

# Naslov seminarskog rada

Seminarski rad u okviru kursa  
Metodologija stručnog i naučnog rada  
Matematički fakultet

Isidora Đurđević, Ana Stanković, Milica Đurić  
kontakt email prvog, drugog (trećeg) autora

5. april 2017.

## Sažetak

U ovom tekstu je ukratko prikazana osnovna forma seminarskog rada. Obratite pažnju da je pored ove .pdf datoteke, u prilogu i odgovarajuća .tex datoteka, kao i .bib datoteka korišćena za generisanje literature. Na prvoj strani seminarskog rada su naslov, apstrakt i sadržaj, i to sve mora da stane na prvu stranu! Kako bi Vaš seminarski zadovoljio standarde i očekivanja, koristite uputstva i materijale sa predavanja na temu pisanja seminarskih radova. Ovo je samo šablon koji se odnosi na fizički izgled seminarskog rada (šablon koji *morate* da ispoštujete!) kao i par tehničkih pomoćnih uputstava. Molim Vas da kada budete predavali seminarski rad, imenujete datoteke tako da sadrže temu seminarskog rada, kao i imena i prezimena članova grupe (ili samo temu i prezimena, ukoliko je sa imenima predugačko). Predaja seminarskih radova biće isključivo preko web forme, a NE slanjem mejla.

## Sadržaj

<b>1</b>	<b>Uvod</b>	<b>2</b>
<b>2</b>	<b>Operaciona semantika</b>	<b>2</b>
<b>3</b>	<b>Denotaciona semantika</b>	<b>2</b>
<b>4</b>	<b>Aksiomska semantika</b>	<b>4</b>
<b>5</b>	<b>Zaključak</b>	<b>5</b>
	<b>Literatura</b>	<b>5</b>

# 1 Uvod

Ovde pišem uvod

## 2 Operaciona semantika

## 3 Denotaciona semantika

Nastala 1960. godina od strane Christopher Strachey-a i njegove istraživačke grupe na Oxford-u [5], *denotaciona semantika* predstavlja jednu vrstu reakcije na operacionu semantiku za koju se smatra da sadrži puno informacija. Naziv je dobila po engleskoj reči označiti (eng. *denote*) jer pridružuje značenja sintaksnim definicijama jezika. Alternativno, može se nazivati i *matematička semantika* zbog njene okrenutosti matematičkim formalizmima pri definisanju ove formalne semantike. Jedan način definisanja denotacione semantike je dat u sledećoj definiciji.

**Definicija 3.1** *Pristup formalizaciji semantike konstruisanjem matematičkih objekata koji opisuju značenje jezika naziva se **denotaciona semantika** [6].*

Dok se u operacionoj semantici vodilo računa o koracima izvršavanja, u denotacionoj to postaje nebitno. Na primer, značenje izraza  $(15 + 3) * (2 + 2)$  jeste 72 i ne treba obraćati pažnju na unutrašnja izračunavanja. Bitan je samo efekat koji izvršavanje programa proizvodi, odnosno odnos između početnog i završnog stanja programa. Za posmatranje ovog efekta potrebno je uočiti odnos između sintakse i semantike programskog jezika.

Ideja ove semantike je da poveže svaki deo programskog jezika sa nekim matematičkim objektom kao što je broj ili funkcija. Odavde se jasno vidi da je potrebno raščlaniti programski jezik na sintaksne delove (to nam pruža apstraktna sintaksa) i svakom delu dodeliti značenje. Svaka sintakсна definicija se tretira kao objekat na koji se može primeniti funkcija koja taj objekat preslikava u matematički objekat koji definiše značenje [3]. Dodeljivanjem značenja delovima programa dodeljuje se značenje celokupnom programu što nam govori o najvažnijem aspektu denotacione semantike.

**Definicija 3.2** *Semantika jedne programske celine definisana je preko semantike njenih poddelova. Ova osobina denotacione semantike naziva se **kompozitivnost**.*

Ovo znači da ukoliko se zameni jedan deo programske celine sa delom koji ima isto značenje, neće se promeniti značenje cele programske celine. Gore pomenuti izraz je imao semantičku vrednost 72, a to isto značenje ima i izraz  $(16 + 2) * (2 + 2)$ . To znači da se semantika izraza nije promenila iako su zamenjeni delovi izraza. Nije došlo do promene jer  $15 + 3$  ima isto značenje kao i  $16 + 2$ . Treba se još pozabaviti dodeljivanjem semantičke vrednosti delovima programske celine.

