeći oblik:

```
insert(Element,[],[Element]).
insert(Element,[G|R],[G|R1]):-
     insert(Element,R,R1).
```

Izbacivanje elemenata iz liste u takvom rešenju je daleko jednostavnije obzirom da je element koji se izbacuje iz niske prvi element liste kojom je niska predstavljena. Tako bi predikat delete kojim obezbeđujemo tu funkciju mogao da ima oblik:

```
delete(_,[],[]):-
     write('Queue underflow'),!.
delete(Početak,[Početak|R],R).
```

Mnogo efikasnije rešenje se postiže upotrebom sistemskog predikata assertz, u kojem se niska predstavlja u obliku činjenica u PROLOG-jeziku tipa:

```
niska(početak).
.............
niska(elementm).
.............
niska(kraj).
```

Predikati insert i delete koji nam služe za manipulaciju sa niskom bi u ovom slučaju imali oblik:

```
insert(Element,Niska):-
     Struktura=..[Niska,Element],
     assertz(Struktura).

delete(Element,Niska):-
     Struktura=..[Niska,Element],
     delete1(Struktura).

delete1(Struktura):-
     retract(Struktura),!.
delete1(Struktura):-
     write('Queue underflow').
```

Drveta i grafovi se mogu predstaviti na razne načine koji variraju od rešenja da se celo drvo (ili graf) predstavi kao lista do rešenja da se prvo čvorovi drveta (grafa) predstave u obliku činjenica u PROLOG-jeziku, a potom, takođe i veze koje postoje među tim čvorovima. Tako se graf može predstaviti sledećom strukturom u PROLOG-jeziku:

```
graf([[čvor1,[čvor11,...,čvor1k]],...,
                [[čvorn,[čvorn1,...,čvornm]]])
```

gde su čvor1,...,čvorn imena čvorova u grafu uz koje je pridružena lista imena čvorova sa kojima je svaki od tih čvorova u vezi. Graf se može predstaviti i na sledeći način:

```
graf(čvor1,[čvor11,...,čvor1k]).
.............................
graf(čvorn,[čvorn1,...,čvornm]).
```

ili:

```
graf(čvor1).
...........
graf(čvorn).
graf(čvor1,čvor11).
...................
graf(čvorn,čvornm).
```

gde se činjenice sa funktorom graf i jednim argumentom odnose na čvorove grafa, a one činjenice sa istim funktorom i dva argumenta na veze koje postoje u grafu. Ovo su samo neke od varijanata predstavljanja grafa (drveta) u PROLOG-jeziku. Svaka od varijanata mora da bude prvenstveno prilagođena problemu. U zavisnosti od izabrane varijante su i predikati koji omogućuju manipulaciju tako predstavljenim grafovima ili drvetima.

3. PREDSTAVLJANJE NEKIH UOBIČAJENIH PROGRAMSKIH STRUKTURA U PROLOG JEZIKU

if-then-else (if-then) struktura se javlja u formi:

```
if uslov then naredba1 else naredba2
```

ili

```
if uslov then naredba.
```

Prvi oblik se u PROLOG-jeziku implementira kao:

```
if_then_else(Uslov,Naredba1,Naredba2):-
     Uslov,!,
     Naredba1.
if_then_else(Uslov,Naredba1,Naredba2):-
     Naredba2.
```

ili:

```
if_then_else(Uslov,Naredba1,Naredba2):-
     ((Uslov,!,Naredba1);Naredba2).
```

Ovde su Uslov, Naredba1 i Naredba2 promenljive koje se postavljaju na imena predikata koji opisuju traženi uslov i odgovarajuće naredbe.

U drugom obliku (if-then) se javljaju problemi u slučaju neispunjenja uslova. Tada se mora obratiti pažnja na to da predikat bude zadovoljen i u slučaju da postavljeni uslov nije ispunjen, pošto bi pad predikata mogao da prouzroči i neizvršenje programa. Tako bi odgovarajući predikat imao oblik:

```
if_then(Uslov,Naredba):-
     Uslov,!,
     Naredba.
if_then.
```

ili:

```
if_then(Uslov,Naredba):-
     ((Uslov,!,Naredba);true).
```

Očito je da se ovakve strukture mogu dalje usložnjavati i da se na sličan način može predstaviti i naredba višestrukog grananja (case naredba u PASCAL-jeziku).

while-do petlja ima oblik:

```
while uslov do naredba.
```

Ukoliko petlja nije beskonačna, uslov je funkcija argumenata koji menjaju svoje vrednosti unutar ciklusa tokom izvršenja naredbe. Jedan od mogućih pristupa je da se ova programska struktura u PROLOG-jeziku predstavi rekurzivnim rešenjem:

```
while_do(Uslov,Naredba):-
     not(Uslov),!.
while_do(Uslov,Naredba):-
     Naredba,
     while_do(Uslov,Naredba).
```

Ovo rešenje ima svoje nedostatke u rekurzivnom pristupu rešenju programskog ciklusa u PROLOG-jeziku i memorisanju parametara u rekurzivnom pozivu koje je potrebno zbog obezbeđenja mehanizma backtracking-a.

