Verifikacija softvera

— Semantika programskih jezika —

Milena Vujošević Janičić

www.matf.bg.ac.rs/~milena

Matematički fakultet, Univerzitet u Beogradu

Pregled

Pregled

Semantika

- Semantika pridružuje značanje ispravnim konstrukcijama na nekom programskom jeziku.
- Semantika programskog jezika određuje značenje jezika, odnosno šta se dešava kada se program izvršava.
- Obično je značajno teže definisati nego sintaksu.

Sintaksa

- Sintaksa se može zadati na različite načine, npr koristeći kontekstno slobodne gramatike, BNF
- Na primer, razmotrimo naredni jezik:

```
S ::= x := a | skip | S1 ; S2 |
      if b then S1 else S2 |
      while b do S
```

- Formalno zadavanje sintakse nam omogućava da automatski generišemo parser
- Slično, formalno zadavanje semantike ima svoje značajne primene

Neformalna semantika

- Semantika može da se opiše formalno i neformalno, često se zadaje samo neformalno.
- Uloga neformalne semantike je da programer može da razume kako se program izvršava pre njegovog pokretanja.
- Na primer, semantika naredbe `if(a<b) a++;` neformalno se opisuje sa „ukoliko je vrednost promenljive a manja od vrednosti promenljive b, onda uvećaj vrednost promenljive a za jedan"
- Programski jezici su kompleksni i zadavanje semantike programskog jezika je kompleksno

Neformalna semantika — Algol 60

A procedure statement serves to invoke (call for) the execution of a procedure body (cf. section 5.4. procedure declarations). Where the procedure body is a statement written in Algol the effect of this execution will be equivalent to the effect of performing the following operations on the program at the time of execution of the procedure statement.

- 4.7.3.1. Value assignment (call by value). All formal parameters quoted in the value part of the procedure declaration heading are assigned the values (cf. section 2.8. Values and types) of the corresponding actual parameters, these assignments being considered as being performed explicitly before entering the procedure body. The effect is as though an additional block embracing the procedure body were created in which these assignments were made to variables local to this fictitious block with types as given in the corresponding specifications (cf. section 5.4.5). As a consequence, variables called by value are to be considered as nonlocal to the body of the procedure, but local to the fictitious block (cf. section 5.4.3).
- 4.7.3.3. Body replacement and execution. Finally the procedure body, modified as above, is inserted in place of the procedure statement and executed. If the procedure is called from a place outside the scope of any non-local quantity of the procedure body the conflicts between the identifiers inserted through this process of body replacement and the identifiers whose declarations are valid at the place of the procedure statement or function designator will be avoided through suitable systematic changes of the latter identifiers.
- 4.7.4. Actual-formal correspondence. The correspondence between the actual parameters of the procedure statement and the formal parameters of the procedure heading is established as follows: The actual parameter list of the procedure statement must have the same number of entries as the formal parameter list of the procedure declaration heading. The correspondence is obtained by taking the entries of these two lists in the same order.

Neformalna semantika danas — Java

Next, a `for` iteration step is performed, as follows:

- If the `Expression` is present, it is evaluated. If the result is of type `Boolean`, it is subject to unboxing conversion (5.1.8). If evaluation of the `Expression` or the subsequent unboxing conversion (if any) completes abruptly, the `for` statement completes abruptly for the same reason. Otherwise, there is then a choice based on the presence or absence of the `Expression` and the resulting value if the `Expression` is present:
- If the `Expression` is not present, or it is present and the value resulting from its evaluation (including any possible unboxing) is true, then the contained `Statement` is executed. Then there is a choice:
- If execution of the `Statement` completes normally, then the following two steps are performed in sequence: 1. First, if the `ForUpdate` part is present, the expressions are evaluated in sequence from left to right; their values, if any, are discarded. If evaluation of any expression completes abruptly for some reason, the for statement completes abruptly for the same reason; any `ForUpdate` statement expressions to the right of the one that completed abruptly are not evaluated. If the `ForUpdate` part is not present, no action is taken. 2. Second, another for iteration step is performed.
- If execution of the `Statement` completes abruptly, see 14.14.1.3 below.
- If the `Expression` is present and the value resulting from its evaluation (including any possible unboxing) is false, no further action is taken and the for statement completes normally.

Problemi sa neformalnom semantikom

- Problemi sa neformalnim opisima:
  - Dvosmislenost
  - Nekonzistentnost (kontradikcije)
  - Nedorečenost
- Nekonzistentnosti u semantici koja je neformalno zadata je teško pronaći, dok u formalno zadatim semantikama je moguće automatski proveriti konzistentnost

Kako savladti semantiku?

