Introducción a la Computación (Matemática)

Primer Cuatrimestre de 2018

Correctitud de Algoritmos

¿Qué vimos la clase anterior?

- Que es una especificación.
- Problemas de sobre y subespecificación.
- Terna Hoare.
- Noción de correctitud de un algoritmos/función respecto de una especificación.

¿Qué vimos la clase anterior?

- Que es una especificación.
- Problemas de sobre y subespecificación.
- Terna Hoare.
- Noción de correctitud de un algoritmos/función respecto de una especificación.

Hoy vamos a ver:

- Como probar que un algoritmos/función es correcto respecto de una especificación.
- Introducir el concepto de poscondicion más fuerte.
- Prueba de correctirud para cada tipo de sentencia.
- En particular, veremos el teorema del invariante para ciclos.

Especificación de problemas. 1

Primero recordemos que nuestro modelo de cómputo es una máquina de cambio de estado. Donde esos estados son representados por los valores que toman las variables que describen un estado en un momento dado.

Especificación de problemas. 1

Primero recordemos que nuestro modelo de cómputo es una máquina de cambio de estado. Donde esos estados son representados por los valores que toman las variables que describen un estado en un momento dado.

Programar, en un sentido abstracto, es dar la secuencia de sentencias que permiten transformar el estado inicial en uno final. Así, la ejecución de un programa puede verse como la secuencia de estado que este genera.

Especificación de problemas. 1

Primero recordemos que nuestro modelo de cómputo es una máquina de cambio de estado. Donde esos estados son representados por los valores que toman las variables que describen un estado en un momento dado.

Programar, en un sentido abstracto, es dar la secuencia de sentencias que permiten transformar el estado inicial en uno final. Así, la ejecución de un programa puede verse como la secuencia de estado que este genera.

Especificar, es establecer condiciones para pasar de un estado a otro. Esencialmente es predicar sobre las variables de estado.

Especificación de problemas. 2

La especificación de un problema se describe en un lenguaje formal:

- Precondición: indicando cuáles son los valores de las variables de entrada aceptables, y
- Poscondición: qué propiedades deben cumplir los valores de las variables de la salida.

Especificación de problemas. 2

La especificación de un problema se describe en un lenguaje formal:

- Precondición: indicando cuáles son los valores de las variables de entrada aceptables, y
- Poscondición: qué propiedades deben cumplir los valores de las variables de la salida.

Eso se interpreta de la siguiente forma: para todo estado que satisfaga la precondición, el algoritmo debe terminar exitosamente en un estado que satisfaga las propiedades especificadas en la poscondición.

Definición (Especificación) de un problema

Encabezado: nombre: $A_1 \in T_1 \times \cdots \times A_n \in T_n \to T_{RV}$
Precondición: $\{\ P\ \}$
Poscondición: $\{\ Q\ \}$

- *nombre*: nombre que le damos al problema
  - será resuelto por una función con ese mismo nombre
- Lista de argumentos:
  - Cada A_i es el nombre del argumeto.
  - Cada T_i es el tipo de datos del argumento.
  - T_{RV} es el tipo del dato de salida.
- P y Q son predicados, denominados la precondición y la poscondición del problema/función.

Argumentos

Los argumentos se comportan como variables.
Como tales, pueden ser modificadas libremente en el algoritmo.

Argumentos

Los argumentos se comportan como variables.
Como tales, pueden ser modificadas libremente en el algoritmo.

En la precondición le ponemos nombre a sus valores iniciales:

Precondición: $\{x = x_0 \wedge y = y_0\}$

Argumentos

Los argumentos se comportan como variables.
Como tales, pueden ser modificadas libremente en el algoritmo.

En la precondición le ponemos nombre a sus valores iniciales:

Precondición: $\{x = x_0 \wedge y = y_0\}$

En la poscondición predicamos sobre esos valores iniciales:

Poscondición: $\{RV = x_0 + y_0\}$

Argumentos

Los argumentos se comportan como variables.
Como tales, pueden ser modificadas libremente en el algoritmo.

En la precondición le ponemos nombre a sus valores iniciales:

Precondición: $\{x = x_0 \land y = y_0\}$

En la poscondición predicamos sobre esos valores iniciales:

Poscondición: $\{RV = x_0 + y_0\}$

Si queremos que una variable de entrada cumpla una propiedad al finalizar el algoritmo, lo especificamos en la poscondición:

Poscondición: $\{RV = x0 + y0 \ \land \ x = 8\}$

Ejemplo

Problema “sumaCuadrados”: dados dos números enteros, devolver la suma de los cuadrados de ambos.

Ejemplo

Problema "sumaCuadrados": dados dos números enteros, devolver la suma de los cuadrados de ambos.

Encabezado: $sumaCuadrados : a \in \mathbb{Z} \times b \in \mathbb{Z} \to \mathbb{Z}$
Precondición: $\{a = a_0 \wedge b = b_0\}$
Poscondición: $\{RV = a_0 * a_0 + b_0 * b_0\}$

RV es el valor de retorno del problema.

Correctitud.

Dada una especificación con precondición $\{P\}$ y poscondición $\{Q\}$, y un algoritmo A, decimos que A es correcto respecto de dicha especificación, si para todo estado que satisface la precondición $\{P\}$ la ejecución del algoritmo A a partir de ese estado inicial, termina en un estado que satisface la poscondición $\{Q\}$.

Correctitud.

Dada una especificación con precondición $\{P\}$ y poscondición $\{Q\}$, y un algoritmo A, decimos que A es correcto respecto de dicha especificación, si para todo estado que satisface la precondición $\{P\}$ la ejecución del algoritmo A a partir de ese estado inicial, termina en un estado que satisface la poscondición $\{Q\}$.
Más formalmente $\{P\}\ A\ \{Q\}$ significará que si $P, Q \subseteq S$ (conjuntos de estados) y $A \subseteq S \times S$ (una relación entre estados), entonces:

$$\forall s, s' \in S(s \in P \wedge (s, s') \in A \rightarrow s' \in Q)$$

Correctitud.

