Prirodna semantika kroz primere i osnovna tvrđenja

Seminarski rad u okviru kursa
Verifikacija softvera
Matematički fakultet

Aleksandra Kovačević, 1079/2016
aleksandrakovacevic099@gmail.com

11. decembar 2018

Sažetak

Formalna semantika opisuje kako se program izvršava, olakšava učenje novog programskog jezika, pisanje stabilnih programa i predstavlja osnovu formalne verifikacije programa. U ovom radu akcenat je na prirodnoj semantici koja predstavlja jedan od pristupa formalne semantike. Cilj je da se prikažu osnovne karakteristike prirodne semantike kao i to kako se ona može definisati na nekom programskom jeziku. U tu svrhu će biti definisan jednostavan programski jezik `While` i na njemu će kroz primere biti prikazana pravila i aksiome prirodne semantike.

Sadržaj

1 Uvod

Pri dizajniranju programskog jezika potrebno je definisati skup pravila na osnovu kojih je moguće pisati konstrukcije na tom jeziku. Taj skup pravila nazivamo sintaksa programskog jezika i njihovom primenom određujemo da li je neka konstrukcija u tom jeziku ispravna. Izraz poput $y = x + +;$ je sintaksno ispravan, ali kakvo je njegovo značenje? Kako bismo pridružili značenje sintaksno ispravnoj konstrukciji koristimo semantiku programskog jezika. Semantika može da se opiše formalno i neformalno. Neformalno opisivanje podrazumeva korišćenje prirodnog jezika i samim tim može biti jako neprecizno i komplikovano za razumevanje. Zbog toga se bavimo formalnom semantikom koja nam daje precizne i matematički podržane definicije.

Jedan od načina formalnog opisivanja semantike je operaciona semantika. Cilj operacione semantike je da opiše kako se program izvršava, a ne samo šta je rezultat izvršavanja. Preciznije, zanima nas kako se menjaju stanja tokom izvršavanja programa. Obuhvata dva pristupa [2]:

- Prirodna semantika (eng. *big-step semantics*) opisuje kako se dobijaju sveobuhvatni rezultati izračunavanja.
- Strukturna operaciona semantika (eng. *small-step semantics*) opisuje pojedinačne korake izračunavanja.

U ovom radu će biti prikazani osnovni principi prirodne semantike 3. Takođe, biće prikazano kako se mogu definisati pravila i aksiome prirodne semantike. Za ove potrebe definisaćemo jednostavan programski jezik `While` 2 i u odnosu na njega objasniti odgovarajuća pravila prirodne semantike 3.1.

2 Programski jezik While

Definisaćemo jednostavan programski jezik `While` kako bismo lakše demonstrirali prirodnu semantiku za jezik za koji znamo intuitivno značenje naredbi. Njegova apstraktna sintaksa je data sa [4], [3]:

$n \in Numeral, x \in Varijabla$
$a \in Aritmeticki\ Izraz, b \in Bulovski\ Izraz, S \in Naredba.$

$$a ::= n \mid x \mid (a_1 + a_2) \mid (a_1 - a_2) \mid (a_1 \times a_2)$$
$$b ::= \texttt{true} \mid \texttt{false} \mid (a_1 = a_2) \mid (a_1 \leq a_2) \mid (\neg b) \mid (b_1 \& b_2)$$
$$S ::= x := a \mid \texttt{skip} \mid (S_1;\ S_2) \mid (\texttt{while}\ b\ \texttt{do}\ S) \mid (\texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2)$$

Primer 2.1. *Primer sintaksno ispravne konstrukcije jezika* ***`While`****.*

$$y := 1\,;\ \textbf{\texttt{while}} \neg(x = 1)\ \textbf{\texttt{do}} (y := y \times x; x := x + 1)$$

Semantika Aritmetičkih izraza za `While` se može opisati kao funkcija koja preslikava Aritmetički izraz u stanju s u skup celih brojeva i data je sa:

- $A[[n]]s = n$
- $A[[x]]s = sx$
- $A[[a_1\ \textbf{op}\ a_2]]s = A[[a_1]]s\ \textbf{op}\ A[[a_2]]s$

Semantika Bulovskih izraza za `While` se može opisati kao funkcija koja preslikava Bulovski izraz u stanju s u skup istinitosnih vrednosti $T = \{tt, ff\}$.