- Semantika programskog jezika se obično ne uči čitanjem specifikacije
- Programeri obično razvijaju svoju mentalnu sliku semantike svakodnevnim korišćenjem jezika, tj. kroz kompajler/interperter
- Znanje koje se stiče iskustvenim putem na osnovu toga šta neki programi rade je neprecizno i nepotpuno, ali je za neke obične upotrebe to u redu

Kako savladti semantiku?

- Kako se radi razvoj kompajlera? Da li razvoj kompajlera sme da ide na takav način? Da li i može da ide na takav način ako ne postoji kompajler za taj programski jezik?
- Za razvoj kompajlera neophodno je da se čita specifikacija.
- Ako je specifikacija neformalna, moguće je da će je različiti programeri različito razumeti i da će se zato različiti kompajleri ponašati drugačije.
- Ako se kompajleri različito ponašaju, kako znati koji je usklađen sa predviđenom semantikom tog programskog jezika?
- Formalnom semantikom mogu se prevazići ovi problemi, a formalna semantika se može zadati i za realne programske jezike (npr, semantika jezika ML je zadata formalno, "The definition of standard ML", MIT Press, M. Tofte, R. Milner, R. Harper)

Formalna semantika

- Cilj: precizno definisati značenje (ponašanje) programa
- Formalnom semantikom se definiše značenje samo sintaksno ispravnih programa
- U skladu sa time, precizna semantika je definisana implementacijom kompajlera: na primer, precizna semantika jezika C/C++ je zapravo data kompajlerom gcc ili clang
- Međutim, iako precizna, takva semantika za mnoge probleme nije upotrebljiva.
  - Na primer, recimo da nas interesuje da li su dve implementacije funkcije koja izračunava faktorijel ekvivalentne. Korišćenjem gcc-a ili clang-a, možemo da proverimo da li se te dve implementacije poklapaju na nekim ulazima, ali ne možemo da dokažemo njihovu ekvivalentnost
  - Takođe, ne može se upotrebom kompajlera, bez dodatnih objašnjenja, razumeti jezik
- Formalna semantika daje okvir za **rezonovanje** o osobinama programa

Ko rezonuje o programima?

Programer

- Da li kôd radi ono što želim? (Testiranjem možemo samo da podignemo pouzdanost da je kôd dobar, ali semantika nam govori šta se tu dešava i da li je to u skladu sa našim očekivanjima.)
- Da li je zamena koda *boljim* kodom u redu (proces refaktorisanja)?
  Optimizacijom od koda koji radi, često dobijamo kôd koji je brži, ali ne radi...
  Zamenu koda rade ne samo programeri, već i optimizacije u kompajlerima.
  Npr, da li je zamena
  `x=x+1; x=x+1;`
  sa
  `x=x+2;`
  u redu?

Ko rezonuje o programima?

Dizajner programskog jezika / Programer koji učestvuje u razvoju kompajlera

- Brz razvoj prototipa jezika
  - Operaciona semantika je opis apstraktne mašine koja izvršava program i ona omogućava brz razvoj prototipa
  - Ako je za neki konstrukt programskog jezika potrebno previše prostora da se formalno zapiše, onda je verovatno u pitanju loša ideja koju ne treba implementirati jer je previše kompleksna i niko neći razumeti kako treba
- Optimizacije
- Napredne implementacione tehnike (npr, funkcionalni programski jezici zahtevaju netrivijalne transformacije kako bi se prevli u kôd koji se brzo izvršava)

Uloga formalne semantike

- Formalna semantika omogućava formalno rezonovanje o svojstvima programa
- Na primer, formalna semantika nekog jezika, zajedno sa modelom programa (tranzicionim sistemom) omogućava ispitivanje različitih uslova ispravnosti/korektnosti tog programa, kao i odnosa između dva programa
- Primer: „Formalizacija LLVM međureprezentacije za verifikaciju transformacija programa"
  `https://www.cis.upenn.edu/~stevez/vellvm/`
  Formalizing the LLVM Intermediate Representationfor Verified Program Transformations
  `https://www.cis.upenn.edu/~stevez/papers/ZNMZ12.pdf`

Osnovne vrste semantika

Postoji veliki broj mogućnosti zadavanja semantike programa, naredna tri su osnovne vrste semantike

- **Operaciona semantika** — Šta program radi?
  Odgovara izvršavanju u okviru apstraktne mašine
- **Denotaciona semantika** — Šta program znači?
  Matematički objekti u nekom adekvatnom semantičkom domenu (npr. funkcija iz stanja u stanje, odnosno transformacija ulaza u izlaz)
- **Aksiomatska semantika** — Koje osobine ima program?
  Kolekcija logičkih osobina koje obezbeđuje program (npr. preduslovi i postuslovi, prisutno u paradigmi *design by contract*)

Koju semantiku korisiti?

Koju semantiku korisiti?

Različitim programskim jezicima prirodno odgovaraju različite semantike

- imperativna
- funkcionalna
- konkurentna
- objektno orijentisana
- nedeterminističko izvršavanje ...