Dada una especificación con precondición $\{P\}$ y poscondición $\{Q\}$, y un algoritmo A, decimos que A es correcto respecto de dicha especificación, si para todo estado que satisface la precondición $\{P\}$ la ejecución del algoritmo A a partir de ese estado inicial, termina en un estado que satisface la poscondición $\{Q\}$.
Más formalmente $\{P\}\ A\ \{Q\}$ significará que si $P, Q \subseteq S$ (conjuntos de estados) y $A \subseteq S \times S$ (una relación entre estados), entonces:

$$\forall s, s' \in S(s \in P \land (s, s') \in A \rightarrow s' \in Q)$$

El objetivo de la clase de hoy es introducir una técnica para probar formalmente la correctitud de algoritmos.

Propiedades de una especificación Pre/Pos

Asumiendo que $\{P\}\ A\ \{Q\}$, i.e. el algoritmo A es correcto respecto de la precondición P y la poscondición Q.
Entonces tenemos que:

1. Si $P' \Rightarrow P$ entonces resulta que $\{P'\}\ A\ \{Q\}$.
2. Si $Q \Rightarrow Q'$ entonces resulta que $\{P\}\ A\ \{Q'\}$.

Propiedades de una especificación Pre/Pos

Asumiendo que $\{P\}\ A\ \{Q\}$, i.e. el algoritmo A es correcto respecto de la precondición P y la poscondición Q.
Entonces tenemos que:

1. Si $P' \Rightarrow P$ entonces resulta que $\{P'\}\ A\ \{Q\}$.
2. Si $Q \Rightarrow Q'$ entonces resulta que $\{P\}\ A\ \{Q'\}$.

Más aún, si se cumple que: $\{P_1\}\ A\ \{Q_1\}$ y $\{P_2\}\ A\ \{Q_2\}$.
Entonces:

1. $\{P_1 \wedge P_2\}\ A\ \{Q_1 \wedge Q_2\}$.
2. $\{P_1 \vee P_2\}\ A\ \{Q_1 \vee Q_2\}$.

Otro ejemplo de especificación

Encabezado: $sumarPares : A \in \mathbb{Z}[] \rightarrow \mathbb{Z}$

Precondición: $\{A = A_0\}$

Poscondición: $\{RV = \sum_{0 \leq i < |A_0| \wedge Par(A_0[i])} A_0[i]\}$

donde $Par(n) \equiv (\exists k)\, n = 2 * k$

Obs: Si no se especifica el tipo de una variable, por defecto es $\mathbb{Z}$.

Otro ejemplo de especificación

Encabezado: $sumarPares : A \in \mathbb{Z}[] \to \mathbb{Z}$
Precondición: $\{A = A_0\}$
Poscondición: $\{RV = \sum_{0 \leq i < |A_0| \wedge Par(A_0[i])} A_0[i]\}$

donde $Par(n) \equiv (\exists k)\, n = 2 * k$

Obs: Si no se especifica el tipo de una variable, por defecto es $\mathbb{Z}$.

Lo siguiente es pensar un algoritmo que satisfaga esta especificación. Lo escribimos usando pseudocógido.

Pseudocódigo

C++	**Pseudocódigo**
a = b;	a $\leftarrow$ b
if (cond) {..} else {..}	if (cond) {..} else {..}
while (cond) {..}	while (cond) {..}
! && ||	$\neg$ $\wedge$ $\vee$
== >= <=	$=$ $\geq$ $\leq$
% /	mod div
return exp;	$RV \leftarrow exp$ (*)

Pseudocódigo

C++	**Pseudocódigo**
a = b;	a $\leftarrow$ b
if (cond) {..} else {..}	if (cond) {..} else {..}
while (cond) {..}	while (cond) {..}
! && ||	$\neg$ $\wedge$ $\vee$
== >= <=	$=$ $\geq$ $\leq$
% /	mod div
return exp;	$RV \leftarrow exp$ (*)

(*) En C++, "return exp;" termina la ejecución de la función. En nuestro pseudocógido, "$RV \leftarrow exp$" es una simple asignación, tras la cual el algoritmo continúa.

Ejemplo de algoritmo en pseudocógido

Encabezado: $sumarPares : A \in \mathbb{Z}[] \rightarrow \mathbb{Z}$

Precondición: $\{A = A_0\}$

Poscondición: $\{RV = \sum_{0 \leq i < |A_0| \wedge Par(A_0[i])} A_0[i]\}$

Ejemplo de algoritmo en pseudocógido

Encabezado: $sumarPares : A \in \mathbb{Z}[] \to \mathbb{Z}$
Precondición: $\{A = A_0\}$
Poscondición: $\{RV = \sum_{0 \leq i < |A_0| \wedge Par(A_0[i])} A_0[i]\}$

$RV \leftarrow 0$
$i \leftarrow 0$
while $(i < |A|)$ {
    if $(A[i]$ mod $2 = 0)$ {
        $RV \leftarrow RV + A[i]$
    }
    $i \leftarrow i + 1$
}

donde $i \in \mathbb{Z}$.

Ejemplo de algoritmo en pseudocógido

Encabezado: $sumarPares : A \in \mathbb{Z}[] \to \mathbb{Z}$

Precondición: $\{A = A_0\}$

Poscondición: $\{RV = \sum_{0 \leq i < |A_0| \wedge Par(A_0[i])} A_0[i]\}$

$RV \leftarrow 0$
$i \leftarrow 0$
while $(i < |A|)$ {
    if $(A[i] \bmod 2 = 0)$ {
        $RV \leftarrow RV + A[i]$
    }
    $i \leftarrow i + 1$
}

donde $i \in \mathbb{Z}$.

¿Cómo sabemos si este algoritmo es correcto respecto de la especificación?

Correctitud de algoritmos. 1

Terna de Hoare

$$\{\text{precondición}\}$$
$$\text{algoritmo}$$
$$\{\text{poscondición}\}$$

Correctitud de algoritmos. 1

Terna de Hoare

{precondición}	$\{P\}$
algoritmo	A
{poscondición}	$\{Q\}$

Ejemplos:

- $\{true\}\ x \leftarrow 5\ \{x = 5\}$.
- $\{x = x_0\}\ x \leftarrow x + 3\ \{x = x_0 + 3\}$.
- $\{x > 0\}\ x \leftarrow x * 2\ \{x > -2\}$.
- $\{x = x_0\}$ if $(x < 0)$ then $x \leftarrow -x\ \{x = |x_0|\}$.
- $\{false\}\ x \leftarrow 5\ \{x = 8\}$.

Correctitud de algoritmos. 2

En general, para probar la correctitud de un algoritmo A alcanzaría con ver que para todo estado P, se lleva a uno Q.

