Aksiomatska semantika kroz primere

Seminarski rad u okviru kursa
Verifikacija softvera
Matematički fakultet

Nikola Katić, 1094/2017
mi13093@alas.matf.bg.ac.rs

12. decembar 2018

Sažetak

Ovaj rad za cilj ima predstavljanje osnovnih pojmova aksiomatske semantike. Prikazan je jedan sistem zaključivanja svojstava parcijalne korektnosti i objašnjene su aksiome iz datog sistema. Na primerima je pokazana primena aksioma i mogućnost rezonovanja o svojstvima u imperativnom jeziku **while** opisanom u dodatku. Rad je pisan tako da ukoliko je to moguće, ne ostavi osnovne matematičko-logičke pojmove nerazjašnjene, pa su i oni sadržani u dodatku.

Sadržaj

1 Uvod

Formalan opis značenja jezičkih konstrukcija poput operacione i denotacione semantike specifikuje značenje programa, ali se takođe može koristiti i za dokazivanje određenih svojstava programa. Dokazi u operacionoj i denotacionoj semantici mogu biti previše detaljni da bi bili korisni u praksi. Razlog tome je činjenica da su usko povezani sa semantikom programskog jezika. Zato bi bilo pogodno da se uoče *suštinska svojstva* različitih jezičkih konstrukata, tako da sprovođenje dokaza bude manje zahtevno. Time biva sužen i tip svojstava koja mogu biti dokazana.

Mogu se razlikovati dve klase svojstava - svojstva *parcijalne* i *potpune* tačnosti. Prva podrazumevaju da ukoliko se traženi program završava, onda postoji određena veza između početne i krajnje vrednosti promenljivih. To znači da ova svojstva ne garantuju da će se program završiti. Svojstva potpune tačnosti dodatno garantuju i da će se program završiti. Sem ovih, postoje druge klase svojstava koje razmatraju i *resurse* koji se koristi tokom izvršavanja programa.

U okviru ovog seminarskog rada će radi jednostavnosti biti obrađivana gotovo isključivo svojstva parcijalne korektnosti. Prilikom realizacije sistema za dedukovanje totalne korektnosti u [3], suštinska razlika u odnosu na sistem za parcijalnu korektnost se ogleda u samo jednoj aksiomi[1].

Ideja je da se navedu svojstva programa kao *tvrdnje* (eng. *assertions*). Svaka tvrdnja je Horova[2] trojka oblika $\{\ P\ \}\ S\ \{\ Q\ \}$ gde je S program, a P i Q su predikati. Ovde P predstavlja *preduslov*, a Q *postuslov*. Intuitivno, značenje trojke $\{\ P\ \}\ S\ \{\ Q\ \}$ je

> *ako* u početnom stanju važi uslov P, i
> *ako* se S zaustavlja kada je pokrenuto u tom stanju,
> *onda* će Q važiti u stanju u kojem se S zaustavilo

Treba primetiti da trojka $\{\ P\ \}\ S\ \{\ Q\ \}$ ne zahteva da se S zaustavlja za sva početna stanja koja zadovoljavaju P, nego da *ukoliko* se zaustavlja, onda važi i Q u krajnjem stanju[3, 6].

2 Osnovni pojmovi

Da bi pravila aksiomatske semantike bilo moguće primenjivati, najpre je potrebno uvesti osnovne pojmove. U ovom poglavlju biće pojašnjen pojam sistema zaključivanja i pojam logičke promenljive. Takođe će biti napravljena razlika između tipova sistema.

2.1 Logičke promenljive

Pojam logičke promenljive biće ilustrovan na primeru. Poštujući prethodno opisanu formu, može se navesti naredna trojka

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

Na ovaj način se izražava da ukoliko je vrednost x bila jednaka vrednosti n neposredno *pre* nego što se program koji treba da računa faktorijel

[1] U pitanju je $[\mathit{while}_t]$ aksioma sa parametrizovanom familijom predikata koja se koristi kao invarijanta petlje.

[2] Toni Hor (eng. *Sir Charles Antony Richard Hoare*), britanski naučnik, dobitnik Tjuringove nagrade 1980. godine.

pokrenuo, onda će vrednost y biti jednaka faktorijelu broja n *nakon* što se program završi – pod uslovom da se program zaista zaustavlja.

Promenljive x i y iz ovog primera se nazivaju *programskim*, jer se javljaju u naredbama. Sem programskih, postoje i *logičke* promenljive, koje se ne smeju pojavljati ni u jednoj naredbi. Njihova uloga je da se "zapamte" početne vrednosti programskih promenljivih, a kako se ne pojavljuju u programu, ni njihova vrednost se ne menja.

