

Algoritmos y Estructuras de Datos

Segundo cuatrimestre - 2025

Departamento de Computación - FCEyN - UBA

Correctitud

Transformación de estados

- ▶ Llamamos **estado** de un programa a los valores de todas sus variables en un punto de su ejecución:
 1. Antes de ejecutar la primera instrucción,
 2. entre dos instrucciones sucesivas, y
 3. después de ejecutar la última instrucción.
- ▶ Podemos considerar la **ejecución** de un programa como una **sucesión de estados**.
- ▶ La asignación es la instrucción que permite pasar de un estado al siguiente en esta sucesión de estados.
- ▶ Las estructuras de control se limitan a especificar el flujo de ejecución (es decir, el orden de ejecución de las asignaciones).

Afirmaciones sobre estados

- ▶ Sea el siguiente programa que se ejecuta con estado inicial $\{True\}$.
- ▶ $\{True\}$
`int x = 0;`
 $\{x = 0\}$
`x = x + 3;`
 $\{x = 3\}$
`x = 2 * x;`
 $\{x = 6\}$
- ▶ ¿Finaliza siempre el programa? Sí, porque no hay ciclos
- ▶ ¿Cuál es el estado final al finalizar su ejecución? $\{x = 6\}$

Afirmaciones sobre estados

- ▶ Sea el siguiente programa que se ejecuta con estado inicial con una variable a ya definida ($\{a = A_0\}$).
- ▶ $\{a = A_0\}$
`int b = a + 2;`
 $\{a = A_0 \wedge b = A_0 + 2\}$
`int result = b - 1;`
 $\{a = A_0 \wedge b = A_0 + 2 \wedge \text{result} = (A_0 + 2) - 1 = A_0 + 1\}$
- ▶ ¿Finaliza siempre el programa? Sí, porque no hay ciclos
- ▶ ¿Cuál es el estado final al finalizar su ejecución?
 $\{a = A_0 \wedge b = A_0 + 2 \wedge \text{result} = A_0 + 1\} \Rightarrow \{\text{result} = a + 1\}$

Corrección de un programa

- **Definición.** Decimos que un programa S es **correcto respecto de una especificación** dada por una precondición P y una postcondición Q , si siempre que el programa comienza en un estado que cumple P , el programa **termina su ejecución**, y en el estado final **se cumple** Q .
- **Notación.** Cuando S es correcto respecto de la especificación (P, Q) , lo denotamos con la siguiente **tripla de Hoare**:

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

Afirmaciones sobre estados

- Sea la siguiente especificación para incrementar en una unidad el valor de un entero.
- $\text{proc } \textit{spec_incrementar}(\text{inout } a : \mathbb{Z})\{$
 requiere $\{a = A_0\}$
 asegura $\{a = A_0 + 1\}$
}
- ¿Es el siguiente programa S **correcto** con respecto a su especificación?

```
int incrementar(int& a) {  
    int b = a + 2;  
    int result = b - 1;  
    a = result;  
}
```

Ejemplo

- ▶ $\text{proc } \textit{spec_incrementar}(\text{inout } a : \mathbb{Z})\{\$
 requiere $\{a = A_0\}$
 asegura $\{a = A_0 + 1\}$
}
- ▶ Sea el siguiente programa que se ejecuta con estado inicial con una variable $a = A_0$.
- ▶ $\{a = A_0\}$
 int $b = a + 2;$
 $\{a = A_0 \wedge b = A_0 + 2\}$
 int $\text{result} = b - 1;$
 $\{a = A_0 \wedge b = A_0 + 2 \wedge \text{result} = (A_0 + 2) - 1 = A_0 + 1\}$
 $a = \text{result};$
 $\{a = A_0 + 1 \wedge b = A_0 + 2 \wedge \text{result} = A_0 + 1\}$
 Por lo tanto, se deduce que:
 $\{a = A_0 + 1\}$

