Denotaciona semantika kroz primere

Seminarski rad u okviru kursa
Verifikacija softvera
Matematički fakultet

Kristina Stanojević, 1084/2017
stanojevic.kristina@gmail.com

16. decembar 2018

Sažetak

U ovom tekstu biće obrađena semantika programskog jezika. Fokusiraćemo se na *denotacionoj semantici* koja predstavlja formalno opisivanje semantike. Uz nju se takođe navode operaciona i aksiomska semantika jezika. Denotaciona semantika definiše značenje programskog jezika svođenjem na neki drugi jezik za koji se pretpostavlja da je semantika poznata. To se najčešće radi svođenjem na matematički jezik tako što se svaka definicija tretira kao objekat koji se funkcijom preslikava u odgovarajući matematički objekat čije je značenje već poznato, na primer preslikava se u broj, istinosnu vrednost, funkciju... Time se zadatoj definiciji pridaje značenje. Denotacionom semantikom je olakšano formalno dokazivanje semantičkih svojstva programa i njome se apstrahuje izvršavanje programa. [4]

Sadržaj

1 Uvod

Reč semantika potiče od grčke reči *semantikos*, u prevodu *značajan, koji daje značenje*, i predstavlja pridruživanje značenja jeziku, programskom kodu i slično [5]. Dok je sintaksom određeno kako se simboli kombinuju u celine, semantika određuje njihovo značenje pa ju je teže definisati. Neformalna semantika se zadaje kako bi bilo jasno izvršavanje programa pre njegovog pokretanja, uglavnom je opisano rečima (dakle predstavlja opisno ponašanje napisanog koda), dok formalna semantika omogućava formalno opisivanje ponašanje programa pomoću koje je kasnije moguće ispitati uslove korektnosti rada programa. Formalna semantika može biti operaciona, denotaciona i aksiomska. Operaciona semantika opisuje kako se neko izračunavanje izvršava (na primer $x := y$ znači sledeće: odredi koja je vrednost promenljive y i dodeli je promenljivoj x), a aksiomska se zasniva na matematičkoj logici i proveri ispravnosti programa. Pažnju ćemo posvetiti denotacionoj semantici.

1.1 Primer

Kako bi ideja bila unapred otprilike jasna, za početak daćemo primer denotacione semantike jednostavnog jezika sa samo pozitivnim brojevima. Taj primer je dat na slici 1. Jedina dodatna stvar koju moramo uvesti jesu dve pomoćne funkcije *plus* i *times* definisane kao:
$plus : Number\ x\ Number \rightarrow Number$
$times : Number\ x\ Number \rightarrow Number$

Bez zadržavanja na ovom primeru, kroz primer koji sledi objasnićemo detalje. Čitalac se poziva da se nakon iščitavanja teksta vrati na ovaj primer i proba sam da ga protumači.

Syntactic Domains

N : Numeral -- nonnegative numerals
D : Digit -- decimal digits

Abstract Production Rules

Numeral ::= Digit | Numeral Digit
Digit ::= **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**

Semantic Domain

Number = { 0, 1, 2, 3, 4, ... } -- natural numbers

Semantic Functions

value : Numeral → Number
digit : Digit → Number

Semantic Equations

value ⟦N D⟧ = *plus* (*times*(10, *value* ⟦N⟧), *digit* ⟦D⟧)
value ⟦D⟧ = *digit* ⟦D⟧

digit ⟦**0**⟧ = 0	*digit* ⟦**3**⟧ = 3	*digit* ⟦**6**⟧ = 6	*digit* ⟦**8**⟧ = 8
digit ⟦**1**⟧ = 1	*digit* ⟦**4**⟧ = 4	*digit* ⟦**7**⟧ = 7	*digit* ⟦**9**⟧ = 9
digit ⟦**2**⟧ = 2	*digit* ⟦**5**⟧ = 5		

Slika 1: Osnovni primer denotacione semantike

2 Ukratko o denotacionoj semantici

Nekada poznata kao matematička semantika, denotaciona semantika dodeljuje značenje simbolima programskog jezika imajući u vidu matematičke objekte čija su značenja poznata. Razvijao ju je Kristofer Strejči (eng. *Christopher Strachey*) sa grupom na Oksfordu (eng. *Oxford*) od sredine 60-ih godina, gde je od 1969. učestvovao i Dejna Skot (eng. *Dana Scott*) [3]. Ideja je da se program posmatra kao skup matematičkih funkcija gde se pokretanjem nekog koda dobija efekat prelaska iz jednog u drugo stanje programa, i to se može predstaviti funkcijom: $Stanje \rightarrow Stanje$. Kako program predstavlja skup izraza odvojenih simbolom ';', efekat celog programa biće kompozicija efekata svih individualnih izraza odvojenih tim simbolom koji program sačinjavaju. Kao što vidimo osnovni korak je definisanje semantičke funkcije za svaki sintaksički objekat koja mapira dati objekat u matematički objekat.

