

# Análisis y Diseño de Algoritmos

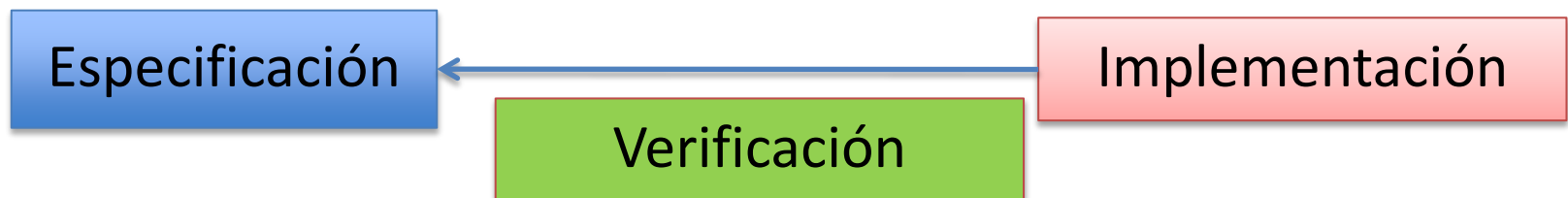
## Tema 2: Especificación

# Contenido

- Especificación vs Implementación
- Especificación formal: Motivación
- Especificación pre/post
- Predicados lógicos
- Semántica
- Especificación de predicados
- Especificación de problemas
- Conclusiones
- Referencias

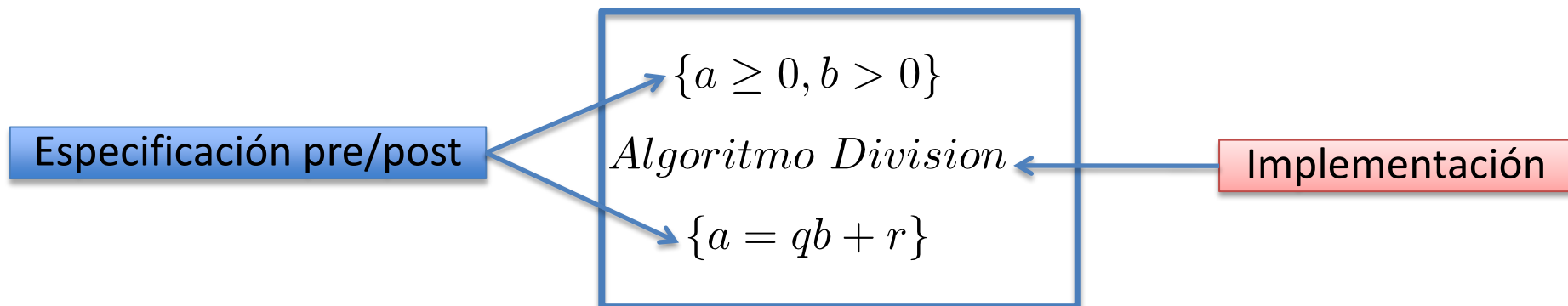
# Especificación vs Implementación

- 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 de un algoritmo es un paso necesario para su implementación
- Una especificación precisa descrita en un lenguaje lógico matemático permite la demostración de que el algoritmo satisface la especificación (verificación)



# Especificación vs Implementación

**Ejemplo** Supongamos que dados dos números naturales  $a \geq 0$  y  $b > 0$  queremos encontrar el cociente  $q$  y el resto  $r$  resultantes al dividir  $a$  entre  $b$ . Una posible especificación sería:



- La especificación está descrita con asertos que son expresiones con un valor booleano (ciertas o falsas)
- La implementación puede aparecer o no.
- La especificación es importante para los usuarios del programa/algoritmo, que pueden ser los programadores, u otros programas que hagan uso del descrito.

