METODOLOGÍA DE LA PROGRAMACIÓN

GRADO EN INGENIERÍA INFORMÁTICA DE GESTIÓN Y SISTEMAS DE INFORMACIÓN

ESCUELA DE INGENIERÍA DE BILBAO (UPV/EHU)

DEPARTAMENTO DE LENGUAJES Y SISTEMAS INFORMÁTICOS

1º

Grupo 01

TEMA 4
DERIVACIÓN DE PROGRAMAS

2022-2023

José Gaintzarain Ibarmia

Despacho: P3I 40
Tutorías: mirar en GAUR

ÍNDICE

4.1. INTRODUCCIÓN

El objetivo es construir un while a partir de la especificación pre-post dada por las fórmulas φ y ψ, el invariante INV y la expresión cota E.

```
{φ}

¿Inicializaciones?

while {INV} ¿B? loop
      ¿Instrucciones?
end loop;

{ψ}
E
```

Por tanto, habrá que calcular **"Inicialización;"**, **"B"** e **"Instrucciones"**.

Para calcular esos componentes del while, hay que seguir los seis puntos de la regla del while:

I. $\varphi \rightarrow INV$
II. $INV \rightarrow def(B)$
III. $\{INV \wedge B\}$
Instrucciones
$\{INV\}$
IV. $(INV \wedge \neg B) \rightarrow \psi$
V. $(INV \wedge B) \rightarrow E > 0$
VI. $\{INV \wedge B \wedge E = v\}$
Instrucciones
$\{E < v\}$

A) Inicialización (utilizando el punto I de la regla del while)
Para calcular la inicialización hay que comprobar si se cumple $\varphi \rightarrow INV$.
- Si se cumple $\varphi \rightarrow INV$, no hace falta inicializar nada.
- Si no se cumple $\varphi \rightarrow INV$, entonces hace falta inicializar alguna variable.

B) Calcular la condición B del while (utilizando los puntos II, IV y V de la regla del while)
- Primero se calcula $\neg B$ respondiendo a la pregunta ¿Cuándo salimos del while?
- Una vez que se tiene $\neg B$ su negación será la condición B.
- Hay que comprobar que se cumplen $INV \rightarrow def(B)$, $(INV \wedge \neg B) \rightarrow \psi$ y $(INV \wedge B) \rightarrow E > 0$, que son las implicaciones de los puntos II, IV y V de la regla del while.

C) Calcular las instrucciones que van dentro del while (utilizando los puntos III y VI de la regla del while)

4.2. EJEMPLOS

4.2.1. CALCULAR LA SUMA DE LOS ELEMENTOS DE UN VECTOR (EJERCICIO 1)

Teniendo en cuenta la regla del while (RWH) y el axioma de la asignación (AA) del Cálculo de Hoare, se va a derivar un programa que dado un vector A(1..n), calcula en la variable s la suma de los elementos de A(1..n). El programa será correcto con respecto al invariante, a la expresión cota E y a la especificación dada (precondición φ y poscondición ψ).

{φ} ≡ {n ≥ 1}
¿Inicializaciones?
while {INV} ¿B? **loop**
    ¿Instrucciones?
end loop;
{ψ} ≡ {s = $\sum_{k=1}^{n} A(k)$ }

{INV} ≡ {(1 ≤ i ≤ n + 1) ∧ s = $\sum_{k=1}^{i-1} A(k)$ }
E = n + 1 – i

Antes de empezar a derivar las instrucciones tenemos que fijarnos en la expresión cota E. Como tiene la forma "límite superior – variable índice", sabemos que el vector se ha de recorrer de izquierda a derecha.

- **Inicializaciones**
  Para obtener las inicializaciones se tiene en cuenta el primer punto de la regla del while: φ ha de implicar el invariante. Si es así, no hace falta inicializar nada, y en caso contrario sí hace falta alguna inicialización. Tras incluir una nueva inicialización se calcula su aserción previa utilizando el Axioma de la Asignación y se comprobará si la nueva fórmula es implicada por φ. El proceso de añadir nuevas inicializaciones continuará hasta obtener una fórmula que sí es implicada por φ.