Na primer:
$Stanje[[z := x; x := y; y := z]] =$
$Stanje[[y := z]] \circ Stanje[[x := y]] \circ Stanje[[z := x]]$

Za konkretne vrednosti sledi:
$Stanje[[z := x; x := y; y := z]]([x \rightarrow 5, y \rightarrow 7, z \rightarrow 0]) =$
$Stanje[[y := z]] \circ Stanje[[x := y]] \circ Stanje[[z := x]]$
$([x \rightarrow 5, y \rightarrow 7, z \rightarrow 0]) =$
$Stanje[[y := z]] \circ Stanje[[x := y]]([x \rightarrow 5, y \rightarrow 7, z \rightarrow 5]) =$
$Stanje[[y := z]]([x \rightarrow 7, y \rightarrow 7, z \rightarrow 5]) = [x \rightarrow 7, y \rightarrow 5, z \rightarrow 5]$

Na slici 2 dat je primer semantike za izraze sa istinostnim vrednostima *true* i *false*.

3 Primer jezika While

Neka imamo dati jezik **While** čiju ćemo semantiku razmatrati. Najpre moramo da odredimo sintaksu tog jezika tj. njegovu gramatiku. Koristićemo BNF, odnosno Bakus-Naurovu formu (eng. *Backus–Naur form*) koja se inače koristi za predstavljanje kontekstno slobodnih gramatika odnosno za formalno opisivanje jezika. Neke kategorije kao i oznake koje ćemo koristiti su sledeće:

- n kao oznaka broja (Num),
- x kao oznaka promenljivih (Var),
- a kao oznaka aritmetičkih izraza (Aexp),
- b kao oznaka istinosnih izraza (Bexp),
- S kao oznaka nekih ostalih izraza (Stm)

Brojeve ćemo predstavljati kao stringove cifara, promenljive kao stringove slova. Ostale konstrukcije definišemo na sledeći način:

- $a = n \mid x \mid a_1 + a_2 \mid a_1 * a_2 \mid a_1 - a_2$
- $b = true \mid false \mid a_1 = a_2 \mid a_1 <= a_2 \mid \ b \mid b_1 \wedge b_2$
- $S = x := a \mid \text{skip} \mid S_1; S_2 \mid \text{if } b \text{ then } S_1 \text{ else } S_2 \mid \text{while } b \text{ do } S$

Iz ove apstraktne sintakse vidimo da aritmetički izrazi mogu da imaju samo ovih pet formi, gde jedna pokazuje da aritmetički izraz može biti samo cifra, ili samo promenljiva ili neka od datih operacija dva elemenata koji predstavljaju neke aritmetičke izraze. Slično tumačimo i ostale zapise [2].

$$\mathcal{B} \quad : \quad \textit{Boolean Expressions} \to \textit{State} \to \mathbb{T}$$

where $\mathbb{T} = \{\mathbf{tt}, \mathbf{ff}\}$, the set of (semantic) truth values – as follows:

$$\begin{array}{rcl}
\mathcal{B}[\![\mathtt{true}]\!]\, s & = & \mathbf{tt} \\
\mathcal{B}[\![\mathtt{false}]\!]\, s & = & \mathbf{ff} \\
\mathcal{B}[\![a_1 = a_2]\!]\, s & = & \mathbf{tt}, \text{ if } \mathcal{A}[\![a_1]\!]\, s = \mathcal{A}[\![a_2]\!]\, s \\
 & = & \mathbf{ff}, \text{ if } \mathcal{A}[\![a_1]\!]\, s \neq \mathcal{A}[\![a_2]\!]\, s \\
\mathcal{B}[\![a_1 \leq a_2]\!]\, s & = & \mathbf{tt}, \text{ if } \mathcal{A}[\![a_1]\!]\, s \leq \mathcal{A}[\![a_2]\!]\, s \\
 & = & \mathbf{ff}, \text{ if } \mathcal{A}[\![a_1]\!]\, s > \mathcal{A}[\![a_2]\!]\, s \\
\mathcal{B}[\![\neg b]\!]\, s & = & \mathbf{tt}, \text{ if } \mathcal{B}[\![b]\!]\, s = \mathbf{ff} \\
 & = & \mathbf{ff}, \text{ if } \mathcal{B}[\![b]\!]\, s = \mathbf{tt} \\
\mathcal{B}[\![b_1 \ \& \ b_2]\!]\, s & = & \mathbf{tt}, \text{ if } \mathcal{B}[\![b_1]\!]\, s = \mathbf{tt} \ \& \ \mathcal{B}[\![b_2]\!]\, s = \mathbf{tt} \\
 & = & \mathbf{ff}, \text{ otherwise}
\end{array}$$