- $B[[true]]s = tt$
- $B[[false]]s = ff$
- $B[[a_1 = a_2]]s = (A[[a_1]]s = A[[a_2]]s)$
- $B[[a_1 <= a_2]]s = (A[[a_1]]s <= A[[a_2]]s)$
- $B[[!b]]s =!(B[[b]]s)$
- $B[[b_1 \& b_2]]s = B[[b_1]]s \& B[[b_2]]s$

3 Prirodna semantika

Prirodna semantika se bavi vezom između početnog i završnog stanja izvršavanja. Ona opisuje kako se dobijaju ukupni rezultati izračunavanja. Može se predstaviti kao tranziciona relacija [1]:

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

Intuitivno ovo znači da će izvršavanje programa S sa ulaznim stanjem s rezultovati novim stanjem s'. Pravilo generalno ima formu

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

gde $S_1, ..., S_n$ predstavljaju neposredne konstituente (eng. *immediate constituents*) od S. Pravilo se sastoji od jednog zaključka koji se nalazi ispod linije i određenog broja premisa koje se nalaze iznad linije. Takođe, pravilo može da sadrži određen broj uslova koji moraju biti ispunjeni kako bi se pravilo primenilo. Pravilo koje nema premise se naziva aksioma.

Kada koristimo aksiome i pravila da izvedemo određenu tranziciju mi kreiramo stablo izvođenja (eng. *derivation tree*). Koren stabla je zaključak a listovi su instance aksioma (kod njih nema daljeg izvođenja). Unutrašnji čvorovi stabla predstavljaju instancirana pravila. Stablo izvođenja za neki izraz može biti prosto ukoliko je on instanca aksiome, inače je složeno.

3.1 Prirodna semantika jezika While

Prirodna semantika programskog jezika `While` je definisana sa nekoliko pravila i aksioma [1, 3]. Posmatrajmo tabelu 1.

Aksioma $[ass_{ns}]$ nam govori da u ulaznom stanju s, naredba $x := a$ se izvršava i vodi do rezultujućeg stanja $s[x \mapsto A[[a]]s]$ koje je praktično isto kao i stanje s osim što x ima vrednost $A[[a]]s$. Objasnićemo ovo na primeru.

Primer 3.1. $\langle y := y + 2, s_0\rangle \rightarrow s_0[y \mapsto 3]$
Neka s_0 bude stanje koje mapira sve promenljive u 1. Tada je navedeni izraz instanca pravila $[ass_{ns}]$ jer je x instancirano sa y, a sa $y + 2$, s sa s_0 a vrednost $A[[y + 2]]s_0$ je 3.

Aksioma $[skip_{ns}]$ sadrži ključnu reč `skip` koja predstavlja praznu naredbu. Aksioma je jednostavna i govori nam da naredba `skip` ne menja trenutno stanje.

Tabela 1: Prirodna semantika za `While`

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

Primer 3.2. $\langle skip, s_0\rangle \to s_0$
Neka s_0 bude proizvoljno stanje. Tada je navedeni izraz instanca pravila $[skip_{ns}]$.

Pravilo $[comp_{ns}]$ se odnosi na kompozitne naredbe i govori da bi se izvršile redom naredbe $S_1; S_2$ u stanju s prvo se mora izvršiti S_1 u stanju s. Izvršavanje S_1 u stanju s rezultuje novim stanjem s' i onda se S_2 izvršava u novom stanju s'. Rezultujuće stanje je s''.

Primer 3.3.

$$\frac{\langle y := x+1,\ s_0\rangle \to s_1 = s_0[y \mapsto 1] \qquad \langle x := y+1,\ s_1\rangle \to s_2 = s_1[x \mapsto 2]}{\langle y := x+1;\ x := y+1,\ s_0\rangle \to s_0[x \mapsto 2, y \mapsto 1]}$$

Neka s_0 bude stanje koje mapira sve promenljive u 0. Tada je navedeni izraz instanca pravila $[comp_{ns}]$. Ovde je S_1 instancirano kao $y := x + 1$, S_2 kao $x := y + 1$.