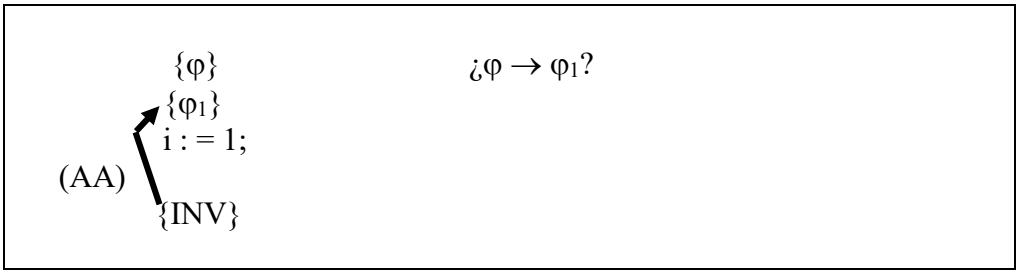
  ¿φ → INV?

$$\underbrace{n \geq 1}_{\delta} \quad \downarrow ? \quad \underbrace{(1 \leq i \leq n+1)}_{\alpha} \wedge \underbrace{s = \sum_{k=1}^{i-1} A(k)}_{\beta}$$

La implicación no se cumple porque en la primera parte de la implicación (δ) no tenemos ninguna información sobre i y s para asegurar que se cumple la segunda parte de la implicación (α y β). Como el objetivo es que la implicación se cumpla, tenemos que inicializar i y s con valores que hagan que la implicación sea cierta. Como β depende de i y s, hay que empezar fijándose en α, que solo depende de i.

Como sabemos que el vector se ha de recorrer de izquierda a derecha, para que se cumpla α basta con asignar 1 a la variable i. Por tanto, mirando en α asignamos a i el valor más pequeño que puede tomar según el invariante.

Ahora calculamos la aserción $\{\varphi_1\}$ utilizando el axioma de la asignación



$$\{\varphi_1\} \equiv \{\text{def}(1) \wedge (\text{INV})_i^1\} \equiv$$

$$\equiv \{\text{true} \wedge (1 \le 1 \le n+1) \wedge s = \sum_{k=1}^{1-1} A(k) \equiv \text{simplificación}$$

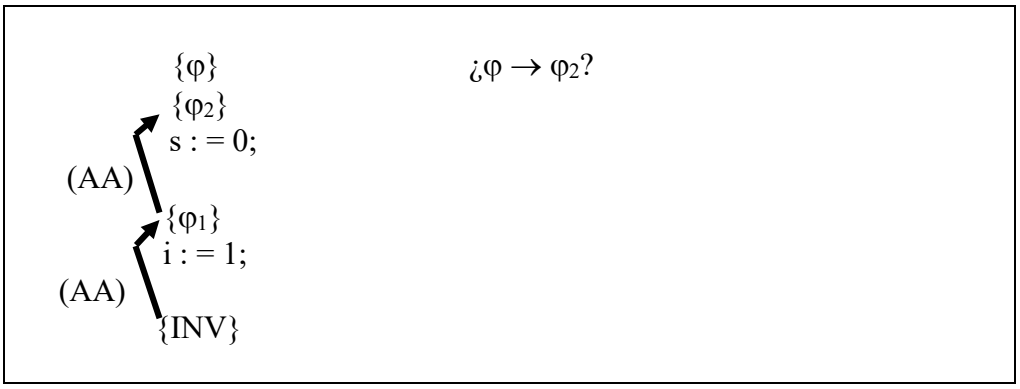
$$\equiv \{(1 \le n+1) \wedge s = \sum_{k=1}^{0} A(k)\} \equiv \text{simplificación}$$

$$\equiv \{(1 \le n+1) \wedge s = 0\}$$

- ¿$\varphi \rightarrow \varphi_1$?

$$\underbrace{n \ge 1}_{\delta} \;\downarrow ? \; \underbrace{(1 \le n+1)}_{\alpha} \wedge \underbrace{s = 0}_{\beta}$$

El que se cumpla $n \ge 1$ sí implica $1 \le n + 1$ pero no implica que se cumpla $s = 0$, es decir, lo que nos falta es que la variable s valga 0, y esa es la inicialización que nos queda.



$\{\varphi_2\} \equiv \{def(0) \wedge (\varphi_1)_s^0\} \equiv$ simplificación
$\equiv \{true \wedge (1 \leq n + 1) \wedge 0 = 0\} \equiv$ simplificación
$\equiv true \wedge (1 \leq n + 1) \wedge true \equiv$ simplificación
$\equiv (1 \leq n + 1)$

- ¿$\varphi \rightarrow \varphi_2$?

$$n \geq 1$$
$$\downarrow ?$$
$$(1 \leq n + 1)$$

La implicación $\varphi \rightarrow \varphi_2$ sí se cumple y, por tanto, ya hemos terminado con las inicializaciones.