U prethodnom primeru, specijalna promenljiva n je *logička*, a može se primetiti da ukoliko se $\{ y = n! \ \wedge \ n > 0 \}$ zameni sa novim postuslovom $\{ y = x! \wedge x > 0 \}$, onda takva trojka predstavlja vezu između krajnje vrednosti y i krajnje vrednosti x, a to nije opis željene veze. Takav problem se prevazilazi korišćenjem logičkih promenljivih, jer one omogućavaju referisanje (u postuslovu) na inicijalne vrednosti programskih promenljivih[3].

2.2 Tipovi sistema zaključivanja

Za navođenje preduslova i postuslova, razlikuju se dva suprotstavljena pristupa. *Intenzionalni* pristup podrazumeva uvođenje eksplicitnog jezika tvrđenja (eng. *assertion language*) čije formule služe za predstavljanje stanja programa. Takav jezik mora biti veoma ekspresivan, kako bi sve preduslove i postuslove bilo moguće opisati. *Ekstenzionalni* pristup se u izvesnom smislu može tumačiti kao lakši od pomenuta dva, pa će u nastavku jedini biti razmatran. Ideja je da se stanja predstave kao predikati, tj. funkcije oblika $\mathbf{Stanje} \to \mathbf{T}$[3]. Stoga se značenje $\{ P \} S \{ Q \}$ može preformulisati kao: P važi za stanje s i ako se S može izvršiti s početnim stanjem s tako da rezultuje stanjem s', onda Q važi za stanje s'. S obzirom da se bilo koji predikati mogu koristiti, gorepomenuti problem sa ekpresivnošću biva prevaziđen. Notacija korišćena u nastavku teksta opisana je u dodatku A.

Svakom bulovskom izrazu b se dodeljuje semantika u vidu predikata $\mathcal{B}[\![b]\!]$. U nastavku će se podrazumevati da izraz b može sadržati i logičke i programske promenljive, tako da prethodno prikazan preduslov $x = n$ predstavlja primer bulovskog izraza. Može se pokazati da važe sledeći stavovi[3]:

- $\mathcal{B}[\![b[x \mapsto a]]\!] = \mathcal{B}[x \mapsto \mathcal{A}[\![a]\!]]$ za svako b i a
- $\mathcal{B}[\![b_1 \wedge b_2]\!] = \mathcal{B}[\![b_1]\!] \wedge \mathcal{B}[\![b_2]\!]$ za svako b_1 i b_2
- $\mathcal{B}[\![\neg b]\!] = \neg \mathcal{B}[\![b]\!]$ za svako b

3 Sistem zaključivanja

Skup aksioma i pravila sačinjava sistem zaključivanja koji će biti korišćen kako bi se iskazala parcijalna korektnost tvrđenja. Sve formule u ovom sistemu se zapisuju u već pomenutom obliku Horovih trojki $\{ P \} S \{ Q \}$, gde je S naredba **While**[4] jezika, a P i Q su predikati. Aksiome i pravila su navedena u tabeli 1.

3.1 Aksiome sistema parcijalne korektnosti

Aksioma naredbe dodele [$\mathbf{ass_p}$] se zapisuje kao

$$\{P[x \mapsto \mathcal{A}[\![a]\!]]\} \ x := a \ \{P\}$$

[3]Pomenuti skupovi su pojašnjeni u dodatku A.

[4]Jednostavan imperativni jezik detaljno opisan u [3]. Radi kompletnosti, u dodatku B se takođe nalazi kratak opis u vidu Bakus-Naurove forme.

i podrazumeva da izvršavanje naredbe $x := a$ započinje u stanju s koje zadovoljava $P[x \mapsto \mathcal{A}[\![a]\!]]$, tj. u stanju s u kojem $s[x \mapsto \mathcal{A}[\![a]\!]s]$ zadovoljava P. Ono što je izraženo aksiomom je da ukoliko se izvršavanje izraza $x := a$ zaustavlja (što će u ovom slučaju uvek biti tačno) onda će krajnje stanje zadovoljavati P.

Aksioma preskakanja [$\mathbf{skip_p}$] je oblika

$$\{\ P\ \}\ skip\ \{\ P\ \}$$

i kazuje da ukoliko je P zadovoljeno *pre* nego što je *skip* izvršeno, tada će takođe biti zadovoljeno i u trenutku kad se zaustavi. Nije potrebno posebno naglašavati uslov zaustavljanja s obzirom da *skip* naredba "ne radi ništa".

Aksiome [$\mathbf{ass_p}$] i [$\mathbf{skip_p}$] su zapravo *sheme aksioma* koje čine različite aksiome, zavisno od odabira predikata P. Preostale aksiome daju način zaključivanja o složenim konstruktima, koristeći tvrđenja o jednostavnijim delovima.n

Pravilo slaganja [$\mathbf{comp_p}$] je dato sa

$$\frac{\{\ P\ \}\ S_1\ \{\ Q\ \},\ \{\ Q\ \}\ S_1\ \{\ R\ \}}{\{\ P\ \}\ S_1;\ S_2\ \{\ R\ \}}$$

Njime se kazuje da ukoliko P važi pre izvršavanja $S_1;\ S_2$ i ukoliko se program zaustavlja, onda se može zaključiti da R važi i u krajnjem stanju, ali uzimajući u obzir da postoji predikat Q za koji se može zaključiti da

• ako je program S_1 pokrenut u stanju u kom je P bilo zadovoljeno, i program se zaustavlja, tada će u krajnjem stanju važiti Q, i

• ako je program S_2 pokrenut u stanju u kom je Q bilo zadovoljeno, i program se zaustavlja, tada će u krajnjem stanju važiti R.

