Osnove formalne semantike programskih jezika

Seminarski rad u okviru kursa
Metodologija strunog i naunog rada
Matematiki fakultet

Isidora urevi, Ana Stankovi,
Milica uri
isidoradjurdjevic.100@gmail.com, anastankovic167@gmail.com,
mdjuric55@gmail.com

1. maj 2017.

Saetak

Semantika ima posebnu ulogu u teoriji i prouavanju znaenja programskih jezika. Cilj ovog rada je da pokae koja je uloga formalnog zadavanja semantike programskih jezika i osnovne primene formalne semantike. Bie definisana tri osnovna pristupa semantike: operaciona, denotaciona i aksiomatska semantika i objanjene njihove uloge. Svaki od pristupa je potkrepljen primerom koji prikazuje primenu te semantike u raunarstvu.

Kljune rei: semantika programskih jezika, formalna semantika, operaciona semantika, denotaciona semantika, znaenje programa.

Sadržaj

1 Uvod

Voenje konverzacije ne bi bilo mogue bez znanja o tome ta koja re koju koristimo predstavlja. U naoj glavi stoje jasne neformalne definicije svake rei koje znamo i pomou kojih povezujemo objekte sa njihovim znaenjem. U lingvistici, nauci o jeziku, svaka re dobija svoju formalnu definiciju, a pravila formiranja reenica su jako rigorozna, odnosno, postoje jasne odredbe ta je ispravno, a ta neispravno. Ove formalne definicije i pravila omoguavaju onima koji se susreu sa jezikom po prvi put da ga lake naue i razumeju kako bi mogli da ga koriste. Slino, ovo moe biti primenjeno i na programske jezike.

Kao to je izuavanje prirodnog jezika podeljeno na izuavanje strukture onoga ime se neto izraava, tj. *sintakse* i izuavanje znaenja onoga to je izraeno, tj. *semantike*, tako se i prouavanje programskog jezika moe podeliti na prouavanje njegovog formiranja i njegovog znaenja. Osnovni zadatak sintakse programskih jezika je da omogui formiranje korektnih programa iz ugla strukture izraza. Izrazi poput $x++; function(x, y);$ su sintaksno ispravni, ali ta oni predstavljaju, koje je njihovo znaenje? Sintaksa nema odgovor na ovo pitanje jer ona nije povezana sa znaenjem ili ponaanjem programa u toku izvoenja. To je zadatak ***semantike programskih jezika***. Meutim, sintaksa mora biti definisana pre semantike jer se znaenje moe pridruiti samo ispravnim izrazima jezika. Definisanje sintakse je obino laki posao od definisanja semantike.

Semantika moe da se opie formalno i neformalno, gde je ee neformalno opisivanje, odnosno opisivanje prirodnim jezikom to moe biti neprecizno. Zato se u narednim glavama bavimo formalnim opisom semantike koja bi trebalo da otkloni sve nepreciznosti i da dâ potpunu definiciju programskog jezika. Bie opisani neki od formalnih pristupa kao to su *operaciona, denotaciona i aksiomatska* semantika. Svakom od pristupa e se prii povrinski zbog same sloenosti formalnog definisanja, a itaocu se ostavlja da proiri svoje znanje u naznaenoj literaturi.

2 Formalna semantika

Kao to je ve napomenuto, semantika se moe definisati formalno i neformalno. Pri neformalnom definisanju koristimo prirodan jezik. Na primer, izraz

$$while(x > 0)\ \{x--;\}$$

moe se opisati reima : "sve dok je vrednost promenljive x vea od nule umanjuj vrednost promenljive x za jedan". Meutim, da li je to jednostavno uraditi za svaki deo programskog koda ili za svaki programski jezik?

