Practica 4

Agustina Sol Rojas y Antonio Felix Glorioso Ceretti

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 > 0 ^ x= X, 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, ya que con la especificación solamente se asegura que x termine con el valor X, pero no contemplan los distintos valores que puede asumir la misma 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 vale, pueden existir casos donde tengamos que S termina en un estado que satisface q pero el estado inicial no satisface p.
Dado a lo anterior no se puede asumir que si tenemos un estado final q hubo un estado inicial p. Lo único que se toma como verdadero es que si tengo un estado inicial p y se tiene un S, este terminará en un estado que satisface q.

Contraejemplo:
Dada la especificación (x = X ^ x > 0 ^ y = Y ^ y > 0, x = Y ^ y = X) y el programa Sswap :: z := x; x := y ; y:= z y el estado inicial σ0, con σ0 (x) = -1 y σ0 (y) = -2 y el estado final σ1, con σ1 (x) = -2 y σ1 (y) = -1 se cumple que el estado final satisface q pero el estado inicial no satisface p llegando a una contradicción con el enunciado.

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

Vale, por definición (σ |= p ^ val(π(S, σ)) ≠ ⊥) → val(π(S, σ)) |= q. Si S termina en un estado que no satisface q el consecuente sería falso y como se sabe que el programa termina se cumple val(π(S, σ)) ≠ ⊥ entonces si σ |= p eso haría que el antecedente sea verdadero y el consecuente falso, por lo que σ |≠ p para que la implicación se cumpla.

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

No vale, ya que puede satisfacer p en su estado inicial pero aun así no terminar ya que la correctitud parcial no asegura terminación.

Contraejemplo:
Dada la especificación (x = 0, x = 0) y el programa Swhile :: while x = 0 do x := 0 od y el estado inicial σ0, con σ0 (x) = 0 se cumple que el estado inicial satisface p pero S no termina.

d. ¿Las respuestas en (a), (b) y (c) son las mismas considerando la fórmula |= ⟨p⟩ S ⟨q⟩?

La que no es la misma es la (c), ya que por definición de correctitud total σ |= p → (val(π(S, σ)) ≠ ⊥ ^ val(π(S, σ)) |= q) si S no termina y σ |= p hace que el consecuente sea falso y el antecedente verdadero, haciendo que la implicación sea falsa, por lo tanto σ |≠ p para que la implicación se cumpla. Es decir para que se cumpla la implicación si S no termina entonces el estado inicial no debe satisfacer p.

Ejercicio 3.

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

a. Se cumple |= {x = 0} while z = 0 do z := 0 od {x = 0}.

   Vale. No se modifica el valor de x, si se tiene como estado inicial σ0, con σ0(x) = 0 , se tendrá el estado final σ1, con σ1(x) = 0, cumpliendo con la especificación. Pero puede suceder que σ0(z) = 0 y el programa loopee constantemente aunque no importa ya que se está hablando de correctitud parcial y no de correctitud completa.

b. Se cumple |= ⟨x = 0⟩ while z = 0 do z := 0 od ⟨x = 0⟩.

   No vale. Puede suceder que si se tiene como estado inicial σ0, con σ0(x) = 0 y σ0(z) = 0 se imposibilita el final del programa, que es lo que se busca con la correctitud completa. Por definición |= ⟨p⟩ S ⟨q⟩, es decir σ |= p → (val(π(S, σ)) ≠ ⊥ ^ val(π(S, σ)) |= q). Como el consecuente es falso (no termina el programa) y el antecedente es verdadero (en efecto se satisface la precondición), no se cumple la implicación, por lo tanto no se cumple la correctitud total.

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

   No vale. Puede ocurrir que si deja de valer una de las precondiciones, ninguna de las postcondiciones valga. Es decir, es necesario que las dos precondiciones sean verdaderas para que se cumplan las postcondiciones.

Contraejemplo:

- Dado:
  - p = (x=X ^ x>0) y q = (y=X ^ x=-X)
  - el programa Scontr :: if (x>0) y=x; x=-x fi
  - Si se tiene como el estado inicial σ0, con σ0 (x) = -1 y σ0 (y) = 3 se termina teniendo el estado final σ1, con σ1 (x) = -1 y σ1 (y) = 3, es decir, si deja de valer una de las precondiciones (x>0) pasa a no valer ninguna de las postcondiciones.
  - Si se tiene como el estado inicial σ0, con σ0 (x) = 1 y σ0 (y) = 3 se termina teniendo el estado final σ1, con σ1 (x) = -1 y σ1 (y) = 1, es decir, ahora valen ambas precondiciones y por lo tanto ambas 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) ∪ var(e2)

var(e1 . e2) = var(e1) ∪ var(e2)

Ejercicio 5.