Intercambiando los valores de dos variables enteras

- ▶ $\text{proc swap}(\text{inout } a : \mathbb{Z}, \text{inout } b : \mathbb{Z})\{$
 requiere $\{a = A_0 \wedge b = B_0\}$
 asegura $\{a = B_0 \wedge b = A_0\}$
}
- ▶ **Ejemplo:** Intercambiamos los valores de dos variables, pero sin una variable auxiliar!
- ▶ $\{a = A_0 \wedge b = B_0\}$
 $a = a + b;$
 $\{a = A_0 + B_0 \wedge b = B_0\}$
 $b = a - b;$
 $\{a = A_0 + B_0 \wedge b = (A_0 + B_0) - B_0\}$
 $\equiv \{a = A_0 + B_0 \wedge b = A_0\}$
 $a = a - b;$
 $\{a = A_0 + B_0 - A_0 \wedge b = A_0\}$
 $\equiv \{a = B_0 \wedge b = A_0\}$

Alternativas

- ▶ Sea el siguiente programa con una variable a de entrada cuyo valor no se modifica (i.e. podemos asumir $a = A_0$ como constante)
- ▶ Cuando tenemos una alternativa, debemos considerar las dos ramas por separado.
- ▶ Por ejemplo:

$$\{a = A_0 \wedge b = B_0\}$$

```
if( a > 0 ) {  
    b = a;  
} else {  
    b = -a;  
}
```

$$\dot{\{b = ||a||\}}?$$

- ▶ Verifiquemos ahora que $b = ||a||$ después de la alternativa.

Alternativas

► Rama positiva:

Se cumple la condición $a > 0$

$$\{a = A_0 \wedge b = B_0 \wedge a > 0\} \equiv \{a = A_0 \wedge b = B_0 \wedge A_0 > 0\}$$

$$b = a;$$

$$\{a = A_0 \wedge b = A_0 \wedge A_0 > 0\}$$

$$\Rightarrow \{b = ||a||\}$$

► Rama negativa:

No se cumple la condición $a > 0$ (o sea $a \leq 0$)

$$\{a = A_0 \wedge b = B_0 \wedge \neg a > 0\} \equiv \{a = A_0 \wedge b = B_0 \wedge A_0 \leq 0\}$$

$$b = -a;$$

$$\{a = A_0 \wedge b = -A_0 \wedge A_0 \leq 0\}$$

$$\Rightarrow \{b = ||a||\}$$

► En ambos casos vale $b = ||a||$

► Por lo tanto, esta condición vale al salir de la instrucción alternativa.

Demostrando que un programa es correcto

- ▶ Sabemos **razonar** sobre la corrección de nuestros programas, anotando el código con predicados que representan los estados.
- ▶ Nos interesa **formalizar** estos razonamientos, para estar seguros de que no cometimos errores en la demostración.
- ▶ Una forma de conseguirlo es la siguiente: A partir de la tripla de Hoare $\{P\} S \{Q\}$, obtener una fórmula lógica α tal que

α es verdadera si y sólo si $\{P\} S \{Q\}$ es verdadera.

- ▶ Entre otras cosas, esto nos permite automatizar la demostración con un **verificador automático** (!)

Un lenguaje imperativo simplificado

- ▶ Para facilitar nuestro trabajo, definamos un lenguaje imperativo más sencillo que Java al que llamaremos **SmallLang**.¹
- ▶ SmallLang únicamente soporta las siguientes instrucciones:
 1. **Nada**: Instrucción **skip** que no hace nada.
 2. **Asignación**: Instrucción **x := E**.
- ▶ Además, soporta las siguientes estructuras de control:
 1. **Secuencia**: **S1; S2** es un programa, si **S1** y **S2** son dos programas.
 2. **Condicional**: **if B then S1 else S2 endif** es un programa, si **B** es una expresión lógica y **S1** y **S2** son dos programas.
 3. **Ciclo**: **while B do S endwhile** es un programa, si **B** es una expresión lógica y **S** es un programa.

¹ *The Semantics of a Small Language* de David Gries

Demostraciones de corrección

- ▶ Buscamos un mecanismo para demostrar “automáticamente” la corrección de un programa respecto de una especificación (es decir, la validez de una tripla de Hoare).
- ▶ ¿Es válida esta tripla?

$$\begin{array}{c} \{x \geq 4\} \\ x := x + 1 \\ \{x \geq 7\} \end{array}$$

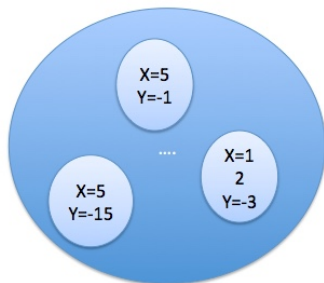
- ▶ No. Contrajemplo: con $x = 4$ no se cumple la postcondición.
- ▶ ¿Es válida esta tripla?

$$\begin{array}{c} \{x \geq 4\} \\ x := x + 1 \\ \{x \geq 5\} \end{array}$$

- ▶ Sí. Es válida!