Primer 3.4. *Prikazaćemo stablo izvođenja za izraz*

$$(x := y; y := z); z := x$$

Neka je s_0 stanje koje mapira sve promenljive osim x i y u `0`. Takođe, važi $s_0\ x = 1$ i $s_0\ y = 2$.

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

Važi $s_1 = s_0[x \mapsto 2]$, $s_2 = s_1[y \mapsto 0]$, $s_3 = s_2[z \mapsto 2]$. Stablo izvođenja ima 3 lista $\langle x := y, s_0\rangle \to s_1$, $\langle y := z, s_1\rangle \to s_2$ i $\langle z := x, s_2\rangle \to s_3$. Listovi odgovaraju primeni aksiome $[ass_{ns}]$. Pravilo $[comp_{ns}]$ je primenjeno 2 puta.

Za `if` konstrukciju imamo dva pravila u zavisnosti od ispunjenosti uslova u samoj konstrukciji:

- $[if^{ns}_{tt}]$ - S_1 se izvršava u stanju s ako se $B[[b]]s$ izračunava u `tt(true)`
- $[if^{ns}_{ff}]$ - S_2 se izvršava u stanju s ako se $B[[b]]s$ izračunava u `ff(false)`

Primer 3.5.

$$\frac{\langle \textbf{\textit{skip}},\ s_0\rangle \to s_0}{\langle if\ x\ =\ 1\ then\ \textbf{\textit{skip}}\ else\ y\ :=\ x+1,\ s_0\rangle \to s_0}$$

Ukoliko stanje s_0 dodeljuje promenljivoj x vrednost `1`, onda važi $B[[x = 1]]s_0 = tt$ i sledeći izraz predstavlja instancu pravila $[if^{ns}_{tt}]$.

U slučaju da u stanju s_0 važi $x! = 1$ onda ovaj izraz ne bi bio instanca pravila $[if^{ns}_{tt}]$ jer bi važilo $B[[x = 0]]s_0 = ff$. Izraz ne bi bio ni instanca pravila $[if^{ns}_{ff}]$ jer njegova premisa ima lošu formu.

`While` konstrukcija je opisana sa aksiomom i pravilom koji objašnjavaju kako se ponaša `while` petlja. Značenje konstrukcije

`while` a `do` S *in state* s

se može objasniti:

- ukoliko se $B[[a]]s$ izračunava u `tt` onda se izvršava telo petlje(S) u stanju s a zatim se ponovo izvršava ceo izraz, samo sa novim stanjem s'
- ukoliko se $B[[a]]s$ izračunava u `ff` onda se prekida izvršavanje petlje i stanje ostaje nepromenjeno.

Pravilo $[while^{ns}_{tt}]$ se odnosi na prvi slučaj, kada se $B[[b]]s$ evaluira u `tt`, dok se aksioma $[while^{ns}_{ff}]$ odnosi na drugi slučaj.

Primer 3.6. $\langle x := 0;\ while(y < 2)\ do\ (x := x + 1; y := y + 1), s_0\rangle$
Neka je stanje s_0 takvo da važi $s_0\ y = 0$. Pokazaćemo da važi

$$\langle x := 0;\ while(y < 2)\ do\ (x := x + 1; y := y + 1), s_0\rangle \to s_0[x \mapsto 2, y \mapsto 2]$$

i korak po korak graditi stablo izvođenja T. Na početku znamo da je koren stabla T

$$\langle x := 0;\ while(y < 2)\ do\ (x := x + 1; y := y + 1), s_0\rangle \to s_{10}$$

Broj stanja 10 je uzet nasumično jer ne znamo koliko ćemo stanja promeniti do rezultujućeg. Možemo uočiti da je izraz oblika $\langle S_1; S_2, s\rangle$ (kompozitna naredba) pa ćemo prvo primeniti pravilo $[comp_{ns}]$. Forma stabla T je sledeća:

$$\frac{\langle x := 0, s_0\rangle \to s_1 \qquad\qquad T_1}{\langle x := 0;\ while(y < 2)\ do\ (x := x + 1; y := y + 1), s_0\rangle \to s_{10}}$$

