Wstęp do programowania

Poprawność programów

Jak pokazać, że program wykonuje to, czego się po nim spodziewamy?

- Używamy w tym celu specyficznej logiki – logiki programów
- W tej logice fragmenty programów są elementami formuł logiicznych

Logika Hoare'a

- Formuły są typu

{α}P{β}, gdzie P jest instrukcją, zaś α i β są formułami klasycznymi, opisującymi stan zmiennych (czyli zależnymi od wartościowania) przed wykonaniem i po wykonaniu programu P.

Czytamy tę formułę następująco:

{α}P{β}

- Jeśli program P zacznie się wykonywać w momencie, w którym zmienne spełniają warunek α i jeśli zakończy swoje działanie, to w stanie końcowym jego zmienne będą spełniać warunek β.
- Gdy jest spełniony powyższy warunek, mówimy że program P jest *częściowo poprawny* względem α i β.

Reguły wnioskowania

- Reguły wnuioskowania określają, na podstawie jakich przesłanek można wyciągać jakie wnioski.
- Zapisujemy je w postaci „ułamka”

$$\frac{\text{Przesłanki oddzielone przecinkami}}{\text{Wniosek}}$$

Instrukcja pusta

- Dla instrukcji pustej reguła jest prosta: jeśli w stanie początkowym wiemy, że formuła α jest prawdziwa, to po wykonaniu tej instrukcji też będzie prawdziwa.

$$\frac{}{\{\alpha\}\ \varepsilon\ \{\alpha\}}$$

Instrukcja przypisania

- Do sformułowania reguły dla instrukcji przypisania umówmy się, że przez jeśli α(z) jest formułą jednej zmiennej z, to przez α(E) będziemy rozumieli formułę, w której wszystkie wystąpienia zmiennej z zastąpiono przez E.

$$\frac{}{\{\alpha(E)\}\ z:=E\ \ \{\alpha(z)\}}$$

Instrukcja złożona

- Przy instrukcji złożonej zakładamy istnienie skończonej liczby stanów pośrednich między kolejnymi wykonaniami isntrukcji składowych

$$\frac{\alpha=\alpha_0,\{\alpha_0\}P_1\{\alpha_1\},\ldots,\{\alpha_{n-1}\}P_n\{\alpha_n\},\alpha_n=\beta}{\{\alpha\}\text{begin } P_1,\ldots, P_n \text{ end}\{\beta\}}$$

Instrukcja warunkowa

$$\frac{\{\alpha \wedge B\}P\{\beta\}, (\alpha \wedge \sim B) \rightarrow \beta}{\{\alpha\}\text{if } B \text{ then } P\{\beta\}}$$

$$\frac{\{\alpha \wedge B\}P_1\{\beta\}, \quad \{\alpha \wedge \sim B\}P_2\{\beta\}}{\{\alpha\}\text{if } B \text{ then } P_1 \text{ else } P_2\{\beta\}}$$

Instrukcja pętli

- W regule wnioskowania dla instrukcji pętli pojawi się tajemnicze N – jest to nizemiennik pętli, czyli formuła zawsze prawdziwa na początku każdego obrotu pętli. Czym ona jest konkretnie – nie wiadomo z góry, musimy ją zgadnąć.

$$\frac{\alpha \rightarrow N, \quad \{N \wedge B\} P \{N\}, \; N \wedge \sim B \rightarrow \beta}{\{\alpha\} \text{while B do P} \{\beta\}}$$

Poprawność pętli - indukcja

- W rzeczywistości poprawność pętli dowodzimy indukcyjnie.
- Warunek $\alpha \rightarrow N$ oznacza, że na samym początku pętli N jest prawdziwe.
- Warunek $\{N \wedge B\}P\{N\}$ oznacza, że jedno wykonanie obrotu pętli nie może nam popsuć niezmiennika.
- Warunek $N \wedge \sim B \rightarrow \beta$ oznacza, że gdy skończymy wykonywać pętlę, mamy zagwarantowane spełnienie β. (nota bene N też jest spełnione!)

Dodatkowe dwie reguły

- Wzmacnianie założenia i osłabianie tezy

$$\frac{\alpha \rightarrow \gamma, \quad \{\gamma\} P \{\beta\}}{\{\alpha\} P \{\beta\}}$$

$$\frac{\{\alpha\} P \{\beta\}, \beta \rightarrow \gamma,}{\{\alpha\} P \{\gamma\}}$$

Przykład

- {i>=0} i:=i+1 {i>0}

jest formułą prawdziwą, bo dla każdych danych, jeśli tylko i jest nieujemne, to po dodaniu jedynki i będzie dodatnie.

Ogólnie może się zdarzyć, że zmienna i po wykonaniu instrukcji i:=i+1 przyjmie wartość <=0, ale nie w przypadku, gdy bezpośrednio przed jej wykonaniem miała wartość nieujemną, co jest zagwarantowane przez α = (i>=0).

Przykład nieco ciekawszy

```
{true}
x:=x0; y:=y0; z:=0;
while x<>0 do begin
   if odd(x) then z:=z+y;
   x:=x div 2;
   y:=y+y;
end
{z=f(x0,y0)}
```

Algorytm mnożenia chłopów rosyjskich

```
{true}
x:=x0; y:=y0; z:=0; {x=x0 i y=y0 i z=0}
while x<>0 do begin {z+xy=x0*y0}
   if odd(x) then z:=z+y;
   x:=x div 2;
   y:=y+y;
end
{z=x0*y0}
```

Algorytm mnożenia chłopów rosyjskich

{true}

x:=x0; y:=y0; z:=0; {x=x0 i y=y0 i z=0}

while x<>0 do begin {z+xy=x0y0}

if odd(x) then z:=z+y;

{(odd(x) ∧ z-y+xy=x0y0) ∨ (~odd(x)∧ z+xy=x0y0)}

x:=x div 2;

{z+2xy=x0y0}

y:=y+y;

{z+xy=x0y0}

end

{z=x0y0}

Przykład jeszcze ciekawszy

```
{true}
x:=x0; y:=y0; z:=1;
while x<>1 do begin
   if odd(x) then z:=z*y;
   x:=x div 2;
   y:=y*y;
end
{z=f(x0,y0)}
```

Potęgowanie binarne (binpower)

{true}

x:=x0; y:=y0; z:=1; {x=x0 i y=y0 i z=1}

while x<>0 do begin {$zy^x=y0^{x0}$}

  if odd(x) then z:=z*y;

{$(odd(x) \wedge (z/y)y^x=y0^{x0}) \vee (\sim odd(x) \wedge zy^x=y0^{x0})$}

  x:=x div 2;

  {$zy^{2x}=y0^{x0}$}

  y:=y*y

  {$zy^x=y0^{x0}$}

end

{$z=y0^{x0}$}

Co dla danych ujemmych?

- W przypadku mnożenia chłopów rosyjskich, jeżeli y<0, to nie ma problemu, ale jeśli x<0, y>0, to algorytm nie obliczy niewątpliwie illoczynu: będzie dodawał do z wyłącznie dodatnie wielokrotności y.
  - x y z
  - -11 2 0
  - -6 4 2
  - -3 8 2
  - -2 16 10
  - -1 32 10
  - ...

Problem z zapętleniem

- Nie chodzi nawet o to, że program się zapętli, ale że wyszło nam że jest poprawny (choć tylko częściowo), mimo że się zapętlił.
- Warto rozważać mocniejszą wersję poprawności, zakładającą że nie tylko ma być program częściowo poprawny, lecz także zakończyć działanie i to bez błędu.