Semantyka aksjomatyczna

Motywacje

1. Punkt widzenia użytkownika języka programowania na problem znaczenia przypisywanego programom zwykle jest różny od punktu widzenia projektanta języka czy osoby piszącej kompilator.

2. Potrzeba badania własności programu i jego poprawności:

   (a) jeżeli dane wejściowe programu p spełniają własność α i program poprawnie zakończy obliczenia, to dane wyjściowe spełniają własność β – *częściowa poprawność* p względem α i β.

(b) jeżeli dane wejściowe programu p spełniają własność α, to program prawidłowo zakończy obliczenia i dane wyjściowe spełniają własność β – *całkowita poprawność* p względem α i β.

3. Potrzeba skonstruowania języka do wyrażania takich stwierdzeń oraz dostarczenia metod ich weryfikacji bez odwoływania się do specyfiki urządzenia wykonującego obliczenia – metody oparte na logice.

Logika Hoare'a

Język

formuły

$$\{\alpha\}\, c\, \{\beta\},$$

gdzie α, β – formuły logiki I rzędu,

α – nazywamy *warunkiem wstępnym*,

β – *warunkiem końcowym*,

c – instrukcja (program).

Wyrażenie takie nazywamy *specyfikacją* lub *trójką Hoare'a*.

W zależności od przyjętych definicji, formułę taką będziemy rozumieć w sensie 2a lub 2b.

Przykład 4
Niech

$$c = \underline{\text{while}}\ tt\ \underline{\text{do}}\ \underline{\text{skip}}$$

Wykonanie c nie kończy się nigdy. Zgodnie z interpretacją *częściowej poprawności* (2a) następująca specyfikacja jest prawdziwa:

$$\{tt\}\ c\ \{ff\} \tag{2}$$

Co więcej, ponieważ c pętli się, specyfikacja

$$\{\alpha\}\ c\ \{\beta\} \tag{3}$$

jest prawdziwa w sensie 2a dla dowolnych α i β.

Oczywiście nie zachodzi to dla (3) w sensie 2b. ∎

Język logiki I rzędu

1. V – zbiór zmiennych indywiduowych;

2. F_m – zbiór m-argumentowych symboli funkcyjnych, $m = 0, 1, \ldots$; F_0 – oznacza stałe indywiduowe;

3. P_m – zbiór m-argumentowych symboli predykatywnych, $m = 1, \ldots$;

4. zbiór stałych logicznych: tt, ff, spójników logicznych: $\wedge, \vee, \Rightarrow, \neg$ i kwantyfikatorów: $\exists, \forall$;

5. W standardowy sposób tworzymy zbiory: *termów (TR)* i *formuł (FR)*.

Interpretacja

1. D – dziedzina interpretacji. $TF = \{tt, ff\}$ – zbiór wartości logicznych.

2. *Wartościowanie* zmiennych – funkcja $val : V \to D$.
   $Val = [V \to D]$

3. Interpretacja I – funkcja, która przyporządkowuje:
   - $a \in F_0 \Rightarrow a_I \in D$
   - $x \in V \Rightarrow val(x) \in D$
   - $f \in F_m \Rightarrow f_I : D^m \to D, m = 1, 2, \ldots$
   - $q \in P_m \Rightarrow q_I : D^m \to \{tt, ff\}, m = 1, 2, \ldots$ (relacja na D^m)

Programy

Niech SP – zbiór specyfikacji.

Specyfikacja – napis formułujący pewne stwierdzenie o programie. Aby związać języki programowania z językiem logiki I rzędu przyjmujemy:

1. $V = Ide$.

2. w konsekwencji przyjmujemy $Val = \Sigma$. Czyli $\Sigma = [Ide \to D]$.
   A więc $I(x)(\sigma) = \sigma(x)$ dla $x \in V = Ide$.

3. Rozszerzamy interpretację I na termy, formuły, programy i specyfikacje względem dowolnego stanu $\sigma \in \Sigma$ w sposób następujący:

(a) $\alpha \in FR \cup SP$ jest prawdziwa w interpretacji I i w stanie σ, jeżeli $\alpha_I(\sigma) = tt$

(b) I jest modelem $\alpha \in FR \cup SP$, jeżeli α jest prawdziwa dla każdego $\sigma \in \Sigma$.

