Strukturna semantika kroz primere i osnovna tvrđenja

Seminarski rad u okviru kursa
Verifikacija softvera
Matematički fakultet

Jana Milutinović, 1109/2016
jana_milutinovic@yahoo.com

11. decembar 2018

Sažetak

U ovom tekstu je ukratko je opisana i objašnjena kroz primere strukturna semantika programskih jezika. Semantika programskih jezika je osnovna i uvek aktuelna tema savremenog računarstva i njeno razumevanje i konstantno izučavanje je neophodno. Cilj rada je da prikaže osnovne principe strukturne semantike i uvede čitaoca u ovu kompleksnu i apstraktnu granu izučavanja programskih jezika. Kroz osnovne primere, čitalac će se konceptualno upoznati sa osnovnim načelima rada.

Sadržaj

1 Uvod

U teoriji jezika, bilo prirodnog ili programskog, neophodno je definisati osnovna pravila za konstruisanje izraza, ali i pridružiti im značenje. Ove dve radnje opisuju pojmovi sintaksa i semantika, koji su osnov za dalje razumevanje i izucavanje prirode jezika. Činjenica da je izraz pravilno struktuiran, tj sintaksno ispravan, nije nam dovoljna potvrda da je izraz korektan u svom jeziku. Ovo svojstvo nam omogućuje **semantika**, koja pridruživanje značenja sintaksno ispravnom izrazu.

Semantika može biti definisana neformalno i formalno. Neformalna podrazumeva pismeno ili usmeno opisivanje izraza, što u priordnom jeziku može biti korisno, dok u programskom nema puno smisla. Formalna semantika podrazumeva preciznu definiciju programskog jezika, zarad potpune korektnosti i minimizovanja grešaka. Formalna semantika deli se na *denotacionu, operacionu i aksiomatsku* semantiku.

Denotaciona semantika definiše značenje izraza prevođenjem istih u neki drugi programski jezik, čija je semantika poznata, najčešće u neki matematički formalizam. Ispravna sintaksna recenica predstavlja objekat na koji se mogu primeniti funkcije koje taj objekat preslikavaju u matematički objekat koji definiše značenje. [3]

Aksiomatska semantika zasniva se na matematičkoj logici. U primeru Horove logike definišu se aksiomi i pravila izvodenja za sve konstrukte jednostavnog imperativnog programskog jezika koji objašnjavaju kako izvršavanje dela koda menja stanje izračunavanja.

Operaciona semantika opisuje način na koji se vrši izračunavanje.

2 Operaciona semantika

Operaciona semantika definiše kako se vrši izračunavanje kroz matematičku reprezentaciju, tj opisuje tok izvršavanja koraka za dobijeni ulaz. Akcenat je na tome *kako* se vrši prelazak iz stanja u stanje kroz program. Opisani postupci mogu se definisati na 2 načina - neformalno, opisujuci elemente i procese izvršavanja ali fokusirajući se na izlaz, rezultat i ovo je tema **Prirodne operacione semantike** . Drugi način je strukturna operaciona semantika o čemu će biti reči u nastavku.

3 Strukturna operaciona semantika

Cilj Strukturne operacione semantike je da opise tok izvršavanja programa korak po korak. Strukturnu semantiku ćemo objasniti preko imperativnih jezika, zbog postojanja implicitnog stanja, koje se menja izvršavanjem naredbi programa. Imperativni programi tranzicijom prelaze iz jednog stanja u drugo, tj izvršavanjem naredbe nad trenutnim stanjem. Uvešćemo pojam *tranzicione relacije* koja će definisati tok izvršavanja, tj. relaciju prelaska između stanja [2]:

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

Gde je S naredba koja se izvršava od početnog stanja s i prelazi u završno ili međustanje γ. Postoje 2 oblika za γ :

1. $\gamma = \langle S', s'\rangle$
   Izvršavanje programa S sa ulaznim stanjem s nije završeno, i ostatak izračunavanja će biti izraženo srednjom konfiguracijom $\langle S', s'\rangle$

2. $\gamma = s'$
   Izvršavanje programa S sa ulaznim stanjem s se završilo sa završnim stanjem s'.

Ukoliko se ne postoji γ za koji važi $\gamma \Longrightarrow \langle S', s' \rangle$, kažemo da je konfiguracija **zaglavljena**.

Definisaćemo aksiome strukturne operacione semantike za **While** jezik:

$$[\text{ass}_{\text{sos}}] \quad \langle x := a,\ s\rangle \Rightarrow s[x \mapsto \mathcal{A}[\![a]\!]s]$$

$$[\text{skip}_{\text{sos}}] \quad \langle \texttt{skip},\ s\rangle \Rightarrow s$$

$$[\text{comp}^1_{\text{sos}}] \quad \frac{\langle S_1,\ s\rangle \Rightarrow \langle S_1',\ s'\rangle}{\langle S_1;S_2,\ s\rangle \Rightarrow \langle S_1';S_2,\ s'\rangle}$$