Pravilo grananja [$\mathbf{if_p}$] je dato sa

$$\frac{\{\ \mathcal{B}[\![b]\!] \wedge P\ \}\ S_1\ \{\ Q\ \}, \{\ \neg\mathcal{B}[\![b]\!] \wedge P\ \}\ S_2\ \{\ Q\ \}}{\{\ P\ \}\ if\ b\ then\ S_1\ else\ S_2\ \{\ Q\ \}}$$

Ono kaze da ukoliko je $if\ b\ then\ S_1\ else\ S_2$ pokrenuto u stanju u kom je važilo P i program se zaustavlja, tada će Q važiti u finalnom stanju, ukoliko su zadovoljene naredne dve stavke

• ako je program S_1 pokrenut u stanju u kom je P bilo zadovoljeno, i program se zaustavlja, tada će u krajnjem stanju važiti Q, i

• ako je program S_2 pokrenut u stanju u kom je P i $\neg b$ bilo zadovoljeno, i program se zaustavlja, tada će u krajnjem stanju važiti Q.

Pravilo iterativnog tvrđenja [$\mathbf{while_p}$] se zapisuje kao

$$\frac{\{\ \mathcal{B}[\![b]\!] \wedge P\ \}\ S\ \{\ P\ \}}{\{\ P\ \}\ while\ b\ do\ S\ \{\ \neg\mathcal{B}[\![b]\!] \wedge P\ \}}$$

Predikat P se naziva *invarijantom* **while**-petlje. Invarijanta važi i *pre* i *posle* izvršavanja tela petlje (tj. programa S). Pravilo kaže da ukoliko dodatno važi da je b tačno pre svakog izvršavanja tela petlje, onda će $\neg b$ biti tačno nakon što se **while**-petlja zaustavi.

Poslednje pravilo zaključivanja u ovom aksiomatskom sistemu je

$$\frac{\{\ P'\ \}\ S\ \{\ Q'\ \}}{\{\ P\ \}\ S\ \{\ Q\ \}}\ \ ako\ \ P' \Rightarrow P\ \ i\ \ Q' \Rightarrow Q$$

Ono se naziva *pravilom posledice* (eng. *rule of consequence*) i daje mogućnost pojačavanja preduslova P' i slabljenje postuslova Q'[3].

Činjenica da trojka $\{\ P\ \}\ S\ \{\ Q\ \}$ može biti dokazana, se zapisuje kao $\vdash_p \{\ P\ \}\ S\ \{\ Q\ \}$. Dokaz (tj. *stablo izvođenja*) se naziva *prostim* ukoliko je korišćena samo jedna aksioma[3]. Jedinstvenost stabla za datu trojku ne važi u opštem slučaju. Višestrukom primenom $[cons_p]$ se mogu dobiti dokazi različitih dužina[4].

$[ass_p]$	$\{P[x \mapsto \mathcal{A}[\![a]\!]]\}\ x := a\ \{P\}$
$[skip_p]$	$\{\ P\ \}\ skip\ \{\ P\ \}$
$[comp_p]$	$\dfrac{\{\ P\ \}\ S_1\ \{\ Q\ \},\ \{\ Q\ \}\ S_1\ \{\ R\ \}}{\{\ P\ \}\ S_1;\ S_2\ \{\ R\ \}}$
$[if_p]$	$\dfrac{\{\ \mathcal{B}[\![b]\!] \wedge P\ \}\ S_1\ \{\ Q\ \}, \{\ \neg\mathcal{B}[\![b]\!] \wedge P\ \}\ S_2\ \{\ Q\ \}}{\{\ P\ \}\ if\ b\ then\ S_1\ else\ S_2\ \{\ Q\ \}}$
$[while_p]$	$\dfrac{\{\ \mathcal{B}[\![b]\!] \wedge P\ \}\ S\ \{\ P\ \}}{\{\ P\ \}\ while\ b\ do\ S\ \{\neg\mathcal{B}[\![b]\!] \wedge P\ \}}$
$[cons_p]$	$\dfrac{\{\ P'\ \}\ S\ \{\ Q'\ \}}{\{\ P\ \}\ S\ \{\ Q\ \}}$ ako $P' \Rightarrow P$ i $Q' \Rightarrow Q$

Tabela 1: Aksiomatski sistem parcijalne korektnosti

3.2 Osobine aksiomatskog sistema

Sistem je *pouzdan*[5] (eng. *sound*) ukoliko je svako parcijalno svojstvo koje se pomoću njega zaključi takođe i semantički tačno. Za sistem zaključivanja se kaže da je *potpun* (eng. *complete*) ukoliko važi suprotan smer, tj. ukoliko neko svojstvo parcijalne tačnosti važi u određenoj semantici, onda se takođe može pronaći i dokaz u sistemu zaključivanja. Sistem parcijalnog zaključivanja dat u 1 je i *pouzdan* i *potpun*[3].