(c) I jest modelem dla zbioru formuł lub specyfikacji, jeżeli jest modelem dla każdej formuły lub specyfikacji tego zbioru.

(d) Rozważamy logikę Hoare'a nad zadanym modelem $\mathcal{M} = \{\alpha \in FR : \models_I \alpha\}$.

System wnioskowania o programach

$$H = (A_{SP}, R_{SP})$$

Wymagamy, aby

1. Własności, które możemy udowodnić w systemie były prawdziwe w rzeczywistości.

2. Fakty rzeczywiście prawdziwe o programach (opisywane przez specyfikacje) dały się w tym systemie udowodnić.

Związek między systemem wnioskowania o programach dla języka programowania a jego semantyką – semantyka jest modelem dla tego systemu.
Rozważmy: Język programów iteracyjnych

$$c \;::=\; \underline{\text{skip}} \mid v := t \mid c; c' \mid \underline{\text{if}} \;\; b \;\underline{\text{then}} \;\; c \;\underline{\text{else}} \;\; c' \mid \underline{\text{while}} \;\; b \;\underline{\text{do}} \;\; c$$

gdzie $t \in TR$ i b jest formułą otwartą.
Rozszerzenie dowolnej interpretcji I na instrukcje języka programowania definiujemy:
Dla każdej instrukcji $c \in Com$ i stanu $\sigma \in \Sigma$

$$I(c)(\sigma) = \mathcal{C}[[c]]\sigma$$

lub równoważnie

$$I(c)(\sigma) = \mathbf{exec}(\mathbf{c}, \sigma)$$

Częściowa poprawność programów

System Hoare'a dla programów iteracyjnych

1. skip

   $\{\alpha\}\underline{\text{skip}}\{\alpha\}$

2. :=

   $\{\alpha(t/v)\}v := t\{\alpha\}$

3. ;

   $$\frac{\{\alpha\}c_1\{\beta\},\ \{\beta\}c_2\{\gamma\}}{\{\alpha\}c_1; c_2\{\gamma\}}$$

4. if

$$\frac{\{\alpha \wedge b\}c_1\{\beta\},\ \{\alpha \wedge \neg b\}c_2\{\beta\}}{\{\alpha\}\underline{\text{if}}\ b\ \underline{\text{then}}\ c_1\ \underline{\text{else}}\ c_2\{\beta\}}$$

5. while

$$\frac{\{\alpha \wedge b\}c\{\alpha\}}{\{\alpha\}\underline{\text{while}}\ b\ \underline{\text{do}}\ c\{\alpha \wedge \neg b\}}$$

6. **Reguła konsekwencji**

$$\frac{\alpha \Rightarrow \alpha_1, \{\alpha_1\}c\{\beta_1\}, \beta_1 \Rightarrow \beta}{\{\alpha\}c\{\beta\}}$$

Dla języka PJP mamy:

1. Aexp=TR

2. Bexp= zbiór formuł otwartych

3. Interpretacja – arytmetyka liczb całkowitych.

Przykład 5

Niech $c \equiv x := x + 1;\ y := y + 1$

Chcemy udowodnić $\{x = y\}\ c\ \{x = y\}$.

Dowód:

z **(:=)** mamy

$\{(x = y)[y + 1/y]\}\ y := y + 1\ \{x = y\}$
$\{(x = y + 1\}\ y := y + 1\ \{x = y\}$

z **(:=)** mamy

$\{(x = y + 1)[x + 1/x]\}\ x := x + 1\ \{x = y + 1\}$
$\{x + 1 = y + 1\}\ x := x + 1\ \{x = y + 1\}$

z **(;)** mamy

$\{x + 1 = y + 1\}\ x := x + 1;\ y := y + 1\ \{x = y\}$

z **reguły konsekwencji** i faktu

$(x = y) \Rightarrow (x + 1 = y + 1)$ mamy

$\{x = y\}\ x := x + 1;\ y := y + 1\ \{x = y\}$ ∎

Przykład 6

$c \equiv \underline{\text{if}}\ x \geq y\ \underline{\text{then}}\ m := x\ \underline{\text{else}}\ m := y$

Pokazać, że $\{tt\}\ c\ \{m = max(x, y)\}$.