Correctitud de algoritmos. 2

En general, para probar la correctitud de un algoritmo A alcanzaría con ver que para todo estado P, se lleva a uno Q.

Una forma de probarlo: Ver que la poscondición más fuerte de ejecutar A a partir de P, implica Q.

Correctitud de algoritmos. 2

En general, para probar la correctitud de un algoritmo A alcanzaría con ver que para todo estado P, se lleva a uno Q.

Una forma de probarlo: Ver que la poscondición más fuerte de ejecutar A a partir de P, implica Q.

- $\{x = 5\}\ x \leftarrow x * 2\ \{true\}$.
- $\{x = 5\}\ x \leftarrow x * 2\ \{x > 0\}$.
- $\{x = 5\}\ x \leftarrow x * 2\ \{x = 10 \vee x = 5\}$.
- $\{x = 5\}\ x \leftarrow x * 2\ \{x = 10\}$.

Correctitud de algoritmos. 2

En general, para probar la correctitud de un algoritmo A alcanzaría con ver que para todo estado P, se lleva a uno Q.

Una forma de probarlo: Ver que la poscondición más fuerte de ejecutar A a partir de P, implica Q.

- $\{x = 5\}\ x \leftarrow x * 2\ \{true\}$.
- $\{x = 5\}\ x \leftarrow x * 2\ \{x > 0\}$.
- $\{x = 5\}\ x \leftarrow x * 2\ \{x = 10 \vee x = 5\}$.
- $\{x = 5\}\ x \leftarrow x * 2\ \{x = 10\}$.

Dicho de otra forma, si $\{P\}A\{Q\}$ y para todo Q' tal que $\{P\}A\{Q'\}$ resulta que $Q \Rightarrow Q'$, entonces Q es la poscondición más fuerte de A respecto de $\{P\}$.

Casos de definiciones de sp.

Para probar la correctitud de un algoritmo, vamos verificar la correctitud de cada sentencia de nuestro lenguaje y comprobar que podemos hallar el sp de la composición de sentencias.

Casos de definiciones de sp.

Para probar la correctitud de un algoritmo, vamos verificar la correctitud de cada sentencia de nuestro lenguaje y comprobar que podemos hallar el sp de la composición de sentencias.
Para eso buscaremos probar que:

$$\{P\}A\{Q\} \equiv sp((A, P)) \Rightarrow Q$$

donde $sp(A, P)$ (*strongest postcondition*) es el predicado más fuerte que resulta de ejecutar A a partir de P, definiendo:

Casos de definiciones de sp.

Para probar la correctitud de un algoritmo, vamos verificar la correctitud de cada sentencia de nuestro lenguaje y comprobar que podemos hallar el sp de la composición de sentencias. Para eso buscaremos probar que:

$$\{P\}A\{Q\} \equiv sp((A, P)) \Rightarrow Q$$

donde $sp(A, P)$ (*strongest postcondition*) es el predicado más fuerte que resulta de ejecutar A a partir de P, definiendo:

- Asignación: $sp(x \leftarrow E,\ P)$.
- Secuencialización: $sp(S_1;S_2,\ P)$.
- Condicional: $sp(\text{if } (B)\ S_1 \text{ else } S_2,\ P)$.
- Ciclos: $sp(\text{while } (B)\ S,\ P)$.

Asignación

$\{P\}\ x\ \leftarrow E\ \{Q\}$

Queremos probar que $sp(x \leftarrow E,\ P) \Rightarrow Q$.

Asignación

$\{P\}\ x \leftarrow E\ \{Q\}$

Queremos probar que $sp(x \leftarrow E,\ P) \Rightarrow Q$.

Definición:

$$sp(x \leftarrow E,\ P) \equiv (\exists v)\ x = E[x : v] \wedge P[x : v]$$

donde:

- v es una variable no usada, qué representa el valor inicial de x.;
- $H[x : E]$ es la sustitución de cada instancia en H de la variable x por la expresión E.
  Ejemplo: $(x + y)[y : z^2 + 1] = (x + z^2 + 1)$.

Asignación: Ejemplo

Probar la correctitud del siguiente algoritmo respecto de su especificación:

$$P \equiv \{x > 1\}$$
$$x \leftarrow x + 1$$
$$Q \equiv \{x > 2\}$$

Asignación: Ejemplo

Probar la correctitud del siguiente algoritmo respecto de su especificación:

$$P \equiv \{x > 1\}$$
$$x \leftarrow x + 1$$
$$Q \equiv \{x > 2\}$$

$$\begin{aligned} & sp(x \leftarrow x + 1, x > 1) \equiv \\ & \quad \equiv (\exists v)\ x = (x + 1)[x : v] \wedge (x > 1)[x : v] \\ & \quad \equiv (\exists v)\ x = v + 1 \wedge v > 1 \\ & \quad \Rightarrow x > 2 \end{aligned}$$

Secuencialización

$\{P\}\ S_1;S_2\ \{Q\}$

Queremos probar que $sp(S_1;S_2,\ P) \Rightarrow Q$.

Secuencialización

$\{P\}\ S_1;S_2\ \{Q\}$

Queremos probar que $sp(S_1;S_2,\ P) \Rightarrow Q$.

Definición:

$$sp(S_1;S_2,\ P) \equiv sp(S_2, sp(S_1, P))$$

Secuencialización

$\{P\}\ S_1;S_2\ \{Q\}$

Queremos probar que $sp(S_1;S_2,\ P) \Rightarrow Q$.

Definición:

$$sp(S_1;S_2,\ P) \equiv sp(S_2, sp(S_1, P))$$

Esto da una visión composicional. La poscondición más fuerte de $S1$ es la precondición de $S2$.

Secuencialización

$\{P\}\ S_1;S_2\ \{Q\}$

Queremos probar que $sp(S_1;S_2,\ P) \Rightarrow Q$.

Definición:

$$sp(S_1;S_2,\ P) \equiv sp(S_2, sp(S_1, P))$$

Esto da una visión composicional. La poscondición más fuerte de $S1$ es la precondición de $S2$. De hecho no vamos a probarlo pero eso corresponde a la precondición más débil de $S2$.