# Especificación formal: motivación

- Una posible implementación sería

```
 $\{a \geq 0, b > 0\}$   
(1) int q = 0;  
(2) int r = a;  
(3) while (r > b){  
(4)   r = r - b;  
(5)   q++;  
   }  
 $\{a = qb + r\}$ 
```

`a = 5, b = 2`

(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`

`a = 5, b = 2, q = 2, r = 1`

Estados  
del programa

# Especificación formal: motivación

```
 $\{a \geq 0, b > 0\}$   
(1) int q = 0;  
(2) int r = a;  
(3) while (r > b){  
(4)   r = r - b;  
(5)   q++;  
   }  
 $\{a = qb + r\}$ 
```

`a = 6, b = 2`

(3) `a = 6, b = 2, q = 0, r = 6`

(4) `a = 6, b = 2, q = 0, r = 6`

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

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

(4) `a = 6, b = 2, q = 1, r = 4`

(5) `a = 6, b = 2, q = 1, r = 2`

(3) `a = 6, b = 2, q = 2, r = 2`

`a = 6, b = 2, q = 2, r = 2 ???`

La división no  
está bien calculada

# Especificación formal: motivación

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

Modificamos la  
expresión booleana  
y el aserto

a = 6, b = 2

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

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

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

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

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

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

(3) a = 6, b = 2, q = 2, r = 2

(4) a = 6, b = 2, q = 2, r = 2

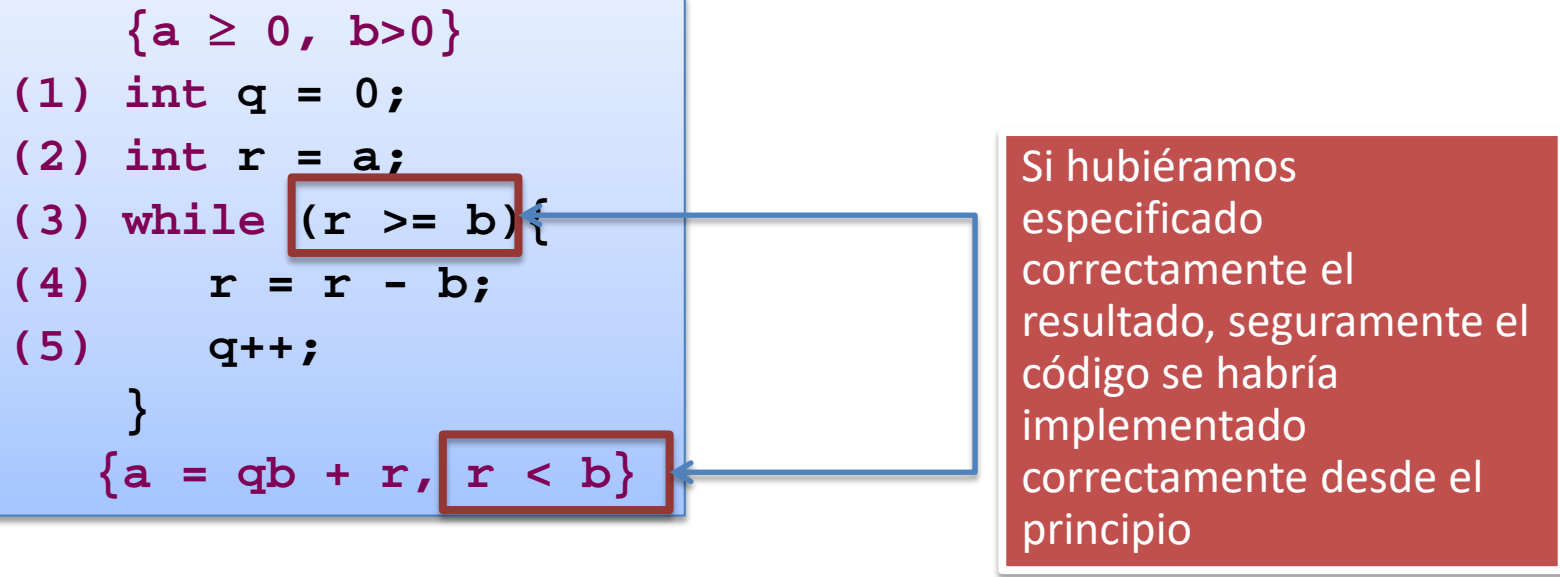
(5) a = 6, b = 2, q = 2, r = 0

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

a = 6, b = 2, q = 3, r = 0

# Especificación formal: motivación

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



Si hubiéramos especificado correctamente el resultado, seguramente el código se habría implementado correctamente desde el principio