La precondition más débil

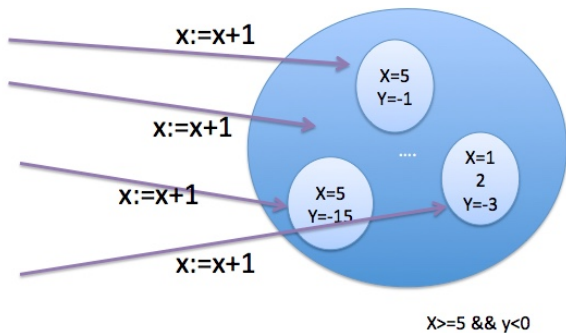
$\{ x \geq 4 \wedge y < -2 \}$
 $x := x + 1$
 $\{ x \geq 5 \wedge y < 0 \}$



$X \geq 5 \ \&\& \ y < 0$

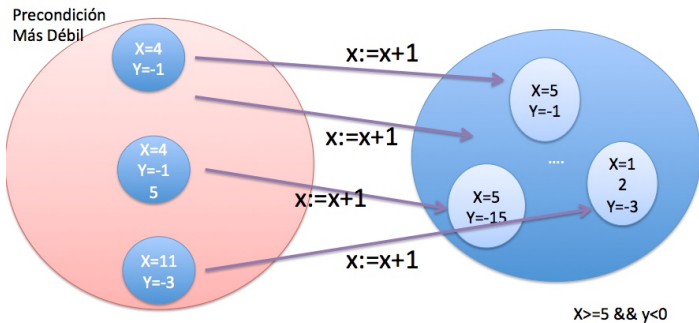
La precondition más débil

$\{ x \geq 4 \wedge y < -2 \}$
 $x := x + 1$
 $\{ x \geq 5 \wedge y < 0 \}$



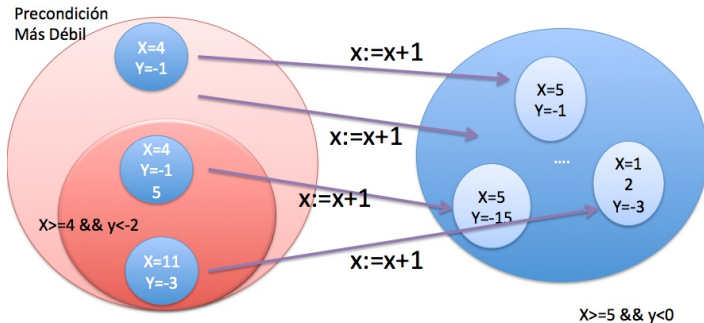
La precondition más débil

$$\begin{aligned} & \{ x \geq 4 \wedge y < -2 \} \\ & \quad x := x + 1 \\ & \{ x \geq 5 \wedge y < 0 \} \end{aligned}$$

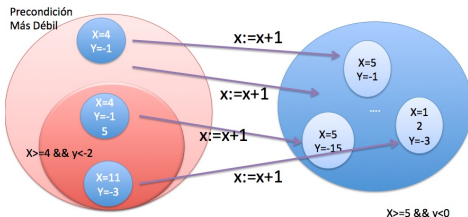


La precondition más débil

$$\{ x \geq 4 \wedge y < -2 \}$$
$$x := x + 1$$
$$\{ x \geq 5 \wedge y < 0 \}$$



La precondition más débil



- Supongamos que tenemos un predicado que captura la precondition más débil del programa S con la postcondición Q (**Notación:** $wp(S, Q)$)
- ¿Qué formula podemos usar para probar que la tripla de Hoare es válida?