Alternatively, since $\mathcal{A}[\![a_1]\!]\, s = \mathcal{A}[\![a_2]\!]\, s$ lives in the semantic world, and has the same value as $\mathbf{tt}$ if both values are equal, we can define the semantics like this:

$$\begin{array}{rcl}
\mathcal{B}[\![\mathtt{true}]\!]\, s & = & \mathbf{tt} \\
\mathcal{B}[\![\mathtt{false}]\!]\, s & = & \mathbf{ff} \\
\mathcal{B}[\![a_1 = a_2]\!]\, s & = & (\mathcal{A}[\![a_1]\!]\, s = \mathcal{A}[\![a_2]\!]\, s) \\
\mathcal{B}[\![a_1 \leq a_2]\!]\, s & = & (\mathcal{A}[\![a_1]\!]\, s \leq \mathcal{A}[\![a_2]\!]\, s) \\
\mathcal{B}[\![\neg b]\!]\, s & = & \neg\, (\mathcal{B}[\![b]\!]\, s) \\
\mathcal{B}[\![b_1 \ \& \ b_2]\!]\, s & = & (\mathcal{B}[\![b_1]\!]\, s \ \& \ \mathcal{B}[\![b_2]\!]\, s)
\end{array}$$

Slika 2: Denotaciona semantika boolean izraza

Semantiku jezika definisaćemo semantičkom funkcijom koja će svakoj gore navedenoj sintaksičkoj kategoriji da dodeli neko značenje. Za početak najprostije je posmatrati brojeve. Ako na primer imamo brojeve iz binarnog sistema (dakle apstraktna sintaksa će biti $n = 0 \mid 1 \mid n0 \mid n1$), onda ćemo definisati funkciju za predstavljanje brojeva binarnim ciframa kao $\mathcal{N} : Num \to Z$ koja svakom broju $n \in Num$ dodeljuje jedinstvenu vrednost. Zapis $\mathcal{N}[[n]]$ označava primenjivanje semantičke funkcije $\mathcal{N}$ na sintatički entitet n. Funkciju ćemo definisati na sledeći način:

- $\mathcal{N}[[0]] = 0$
- $\mathcal{N}[[1]] = 1$
- $\mathcal{N}[[n0]] = 2 * \mathcal{N}[[n]]$
- $\mathcal{N}[[n1]] = 2 * \mathcal{N}[[n]] + 1$

Ovo je ujedno primer kompozitivnosti, jer smo u delu gramatike broj predstavili preko njegove podkonstrukcije. Kompozitivnost predstavlja važnu osobinu denotacione semantike jer se njome semantika jedne celine predstavlja semantikama poddelova te celine. To možemo da primetimo na primeru:

$\mathcal{N}[[011]] = 2{*}\mathcal{N}[[01]]+1 = 2{*}(2{*}\mathcal{N}[[0]]+1)+1 = 2{*}(2{*}0+1)+1 = 3.$

Takođe koncept izraza koji dodeljuje vrednost promenljivoj možemo prikazati kao $State = Var \to Z$. Dodeljivanje neke vrednosti svakoj promenljivoj $x \in Var$ zapisivaćemo kao sx što označava stanje promenljive x, tj. njenu vrednost u stanju s. Dalje, ako nam je dat aritmetički izraz a i stanje s, možemo definisati vrednost aritmetičkog izraza.