# Especificación pre/post

**Definición:** Sea  $Var$  el conjunto de variables de un programa, y  $T$  el conjunto de todos los valores que estas variables pueden tomar, un **estado** es una función  $\sigma : Var \rightarrow T$  que asocia a cada variable un valor de su tipo.

**Ejemplo** Dado el programa *División*,  $\{a = 6, b = 2, q = 1, r = 4\}$  representa el estado del programa  $\sigma : \{a, b, q, r\} \rightarrow int$  tal que  $\sigma(a) = 6, \sigma(b) = 2, \sigma(q) = 1, \sigma(r) = 4$ .

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

a = 5, b = 2
a = 5, b = 2, q = 0, r = 5
a = 5, b = 2, q = 0, r = 5
a = 5, b = 2, q = 0, r = 3
a = 5, b = 2, q = 1, r = 3
a = 5, b = 2, q = 1, r = 3
a = 5, b = 2, q = 1, r = 1
a = 5, b = 2, q = 2, r = 1
a = 5, b = 2, q = 2, r = 1

Estados del programa

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

# Especificación pre/post

**Definición:** Un **aserto**  $A$  es una expresión sobre las variables de un programa que se evalúa a cierto o a falso.

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

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

Asertos

a = 5, b = 2

a = 5, b = 2, q = 0, r = 5

a = 5, b = 2, q = 0, r = 5

a = 5, b = 2, q = 0, r = 3

a = 5, b = 2, q = 1, r = 3

a = 5, b = 2, q = 1, r = 3

a = 5, b = 2, q = 1, r = 1

a = 5, b = 2, q = 2, r = 1

a = 5, b = 2, q = 2, r = 1

**Ejemplo** El estado  $\sigma = \{a = 5, b = 2, q = 2, r = 1\}$  satisface el aserto  $\{a = qb + r, r < b\}$  porque  $5 = 2 * 2 + 1$ , y  $1 < 2$

**Nota:** Observa que la *coma* (,) en el aserto representa la conjunción “y”.

# Especificación pre/post

**Definición:** Una **especificación pre/post** es una terna del tipo  $\{Q\}S\{R\}$  donde

1.  $\{Q\}$  es el aserto **precondición**, que caracteriza los estados iniciales válidos
2.  $\{R\}$  es el aserto **postcondición**, que establece la relación válida entre los datos de entrada y los de salida
3.  $S$  es un programa, algoritmo o una secuencia de instrucciones

## Ejemplo

$$\{a \geq 0, b > 0\} \text{ Division } \{a = qb + r, r < b\}$$

Una **especificación pre/post**  $\{Q\}S\{R\}$  se lee como:

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

La **especificación pre/post**  $\{Q\}S\{R\}$  puede utilizarse:

1. para establecer las condiciones bajo las cuales hay que diseñar el código  $S$
2. como un medio para verificar que el código  $S$  satisface las condiciones de entrada/salida  $\{Q\}$  y  $\{R\}$

# Predicados lógicos

Un predicado lógico se construye con

1. Expresiones algebraicas construidas con las variables del programa relacionadas con un operador relacional:
  - $<, \leq, >, \geq, =, \neq$
  - Los predicados contruidos sólo con estos operadores se llaman **atómicos**
2. Uno o varios predicados relacionados con un operador lógico:
  - $\neg, \vee, \wedge, \Rightarrow, \Leftrightarrow, \dots$
3. Cuantificadores sobre predicados:
  - $\forall, \exists, \sum, \prod, \dots$
  - Los cuantificadores llevan asociadas variables que se denominan *ligadas* en contraposición al resto de variables que se llaman *libres*