- **<u>Condición del while (B)</u>**

Primero se calcula **¬B** respondiendo a la pregunta **¿Cuándo salimos del while?**

Sabiendo que estamos recorriendo la tabla de izquierda a derecha y que según el invariante el límite superior de i es n + 1, cuando i valga n + 1 hay que salir del while:

$\neg B \equiv i = n + 1$
Y por tanto, $B \equiv i \neq n + 1$

Ahora hay que comprobar que realmente la condición B es correcta y para ello se tendrán en cuenta los puntos II, IV y V de la regla del while.

II. ¿INV → def(B)?
¿INV → def($i \neq n + 1$)?
¿INV → true? Sí porque en la segunda parte de la implicación tenemos true.

IV. ¿(INV ∧ ¬B) → ψ?

$$\underbrace{(1 \le i \le n+1)}_{\alpha} \wedge \underbrace{s = \sum_{k=1}^{i-1} A(k)}_{\beta} \wedge \underbrace{(i = n+1)}_{\delta}$$

↓?

$$\underbrace{s = \sum_{k=1}^{n} A(k)}_{\text{por } \beta \text{ y } \delta}$$

Ya que al ser i = n + 1, i – 1 es igual a n

V. ¿INV ∧ B → E > 0?

$$(1 \le \underbrace{i \le n+1}_{\alpha}) \wedge s = \sum_{k=1}^{i-1} A(k) \wedge \underbrace{(i \ne n+1)}_{\beta}$$

↓?

$$\underbrace{n + 1 - i > 0}_{\delta}$$

δ se cumple por α y β.

- **<u>Instrucciones</u>**

  Tenemos que coger los dos programas de los puntos III y VI de la regla del while

Prog 1
{INV ∧ B}
Instrucciones
{INV}

Prog 2
{INV ∧ B ∧ E = v}
Instrucciones
{E < v}

  - Nos tenemos que preguntar
    - ✓ ¿(INV ∧ B) → INV?
    - ✓ ¿(INV ∧ B ∧ E = v) → E < v?

La implicación (INV $\wedge$ B) $\rightarrow$ INV se cumplirá siempre porque si INV y B son ciertos entonces INV será cierto.
En cambio la implicación (INV $\wedge$ B $\wedge$ E = v) $\rightarrow$ E < v no se cumplirá porque si E es igual a v entonces E no puede ser menor que v.

El que la segunda implicación no se cumpla quiere decir que tenemos que añadir una instrucción para conseguir que se cumpla E < v. Como sabemos que estamos recorriendo el vector de izquierda a derecha, para que el valor de E decrezca, tendremos que asignar i + 1 a i y calcularemos la aserción φ_3' utilizando el axioma de la asignación:

	Prog 2
AA	$\{INV \wedge B \wedge E = v\} \equiv \{(1 \leq i \leq n + 1) \wedge s = \sum_{k=1}^{i-1} A(k) \wedge (i \neq n + 1) \wedge n + 1 - i = v\}$ $\{\varphi_3'\} \equiv \{def(i + 1) \wedge (E < v)_i^{i+1}\} \equiv \{true \wedge n + 1 - (i + 1) < v\} \equiv \{n - i < v\}$ i : = i + 1; $\{n + 1 - i < v\}$

Ahora nos preguntamos ¿$(INV \wedge B \wedge E = v) \rightarrow \varphi_1$'?

$$(1 \leq i \leq n + 1) \wedge s = \sum_{k=1}^{i-1} A(k) \wedge (i \neq n + 1) \wedge \underbrace{n + 1 - i = v}_{\alpha}$$

$$\underbrace{\downarrow ? \; n - i < v}_{\beta}$$

La implicación se cumple porque β es cierto por α. Si n + 1 – i = v, entonces n – i = v – 1 y v – 1 es menor que v.

Esto quiere decir que Prog 2 ahora es correcto pero como Prog 1 y Prog 2 han de tener las mismas instrucciones tenemos que añadir la asignación i : = i + 1; a Prog 1, luego calcularemos φ_1 utilizando el axioma de la asignación y comprobaremos si se cumple (INV $\wedge$ B) $\rightarrow \varphi_3$.