Funkcija preslikavanja koja dodeljuje značenje aritmetičkom izrazu biće $\mathcal{A} : Aexp \to (State \to Z)$. Dakle semantiku jezika proširujemo definisanjem ove funkcije:

- $\mathcal{A}[[n]]s = \mathcal{N}[[n]]$
- $\mathcal{A}[[x]]s = sx$
- $\mathcal{A}[[a_1 + a_2]]s = \mathcal{A}[[a_1]]s + \mathcal{A}[[a_2]]s$
- $\mathcal{A}[[a_1 * a_2]]s = \mathcal{A}[[a_1]]s * \mathcal{A}[[a_2]]s$
- $\mathcal{A}[[a_1 - a_2]]s = \mathcal{A}[[a_1]]s - \mathcal{A}[[a_2]]s$

Oznaka $\mathcal{A}[[a]]s$ predstavlja definisanje vrednosti bilo kog aritmetičkog izraza a u stanju s. Vidimo da je vrednost izraza $a_1 + a_2$ u s zbir vrednosti a_1 u s i a_2 u s. Na primer za $x = 5$, $\mathcal{A}[[x - 2]]s = \mathcal{A}[[x]]s - \mathcal{A}[[2]]s = (sx) - \mathcal{N}[[2]] = 5 - 2 = 3$.
Dakle sx predstavlja stanje promenljive x tj. njenu vrednost 5. Takođe 2 unutar zagrada '[[' i ']]' predstavlja sintaksičku oznaku, dok 2 u pretposlednjoj jednakosti predstavlja broj 2 (ima vrednost). Slično je na slici 3 prikazana funkcija za dodelu vrednosti istinosnim izrazima $\mathcal{B} : Bexp \to (State \to T)$. Dakle tehnika definisanja semantike je definisanje preko kompozicijske funkcije, postojanjem semantike za svaki osnovni i svaki kompozitni element sintakse.

$$
\begin{aligned}
\mathcal{B}[\![\texttt{true}]\!]s &= \mathbf{tt} \\
\mathcal{B}[\![\texttt{false}]\!]s &= \mathbf{ff} \\
\mathcal{B}[\![a_1 = a_2]\!]s &= \begin{cases} \mathbf{tt} & \text{if } \mathcal{A}[\![a_1]\!]s = \mathcal{A}[\![a_2]\!]s \\ \mathbf{ff} & \text{if } \mathcal{A}[\![a_1]\!]s \neq \mathcal{A}[\![a_2]\!]s \end{cases} \\
\mathcal{B}[\![a_1 \leq a_2]\!]s &= \begin{cases} \mathbf{tt} & \text{if } \mathcal{A}[\![a_1]\!]s \leq \mathcal{A}[\![a_2]\!]s \\ \mathbf{ff} & \text{if } \mathcal{A}[\![a_1]\!]s > \mathcal{A}[\![a_2]\!]s \end{cases} \\
\mathcal{B}[\![\neg\, b]\!]s &= \begin{cases} \mathbf{tt} & \text{if } \mathcal{B}[\![b]\!]s = \mathbf{ff} \\ \mathbf{ff} & \text{if } \mathcal{B}[\![b]\!]s = \mathbf{tt} \end{cases} \\
\mathcal{B}[\![b_1 \wedge b_2]\!]s &= \begin{cases} \mathbf{tt} & \text{if } \mathcal{B}[\![b_1]\!]s = \mathbf{tt} \text{ and } \mathcal{B}[\![b_2]\!]s = \mathbf{tt} \\ \mathbf{ff} & \text{if } \mathcal{B}[\![b_1]\!]s = \mathbf{ff} \text{ or } \mathcal{B}[\![b_2]\!]s = \mathbf{ff} \end{cases}
\end{aligned}
$$

Slika 3: Funkcija istinosnih izraza

4 Denotaciona semantika kroz jezik While

Kao što smo već spomenuli, denotaciona semantika razmatra samo efekat izvršavanja programa, tj. zanima nas povezanost između početnog i krajnjeg stanja programa, a nju određujemo pomoću sintaksičke funkcije mapiranja. Takođe smo već spomenuli da kompozitivnost ovde dolazi do izražaja, jer su te funkcije kompozitno definisane. Osnovni (bazni) sintaksički elementi imaju svoju semantičku prezentaciju određenu nekom klauzom (jednakostima). Svaki mogući element koji nastaje kompozicijom osnovnih elementa takođe ima semantičku klauzu definisanu preko semantičkih funkcija tih osnovnih elemenata.