**Ejemplo** Algunos predicados

$$(b^2 - 4ac \geq 0)$$

*$b^2 - 4ac$  es positivo o cero*

$$(\exists n : n \geq 0 : j = 2^n)$$

*$j$  es una potencia de 2*

$$(\forall i : 0 \leq i < a.length : a[i] = 0)$$

*todas las componentes del array  $a$  son cero*

# Predicados lógicos

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 de  $P$  se definen como sigue:

1.  $libres(P) = Var(P)$  y  $ligadas(P) = \emptyset$ , si  $P$  es atómico

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

3.  $libres(P \diamond Q) = libres(P) \cup libres(Q)$ , si  $\diamond \in \{\vee, \wedge, \Rightarrow, \Leftrightarrow\}$

$ligadas(P \diamond Q) = ligadas(P) \cup ligadas(Q)$ , si  $\diamond \in \{\vee, \wedge, \Rightarrow, \Leftrightarrow\}$

Si  $libres(P) \cap ligadas(Q) \neq \emptyset$ , o  $libres(Q) \cap ligadas(P) \neq \emptyset$ , las variables ligadas deben **renombrarse**

4.  $libres(\delta x : D : P) = libres(P) - \{x\}$

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

siendo  $\delta$  un cuantificador y  $D$  el rango que recorre.

**Ejemplo.** *Algunos predicados*

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

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

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

# Predicados lógicos

**Definición:** 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 de  $P$  se definen como sigue:

1.  $libres(P) = Var(P)$  y  $ligadas(P) = \emptyset$ , si  $P$  es atómico
2.  $libres(\neg P) = libres(P)$  y  $ligadas(\neg P) = ligadas(P)$
3.  $libres(P \diamond Q) = libres(P) \cup libres(Q)$ , si  $\diamond \in \{\vee, \wedge, \Rightarrow, \Leftrightarrow\}$   
 $ligadas(P \diamond Q) = ligadas(P) \cup ligadas(Q)$ , si  $\diamond \in \{\vee, \wedge, \Rightarrow, \Leftrightarrow\}$

Si  $libres(P) \cap ligadas(Q) \neq \emptyset$ , o  $libres(Q) \cap ligadas(P) \neq \emptyset$ , las variables ligadas deben **renombrarse**

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

**Ejemplo** Considera el siguiente predicado

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

**Definición:** Dado un predicado  $P$ , y  $x \in Var(P)$ , denotamos con  $P[y/x]$  al predicado resultante de sustituir todas las apariciones de  $x$  en  $P$  por  $y$ .

**Ejemplo** El predicado  $Q$  es equivalente a

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

# Semántica

**Definición.** Dado  $P$  un predicado y  $\sigma$  un estado, denotamos con  $[[P]]\sigma$  al resultado de evaluar  $P$  en el estado  $\sigma$ , que puede ser cierto o falso.

**Ejemplo** Sean  $x, y \in Var$  dos variables, y  $\sigma = \{x = 9, y = 1\}$  un estado entonces:

$$[[x \text{ div } y > 0]]\sigma = \text{cierto}; \quad [[x > y + 7]]\sigma = \text{cierto}; \quad [[x < 0]]\sigma = \text{falso};$$

**Definición:** Dado  $P$  un predicado y un estado  $\sigma$ , decimos que  $P$  **está bien definido en**  $\sigma$  sii todas las variables del dominio de  $\sigma$  tienen un valor de su tipo, y todas las funciones de  $P$  están definidas en  $\sigma$ . En otro caso, decimos que  $P$  está **indefinido** en  $\sigma$ .

