Análisis y Diseño de Algoritmos

Tema 2: Especificación de Algoritmos

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA

Contenido

- Especificación vs Implementación
- Especificación pre/post
- Predicados lógicos
- Ejemplos
- Referencias

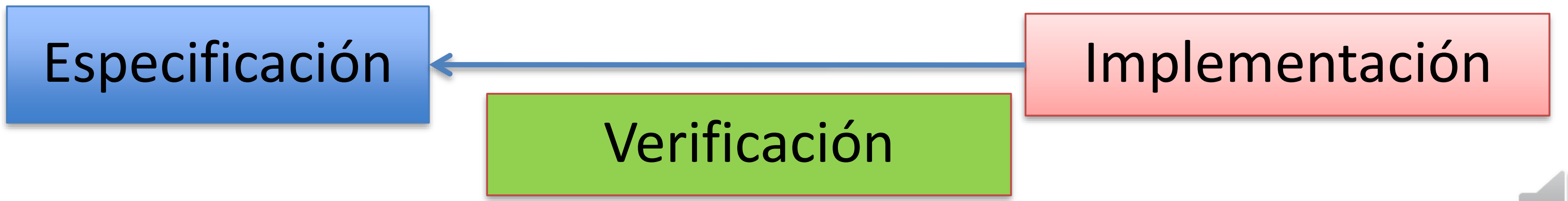
Especificación vs Implementación

- La especificación de un algoritmo es la descripción de qué es lo que hace y bajo qué condiciones lo hace.

- La implementación de un algoritmo es la descripción de la secuencia de instrucciones que hacen que la especificación se satisfaga.

- La especificación es importante para
  - Usuarios del programa: indica cómo utilizar el algoritmo.
  - Implementadores: indica las implementaciones válidas del algoritmo.

- Una especificación precisa descrita en un lenguaje lógico matemático permite la demostración de que el algoritmo es correcto y satisface la especificación (verificación).



Estado de un Algoritmo (I)

- El estado de un algoritmo se puede considerar como una función σ que asocia a cada variable del programa uno de los posibles valores que dicha variable puede tomar (tipo de datos)

$$\sigma : (v_1, v_2, \ldots v_n) \to (Tipo(v_1), Tipo(v_2), \ldots, Tipo(v_n))$$

- Ejemplo: Dado un algoritmo que utilice dos variables $a \in \mathbb{N}$ y $b \in \mathbb{Z}$, posibles estados $\sigma_i : (a, b) \to (\mathbb{N}, \mathbb{Z})$ son:

$$\sigma_1(a, b) = (0{,}500)$$
$$\sigma_2 : (a, b) = (0{,}0)$$
$$\sigma_3 : (a, b) = (10, -3)$$

Estado de un algoritmo (II)

- La ejecución de un programa puede definirse como una secuencia de estados.

- Para valores de entrada a = 5 y b = 2, y si las variables se inicializan a 0 por defecto, la ejecución del algoritmo determina los siguientes estados

$$\sigma: (a, b, q, r) \rightarrow (int, int, int, int)$$

```
Entradas: int a , int  b
(1) int q = 0;
(2) int r = a;
(3) while (r >= b){
(4)     r = r - b;
(5)     q++;
    }
Salidas: q, r
```

1, (a=5, b =2, q = 0, r = 0)
2, (a = 5, b =2, q = 0, r = 0)
3, (a = 5, b =2, q = 0, r = 5)
4, (a = 5, b =2, q = 0, r = 5)
5, (a = 5, b =2, q = 0, r = 3)
3, (a = 5, b =2, q = 1, r = 3)
4, (a = 5, b =2, q = 1, r = 3)
5, (a = 5, b =2, q = 1, r = 1)
3, (a = 5, b =2, q = 2, r = 1)
6, (a = 5, b =2, q = 2, r = 1)

Asertos

- Un aserto es una expresión sobre las variables de un programa que se evalúa a cierto o falso (predicado)

- Un estado satisface un aserto *A* ($\sigma \vDash A$), si *A* es cierto cuando se sustituyen las variables del aserto por los valores definidos en σ.