Planirana upotreba semantike

- razumevanje jezika
- verifikacija i analiza programa
- pravljenje prototipa
- konstrukcija kompilatora ...

Sintaksa **While** jezika

- Razmotrimo naredni jezik:

  n ::= 0 | 1 | n 0 | n 1

  a ::= n | x | a_1 + a_2 | a_1 * a_2 | a_1 - a_2

  b ::= true | false | a_1 = a_2 | a_1 <= a_2 | ¬b | $b_1 \wedge b_2$

  S ::= x := a |
  skip |
  S1 ; S2 |
  if b then S1 else S2 |
  while b do S

Numerali

Nezavisno od tipa semantike, može se definisati preslikavanje koje numeralu dodeljuje broj

$$\mathcal{N} : Num \to Z$$

$$\begin{aligned} \mathcal{N}[\![0]\!] &= \mathbf{0} \\ \mathcal{N}[\![1]\!] &= \mathbf{1} \\ \mathcal{N}[\![n\ 0]\!] &= \mathbf{2} \cdot \mathcal{N}[\![n]\!] \\ \mathcal{N}[\![n\ 1]\!] &= \mathbf{2} \cdot \mathcal{N}[\![n]\!] + \mathbf{1} \end{aligned}$$

Stanje

Stanje označava trenutnu sliku memorije, tj koju vrednost ima koja promenljiva: $State : Var \to Z$, pri čemu je $Var \to Z$ funkcija iz skupa promenljivih Var u skup celih brojeva Z

- Neka je $Var = \{x, y, z\}$. Jedno stanje može biti zadato listom $[x \to 2, y \to 5, z \to 0]$
- Stanje određuje vrednost za promenljivu, pa tako $s\ x$ označava vrednost promenljive x u stanju s. Ta vrednost, za prethodno stanje je 2. Vrednost izraza $x + 1$ u tom stanju bi bila 3.

Semantika aritmetičkog izraza

Za dat aritmetički izraz *Aexp* i za zadato stanje *State* funkcija $\mathcal{A}$ definiše vrednost aritmetičkog izraza u datom stanju.

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

$$\begin{aligned}
\mathcal{A}[\![n]\!]s &= \mathcal{N}[\![n]\!] \\
\mathcal{A}[\![x]\!]s &= s\ x \\
\mathcal{A}[\![a_1 + a_2]\!]s &= \mathcal{A}[\![a_1]\!]s + \mathcal{A}[\![a_2]\!]s \\
\mathcal{A}[\![a_1 \star a_2]\!]s &= \mathcal{A}[\![a_1]\!]s \cdot \mathcal{A}[\![a_2]\!]s \\
\mathcal{A}[\![a_1 - a_2]\!]s &= \mathcal{A}[\![a_1]\!]s - \mathcal{A}[\![a_2]\!]s
\end{aligned}$$

Semantika logičkog izraza

Neka je $T = \{tt, ff\}$ gde tt označava tačno a ff netačno. Za dat logički izraz $Bexp$ i za zadato stanje $State$ funkcija $\mathcal{B}$ definiše vrednost logičkog izraza u datom stanju

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

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

Semantika

- *Stm* je skup svih programa, dakle za dati program funkcija $\mathcal{S}$ vraća značenje (odnosno semantiku) kao preslikavanje iz stanja u stanje

$$\mathcal{S} : Stm \to (State \hookrightarrow State)$$

pri čemu je $State \hookrightarrow State$ skup parcijalnih funkcija iz *State* u *State*. Parcijalnost označava da funkcija može biti nedefinisana za neke ulazne parametre (npr program se ne zaustavlja, beskonačna petlja, rekurzija...)

Pregled

Operaciona semantika

- Operaciona semantika opisuje kako se izračunavanje izvršava.
- Najčešće se koristi za opis i rezonovanje o imperativnim jezicima jer individualni koraci izračunavanja opisuju na koji način se menja stanje programa.
- U okviru ove semantike postoje prirodna operaciona semantika (*big-step semantics*, opisuje ukupne rezultate izračunavanja) i strukturna operaciona semantika (*small-step semantics*, opisuje individualne korake izračunavanja).
- Naredni primer je prikaz na koji način se može zadati operaciona semantika za jedan mali programski jezik. Zadavanje semantike za različite jezike i koncepte može da varira

Primer operacione semantike: prirodna semantika

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

Intuitivno ovo znači da izvršavanje programa S sa ulaznim stanjem s će se završiti i rezultujuće stanje će biti s'.
Pravilo generalno ima formu

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

gde S_1, ..., S_n nazivamo neposrednim konstituentima (eng. *immediate constituents*) od S. Pravilo se sastoji iz određenog broja *premisa* (nalaze se iznad linije) i jednog *zaključka* (nalazi se ispod linije). Pravilo sa praznim skupom premisa se naziva *aksioma*. Pravilo takođe može imati određeni broj *uslova* (nalaze se sa desne strane linije) koji moraju biti ispunjeni kako bi se primenilo pravilo.

