Algoritmusok tervezése és elemzése
2012. 02. 22.

Még 2 hét gyakorlat és ZH.
Feladatok a ZH-n:

- Algoritmus és ciklus-invariáns adott: dekoráció.
- Algoritmus (utasítások) adottak: ciklus-invariáns meghatározása.

Totális Hoare-kalkulus (**[P] C [Q]**) // Más a jelölés! (nem {}, hanem [])
Ha P teljesül, akkor C eredményesen befejeződik és a végén teljesül Q.

Totális helyességnél csak a <u>ciklus-szabály más</u>:

[P ∧ B ∧ U = N] C [P ∧ U < N], P ∧ B ∋ U > 0 // ∋ = implikáció
--- // N egy friss változó, ami nem változhat
[P] while B do C od [P ∧ ¬B] // Az **U = N** közötti reláció egy hasonlítás

U = egész értékű term, amely felülről becsli a ciklus hosszát (avagy ez egy ciklus-számláló).
Ha B teljesül, akkor U > 0. C végrehajtása során U értéke csak csökkenhet és csökken is.

Lnko:

$$(X,Y) = \begin{cases} lnko(X,Y), ha\ X > 0,\ Y > 0 \\ w, egyébként \end{cases}$$

(X,X) = X
(X,Y) = (Y,X)
(X+Y,Y) = (X,Y)

[a > 0 ∧ b > 0] X := a; Y := b; while X ≠ Y do
  if X > Y then X := X-Y
  else Y := Y-X fi
od [X = (a,b)]

Dekorációs feladat totális helyességgel:

1.	[a > 0 ∧ b > 0]	P = **(X,Y) = (a,b)**	Ciklus-invariáns
23.	[<u>(a,b) = (a,b)</u>]	U = **X+Y**	Ciklus-számláló (felülről becsül)
2.	X := a;	A ciklus-számlálót is manuálisan határozzuk meg úgy, hogy	
22.	[<u>(X,b) = (a,b)</u>]	minden ciklusban teljesüljön a kifejezés.	
3.	Y := b;		
12.	[(X,Y) = (a,b)]	// while feletti sorban: P (ciklus-invariáns)	
4.	while X ≠ Y do		
13.	[(X,Y) = (a,b) ∧ X ≠ Y **∧ X+Y = n**]	// while alatti sorban: (P ∧ B **∧ U = N**) **(=> P : IF P-je)**	
5.	if X > Y then		

16. [(X,Y) = (a,b) ∧ X ≠ Y ∧ X+Y = n **∧ X > Y**] // if alatti sorban: P **∧ B** (az IF saját P-je és B-je)

21. [(X-Y,Y) = (a,b) ∧ (X-Y)+Y < n]

6. X := X-Y

17. [(X,Y) = (a,b) ∧ X+Y < n] // Q (meghatározása hamarabb történik mint leírása)

7. else

18. [(X,Y) = (a,b) ∧ X ≠ Y ∧ X+Y = n **∧ X ≤ Y**] // else alatti sorban: (P **∧ ¬B**) (minden az IF-é)

20. [(X,Y-X) = (a,b) ∧ X+(Y-X) < n]

8. Y : = Y-X

19. [(X,Y) = (a,b) ∧ X+Y < n] // Q (a hozzá tartozó if alatti sorból jön)

9. fi

14. [(X,Y) = (a,b) ∧ **X+Y < n**] // od feletti sorban: (P **∧ U < N**) **(=> Q, fi alatti)**

10. od

15. [(X,Y) = (a,b) **∧ X = Y**] // od alatti sorban: P **∧ ¬B**

11. [X = (a,b)]

Külön fel kell írnunk a „**P ∧ B Ǝ U > 0**” formulát is, mert a totális helyesség definíciója ezt előírja: (X,Y) = (a,b) ∧ (X≠Y) Ǝ (X+Y > 0) // Természetesen P itt a ciklus P-je.

Az (1,23), (16,21), (18,20) és (15,11) számpárokkal jelölt sorok között implikáció áll fenn. A lényeg, hogy két egymás után közvetlenül következő formula között implikáció áll fenn, amelyet a Hoare-kalkulus helyessége bizonyít.

A ciklus-számláló megértéséhez nézzünk néhány feladatot, amelyben csak a ciklus-invariánst és a ciklus-számlálót határozzuk meg.