Nekim sintaksnim delovima programa je lako dodeliti semantičku vrednost. Takvi su brojevi ili aritmetički operatori jer oni već imaju svoje matematičko značenje. Ali neke sintaksne definicije poput rekurzije ili goto naredbe je teško videti kroz matematičko značenje. Daćemo primer definisanja denotacione semantike aritmetičkih izraza, dok se o rekurziji ili nekim naprednijim primerima može pročitati više u [2].

Potrebno je prvo definisati apstraktnu sintaksu aritmetičkih izraza. Neka su podržani samo prirodni brojevi i od aritmetičkih operatora  $+$ . Ovo znači da će semantička vrednost nekog izraza biti prirodan broj. Primer definicije apstraktne sintakse ovakvih aritmetičkih izraza dat je u nastavku.

#### Sintaksni domeni i pravila:

$B : Broj$                       Nenegativan broj  
 $C : Cifra$                       Cifra 0,1,...,9  
 $I : Izraz$   
 $Broj ::= Cifra | Broj Cifra$   
 $Cifra ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9$   
 $Izraz ::= Broj | Izraz + Izraz$

Sledeći korak jeste definisanje matematičkih objekata koji će predstavljati semantičke vrednosti. Ti matematički objekti nazivaju se **semantički domeni**. Njihova kompleksnost zavisi od toga koliko je kompleksan programski jezik kojem dajemo značenje. U našem jednostavnom primeru, kao što je već rečeno, semantička vrednost može biti samo prirodan broj.

#### Semantički domeni

$N = 0, 1, 2, 3, \dots$                       Skup prirodnih brojeva

Posle uvedenih objekata, treba uvesti neke matematičke funkcije koje će davati značenje prethodno uvedenim sintaksnim definicijama. Takve funkcije se nazivaju **funkcije značenja** (eng. *meaning functions*). Definicije funkcija koje su potreban za naš jednostavan primer su date u nastavku.

#### Funkcije značenja

$povezibn : B \rightarrow N$                       Unarna funkcija - povezuje broj sa N  
 $povezicn : C \rightarrow N$                       Unarna funkcija - povezuje cifru sa N  
 $semantika : I \rightarrow N$                       Unarna funkcija - povezuje izraz sa N  
 $plus : N \times N \rightarrow N$                       Binarna funkcija plus - isto što i  $+$   
 $pom : N \times N \rightarrow N$                       Binarna funkcija pom - isto što i  $*$   
 $povezicn[[0]] = 0, \dots, povezicn[[9]] = 9$   
 $povezibn[[D]] = povezicn[[D]]$   
 $povezibn[[BD]] = plus(pom(10, povezibn[[B]]), povezibn[[D]])$   
 $semantika[[B]] = povezibn[[B]]$   
 $semantika[[I1 + I2]] = plus(semantika[[I1]], semantika[[I2]])$

Primetimo da su korišćene zagrade  $[[, ]]$  koje imaju ulogu da razdvoje semantički deo od sintaksnog dela. U okviru zagrada nalazi se sintaksní deo definicija. Primer koji oslikava korišćenje denotacione semantike je dat u nastavku.

**Primer 3.1** Pronaći značenje izraza  $2+32+61$ .

Rešenje:

$semantika[[2+32+61]] = plus(semantika[[2+32]], semantika[[61]])$   
 $= plus(plus(semantika[[2]], semantika[[32]]), povezibn[[61]])$

$$\begin{aligned}
&= \text{plus}(\text{plus}(2, 32), 61) \\
&= \text{plus}(2+32, 61) \\
&= \text{plus}(34+61) = 34+61 = 95
\end{aligned}$$

jer je:

$$\text{semantika}[[2]] = \text{povezibn}[[2]] = \text{povezicn}[[2]] = 2$$

$$\begin{aligned}
&\text{semantika}[[32]] = \text{povezibn}[[32]] \\
&= \text{plus}(\text{pom}(10, \text{povezibn}[[3]]), \text{povezibn}[[2]]) \\
&= \text{plus}(\text{pom}(10, \text{povezicn}[[3]]), \text{povezicn}[[2]]) \\
&= \text{plus}(\text{pom}(10, 3), 2) = \text{plus}(10*3, 2) \\
&= \text{plus}(30, 2) = 30+2 = 32
\end{aligned}$$

$$\begin{aligned}
&\text{semantika}[[61]] = \text{povezibn}[[61]] \\
&= \text{plus}(\text{pom}(10, \text{povezibn}[[6]]), \text{povezibn}[[1]]) \\
&= \text{plus}(\text{pom}(10, \text{povezicn}[[6]]), \text{povezicn}[[1]]) \\
&= \text{plus}(\text{pom}(10, 6), 1) = \text{plus}(10*6, 1) \\
&= \text{plus}(60, 1) = 60+1 = 61
\end{aligned}$$

Prednost denotacione semantike je u tome što apstrahuje kako se programi izvršavaju. Analiziranje programa se svodi na analiziranje matematičkih objekata, što olakšava stvar. Denotaciona semantika ima veliku primenu u konstrukciji programa za prevođenje [3].