Kao što smo predstavili aritmetičke izraze i izraze sa istinosnim vrednostima tako možemo prikazati i ostale izraze S. Efekat izvršavanja izraza

$$\mathcal{S}_{\mathrm{ds}}[\![x := a]\!]s = s[x \mapsto \mathcal{A}[\![a]\!]s]$$

$$\mathcal{S}_{\mathrm{ds}}[\![\texttt{skip}]\!] = \mathrm{id}$$

$$\mathcal{S}_{\mathrm{ds}}[\![S_1 \ ; \ S_2]\!] = \mathcal{S}_{\mathrm{ds}}[\![S_2]\!] \circ \mathcal{S}_{\mathrm{ds}}[\![S_1]\!]$$

$$\mathcal{S}_{\mathrm{ds}}[\![\texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2]\!] = \mathrm{cond}(\mathcal{B}[\![b]\!], \mathcal{S}_{\mathrm{ds}}[\![S_1]\!], \mathcal{S}_{\mathrm{ds}}[\![S_2]\!])$$

$$\mathcal{S}_{\mathrm{ds}}[\![\texttt{while } b \texttt{ do } S]\!] = \mathrm{FIX}\ F$$

$$\text{where } F\ g = \mathrm{cond}(\mathcal{B}[\![b]\!], g \circ \mathcal{S}_{\mathrm{ds}}[\![S]\!], \mathrm{id})$$

Slika 4: Funkcija izraza

S je takođe promena stanja, tako da ćemo definisati funkciju izraza kao $\mathcal{S}_{ds} : Stm \to (State \to State)$ prikazanom na slici 4.

Prva jednakost označava da ako se x-u dodeljuje vrednost aritmetičkog izraza a u nekom stanju (tj. ima neku vrednost), to zapravo označava da pojavljivanje promenljive x menjamo sa vrednosti koju dobijamo kada funkciju mapiranja $\mathcal{A}$ primenimo na izraz a. Dalje **skip** znači nepostojanje promene stanja izraza, odnosno $\mathcal{S}_{ds}[[skip]]s = s$. Treća jednakost predstavlja kompoziciju stanja izraza S_1 i S_2. Sledeća klauza opisuje uslovni izraz gde je *cond* pomoćna funkcija definisana kao

$$cond(p, g_1, g_2) = \begin{cases} g_1 s & \text{if } ps = tt \\ g_2 s & \text{if } ps = ff \end{cases}$$

Na kraju objasnimo još efekat *while* izraza. Najpre on ima značenje $\mathcal{S}_{ds}[[\text{while } b \text{ do } S]] = cond(B[[b]], \mathcal{S}_{ds}[[\text{while } b \text{ do } S]] \circ \mathcal{S}_{ds}[[S]], id)$, dakle predstavljen preko uslovnog izraza: *if b then (S; while b do S) else skip.*

Pošto mi želimo samo kompozicijske funkcije, ovu definiciju ne možemo koristiti, ali koristeći tu ideju vidimo da $\mathcal{S}_{ds}[[\text{while } b \text{ do } S]]$ predstavlja fiksnu tačku funkcije F definisanu kao $Fg = cond(B[[b]], g \circ \mathcal{S}_{ds}[[S]], id)$, tako da je $\mathcal{S}_{ds}[[\text{while } b \text{ do } S]] = F(\mathcal{S}_{ds}[[while \ b \text{ do } S]])$. Time S_{ds} primenjujemo samo na podelemente naredbe *while*, ne na celu naredbu. Efekat funkcije FIX je sledeći: $FIX : ((State \to State) \to (State \to State)) \to (State \to State)$. Detaljnije objašnjenje ovog problema kao i matematičku osnovu koja je potrebna za pokazivanje da ova semantička klauza zaista definiše funkciju može se pronaći u radu *Semantics with applications. A Formal Introduction* koju navodimo u literaturi [1].

5 Zaključak

Summa summarum, glavno svojstvo denotacionalne semantike jeste pridruživanje odgovarajućeg matematičkog objekata (broj, funkcija) svakom objektu jezika koji do tad nema definisano značenje. Na primer izrazi "2*3", "(3+3)", "6"predstavljaju neke sintaksički ispravno zadate izraze i svaki do njih biće mapiran brojem 6, tj. dodeljena im je vrednost 6. Takođe kompozitivnost kao važna osobina denotacione semantike omogućava jednostavniju dodelu značenja celom programu (ili nekom njegovom delu) tako što su poznata značenja elemenata od kojih se program sastoji. Formalna semantika je važna u razmatranju različitih aspekata ponašanja programskog jezika.

Literatura

[1] Hanne Riis Nielson and Flemming Nielson. *SEMANTICS WITH APPLICATIONS. A Formal Introduction.* John Wiley & Sons, 1999.

[2] D. A. Schmidt. *Denotational Semantics: a Methodology for Language Development.* Allyn & Bacon, Inc., 1986.

[3] Ken Slonneger and Barry Kurtz. Formal Syntax and Semantics of Programming Languages.

[4] Steffen van Bakel. Formal Syntax and Semantics of Programming Languages. 2002.

[5] Wikipedia. Denotaciona semantika. 2017.