Za tvrđenje parcijalne korektnosti $\{\ P\ \}\ S\ \{\ Q\ \}$ se kaže da je *valjano* ako i samo ako za svako stanje s važi da

$$ukoliko\ je\ P\ s = \mathbf{tt}\ i\ \langle S, s\rangle \to s'\ za\ neko\ s',\ onda\ je\ Qs' = \mathbf{tt}$$

Valjanost tvrđenja se označava sa

$$\models_p \{\ P\ \}\ S\ \{\ Q\ \}$$

Svojstvo pouzdanosti se onda može izraziti kao

$$\vdash_p \{\ P\ \}\ S\ \{\ Q\ \}\ implicira\ \models_p \{\ P\ \}\ S\ \{\ Q\ \}$$

a svojstvo potpunosti kao

$$\models_p \{\ P\ \}\ S\ \{\ Q\ \}\ implicira\ \vdash_p \{\ P\ \}\ S\ \{\ Q\ \}$$

U [3] se moze pronaći dokaz[6] naredne teoreme

$$\models_p \{\ P\ \}\ S\ \{\ Q\ \}\ ako\ i\ samo\ ako\ \vdash_p \{\ P\ \}\ S\ \{\ Q\ \}$$

[5]Prevod preuzet iz [5].

[6]Sprovodi se indukcijom po dužini stabla izvođenja u datom sistemu.

4 Primeri rezonovanja

Primer 4.1 *Razmotrimo naredbu* **while true do skip**.

Iz aksiome $[skip_p]$ *sledi*

$$\vdash_p \{\ true\ \}\ skip\ \{\ true\ \}$$

Kako važi $(true \wedge true) \Rightarrow true$, *može biti primenjeno* $[cons_p]$ *i to daje*

$$\vdash_p \{\ true \wedge\ true\ \}\ skip\ \{\ true\ \}$$

Iz pravila $[while_p]$ *dobija se*

$$\vdash_p \{\ true\ \}\ while\ true\ do\ skip\ \{\ \neg true \wedge true\ \}$$

Kako je $\neg true \wedge true \Rightarrow true$, *ponovnom primenom* $[cons_p]$ *dobija se*

$$\vdash_p \{\ true\ \}\ while\ true\ do\ skip\ \{\ true\ \}$$

□

Iz prethodnog primera je lako uočiti da se iz $\{\ P\ \}\ S\ \{\ Q\ \}$ ne garantuje da će se S završiti u stanju koje zadovoljava Q. Kada bi to važilo, $\{\ true\ \}\ while\ true\ do\ skip\ \{\ true\ \}$ bi značilo da se program uvek završava, a to nije slučaj[3].

Primer 4.2 *Dokazati da sledeći program računa količnik i ostatak pri deljenju.*

$$\{\ (x \geq 0) \wedge (y > 0)\ \}$$
$$q := 0;\ r := x;\ while\ (r \geq y)\ do\ (q := q + 1;\ r := r - y)$$
$$\{(x = q * y + r) \wedge (0 \leq r) \wedge (r < y)\}$$

Radi čitljivosti, zapis $\{(x = q * y + r) \wedge (0 \leq r) \wedge (r < y)\}$ *će označavati predikat* P *za koji važi*

$$P\ s = (s\ x = (s\ q) * (s\ y) + (s\ r) \wedge (0 \leq (s\ r)) \wedge ((s\ r) < (s\ y)))$$

Najpre se definiše predikat INV koji predstavlja invarijantu **while***-petlje:*

$$INV\ s = (s\ x = (s\ q) * (s\ y) + (s\ r) \wedge (0 \leq (s\ r)))$$

Dva puta se primenjuje aksioma dodele $[\mathbf{ass_p}]$

$$\vdash_p \{\ INV[q \mapsto q + 1]\ \}\ q := q + 1\ \{\ INV\ \}$$

$$\vdash_p \{\ (INV[q \mapsto q + 1])[r \mapsto r - y]\ \}\ r := r - y\ \{\ INV[q \mapsto q + 1]\ \}$$

Na prethodna dva tvrđenja se može primeniti $[comp_p]$

$$\vdash_p \{\ (P[q \mapsto q + 1])[r \mapsto r - y]\ \}\ r := r - y;\ q := q + y\ \{\ P\ \}$$

