Verificación de programas I

Algoritmos y estructuras de datos

by Damy

Repaso

A la hora de especificar un procedimiento, teníamos nuestro "Requiere" y nuestro "Asegura" :

Proc Ejemplo(...) {

Requiere : {P}

Asegura:{Q}

}

Repaso

Proc Ejemplo(...){

Requiere : {P}

S

Asegura:{Q}

}

No hay que olvidar que, en el medio, existirá un programa "**S**", el cual querremos que cumpla dicha especificación.

Repaso

¿Hay alguna forma que, dados **P**, Q y S, podamos verificar de manera formal que el programa es correcto respecto a dicha especificación? Si!

A lo largo de esta clase, lo haremos mediante la Precondición más débil (WP), en donde son fundamentales los conceptos de "Lógica y especificación" y "Relación de fuerza", vistos anteriormente.

Tripla de Hoare

Cuando un programa S es correcto respecto a la especificación (P, Q), lo denotamos con la siguiente tripla de Hoare:

$$\{P\}\ S\ \{Q\}$$

Precondición más débil (WP)

Dadas Q y S.

¿Qué es la WP(S, Q)?

Es la P más débil tal que {P} S {Q}

Ejemplo:

Requiere: {P}

$x := x + 2$

Asegura: $\{x > 10\}$

¿Cuál seria la WP$(x := x+2,\ x > 10)$ **en este caso?**

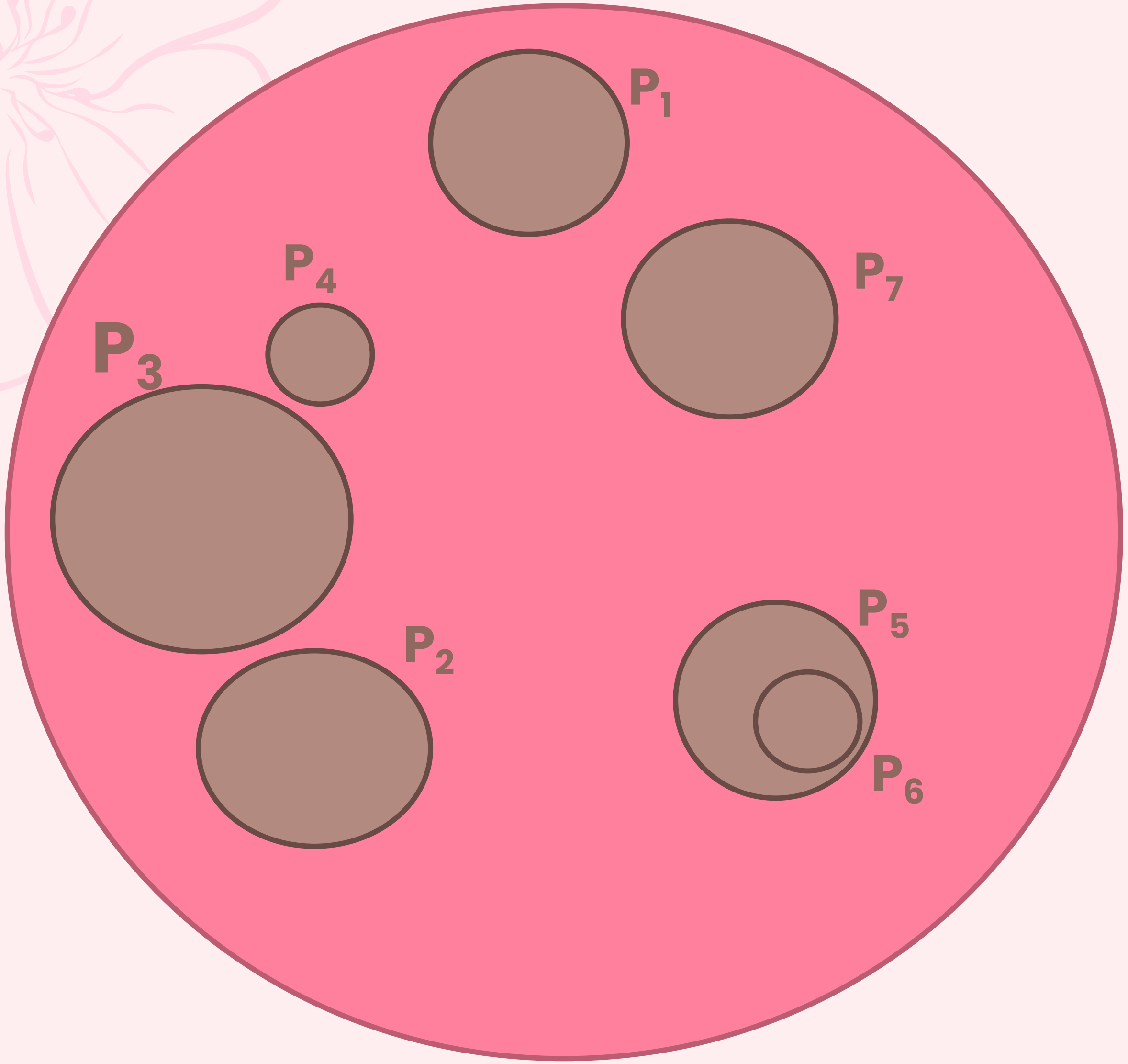
Claramente $x > 8$.