Dowód:

z **(:=)**

$\{y = max(x, y)\}\ m := y\ \{m = max(x, y)\}$

z **(:=)**

$\{x = max(x, y)\}\ m := x\ \{m = max(x, y)\}$

z **reguły konsekwencji** i faktu

$(x \geq y) \Rightarrow (x = max(x, y))$ mamy

$(x \geq y)\ m := x\ \{m = max(x, y)\}$

z **reguły konsekwencji** i faktu

$(\neg(x \geq y)) \Rightarrow (y = max(x, y))$ mamy

$(x < y)\ m := y\ \{m = max(x, y)\}$

z **if** dostajemy

$\{tt\}\ c\ \{m = max(x, y)\}$. ∎

Przykład 7

$w \equiv \underline{\text{while}}\ x > 0\ \underline{\text{do}}\ y := x * y;\ x := x - 1$

Pokazać, że $\{x = n \wedge n \geq 0 \wedge y = 1\}\ w\ \{y = n!\}$

Weźmy jako niezmiennik

$$\mathcal{I} \equiv (y * x! = n! \wedge x \geq 0)$$

Pokażemy, że $\mathcal{I}$ jest rzeczywiście niezmiennikiem, tzn.

$\{\mathcal{I} \wedge b\}\ c\ \{\mathcal{I}\}$

z **(:=)**

$\{y * (x - 1)! = n! \wedge x - 1 \geq 0\}\ x := x - 1\ \{y * x! = n! \wedge x \geq 0\}$

z **(:=)**

$\{x * y * (x-1)! = n! \wedge x - 1 \geq 0\}\ y := x * y\ \{y * (x-1)! = n! \wedge x - 1 \geq 0\}$

z **reguły konsekwencji** i faktu

$(y * x! = n! \wedge x \geq 0 \wedge x > 0) \Rightarrow (x * y * (x-1)! = n! \wedge x - 1 \geq 0)$

oraz **(;)** mamy

$\{\mathcal{I} \wedge b\}\ c\ \{\mathcal{I}\}$

z **while** dostajemy

$\{y * x! = n! \wedge x \geq 0\}\ w\ \{y * x! = n! \wedge x \geq 0 \wedge x \leq 0\}$

z **reguły konsekwencji** i faktu

$$(x = n \wedge n \geq 0 \wedge y = 1) \Rightarrow (y * x! = n! \wedge x \geq 0)$$

oraz faktu

$$(y * x! = n! \wedge x \geq 0 \wedge x \leq 0) \Rightarrow (y * x! = n! \wedge x = 0) \Rightarrow (y = n!)$$

dostajemy

$$\{x = n \wedge n \geq 0 \wedge y = 1\}\ w\ \{y = n!\}$$ ∎

Własności systemu H
Poprawność

Definicja 14
System T wnioskowania o programach nazywamy *niesprzecznym* (lub *poprawnym*) względem klasy interpretacji INT, jeżeli dla każdej interpretacji $I \in INT$ i każdej specyfikacji Φ

$$\vdash_{TI} \Phi \Rightarrow \models_I \Phi.$$

■

Przy przyjętych założeniach mamy

Twierdzenie 15
System wnioskowania H jest poprawny. ■

Zupełność

Definicja 16
System T wnioskowania o programach nazywamy *zupełnym* względem klasy interpretacji INT, jeżeli dla każdej interpretacji $I \in INT$ i każdej specyfikacji Φ

$$\models_I \Phi \Rightarrow \vdash_{TI} \Phi$$

■

Wyrażalność FR względem *PR*

najsłabszy warunek wstępny:

$$wp_I(c, \beta) = \{\sigma \in \Sigma : \text{jeśli } I(c)(\sigma) \text{ jest zdefiniowane, to } \models_I \beta(I(c)(\sigma))\}.$$

Interpretacja I jest *wyrażalna w FR względem PR*, jeżeli dla każdego $c \in PR$ istnieje $\alpha \in FR$ taka, że $\{\sigma : \models_I \alpha(\sigma)\} = wp_I(c, \beta)$.