Koristimo tvrđenje

$$((r \geq y) \wedge INV) \Rightarrow (INV[q \mapsto q + 1])[r \mapsto r - y]$$

Primenom pravila $[cons_p]$ *se dobija*

$$\{\ (r \geq y) \wedge INV\ \}\ r := r - y;\ q := q + 1\ \{\ INV\ \}$$

Sada se može primeniti pravilo $[while_p]$

$$\vdash_p \{\ INV\ \}$$

$$while\ (r \geq y)\ do\ (r := r - y;\ q := q + 1)$$
$$\{\ \neg(r \geq y) \wedge INV\ \}$$

Iz definicije invarijante sledi

$$\neg(r \geq y) \wedge INV \Rightarrow (x = q * y + r) \wedge (0 \leq r) \wedge (r < y)$$

pa se primenom pravila $[cons_p]$ *zaključuje*

$$\vdash_p \{\ INV\ \}$$

$$while\ (r \geq y)\ do\ (r := r - y;\ q := q + 1)$$
$$\{(x = q * y + r) \wedge (0 \leq r) \wedge (r < y)\}$$

Dvostrukom primenom aksiome dodele $[\mathbf{ass_p}]$ *se dobija*

$$\vdash_p \{\ INV[q \mapsto 0]\ \}\ q := 0\ \{\ INV\ \}$$

$$\vdash_p \{\ (INV[q \mapsto 0])[r \mapsto x]\ \}\ r := x\ \{\ INV[q \mapsto 0]\ \}$$

Naredno tvrđenje trivijalno važi

$$(x \geq 0) \wedge (y > 0) \Rightarrow (INV[q \mapsto 0])[r \mapsto x]$$

Kada se dva puta primeni $[cons_p]$*, dobija se*

$$\vdash_p \{\ (x \geq 0) \wedge (y > 0)\ \}\ r := x;\ q := 0\ \{\ INV\ \}$$

$$\vdash_p \{\ (x \geq 0) \wedge (y > 0)\ \}$$
$$r := x;\ q := 0;\ while\ (r \geq y)\ do\ (r := r - y;\ q := q + 1)$$
$$\{(x = q * y + r) \wedge (0 \leq r) \wedge (r < y)\}$$

što je bilo i potrebno dokazati[1]. □

Primer 4.3 *Treba dokazati naredni program za izračunavanje faktorijela.*

$$\{\ x = n\ \}$$

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

Radi čitljivosti, zapis $y = n! \wedge n > 0$ *će označavati predikat* P *za koji važi*

$$P\ s = (s\ y = (s\ n)! \wedge s\ n > 0)$$

Dokaz će biti sproveden u velikom broju koraka. Najpre se definiše predikat INV *koji će predstavljati invarijantu* **while***-petlje:*

$$INV\ s = (s\ x > 0\ implicira\ ((s\ y) * (s\ x)! = (s\ n)!\ i\ n \geq s\ x))$$

Dalje će se razmatrati telo petlje koristeći $[ass_p]$

$$\vdash_p \{\ INV[x \mapsto x - 1]\ \}\ x := x - 1\ \{\ INV\ \}$$

Slično se dobija i

$$\vdash_p \{\ (INV[x \mapsto x-1])[y \mapsto y*x]\ \}\ y := y*x\ \{\ INV[x \mapsto x-1]\ \}$$

Sada se na prethodna dva tvrđenja može primeniti $[comp_p]$

$$\vdash_p \{\ (INV[x \mapsto x-1])[y \mapsto y*x]\ \}\ y := y*x;\ x := x-1\ \{\ INV\ \}$$

Bez dokazivanja, biće iskorišćena naredna tvrdnja

$$(\neg(x=1) \wedge INV) \Rightarrow (INV[x \mapsto x-1])[y \mapsto y*x]$$

Pa se korišćenjem $[cons_p]$ *dobija*

$$\vdash_p \{\ \neg(x=1) \wedge INV\ \}\ y := y*x;\ x := x-1\ \{\ INV\ \}$$

Sada može da se primeni $[while_p]$

$$\vdash_p \{\ INV\ \}$$

$$while\ \neg(x=1)\ do\ (y := y*x;\ x := x-1)$$
$$\{\ \neg(\neg(x=1)) \wedge INV\ \}$$

Iz definicije invarijante petlje sledi

$$\neg(\neg(x=1)) \wedge INV\ \Rightarrow\ y = n!\ \wedge\ n > 0$$

pa se primenom $[cons_p]$ *dobija*

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

Primenom aksiome $[ass_p]$ *na naredbu* $y := 1$ *se dobija*

$$\vdash_p \{\ INV[y \mapsto 1]\ \}\ y := 1\ \{\ INV\ \}$$

Koristeći se definicijom invarijante i prethodnom trojkom, zaključuje se

$$x = n \Rightarrow INV[y \mapsto 1]$$

zajedno sa $[cons_p]$ *se dobija*

$$\vdash_p \{\ x = n\ \}\ y := 1\ \{\ INV\ \}$$

Konačno, može se primeniti $[comp_p]$ *i tada se dobija da*

$$\vdash_p \{\ x = n\ \}$$

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

što je bilo i potrebno dokazati[3, 6]. □

Još neki od primera koji se sreću u literaturi [6, 1] su dokazivanje programa koji izvršava Euklidski algoritam i programa koji računaju Fibonačijev niz. Jednostavniji primeri koji se odnose na osnovne pojmove, mogu se naći u [1].