Secuencialización: Ejemplo

$$P \equiv \{x > 1\}$$
$$x \leftarrow x + 1$$
$$y \leftarrow 2 * x$$
$$Q \equiv \{y > 4\}$$

Secuencialización: Ejemplo

$$P \equiv \{x > 1\}$$
$$x \leftarrow x + 1$$
$$y \leftarrow 2 * x$$
$$Q \equiv \{y > 4\}$$

$$\begin{aligned}
&sp(x \leftarrow x + 1; y \leftarrow 2 * x, P) \equiv \\
&\quad \equiv sp(y \leftarrow 2 * x, sp(x \leftarrow x + 1, P)) \\
&\quad \equiv sp(y \leftarrow 2 * x, (\exists a)\ x = (x + 1)[x : a] \wedge (x > 1)[x : a]) \\
&\quad \equiv sp(y \leftarrow 2 * x, (\exists a)\ x = a + 1 \wedge a > 1) \\
&\quad \equiv (\exists b)\ y = (2 * x)[y : b] \wedge ((\exists a)\ x = a + 1 \wedge a > 1)[y : b] \\
&\quad \equiv (\exists b)\ y = 2 * x \wedge (\exists a)\ x = a + 1 \wedge a > 1 \\
&\quad \equiv y = 2 * x \wedge (\exists a)\ x = a + 1 \wedge a > 1 \\
&\quad \Rightarrow y = 2 * x \wedge x > 2 \\
&\quad \Rightarrow y > 4
\end{aligned}$$

Condicional

$\{P\}$ if (B) S_1 else S_2 $\{Q\}$

Queremos probar que $sp(\text{if } (B) \ S_1 \text{ else } S_2,\ P) \Rightarrow Q$.

Condicional

$\{P\}$ if (B) S_1 else S_2 $\{Q\}$

Queremos probar que $sp(\text{if } (B) \ S_1 \text{ else } S_2,\ P) \Rightarrow Q$.

Definición:

$$sp(\text{if } (B) \ S_1 \text{ else } S_2,\ P) \equiv sp(S_1, B \wedge P) \vee sp(S_2, \neg B \wedge P)$$

Condicional

$\{P\}$ if (B) S_1 else S_2 $\{Q\}$

Queremos probar que $sp(\text{if } (B) \ S_1 \text{ else } S_2,\ P) \Rightarrow Q$.

Definición:

$$sp(\text{if } (B) \ S_1 \text{ else } S_2,\ P) \equiv sp(S_1, B \wedge P) \vee sp(S_2, \neg B \wedge P)$$

¿Qué pasa si no hay "else"?

Condicional

$\{P\}$ if (B) S_1 else S_2 $\{Q\}$

Queremos probar que $sp(\text{if } (B) \ S_1 \text{ else } S_2,\ P) \Rightarrow Q$.

Definición:

$$sp(\text{if } (B) \ S_1 \text{ else } S_2,\ P) \equiv sp(S_1, B \wedge P) \vee sp(S_2, \neg B \wedge P)$$

¿Qué pasa si no hay "else"? Formalmente, if (B) S_1 equivale a if (B) S_1 else pass, donde pass es una instrucción especial que no tiene efecto:

$$sp(\text{pass},\ P) \equiv P$$

Condicional

$\{P\}$ if (B) S_1 else S_2 $\{Q\}$

Queremos probar que $sp(\text{if } (B) \ S_1 \text{ else } S_2, \ P) \Rightarrow Q$.

Definición:

$$sp(\text{if } (B) \ S_1 \text{ else } S_2, \ P) \equiv sp(S_1, B \wedge P) \vee sp(S_2, \neg B \wedge P)$$

¿Qué pasa si no hay "else"? Formalmente, if (B) S_1 equivale a if (B) S_1 else pass, donde pass es una instrucción especial que no tiene efecto:

$$sp(\text{pass}, \ P) \equiv P$$

Por lo tanto:

$$sp(\text{if } (B) \ S_1, \ P) \equiv sp(S_1, B \wedge P) \vee (\neg B \wedge P)$$

Condicional: Ejemplo

$P \equiv \{a = a_0 \wedge b = b_0\}$
if $(a \leq b)$ {
  $RV \leftarrow 0$
} else {
  $RV \leftarrow a - b$
}
$Q \equiv \{RV = a_0 \dot{-} b_0\}$

donde $x \dot{-} y = \begin{cases} 0 & \text{si} \quad x \leq y \\ x - y & \text{si} \quad x > y \end{cases}$

Condicional: Ejemplo

$P \equiv \{a = a_0 \wedge b = b_0\}$
if $(a \leq b)$ $\{$ $RV \leftarrow 0$ $\}$ else $\{$ $RV \leftarrow a - b$ $\}$
$Q \equiv \{RV = a_0 \dot{-} b_0\}$

Condicional: Ejemplo

$P \equiv \{a = a_0 \wedge b = b_0\}$
if $(a \leq b)$ { $RV \leftarrow 0$ } else { $RV \leftarrow a - b$ }
$Q \equiv \{RV = a_0 \dot{-} b_0\}$

$sp(if(a \leq b)\{RV \leftarrow 0\}$ else $\{RV \leftarrow a - b\}, P) \equiv$

Condicional: Ejemplo

$P \equiv \{a = a_0 \wedge b = b_0\}$
if $(a \leq b)$ { $RV \leftarrow 0$ } else { $RV \leftarrow a - b$ }
$Q \equiv \{RV = a_0 \dot{-} b_0\}$

$sp(if(a \leq b)\{RV \leftarrow 0\} \text{ else } \{RV \leftarrow a - b\}, P) \equiv$
$\equiv sp(RV \leftarrow 0, \quad a \leq b \wedge P) \vee$
$sp(RV \leftarrow a - b, \; a > b \wedge P)$

Condicional: Ejemplo

$P \equiv \{a = a_0 \wedge b = b_0\}$
if $(a \leq b)$ { $RV \leftarrow 0$ } else { $RV \leftarrow a - b$ }
$Q \equiv \{RV = a_0 \dot{-} b_0\}$

$$
\begin{aligned}
& sp(if(a \leq b)\{RV \leftarrow 0\} \text{ else } \{RV \leftarrow a - b\}, P) \equiv \\
& \quad \equiv sp(RV \leftarrow 0, \quad a \leq b \wedge P) \vee \\
& \qquad sp(RV \leftarrow a - b, \; a > b \wedge P) \\
& \quad \equiv ((\exists x)\, RV = 0[RV : x] \qquad \wedge (a \leq b \wedge P)[RV : x]) \vee \\
& \qquad ((\exists y)\, RV = (a - b)[RV : y] \; \wedge (a > b \wedge P)[RV : y])
\end{aligned}
$$

