# 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

# 1 Uvod

Ovde pišem uvod

# 2 Operaciona semantika

Operaciona semantika je način davanja značenja programskim jezicima kroz matematičku reprezentaciju. Svrha operacione semantike je da opiše *kako* se izvršavaju programi, ne samo koji su rezultati izvršavanja tog programa. Preciznije, ona treba da opiše kako se stanja menjaju tokom izvršavanja naredbe programa. Da bi bolje razumeli operacionu semantiku posmatraćemo je iz dva ugla:

- *Prirodna semantika* (ili veliki koraci): svrha joj je da opiše kako su *sveukupna rešenja* izvršavanja dobijena.
- *Strukturna operacijska semantika* (ili mali koraci): svrha joj je da opiše kako se *svaki korak zasebno* izvršava.

[**?**].

Za definiciju operacione semantike se uglavnom koristi strukturna operaciona semantika, dok neki koriste *apstraktne mašine*, koje više koriste ad hok matematičke konstrukcije.

Za obe vrste operacione semantike, prirodne i strukturno operacione, značenje naredbe će biti određeno *tranzicionim sistemom*, i ona će imati dve vrste konfiguracija[**?**]:

- $\langle$ S, s $\rangle$ - Naredba $S$ će biti izvršena od stanja $s$.
- s predstavlja završno stanje.

[**?**]. *Tranziciona relacija* će onda opisivati kako izgleda tok izvršavanja. U prirodnoj semantici je bitna veza između početnog i završnog stanja izvršavanja. Tada će tranziciona relacija prikazivati vezu između početnog stanja i završnog stanja za svaku naredbu. Tranziciju ćemo obeležavati kao

$\langle S, s \rangle \rightarrow s'$

Intuitivno ovo znači da izvršavanje $S$ iz $s$ će se završiti i rezultujuće stanje će biti $s'$.

Definiciju za $\rightarrow$ možemo videti u tabeli ¡broj¿. *Pravilo* generalno ima formu

$\langle S_1, s_1 \rangle \rightarrow s'_1, ..., \langle S_n, s_n \rangle \rightarrow s'_n$

gde $S_1, ..., S_n$ nazivamo neposrednim konstituentima od $S$. Pravilo se sastoji iz određenog broja *premisi* (nalazi se iznad linije) i jednog *zaključka* (nalazi se ispod linije). Pravilo takođe može imati određeni broj *uslova* (nalazi se sa desne strane linije) koji moraju biti ispunjeni kako bi se primenilo pravilo. Pravila sa praznim skupom premisa se naziva *aksiom*. [**?**]

Kada se koriste aksiomi i pravila da se izvede tranzicija $\langle S, s \rangle \rightarrow s'$ dobija se *stablo izvođenja*. *Koren* stabla izvođenja je upravo $\langle S, s \rangle \rightarrow s'$, dok su listovi instance aksioma. Unutrašnji čvorovi su zaključci instanciranih pravila i njihova neposredna deca će biti odgovarajuće premise. Stablo izvođenja se naziva *jednostavnim stablom* ako je instanca aksioma, inače se naziva *kompozitnim stablom.*// Posmatrajmo problem konstruisanja stabla izvođenja za datu naredbu $S$ i stanje $s$. Najbolji pristup ovakvom problemu jeste konstruisati stablo od korena *nagore*. Dakle, počinjemo pronalaskom aksioma ili pravila sa zaključkom gde se leva strana slaže sa konfiguracijom $\langle S, s \rangle$. Imaćemo dva slučaja:

- Ako je u pitanju aksiom i ako su uslovi aksioma ispunjeni onda možemo da zaključimo završno stanje i konstrukcija stabla izvođenja je gotova.
- Ako je u pitanju pravilo, sledeći korak je pokušati konstruisati stablo izvođenja za premise datog pravila. Kad se ovaj deo odradi, neophodno je proveriti da li su uslovi pravila ispunjeni i tek onda možemo zaključiti završno stanje $\langle S, s \rangle$.

# 3 Denotaciona semantika

Nastala 1960. godina od strane Christopher Strachey-a i njegove istraživačke grupe na Oxford-u [**?**], *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*** *[**?**].*

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 sintaksna definicija se tretira kao objekat na koji se može primeniti funkcija koja taj objekat preslikava u matematički objekat koji definiše značenje [**?**]. 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 [**?**].