5 Zaključak

Sistem aksiomatske semantike opisan u ovom seminarskom radu pruža mogućnost dokazivanja svojstava parcijalne korektnosti. Jedna od prednosti takvog formalnog sistema je u mogućnosti preciznog opisivanja postupaka tokom rezonovanja, da postoji nada da će veliki deo posla biti automatizovan u budućnosti. Postoje određeni aspekti koji iz razloga obimnosti nisu pokriveni ovim radom, a to je verifikovanje procedura, verifikovanje koda koji se izvršava konkurentno, proširivanje sistema koji rezonuju parcijalna svojstva, klasa svojstava koje u obzir uzimaju i resurse tokom izvršavanja programa. Jezik **while** predstavlja ilustraciju koncepata aksiomatske semantike i njenih potencijala, te otvara vrata onima koji se prvi put susreću sa aksiomatskom semantikom.

Literatura

[1] Mike Bond. Cse 6341: Foundations of programming languages, November 2014.

[2] Univerzitet u Beogradu Milena Vujošević Janičić, Matematički fakultet. Materijali i slajdovi sa predavanja.

[3] Hanne Riis Nielson and Flemming Nielson. *Semantics with Applications: A Formal Introduction*. John Wiley & Sons, Inc., New York, NY, USA, 1992.

[4] Noam Rinetzky. Advanced course: Program analysis and verification, March 2014.

[5] Rozália Sz. Madarász. Matematička logika. http://sites.dmi.rs/personal/madaraszr/. Online; Novi Sad, novembar 2012.

[6] Mooly Sagiv. Advanced topics in programming languages, April 2012.

A Notacija

Ovakav vid notacije je u potpunosti preuzet iz [3]. Najpre će biti definisani numerički literali (eng. *numerals*), a u nastavku se pretpostavlja da su oni predstavljeni u binarnom sistemu. Njihova apstraktna sintaksa se može predstaviti kao

$$n \ ::= \ 0 \mid 1 \mid n\ 0 \mid n\ 1$$

Za definisanje broja koji je predstavljen numeričkim literalom, uvodi se funkcija

$$\mathcal{N} : \mathbf{Num} \ \rightarrow \ \mathbf{Z}$$

Naziva se *semantička funkcija*, jer definiše semantiku numeričkih literala. Ako je $n \ \in \ \mathbf{Num}$, onda se primena funkcije $\mathcal{N}$ na n zapisuje $\mathcal{N}[\![n]\!]$. Generalno, primena semantičke funkcije na sintaksne entitete se zapisuju u okviru "sintaksnih" zagrada "$[\![$" i "$]\!]$" umesto uobičajenih "(" i ")". Ove zagrade nemaju posebno značenje.

Semantičku funkciju $\mathcal{N}$ se definiše pomoću sledećih semantičkih iskaza:

$$\begin{array}{lcl} \mathcal{N}[\![0]\!] & = & \mathbf{0} \\ \mathcal{N}[\![1]\!] & = & \mathbf{1} \\ \mathcal{N}[\![n\ 0]\!] & = & \mathbf{2} * \mathcal{N}[\![n]\!] \end{array}$$

$$\mathcal{N}[\![n\ 1]\!] \ = \ \mathbf{2} * \mathcal{N}[\![n]\!] + \mathbf{1}$$

Ovde su **1** i **2** elementi iz skupa **Z**. Znakovi ***** i **+** su aritmetičke operacije nad brojevima sa uobičajenim značenjem. Prethodna definicija je primer *kompozicijske* definicije – za svaki mogući način konstrukcije numeričkog literala, navodi se kako se odgovarajući broj dobija od značenja potkonstrukata (eng. *subconstructs*).

Primer A.1 *Broj* $\mathcal{N}[\![101]\!]$ *koji odgovara numeričkom literalu* 101 *se može izračunati na sledeći način:*

$$\begin{aligned}\mathcal{N}[\![101]\!] &= \mathbf{2} * \mathcal{N}[\![10]\!] + \mathbf{1} \\ &= \mathbf{2} * (\mathbf{2} * \mathcal{N}[\![1]\!]) + \mathbf{1} \\ &= \mathbf{2} * (\mathbf{2} \ * \ \mathbf{1}) + \mathbf{1} \\ &= \mathbf{5}\end{aligned}$$

Niska 101 *je dekomponovana prema sintaksi za numeričke literale.*

Funkcija $\mathcal{N}$ je dobro definisana[7] totalna[8] funkcija.