Con $x>8$ se cumple que vale la tripla de Hoare:

$$\{x>8\}\ x:=x+2\ \{x>10\}$$

A parte, P es la más débil de todas, ya que, a pesar de que existen otras P que hacen valer la tripla de Hoare ($P_1 \equiv x > 9$, $P_2 \equiv x=10$, $P_3 \equiv x > 100$, $P_4 \equiv x > 10 \wedge x < 25$, etc), todas esas P son más fuertes que $x>8$

WP(S, Q)



Es decir, al ser la más débil de todas, se cumple que para toda P que cumpla:

{P} S {Q}

La WP(S, Q) va a ser más débil que dicha P. En otras palabras:

P -> WP(S, Q)

Recapitulando:

¿Cómo podemos estar seguros que la tripla de Hoare:

{P′} S′ {Q′}

Es formalmente valida?

- Calculamos la WP(S′, Q′)
- Si P′ -> WP(S′, Q′), eso significa que la tripla es valida, pues WP(S′, Q′) es la más débil de todas tal que hace valer la tripla, y P′ logro implicarla.

SmallLang: Sintaxis

Es el lenguaje que usaremos para escribir S

- Asignación: **x := E**
- Nada: **Skip**
- Secuencia: **S1;S2** es un programa, si **S1** y **S2** son programas
- Condicional: **if B then S1 else S2 endif** es un programa, si **B** es una expresión lógica y **S1** y **S2** son dos programas
- Ciclo: **While B do S endwhile** es un programa, si **B** es una expresión lógica y **S** es un programa

SmallLang: Ejemplos

```
x := x+ 2;
Skip;
If(x = 3) then
  y := 5
Else
  y:= s[3]+2
Endif;
Skip
```

```
i := 0;
While(i < x) do
  y:= y+10;
  i:= i+1
Endwhile;
If(y <0) then
  y := 0
Else
  Skip
endif
```

SmallLang: Ejemplos

Observación: El programa de la izquierda puede ser escrito de la siguiente manera:

x := x+ 2; Skip; If(x = 3) then y := 5 Else y:= S[3]+2 endif; Skip

Esto lo hara más sencillo a la hora de calcular la WP cuando apliquemos el "Axioma 3" de las diapositivas posteriores.

Ejercicio

¿Cuál sería la wp en estos casos?

A) wp(Skip; x:= x+2; Skip, x = 19)

B) wp(a:=a+1; b:=a, b < 245)

C) wp(x:=5, true)

d) wp(x:=1/y, true)

e) wp(L[i] := 4, $(\forall j: \mathbb{Z})(0 \leq j < |L| \rightarrow_L (L[j] = 4)))$

f) wp(if (a=3) then (b:=b+2) else (b:= b-2) endif, b ≤ 2))

g) wp(b:=10, b ≤ 2))

Ejercicio

¿Cuál sería la wp en estos casos?