Probar que para todo estado σ y para todo par de aserciones p, q, se cumple:

$$val(\pi(S1, \sigma)) = val(\pi(S2, \sigma)) \text{ si y sólo si } \models \{p\}\ S1\ \{q\} \leftrightarrow \models \{p\}\ S2\ \{q\}$$

Comentario: para facilitar la notación, se puede utilizar M(S)(σ) en lugar de val(π(S, σ)).

Para probar $val(\pi(S1, \sigma)) = val(\pi(S2, \sigma)) \leftrightarrow \models \{p\}\ S1\ \{q\} \leftrightarrow \models \{p\}\ S2\ \{q\}$

a) $val(\pi(S1, \sigma)) = val(\pi(S2, \sigma)) \rightarrow \models \{p\}\ S1\ \{q\} \leftrightarrow \models \{p\}\ S2\ \{q\}$

   (a) Asumiendo que $val(\pi(S1, \sigma)) = val(\pi(S2, \sigma))$, se sabe que si dado un estado inicial $\sigma \models p$ tanto $val(\pi(S1, \sigma))$ como $val(\pi(S2, \sigma)) \models q$, es decir, ambos denotan el mismo estado final. Por lo tanto $\models \{p\}\ S1\ \{q\} \rightarrow \models \{p\}\ S2\ \{q\}$, es decir si el estado inicial satisface p y este es igual en S1 como en S2, como los dos terminan en el mismo estado final este satisfacerá q en S2 por $\models \{p\}\ S1\ \{q\}$.

   (b) Asumiendo que $val(\pi(S1, \sigma)) = val(\pi(S2, \sigma))$, se sabe que si dado un estado inicial $\sigma \models p$ tanto $val(\pi(S1, \sigma))$ como $val(\pi(S2, \sigma)) \models q$, es decir, ambos denotan el mismo estado final. Por lo tanto $\models \{p\}\ S2\ \{q\} \rightarrow \models \{p\}\ S1\ \{q\}$, es decir si el estado inicial satisface p y este es igual en S1 como en S2, como los dos terminan en el mismo estado final este satisfacerá q en S1 por $\models \{p\}\ S2\ \{q\}$.

   (c) Por (a) y (b): $\models \{p\}\ S1\ \{q\} \leftrightarrow \models \{p\}\ S2\ \{q\}$

   (d) Por (a), (b) y (c) $val(\pi(S1, \sigma)) = val(\pi(S2, \sigma)) \rightarrow \models \{p\}\ S1\ \{q\} \leftrightarrow \models \{p\}\ S2\ \{q\}$

b) $\models \{p\}\ S1\ \{q\} \leftrightarrow \models \{p\}\ S2\ \{q\} \rightarrow val(\pi(S1, \sigma)) = val(\pi(S2, \sigma))$

   (a) Asumiendo $\models \{p\}\ S1\ \{q\} \leftrightarrow \models \{p\}\ S2\ \{q\}$. Entonces dado σ, vale:

      a. $\models \{p\}\ S1\ \{q\} = (\sigma \models p\ \ni val(\pi(S1, \sigma))\ \neg\bot) \rightarrow val(\pi(S1, \sigma)) \models q$

      b. $\models \{p\}\ S2\ \{q\} = (\sigma \models p\ \ni val(\pi(S2, \sigma))\ \neg\bot) \rightarrow val(\pi(S2, \sigma)) \models q$

      c. Por a. y b.: $\sigma \models p \rightarrow val(\pi(S1, \sigma)) \models q$ y $\sigma \models p \rightarrow val(\pi(S2, \sigma)) \models q$, entonces $val(\pi(S1, \sigma)) = val(\pi(S2, \sigma))$ dado que sabemos que los dos terminaran en el mismo q, por enunciado, dado su S1 y S2.

   (b) Por todo lo probado en (a) $\models \{p\}\ S1\ \{q\} \leftrightarrow \models \{p\}\ S2\ \{q\} \rightarrow val(\pi(S1, \sigma)) = val(\pi(S2, \sigma))$

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 $\sigma(B)$ = 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.

Regla del repeat (REPU)

$$\frac{\{p\}\ S\ \{q\},\ \{q \wedge \neg B\}\ S\ \{q\}}{\{p\}\ \text{repeat S until B}\ \{q \wedge B\}}$$

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.

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|} (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; {p}

(b) {p} while k > 0 do prod := prod + x; k := k – 1 od {p ^ ¬(k > 0)}

(c) Aplicando SEC a (a) y (b) (p ^ ¬(k > 0)) → prod = x.y, por 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}.

Prueba (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)

Prueba (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)

Prueba (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)