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

$\vdots$

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 $\alpha$–promenljiva, $\beta$–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 $\alpha$ zameni izrazom $\beta$.

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

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

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 složenijh 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

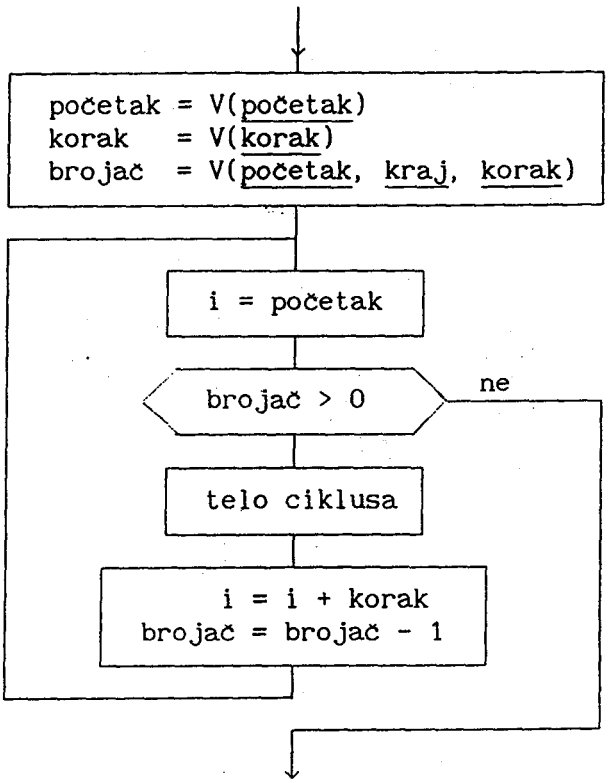
Vizuelizacija je moćno sredstvo u mnogim oblastima istraživanja, a naročito nastave. To je grafičko predstavljanje odnosa i procesa u cilju lakšeg stavaranja misaonih modela o objektu koji je predmet vizuelizacije. Kako grafički prikazi, po pravilu, nose konciznu i jasnu informaciju to su posebno pogodni 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.

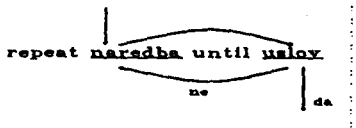
- Po izlasku iz ciklusa ciklusna promenljiva ima vrednost za koju je poslednji put izvršen ciklus uvećanu za korak. Ako se ciklus ne izvrši onda je to početna vrednost.

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

### Vizuelizacija nad sintaksnim definicijama

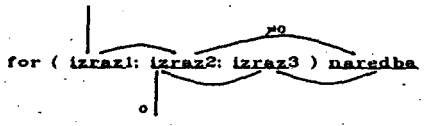
Korisnici programskih jezika moraju naučiti sintaksu jezika. Vezivanje semantike za sintaksne definicije ima višestruke koristi u učenju programskih jezika. To će omogućiti lakše pamćenje sintakse, jer se semantički modeli lakše i duže pamte od strane korisnika. S druge strane, to će omogućiti da sintaksa 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