Primer prirodne semantike za jezik **While**

$$
\begin{array}{ll}
[ass_{ns}] & \langle x := a, s\rangle \to s[x \mapsto \mathcal{A}[[a]]s] \\
[skip_{ns}] & \langle skip, s\rangle \to s \\
[comp_{ns}] & \dfrac{\langle S_1, s\rangle \to s', \langle S_2, s'\rangle \to s''}{\langle S_1; S_2, s\rangle \to s''} \\
[if^{tt}_{ns}] & \dfrac{\langle S_1, s\rangle \to s'}{\langle \text{if } b \text{ then } S_1 \text{ else } S_2,\ s\rangle \to s'} \ \text{if } \mathcal{B}[[b]]s = tt \\
[if^{ff}_{ns}] & \dfrac{\langle S_2, s\rangle \to s'}{\langle \text{if } b \text{ then } S_1 \text{ else } S_2,\ s\rangle \to s'} \ \text{if } \mathcal{B}[[b]]s = ff \\
[while^{tt}_{ns}] & \dfrac{\langle S, s\rangle \to s', \langle \text{ while } b \text{ do } S,\ s'\rangle \to s''}{\langle \text{ while } b \text{ do } S,\ s\rangle \to s''} \ \text{if } \mathcal{B}[[b]]s = tt \\
[while^{ff}_{ns}] & \langle \text{ while } b \text{ do } S,\ s\rangle \to s \ \text{if } \mathcal{B}[[b]]s = ff
\end{array}
$$

Primer

z := x; x := y; y := z

Neka s_0 bude stanje koje preslikava x u 5, y u 7 i z u 0. Tada dobijamo sledeće stablo izvođenja:

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

Važi $s_1 = s_0[z \mapsto 5]$, $s_2 = s_1[x \mapsto 7]$, $s_3 = s_2[y \mapsto 5]$. Stablo izvođenja ima 3 primene aksiome $[ass_{ns}]$, dok je $[comp_{ns}]$ primenjeno 2 puta.

Osobine semantike

- Prethodno izvođenje nije interesantno u smislu da ga izvodi čovek, već je interesantno jer može da ga izvodi računar
- Kada imamo definisanu semantiku, to nam omogućava da rezonujemo o osobinama te semantike
  - Ako želimo da dokažemo da nešto nije ispunjeno, dovoljno je naći primer koji narušava dato svojstvo
  - Ako želimo da dokažemo da nešto uvek važi, onda to ne može da se uradi primerom, potrebno je da postoji semantika koja nam daje mogućnost da matematički dokažemo da je svojstvo uvek ispunjeno

Osobine semantike

- Primer: Može da nas interesuje da li su dve naredbe semantički ekvivalentne, donosno da li važi da za svaka dva stanja s i s' važi sledeće

$$\langle S_1, s\rangle \to s' \text{ akko } \langle S_2, s\rangle \to s'$$

- Možemo da formulišemo sledeću lemu
  Naredba
  `while b do S`
  je semantički ekvivalentna sa
  `if b then (S; while b do S) else skip`

Skica dokaza

- Potrebno je dokazati dva smera, tj

$$\langle \textit{while } b \textit{ do } S, s\rangle \to s''$$

ako i samo ako

$$\langle \textit{if } b \textit{ then } (S;\ \textit{while } b \textit{ do } S) \textit{ else skip}, s\rangle \to s''$$

Skica dokaza

- Dokaz se izvodi konstruisanjem stabla izvođenja na osnovu pravila izvođenja koja su data semantikom. Na primer, ako pretpostavimo da važi

$$\langle \mathit{while}\ b\ \mathit{do}\ S, s\rangle \to s''$$

onda postoji stablo izvođenja kojima se dolazi do stanja s". Ukoliko pogledamo pravila izvođenja, do takvog stabla možemo da dođemo samo primenim pravila $[\mathit{while}^{tt}_{ns}]$ ili pravila $[\mathit{while}^{ff}_{ns}]$. Potrebno je razmotriti i jedan i drugi slučaj.