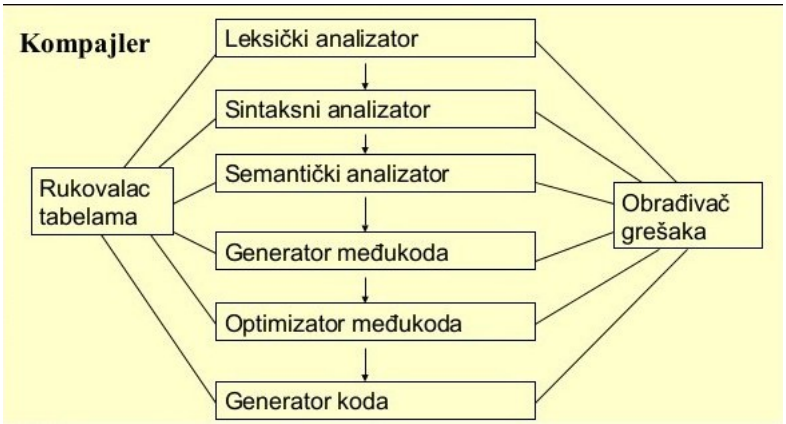
Teko je pisati vrlo precizne definicije neformalnim jezikom, obino postane komplikovano za razumeti i predugako za neke kompleksne programske jezike to dovodi do loe napisanih programa jer se stvaralac programskog jezika i onaj ko ga koristi ne razumeju. To je jedan od razloga zato se uvodi *formalna semantika*, to znai precizna semantika.

Pri uenju novog programskog jezika, programer bi trebalo da ui i samu logiku jezika. Neki vid opisa logike programskog jezika olakava programerima pisanje jakih i stabilnih programa. Ovaj opis nam moe pruiti, takoe, formalna semantika.

Jo jedan od razloga zato se uvodi formalna semantika jeste to to ona ini osnovu formalne verifikacije programa, odnosno, deo je metoda za ispitiva-

nje ispravnosti programa koji koristi matematike dokaze kako bi pokazao da program zadovoljava zadatu specifikaciju.

Formalna semantika ini i deo interpretera i kompilatora. Koristi se kako bi se proverilo da li je program semantiki zadovoljavaju. Obino izbacuje upozerenje, a ne greku ako uoi neke nepravilnosti. Na slici 1 dat je prikaz strukture kompilatora.



Slika 1: Struktura kompilatora [1]

Kako u 21. veku sve ide ka tome da bude automatizovano, tako se trailo i reenje za automatsko rezonovanje o semantikim svojstvima programa. Formalna semantika se, pre svega, koristi ba u te svrhe. Da nema formalne semantike, proces automatskog rezonovanja o semantikim svojstvima programa bio bi nemogu jer maini je nemogue zadati neformalnu semantiku kao sredstvo za opisivanje semantike.

Kompleksnost uvoenja formalne semantike daje vie naina pristupa njenom definisanju. Mi emo se fokusirati na tri pristupa:

1. **Operaciona semantika**

   Znaenje programa predstavlja tok koraka izvravanja programa sa datim ulazom.

2. **Denotaciona semantika**

   Znaenje programa je matematika funkcija koja prevodi ulaz u izlaz programa.

3. **Aksiomatska semantika**

   Znaenje programa je ono to moe biti dokazano o njemu koristei aksiome.

U sledeim poglavljima, svaka od ovih formalnih semantika e biti opisana ire.

2.1 Operaciona semantika

Operaciona semantika je nain davanja znaenja programskim jezicima kroz matematiku reprezentaciju. Svrha operacione semantike je da opie *kako* se izvravaju programi, ne samo koji su rezultati izvravanja tog programa. Preciznije, ona treba da opie kako se stanja menjaju tokom izvravanja naredbe programa. U okviru ove semantike postoje [3]:

- *Prirodna semantika* (ili veliki koraci (eng. *big-step semantics*)): svrha joj je da opie kako su *ukupni rezultati* izvravanja dobijeni.
- *Strukturna operaciona semantika* (ili mali koraci (eng. *small-step semantics*)): svrha joj je da opie kako se *svaki korak* izvrava.

Jedna od osnovnih karakteristika imperativnih jezika je da ovi jezici imaju implicitno stanje i da se izvravanje programa svodi na postepeno menjanje tog stanja izvoenjem pojedinanih naredbi [10]. Takoe, redosled naredbi je bitan [10]. S obzirom da sve ove karakteristike zadovoljava operaciona semantika, moemo rei da e ova semantika najvie odgovarati imperativnim jezicima.