**Ejemplo** Las siguientes funciones son *parciales*:

$\text{div}, \text{mod} : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z};$  no estan definidas si el segundo argumento es cero  
 $\text{cima} : Pila \rightarrow Elem;$  no esta definida si la pila esta vacia

# Semántica

**Definición:** Sea  $\Sigma$  el conjunto de todos los estados  $\sigma$  y  $P$  un predicado/aserto, entonces

1.  $P$  es **satisfacible** sii  $\exists \sigma : \sigma \in \Sigma : \sigma \models P$
2.  $P$  es **válido** sii  $\forall \sigma : \sigma \in \Sigma : \sigma \models P$  ( $\models P$ )
3.  $P$  es una **contradicción** sii  $\forall \sigma : \sigma \in \Sigma : \sigma \not\models P$

<b>Ejemplo</b>	$(a > b^2)$ es satisfacible	$\sigma = \{a = 5, b = 2\}$
	$(b \neq 0 \Rightarrow b^2 > 0)$ es valida	—
	$(b^2 < 0)$ es contradiccion	—



# Semántica

**Definición:** Sean  $P$  y  $Q$  dos predicados, entonces

1.  $Q$  es **una consecuencia lógica de  $P$**  ( $P \models Q$ ) sii  
$$\forall \sigma : \sigma \in \Sigma : (\sigma \models P \Rightarrow \sigma \models Q)$$
2.  $\models$  es una relación de orden parcial entre predicados (asertos), que puede extenderse a conjuntos de predicados:  $\{P_1, \dots, P_n\} \models P$  sii  
$$\models P_1 \wedge \dots \wedge P_n \Rightarrow P$$

**Ejemplo** Dados los asertos  $P$  y  $Q$

$$P \equiv (a = bq + r \wedge r < 0); \quad Q \equiv (a = bq + r); \quad P \models Q$$

# Especificación con predicados

**Definición:** Dado un predicado  $P$ , el conjunto de estados definidos por  $P$  es:

$$\text{estados}(P) = \{\sigma \in \Sigma \mid [[P]]\sigma = \text{cierto}\}$$

**Definición:** Dados dos predicados  $P$  y  $Q$

1.  $P$  es más fuerte que  $Q$ , sii  $\text{estados}(P) \subseteq \text{estados}(Q)$  sii  $P \Rightarrow Q$
2.  $P$  es más débil que  $Q$ , sii  $\text{estados}(P) \supseteq \text{estados}(Q)$  sii  $Q \Rightarrow P$

**Ejemplo** Dados los predicados  $x > 0$  y  $x \geq 0$ :

$$x > 0 \Rightarrow x \geq 0$$

$$x \geq 0 \not\Rightarrow x > 0$$

**Ejemplo** Ordenar los siguientes predicados:

$$P_1 \equiv x > 0$$

$$P_2 \equiv (x > 0 \wedge y > 0)$$

$$P_3 \equiv (x > 0 \vee y > 0)$$

$$P_4 \equiv (y \geq 0)$$

$$P_5 \equiv (x \geq 0 \wedge y \geq 0)$$

# Especificación con predicados

**Definición:** Dos predicados  $P$  y  $Q$  son equivalentes ( $P \equiv Q$ ) sii los satisfacen los mismos estados, es decir, sii  $\forall \sigma : \sigma \in \Sigma : [[P]]\sigma = [[Q]]\sigma$ . La relación  $\equiv$  es una *relación de equivalencia*.

**Notación:** Si  $E_1, \dots, E_n, E$  son predicados lógicos, la **regla de inferencia**

$$\frac{E_1, \dots, E_n}{E}$$

quiere decir que *si se puede probar que  $E_1, \dots, E_n$  son ciertos, entonces se puede deducir que  $E$  es cierto*.

**Regla 1:** Transitividad

$$\frac{P_1 \equiv P_2, P_2 \equiv P_3}{P_1 \equiv P_3}$$

**Regla 2:** Sustitución

$$\frac{P_1 \equiv P_2}{Q(P_1) \equiv Q(P_2)}$$

donde  $Q(P)$  representa un predicado  $Q$  que tiene a un predicado  $P$  anidado.