$$(x \geq 4 \wedge y < -2) \Rightarrow_L wp(x := x + 1, x \geq 5 \wedge y < 0)$$

Precondición más débil

- **Definición.** La **precondición más débil** de un programa **S** respecto de una postcondición Q es el predicado P más débil posible tal que $\{P\}\mathbf{S}\{Q\}$.
- **Notación.** $wp(\mathbf{S}, Q)$.
- **Teorema:** Una tripla de Hoare $\{P\}\mathbf{S}\{Q\}$ es válida si y sólo si:

$$P \Rightarrow_L wp(S, Q)$$

Precondición más débil

- Ejemplo:

$$\{wp(x := x+1, Q)\}$$

$$x := x + 1$$

$$\{Q : x \geq 7\}$$

- ¿Cuál es la precondición más débil de $x := x+1$ con respecto a la postcondición $x \geq 7$?
- $wp(x := x+1, Q) \equiv x \geq 6$.

Precondición más débil

- ▶ Otro ejemplo:

$$\{wp(\mathbf{S2}, Q)\}$$

$$\mathbf{S2: } x := 2 * |x| + 1$$

$$\{Q : x \geq 5\}$$

- ▶ $wp(\mathbf{S2}, Q) \equiv x \geq 2 \vee x \leq -2.$

- ▶ Otro más:

$$\{wp(\mathbf{S3}, Q)\}$$

$$\mathbf{S3: } x := y*y$$

$$\{Q : x \geq 0\}$$

- ▶ $wp(\mathbf{S3}, Q) \equiv \text{True}.$

Precondición más débil

- ▶ Si para demostrar la validez de $\{P\}\mathbf{S}\{Q\}$ nos alcanza con probar la fórmula:

$$P \Rightarrow_L wp(S, Q)$$

- ▶ Entonces lo que necesitamos un mecanismo para obtener la wp de (S, Q) .
- ▶ Afortunadamente, existe un conjunto de **axiomas** que podemos usar para obtener la wp
- ▶ Antes de empezar a ver estos axiomas, definamos primero dos predicados: $def(E)$ y Q_E^x

Predicado $\text{def}(E)$

- **Definición.** Dada una expresión E , llamamos $\text{def}(E)$ a las condiciones necesarias para que E esté **definida**. Por ejemplo:

1. $\text{def}(x + y) \equiv \text{def}(x) \wedge \text{def}(y)$.
2. $\text{def}(x/y) \equiv \text{def}(x) \wedge (\text{def}(y) \wedge_L y \neq 0)$.
3. $\text{def}(\sqrt{x}) \equiv \text{def}(x) \wedge_L x \geq 0$.
4. $\text{def}(a[i] + 3) \equiv (\text{def}(a) \wedge \text{def}(i)) \wedge_L 0 \leq i < |a|$.

- Suponemos $\text{def}(x) \equiv \text{True}$ para todas las variables, para simplificar la notación.

- Con esta hipótesis extra:

1. $\text{def}(x + y) \equiv \text{True}$.
2. $\text{def}(x/y) \equiv y \neq 0$.
3. $\text{def}(\sqrt{x}) \equiv x \geq 0$.
4. $\text{def}(a[i] + 3) \equiv 0 \leq i < |a|$.

Predicado Q_E^x

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

1. $Q \equiv 0 \leq i < j < n \wedge_L a[i] \leq x < a[j].$
 $Q_k^i \equiv 0 \leq k < j < n \wedge_L a[k] \leq x < a[j].$
 $Q_{i+1}^i \equiv 0 \leq i + 1 < j < n \wedge_L a[i + 1] \leq x < a[j].$
2. $Q \equiv 0 \leq i < n \wedge_L (\forall j : \mathbb{Z})(a[j] = x).$
 $Q_k^j \equiv 0 \leq i < n \wedge_L (\forall j : \mathbb{Z})(a[j] = x).$

Axioma 1: Asignación

► **Axioma 1.** $wp(x := E, Q) \equiv \text{def}(E) \wedge_L Q_E^x$.

► Ejemplo:

$\{??\}$

$x := x + 1$

$\{Q : x \geq 7\}$

► Tenemos que ...

$$\begin{aligned} wp(x := x+1, Q) &\equiv \text{def}(x+1) \wedge_L Q_{x+1}^x \\ &\equiv \text{True} \wedge_L (x+1) \geq 7 \\ &\equiv x \geq 6 \end{aligned}$$

Axioma 1: Asignación

- Este axioma está **justificado** por la siguiente observación. Si buscamos la precondition más débil para el siguiente programa ...