Ponaanje se formalno moe definisati korienjem apstraktnih maina, formalnih automata, tranzicionih sistema... U daljem tekstu emo prikazati primer korienja apstraktne maine za definiciju, kao i upotrebu tranzicionih sistema.

Apstraktne maine se koriste za interpretaciju implementacije odreenog programskog jezika (bilo imperativnog, funkcionalnog ili logikog) jer olakavaju prenosivost, optimizaciju koda kao i prevoenje na mainski jezik. Takoe, olakavaju razumevanje pojedinih konstrukcija programskog jezika kao i kako se stanja programa menjaju pri njihovoj primeni. Posmatrajmo zapis for petlje u jeziku C:

for (i = 0; i <end; i++)
.
.
.

Iz ovakvog zapisa se vidi da e se ciklus izvravati *end* puta, sve dok je vrednost promenljive *i* manja od vrednosti promenljive *end* i da e u trenutku kada te dve promenljive dobiju iste vrednosti prestati izvravanje petlje, gde u ovom konkretnom sluaju ignoriemo ta se deava u telu ciklusa. Takoe treba primetiti da nam ovo objanjenje daje *apstraktnu* definiciju izvravanja for petlje na maini, jer ne posmatramo stvari kao to su vrednosti adresa i registara. Operaciona semantika je nezavisna od arhitekture maine i naina implementacije [6].

Uvedimo pojam konfiguracije [4].

Definicija 2.1 *Konfiguracija je par* $\langle\ c, \omega\ \rangle$*, gde c predstavlja naredbu i pripada skupu naredbi C, dok* ω *predstavlja ulazne vrednosti za naredbu c.*

Opiimo sada znaenje koje se predstavlja na isti nain za obe vrste operacione semantike, prirodne i strukturno operacione, *tranzicionim sistemom* (eng. *transition system*) na primeru rada *While* petlje. Ona e imati dve vrste konfiguracija [3]:

- $\langle$ S, s $\rangle$ - Naredba S e biti izvrena od stanja s.
- s' predstavlja zavrno stanje.

Tranziciona relacija (eng. *transition relation*) e onda opisivati kako izgleda tok izvravanja.

2.1.1 Prirodna semantika

Prirodna semantika je apstrakcija. U njoj je bitna veza izmeu poetnog i zavrnog stanja izvravanja. Tada e tranziciona relacija prikazivati vezu

izmeu poetnog stanja i zavrnog stanja za svaku naredbu. Tranziciju emo obeleavati kao:

$$\langle\ S,\ s\ \rangle \rightarrow s\text{'}$$

Intuitivno ovo znai da izvravanje programa S sa ulaznim stanjem s e se zavriti i rezultujue stanje e biti s'. Drugi deo formule je samo delimino definisan, neophodno je naglasiti da je mogue da program S zavri sa radom i da rezultujue stanje bude neko drugo stanje razliito od s', kao i da je mogue da program S uopte ne zavri sa radom. Pravo znaenje programa S bi tada bilo:

$$\{(s,\ s\text{'}) : \langle S,\ s\rangle \rightarrow s\text{'}\}$$

pa nam semantika pravila omoguavaju da generiemo skup tanih to jest istinitih iskaza, gde su svi ostali iskazi netani po definiciji.[3] Definicija za $\rightarrow$ je prikazana preko pravila datih u tabeli 1. *Pravilo* generalno ima formu

$$\frac{\langle S_1,s_1\rangle\rightarrow s'_1,\ldots,\langle S_n,s_n\rangle\rightarrow s'_n}{\langle S,s\rangle\rightarrow s'}$$

gde S_1, ..., S_n nazivamo neposrednim konstituentima (eng. *immediate constituents*) od S. Pravilo se sastoji iz odreenog broja *premisa* (nalaze se iznad linije) i jednog *zakljuka* (nalazi se ispod linije).

