TP5

lunes, 24 de mayo de 2021 10:35

Ejercicio 1.

a) Se probó en clase el programa que calcula en una variable *y* el factorial de una variable x > 0, con la especificación (x > 0, y = x!). Justificar por qué esta especificación no es la adecuada para el programa indicado. Además, proponer una especificación adecuada, y en la que el valor de x al final sea el mismo que el del comienzo.

b) La raíz cuadrada entera de un número natural n es el mayor de los números enteros m que cumplen $m^2 \leq n$. Se pide:
   i. Especificar un programa que dado un número natural n como input, obtenga como output su raíz cuadrada entera m, y mantenga al final el valor del input.
   ii. ¿Podría agregarse a la especificación que el valor del input no se altere a lo largo de todo el programa? Justificar.

Símbolos útiles: ⊢λεδΣΓ∀⊂⊆⇒ℒαΩΘ

a) Esta especificación no es correcta, ya que se podría dar un programa que cumpla con la pre-condición y la postcondición y no haga el factorial, por ejemplo S:: x = 1; y = 1.
Una especificación adecuada sería la siguiente:
(x = X, X > 0) S (y = X!, x = X)
Anda a chequearlo

b)
i. $\Phi = (n = N \wedge N \in \mathbb{N}, m \in \mathbb{N} \wedge m^2 \leq N \wedge m > \sqrt{N} - 1)$
ii. No, con el nivel de lógica que utilizamos esto no es posible, solo podemos indicar que el valor sea el mismo al comienzo y al final del programa, pero lo que pase en medio no lo podemos controlar.

Ejercicio 2. Asumiendo $\models \{p\}\ S\ \{q\}$, 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.

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

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

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

a) Falso. Podríamos tener un programa que sea {p} S {q} con p = (n = N ^ N > 0) y q = (x = 2N ^ x > 0), donde el programa multiplique el número x2. En caso de tener un N = -10 no se cumpliría la precondición, el programa haría -10x2, por lo cual terminaría, y el resultado de x sería 100, lo cual cumpliría con la postcondición.

b) Verdadero, por la definición de la correctitud parcial:

(σ |= p ^ val(π(S, σ)) ≠ ⊥) → val(π(S, σ)) |= q
(estadobueno ^ termina) --> resultadobueno
no resultado bueno --> no (estadobueno y termina)
Como sabemos que el programa termina, entonces significa que el estado inicial no satisface la precondición

c) Falso. La definición de la correctitud parcial no nos dice que ocurre si el programa no termina, ni cuales son los valores de la pre o postcondición.

d)

La a) también es falsa, no cambia el hecho de que pueda existir un estado que no satisface p y termina en un estado que satisface q.

La b) también es verdadera, si el estado inicial lo satisface termina bien, por lo que para que el programa termine mal el estado inicial no puede cumplir p.

La c) ahora es verdadera, ya que si el estado inicial satisface p el programa termina, por lo que para que el programa no termina el estado inicial no tiene que satisfacier p.

Ejercicio 3. Indicar en cada caso si vale lo afirmado. Justificar las respuestas:

a) Si p es x = 0, q es x = 1, y S es while z = 0 do z := 0 od, entonces $\models \{p\}\ S\ \{q\}$.

b) Si se cumple $\models \{p_1 \wedge p_2\}\ S\ \{q_1 \wedge q_2\}$, entonces $\models \{p_1\}\ S\ \{q_1\}$ o bien $\models \{p_2\}\ S\ \{q_2\}$.

a) Falso. Si se cumple la pre-condición y el programa termina, entonces no hay seguridad que termine en un estado que satisface q, ya que en el programa nunca se modifica el valor de x para que pase de 0 a 1, por lo que nunca va a terminar con x = 1. La única forma de que el programa termine con x = 1 es que empiece con x = 1, y si empieza con x = 1 no se cumple la pre-condición, por lo que no hay correctitud parcial.

b) nidea
Hacer 2 y 3 con la definición de implicación (que no se cumple con antecedente verdadero y consecuente falso)

b) La propiedad es falsa
Para negar la propiedad hay que demostrar que la implicación de la correctitud parcial es falsa, para eso tenemos que partir de una situación en donde el antecedente sea verdadero (se satisface p1 y p2 y el programa termina) y el consecuente falso.

Entonces, tenemos un estado que satisface p1 y p2, dando por verdadero |= {p1} S {q1}
Si la propiedad fuera verdadera, estas condiciones alcanzaría para demostrar que la postcondición se cumple, pero no es

el caso.
Como se cumple que |= {p1} S {q1}, entonces sabemos 2 cosas, ya que se satisface p1 y el programa termina, se va a satisfacer q1
Sobre la correctitud de p2 y q2 no tenemos información, por lo que si p2 se cumple y el programa termina, no podemos asegurar que q2 se satisface, q2 podría no satisfacerse, ya que la correctitud parcial está asegurada solo para p1 y q1
Si q1 se satisface y q2 no, la postcondición no se satisface, lo que nos daría un ejemplo donde el antecedente de la propiedad de correctitud parcial es verdadero, se satisface la precondición (p1 y p2 se satisfacen) y el programa termina, y el consecuente es falso, porque no se cumple la postcondición (solo se satisface q1, lo que no alcanza para satisfacer la postcondición).

Ejercicio 4. Sea el siguiente lenguaje de expresiones enteras: e :: 0 | 1 | x | (e_1 + e_2) | (e_1 . e_2).
Y sea var(e) el conjunto de las variables de e. Se pide definir inductivamente var(e).
Ayuda: Por ejemplo, var(x) = {x}.