Rešenje može da ima i sledeći oblik u kome se koristi sistemski predikat repeat:

```
while_do(Uslov,Naredba):-
     Uslov,
     repeat_while(Naredba,Uslov).
while_do(_,_).

repeat_while(Naredba,Uslov):-
     repeat,
     Naredba,
     not(Uslov).
```

U ovom rešenju je izbegnuto nepotrebno memorisanje parametara u rekurzivnom pozivu koje je glavni nedostatak prethodno navedenog rešenja.

Komponovane naredbe visokog nivoa u velikim sistemima se javljaju kao poseban problem prilikom izgradnje aplikacija u PROLOG-jeziku. Naime, vrlo često se dešava da je potrebno po određenom redosledu zadovoljiti neke predikate, bez potrebe memorisanja puta kojim se prošlo (radi eventualnog backtracking-a). U ovakvoj situaciji predikati u PROLOG-jeziku se ne posmatraju u svom deklarativnom, već prvenstveno u proceduralnom značenju. Tako se predikati posmatraju kao procedure koje obavljaju određenu akciju. Memorisanje puta kojim se prošlo se prekida sistemskim predikatom fail. Parametri koji se prenose između tako posmatranih procedura se mogu upisivati u bazu podataka PROLOG-jezika pomoću sistemskih predikata asserta i assertz, kao i čitati i brisati pomoću predikata retract. Primera radi komponovana naredba:

```
begin naredba1;naredba2;...;naredbak end
```

bi bila realizovana zadovoljenjem predikata komp_naredba:

```
komp_naredba:-
     naredba1,
     fail.
komp_naredba:-
     naredba2,
     fail.
.............
komp_naredba:-
     naredbak.
```

Na ovaj način je moguće, vodeći računa o tome kada je potreban pojedini predikat u svom proceduralnom, a kada u deklarativnom značenju, izgraditi dosta velike sisteme koji imaju i proceduralne delove.

4. PRIMER PROCEDURALNOG PROGRAMIRANJA U PROLOG-JEZIKU - MNOŽENJE MATRICA

Klasičan proceduralni problem je izrada programa za množenje matrica. Neka su matrice a i b date u obliku činjenica u PROLOG-jeziku, na način koji je naveden u tekstu za predstavljanje nizova:

```
vrednost(a(1,1),Va11).
.....................
vrednost(a(1,M),Va1M).
vrednost(a(2,1),Va21).
.....................
.....................
vrednost(a(N,M),VaNM).
vrednost(b(1,1),Vb11).
.....................
vrednost(b(1,K),Vb1K).
vrednost(b(2,1),Vb21).
.....................
.....................
vrednost(b(M,K),VbMK).
```

i neka su promenljive N, M, K i Va11,...,VaNM,Vb11,...,VbMK postavljene na konkretne numeričke vrednosti. Program za množenje tako zadatih matrica i formiranje odgovarajućih činjenica o matrici-rezultatu u bazi podataka ima sledeći oblik:

```
mnozenje_matrica(A,B,C):-
     Struktura1=..[A,A1,A2],
     Struktura2=..[B,A2,B2],
     Struktura3=..[C,A1,B2],
     vrednost(Struktura1,V1),
     vrednost(Struktura2,V2),
     element_prod(Struktura3,V1,V2,V),
     assertz(vrednost(Struktura3,V)),
     fail.
mnozenje_matrica(_,_,_).
element_prod(Struktura3,V1,V2,V):-
     retract(vrednost(Struktura3,V3)),!,
     V is V3+V1*V2.
element_prod(Struktura3,V1,V2,V):-
     V is V1*V2.
```

Pozivom mnozenje_matrica(a,b,c) u PROLOG bazi podataka se formiraju strukture vrednost koje odgovaraju matrici c koja je rezultat množenja matrica a i b. Koristeći univ-predikat i ne postavljajući promenljive A1,A2 i B2 formiraju se odgovarajuće strukture koje odgovaraju onim elementima matrice koji se trenutno množe

