Repaso Practica 4 Ejercicios

Ejercicio 1.

Se verificó el programa que calcula el factorial de x > 0, con la especificación (x > 0, y = x!).

a. Justificar por qué esta especificación no es correcta.

   Es incorrecta porque hay programas que no calculan el factorial que aun así la satisfacen por ejemplo:

   Sp :: if x > 0 then x = 1; y := 1 fi

b. Proponer una que sí lo sea y que además establezca que el valor de x al final sea el mismo que al comienzo.

   (x = X ^ x > 0, x = X y = |X|!)

c. ¿Podría agregarse a la especificación que el valor de x no se altere a lo largo de todo el programa? Justificar.

   No, porque la especificación es un par de aserciones (p,q) en donde p es el conjunto de estados iniciales de S y q es el conjunto de estados finales de S, es decir p es la precondición y q la postcondición de S. Esta especificación garantiza el cumplimiento de ciertos estados iniciales y ciertos estados finales, pero no de estados a lo largo del programa. Pasandolo a este programa, la especificación asegura que x termine con el valor X, pero no que mantenga ese valor a lo largo del programa.

Ejercicio 2.

Asumiendo |= {p} S {q}, indicar en cada caso si vale lo afirmado. Justificar las respuestas:

a. Si S termina en un estado que satisface q, entonces su estado inicial satisface p.

   No, no vale. |= {p} S {q} solamente garantiza que sii se parte desde cualquier estado σ que satisface p, si S termina lo hace en un estado σ' que satisface q.

   Contraejemplo:
   Sea la especificación → Φ = (x = X ^ x > 0, y = X).
   Y el programa → S :: y = x
   Si se tiene como estado inicial σ0 con σ0(x) = -1, al finalizar S se tendrá val(π(Sp,σ0)) = σ1 con σ1(y) = -1.

   Se puede observar cómo se satisface q pero no se satisface p.

b. Si S termina en un estado que no satisface q, entonces su estado inicial no satisface p.

   Vale, por definicion se sabe que sii se parte de cualquier estado σ que satisface p, si el programa termina lo hace en un estado σ' que satisface q.

   $(\sigma \models p \wedge val(\pi(S, \sigma)) \neq \perp) \rightarrow val(\pi(S, \sigma)) = \sigma' \models q$

   Si se terminó en un estado σ que no satisface q, entonces, por definición, se partió de un estado inicial σ que no satisfizo p.

c. Si S no termina, entonces su estado inicial no satisface p.

   No vale. Se asume $\models \{p\}\ S\ \{q\}$ que es correctitud parcial, no garantiza terminación. Puede S no terminar pero aun así el estado inicial satisfacer p.

   Contraejemplo:
   Sea la especificación → Φ = (x = 0, x = 0).
   Y el programa → S :: while x := 0 do x := 0 od
   Si se tiene como estado inicial σ0 con σ0(x) = 0

   Se puede observar cómo se satisface p pero el programa aun así no termina.

d. ¿Las respuestas en (a), (b) y (c) son las mismas considerando la fórmula $\models \langle p \rangle\ S\ \langle q \rangle$?

   Las respuestas (a) y (b) si, pero la (c), por definición de correctitud total ($\models \langle p \rangle\ S\ \langle q \rangle$) que dice que sii desde cualquier estado σ que satisface p, S termina en un estado σ' que satisface q.

   $\sigma \models p \rightarrow (val(\pi(S, \sigma)) \neq \perp) \wedge val(\pi(S, \sigma)) = \sigma' \models q)$.

   Osea, si el estado inicial satisface p, entonces el programa va a terminar y en q.

Ejercicio 3.

Indicar en cada caso si vale lo afirmado. Justificar las respuestas:

a. Se cumple $\models \{x = 0\}$ while z = 0 do z := 0 od $\{x = 0\}$.

   Vale, nunca se modifica el valor de la x, por lo que si x inicia con el valor 0, entonces termina con el valor 0, cumpliéndose p y q.

b. Se cumple $\models \langle x = 0 \rangle$ while z = 0 do z := 0 od $\langle x = 0 \rangle$.

c. No vale. Por definición de completitud total sii se parte desde cualquier estado σ que satisface p, S termina en un estado σ' que satisface q. Si se tiene como estado inicial con σ0(x) = 0 y σ0(z) = 0 entonces el programa no terminara, por lo que no se cumple |= ⟨x = 0⟩ Swhile ⟨x = 0⟩.

d. Si se cumple |= {p1 ∧ p2} S {q1 ∧ q2}, entonces |= {p1} S {q1} o bien |= {p2} S {q2}.

   Falso. Con que una de las precondiciones no se cumpla, no se cumple ninguna de las postcondiciones. Es necesario que las dos precondiciones se cumplan para que se cumplan las postcondiciones.

   Contraejemplo

   Sea la especificación → Φ = (x = X ^ x > 0, x = -X ^ y = X).
   Y el programa → S :: if x > 0 then y = x ; x = -x fi
   Si no se cumple x > 0, entonces al finalizar S no se cumplirá ninguna de las dos postcondiciones.

Ejercicio 4.