- Incluir asertos en el algoritmo permite verificar su corrección. Al menos se necesita indicar:
  - Restricciones de los valores de entrada.
  - Cuándo son correctos los valores de salida.

```
{a ≥ 0, b>0}
Entradas: int a , int  b
(1) int q = 0;
(2) int r = a;
(3) while (r >= b){
(4)     r = r - b;
(5)     q++;
    }
Salidas: q, r
{a = qb + r,r < b}
```

- La coma ‘,’ en un aserto representa la conjunción lógica (and).
- El estado

$$\sigma: (a, b, q, r) \rightarrow (int, int, int, int)$$
$$\sigma(a, b, q, r) = (5,2,2,1)$$

satisface el aserto $\{a = qb + r, r < b\}$,

porque $5 = 2 \cdot 2 + 1$ y $1 < 2$

Especificación pre/post (I)

- Una especificación pre/post es una terna de tipo *{Q} S {R}* donde
  - *{Q}* es el aserto precondición, que caracteriza los estados iniciales válidos.
  - *S* es un programa, algoritmo o secuencia de instrucciones.
  - *{R}* es el aserto postcondición, que establece la relación válida entre los datos de entrada y los de salida.

- Una especificación pre/post se interpreta como:

> Si *S* empieza a ejecutarse en un estado que satisface *{Q}*, entonces **termina** su ejecución y lo hace en un estado que satisface *{R}*.

- Si se utilizan valores de entrada no contemplados en la precondición no es posible afirmar si la ejecución acabará, ni cuál será el resultado final.

Especificación pre/post (II)

- Vamos a considerar un algoritmo *S*, ya sea un programa completo o un trozo de otro procedimiento más complejo, como una abstracción funcional.

- No estamos interesados en los detalles de implementación. El algoritmo será una caja negra de la que definimos la interfaz
  - Variables de entrada ($\downarrow$)
  - Variables de salida ($\uparrow$)
  - Variables de entrada/salida ($\downarrow \uparrow$)
  - Nombre del algoritmo.

$\{a >= 0, b > 0\}$

fun Division ($\downarrow$a , b : $\mathbb{Z}$, $\uparrow$q, r : $\mathbb{Z}$)

$\{a = q \cdot b + r, \; r < b\}$

Especificación pre/post (III)

- Otra notación cuando *S* tiene varios parámetros de salida*:*
  fun Division (a , b : $\mathbb{Z}$) **dev** (q, r : $\mathbb{Z}$)

- Notaciones cuando *S* tiene un parámetro de salida:
  fun maximo ($\downarrow$ A[1..i] : $\mathbb{Z}$, $\uparrow$ x : $\mathbb{Z}$)

  fun maximo (A[1..i] : $\mathbb{Z}$) **dev** (x : $\mathbb{Z}$)

  fun maximo ($\downarrow$ A[1..i] : $\mathbb{Z}$) : $\mathbb{Z}$

¿Nombre de la salida?
//x
maximo(A)

- Se podría utilizar incluso la signatura en Java:
  int maximo (int [ ] A) *// x*
  – Parámetro de tipos básicos sólo de entrada.
  – Parámetros de tipo Objeto, de entrada y salida.
  – Hay que indicar el nombre del parámetro de salida.

Predicados Lógicos (I)

Un predicado lógico se construye con

1. Expresiones algebraicas construidas con las variables del programa y los operadores relacionales:
   - $<$, $\leq$, $>$, $\geq$, $=$, $\neq$
   - Se denominan predicados atómicos.

2. Uno o varios predicados, a los que se aplica un operador lógico.
   - $\neg$, $\wedge$, $\vee$, $\Longrightarrow$, $\Longleftrightarrow$, …

3. Cuantificadores sobre predicados
   - $\forall\, x \in D : P$ donde *D* define el rango de valores de *x* y *P* es un predicado.
   - $\exists\, x \in D : P$
   - Las variables asociadas a los cuantificadores se denominan ligadas, en contraposición a las variables libres.