$\{??\}$

$\mathbf{x} := \mathbf{E}$

$\{Q : x = 25\}$

- ... entonces tenemos $wp(\mathbf{x} := \mathbf{E}, Q) \equiv \text{def}(E) \wedge_L E = 25$.
- Es decir, si luego de $\mathbf{x} := \mathbf{E}$ queremos que $x = 25$, entonces se debe cumplir $E = 25$ **antes** de la asignación!

Axioma 1: Asignación

- ▶ Otro ejemplo:

$$\begin{array}{c} \{??\} \\ \mathbf{x} := 2 * |\mathbf{x}| + 1 \\ \{Q : x \geq 5\} \end{array}$$

- ▶ Tenemos que ...

$$\begin{aligned} wp(\mathbf{x} := 2 * |\mathbf{x}| + 1, Q) &\equiv \text{def}(2 * |\mathbf{x}| + 1) \wedge_L Q_{2*|\mathbf{x}|+1}^x \\ &\equiv \text{True} \wedge_L 2 * |\mathbf{x}| + 1 \geq 5 \\ &\equiv |\mathbf{x}| \geq 2 \\ &\equiv x \geq 2 \vee x \leq -2 \end{aligned}$$

Axioma 1: Asignación

- Un ejemplo más:

$\{??\}$

$\mathbf{x} := \mathbf{y} * \mathbf{y}$

$\{Q : x \geq 0\}$

- Tenemos que ...

$$\begin{aligned} wp(\mathbf{x} := \mathbf{y} * \mathbf{y}, Q) &\equiv \text{def}(y * y) \wedge_L Q_{y*y}^x \\ &\equiv \text{True} \wedge_L y * y \geq 0 \\ &\equiv \text{True} \end{aligned}$$

Demostraciones de corrección

- ▶ Dijimos que $\{P\} \mathbf{S} \{Q\}$ sii $P \Rightarrow_L wp(\mathbf{S}, Q)$.
- ▶ Es decir, queremos que $P \Rightarrow_L wp(\mathbf{S}, Q)$ capture el hecho de que si \mathbf{S} comienza en un estado que satisface P , entonces termina y lo hace en un estado que satisface Q .
- ▶ Por ejemplo, la siguiente tripla de Hoare es **válida** ...

$$\{P : x \geq 10\}$$

$$\mathbf{S}: x := x+3$$

$$\{Q : x \neq 4\}$$

- ▶ ... puesto que:
 - ▶ $wp(\mathbf{S}, Q) \equiv x \neq 1$ y
 - ▶ $x \geq 10 \Rightarrow_L x \neq 1$.

Demostraciones de corrección

► La definición anterior implica que:

1. Si $P \Rightarrow_L wp(\mathbf{S}, Q)$, entonces $\{P\} \mathbf{S} \{Q\}$ es válida (i.e., es verdadera).
2. Si $P \not\Rightarrow_L wp(\mathbf{S}, Q)$, entonces $\{P\} \mathbf{S} \{Q\}$ no es válida (i.e., es falsa).

► Por ejemplo: $wp(\mathbf{x}:=\mathbf{x}+1, x \geq 7) \equiv x \geq 6$.

► Como $x \geq 4 \not\Rightarrow_L x \geq 6$ (contraejemplo, $x = 5$), entonces se concluye que

$$\{P : x \geq 4\}$$

$$\mathbf{S}: \mathbf{x} := \mathbf{x} + 1$$

$$\{Q : x \geq 7\}$$

no es válida.

Más axiomas

► **Axioma 2.** $wp(\text{skip}, Q) \equiv Q$.

► **Axioma 3.** $wp(\mathbf{S1}; \mathbf{S2}, Q) \equiv wp(\mathbf{S1}, wp(\mathbf{S2}, Q))$.

► Ejemplo:

$$\begin{aligned}\{wp(\mathbf{y} := 2*\mathbf{x}, R)\} &\equiv \{\text{def}(2 * x) \wedge_L 2 * x \geq 6\} \\ &\equiv \{x \geq 3\}\end{aligned}$$

y := 2*x;

$$\begin{aligned}\{wp(\mathbf{x} := \mathbf{y}+1, Q)\} &\equiv \{\text{def}(y + 1) \wedge_L y + 1 \geq 7\} \\ &\equiv \{y \geq 6\}\end{aligned}$$

x := y + 1

{Q : x ≥ 7}

Intercambiando los valores de dos variables

- **Ejemplo:** Recordemos el programa para intercambiar dos variables numéricas.