Condicional: Ejemplo

$P \equiv \{a = a_0 \wedge b = b_0\}$

if $(a \leq b)$ { $RV \leftarrow 0$ } else { $RV \leftarrow a - b$ }

$Q \equiv \{RV = a_0 \dot{-} b_0\}$

$$
\begin{aligned}
& sp(if(a \leq b)\{RV \leftarrow 0\} \text{ else } \{RV \leftarrow a - b\}, P) \equiv \\
& \quad \equiv sp(RV \leftarrow 0, \quad a \leq b \wedge P) \vee \\
& \qquad sp(RV \leftarrow a - b, \; a > b \wedge P) \\
& \quad \equiv ((\exists x)\, RV = 0[RV : x] \quad \wedge (a \leq b \wedge P)[RV : x]) \vee \\
& \qquad ((\exists y)\, RV = (a - b)[RV : y] \; \wedge (a > b \wedge P)[RV : y]) \\
& \quad \equiv ((\exists x)\, RV = 0 \quad \wedge (a \leq b \wedge P)) \vee \\
& \qquad ((\exists y)\, RV = (a - b) \; \wedge (a > b \wedge P))
\end{aligned}
$$

Condicional: Ejemplo

$P \equiv \{a = a_0 \wedge b = b_0\}$

if $(a \leq b)$ { $RV \leftarrow 0$ } else { $RV \leftarrow a - b$ }

$Q \equiv \{RV = a_0 \dot{-} b_0\}$

$$
\begin{aligned}
& sp(if(a \leq b)\{RV \leftarrow 0\} \text{ else } \{RV \leftarrow a - b\}, P) \equiv \\
& \quad \equiv sp(RV \leftarrow 0, \quad a \leq b \wedge P) \vee \\
& \qquad sp(RV \leftarrow a - b, \; a > b \wedge P) \\
& \quad \equiv ((\exists x)\, RV = 0[RV : x] \quad \wedge (a \leq b \wedge P)[RV : x]) \vee \\
& \qquad ((\exists y)\, RV = (a - b)[RV : y] \; \wedge (a > b \wedge P)[RV : y]) \\
& \quad \equiv ((\exists x)\, RV = 0 \quad \wedge (a \leq b \wedge P)) \vee \\
& \qquad ((\exists y)\, RV = (a - b) \; \wedge (a > b \wedge P)) \\
& \quad \equiv (RV = 0 \quad \wedge (a \leq b \wedge a = a_0 \wedge b = b_0)) \vee \\
& \qquad (RV = (a - b) \; \wedge (a > b \wedge a = a_0 \wedge b = b_0))
\end{aligned}
$$

Condicional: Ejemplo

$P \equiv \{a = a_0 \wedge b = b_0\}$
if $(a \leq b)$ { $RV \leftarrow 0$ } else { $RV \leftarrow a - b$ }
$Q \equiv \{RV = a_0 \dot{-} b_0\}$

$$
\begin{aligned}
& sp(if(a \leq b)\{RV \leftarrow 0\} \text{ else } \{RV \leftarrow a - b\}, P) \equiv \\
& \quad \equiv sp(RV \leftarrow 0, \quad a \leq b \wedge P) \vee \\
& \qquad sp(RV \leftarrow a - b, \; a > b \wedge P) \\
& \quad \equiv ((\exists x)\, RV = 0[RV : x] \qquad \wedge (a \leq b \wedge P)[RV : x]) \vee \\
& \qquad ((\exists y)\, RV = (a - b)[RV : y] \; \wedge (a > b \wedge P)[RV : y]) \\
& \quad \equiv ((\exists x)\, RV = 0 \qquad \wedge (a \leq b \wedge P)) \vee \\
& \qquad ((\exists y)\, RV = (a - b) \; \wedge (a > b \wedge P)) \\
& \quad \equiv (RV = 0 \qquad \wedge (a \leq b \wedge a = a_0 \wedge b = b_0)) \vee \\
& \qquad (RV = (a - b) \; \wedge (a > b \wedge a = a_0 \wedge b = b_0)) \\
& \quad \Rightarrow RV = a_0 \dot{-} b_0
\end{aligned}
$$

Ciclo

$\{P\}$ while (B) S $\{Q\}$

Queremos probar que $sp(\text{while } (B) \ S, \ P) \Rightarrow Q$.

$$sp(\text{while } (B) \ S, \ P) \equiv \ ?$$

Ciclo

$\{P\}$ while (B) S $\{Q\}$

Queremos probar que $sp(\text{while } (B)\ S,\ P) \Rightarrow Q$.

Para lo cual podríamos definir:

$$
\begin{array}{lrcl}
sp(\text{while } (B)\ S,\ P) & & \equiv & \neg B \wedge \exists i L_i(P) \\
\text{donde} & L_0(P) & \equiv & P \\
& L_{i+1}(P) & \equiv & sp(S, B \wedge L_i(P))
\end{array}
$$

Ciclo

$\{P\}$ while (B) S $\{Q\}$

Queremos probar que $sp(\text{while } (B)\ S,\ P) \Rightarrow Q$.

Para lo cual podríamos definir:

$$
\begin{array}{lrcl}
sp(\text{while } (B)\ S,\ P) & & \equiv & \neg B \wedge \exists i L_i(P) \\
\text{donde} & L_0(P) & \equiv & P \\
& L_{i+1}(P) & \equiv & sp(S, B \wedge L_i(P))
\end{array}
$$

... pero esta forma de encarar la correctitud de ciclos es muy complicada. Mejor veamos otra forma.

Encabezado: $sumatoria : x \in \mathbb{Z} \to \mathbb{Z}$

Precondición: $\{x = x_0 \land x_0 \geq 0\}$

Poscondición: $\{RV = \sum_{1 \leq j \leq x_0} j\}$

Encabezado: $sumatoria : x \in \mathbb{Z} \to \mathbb{Z}$

Precondición: $\{x = x_0 \wedge x_0 \geq 0\}$

Poscondición: $\{RV = \sum_{1 \leq j \leq x_0} j\}$

$RV \leftarrow 0$

$i \leftarrow 1$

while $(i \leq x)$ {

    // estado e1

    $RV \leftarrow RV + i$

    $i \leftarrow i + 1$

    // estado e2

}

donde $i \in \mathbb{Z}$.