A.1 Semantičke funkcije

Potrebno je uvesti pojam *stanja* – vezuje promenljivu sa njenom trenutnom vrednošću. Stanje se predstavlja kao funkcija koja preslikava promenljive u vrednosti, tj. kao element skupa

$$\mathbf{Stanje} = \mathbf{Prom} \to \mathbf{Z}$$

Svako stanje s određuje neku vrednost, što se zapisuje kao $s\ x$, za svaku promenljivu x iz **Prom**. Dakle, ako je $s\ x \ = \ \mathbf{3}$, onda je vrednost $x + 1$ u stanju s jednako **4**. Ovo je, u stvari, jedna od nekoliko reprezentacija stanja. Jedna od mogućnosti je upotreba tabele, u čijoj se jednoj koloni nalaze promenljive, a u drugoj odgovarajuće vrednosti promenljivih. Stanja se mogu predstavljati i kao "liste" oblika:

$$[\ x \mapsto \mathbf{5},\ y \mapsto \mathbf{7},\ z \mapsto \mathbf{0}\ ]$$

U svakom slučaju mora biti obezbeđeno da tačno jedna vrednost bude vezana za svaku promenljivu. Time što se zahteva da stanje bude funkcija, ovaj uslov je trivijalno zadovoljen.

Za dati artimetički izraz a i stanje s može se odrediti vrednost tog izraza. Zbog toga se značenje aritmetičkih izraza definiše kao totalna funkcija $\mathcal{A}$ koja prima dva argumenta – sintaksnu konstrukciju i stanje. Funkcija $\mathcal{A}$ se opisuje pomoću

$$\mathcal{A} : \mathbf{Aexp} \to (\mathbf{Stanje} \to \mathbf{Z})$$

Definicija funkcije koja opisuje aritmetičke izraze je data u tabeli 2.

Vrednosti bulovskih izraza su istinitosne vrednosti, pa se njihovo značenje definiše na sličan način pomoću totalne funkcije:

$$\mathcal{B} : \mathbf{Bexp} \to (\mathbf{Stanje} \to \mathbf{T})$$

Ovde **T** sadrži istinitosne vrednosti **tt** (za tačno) i **ff** (za netačno). Definicija funkcije koja opisuje bulovske izraze je data u tabeli 3.

[7] Funkcija $f : \ A \to B$ je dobro definisana ako za svako $a \ \in \ A$ postoji jedinstveno $b \ \in \ B$ za koje važi $f(a) \ = \ b$.

[8] Funkcija $f : \ A \to B$ je totalna ako za svaki argument $a \ \in \ A$ postoji tačno jedan element $b \ \in \ B$ tako da važi $f(a) \ = \ b$.

$$
\begin{array}{lcl}
\mathcal{A}[\![n]\!]s & = & \mathcal{N}[\![n]\!] \\
\mathcal{A}[\![x]\!]s & = & s\ x \\
\mathcal{A}[\![a_1 + a_2]\!]s & = & \mathcal{A}[\![a_1]\!]s + \mathcal{A}[\![a_2]\!]s \\
\mathcal{A}[\![a_1 * a_2]\!]s & = & \mathcal{A}[\![a_1]\!]s * \mathcal{A}[\![a_2]\!]s \\
\mathcal{A}[\![a_1 - a_2]\!]s & = & \mathcal{A}[\![a_1]\!]s - \mathcal{A}[\![a_2]\!]s
\end{array}
$$

Tabela 2: Semantika aritmetičkih izraza

$$
\begin{array}{lcl}
\mathcal{B}[\![true]\!]s & = & \mathbf{tt} \\
\mathcal{B}[\![false]\!]s & = & \mathbf{ff} \\
\mathcal{B}[\![a_1 \ = \ a_2]\!]s & = & \begin{cases} \mathbf{tt} & ako\ \mathcal{A}[\![a_1]\!]s \ = \ \mathcal{A}[\![a_2]\!]s \\ \mathbf{ff} & ako\ \mathcal{A}[\![a_1]\!]s \ \neq \ \mathcal{A}[\![a_2]\!]s \end{cases} \\
\mathcal{B}[\![a_1 \ \leq \ a_2]\!]s & = & \begin{cases} \mathbf{tt} & ako\ \mathcal{A}[\![a_1]\!]s \ \leq \ \mathcal{A}[\![a_2]\!]s \\ \mathbf{ff} & ako\ \mathcal{A}[\![a_1]\!]s \ > \ \mathcal{A}[\![a_2]\!]s \end{cases} \\
\mathcal{B}[\![\neg\ b]\!]s & = & \begin{cases} \mathbf{tt} & ako\ \mathcal{B}[\![b]\!]s \ = \ \mathbf{ff} \\ \mathbf{ff} & ako\ \mathcal{B}[\![b]\!]s \ = \ \mathbf{tt} \end{cases} \\
\mathcal{B}[\![\neg\ b]\!]s & = & \begin{cases} \mathbf{tt} & ako\ \mathcal{B}[\![b_1]\!]s \ = \ \mathbf{tt}\ i\ \mathcal{B}[\![b_2]\!]s \ = \ \mathbf{tt} \\ \mathbf{ff} & ako\ \mathcal{B}[\![b_1]\!]s \ = \ \mathbf{ff}\ ili\ \mathcal{B}[\![b_2]\!]s \ = \ \mathbf{ff} \end{cases}
\end{array}
$$