Skica dokaza

- (prvi slučaj): ukoliko se primeni pravilo $[while_{ns}^{tt}]$ to se onda stablo izvođenja svodi na primenu ovog pravila:

$$\frac{\langle S, s\rangle \to s', \langle\ while\ b\ do\ S,\ s'\rangle \to\ s''}{\langle\ while\ b\ do\ S,\ s\rangle \to\ s''}$$

pri čemu važi $\mathcal{B}[[b]]s\ =\ tt$ odnosno stablo izvođenja izgleda ovako

$$\frac{T_1 \quad T_2}{\langle\ while\ b\ do\ S,\ s\rangle \to\ s''}$$

pri čemu je T_1 nekakvo izvođenje za $\langle S, s\rangle \to s'$ a T_2 nekakvo izvođenje za $\langle\ while\ b\ do\ S,\ s'\rangle \to\ s''$

Skica dokaza

- Koristeći T_1 i T_2, možemo da primenimo i pravilo kompozicije $[comp_{ns}]$ i da izvedemo sledeći zaključak

$$\frac{T_1 \quad T_2}{\langle S;\ while\ b\ do\ S,\ s\rangle \rightarrow\ s''}$$

- Pošto znamo da je $\mathcal{B}[[b]]s\ =\ tt$, možemo da primenimo $[if_{ns}^{tt}]$ i da nam stablo izvođenja sada izgleda ovako, čime smo pokazali da željeno svojstvo važi:

$$\frac{\dfrac{T_1 \quad T_2}{\langle S;\ while\ b\ do\ S,\ s\rangle \rightarrow\ s''}}{\langle if\ b\ then\ (S;\ while\ b\ do\ S)\ else\ skip, s\rangle \rightarrow s''}$$

- Dalje je potrebno razmotriti $[while_{ns}^{ff}]$

Dokazi — Indukcija po obliku stabla izvođenja

- Dokaži da svojstvo važi za svako jednostavno stablo pokazivanjem da važi za aksiome tranzicionog sistema
- Dokaži da svojstvo važi i za sva složena stabla izvođenja: za svako pravilo pretpostavi da svojstvo važi za njegove premise (induktivna hipoteza) i dokaži da takođe važi i za zaključak pravila ukoliko su uslovi pravila zadovoljeni

Strukturna operaciona semantika

Tranzicionu relaciju zapisujemo ovako

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

ovo treba razmatrati kao *prvi korak* izvršavanja programa S u stanju s koji vodi do stanja γ.
Možemo razlikovati dva slučaja 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'.

Strukturna operaciona semantika

$$
\begin{array}{ll}
[\mathrm{ass}_{\mathrm{sos}}] & \langle x := a,\ s\rangle \Rightarrow s[x \mapsto \mathcal{A}[\![a]\!]s] \\
[\mathrm{skip}_{\mathrm{sos}}] & \langle \mathtt{skip},\ s\rangle \Rightarrow s \\
[\mathrm{comp}^1_{\mathrm{sos}}] & \dfrac{\langle S_1,\ s\rangle \Rightarrow \langle S_1',\ s'\rangle}{\langle S_1;S_2,\ s\rangle \Rightarrow \langle S_1';S_2,\ s'\rangle} \\
[\mathrm{comp}^2_{\mathrm{sos}}] & \dfrac{\langle S_1,\ s\rangle \Rightarrow s'}{\langle S_1;S_2,\ s\rangle \Rightarrow \langle S_2,\ s'\rangle} \\
[\mathrm{if}^{\mathrm{tt}}_{\mathrm{sos}}] & \langle \mathtt{if}\ b\ \mathtt{then}\ S_1\ \mathtt{else}\ S_2,\ s\rangle \Rightarrow \langle S_1,\ s\rangle \text{ if } \mathcal{B}[\![b]\!]s = \mathbf{tt} \\
[\mathrm{if}^{\mathrm{ff}}_{\mathrm{sos}}] & \langle \mathtt{if}\ b\ \mathtt{then}\ S_1\ \mathtt{else}\ S_2,\ s\rangle \Rightarrow \langle S_2,\ s\rangle \text{ if } \mathcal{B}[\![b]\!]s = \mathbf{ff} \\
[\mathrm{while}_{\mathrm{sos}}] & \langle \mathtt{while}\ b\ \mathtt{do}\ S,\ s\rangle \Rightarrow \\
 & \qquad \langle \mathtt{if}\ b\ \mathtt{then}\ (S;\ \mathtt{while}\ b\ \mathtt{do}\ S)\ \mathtt{else}\ \mathtt{skip},\ s\rangle
\end{array}
$$

SOS omogućava praćenje „sitnijih detalja” izračunavanja

Strukturna operaciona semantika

- Slično kao kod prirodne semantike, i u okviru strukturne operacione semantike moguće je dokazati razna interesantna svojstva jezika
- Dokazi — indukcija po dužini izvođenja
  - Dokaži da svojstvo važi za izvođenje dužine nula
  - Dokaži da svojstvo važi i za sva konačna izvođenja: pretpostavi da svojstvo važi za dužinu izvođenja k (induktivna hipoteza) i dokaži da takođe važi i za dužinu izvođenja $k + 1$

Odnos prirodne i strukturne opearcione semantike

- Za ove dve semantike se može dokazati ekvivalentnost: za svako izvođenje u okviru prirodne semantike postoji odgovarajuće izvođenje u okviru strukturne operacione semantike i obrnuto

Pregled

Denotaciona semantika

- Denotaciona semantika definiše značenje prevođenjem u drugi jezik, za koji se pretpostavlja da je poznata semantika
- Najčešće je taj drugi jezik nekakav matematički formalizam
- Povezivanje svakog dela programskog jezika sa nekim matematičkim objektom kao što je broj ili funkcija: svaka sintaksna definicija se tretira kao objekat na koji se može primeniti funkcija koja taj objekat preslikava u matematički objekat koji definiše značenje.