Pravilo takoe moe imati odreeni broj *uslova* (nalaze se sa desne strane linije) koji moraju biti ispunjeni kako bi se primenilo pravilo. Pravilo sa praznim skupom premisa se naziva *aksiom*. [3]

Tablica 1: Prirodna semantika za While jezik

$[ass_{ns}]$	$\langle x := a, s\rangle \rightarrow s[x \mapsto A[[a]]s]$
$[skip_{ns}]$	$\langle skip, s\rangle \rightarrow s$
$[comp_{ns}]$	$\frac{\langle S_1,s\rangle\rightarrow s',\langle S_2,s'\rangle\rightarrow s''}{\langle S_1;S_2,s\rangle\rightarrow s''}$
$[if^{tt}_{ns}]$	$\frac{\langle S_1,s\rangle\rightarrow s'}{\langle if\ b\ then\ S_1\ else\ S_2,\ s\rangle\rightarrow s'}\ if\ B[[b]]s\ =\ tt$
$[if^{ff}_{ns}]$	$\frac{\langle S_2,s\rangle\rightarrow s'}{\langle if\ b\ then\ S_1\ else\ S_2,\ s\rangle\rightarrow\ s'}\ if\ B[[b]]s\ =\ ff$
$[while^{tt}_{ns}]$	$\frac{\langle S,s\rangle\rightarrow s',\langle\ while\ b\ do\ S,\ s'\rangle\rightarrow\ s''}{\langle\ while\ b\ do\ S,\ s\rangle\rightarrow\ s''}\ if\ B[[b]]s\ =\ tt$
$[while^{ff}_{ns}]$	$\langle\ while\ b\ do\ S,\ s\rangle \rightarrow\ s\ if\ B[b]s\ =\ ff$

Kada se koriste aksiomi i pravila da se izvede tranzicija $\langle\ S,\ s\ \rangle \rightarrow s$', to jest da se dokae iskaz, dobija se *stablo izvoenja*. *Koren* stabla izvoenja je upravo $\langle\ S,\ s\ \rangle \rightarrow s$', dok su listovi instance aksioma. Unutranji vorovi su zakljuci instanciranih pravila i njihova neposredna deca e biti odgovarajue premise. Stablo izvoenja se naziva *jednostavnim stablom* ako je instanca aksioma, inae se naziva *kompozitnim stablom*.

Primer 2.1 *($z := x$, $x := y$); $y := z$*
Neka s_0 bude stanje koje mapira sve promenljive osim x i y u 0 i ima s_0 $x = 5$ i s_0 $y = 7$. Tada dobijamo sledee stablo izvoenja:

$$\frac{\dfrac{\langle z{:=}x,s_0\rangle\to s_1 \quad \langle x{:=}y,s_1\rangle\to s_2}{\langle z{:=}x,x{:=}y,s_0\rangle\to s_2} \quad \langle y{:=}z,s_2\rangle\to s_3}{\langle (z{:=}x;x{:=}y);y{:=}z,s_0\rangle\to s_3}$$

Posmatrajmo problem konstruisanja stabla izvoenja za datu naredbu S i stanje s. Najbolji pristup ovakvom problemu jeste konstruisati stablo od korena *nagore*. Dakle, poinjemo pronalaskom aksioma ili pravila sa zakljukom gde se leva strana slae sa konfiguracijom $\langle\ S,\ s\ \rangle$. Imaemo dva sluaja:

- Ako je u pitanju aksiom i ako su uslovi aksioma ispunjeni onda moemo da zakljuimo zavrno stanje i konstrukcija stabla izvoenja je gotova.
- Ako je u pitanju pravilo, sledei korak je pokuati konstruisati stablo izvoenja za premise datog pravila. Kad se ovaj deo odradi, neophodno je proveriti da li su uslovi pravila ispunjeni i tek onda moemo zakljuiti zavrno stanje $\langle\ S,\ s\ \rangle$.

Vie o ovoj temi se moe pronai u [3].