Si var(0) --> ∅
Si var(1) --> ∅
Si var(x) --> x
Si var(e_1 + e_2) --> var(e_1) U var(e_2)
Si var(e_1 . e_2) --> var(e_1) U var(e_2)

Ejercicio 5. Probar que se cumple, para todo estado σ y para todo par de aserciones p, q, que
val(π(S_1, σ)) = val(π(S_2, σ)) si y sólo si |= {p} S_1 {q} ↔ |= {p} S_2 {q}.
Comentario: Para facilitar la notación, se puede utilizar M(S)(σ) en lugar de val(π(S, σ)).

Tenemos que probar la bicondicionalidad:
(M(S_1)(σ) = M(S_2)(σ)) <-> (|= {p} S_1 {q} <-> |= {p} S_2 {q})

Primero probamos
Si (|= {p} S_1 {q} <-> |= {p} S_2 {q}) entonces (M(S_1)(σ) = M(S_2)(σ))

Luego probamos
Si (M(S_1)(σ) = M(S_2)(σ)) entonces (|= {p} S_1 {q} <-> |= {p} S_2 {q})

No puedo bro

Ejercicio 6. Supóngase que se agrega al lenguaje PLW la instrucción repeat S until B, con la semántica habitual. Definir la semántica operacional de dicha instrucción, y extender el método H con una regla para la misma.

Se agrega al lenguaje PLW:
Definición mediante relación de transición -->:
Si (repeat S until B, σ) --> (S ; while B do S od, σ)

Regla para el método H:
{p} S {r}, {r ^ B} S {r}

{p} repeat S until B {p ^ ¬B}

Está bien esta regla?

Ejercicio 7. Probar utilizando H las fórmulas de correctitud siguientes:

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

b) {x≥0 ∧ y≥0} S_{prod} :: prod := 0; k := y; while k>0 do prod := prod + x; k := k – 1 od {prod=x.y}.
Ayuda: S_{prod} calcula en la variable prod el producto entre x e y.

a)
{x = X} Sabs :: if x > 0 then y := x else y := - x {y = |X|}

1. {?} y := x {y >= 0}
2. {?} y := -x {y >= 0}
3. {x >= 0} y := x {y >= 0} (ASI)
4. {-x >= 0} y := -x {y >= 0} (ASI)
5. (x = X ^ (x > 0)) --> {x >= 0}
6. (x = X ^ -(x > 0)) --> {-x >= 0}
7. {x = X ^ (x > 0)} y := x {y >= 0} (3, 5, CONS)
8. {x = X ^ -(x > 0)} y := -x {y >= 0} (4, 6, CONS)
9. {x = X} if x > 0 then y := x else y := -x fi {y >= 0} (7,8,COND)

b)
{x >= 0 ^ y >= 0} Sprod :: prod := 0; k := y; while k > 0 do prod := prod + x; k := k - 1 od {prod = x.y}

El invariante que se propone es:
p = (prod = x.y-k ^ k >= 0)

I) {x >= 0 ^ y >= 0} prod := 0; k := y; {prod = x.y-k ^ k >= 0}
II) {prod = x.y-k ^ k >= 0} while k > 0 do prod := prod + x; k := k - 1 od {prod = x.y}
III) {x >= 0 ^ y >= 0} Sprod {prod = x.y}, por SEC a partir de (a) y (b).

Vamos a probar (I)
1. {?} k := y {prod = x.y-k ^ k >= 0}
2. {prod = x.y-y ^ y >= 0} k := y {prod = x.y-k ^ k >= 0} (ASI)
3. {prod = 0 ^ y >= 0} k := y {prod = x.y-k ^ k >= 0}
4. {?} prod := 0 {prod = 0 ^ y >= 0}
5. {0 = 0 ^ y >= 0} prod := 0 {prod = 0 ^ y >= 0} (ASI)
6. {x >= 0 ^ y >= 0} --> {0 = 0 ^ y >= 0}
7. {x >= 0 ^ y >= 0} prod := 0; k := y; {prod = x.y-k ^ k >= 0} (SEC de 5 y 3, CONS con 6)

Vamos a probar (II)
8. {?} k := k-1 {prod = x.y-k ^ k >= 0}
9. {prod = x.y-(k-1) ^ k-1 >= 0} k := k-1 {prod = x.y-k ^ k >= 0} (ASI)
10. {?} prod := prod + x {prod = x.y-(k-1) ^ k-1 >= 0}
11. {prod + x = x.y-(k-1) ^ k-1 >= 0} prod := prod + x {prod = x.y-(k-1) ^ k-1 >= 0} (ASI)
12. {prod = x.y-k ^ k >= 0 ^ k > 0} --> {prod + x = x.y-(k-1) ^ k-1 >= 0}
13. {prod = x.y-k ^ k >= 0 ^ k > 0} prod := prod + x; k := k - 1 {prod = x.y-k ^ k >= 0} (SEC de 11 y 9, y CONS con 12)
14. {prod = x.y-k ^ k >= 0} while k > 0 do prod := prod + x; k := k - 1 od {prod = x.y-k ^ k >= 0 ^ ¬(k > 0)} (13, REP)
15. {prod = x.y-k ^ k >= 0 ^ ¬(k > 0)} --> {prod = x.y}
16. {prod = x.y-k ^ k >= 0} while k > 0 do prod := prod + x; k := k - 1 od {prod = x.y} (CONS con 15)

Vamos a probar (III)
17. {x >= 0 ^ y >= 0} Sprod {prod = x.y} (SEC de 7 y 16)