Svojstvo kompozitivnosti

- Dodeljivanjem značenja delovima programa dodeljuje se značenje celokupnom programu, tj **semantika jedne programske celine definisana je preko semantike njenih poddelova.**
- Ova osobina denotacione semantike naziva se **kompozitivnost**.
- Definisane funkcije $\mathcal{A}$ i $\mathcal{B}$ su primeri denotacionih definicija jer su to funkcije koje zadovoljavaju svojstvo kompozitivnosti.
- Funkcije prirodne i strukturne operacione semantike nisu primeri denotacionih definicija jer ne zadovoljavaju svojstvo kompozitivnosti
- Razmotrimo sličan primer koji uključuje brojni sistem u osnovi 10

Denotaciona semantika

Sintaksni domeni i pravila:

N : *Numeral* N je numeral
C : *Cifra* C je cifra 0,1...,9
I : *Izraz*
Cifra ::== 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
Numeral ::== *Cifra* | *NumeralCifra*
Izraz ::== *Numeral* | *Izraz* + *Izraz*

Denotaciona semantika

Sledeći korak jeste definisanje matematičkih objekata koji će predstavljati semantičke vrednosti. Ti matematički objekti nazivaju se **semantički domeni**. Njihova kompleksnost zavisi od toga koliko je kompleksan programski jezik kojem dajemo značenje, u ovom jednostavnom primeru, semantička vrednost može biti i samo prirodan broj.

Semantički domeni

$\mathbb{N} = 0, 1, 2, 3, ...$ skup prirodnih brojeva

Denotaciona semantika

Funkcije značenja daju značenje uvedenim sintaksnim definicijama.

Funkcije značenja

$povezicn : C \to \mathbb{N}$ unarna funkcija - povezuje cifru sa prirodnim brojem
$povezinn : N \to \mathbb{N}$ unarna funkcija - povezuje numeral sa prirodnim brojem
$semantika : I \to \mathbb{N}$ unarna funkcija - povezuje izraz sa prirodnim brojem
$plus : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ binarna funkcija plus - odgovara sabiranju prirodnih brojeva
$puta : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ binarna funkcija puta - odgovara množenju prirodnih brojeva

$povezicn[[0]] = 0, ..., povezicn[[9]] = 9$
$povezinn[[C]] = povezicn[[C]]$
$povezinn[[NC]] = plus(puta(10, povezinn[[N]]), povezinn[[C]])$
$semantika[[N]] = povezinn[[N]]$
$semantika[[I_1 + I_2]] = plus(semantika[[I_1]], semantika[[I_2]])$

Zagrade [[,]] imaju ulogu da razdvoje semantički deo od sintaksnog dela. U okviru zagrada nalazi se sintaksni deo definicija.

Denotaciona semantika

Pronaći značenje izraza 2+32+61.

semantika[[2+32+61]]
= plus(semantika[[2+32]],semantika[[61]])
= plus(plus(semantika[[2]],semantika[[32]]),povezinn[[61]])
= plus(plus(2,32),61) = plus(34,61) = 95

jer je:

semantika[[2]] = povezinn[[2]] = povezicn[[2]] = 2
semantika[[32]] = povezinn[[32]] = plus(puta(10,povezinn[[3]]),povezinn[[2]])
=plus(puta(10,povezicn[[3]]),povezicn[[2]]) = plus(puta(10,3),2)
= plus(30,2) = 32
semantika[[61]] = povezinn[[61]] = plus(puta(10,povezinn[[6]]),povezinn[[1]])
= plus(puta(10,povezicn[[6]]),povezicn[[1]]) = plus(puta(10,6),1) = plus(60,1) = 61

Primer denotacione semantike za jezik **While**

Oznaka $\circ$ je kompozicija funkcija, koriste se pomoćne funkcije *cond* i *FIX*.