Pošto važi da je naredba $\langle x := 0, s_0\rangle \to s_1$ instanca aksiome $[ass_{ns}]$ ona postaje list stabla T a novo stanje $s_1 = s_0[x \mapsto 0]$. Nastavljamo sa razvijanjem dela stabla T_1. Pošto važi $B[[(y < 2)]]s_1 = tt$ naredno pravilo koje koristimo je $while^{tt}_{ns}$. T_1 će imati oblik:

$$\frac{T_2 \qquad\qquad T_3}{T_1 = \langle while(y < 2)\ do\ (x := x + 1; y := y + 1), s_1\rangle \to s_{10}}$$

Prvo se razvija podstablo T_2 na koje se može primeniti pravilo $[comp_{ns}]$.

$$\frac{\langle x := x + 1, s_1\rangle \to s_2 \qquad \langle y := y + 1, s_2\rangle \to s_3}{T_2 = \langle x := x + 1;\ y := y + 1, s_1\rangle \to s_3}$$

Važi $s_2 = s_0[x \mapsto 1]$, $s_3 = s_0[x \mapsto 1, y \mapsto 1]$. *Listovi stabla* T_2 *su instance aksiome* $[ass_{ns}]$ *tako da je stablo* T_2 *u potpunosti izgrađeno. Stablo* T_3 *kao koren sada ima* $\langle while(y < 2)\ do\ (x := x + 1; y := y + 1), s_3\rangle \to s_{10}$ *, pa se ponavlja prethodni postupak. Pošto važi* $s_3 = s_0[x \mapsto 1, y \mapsto 1]$ *kao i* $B[[(y < 2)]]s_3 = tt$ *koristimo ponovo pravilo* $while^{tt}_{ns}$. *Stablo T*3 *se razvija kao :*

$$\dfrac{\dfrac{\langle x := x + 1, s_3\rangle \to s_4 \qquad \langle y := y + 1, s_4\rangle \to s_5}{\langle x := x + 1; y := y + 1, s_3\rangle \to s_5}\ T_4}{\langle while(y < 2)\ do\ (x := x + 1; y := y + 1), s_3\rangle \to s_{10}}$$

Važi $s_4 = s_0[x \mapsto 2, y \mapsto 1]$, $s_5 = s_0[x \mapsto 2, y \mapsto 2]$. *Listovi stabla* T_3 *su instance aksiome* $[ass_{ns}]$ *tako da je stablo* T_3 *u potpunosti izgrađeno. Stablo* T_4 *kao koren sada ima :*

$$\langle while(y < 2)\ do\ (x := x + 1; y := y + 1), s_5\rangle \to s_{10}$$

Pošto važi $s_5 = s_0[x \mapsto 2, y \mapsto 2]$ *kao i* $B[[(y < 2)]]s_5 = ff$ *koristimo aksiomu* $while^{ff}_{ns}$ *i izvršavanje petlje se završava. Na kraju važi* $s_{10} = s_5$.

4 Zaključak

Predstavljeni su osnovni principi prirodne semantike kao i kako se ona može definisati na nekom programskom jeziku. Dalje istraživanje se može usmeriti ka složenijim fragmentima programskih kodova, odnosno kako se oni mogu predstaviti pomoću pravila prirodne semantike. Takođe, bilo bi zanimljivo istražiti i druge pristupe formalnog definisanja semantike i uočiti sličnosti i razlike kao i prednosti i mane.

Literatura

[1] Flemming Nielson Hanne Riis Nielson. *Semantics with Applications: A Formal Introduction* . John Wiley & Sons, New York, 1999.

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

[3] Hanne Riis Nielson. Introduction to Semantics, 2005. on-line at: http://www2.imm.dtu.dk/courses/02240/lecture1A.pdf.

[4] Steffen van Bakel. Operational Semantics, 2002. on-line at: http://www.doc.ic.ac.uk/~svb/AssuredSoftware/notes.pdf.