Twierdzenie 17
System wnioskowania H jest relatywnie zupełny w klasie interpretacji wyrażalnych dla (FR, PR). ∎

Niech I będzie standardową interpretacją dla PJP, a FR językiem formuł generowanym przez wyrażenia języka PJP. Wtedy

Twierdzenie 18
Interpretacja I jest wyrażalna w FR względem PJP. ∎

Całkowita poprawność programów

Formalnie:
Dla interpretacji I i dowolnego stanu $\sigma \in \Sigma$

1. *Poprawność częściowa*
   $I(\{\alpha\}\, c\, \{\beta\})(\sigma) = tt \stackrel{df}{\equiv}$ jeżeli $\models_I \alpha(\sigma)$ i
   $I(c)(\sigma)$ jest zdefiniowane, to $\models_I \beta(I(c)(\sigma))$

2. *Poprawność całkowita*
   $I(\{\alpha\}\, c\, \{\beta\})(\sigma) = tt \stackrel{df}{\equiv}$ jeżeli $\models_I \alpha(\sigma)$, to
   $I(c)(\sigma)$ jest zdefiniowane i $\models_I \beta(I(c)(\sigma))$

Poprawność całkowita

$=$

Poprawność częściowa

$+$

Własność stopu

Całkowita poprawność programów

System Hoare'a dla programów iteracyjnych

Aksjomaty i reguły 1.–4.,6. oraz

5'. while

$$\frac{\begin{array}{c}\{\alpha \wedge b\}\ c\ \{\alpha\} \\ \{\alpha \wedge b \wedge t = z\}\ c\ \{t < z\} \\ \alpha \Rightarrow t \geq 0\end{array}}{\{\alpha\}\underline{\text{while}}\ b\ \underline{\text{do}}\ c\{\alpha \wedge \neg b\}}$$

gdzie t jest wyrażeniem całkowitoliczbowym, a z zmienną nie występującą w α, b, t ani c. Wyrażenie t nazywamy *funkcją ograniczającą*.

Uwaga 2

Chociaż aksjomaty i reguły 1.-4. i 6. mają tę samą postać, co w systemie poprawności częściowej to występujące w nich specyfikacje $\{\alpha\}\ c\ \{\beta\}$ mają inne znaczenie, mianowicie są interpretowane w sensie poprawności całkowitej.

Metody wyznaczania niezmiennika

"Teoria balona" Griesa.
$\{\alpha\}$<u>while</u> b <u>do</u> $c\{\beta\}$

Niezmiennik $\mathcal{I}$ jest prawdziwy przed i po każdej iteracji.

Tak więc, zbiór stanów reprezentowany przez $\mathcal{I}$ musi zawierać zarówno stany początkowe jak i stany końcowe.

Należy więc tak osłabić warunek końcowy β, aby objął on także stany początkowe ("nadmuchanie balona").

Każda iteracja powinna "wypuszczać trochę powietrza z balona" tak, aby po ostatniej zredukował się z powrotem do β.

Osłabianie formuły β.

1. **Usunięcie koniunktu**.
   Np. $A \wedge B \wedge C$ może być osłabiony do $A \wedge B$. Wtedy $\neg C$ przyjmujemy za warunek pętli.

2. **Zastąpienie stałej przez zmienną.**
   Np.x=min b[1..10] może być osłabione do x=min b[1..i]$\wedge 1 \leq i \leq 10$, gdzie i jest nową zmienną.

3. **Rozszerzenie zakresu zmiennej.**
   Np. $5 \leq i \leq 10$ może być osłabiony do $0 \leq i \leq 10$.

4. **Dodanie dysjunktu**.
   Np. A może byc osłabiony do $A \vee B$.

Przykład 8

Rozważmy program *DIV* obliczający iloraz całkowity i resztę z dzielenia liczb całkowitych nieujemnych x i y.

$q := 0;\ r := x;$
$\underline{\text{while}}\ r \geq y\ \underline{\text{do}}\ (r := r - y;\ q := q + 1)$

Rozważmy formuły:

$$\{x \geq 0 \wedge y \geq 0\} DIV \{q * y + r = x \wedge 0 \leq r < y\} \tag{4}$$

$$\{x \geq 0 \wedge y > 0\} DIV \{q * y + r = x \wedge 0 \leq r < y\} \tag{5}$$

Zauważmy, że jeśli udowodnimy prawdziwość specyfikacji (4), to ponieważ

$$(x \geq 0 \wedge y > 0) \Rightarrow (x \geq 0 \wedge y \geq 0),$$

więc z **reguły konsekwencji** otrzymamy prawdziwość (5).