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

$$[\text{if}^{\text{tt}}_{\text{sos}}] \quad \langle \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2,\ s\rangle \Rightarrow \langle S_1,\ s\rangle \text{ if } \mathcal{B}[\![b]\!]s = \texttt{tt}$$

$$[\text{if}^{\text{ff}}_{\text{sos}}] \quad \langle \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2,\ s\rangle \Rightarrow \langle S_2,\ s\rangle \text{ if } \mathcal{B}[\![b]\!]s = \texttt{ff}$$

$$[\text{while}_{\text{sos}}] \quad \langle \texttt{while } b \texttt{ do } S,\ s\rangle \Rightarrow \langle \texttt{if } b \texttt{ then } (S; \texttt{while } b \texttt{ do } S) \texttt{ else skip},\ s\rangle$$

Slika 1: Strukturna semantika za While [4]

Aksiom $[ass_{sos}]$ prikazan na slici 1 opisuje da će izvršavanje naredbe $x := a$ iz početnog stanja s, takođe dovesti do stanja s, ali u kome je vrednost x zamenjena vrednošću izraza $\mathcal{A}[[a]s]$.

Primenom aksioma $[skip_{sos}]$ ne dolazi do promene stanja, tj. ne menja se trenutno stanje.

Askiomi $[\text{comp}^1_{sos}]$ i $[\text{comp}^2_{sos}]$ opisuju tok izvršavanja 2 programa S_1 i S_2 u zavisnosti od pomenuta 2 moguća ishoda $\langle S, s\rangle$:

- Ukoliko je rezultat izvršavanja prvog koraka $\langle S, s\rangle$ međustanje $\langle S_1', s'\rangle$, naredna kofiguracija je $\langle S_1'; S_2, s'\rangle$. Ovo je opisano pravilom $[\text{comp}^1_{sos}]$.
- Ukoliko je rezultat izvršavanja programa S_1 iz početnog stanja s, završno stanje s', tada je rec o pravilu $[\text{comp}^2_{sos}]$ i sledeća kofiguracija je $\langle S_2, s'\rangle$. To znači da je program S_1 izvršen i da je spremno izvršavanje programa S_2.

Aksiomi $[\text{iff}^{tt}_{sos}]$ i $[\text{iff}^{ff}_{sos}]$ opisuju naredbe grananja. Prvi korak je ispitivanje istinitosne vrednosti uslova, a drugi odnosi se na određivanje grane u koju treba ući.

Aksiom $[While_{sos}]$ opisuje while petlju. Prvi korak je određivanje uslova izlaska iz petlje, a drugi je ispitivanje da li je taj uslov ispunjen i da li je moguć nastavak izvršavanja naredbi petlje.

Nakon definisanja osnovnih pravila strukturne semantike, njihova primena će biti iskazana kroz nekoliko primera.

4 Strukturna semantika kroz primere

U ovom poglavlju će biti pokušano da se na što jednostavniji način objasni primena Strukturne operacione semantike na praktične probleme u programima imperativne paradigme, koristeći semantiku While jezika prethodno opisanu.

Primer 4.1 *Razmotrimo primer* $(z := x; x := y); y := z$ *[1].*
Neka je početno stanje s_0 *takvo da mapira vrednosti svih promenljivih osim* x *i* y *na 0, a vrednosti* x *i* y *zadaje proizvoljno,* s_0 ***x=4*** *i* s_0 ***y=1****. Zapisujemo tranzicionu relaciju na osnovu zadatog skupa naredbi* S *i početnog stanja* s_0*.*

$$\begin{aligned}&\langle (z := x; x := y); y := z, s_0\rangle\\ &\Longrightarrow \langle x := y; y := z, s_0[z \mapsto 4]\rangle\\ &\Longrightarrow \langle y := z, (s_0[z \mapsto 4])[x \mapsto 1]\rangle\\ &\Longrightarrow \langle ((s_0[z \mapsto 4])[x \mapsto 1])[y \mapsto 4]\rangle\end{aligned}$$

Za svaki korak možemo konstruisati derivaciono stablo koje opisuje način na koji se dolazi do rešenja.

Korak 1*:* $\langle (z := x; x := y); y := z, s_0\rangle \Longrightarrow \langle x := y; y := z, s_0[z \mapsto 4]\rangle$*.*
Derivaciono stablo:

$$\dfrac{\dfrac{}{\langle z := x, s_0\rangle \Rightarrow s_0[z \mapsto 4]}\,[ass_{sos}]}{\langle z := x; x := y, s_0\rangle \Rightarrow \langle x := y, s_0[z \mapsto 4]\rangle}\,[comp^2_{sos}]}{\langle (z := x; x := y); y := z, s_0\rangle \Rightarrow \langle x := y; y := z, s_0[z \mapsto 4]\rangle}\,[comp^1_{sos}]$$

Stablo za ovaj korak kreirano je primenom aksioma $[comp^1_{sos}]$*, zatim* $[comp^2_{sos}]$ *i na kraju* $[ass_{sos}]$*.*