- $$\begin{aligned} & \{wp(\mathbf{a} := \mathbf{a} + \mathbf{b}, E_2)\} \\ & \equiv \{def(\mathbf{a} + \mathbf{b}) \wedge_L (b = B_0 \wedge (\mathbf{a} + \mathbf{b}) - \mathbf{b} = A_0)\} \\ & \equiv \{b = B_0 \wedge \mathbf{a} = A_0\} \equiv \{E_3\} \end{aligned}$$

$\mathbf{a} := \mathbf{a} + \mathbf{b};$

$$\begin{aligned} & \{wp(\mathbf{b} := \mathbf{a} - \mathbf{b}, E_1)\} \\ & \equiv \{def(\mathbf{a} - \mathbf{b}) \wedge_L (\mathbf{a} - (\mathbf{a} - \mathbf{b}) = B_0 \wedge \mathbf{a} - \mathbf{b} = A_0)\} \\ & \equiv \{b = B_0 \wedge \mathbf{a} - \mathbf{b} = A_0\} \equiv \{E_2\} \end{aligned}$$

$\mathbf{b} := \mathbf{a} - \mathbf{b};$

$$\begin{aligned} & \{wp(\mathbf{a} := \mathbf{a} - \mathbf{b}, Q)\} \\ & \equiv \{def(\mathbf{a} - \mathbf{b}) \wedge_L (\mathbf{a} - \mathbf{b} = B_0 \wedge \mathbf{b} = A_0)\} \\ & \equiv \{\mathbf{a} - \mathbf{b} = B_0 \wedge \mathbf{b} = A_0\} \equiv \{E_1\} \end{aligned}$$

$\mathbf{a} := \mathbf{a} - \mathbf{b};$

$$\{Q\} \equiv \{\mathbf{a} = B_0 \wedge \mathbf{b} = A_0\}$$

Intercambiando los valores de dos variables

- ▶ Como $P \Rightarrow E_3 \equiv wp(S, Q)$, entonces podemos concluir que el algoritmo es correcto respecto de su especificación.
- ▶ Observar que los estados intermedios que obtuvimos aplicando wp son los mismos que habíamos usado para razonar sobre la corrección de este programa!

```
{a = A0 ∧ b = B0}  
a := a + b;  
{a = A0 + B0 ∧ b = B0}  
b := a - b;  
{a = A0 + B0 ∧ b = A0}  
a := a - b;  
{a = B0 ∧ b = A0}
```

- ▶ En lugar de razonar de manera informal, ahora podemos dar una **demostración** de que estos estados describen el comportamiento del algoritmo.

Recap: Axiomas wp

- ▶ **Axioma 1.** $wp(x := E, Q) \equiv \text{def}(E) \wedge_L Q_E^x$.
- ▶ **Axioma 2.** $wp(\text{skip}, Q) \equiv Q$.
- ▶ **Axioma 3.** $wp(S1; S2, Q) \equiv wp(S1, wp(S2, Q))$.

Alternativas

- **Axioma 4.** Si $S = \text{if } B \text{ then } S1 \text{ else } S2 \text{ endif}$, entonces

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

- Ejemplo:

{??}

S: if ($x > 0$) then $y := x$ else $y := -x$ endif

{ $Q : y \geq 2$ }

- Tenemos que ...

$$\begin{aligned} wp(S, Q) &\equiv (x > 0 \wedge x \geq 2) \vee (x \leq 0 \wedge -x \geq 2) \\ &\equiv (x \geq 2) \vee (x \leq -2) \\ &\equiv |x| \geq 2 \end{aligned}$$

Alternativas

- La definición operacional que vimos para demostrar la corrección de una alternativa es ahora un **teorema** derivado de este axioma!
- **Teorema.** Si $P \Rightarrow \text{def}(B)$ y

$$\begin{array}{lll} \{P \wedge B\} & \mathbf{S1} & \{Q\} \\ \{P \wedge \neg B\} & \mathbf{S2} & \{Q\} \end{array}$$

entonces

$$\{P\} \quad \mathbf{if\ B\ then\ S1\ else\ S2\ endif} \quad \{Q\}.$$

► Demostración.