Dowód dla poprawności częściowej

Weźmy jako niezmiennik pętli

$$\mathcal{I} \equiv q * y + r = x \wedge r \geq 0$$

Udowodnimy następujące trzy fakty:

$$\{x \geq 0 \wedge y \geq 0\}\ q := 0;\ r := x;\ \{\mathcal{I}\} \tag{6}$$

$$\{\mathcal{I} \wedge r \geq y\}\ r := r - y;\ q := q + 1\ \{\mathcal{I}\} \tag{7}$$

$$\mathcal{I} \wedge \neg(r \geq y) \Rightarrow (q * y + r = x \wedge 0 \leq r < y) \tag{8}$$

Ad.(6) Stosując dwukrotnie aksjomat dla **(:=)** mamy

$$\{q * y + x = x \wedge x \geq 0\}\ r := x\ \{\mathcal{I}\}$$

oraz

$$\{0 * y + x = x \wedge x \geq 0\}\ q := 0\ \{q * y + x = x \wedge x \geq 0\}$$

więc z **(;)** otrzymujemy

$$\{0 * y + x = x \wedge x \geq 0\}\ q := 0;\ r := x\ \{\mathcal{I}\}$$

co wraz z faktem

$$(x \geq 0 \wedge y \geq 0) \Rightarrow (0 * y + x = x \wedge x \geq 0)$$

daje (6).

Ad.(7) Stosując dwukrotnie aksjomat dla **(:=)** mamy

$$\{(q + 1) * y + r = x \wedge r \geq 0\}\ q := q + 1\ \{\mathcal{I}\}$$

oraz

$$\{(q+1)*y+(r-y)=x \wedge (r-y) \geq 0\}$$
$$r := r-y$$
$$\{(q+1)*y+r=x \wedge r \geq 0\}$$

więc z **(;)** otrzymujemy

$$\{(q+1)*y+(r-y)=x \wedge (r-y) \geq 0\}$$
$$r := r-y;\ q := q+1$$
$$\{\mathcal{I}\}$$

co wraz z faktem

$$(\mathcal{I} \wedge r \geq y) \Rightarrow ((q+1)*y+(r-y)=x \wedge (r-y) \geq 0)$$

daje (7).

Ad.(8) Oczywiste.

Ad.(4) (4) otrzymujemy z (6), (7) i (8) stosując regułę (while) oraz (;).

Dowód dla poprawności całkowitej

Udowodnimy specyfikację (5)*. Weźmy jako niezmiennik pętli

$$\mathcal{I}' \equiv \mathcal{I} \wedge y > 0$$

i niech funkcją ograniczającą będzie

$$t \equiv r$$

Mamy pokazać, że

$$\{x \geq 0 \wedge y > 0\}\ q := 0;\ r := x;\ \{\mathcal{I}'\} \tag{9}$$

$$\{\mathcal{I}' \wedge r \geq y\}\ r := r - y;\ q := q + 1\ \{\mathcal{I}'\} \tag{10}$$

*Specyfikacja (4) w tym przypadku nie zachodzi,zauważmy, że program się pętli dla $y = 0$.

$$\{\mathcal{I}' \wedge r \geq y \wedge r = z\} \tag{11}$$
$$r := r - y;\ q := q + 1\ \{r < z\}$$

$$\mathcal{I}' \Rightarrow r \geq 0 \tag{12}$$

$$\mathcal{I}' \wedge \neg(r \geq y) \Rightarrow (q * y + r = x \wedge 0 \leq r < y) \tag{13}$$

Dowody (9), (10) i (13) są analogiczne do (6), (7) i (8).

Ad.(11) z **(:=)** i **(;)** mamy

$$\{r < z\}\, q := q + 1\, \{r < z\}$$

i

$$\{(r - y) < z\}\, r := r - y\, \{r < z\}.$$

Ale

$$(\mathcal{I} \wedge y > 0 \wedge r \geq y \wedge r = z) \Rightarrow ((r - y) < z)$$

więc (11) zachodzi z reguły konsekwencji.

Ad.(12) Oczywiste.

Ad.(5) (5) otrzymujemy z (9)– (13) stosując regułę (while) oraz (;). ∎

Twierdzenie 19
Problem stopu jest problemem nierozstrzygalnym. ∎

Przykład programu dla którego nie znaleziono dotąd dowodu, że się kończy ani kontrprzykładu:

while $x \neq 1$	do
if x *parzyste*	then $x := x/2$
	else $x := 3 * x + 1$