$$
\begin{aligned}
\mathcal{S}_{\mathrm{ds}}[\![x := a]\!]s &= s[x \mapsto \mathcal{A}[\![a]\!]s] \\
\mathcal{S}_{\mathrm{ds}}[\![\texttt{skip}]\!] &= \mathrm{id} \\
\mathcal{S}_{\mathrm{ds}}[\![S_1 \texttt{ ; } S_2]\!] &= \mathcal{S}_{\mathrm{ds}}[\![S_2]\!] \circ \mathcal{S}_{\mathrm{ds}}[\![S_1]\!] \\
\mathcal{S}_{\mathrm{ds}}[\![\texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2]\!] &= \mathrm{cond}(\mathcal{B}[\![b]\!], \mathcal{S}_{\mathrm{ds}}[\![S_1]\!], \mathcal{S}_{\mathrm{ds}}[\![S_2]\!]) \\
\mathcal{S}_{\mathrm{ds}}[\![\texttt{while } b \texttt{ do } S]\!] &= \mathrm{FIX}\ F \\
\text{where } F\ g &= \mathrm{cond}(\mathcal{B}[\![b]\!], g \circ \mathcal{S}_{\mathrm{ds}}[\![S]\!], \mathrm{id})
\end{aligned}
$$

Kompozicija funkcija

Efekat kopozicije naredbi je kompozicija funkcija izvršavanja nardbi S_1 i S_2. Kompozicija funkcija je definisana tako da ako je jedna funkcija nedefinisana za dato stanje, onda je njihova kompozicija takođe nedefinisana.

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

$$= \begin{cases} s'' & \text{if there exists } s' \text{ such that } \mathcal{S}_{\mathrm{ds}}[\![S_1]\!]s = s' \\ & \text{and } \mathcal{S}_{\mathrm{ds}}[\![S_2]\!]s' = s'' \\ \underline{\text{undef}} & \text{if } \mathcal{S}_{\mathrm{ds}}[\![S_1]\!]s = \underline{\text{undef}} \\ & \text{or if there exists } s' \text{ such that } \mathcal{S}_{\mathrm{ds}}[\![S_1]\!]s = s' \\ & \text{but } \mathcal{S}_{\mathrm{ds}}[\![S_2]\!]s' = \underline{\text{undef}} \end{cases}$$

Primer kompozicije naredbi dodele

Kako program predstavlja skup izraza odvojenih simbolom ';', efekat celog programa je kompozicija efekata svih individualnih izraza. Primer:
$Stanje[[z := x; x := y; y := z]] =$
$Stanje[[y := z]] \circ Stanje[[x := y]] \circ Stanje[[z := x]]$
Za konkretne vrednosti:

$$Stanje[[z := x; x := y; y := z]]([x \to 5, y \to 7, z \to 0]) =$$
$$Stanje[[y := z]] \circ Stanje[[x := y]] \circ Stanje[[z := x]]$$
$$([x \to 5, y \to 7, z \to 0]) =$$
$$Stanje[[y := z]] \circ Stanje[[x := y]]([x \to 5, y \to 7, z \to 5]) =$$
$$Stanje[[y := z]]([x \to 7, y \to 7, z \to 5]) = [x \to 7, y \to 5, z \to 5]$$

Funkcija *cond*

Funkcija *cond* je uvedena za modelovanje `if-then-else` naredbe

$$\text{cond: } (\mathbf{State} \to \mathbf{T}) \times (\mathbf{State} \hookrightarrow \mathbf{State}) \times (\mathbf{State} \hookrightarrow \mathbf{State}) \to (\mathbf{State} \hookrightarrow \mathbf{State})$$

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

Funkcija *cond*

$$
\begin{aligned}
& \mathcal{S}_{\mathrm{ds}}[\![\texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2]\!]\ s \\
= \quad & \mathsf{cond}(\mathcal{B}[\![b]\!],\ \mathcal{S}_{\mathrm{ds}}[\![S_1]\!],\ \mathcal{S}_{\mathrm{ds}}[\![S_2]\!])\ s \\
= \quad & \begin{cases}
s' & \text{if } \mathcal{B}[\![b]\!]s = \mathbf{tt} \text{ and } \mathcal{S}_{\mathrm{ds}}[\![S_1]\!]s = s' \\
 & \text{or if } \mathcal{B}[\![b]\!]s = \mathbf{ff} \text{ and } \mathcal{S}_{\mathrm{ds}}[\![S_2]\!]s = s' \\
\underline{\text{undef}} & \text{if } \mathcal{B}[\![b]\!]s = \mathbf{tt} \text{ and } \mathcal{S}_{\mathrm{ds}}[\![S_1]\!]s = \underline{\text{undef}} \\
 & \text{or if } \mathcal{B}[\![b]\!]s = \mathbf{ff} \text{ and } \mathcal{S}_{\mathrm{ds}}[\![S_2]\!]s = \underline{\text{undef}}
\end{cases}
\end{aligned}
$$

Funkcija FIX

Najteže je definisati semantiku za naredbu *while* a da se pritom zadrži kopozitivnost. Prilikom definisanja, mora da važi ekvivalentnost naredbe *while* sa narednom naredbom

$$\texttt{if } b \texttt{ then } (S; \texttt{while } b \texttt{ do } S) \texttt{ else skip}$$