2.1.2 Strukturna operaciona semantika

Prirodna semantika nekada nije pogodna za obimnu analizu, tada se uglavnom koristi *strukturna operaciona semantika* (skraeno SOS), kojom je mogue opisati imperativne programe kao i neke sloene sluajeve, kao to su na primer, semantika pokazivaa i semantika vienitne obrade ili ak baratati sa „goto" naredbama. U SOS-u emo tranzicionu relaciju zapisivati kao:

$$\langle\ S,\ s\ \rangle \Longrightarrow \gamma$$

ovo treba razmatrati kao *prvi korak* izvravanja programa S u stanju s koji vodi do stanja γ [3].
Moemo razlikovati dva sluaja za γ:

1. $\gamma = \langle\ S',\ s'\ \rangle$: izvravanje programa S sa ulaznim stanjem s *nije zavreno*, i ostatak izraunavanja e biti izraeno srednjom konfiguracijom $\langle\ S',\ s'\ \rangle$.
2. $\gamma = s'$: izvravanje programa S sa ulaznim stanjem s se zavrilo sa zavrnim stanjem s'.

U sluaju da resultat izvravanja $\langle\ S',\ s'\ \rangle$ nije dostupan kaemo da je γ *zaglavljena* (eng. *stuck*) konfiguracija i da nema sledeih tranzicija. *Znaenje* programa P za ulazno stanje s je skup *zavrnih* stanja (kao i zaglavljenih konfiguracija) koji mogu biti izvravani u proizvoljnom redosledu u konanom broju koraka.

Primer 2.2 *Posmatrajmo ponovo sluaj While-a.*
Prva dva aksioma $[ass_{sos}]$ *i* $[skip_{sos}]$ *su nepromenjena jer se oba izraza izvravaju u potpunosti u jednom koraku.*

- $[ass_{sos}]$ $\qquad \langle\ x := a,\ s\ \rangle \Longrightarrow s[x \mapsto A[a]]$
- $[skip_{sos}]$ $\qquad \langle\ skip,\ s\ \rangle \Longrightarrow s$

Druga dva pravila $[comp^1_{sos}]$ *i* $[comp^2_{sos}]$ *za programe* $S_1;S_2$ *govore da izvravanje poinje prvim korakom iz* S_1 *i ulaznim stanjem s. Tada postoje dva mogua zavrna stanja u zavisnosti od dva sluaja:*

- $[comp^1_{sos}]$ - *Izvravanje programa* S_1 *nije zavreno. Tada je neophodno da taj program zavri sa radom pre poetka rada programa* S_2*. U ovom sluaju prvi korak* $\langle\ S,\ s\ \rangle$ *je srednja konfiguracija* $\langle\ S_1',\ s'\ \rangle$ *i sledea konfiguracija je* $\langle\ S_1';S_2,s'\ \rangle$.
  $[comp^1_{sos}]$ $\qquad \dfrac{\langle S_1,s\rangle \Longrightarrow \langle S_1',s'\rangle}{\langle S_1;S_2,s\rangle \Longrightarrow \langle S_1';S_2,s'\rangle}$

- $[comp^2_{sos}]$ - *Izvravanje programa S_1 je zavreno. Tada moemo poeti sa izvravanjem programa S_2. U sluaju da je rezultat izvravanja programa S_1 sa ulaznim stanjem s stanje s' tada je sledea konfiguracija* $\langle\ S_2,\ s'\rangle$

  $[comp^2_{sos}]$ $\quad \frac{\langle S_1,s\rangle \Longrightarrow s'}{\langle S_1;S_2,s\rangle \Longrightarrow \langle S_2,s'\rangle}$

Prvi korak uslovnog grananja poinje sa ispitivanjem istinitosne vrednosti izraza b-a, zatim grananjem u zavisnosti od ishoda:

- $[if^{tt}_{sos}]\ \langle\ if\ b\ then\ S_1\ \ else\ S_2,\ s\ \rangle \Longrightarrow \langle\ S_1,\ s\ \rangle \qquad if\ B[b]s = tt$
- $[if^{ff}_{sos}]\ \langle\ if\ b\ then\ S_1\ \ else\ S_2,\ s\ \rangle \Longrightarrow \langle\ S_2,\ s\ \rangle \qquad if\ B[b]s = ff$

Prvi korak kod ,,while" petlje jeste odrediti uslov kada se izlazi iz petlje, zatim u sledeem koraku izvravanja treba proveriti da li je ispunjen taj uslov i da li je samim tim mogue nastaviti izvravanje petlje.