Ejemplos

• $b^2 - 4ac \geq 0$	$b^2 - 4ac$ es positivo o cero
• $\exists n \in \mathbb{N} : j = 2^n$	*j* es una potencia de 2
• $\forall i \in \mathbb{Z}, 0 \leq i < a.length : a[i] = 0$	todas las componentes del array son cero

Predicados Lógicos (II)

Si *Var(P)* es el conjunto de variables que aparecen en el predicado *P*, los conjuntos *libres(P)* y *ligadas(P)* de variables libres y ligadas se definen como sigue:

1. Si *P* es atómico,

$$libres(P) = Var(P) \;\; y \;\; ligadas(P) = \emptyset$$

Ejemplo:

$$P \equiv (x + y + z > 0) \quad libres(P) = \{x, y, z\}, ligadas(P) = \emptyset$$

2. Si δ es un cuantificador y *D* el rango que recorre,

$$libres(\delta x \in D : P) = libres\ (P) - \{x\}$$
$$ligadas(\delta x \in D : P) = ligadas\ (P) \cup \{x\}$$

Ejemplos:

$$Q \equiv (\exists y \in \mathbb{N} : P) \quad libres(Q) = \{x, z\}, ligadas(Q) = \{y\}$$
$$R \equiv (\forall x \in \mathbb{N} : Q) \quad libres(R) = \{z\}, ligadas(Q) = \{x, y\}$$

Predicados Lógicos (III)

3. $libres(\neg P) = libres(P)\ \ y\ \ ligadas(\neg P) = ligadas(P)$

4. Si $\oplus \in \{\wedge, \vee, \Longrightarrow, \Longleftrightarrow\}$,

$$libres(P \oplus Q) = libres(P) \cup libres(Q)$$

$$ligadas(P \oplus Q) = ligadas(P) \cup ligadas(Q)$$

Ejemplo: $P_1 \equiv f(x) > 0\ ,\ \ P_2 \equiv \exists y\ \in D : g(y) = a$

$$libres(P_1 \oplus P_2)\ = \{x, a\}, \qquad ligadas(P_1 \oplus P_2) = \{y\}$$

Se deben renombrar las variables ligadas cuando
$\mathrm{libres(P)} \cap ligadas(Q) \neq \emptyset\ \ o\ libres(Q) \cap ligadas(P) \neq \emptyset$.

Ejemplo:

$$Q \equiv \forall x \in D_1 : \left(\boxed{f(x)} \vee \boxed{\exists x \in D_2 : g(x)}\right)$$

$$Q \equiv \forall x \in D_1 : \left(f(x) \vee \exists y \in D_2 : g(y)\right)$$

Predicados Lógicos (IV)

- Convenio para cuantificadores:
  - $\{a..b\} = \begin{cases} \emptyset & si\ a > b \\ \{a\} \cup \{a+1..b\} & si\ a \leq b \end{cases}$
  - Si $D = \emptyset$ entonces $\forall \alpha \in D : P \equiv Cierto$ y $\exists \alpha \in D : P \equiv Falso$

- Otros cuantificadores
  - $\sum_{\alpha \in \{a..b\}} E(\alpha) = \begin{cases} 0 & si\ a > b \\ E(a) + \sum_{\alpha \in \{a+1..b\}} E(\alpha) & si\ a \leq b \end{cases}$
  - $\prod_{\alpha \in \{a..b\}} E(\alpha) = \begin{cases} 1 & si\ a > b \\ E(a) * \prod_{\alpha \in \{a+1..b\}} E(\alpha) & si\ a \leq b \end{cases}$
  - $N_{\alpha \in \{a..b\}} P(\alpha) = \begin{cases} 0 & si\ a > b \\ 1 + N_{\alpha \in \{a+1..b\}} P(\alpha) & si\ a \leq b\ y\ P(a) \equiv Cierto \\ N_{\alpha \in \{a+1..b\}} P(\alpha) & si\ a \leq b\ y\ P(a) \equiv Falso \end{cases}$