(Struktura1 i Struktura2) i koji se trenutno izračunava (Struktura3). Kao elementi ovakvih struktura, nalaze se promenljive koje dobijaju vrednosti koje odgovaraju indeksima matrice. Pomoću tih promenljivih se uspostavljaju i relacije unutar baze podataka PROLOG-jezika i time obezbeđuje množenje odgovarajućih članova matrica i upisivanje rezultata na odgovarajuće mesto. Zadovoljavanjem predikata vrednost dobijaju se konkretne vrednosti za elemente matrica sa konkretnim indeksima. Zadovoljavanjem predikata element_prod se sračunava proizvod vrednosti dva odgovarajuća elementa matrica i dodaje na ukupan međuzbir pri izračunavanju elemenata u matrici-rezultatu. Tako izračunata vrednost (međuzbir ili konačni element matrice-proizvoda) se pomoću sistemskog predikata assertz upisuje u bazu podataka PROLOG-jezika. Nakon ovoga predikat mnozenje_matrica pada (sistemski predikat fail) i pokušava da se ponovo zadovolji postavljajući nove vrednosti promenljivih A1,A2 i B2. One su određene činjenicama vrednost u bazi podataka PROLOG-jezika. Nastavljajući tako, iscrpljujući skup svih proizvoda elemenata matrica definisanih činjenicom vrednost koji zadovoljavaju relacije određene u definisanim strukturama, dobija se konačan rezultat. Druga rečenica koja definiše predikat mnozenje_matrica služi isključivo da bi predikat na kraju bio zadovoljen.

Interesantno je primetiti da činjenice o vrednostima elemenata matrice u bazi podataka ne moraju uopšte biti poređane ni po vrstama ni po kolonama, i da matrice ne moraju biti kompletne (mogu nedostajati pojedini elementi) da bi bilo izračunato ono što se u tom slučaju može izračunati u množenju takvih matrica.

Važno je takođe primetiti da se korišćenjem sistemskog predikata fail može postići da se neprekidnim ponovnim zadovoljavanjem željenih predikata ponavlja određena procedura (posmatrajući proceduralno značenje tih predikata). Izmenom nekih elemenata (upisivanjem ili brisanjem u bazi podataka PROLOG-jezika pomoću asserta,assertz i retract) može se različito uticati na trajanje tih ponavljanja. Na taj način se mogu dati rešenja u programskim ciklusima, a time se i izbegava odgovarajuće rekurzivno rešenje koje je nepogodno zbog memorisanja pređenog puta.

5. UMESTO ZAKLJUČKA

Ovaj rad ima dve namene. Prva je da programerima koji su početnici u pisanju programa u PROLOG-jeziku omogući da svoje proceduralne programe lako prevedu u PROLOG-jezik. Ipak se mora naglasiti da tako prevedeni programi nisu najsretnije rešenje. Bez obzira na to dosta velik broj programera razmišlja proceduralno prilikom izrade svojih programa. Na taj način se dolazi i do druge namene rada. Naime, radom se želelo pokazati da se iskusan programer u drugim programskim jezicima, prilikom programiranja u PROLOG-jeziku mora osloboditi nekih ranije stečenih dogmi u gledanju na pojedine probleme. Lep primer za to je množenje matrica. Uobičajena navika iz drugih programskih jezika je da procedura bude tačno završena (da ne "padne" tokom izvršavanja). U većini programskih jezika se takva mogućnost "pada" ni ne razmatra. "Side"-efekti tokom izvršenja procedure se uglavnom posmatraju kao nepovoljna osobina, pa se ili izbegavaju, ili upotrebljavaju u ograničenim količinama. Otuda je često mišljenje koje ide linijom manjeg otpora da se "side"-efekat izbacuje iz razmišljanja tokom izrade procedura. O "padu" procedure tokom njenog izvršenja se u početku pisanja programa u PROLOG-jeziku uglavnom ne razmišlja. Rešenje koje je dato za problem množenja matrica ilustruje neprestano "padanje" predikata mnozenje_matrica (posmatranog kao procedura) kao i "side"-efekte dobijene tom prilikom (upis novoizračunatih međuzbirova u bazu podataka PROLOG-jezika pomoću sistemskih predikata assertz i retract).

Usvajanjem ovakvog načina razmišljanja prilikom pisanja programa u PROLOG-jeziku bitno se doprinosi poboljšanju stila pisanja tih programa, a i efikasnosti njihovog izvršenja.

6. LITERATURA

1. Bratko, Ivan, Inteligentni informacijski sistemi, Univerza Edvarda Kardelja v Ljubljani, Fakulteta za elektrotehniko, Ljubljana, 1984
2. Clocksin, W. F., Melish, C. S., Programming in Prolog, Springer Verlag, Berlin Heidelberg New York, 1981
3. Munakata, Toshinori, Procedurally Oriented Programming Techniques in Prolog, IEEE Expert, Summer 1986, str 41-47
4. Race, Iztok Z., Programska implementacija ekspertnog sistema za dijagnostiku u PROLOG-jeziku, magistarski rad, Univerzitet u Beogradu, Prirodno matematički fakultet, Matematički fakultet, Beograd, 1988
5. Sterling, Leon, Shapiro, Ehud, The Art of Prolog: Advanced Programming Techniques, The MIT Press, Cambridge, Massachusetts, London, England, 1986