Estados para $x_0 = 6$:

e1		e2	
RV	i	RV	i
0	1	1	2
1	2	3	3
3	3	6	4
6	4	10	5
10	5	15	6
15	6	21	7

Encabezado: $sumatoria : x \in \mathbb{Z} \to \mathbb{Z}$

Precondición: $\{x = x_0 \land x_0 \geq 0\}$

Poscondición: $\{RV = \sum_{1 \leq j \leq x_0} j\}$

$RV \leftarrow 0$

$i \leftarrow 1$

while $(i \leq x)$ {

    // estado e1

    $RV \leftarrow RV + i$

    $i \leftarrow i + 1$

    // estado e2

}

donde $i \in \mathbb{Z}$.

Estados para $x_0 = 6$:

e1		e2	
RV	i	RV	i
0	1	1	2
1	2	3	3
3	3	6	4
6	4	10	5
10	5	15	6
15	6	21	7

Obs#1: $1 \leq i \leq x + 1$ y $RV = \sum_{1 \leq j < i} j$ valen en cada paso en e1 y en e2, pero no en el medio.

Encabezado: $sumatoria : x \in \mathbb{Z} \to \mathbb{Z}$

Precondición: $\{x = x_0 \wedge x_0 \geq 0\}$

Poscondición: $\{RV = \sum_{1 \leq j \leq x_0} j\}$

$RV \leftarrow 0$

$i \leftarrow 1$

while $(i \leq x)$ {

    // estado e1

    $RV \leftarrow RV + i$

    $i \leftarrow i + 1$

    // estado e2

}

donde $i \in \mathbb{Z}$.

Estados para $x_0 = 6$:

e1		e2	
RV	i	RV	i
0	1	1	2
1	2	3	3
3	3	6	4
6	4	10	5
10	5	15	6
15	6	21	7

Obs#1: $1 \leq i \leq x + 1$ y $RV = \sum_{1 \leq j < i} j$ valen en cada paso en e1 y en e2, pero no en el medio.

Obs#2: Cuando $i = x + 1$, la ejecución del ciclo termina.

Invariante

Predicado que describe la *idea* de un ciclo.

- Vale justo antes del ciclo (antes de evaluar B por primera vez).
- Vale en cada iteración:
  - justo antes de ejecutar la primera instrucción de S, y
  - justo después de ejecutar la última instrucción de S;
  - pero no vale durante la ejecución de S.
- Vale justo después del ciclo (después de evaluar B por última vez).
- No se conoce una forma algorítmica de encontrarlo.

Ejemplo de Invariante

$RV \leftarrow 0$
$i \leftarrow 1$
// vale I
while $(i \leq x)$ {
    // vale $I \wedge B$
    $RV \leftarrow RV + i$
    $i \leftarrow i + 1$
    // vale I
}
// vale $I \wedge \neg B$

Ejemplo de Invariante

$RV \leftarrow 0$
$i \leftarrow 1$
// vale I
while $(i \leq x)$ {
    // vale $I \wedge B$
    $RV \leftarrow RV + i$
    $i \leftarrow i + 1$
    // vale I
}
// vale $I \wedge \neg B$

$$I \equiv 1 \leq i \leq x + 1 \wedge RV = \sum_{1 \leq j < i} j \wedge x = x_0 \wedge x_0 \geq 0$$

Otro ejemplo de Invariante

Enc: $creciente : A \in \mathbb{Z}[\,] \rightarrow bool$

Pre: $\{A = A_0 \wedge |A_0| > 0\}$

Pos: $\{RV = true \Leftrightarrow \big((\forall j)\ 0 \leq j < |A_0| - 1 \Rightarrow A_0[j] < A_0[j+1]\big)\}$

Otro ejemplo de Invariante

Enc: $creciente : A \in \mathbb{Z}[\,] \to bool$

Pre: $\{A = A_0 \wedge |A_0| > 0\}$

Pos: $\{RV = true \Leftrightarrow \big((\forall j)\ 0 \leq j < |A_0| - 1 \Rightarrow A_0[j] < A_0[j+1]\big)\}$

$i \leftarrow 0$

// vale I

while $(i < |A| - 1 \ \wedge \ A[i] < A[i+1])$ {

    // vale $I \wedge B$

    $i \leftarrow i + 1$

    // vale I

}

// vale $I \wedge \neg B$

$RV \leftarrow (i = |A| - 1)$

Otro ejemplo de Invariante

Enc: $creciente : A \in \mathbb{Z}[\,] \to bool$

Pre: $\{A = A_0 \wedge |A_0| > 0\}$

Pos: $\{RV = true \Leftrightarrow \big((\forall j)\ 0 \leq j < |A_0| - 1 \Rightarrow A_0[j] < A_0[j+1]\big)\}$

$i \leftarrow 0$

// vale I

while $(i < |A| - 1 \ \wedge \ A[i] < A[i+1])$ {

    // vale $I \wedge B$

    $i \leftarrow i + 1$

    // vale I

}

// vale $I \wedge \neg B$

$RV \leftarrow (i = |A| - 1)$

$I \ \equiv \ 0 \leq i \leq |A_0| - 1 \ \wedge \ \big((\forall j)\ 0 \leq j < i \Rightarrow A_0[j] < A_0[j+1]\big) \ \wedge$
$A = A_0 \ \wedge \ |A_0| > 0$

¿Cómo probamos que un ciclo termina?

Acotamos superiormente la cantidad de iteraciones del ciclo mediante una función variante (fv).

Es una función del lenguaje de especificación, de tipo $\mathbb{Z}$:

- definida a partir de las variables del programa;
- debe decrecer estrictamente en cada iteración del ciclo.

Damos una cota (c) (valor entero fijo) tal que cuando fv la alcanza, se niega la guarda y termina la ejecución del ciclo.

Ejemplo de función variante y cota

$RV \leftarrow 0$
$i \leftarrow 1$
while $(i \leq x)$ {
    $RV \leftarrow RV + i$
    $i \leftarrow i + 1$
}

Ejemplo de función variante y cota

$RV \leftarrow 0$
$i \leftarrow 1$
while $(i \leq x)$ {
    $RV \leftarrow RV + i$
    $i \leftarrow i + 1$
}

$fv = x - i$
$c = -1$

En cada iteración del ciclo, i se incrementa y x no se modifica. Por lo tanto, $x - i$ es estrictamente decreciente.

Cuando $x - i \leq c$, se niega la guarda y termina el ciclo.