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
$I : Izraz$
$Broj ::== Cifra|BrojCifra$
$Cifra ::== 0|1|2|3|4|5|6|7|8|9$
$Izraz ::== Broj|Izraz + Izraz$

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**

$N = 0, 1, 2, 3, ....$ 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**

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

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

**Primer 3.1** *Pronaći značenje izraza 2+32+61.*
*Rešenje:*

$$\textbf{\textit{semantika[[2+32+61]]}} = plus(semantika[[2+32]],semantika[[61]])$$
$$= plus(plus(semantika[[2]],semantika[[32]]),povezibn[[61]])$$
$$= plus(plus(2,32),61)$$
$$= plus(2+32,61)$$
$$= plus(34+61) = 34+61 = 95$$

*jer je:*

$$\textbf{\textit{semantika[[2]]}} = povezibn[[2]] = povezicn[[2]] = 2$$

$$\textbf{\textit{semantika[[32]]}} = povezibn[[32]]$$
$$= plus(pom(10,povezibn[[3]]),povezibn[[2]])$$
$$=plus(pom(10,povezicn[[3]]),povezicn[[2]])$$
$$=plus(pom(10,3),2) = plus(10*3,2)$$
$$= plus(30,2) = 30+2 = 32$$

$$\textbf{\textit{semantika[[61]]}} = povezibn[[61]]$$
$$=plus(pom(10,povezibn[[6]]),povezibn[[1]])$$
$$=plus(pom(10,povezicn[[6]]),povezicn[[1]])$$
$$=plus(pom(10,6),1) = plus(10*6,1)$$
$$= plus(60,1) = 60+1 = 61$$

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 [**?**].

# 4 Aksiomatska semantika

Za nastanak i razvoj *aksiomatske semantike* su zaslužni pre svega Floyd, Hoar i Dijkstra. Zasniva na matematickoj logici pa je kao takva apstraktnija od denotacione i operacione sematike. Ova semantika značenje programa zasniva na tvrdnjama o vezama koje ostaju iste svaki put kad se program izvrši [**?**]. Aksiomatska semantika razvija metode za proveru korektnosti programa. Za svaku kontrolnu strukturu i komandu se definišu logicki izrazi. Ovi izrazi se nazivaju **tvrđenja** (eng. *assertions*) i u njima se zadaju ogranicenja 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*)* $\{Q\}$ *[**?**] .*

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".

Delimične ispravnosti tvrdnje će biti određen sistemom zaključivanja koji se sastoji iz skupa aksioma i pravila. Za sve konstruktore jednostavnog imperativnog programskog jezika, Horova logika obezbeđuje aksiome

| | |
|---|---|
| $[ass_p]$ | $\{P[x \to [\![A]\!]\}x := a\{P\}$ |
| $[skip_p]$ | $\{P\}$ skip $\{P\}$ |
| $[comp_p]$ | $\frac{\{P\}S_1\{Q\},\{Q\}S_2\{R\}}{\{P\}S_1;S_2\{R\}}$ |
| $[if_p]$ | $\frac{\{B[\![b]\!]\wedge P\}S_1\{Q\},\{\neg B[\![b]\!]\wedge P\}S_1\{Q\}}{\{P\}if b then S_1 else S_2\{Q\}}$ |
| $[while_p]$ | $\frac{\{B[\![b]\!]\wedge P\}S\{P\}}{\{P\}while b do S\{\neg B[\![b]\!]\wedge P\}}$ |
| $[cons_p p]$ | $\frac{\{P^{'}\}S\{Q^{'}\}}{\{P\}S\{Q\}}$ if $P \Rightarrow P^{'}$ and $Q \Rightarrow Q^{'}$ |

Tablica 1: Aksiomatski sistem za delimičnu ispravnost

i pravila izvođenja [**?**]. Aksiomatski sistem za delimičnu ispravnost je dat u Tabeli 1 ispod.

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

Pored dokazivanja korektnosti programa i algoritama, uloga aksiomatske sematike je i dokazivanje ispravnosti harverdskih opisa (ili nalaženje bagova), proširena statička provera (npr. provera granice niza), dokumentacija programa i interfejsa.

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 [**?**].

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

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

*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.