Sea el siguiente lenguaje de expresiones enteras: e :: 0 | 1 | x | (e1 + e2) | (e1 . e2). Y sea var(e) el conjunto de las variables de e. Se pide definir inductivamente var(e). Por ejemplo: var(0) = ∅.

var(0) = ∅

var (1) = ∅

var (x) = {x}

var(e1 + e2) = var(e1) U var(e2)

var(e1 . e2) = var(e1) U var(e2)

Ejercicio 6.

Supóngase que se agrega al lenguaje PLW la instrucción repeat S until B, con la semántica habitual (se ejecuta S, se evalúa B, si se cumple B se termina la repetición, y si no se cumple B se vuelve al comienzo).

a. Definir la semántica operacional de la instrucción.

   Si σ(B) = verdadero, entonces (repeat S until B, σ) → (S; E, σ).
   = falso, entonces (repeat S until B, σ) → (S; repeat S until B, σ).

b. Proponer una regla de prueba para la misma.

$$\{p\}\ S\ \{q\},\ \{q \wedge \neg B\}\ S\ \{q\}$$

{p} repeat S until {q ^ B}

Ejercicio 7.

Probar por medio del método H las fórmulas de correctitud parcial siguientes, relacionadas respectivamente a programas que calculan el valor absoluto de un número entero y el producto de dos números naturales:

a. {x = X} if x > 0 then y := x else y := – x {y = |X|}, siendo |X| el valor absoluto de X.

precondición = {x = X}
postcondición = {y = |X|}

{p ^ B} S1 {q}, {p ^ ¬B} S2 {q}

{p} if B then S1 else S2 fi {q}

1. {x = |X|} y := x {y = |X|} (ASI)
2. {-x = |X|} y := – x {y = |X|} (ASI)
3. (x = X ^ x > 0) → x = |X| (MAT)
4. (x = X ^ ¬(x > 0)) → -x = |X| (MAT)
5. {x = X ^ x > 0} y := x {y = |X|} (1,3,CONS)
6. {x = X ^ ¬(x > 0)} y := – x {y = |X|} (2,4,CONS)
7. {x = X} if x > 0 then y := x else y := – x {y = |X|} fi (5,6,COND)

b. {x ≥ 0 ∧ y ≥ 0} prod := 0; k := y; while k > 0 do prod := prod + x; k := k – 1 od {prod = x.y}.

p = (x.y = x.k + prod ^ k >= 0)

a) {x ≥ 0 ∧ y ≥ 0} prod := 0; k := y; {x.y = x.k + prod ^ k >= 0}
b) {x.y = x.k + prod ^ k >= 0} while k > 0 do prod := prod + x; k := k – 1 od {x.y = x.k + prod ^ k >= 0 ^ ¬(k > 0)}
c) Aplicando SEC a (a) y (b) y CONS se llega a {x ≥ 0 ∧ y ≥ 0} prod := 0; k := y; while k > 0 do prod := prod + x; k := k – 1 od {prod = x.y}.

Parte a)

1. {x.y = x. y + prod ^ y >= 0} k := y {x.y = x.k + prod ^ k >= 0} (ASI)
2. {x.y = x. y + 0 ^ y >= 0} prod := 0 {x.y = x. y + prod ^ y >= 0} (ASI)
3. {x.y = x. y + 0 ^ y >= 0} prod := 0; k := y {x.y = x.k + prod ^ k >= 0} (1, 2, SEC)
4. (x ≥ 0 ∧ y ≥ 0) → (x.y = x. y + 0 ^ y >= 0) (MAT)
5. {x ≥ 0 ∧ y ≥ 0} prod := 0; k := y {x.y = x.k + prod ^ k >= 0} (3, 4, CONS)

Parte b)

6. {x.y = x.(k-1) + prod ^ (k-1)>= 0} k := k – 1 {x.y = x.k + prod ^ k >= 0} (ASI)
7. {x.y = x.(k-1) + (prod + x) ^ (k-1)>= 0} prod := prod + x {x.y = x.(k-1) + prod ^ (k-1)>= 0} (ASI)
8. {x.y = x.(k-1) + (prod + x) ^ (k-1)>= 0} prod := prod + x ; k := k – 1 {x.y = x.k + prod ^ k >= 0} (6, 7, SEC)
9. (x.y = x.k + prod ^ k >= 0 ^ k > 0) → (x.y = x.(k-1) + (prod + x) ^ (k-1)>= 0) (MAT)
10. {x.y = x.k + prod ^ k >= 0 ^ k > 0} prod := prod + x ; k := k – 1 {x.y = x.k + prod ^ k >= 0} (8,9, CONS)
11. {x.y = x.k + prod ^ k >= 0} while k > 0 do prod := prod + x; k := k – 1 od {x.y = x.k + prod ^ k >= 0 ^ ¬(k > 0)} (10, REP)

Parte c)

12. {x ≥ 0 ∧ y ≥ 0} prod := 0; k := y; while k > 0 do prod := prod + x; k := k – 1 od {x.y = x.k + prod ^ k >= 0 ^ ¬(k > 0)} (5, 11, SEC)
13. (x.y = x.k + prod ^ k >= 0 ^ ¬(k > 0)) → (prod = x.y) (MAT)
14. {x ≥ 0 ∧ y ≥ 0} prod := 0; k := y; while k > 0 do prod := prod + x; k := k – 1 od {prod = x.y} (12, 13, CONS)