Ejemplos Cuantificadores

- Todos los cuadrados de números reales son positivos

$$\forall i \in \mathbb{R} : i^2 > 0$$

- Hay, al menos, un elemento del array *a* que es negativo

$$\exists i \in \mathbb{Z}\,, 0 \leq i < a.length : a[i] < 0$$

- La suma de la secuencia 1, 2, 3,…,n es n(n+1)/2

$$\sum_{a=1}^{n} a = \frac{n(n+1)}{2}$$

- El logaritmo del producto de los *D* primeros términos de la sucesión s_i es la suma de sus logaritmos.

$$\log\left(\prod_{i=1}^{D} s_i\right) = \sum_{i=1}^{D} \log(s_i)$$

Ejemplos Cuantificadores

- El cuantificador de conteo sirve para contar cuantos valores del dominio de la variable ligada al cuantificador satisfacen el predicado.

- Ejemplo1: Sea C un conjunto de números enteros, el número de elementos del conjunto C cuyo cuadrado es mayor que 10 ha de ser menor o igual que el cardinal del conjunto

$$(N_{i \in C}\ i^2 > 10) \leq |C|$$

- Ejemplo2: existen exactamente dos elementos del array *a* de enteros que son *0*

$$\left(N_{i=0}^{a.length-1} a[i] = 0\right) == 2$$

Especificación de problemas

```
/**
 * @param a array arbitrario
 * @return max el mayor valor
 */
```

Q ≡ {a.length > 0}

```
int maximo(int[] a) //max
```

R ≡ $\{(\forall i \in \mathbb{N}, 0 \leq i < a.length : max \geq a[i]) \land (\exists j \in \mathbb{N}, 0 \leq j < a.length : max = a[j])\}$

```
/**
 * @param a array arbitrario
 * @return b = true, si alguna de sus componentes es igual a la suma
 *         de las que la preceden, o b = false, en otro caso
 */
```

Q ≡{a.length > 0}

```
boolean suma(int[] a) // b
```

R ≡ $\{\mathrm{b} = (\exists j \in \mathbb{N}, 0 \leq j < a.length : a[j] = \sum_{i=0}^{j-1} a[i])\}$

Especificación de problemas

```
/**
 * @param a array arbitrario
 * @return No hay variable de salida. Se modifica el array a, sustituyendo
 *         todos los elementos menores que cero por sus cuadrados */
```

Q ≡{a = A}

Contenido previo

```
void negativosCuadrado(int[] a)
```

$$R \equiv \{\forall j \in \mathbb{N}, 0 \le j < a.length : (A[j] < 0) \Longrightarrow (a[j] = A[j]^2) \wedge$$
$$\forall i \in \mathbb{N}, 0 \le i < a.length : (A[i] \ge 0) \Longrightarrow (a[i] = A[i])\}$$

Ejercicios

Dado un array *V* de números enteros, escribir la especificación de una función que compruebe si:

a) *V* tiene valores positivos en todas sus componentes.

b) *x* aparece una sola vez como componente de *V*.

c) todos los valores de *V* son distintos.

Ejercicios

Dado un array *V* de números enteros, escribir la especificación de una función que calcule:

- El número de elementos de *V* que valen *0*.
- El número de veces que aparece en *V* su primer elemento.
- El menor índice que contiene el valor *x*
  a) Imponiendo que al menos una componente de *V* es *x*.
  b) Permitiendo cualquier array no vacío de entrada y devolviendo -1 en caso de que ningún elemento sea x.

Referencias

- *Diseño de Programas: Formalismo y Abstracción*. R. Peña. Ed. Prentice-Hall
- *Algoritmos correctos y eficientes. N. Martí Oliet. Grupo editorial Garceta.*