Otro ejemplo de función variante y cota

$i \leftarrow 0$
while $(i < |A| - 1 \ \wedge \ A[i] < A[i+1])$ {
    $i \leftarrow i + 1$
}
$RV \leftarrow (i = |A| - 1)$

Otro ejemplo de función variante y cota

$i \leftarrow 0$
while $(i < |A| - 1 \ \wedge \ A[i] < A[i+1])$ {
    $i \leftarrow i + 1$
}
$RV \leftarrow (i = |A| - 1)$

$fv = |A| - 1 - i$
$c = 0$

En cada iteración del ciclo, i se incrementa y $|A|$ no se modifica. Por lo tanto, $|A| - 1 - i$ es estrictamente decreciente.

Cuando $|A| - 1 - i \leq c$, se niega la guarda y termina el ciclo.

Obs: El ciclo quizá termine *antes*, cuando $A[i] \geq A[i+1]$.
Sólo nos interesa *acotar* la cantidad de iteraciones.

Teorema de terminación

Sean I el invariante de un ciclo con guarda B, fv una función entera estrictamente decreciente y $c \in \mathbb{Z}$, tales que $(I \wedge fv \leq c) \Rightarrow \neg B$. Entonces el ciclo termina.

Teorema de terminación

Sean I el invariante de un ciclo con guarda B, fv una función entera estrictamente decreciente y $c \in \mathbb{Z}$, tales que $(I \wedge fv \leq c) \Rightarrow \neg B$. Entonces el ciclo termina.

Demostración

Sea fv_j el valor que toma fv luego de ejecutar el cuerpo del ciclo por j-ésima vez. Dado que $fv \in \mathbb{Z}$, $fv_j \in \mathbb{Z}$ para todo j.

Como fv es estrictamente decreciente,
$fv_0 > fv_1 > fv_2 > \ldots$,
y necesariamente existe un k tal que $fv_k \leq c$.

Dado que $(I \wedge fv \leq c) \Rightarrow \neg B$, luego de k iteraciones vale $\neg B$. Por lo tanto el ciclo termina. $\square$

Teorema de correctitud de ciclos

Sea while (B) S un ciclo con precondición P, poscondición Q, invariante I, función variante fv y cota c. Si valen:

1. $P \Rightarrow I$
2. $\{I \wedge B\}\ S\ \{I\}$ es correcto
3. fv es estrictamente decreciente
4. $(I \wedge fv \leq c) \Rightarrow \neg B$
5. $(I \wedge \neg B) \Rightarrow Q$

entonces el ciclo es correcto para su especificación (P, Q).

Teorema de correctitud de ciclos

Sea while (B) S un ciclo con precondición P, poscondición Q, invariante I, función variante fv y cota c. Si valen:

1. $P \Rightarrow I$
2. $\{I \wedge B\}\ S\ \{I\}$ es correcto
3. fv es estrictamente decreciente
4. $(I \wedge fv \leq c) \Rightarrow \neg B$
5. $(I \wedge \neg B) \Rightarrow Q$

entonces el ciclo es correcto para su especificación (P, Q).

Dem: Por hip. 1, si suponemos que antes del ciclo vale P, también vale I. Si la valuación de B es verdadera, se ejecuta S. Por hip. 2, al concluir S sigue valiendo I, y se vuelve a evaluar B. Por hip. 3 y 4 y el teorema de terminación, sabemos que este proceso ocurre un número finito de veces. Al final de cada ejecución de S sigue valiendo I. Cuando finalmente B se niega, tenemos $(I \wedge \neg B)$, lo cual por hip. 5 implica Q. $\square$

Ejemplo completo: 1. $P \Rightarrow I$

Pre: $\{x = x_0 \wedge x_0 \geq 0\}$
$RV \leftarrow 0$
$i \leftarrow 1$
while $(i \leq x)$ {
    $RV \leftarrow RV + i$
    $i \leftarrow i + 1$
}
Pos: $\{RV = \sum_{1 \leq j \leq x_0} j\}$

$$I \equiv 1 \leq i \leq x + 1 \wedge RV = \sum_{1 \leq j < i} j \wedge x = x_0 \wedge x_0 \geq 0$$

Ejemplo completo: 1. $P \Rightarrow I$

Pre: $\{x = x_0 \wedge x_0 \geq 0\}$
$RV \leftarrow 0$
$i \leftarrow 1$
while $(i \leq x)$ {
$\quad RV \leftarrow RV + i$
$\quad i \leftarrow i + 1$
}
Pos: $\{RV = \sum_{1 \leq j \leq x_0} j\}$

$$I \equiv 1 \leq i \leq x + 1 \wedge RV = \sum_{1 \leq j < i} j \wedge x = x_0 \wedge x_0 \geq 0$$

Queremos ver que $sp(i \leftarrow 1, sp(RV \leftarrow 0, Pre)) \Rightarrow I$.

$sp(i \leftarrow 1, sp(RV \leftarrow 0, Pre)) \equiv ...$
$\equiv (RV = 0 \wedge i = 1 \wedge x = x_0 \wedge x_0 \geq 0)$

Ejemplo completo: 1. $P \Rightarrow I$

Pre: $\{x = x_0 \wedge x_0 \geq 0\}$
$RV \leftarrow 0$
$i \leftarrow 1$
while $(i \leq x)$ {
$\quad RV \leftarrow RV + i$
$\quad i \leftarrow i + 1$
}
Pos: $\{RV = \sum_{1 \leq j \leq x_0} j\}$

$$I \equiv 1 \leq i \leq x + 1 \wedge RV = \sum_{1 \leq j < i} j \wedge x = x_0 \wedge x_0 \geq 0$$

Queremos ver que $sp(i \leftarrow 1, sp(RV \leftarrow 0, Pre)) \Rightarrow I$.

$sp(i \leftarrow 1, sp(RV \leftarrow 0, Pre)) \equiv ...$
$\equiv (RV = 0 \wedge i = 1 \wedge x = x_0 \wedge x_0 \geq 0)$

$i = 1 \wedge x = x_0 \wedge x_0 \geq 0$ implica $1 \leq i \leq x + 1$.