$$\begin{aligned} & [P \wedge B \Rightarrow wp(\mathbf{S1}, Q)] \wedge [P \wedge \neg B \Rightarrow wp(\mathbf{S2}, Q)] \\ \equiv & [\neg(P \wedge B) \vee wp(\mathbf{S1}, Q)] \wedge [\neg(P \wedge \neg B) \vee wp(\mathbf{S2}, Q)] \\ \equiv & [\neg P \vee \neg B \vee wp(\mathbf{S1}, Q)] \wedge [\neg P \vee B \vee wp(\mathbf{S2}, Q)] \\ \equiv & \neg P \vee ([\neg B \vee wp(\mathbf{S1}, Q)] \wedge [B \vee wp(\mathbf{S2}, Q)]) \\ \equiv & P \Rightarrow [B \Rightarrow wp(\mathbf{S1}, Q)] \wedge [\neg B \Rightarrow wp(\mathbf{S2}, Q)] \\ \equiv & P \Rightarrow [B \wedge wp(\mathbf{S1}, Q)] \vee [\neg B \wedge wp(\mathbf{S2}, Q)] \\ \equiv & P \Rightarrow \text{def}(B) \wedge_L ([B \wedge wp(\mathbf{S1}, Q)] \vee [\neg B \wedge wp(\mathbf{S2}, Q)]) \\ \equiv & P \Rightarrow wp(\text{if } B \text{ then } \mathbf{S1} \text{ else } \mathbf{S2} \text{ endif}, Q) \quad \square \end{aligned}$$

Alternativas

- En el ejemplo anterior, vimos que:

$$\{P : |x| \geq 2\}$$

S: if (x > 0) then y := x else y := -x endif

$$\{Q : y \geq 2\}$$

- Veamos ahora la validez de esta tripla de Hoare por medio del teorema anterior.

$$\begin{aligned} P \wedge B &\Rightarrow_L wp(y := x, Q) \\ |x| \geq 2 \wedge x > 0 &\Rightarrow_L def(x) \wedge_L x \geq 2 \equiv x \geq 2 \quad \checkmark \end{aligned}$$

$$\begin{aligned} P \wedge \neg B &\Rightarrow_L wp(y := -x, Q) \\ |x| \geq 2 \wedge x \leq 0 &\Rightarrow_L def(x) \wedge_L -x \geq 2 \equiv x \leq -2 \quad \checkmark \end{aligned}$$

Asignación a elementos de una secuencia

- ¿Podemos usar el Axioma 1 para el programa $b[i] := E$?
- El Axioma 1 matchea con $x := E$, pero x es una variable, no una posición de una secuencia
- Entonces, necesitamos reescribir $b[i] := E$ como $b := \text{setAt}(b, i, E)$.
- Donde

$$\begin{aligned} \text{def}(\text{setAt}(b, i, E)) &= (\text{def}(E) \wedge \text{def}(b) \wedge \text{def}(i)) \\ &\wedge_L (0 \leq i < |b|). \end{aligned}$$

- **Observación:** En el libro de Gries se usa la notación $(b; i; E)$ en lugar de $\text{setAt}(b, i, E)$

Asignación a elementos de una secuencia

- Aplicando el Axioma 1, tenemos:

$$wp(b[i] := E, Q)$$

$$\equiv wp(b := setAt(b, i, E), Q)$$

$$\equiv def(setAt(b, i, E)) \wedge_L Q_{setAt(b, i, E)}^b$$

$$\equiv ((def(b) \wedge def(i)) \wedge_L 0 \leq i < |b|) \wedge def(E) \wedge_L Q_{setAt(b, i, E)}^b$$

Además, se cumple que dados $0 \leq i, j < |b|$ sabemos que:

$$setAt(b, i, E)[j] = \begin{cases} E & \text{si } i = j \\ b[j] & \text{si } i \neq j \end{cases}$$

Asignación a elementos de una secuencia

- **Ejemplo.** Supongamos que i está definida y dentro del rango de la secuencia b .