# Especificación de problemas

```
/**
 * @param a array arbitrario
 * @return max el mayor valor
 */
//pre {true}
int maximo(int[] a)
//post { $\exists 0 \leq j \leq a.length : (max = a[j], \forall 0 \leq i \leq a.length : a[j] \geq a[i])$ }
```

```
/**
 * @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
 */
//pre {a.length > 0}
boolean suma(int[] a)
//post { $b = (\exists 0 \leq j \leq a.length : (a[j] = \sum_{i=0}^{j-1} a[i]))$ }
```

# Especificación de problemas

**Definición:** Las siguientes definiciones son útiles:

1. **Reforzar** un predicado  $A$  consiste en construir a partir de él otro aserto  $F$  más fuerte que él ( $F \Rightarrow A$ ). De forma dual, **debilitar** un predicado  $A$  es construir a partir de él otro predicado  $D$  más débil ( $A \Rightarrow D$ ). En los extremos, tenemos que para todo predicado  $A$ ,  $falso \Rightarrow A$  y  $A \Rightarrow cierto$
2. Siempre podemos reforzar por conjunción  $P \wedge Q \Rightarrow P$ , y debilitar por disyunción ( $P \Rightarrow P \vee Q$  y  $Q \Rightarrow P \vee Q$ )
3. Una técnica para debilitar consiste en sustituir una constante por una variable cuyo rango de valores contenga a la variable:

$$P \equiv s = \sum_{i=1}^{10} a[i]$$

puede debilitarse como

$$P' \equiv s = \sum_{i=1}^n a[i] \wedge 0 \leq n \leq 10$$

# Especificación de problemas

## Convenios y notación

- $\{a..b\} = \begin{cases} \emptyset & \text{si } a > b \\ \{a\} \cup \{a+1..b\} & \text{si } a \leq b \end{cases}$
- Si  $D = \emptyset$  entonces  $(\forall \alpha \in D : P) \equiv \text{cierto}$  y  $(\exists \alpha \in D : P) \equiv \text{falso}$
- $\sum_{\alpha \in \{a..b\}} E(\alpha) = \begin{cases} 0 & \text{si } a > b \\ E(a) + \sum_{\alpha \in \{a+1..b\}} E(\alpha) & \text{si } a \leq b \end{cases}$
- $\prod_{\alpha \in \{a..b\}} E(\alpha) = \begin{cases} 1 & \text{si } a > b \\ E(a) * \prod_{\alpha \in \{a+1..b\}} E(\alpha) & \text{si } a \leq b \end{cases}$
- $N_{\alpha \in \{a..b\}} P(\alpha) = \begin{cases} 0 & \text{si } a > b \\ 1 + N_{\alpha \in \{a+1..b\}} P(\alpha) & \text{si } a \leq b \text{ y } [[P(a)]] = \text{cierto} \\ N_{\alpha \in \{a+1..b\}} P(\alpha) & \text{si } a \leq b \text{ y } [[P(a)]] = \text{falso} \end{cases}$

# Consejos prácticos

- Antes de implementar, especificar el programa lo más formalmente que se pueda
- Ser informal en lo obvio, y riguroso en lo complejo
- Pensar en los invariantes antes de escribir nada
- Documentar el texto con asertos intermedios :
  - Invariantes de los bucles
  - Pre y post condiciones de cada método

# Conclusiones

- Aplicar estas teorías habitualmente es necesario para ahorrar costes debidos al mal funcionamiento del programa...
- Pero puede ser inabordable por la complejidad y lentitud que conlleva.
- Además, sólo hemos trabajado con enteros, booleanos y con algunos aspectos de los vectores....



# Conclusiones

- Además de los tipos estructurados, objetos,.. quedan algunas otros aspectos que estudiar:
  - Paso de parámetros
  - Programas concurrentes
  - Derivación de programas
  - Otro tipo de propiedades
  - Automatización de la verificación
  - ....

# Referencias

- *Diseño de Programas: Formalismo y Abstracción*. R. Peña. Ed. Prentice-Hall
- *The Science of Programming*. D. Gries. Ed. Springer-Verlag