$RV = 0 \wedge i = 1$ implica $RV = \sum_{1 \leq j < i} j$. $\square$

Ejemplo completo: 2. $\{I \wedge B\}\ S\ \{I\}$

$RV \leftarrow 0$
$i \leftarrow 1$
while $(i \leq x)$ {
    $RV \leftarrow RV + i$
    $i \leftarrow i + 1$
}

$$I \equiv 1 \leq i \leq x + 1 \wedge RV = \sum_{1 \leq j < i} j \wedge x = x_0 \wedge x_0 \geq 0$$

Ejemplo completo: 2. $\{I \wedge B\}\ S\ \{I\}$

$RV \leftarrow 0$
$i \leftarrow 1$
while $(i \leq x)$ {
    $RV \leftarrow RV + i$
    $i \leftarrow i + 1$
}

$$I \equiv 1 \leq i \leq x+1 \wedge RV = \sum_{1 \leq j < i} j \wedge x = x_0 \wedge x_0 \geq 0$$

$sp(RV \leftarrow RV + 1; i \leftarrow i + 1,\ I \wedge B) \equiv$
$\equiv sp(i \leftarrow i + 1, sp(RV \leftarrow RV + i,\ I \wedge B)) \equiv ... \equiv$
$\equiv (\exists b)\ i = b + 1\ \wedge\ (\exists a)\ RV = a + b\ \wedge\ 1 \leq b \leq x + 1\ \wedge$
$a = \sum_{1 \leq j < b} j\ \wedge\ x = x_0\ \wedge\ x_0 \geq 0\ \wedge\ b \leq x$

Ejemplo completo: 2. $\{I \wedge B\}\ S\ \{I\}$

$RV \leftarrow 0$
$i \leftarrow 1$
while $(i \leq x)$ {
  $RV \leftarrow RV + i$
  $i \leftarrow i + 1$
}

$I \equiv 1 \leq i \leq x+1 \wedge RV = \sum_{1 \leq j < i} j \wedge x = x_0 \wedge x_0 \geq 0$

$sp(RV \leftarrow RV + 1; i \leftarrow i + 1,\ I \wedge B) \equiv$
$\equiv sp(i \leftarrow i + 1, sp(RV \leftarrow RV + i,\ I \wedge B)) \equiv ... \equiv$
$\equiv (\exists b)\ i = b + 1\ \wedge\ (\exists a)\ RV = a + b\ \wedge\ 1 \leq b \leq x + 1\ \wedge$
$a = \sum_{1 \leq j < b} j\ \wedge\ x = x_0\ \wedge\ x_0 \geq 0\ \wedge\ b \leq x$

$i = b + 1 \wedge 1 \leq b \leq x + 1 \wedge b \leq x$ implica $1 \leq i \leq x + 1$.

$i = b + 1 \wedge RV = a + b \wedge a = \sum_{1 \leq j < b} j$ implica $RV = \sum_{1 \leq j < i} j$. $\square$

Ejemplo completo: 3. fv es estrict. decreciente

$RV \leftarrow 0$
$i \leftarrow 1$
while $(i \leq x)$ {
    $RV \leftarrow RV + i$
    $i \leftarrow i + 1$
}

$fv = x - i$

En cada iteración del ciclo, i se incrementa y x no se modifica.
Por lo tanto, $x - i$ es estrictamente decreciente. $\square$

Ejemplo completo: 4. $(I \wedge fv \leq c) \Rightarrow \neg B$

$RV \leftarrow 0$
$i \leftarrow 1$
while $(i \leq x)$ {
    $RV \leftarrow RV + i$
    $i \leftarrow i + 1$
}

$fv = x - i$
$c = -1$

$x - i \leq -1$ implica $x \leq i - 1$, o sea $x < i$, lo cual equivale a $\neg B$. $\square$

Obs: En este caso I no fue necesaria para la demostración, pero en el caso general sí hace falta.

Ejemplo completo: 5. $(I \wedge \neg B) \Rightarrow Q$

Pre: $\{x = x_0 \wedge x_0 \geq 0\}$
$RV \leftarrow 0$
$i \leftarrow 1$
while $(i \leq x)$ {
$\quad RV \leftarrow RV + i$
$\quad i \leftarrow i + 1$
}
Pos: $\{RV = \sum_{1 \leq j \leq x_0} j\}$

$$I \equiv 1 \leq i \leq x + 1 \wedge RV = \sum_{1 \leq j < i} j \wedge x = x_0 \wedge x_0 \geq 0$$

Queremos ver que $I \wedge \neg B \Rightarrow$ Pos.

Ejemplo completo: 5. $(I \wedge \neg B) \Rightarrow Q$

Pre: $\{x = x_0 \wedge x_0 \geq 0\}$
$RV \leftarrow 0$
$i \leftarrow 1$
while $(i \leq x)$ {
$\quad RV \leftarrow RV + i$
$\quad i \leftarrow i + 1$
}
Pos: $\{RV = \sum_{1 \leq j \leq x_0} j\}$

$$I \equiv 1 \leq i \leq x+1 \wedge RV = \sum_{1 \leq j < i} j \wedge x = x_0 \wedge x_0 \geq 0$$

Queremos ver que $I \wedge \neg B \Rightarrow$ Pos.

$I \wedge \neg B \equiv 1 \leq i \leq x+1 \wedge RV = \sum_{1 \leq j < i} j \wedge x = x_0 \wedge x_0 \geq 0 \wedge i > x$

$1 \leq i \leq x+1 \wedge i > x \wedge x = x_0$ implica $i = x_0 + 1$.

$i = x_0 + 1 \wedge RV = \sum_{1 \leq j < i} j$ implica $RV = \sum_{1 \leq j \leq x_0} j$. $\square$

Ejemplo completo: Grande Finale

Ya probamos los siguientes puntos:

1. $P \Rightarrow I$
2. $\{I \wedge B\}\ S\ \{I\}$
3. fv es estrictamente decreciente
4. $(I \wedge fv \leq c) \Rightarrow \neg B$
5. $(I \wedge \neg B) \Rightarrow Q$

Por lo tanto, según el teorema de correctitud de ciclos, el algoritmo dado es correcto respecto de la especificación. $\square$

Repaso de la clase de hoy

- Testing vs. correctitud de algoritmos
- Terna de Hoare; poscondición más fuerte (sp).
- Asignación, secuencialización, condicional.
- Correctitud de ciclos.
- Invariante, función variante.
- Teoremas de terminación y correctitud de ciclos.

Clases que vienen...

- Lu 23/4: Clase de repaso. Ejercicios y consultas.
- Ju 26/4: Clase de repaso. Ejercicios y consultas.
- Jueves 3/5: Primer parcial, en el horario de la materia (13:00) y en lugar a confirmar.