Práctica 6

Ejercicio 1.

a) Definir Lógica de Hoare. ¿Para qué se usa? Definir y explicar cada parte de una "terna de Hoare".

La Lógica de Hoare es un sistema formal con un enfoque axiomático, basado en la lógica de predicados. Se utiliza para verificar la correctitud de los programas. Permite que los programas se pueden verificar sintácticamente haciendo uso de axiomas y reglas. Es decir, permite demostrar matemáticamente que un programa cumple con su especificación.

Una terna de Hoare, también conocida como fórmula de correctitud, tiene la forma {p} S {q} donde:

- Las condiciones p y q son fórmulas de primer orden interpretadas en el dominio de los números enteros.
- La fórmula {p} S {q} establece que si el programa S se ejecuta cuando se cumple la precondición p, y termina, entonces al final se cumple la postcondición q.
    - {p} representa la precondición, una aserción lógica que debe ser verdadera antes de que se ejecute el programa S. Describe el estado del sistema antes de la ejecución.
    - S es el programa o fragmento de código cuya correctitud se quiere demostrar.
    - {q} representa la postcondición, una aserción lógica que debe ser verdadera después de que se ejecute el programa S. Describe el estado del sistema después de la ejecución.

b) Explicar la diferencia entre correctitud parcial y correctitud total. Dar un ejemplo de cada una.

- Un programa S es parcialmente correcto con respecto a una especificación (p, q) sii para todo estado σ:

$$(\sigma \models p \land val(\pi(S, \sigma)) \neq \bot) \rightarrow val(\pi(S, \sigma)) \models q$$

  o sea, sii desde cualquier estado σ que satisface p, si S termina lo hace en un estado σ' que satisface q. La expresión |= {p} S {q} denota que S es parcialmente correcto con respecto a (p, q).

- Un programa S es totalmente correcto con respecto a una especificación (p, q) sii para todo estado σ:

$$\sigma \models p \rightarrow (val(\pi(S, \sigma)) \neq \bot \land val(\pi(S, \sigma)) \models q)$$

  o sea, sii desde cualquier estado σ que satisface p, S termina en un estado σ' que satisface q.

  La expresión |= ⟨p⟩ S ⟨q⟩ denota que S es totalmente correcto con respecto a (p, q). En particular, |= ⟨p⟩ S ⟨true⟩ denota que a partir de un estado que satisface p, S termina (en algún estado).

- Ejemplo parcialmente correcto:

  {x = 0} x := x + 1 {x = 1}

- Ejemplo totalmente correcto:
  ⟨x ≠ 0⟩ while x = 0 do skip od ⟨x ≠ 0⟩

c) Enunciar y explicar los axiomas de la Lógica de Hoare.

1. Axioma del skip (SKIP):

$$\{p\}\ skip\ \{p\}$$

   El skip no modifica el estado inicial.

2. Axioma de la asignación (ASI):

$$\{p[x|e])\}x := e\{p(x)\}$$

Si vale p(x) al final, es que valía p(e) antes. ASI impone pruebas de atrás para adelante

3. Regla de la secuencia (SEC):

$$\frac{\{p\}\ S_1\ \{r\}\ ;\ \{r\}\ S_2\ \{q\}}{\{p\}\ S_1\ ;\ S_2\ \{q\}}$$

El predicado r hace de nexo entre S1 y S2, y luego se elimina. La regla se puede generalizar a más de dos premisas.

4. Regla del condicional (COND):

$$\frac{\{p \wedge B\}\ S_1\ \{q\}\ ,\{p \wedge \neg B\}\ S_2\ \{q\}}{\{p\}\ if\ B\ then\ S_1\ else\ S_2\ fi\ \{q\}}$$

Único punto de entrada en que vale p, y único punto de salida en que vale q, independientemente de si B se evalúa verdadera o falsa.

5. Regla de la repetición (REP):

$$\frac{\{p \wedge B\}\ S\ \{p\}}{\{p\}\ while\ B\ do\ S\ od\ \{p \wedge \neg B\}}$$

Si p vale al comienzo del while, y mientras vale B el cuerpo S preserva p, entonces por un razonamiento inductivo p vale al finalizar el while.
El predicado p es un invariante del while, vale antes, durante y después. A la salida del while, se cumple naturalmente p ∧ ¬B.
No asegura que el while termine

6. Regla de la consecuencia (CONS):

$$\frac{p \rightarrow p_1, \{p_1\}\ S\ \{q_1\}, q_1 \rightarrow q}{\{p\}\ S\ \{q\}}$$

Permite reforzar precondiciones y debilitar postcondiciones.

La regla no depende del lenguaje de programación sino del dominio semántico. Es una regla semántica más que sintáctica. Permite manipular todos los axiomas del dominio semántico en las pruebas.