Korak 2*:* $\langle x := y; y := z, s_0[z \mapsto 4]\rangle \Longrightarrow \langle y := z, (s_0[z \mapsto 4])[x \mapsto 1]\rangle$ *.*
Derivaciono stablo:

$$\dfrac{\dfrac{}{\langle x := y, s_0[z \mapsto 4]\rangle \Rightarrow \langle (s_0[z \mapsto 4])[x \mapsto 1]\rangle}\,[ass_{sos}]}{\langle x := y; y := z, s_0[z \mapsto 4]\rangle \Longrightarrow \langle y := z, (s_0[z \mapsto 4])[x \mapsto 1]}\,[comp^2_{sos}]$$

Stablo za ovaj korak kreirano je samo primenom aksioma $[comp^2_{sos}]$ *i* $[ass_{sos}]$*.*

Korak 3*:* $\langle y := z, (s_0[z \mapsto 4])[x \mapsto 1]\rangle \Longrightarrow \langle ((s_0[z \mapsto 4])[x \mapsto 1])[y \mapsto 4]\rangle$
Derivaciono stablo za ovaj korak kreirano je samo primenom aksioma $[ass_{sos}]$*.*

Pokazali smo na primeru na koji način se vrši prepoznavanje i primena najjednostavnijih aksioma. U narednom primeru ćemo se fokusirati na analiziranje komplikovanijih aksioma While jezika

Primer 4.2 *Pretpostavimo da je $x = 3$ i zadata je početna konfiguracija:*
$\langle\ y := 1;\ while\ \neg(x = 1)\ do\ (y := y * x; x := x - 1),\ s\ \rangle$

Nakon primene aksioma $[comp^2_{sos}]$ i$[ass_{sos}]$ kao u prethodnom koraku dobijamo sledece derivaciono stablo:

$$\dfrac{\dfrac{}{\langle y := 1, s\rangle \Rightarrow s[y \mapsto 1]}\,[ass_{sos}]}{\langle\ y := 1;\ \text{while}\ \neg(x = 1)\ \text{do}\ (y := y * x; x := x - 1),\ \text{s}\ \rangle \Rightarrow \langle\ \text{while}\ \neg(x = 1)\ \text{do}\ (y := y * x; x := x - 1), s[y \mapsto 1]\rangle}\,[comp^2_{sos}]$$

Sledeći korak je da prezapišemo while petlju kroz uslovnu naredbu i da primenimo aksiom $[While_{sos}]$:
$\langle$ if $\neg(x = 1)$ then $((y := y; x := x1);$
while $\neg(x = 1)$ do $(y := y; x := x1))$
else skip, $s[y \mapsto 1]\rangle$

Naredni korak će izvršiti ispitivanje istinitosne vrednosti i primenom aksioma $[\text{iff}^{tt}_{sos}]$ dobijamo konfiguraciju:

$\langle\ (y :=; x := x1);$ while $\neg(x = 1)$ do $(y := y; x := x1), s[y \mapsto 1]\ \rangle$

Daljom primenom pravila $[comp^1_{sos}]$, $[comp^2_{sos}]$ i $[ass_{sos}]$ dobija se konfiguracija: $\langle\ x := x1;$ while $\neg(x = 1)$ do $(y := y; x := x1), s[y \mapsto 3]\ \rangle$

Kreiranjem derivacionog stabla kao i ranije, primenom aksioma $[comp^2_{sos}]$ i $[ass_{sos}]$ dobijamo konfiguraciju:

$\langle$ while $\neg(x = 1)$ do $(y := y; x := x1), s[y \mapsto 3][x \mapsto 2]\ \rangle$

Primenom pravila, u jednom trenutku ćemo dostići završno stanje $s[y \mapsto 6][x \mapsto 1]$.

5 Zaključak

Strukturna semantika programskih jezika pogodna je za detaljno razumevanje toka izvršavanja proceduralnih programa, zato što praćenjem osnovih aksioma program možemo razložiti naredbu po naredbu i analizirati i donositi zaključke o krajnjem rezultatu. Ovaj tip semantike, koja se još naziva i semantika malih koraka (eng. *small step semantics*), zasniva se na matematičkim pravilima kojima se vrši prevođenje iz jednog u drugo stanje programa, ulazeći duboko u strukturu naredbi. Ovakve tehnike omogućavaju programeru formalni uvid u tok izvršavanja programa, korak po korak, čime obezbeđujemo sigurnost i tačnost u rezultat koji dobijamo.

Ovaj rad je uvod u formalnu semantiku programskih jezika, sa osvrtom na operacionu strukturnu semantiku, čija je logika prikazana kroz par jednostavnih primera. Postavljeni cilj rada da prikaže osnovne principe strukturne semantike kroz primere mogao bi se dopuniti definisanjem aksioma specificnog jezika i prikazati nove primere i implementaciju.

Literatura

[1] Operational Semantics. on-line at:http://www.doc.ic.ac.uk/ svb/AssuredSoftware/notes.pdf.

[2] Semantika programskih jezika. on-line at:http://www.programskijezici.matf.bg.ac.rs/ppR/2017/predavanja/12/SemantikaProgramskihJezika.pdf.

[3] Milena Vujošević Janičić. *Dizajn programskih jezika*. Beograd, 2016.

[4] Hanne Riis Nielson and Flemming Nielson. *SEMANTICS WITH APPLICATIONS A Formal Introduction*. John Wiley Son, 1999.