A) $wp(Skip;\ x:=x+2;\ Skip,\ x = 19) \equiv x = 17$

B) $wp(a:=a+1;\ b:=a,\ b < 245) \equiv a < 244$

C) $wp(x:=5,\ true) \equiv true$

d) $wp(x:=1/y,\ true) \equiv y \neq 0$

e) $wp(L[i] := 4,\ (\forall j: \mathbb{Z})(0 \leq j < |L| \to_L (L[j] = 4))) \equiv 0 \leq i < |L| \land_L (\forall j: \mathbb{Z})(0 \leq j < |L| \land_L j \neq i \to_L (L[j] = 4)))$

f) $wp(\text{if } (a=3) \text{ then } (b:=b+2) \text{ else } (b:= b-2) \text{ endif},\ b \leq 2)) \equiv (a = 3 \land b \leq 0) \lor (a \neq 3 \land b \leq 4)$

g) $wp(b:=10,\ b \leq 2)) \equiv false$

Predicados útiles

- Dada una expresión E, def(E) son las condiciones necesarias para que E este definida.

- Dado un predicado Q, el predicado Q_E^x se obtiene reemplazando todas las apariciones libres de la variable x por E

Ejemplos

Considerando que: $x, z, y: \mathbb{Z}$, $S: seq\langle\mathbb{Z}\rangle$:

- $Def(x) \equiv True$ (Asumimos que las variables siempre están definidas)
- $Def(S[i]) \equiv 0 \leq i < |S|$
- $Def(S[0]) \equiv |S| \geq 1$
- $Def(1/y) \equiv y \neq 0$

Si $Q \equiv z+4$:

- $Q^z_y \equiv y+4$

Axiomas

- Axioma 1: $wp(x := E, Q) \equiv def(E) \wedge_L Q^x_E$
- Axioma 2: $wp(skip, Q) \equiv Q$
- Axioma 3: $wp(S1;S2, Q) \equiv wp(S1, wp(S2, Q))$
- Axioma 4: Si **S = if B then S1 else S2 endif** entonces

  $wp(S, Q) \equiv def(B) \wedge_L ((B \wedge wp(S1, Q) \vee (\neg B \wedge wp(S2, Q))$

Ejercicios

a) Explicar cual seria la wp con sus palabras

b) Calcular la wp a partir de los axiomas

Asumimos que, para lo que sigue:

$x, y \in \mathbb{R} \mid i, a \in \mathbb{Z} \mid L: seq\langle \mathbb{Z} \rangle$

(1) $wp(x := L[i] / 2;\ y := x+2;\ \text{Skip},\ y < 0)$

a) ...

b) ...

(1) wp(x := L[i] / 2; y := x+2; Skip, y < 0)

¿Cómo razono el a)?

El valor de "y" termina siendo "x" aumentado en 2, sin embargo anteriormente a "x" se le asigna lo que solía haber en "L[i]" dividido 2. Entonces, lo que había en la posición "i" de "L" tiene que ser algo que luego de dividirlo entre 2, y aumentarle 2 sea menor que 0, ósea el valor en L[i] debe ser menor que -4

Noto que el Skip no me modifica el programa entonces no lo tengo en cuenta.

Finalmente, veo que "i" debe estar en el rango de la secuencia, para que al indexar, no se indefina.

a) "i" debe estar en rango de "L", y "L[i]" debe ser menor que -4

b) $wp(x := L[i] / 2;\ y := x+2;\ Skip,\ y < 0)$

Ax (3) $\equiv wp(x := L[i] / 2;\ y := x+2,\ wp(Skip,\ y<0))$

Ax (2) $\equiv wp(x := L[i] / 2;\ y := x+2,\ y<0)$

Ax (3) $\equiv wp(x := L[i] / 2,\ wp(y := x+2,\ y<0))$

Ax (1) $\equiv wp(x := L[i] / 2,\ def(x+2) \wedge_L x+2<0)$

$\equiv wp(x := L[i] / 2,\ True \wedge_L x<-2)$

Ax (1) $\equiv def(L[i]/2) \wedge_L L[i]/2 < -2$

$\equiv 0 \leq i < |L| \wedge_L L[i] < -4$

Que es lo que habíamos dicho en a)

$$(2)\ wp(L[i+1] := a,\ (\forall j : \mathbb{Z})(0 \leq j < |L| \rightarrow_L (L[j] > 0 \land L[j] \bmod 2 = 0)))$$

a)…

b)…

(2) $wp(L[i+1] := a, (\forall j: \mathbb{Z})(0 \leq j < |L| \rightarrow_L (L[j] > 0 \wedge L[j] \bmod 2 = 0)))$

¿Cómo razono el a)?