7. Regla de la terminación (REP*):

$$\frac{\langle p \wedge B\rangle\ S\ \langle p\rangle, \langle p \wedge B\ \wedge t = Z\rangle\ S\ \langle t < Z\rangle, p \rightarrow t \geq 0}{\{p\}\ while\ B\ do\ S\ od\ \{p \wedge \neg B\}}$$

- A REP se le agrega un variante, que es una función entera t definida, como el invariante, en términos de las variables del programa.
- La variable Z es una variable lógica, no aparece en p ni en t, su objetivo es conservar el valor de t antes de la ejecución del cuerpo del while.
- La primera premisa es la de REP (p es el invariante).
- Por la segunda premisa, t se decrementa en cada iteración.
- Por la tercera premisa, t arranca y se mantiene positiva.
- De esta manera, el while debe terminar, es imposible que haya una cadena infinita de números naturales que cumpla: $n_1 > n_2 > n_3 > \cdots$
    - El valor de t decrece de iteración en iteración. Como t es una función entera positiva, indefectiblemente la cadena descendente de los t_i es finita, termina

Ejercicio 2.

Aplicar el axioma de asignación (ASI) para obtener las precondiciones correspondientes:

a) {?} x := x + 1 {x + 1 > 0}

{ x + 2 > 0} x := x + 1 {x + 1 > 0} (ASI)

b) {?} x := y {x = y}

{y = y} x := y {x = y} (ASI)

Ejercicio 3.

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

a) Se cumple {x = 0} while y = 0 do skip od {x = 0}.

Sea p = (x = 0) y q = (x = 0), S = while y = 0 do skip od, {p} S {q} se cumple sii para todo estado σ: (σ |= p ∧ val(π(S, σ)) ≠ ⊥) → val(π(S, σ)) |= q. O sea, sii desde cualquier estado σ que satisface p, si S termina lo hace en un estado σ' que satisface q

Hay 3 posibles casos:

- Si p no se satisface desde σ, la implicación se cumple trivialmente, ya que el antecedente será falso.
- Si p se satisface desde σ y S termina, como el valor de x no se modifica durante toda la ejecución de S, se seguirá cumpliendo que x = 0 al terminar S en un estado σ', es decir, σ' va a satisfacer q. Esto hace que la implicación se cumpla ya que tanto el antecedente como el consecuente son verdaderos.
- Si p se satisface desde σ y S no termina, la implicación se cumple trivialmente, ya que el antecedente será falso.

Como la implicación nunca será falsa, se cumple la correctitud parcial.

b) Se cumple ⟨x = 0⟩ while y = 0 do skip od ⟨x = 0⟩.

Sea p = (x = 0) y q = (x = 0), S = while y = 0 do skip od, ⟨p⟩ S ⟨q⟩ se cumple sii para todo estado σ: σ |= p → (val(π(S, σ)) ≠ ⊥ ∧ val(π(S, σ)) |= q). O sea, sii desde cualquier estado σ que satisface p, S termina en un estado σ' que satisface q.

Hay 3 posibles casos:

- Si p no se satisface desde σ, la implicación se cumple trivialmente, ya que el antecedente será falso.
- Si p se satisface desde σ y S termina, como el valor de x no se modifica durante toda la ejecución de S, se seguirá cumpliendo que x = 0 al terminar S en un estado

σ', es decir, σ' va a satisfacer q. Esto hace que la implicación se cumpla ya que tanto el antecedente como el consecuente son verdaderos.

- Si p se satisface desde σ y S no termina, la implicación no se cumple puesto que el antecedente será verdadero pero el consecuente será falso.

Como existe un caso donde la implicación toma el valor falso, no se cumple la correctitud total.

Contraejemplo:

Sea el estado inicial σ, con σ(x) = 0 y σ(y) = 0, el programa entra en un loop imposibilitando su final. Se puede observar como el antecedente es verdadero puesto que σ satisface p, pero el consecuente es falso puesto que el programa no termina, haciendo que no se cumpla la implicación.

c) Se cumple {x > 0} while x ≠ 0 do x := x – 1 od {x = 0}

Sea p = (x > 0) y q = (x = 0), S = while x ≠ 0 do x := x – 1 od, {p} S {q} se cumple sii para todo estado σ: (σ |= p ∧ val(π(S, σ)) ≠ ⊥) → val(π(S, σ)) |= q. O sea, sii desde cualquier estado σ que satisface p, si S termina lo hace en un estado σ' que satisface q

Hay 2 posibles casos:

- Si p no se satisface desde σ, la implicación se cumple trivialmente, ya que el antecedente será falso.
- Si p se satisface desde σ, al inicio del programa se tiene que x es mayor a 0, como a x se le restara 1 hasta que sea igual a 0, llegara un punto en el programa en el que x quedara con el valor 0, haciendo que no se entre más al while, puesto que no se cumple con su condición. Es en este punto que S finaliza en el estado σ' que satisface q, puesto que x = 0. Es importante notar que S siempre terminara. Esto hace que la implicación se cumpla ya que tanto el antecedente como el consecuente son verdaderos.

Como la implicación nunca será falsa, se cumple la correctitud parcial.

d) Se cumple ⟨true⟩ while x ≠ 0 do x := x – 1 od ⟨true⟩

Contraejemplo:

Teniendo en cuenta que true denota a todos los estados, se puede partir desde el estado inicial σ, con σ(x) = -1. En este caso se satisface p desde σ, pero como se entra en un loop en la ejecución del programa, este nunca finalizará, provocando que el consecuente sea falso. Entonces, la implicación no se cumple puesto que el antecedente será verdadero pero el consecuente será falso.

Como la implicación no se cumple, no se cumple la correctitud total.