## 4 Aksiomatska semantika

Za nastanak i razvoj *aksiomatske semantike* su zaslužni pre svega Floyd, Hoar i Dijkstra [1]. Aksiomatska semantika razvija metode za proveru korektnosti programa. Zasniva se na matematičkoj logici. Za svaku kontrolnu strukturu i komandu se definišu logički izrazi. Ovi izrazi se nazivaju tvrđenja (eng. *assertions*) i u njima se zadaju ograničenja za promenljive koja se javljaju u tim kontrolnim strukturama i komandama.

Tvrđenja su data u obliku Horovih trojki:  $\{P\}c\{Q\}$

**Definicija 4.1** *Horova trojka  $\{P\}C\{Q\}$  opisuje kako izvršavanje dela koda menja stanje izracunavanja ako je ispunjen preduslov (eng. precondition)  $\{P\}$ , izvršavanje komande  $C$  vodi do postuslova (eng. postcondition)  $\{Q\}$  [6].*

Preduslov je logički izraz u kome se definišu ograničenja promenljivih pre izvršavanja komande, a postuslov definiše ograničenja promenljivih posle izvršavanja komande.

Horove trojke se drugačije nazivaju i *delimična ispravnost specifikacije* (eng. *partial correctness specification*).

Ali one ne mogu da osiguraju da će se program završiti pa se zbog toga i nazivaju “delimičnim”.

Pored delimične ispravnosti specifikacije, imamo i *potpunu ispravnost naredbe* (eng. *total correctness statements*) koja osigurava da će se program završiti dok god preduslov važi.

Preduslovi i postuslovi mogu se smatrati interfejsom ili ugovorom između programa i njegovih klijenata. Oni pomažu korisnicima da razumeju šta program treba da proizvede bez potrebe da shvati kako se program

izvršava. Tipično, programeri ih pišu kao komentari za funkcije i funkcionišu kao dokumentacija i olakšavaju održavanje programa. Takve specifikacije su posebno korisne za bibliotečke funkcije za koje izvorni kod često nije dostupan korisnicima [4].

Način funkcionisanja ove semantike možemo prikazati u primeru koji sledi (primer je preuzet iz knjige [2]).

**Primer 4.1**  $\{ x=n \}$   
 $y := 1; \text{ while } \text{neg}(x=1) \text{ do } (y := x*y; x := x-1)$   
 $\{ y=n! \text{ and } n \text{ veće od } 0 \}$

*n* je u primeru specijalna promenljiva koja se naziva logička promenljiva i koja, za razliku od programskih promenljivih, ne sme se pojaviti ni u jednoj naredbi koja se izvršava u programu i njena vrednost će uvek biti ista. Njena uloga je da pamte inicijalne vrednosti programskih promenljivih.

Zapisali smo preduslov da promenljive *n* ima jednaku vrednost kao i *x* u početnom stanju (tj. nego što program sa faktorijalom krene da se izvršava). S obzirom da program neće promeniti vrednost promenljive *n*, postuslov  $y=n!$  će izraziti da konačna vrednost *y* će biti jednaka faktorijalu početne vrednosti promenljive *x*, kada se izvršavanje programa završi.

## 5 Zaključak

Ovde pišem zaključak.

## Literatura

- [1] V. Kunčak. *Modularni interpreteri u Haskell-u*. Novi Sad, 2000.
- [2] H Nielson and F Nielson. *Semantics with Applications: An Appetizer*. Denmark, 2007.
- [3] N. Parezanović. Metodčki aspekti opisa semantike programskih jezika. *Računarstvo*, (2), 1992.
- [4] A. Sampson. *Programming Languages and Logics, Axiomatic Semantics*. United States of America, New York, 2012.
- [5] K. Slonneger and B. Kurtz. *Formal Syntax and Semantics of Programming Languages*. Addison-Wesley Publishing Company, United States of America, 1995.
- [6] M. Vujošević Janičić. *Dizajn programskih jezika, Osnovna svojstva programskih jezika*. Beograd, 2016.