$[while_{sos}]\ \langle\ while\ b\ do\ S,s\ \rangle \Longrightarrow \langle\ if\ b\ then\ (S;\ while\ b\ do\ S)\ else\ skip,s\ \rangle$

Vie o navedenom moete pronai u [3] i podstiemo itaoca da pogleda navedene izvore.

2.2 Denotaciona semantika

Nastala 1960. godina od strane Kristofera Strejija (eng. *Christopher Strachey*) i njegove istraivake grupe na Oksfordu [8], *denotaciona semantika* predstavlja jednu vrstu reakcije na operacionu semantiku za koju se smatra da sadri puno informacija. Naziv je dobila po engleskoj rei oznaiti (eng. *denote*) jer pridruuje znaenja sintaksnim definicijama jezika. Alternativno, moe se nazivati i *matematika semantika* zbog njene okrenutosti matematikim formalizmima pri definisanju ove formalne semantike. Jedan nain definisanja denotacione semantike je dat u sledeoj definiciji.

Definicija 2.2 *Pristup formalizaciji semantike konstruisanjem matematikih objekata koji opisuju znaenje jezika naziva se* ***denotaciona semantika*** *[9].*

Dok se u operacionoj semantici vodilo rauna o koracima izvravanja, u denotacionoj to postaje nebitno. Na primer, znaenje izraza (15+3)∗(2+2) jeste 72 i ne treba obraati panju na unutranja izraunavanja. Bitan je samo efekat koji izvravanje programa proizvodi, odnosno odnos izmeu poetnog i zavrnog stanja programa. Za posmatranje ovog efekta potrebno je uoiti odnos izmeu sintakse i semantike programskog jezika.

Ideja ove semantike je da povee svaki deo programskog jezika sa nekim matematikim objektom kao to je broj ili funkcija. Odavde se jasno vidi da je potrebno ralaniti programski jezik na sintaksne delove (to nam prua apstraktna sintaksa) i svakom delu dodeliti znaenje. Svaka sintaksna definicija se tretira kao objekat na koji se moe primeniti funkcija koja taj objekat preslikava u matematiki objekat koji definie znaenje [6]. Dodeljivanjem znaenja delovima programa dodeljuje se znaenje celokupnom programu to nam govori o najvanijem aspektu denotacione semantike.

Definicija 2.3 *Semantika jedne programske celine definisana je preko semantike njenih poddelova. Ova osobina denotacione semantike naziva se* ***kompozitivnost****.*

Ovo znai da ukoliko se zameni jedan deo programske celine sa delom koji ima isto znaenje, nee se promeniti znaenje cele programske celine. Gore pomenuti izraz je imao semantiku vrednost 72, a to isto znaenje ima i izraz $(16 + 2) * (2 + 2)$. To znai da se semantika izraza nije promenila

iako su zamenjeni delovi izraza. Nije dolo do promene jer $15 + 3$ ima isto znaenje kao i $16 + 2$. Treba se jo pozabaviti dodeljivanjem semantike vrednosti delovima programske celine.

Nekim sintaksnim delovima programa je lako dodeliti semantiku vrednost. Takvi su brojevi ili aritmetiki operatori jer oni ve imaju svoje matematiko znaenje. Ali neke sintaksne definicije poput rekurzije ili goto naredbe je teko videti kroz matematiko znaenje. Daemo primer definisanja denotacione semantike aritmetikih izraza, dok se o rekurziji ili nekim naprednijim primerima moe proitati vie u [5].

Potrebno je prvo definisati apstraktnu sintaksu aritmetikih izraza. Neka su podrani samo prirodni brojevi i od aritmetikih operatora $+$. Ovo znai da e semantika vrednost nekog izraza biti prirodan broj. Primer definicije apstraktne sintakse ovakvih aritmetikih izraza dat je u nastavku.

Sintaksni domeni i pravila:

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

Sledei korak jeste definisanje matematikih objekata koji e predstavljati semantike vrednosti. Ti matematiki objekti nazivaju se **semantiki domeni**. Njihova kompleksnost zavisi od toga koliko je kompleksan programski jezik kojem dajemo znaenje. U naem jednostavnom primeru, kao to je ve reeno, semantika vrednost moe biti samo prirodan broj.