Según "Q", todos elementos de L deben ser positivos y pares. Entonces todos los valores que no modifiqué de L, es decir, aquellos que no estén en la posición "i+1" deberían de por si ser positivos y pares. Luego el valor que se le asigna en la posición "i+1", ósea "a", debe ser positivo y par. Finalmente, "i+1" debe estar entre 0 y la |L| para que no se indefina

a) "i+1" debe estar en rango de "L", todos los elementos en las posiciones distintas a "i+1" deben ser positivos y pares, y "a" debe ser positivo y par

$(2)\ wp(L[i+1] := a, (\forall j: \mathbb{Z})(0 \leq j < |L| \rightarrow_L (L[j] > 0 \wedge L[j] \bmod 2 = 0)))$

b)...

Wait

¿Puedo aplicar el axioma 1 con $L[i+1] := a$?

Nop

¿Cómo podemos reescribirlo

para aplicarlo?

setAt al rescate

`setAt(L, i, E)` Devuelve una secuencia igual a la original, pero con el elemento en la posición `L[i]` cambiado por `E`

Puedo reescribir `L[i] := E` como `L := setAt(L, i, E)` y así aplicar el axioma 1

También hay algunas observaciones
a tener en cuenta

Observaciones importantes

I. $\mathrm{def}(\mathrm{setAt}(L, i, E)) \equiv (\mathrm{def}(E) \wedge \mathrm{def}(L) \wedge \mathrm{def}(i)) \wedge_L (0 \leq i < |L|)$

II. $|\mathrm{setAt}(L, i, E)| = |L|$

III. $0 \leq i < |L| \wedge_L (\forall j : \mathbb{Z})((0 \leq j < |L| \wedge j \neq i) \to_L \mathrm{setAt}(L, i, E)[j] = L[j])$

IV. $0 \leq i < |L| \wedge_L \mathrm{setAt}(L, i, E)[i] = E$

V. $0 \leq k < |L| \wedge_L (\forall j : \mathbb{Z})((0 \leq j < |L| \wedge j = k) \to_L S[j] = 2)$ es lo mismo que: $0 \leq k < |L| \wedge_L S[k] = 2$

Observaciones importantes
En español

I. Para que setAt(L, i, E)este definido, deben estar definidos todos sus parámetros, e "i" debe estar en rango

II. setAt(L, i, E)solo cambia un valor de L, el tamaño se preserva

III. Todo valor en L distinto de i, sigue siendo igual al que había antes

IV. El setAt(L, i, E) cambia lo que había en la posición i por "E", entonces al indexarlo en "i" será igual a "E"

V. Si tengo un $(\forall j: \mathbb{Z})((j = \text{"valor"}) \rightarrow_L \text{"algo"})$, el ∀ deja de tener utilidad, ya que en el antecedente ya estas fijando a que "j" sea igual a cierto valor/variable, podrías escribir directamente ese "algo" reemplazando "j" por dicho valor

b) $wp(L[i+1] := a, (\forall j: \mathbb{Z})(0 \leq j < |L| \rightarrow_L (L[j] > 0 \wedge L[j] \bmod 2 = 0)))$

Reescribo

$\equiv wp(L := setAt(L, i+1, a), (\forall j: \mathbb{Z})(0 \leq j < |L| \rightarrow_L (L[j] > 0 \wedge L[j] \bmod 2 = 0)))$

Ax (1)

$\equiv def(setAt(L, i+1, a)) \wedge_L (\forall j: \mathbb{Z})(0 \leq j < |setAt(L, i+1, a)| \rightarrow_L (setAt(L, i+1, a)[j] > 0 \wedge setAt(L, i+1, a)[j] \bmod 2 = 0))$

Obs (I)

$\equiv (def(L) \wedge def(i+1) \wedge def(a)) \wedge_L 0 \leq i+1 < |L| \wedge_L (\forall j: \mathbb{Z})(0 \leq j < |setAt(L, i+1, a)| \rightarrow_L (setAt(L, i+1, a)[j] > 0 \wedge setAt(L, i+1, a)[j] \bmod 2 = 0))$

$\equiv$ True $\wedge_L$ $-1 \leq i < |L|-1$ $\wedge_L$ $(\forall j: \mathbb{Z})(0 \leq j < |setAt(L, i+1, a)| \rightarrow_L (setAt(L, i+1, a)[j] > 0$ $\wedge$ $setAt(L, i+1, a)[j] \bmod 2 = 0))$

Obs (II)

$\equiv$ $-1 \leq i < |L|-1$ $\wedge_L$ $(\forall j: \mathbb{Z})(0 \leq j < |L| \rightarrow_L (setAt(L, i+1, a)[j] > 0 \wedge setAt(L, i+1, a)[j] \bmod 2 = 0))$

Queremos **deshacernos** de los setAt

Podemos aprovechar que dicha función devuelve una secuencia exactamente igual a L, salvo una posición.