Tabela 3: Semantika bulovskih izraza

Da bi se olakšala čitljivost, uvode se naredna pojednostavljena zapisa

$P_1 \wedge P_2$	za P gde je $P\ s = (P_1\ s)$ i $(P_2\ s)$
$P_1 \vee P_2$	za P gde je $P\ s = (P_1\ s)$ ili $(P_2\ s)$
$\neg P$	za P' gde je $P'\ s = \neg(P\ s)$
$P[x \mapsto \mathcal{A}[\![a]\!]]$	za P' gde je $P'\ s = P\ (s[x \mapsto \mathcal{A}[\![a]\!]\ s])$
$P_1 \Rightarrow P_2$	za $\forall s \in \textbf{Stanja}:\ P_1\ s$ implicira $P_2\ s$

A.2 Supstitucija

Potrebno je definisati **supstituciju**, tj. zamenu svakog pojavljivanja neke promenljive y u aritmetičkom izrazu a nekim drugim aritmetičkim izrazom a_0. Koristi se zapis $a[y \mapsto a_0]$ za označavanje tako dobijenog aritmetičkog izraza. Upotrebljava se i notacija za supstituciju (ili ažuriranje) kod stanja. Definiše se $s[y \mapsto v]$ kao stanje koje je isto kao s, osim što vrednost koja je vezana za y je v.

Supstitucija u kontekstu funkcija definisana je na sledeći način. Neka je $f : X \to Y$, $x \in X$ i $y \in Y$, onda je $f[x \mapsto y] : X \to Y$ funkcija definisana kao

$$
f[x \mapsto y]\ x' = \begin{cases} y, & \text{ako je } x = x' \\ f\ x', & \text{inače} \end{cases}
$$

A.3 Tranzicione relacije

Razmatraju se dva različita pristupa operacionoj semantici:

- *Prirodna semantika* (eng. *natural semantics*), čija svrha je opisivanje kako se došlo do sveukupnog rezultata izvršavanja.
- *Strukturna operaciona semantika* (eng. *structural operational semantics*), koja opisuje kako se odigravaju pojedinačni koraci izračunavanja[2].

Zapis $\langle S,\ s\rangle$ označava da se naredba S izvršava iz stanja s, a kada se zapis s nalazi van "$\langle$" i "$\rangle$" zagrada, onda predstavlja završno stanje. U prirodnoj semantici se posmatra veza između početnog i završnog stanja. Ta tranziciona relacija se zapisuje pomoću

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

U strukturnoj operacionoj semantici naglasak je na pojedinačnim koracima izvršavanja. Tranziciona relacija se zapisuje

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

i tumači kao prvi korak izvršavanja naredbe S u stanju s koje vodi do stanja γ. Stanje γ može biti $\langle S',\ s'\rangle$ ili s'. U prvom slučaju, izvršavanje S nije završeno i izračunavanje koje je preostalo predstavljeno je konfiguracijom $\langle S',\ s'\rangle$. U drugom slučaju, izvršavanje S se završilo i završno stanje je s'.

B While jezik

While je jednostavan imperativni programski jezik definisan u [3]. Meta-promenljive i kategorije koje se koriste za opis jezika su:

n uzima vrednosti iz opsega numeričkih literala, **Num**

x uzima vrednosti iz opsega promenljivih, **Var**

a predstavlja aritmetički izraz iz **Aexp**

b predstavlja bulovski izraz iz **Bexp**

S predstavlja naredbu iz **Stm**

Numerali mogu biti niske cifara, a promenljive niske slova ili cifara koje započinju slovom. Struktura ostalih konstrukcija je sledeća:

$$\begin{aligned} a &::= n \mid x \mid a_1 + a_2 \mid a_1 * a_2 \mid a_1 - a_2 \\ b &::= true \mid false \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2 \\ S &::= x := a \mid skip \mid S_1;\ S_2 \mid if\ b\ then\ S_1\ else\ S_2 \mid while\ b\ do\ S \end{aligned}$$

Uloga naredbe u **While** jeziku je promena stanja. Na primer, ako je x vezano za **3** u stanju s i izvrši se naredba $x := x + 1$, onda se dobija novo stanje u kojem je x vezano za **4**.