Prímosztók száma multiplicitással:

α(12) = 3 (2*2*3)

$$\alpha(X) = \begin{cases} valami, ha\ X > 0 \\ X,\ ha\ X \leq 0 \end{cases}$$

[n > 0] X := n; Y := 0; P := 2; while X > 1 do
if P|X then X := X/P; Y := Y+1 else P := P+1 fi
od [Y = α(n)]

Az algoritmus ciklus-invariánsa a következő:

α(X) + Y = α(n) ∧ ∀k(k < P Ǝ K†X) // ahol „Ǝ” az implikáció, „†” a nem osztója jelölés

Természetesen ilyen beteg feladat nem lesz a zh-n, amúgy sem tudom miért ez a válasz. ☺

Határozzuk meg a ciklus-számlálót, teszteljük több bemenetre. A ciklus-számlálót is manuálisan kell „eltalálnunk”. Először megadunk egy lehetséges kifejezést, majd megnézzük, hogy minden ciklusban teljesül-e, hogy folyamatosan csökken.

n	X	Y	P	**X-P**
63	63	0	2	61
	63	0	3	60
	21	1	3	18
	7	2	3	4
	7	2	4	3
	7	2	5	2
	7	2	6	1
	7	2	7	**0**
	1	3	7	-6

n	X	Y	P	**X-P**
72	72	0	2	70
	36	1	2	34
	18	2	2	16
	9	3	2	7
	9	3	3	6
	3	4	3	**0**
	1	5	3	-2

Mivel az algoritmusunk kilép a ciklusból, **ha X nem nagyobb, mint 1**, ezért olyan ciklus-számlálót kell meghatároznunk, amely minden ciklusban teljesíti a feltételt, hogy X nagyobb, mint 1 (kivéve az utolsó ciklusban, mert akkor már úgy sem kezdünk új ciklusba). A ciklus szó alatt azokat az utasításokat értem, amelyek többször is lefutnak egy while ciklus végrehajtása alatt.

Láthatjuk, hogy az X-P kifejezés nem teljesen jó nekünk, mert az utolsó előtti ciklusoknál elér egy olyan értéket, amelyre már nem igaz, hogy nagyobb, mint 1. Ezért nem futna le az utolsó kör és az algoritmusunkban található ciklus ciklus-számlálója nem lenne helyes. Ügyeskednünk kell, amely minden esetben máshogy zajlik. Esetünkben ha az X-P kifejezéshez hozzáadunk egyet, azaz **(X-P)+1** –t tekintjük a ciklus-számlálónak, akkor minden lépésben teljesül a ciklus feltétele (X > 1).

Redukálós feladat újratöltve:

Dom = V*

[T] Y := X; Z := λ; while Y ≠ λ do
  if (Y) = f(Z) then Y := t(Y) else Z := f(Y)Z fi;
od [Z = red(X)$^{-1}$]

X	**Y**	**Z**	
aabccc	aabccc	λ	Ciklus-invariáns: **red(Y^{-1} Z) = red(X)$^{-1}$ Λ Z = red(Z)**
	aabccc	a	Erről a képletről korábban már szó volt, nem fejtem ki.
	abccc	a	Ciklus-számláló: **| Y | - | Z | + | X |**
	bccc	a	(ahol | Y | nem más, mint Y karakterszáma)
	bccc	ba	
	ccc	ba	Eredetileg | Y | - | Z | -t tekintettük ciklus-számlálónak,
	ccc	cba	de úgy nem teljesült a ciklus feltétele a megfelelő körökben.
	cc	cba	
	c	cba	Ciklus-számláló értékek „**+ | X |**” nélkül: 6,5,4,3,2,1,**0,-1,-2,**-3
	λ	cba	Ciklus-számláló értékek „**+ | X |**” -el: 12,11,10,9,8,7,6,5,4,3

Random feladat:

[T] Y := 0; while X > 0 do X := X-Y; Y := 1-Y od [T]

X	Y
4	0
4	1
3	0
3	1
2	0
2	1
1	0
1	1
0	0

Mivel X értékéről semmit nem tudunk az algoritmus kezdetekor, ezért **csak Y értékével írhatunk fel** ciklus-invariánst.

Ciklus-invariáns: **(Y = 0) V (Y = 1)** // Máshogy nem lehet kifejezni

Ciklus-számláló: 2X-Y

A megalkotott ciklus-számlálót alkalmazva **a ciklus-számláló értéke az egyes ciklusokban** rendre: 8,7,6,5,4,3,2,1,0

Ez kielégíti a számláló monoton csökkenését.