Idea: Como indexo en j y en el setAt se reemplaza la posición i+1, podríamos separar en casos, cuando j = i+1 y cuando j ≠ i+1

$\equiv$ $-1 \leq i < |L|-1 \wedge_L ((\forall j: \mathbb{Z})(0 \leq j < |L| \wedge j = i+1 \to_L (setAt(L, i+1, a)[j] > 0 \wedge setAt(L, i+1, a)[j] \bmod 2 = 0)) \wedge (\forall j: \mathbb{Z})(0 \leq j < |L| \wedge j \neq i+1 \to_L (setAt(L, i+1, a)[j] > 0 \wedge setAt(L, i+1, a)[j] \bmod 2 = 0)))$

Obs (V)

$\equiv$ $-1 \leq i < |L|-1 \wedge_L ((setAt(L, i+1, a)[i+1] > 0 \wedge setAt(L, i+1, a)[i+1] \bmod 2 = 0) \wedge (\forall j: \mathbb{Z})(0 \leq j < |L| \wedge j \neq i+1 \to_L (setAt(L, i+1, a)[j] > 0 \wedge setAt(L, i+1, a)[j] \bmod 2 = 0)))$

Obs (IV)

$\equiv$ $-1 \leq i < |L|-1 \wedge_L (a > 0 \wedge a \bmod 2 = 0 \wedge (\forall j: \mathbb{Z})(0 \leq j < |L| \wedge j \neq i+1 \to_L (setAt(L, i+1, a)[j] > 0 \wedge setAt(L, i+1, a)[j] \bmod 2 = 0)))$

Obs (III)

$$\equiv -1 \leq i < |L|-1 \wedge_L \left(a > 0 \wedge a \bmod 2 = 0 \wedge (\forall j: \mathbb{Z})(0 \leq j < |L| \wedge j \neq i+1 \rightarrow_L (L[j] > 0 \wedge L[j] \bmod 2 = 0)) \right)$$

$$\equiv -1 \leq i < |L|-1 \wedge_L (\forall j: \mathbb{Z})((0 \leq j < |L| \wedge j \neq i+1) \rightarrow_L (L[j] > 0 \wedge L[j] \bmod 2 = 0)) \wedge a > 0 \wedge a \bmod 2 = 0$$

Que es lo mismo que habíamos dicho en a)

$S' \equiv$ if$(a \geq 10)$ then

$x := 3$

else

$x := x+5$

endif

(3) wp$(S', x > 4)$

a)...

b)...

$$S' \equiv \text{if}(a \geq 10) \text{ then}$$
$$x := 3$$
$$\text{else}$$
$$x := x+5$$
$$\text{endif}$$

(3) wp(S', x > 4)

¿Cómo razono el a)?

Noto que si "a" fuese mayor/igual a 10, jamás se cumpliría "Q", ya que se le asignaría el valor 3 a "x". Entonces "a" debe ser si o si menor que 10. Luego, la única manera de que "x" sea mayor que 4 luego de ser aumentado consigo mismo por 5, es que en primera instancia haya sido mayor que -1.

a) "a" debe ser menor que 10, y "x" debe ser mayor que -1

$S' \equiv$ if$(a \geq 10)$ then

$x := 3$

else

$x := x+5$

endif

b) $wp(S', x > 4)$

Ax (4) $\equiv def(a \geq 10) \wedge_L ((a \geq 10 \wedge wp(x{:=}3, x>4)) \vee$

$(\neg(a \geq 10) \wedge wp(x{:=}x+5, x>4)))$

Ax (1) $\equiv True \wedge_L ((a \geq 10 \wedge (def(3) \wedge_L 3 > 4)) \vee$

$(a < 10 \wedge def(x+5) \wedge_L x+5 > 4))$

$\equiv (a \geq 10 \wedge True \wedge_L False) \vee (a < 10 \wedge True \wedge_L x > -1)$

$\equiv False \vee (a < 10 \wedge x > -1)$

$\equiv a < 10 \wedge x > -1$

Que es lo mismo que habíamos dicho en a)

Terminamos