Semantiki domeni

$N = 0, 1, 2, 3, ...$ skup prirodnih brojeva

Posle uvedenih objekata, treba uvesti neke matematike funkcije koje e davati znaenje prethodno uvedenim sintaksnim definicijama. Takve funkcije se nazivaju **funkcije znaenja** (eng. *meaning functions*). Definicije funkcija koje su potrebne za na jednostavan primer su date u nastavku.

Funkcije znaenja

$povezibn : B \to N$ 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 to i $+$
$pom : N \times N \to N$ binarna funkcija pom - isto to i *
$povezicn[[0]] = 0, ..., povezicn[[9]] = 9$
$povezibn[[C]] = povezicn[[C]]$
$povezibn[[BC]] = plus(pom(10, povezibn[[B]]), povezibn[[C]])$
$semantika[[B]] = povezibn[[B]]$
$semantika[[I1 + I2]] = plus(semantika[[I1]], semantika[[I2]])$

Primetimo da su koriene zagrade $[[,]]$ koje imaju ulogu da razdvoje

semantiki deo od sintaksnog dela. U okviru zagrada nalazi se sintaksni deo definicija. Primer koji oslikava korienje denotacione semantike je dat u nastavku.

Primer 2.3 *Pronai znaenje izraza 2+32+61.*

Reenje:

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

jer je:

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

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

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

S obzirom na to da se funkcionalno programiranje zasniva na pojmu matematikih funkcija i da se izvravanje programa svodi na evaluaciju funkcija [10], denotaciona semantika je najbolja za definisanje semantike funkcionalnih programskih jezika.

Prednost denotacione semantike je u tome to apstrahuje kako se programi izvravaju. Analiziranje programa se svodi na analiziranje matematikih objekata, to umnogome olakava stvar.

2.3 Aksiomatska semantika

Za nastanak i razvoj *aksiomatske semantike* su zasluni pre svega Robert Flojd (eng. *Robert Floyd*), Toni Hor (eng. *Antony Hoare*) i Edsger Dijkstra (hol. *Edsger Dijkstra*). Zasniva se na matematikoj logici pa je kao takva apstraktnija od denotacione i operacione sematike. Ova semantika znaenje programa zasniva na tvrdnjama o vezama koje ostaju iste svaki put kad se program izvri [8]. Aksiomatska semantika razvija metode za proveru korektnosti programa. Za svaku kontrolnu strukturu i komandu se definiu logiki izrazi. Ovi izrazi se nazivaju **tvrenja** (eng. *assertions*) i u njima se zadaju ogranienja za promenljive koja se javljaju u tim kontrolnim strukturama i komandama.

Tvrenja su data u obliku Horovih trojki:

$$\mathbf{\{P\}c\{Q\}}$$

Definicija 2.4 ***Horova trojka*** $\mathbf{\{P\}C\{Q\}}$ *opisuje kako izvravanje dela koda menja stanje izraunavanja ako je ispunjen preduslov (eng.* precondition*)* $\{P\}$*, izvravanje komande C vodi do postuslova (eng.* postcondition*)* $\{Q\}$ *[9] .*

Preduslov je logiki izraz u kome se definiu ogranienja promenljivih pre izvravanja komande, a postuslov definie ogranienja promenljivih posle izvravanja komande. Horove trojke se drugaije nazivaju i *parcijalna ispravnost specifikacije* (eng. *partial correctness specification*) . Ali one ne mogu da osiguraju da e se program zavriti pa se zbog toga i nazivaju parcijalnim.

Parcijalne ispravnosti specifikacija bie odreene sistemom zakljuivanja koji se sastoji iz skupa aksioma i pravila. Za sve konstruktore jednostavnog imperativnog programskog jezika, Horova logika obezbeuje aksiome i pravila izvoenja [9]. Aksiomatski sistem za parcijalnu ispravnost je dat u Tabeli 2 ispod.