S obzirom na defnisan *if* i definisan *skip*, potrebno je da važi sledeće

$$\mathcal{S}_{\mathrm{ds}}[\![\texttt{while } b \texttt{ do } S]\!] = \mathrm{cond}(\mathcal{B}[\![b]\!], \mathcal{S}_{\mathrm{ds}}[\![\texttt{while } b \texttt{ do } S]\!] \circ \mathcal{S}_{\mathrm{ds}}[\![S]\!], \mathrm{id})$$

Primetimo da to ne može da se koristi kao definicija za *while*, jer nema svojstvo kompozitivnosti

Funkcija FIX

Međutim, prethodno nam pokazuje da je

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

fiksna tačka funkcionala

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

odnosno, važi

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

Funkcija FIC

Na taj način se dobija kompozitivnost, i to zapisujemo na sledeći način:

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

Pri čemu je tip funkcije FIX

$$\mathsf{FIX}\colon ((\mathbf{State} \hookrightarrow \mathbf{State}) \to (\mathbf{State} \hookrightarrow \mathbf{State})) \to (\mathbf{State} \hookrightarrow \mathbf{State})$$

Međutim, ne mora svaka funkcija da ima fiksnu tačku, i takođe ako postoji fiksna tačka ne mora da bude jedinstvena. Zbog toga je potrebno dodatno definisati osobine funkcije FIX kako bi ova definicija bila ispravna.

Denotaciona semantika

- Denotaciona semantika apstrahuje izvršavanje programa.
- Koristi se često za definisanje semantike funkcionalnih programskih jezika — funkcionalno programiranje zasniva se na pojmu matematičkih funkcija i izvršavanje programa svodi se na evaluaciju funkcija. Međutim, prethodni primer je bio vezan za jezik koji je imperativan.
- Analiziranje programa se svodi na analiziranje matematičkih objekata, što olakšava formalno dokazivanje semantičkih svojstava programa.
- Može se dokazati ekvivalentnost prethodno definisanih operacionih semantika sa denotacionom semantikom.

Pregled

Horove trojke

- Aksiomatska semantika omogućava viši nivo apstrakcije i udaljavanje od konkretne semantike programskog jezika
- Aksiomska semantika zasniva se na Horovoj logici
- Horova trojka `{P} S {Q}`; gde se `{P}` naziva preduslov a `{Q}` postuslov, opisuje kako izvršavanje dela koda menja stanje izračunavanja:
  - ako je ispunjen preduslov `{P}`,
  - ako se izvršavanje komande `S` završava za dato stanje
  - onda će `{Q}` da važi u stanju u kojme se `S` završilo

Aksiomatska semantika — primer

- Na primer

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

- Promenljiva n u ovom primeru je specijalna *logička* promenljiva i ona se ne pojavljuje u razmatranim naredbama. Njena uloga je da zapamti inicijalnu vrednost promenljive x. Ne možemo zapisati $y = x! \wedge x > 0$, jer bi to označavalo finalnu vrednost promenljive x, koja nije ista sa početnom.
- Dakle, imamo dve vrste promenljivih: logičke i programske promenljive. Logičke promenljive se ne menjaju u okviru programa i njihove vrednosti su fiksirane.

Vrste pristupa

- Postoje dva pristupa definisanju preduslova i postuslova
  - intenzionalni pristup — ideja je da se uvede novi eksplicitini jezik koji se koristi za opisivanje uslova, pri čemu taj jezik treba da bude dovoljno bogat i izražajan
  - ekstenzioni pristup — ideja je da se uslovi opisuju kao predikati, tj funkcije iz $State \to T$, jednostavniji pristup.

Aksiomatska semantika za definisani jezik

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

Aksiomatska semantika

- Aksioma dodele: ako stanje s u okviru kojeg je je x postavljeno na $\mathcal{A}[a]$ zadovoljava svojstvo P, tada će svojstvo P da važi nakon dodele
- Aksioma skip: ako P važi pre naredbe skip, onda važi i posle
- To su zapravo sheme aksioma koje se generišu posebno za svako svojstvo P koje je od intresa
- Pravila izvođenja: pravilo kompozicije, uslova i petlje
- Pravilo petlje definiše P kao invarijantu petlje koja važi pre i nakon izvršavanja petlje
- Pravilo jačanja preduslova i slabljenja postuslova

Aksiomatska semantika

- Sa ovako postavljenim sistemom, mogu se dokazati razna interesantna svojstva programa koji se razmatra

Pregled

Literatura

Literatura

- Semantics with Applications: A Formal Introduction. Hanne Riis Nielson, Flemming Nielson. John Wiley & Sons, Inc. http://faculty.sist.shanghaitech.edu.cn/faculty/songfu/course/spring2018/CS131/sa.pdf