$$\begin{aligned} & wp(\mathbf{b}[i] := 5, b[i] = 5) \\ \equiv & ((\text{def}(i) \wedge_L 0 \leq i < |b|) \wedge \text{def}(5)) \wedge_L \text{setAt}(b, i, 5)[i] = 5 \\ \equiv & \text{setAt}(b, i, 5)[i] = 5 \\ \equiv & 5 = 5 \equiv \text{True} \end{aligned}$$

- **Ejemplo.** Con las mismas hipótesis.

$$\begin{aligned} & wp(\mathbf{b}[i] := 5, b[j] = 2) \\ \equiv & \text{setAt}(b, i, 5)[j] = 2 \\ \equiv & (i \neq j \wedge \text{setAt}(b, i, 5)[j] = 2) \vee (i = j \wedge \text{setAt}(b, i, 5)[j] = 2) \\ \equiv & (i \neq j \wedge b[j] = 2) \vee (i = j \wedge \text{setAt}(b, i, 5)[i] = 2) \\ \equiv & (i \neq j \wedge b[j] = 2) \vee (i = j \wedge 5 = 2) \\ \equiv & i \neq j \wedge b[j] = 2 \end{aligned}$$

Propiedades

- ▶ Monotonía:

- ▶ Si $Q \Rightarrow R$ entonces $wp(S, Q) \Rightarrow wp(S, R)$.

- ▶ Distributividad:

- ▶ $wp(S, Q) \wedge wp(S, R) \Rightarrow wp(S, Q \wedge R)$,

- ▶ $wp(S, Q) \vee wp(S, R) \Rightarrow wp(S, Q \vee R)$.

- ▶ “*Excluded Miracle*”:

- ▶ $wp(S, false) \equiv false$.

Corolario de la monotonía

► **Corolario:** Si

► $P \Rightarrow wp(S1, Q),$

► $Q \Rightarrow wp(S2, R),$

entonces

► $P \Rightarrow wp(S1; S2, R).$

► **Demostración.**

$$\begin{array}{ll} P \Rightarrow wp(S1, Q) & \text{(por hipótesis)} \\ \Rightarrow wp(S1, wp(S2, R)) & \text{(monotonía)} \\ \equiv wp(S1; S2, R) & \text{(Axioma 2)} \end{array}$$

Ciclos (repaso)

- Recordemos la **sintaxis** de un ciclo:

```
while (guarda B) {  
    cuerpo del ciclo S  
}
```

- Se repite el cuerpo del ciclo S mientras la **guarda** B se cumpla, cero o más veces. Cada repetición se llama una **iteración**.
- La ejecución del ciclo **termina** si no se cumple la guarda al comienzo de su ejecución o bien luego de ejecutar una iteración.
- Si/cuando el ciclo termina, el estado resultante es el estado posterior a la última instrucción del cuerpo del ciclo.

¿Cuál es la precondition más débil?

{???

while (x>0) do

 x := x - 1

endwhile

{x = 0}

$$wp(\text{while } \dots, x = 0) \equiv x \geq 0$$

¿Cuál es la precondition más débil?

{???

i := 0;

while (x<5) do

 x := x + 1;

 i := i + 1

endwhile

{x = 5 ∧ i = 5}

$wp(i:=0; \text{ while } \dots, x = 5 \wedge i = 5) \equiv x = 0$

¿Cuál es la precondition más débil?

{???

```
while (x==5) do
```

```
  x := 5
```

```
endwhile
```

{ $x \neq 5$ }

$$wp(\text{while } \dots, x \neq 5) \equiv x \neq 5$$

¿Es válida la siguiente tripla de Hoare?

$$\{n \geq 0 \wedge i = 1 \wedge s = 0\}$$

```
while (i <= n) do
```

```
    s := s + i;
```

```
    i := i + 1
```

```
endwhile
```

$$\{s = \sum_{k=1}^n k\}$$

Bibliografía

- ▶ David Gries - The Science of Programming
 - ▶ Part II - The Semantics of a Small Language
 - ▶ Chapter 7 - The Predicate Transformer wp
 - ▶ Chapter 8 - The Commands skip, abort and Composition
 - ▶ Chapter 9 - The Assignment Command
 - ▶ Chapter 10 - The Alternative Command