Tablica 2: Aksiomatski sistem za parcijalnu ispravnost

$[ass_p]$	$\{P[x \to [\![A]\!]\}\ x := a\ \{P\}$
$[skip_p]$	$\{P\}$ 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 $Q \Rightarrow Q^{'}$

Pored parcijalne ispravnosti specifikacija, imamo i *potpunu ispravnost naredbe* (eng. *total correctness statements*) koja osigurava da e se program zavriti dok god preduslov vai.

Osim dokazivanja korektnosti programa i algoritama, uloga aksiomatske sematike je i dokazivanje ispravnosti harverdskih opisa (ili nalaenje bagova), proirena statika provera (npr. provera granice niza), dokumentacija programa i interfejsa [2] .

Preduslovi i postuslovi mogu se smatrati interfejsom ili ugovorom izmeu programa i njegovih klijenata. Oni pomau korisnicima da razumeju ta program treba da proizvede bez potrebe da shvati kako se program izvrava. Tipino, programeri ih piu kao komentare za funkcije i funkcioniu kao dokumentacija i olakavaju odravanje programa. Takve specifikacije su posebno korisne za biblioteke funkcije za koje izvorni kod esto nije dostupan korisnicima [7].

Nain funkcionisanja ove semantike moemo prikazati u primeru sa faktorijalom koji sledi (primer je preuzet iz knjige [5]).

Primer 2.4 *Izraunati faktorijal*

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

n je u primeru specijalna promenljiva koja se naziva logika promenljiva i koja, za razliku od programskih promenljivih, ne sme se pojaviti ni u jednoj naredbi koja se izvrava u programu i njena vrednost e uvek biti ista. Njena uloga je da pamte inicijalne vrednosti programskih promenljivih. Zapisali smo preduslov da promenljiva n ima jednaku vrednost kao i x

u poetnom stanju (tj. pre nego to program sa faktorijalom krene da se izvrava). S obzirom da program nee promeniti vrednost promenljive n, postuslov y=n! e izraziti da e konana vrednost y biti jednaka faktorijalu poetne vrednosti promenljive x, kada se izvravanje programa zavri.

3 Zakljuak

Prilikom izuavanja programskih jezika treba obratiti panju kako na sintaksu, tako i na semantiku. Primena formalnih opisa semantike zahteva matematiko znanje i apstraktno razmiljanje, a kada je ono omogueno, onda poznavanje osnova formalne semantike programskih jezika omoguava programerima da lake shvate tok i rezultat izvravanja programa.

Rad se zasniva samo na osnovama formalne semantike i slui da uvede i zainteresuje itaoca za jednu od veoma kompleksnih tema programiranja. Dalje istraivanje bi ilo u dubinu, odnosno, trebalo bi istraiti kako se neki kompleksniji fragmenti programskih kodova semantiki definiu iz sva tri ugla - operacionog, denotacionog i aksiomatskog. Takoe, mogao bi se pruiti uvid u primenu formalne semantike pri konstruisanju kompilatora.

Literatura

[1] Programski jezici. https://www.slideshare.net/SndorKvg/03-programski-jezici, 2014. [Online; accessed 14-April-2017].

[2] B. Evan Chang. *Introduction to Axiomatic Semantics*. United States of America, 2009.

[3] Flemming Neilson Hanne Riis Nielson. *Semantics with applications*. John Willey and Sons, 1999.

[4] Andrew Myers. IMP: Big-Step and Small-Step Semantics. *CS611 Lecture 5*, 2007.

[5] H.R. Nielson and F. Nielson. *Semantics with Applications: An Appetizer*. Springer, London, 2007.

[6] N. Parezanović. Metodički aspekti opisa semantike programskih jezika. *Računarstvo*, (2), 1992.

[7] A. Sampson. *Programming Languages and Logics, Axiomatic Semantics*. United States of America, New York, 2012.

[8] K. Slonneger and B. Kurtz. *Formal Syntax and Semantics of Programming Languages*. Addison-Wesley Publishing Company, United States of America, 1995.

[9] M. Vujošević Janičić. *Dizajn programskih jezika, Osnovna svojstva programskih jezika*. Beograd, 2016.

[10] M. Vujošević Janičić. *Programske paradigme, Funkcionalna paradigma*. Beograd, 2016.