	Prog 1
AA	$\{INV \wedge B\} \equiv \{(1 \le i \le n+1) \wedge s = \sum_{k=1}^{i-1} A(k) \wedge (i \ne n+1)\}$ $\{\varphi_3\} \equiv \{def(i+1) \wedge (INV)_i^{i+1}\}$ i : = i + 1; $\{INV\} \equiv \{(1 \le i \le n+1) \wedge s = \sum_{k=1}^{i-1} A(k)\}$

$$\{\varphi_3\} \equiv \{def(i+1) \wedge (INV)_i^{i+1}\} \equiv$$

$$\equiv \{true \wedge (1 \le i+1 \le n+1) \wedge s = \sum_{k=1}^{i+1-1} A(k)\} \equiv^{\text{simplificación}}$$

$$\equiv \{(0 \le i \le n) \wedge s = \sum_{k=1}^{i} A(k)\}$$

Ahora nos preguntamos ¿$(INV \wedge B) \rightarrow \varphi_3$?

$$\underbrace{(1 \le i \le n+1)}_{\alpha} \wedge \underbrace{s = \sum_{k=1}^{i-1} A(k)}_{\beta} \wedge \underbrace{(i \ne n+1)}_{\delta}$$

$$\downarrow ?$$

$$\underbrace{(0 \le i \le n)}_{\lambda} \wedge \underbrace{s = \sum_{k=1}^{i} A(k)}_{\pi}$$

- λ se cumple por α y δ.
- Por β sabemos que en s tenemos la suma de A(1..i – 1) pero en π se nos pregunta si em s tenemos la suma de A(1..i) y como la respuesta es que no, la implicación $(INV \wedge B) \rightarrow \varphi_3$ no se cumple. Como el objetivo en este momento es que φ_3 se cumpla, es decir, el objetivo es que en s se tenga la suma de los elementos de A(1..i), habrá que sumar A(i) a s.

  s : = s + A(i);

  luego calcularemos φ_4 comprobaremos si se cumple $(INV \wedge B) \rightarrow \varphi_4$

	Prog 1
AA	$\{INV \wedge B\} \equiv \{(1 \le i \le n+1) \wedge s = \sum_{k=1}^{i-1} A(k) \wedge (i \ne n+1)\}$ $\{\varphi_4\} \equiv \{def(s + A(i)) \wedge (\varphi_1)_s^{\,s + A(i)}\}$ **s : = s + A(i);** $\{\varphi_3\} \equiv \{(0 \le i \le n) \wedge s = \sum_{k=1}^{i} A(k)\}$ i : = i + 1; $\{INV\} \equiv \{(1 \le i \le n+1) \wedge s = \sum_{k=1}^{i-1} A(k)\}$

$$\{\varphi_4\} \equiv \{def(s + A(i)) \wedge (\varphi_3)_s^{\,s + A(i)}\} \equiv$$
$$\equiv \{(1 \le i \le n) \wedge (0 \le i \le n) \wedge s + A(i) = \sum_{k=1}^{i} A(k)\} \equiv \text{simplificación}$$
$$\equiv \{(1 \le i \le n) \wedge s + A(i) = \sum_{k=1}^{i} A(k)\}$$

Ahora nos preguntamos ¿$(INV \wedge B) \rightarrow \varphi_4$?

$$\underbrace{(1 \le i \le n+1)}_{\alpha} \wedge \underbrace{s = \sum_{k=1}^{i-1} A(k)}_{\beta} \wedge \underbrace{(i \ne n+1)}_{\delta}$$
$$\downarrow ?$$
$$\underbrace{(1 \le i \le n)}_{\lambda} \wedge \underbrace{s + A(i) = \sum_{k=1}^{i} A(k)}_{\pi}$$

- λ se cumple por α y δ.
- π se cumple por β.
- Por tanto la implicación $(INV \wedge B) \rightarrow \varphi_4$ se cumple.

Prog 1 ahora es correcto. Pero como Prog 1 y Prog 2 han de tener las mismas instrucciones tenemos que añadir la asignación **s : = s + A(i);** a Prog 2, luego calcular φ_4' utilizando el axioma de la asignación y comprobar si se cumple $(INV \wedge B \wedge E = v) \rightarrow \varphi_4'$.

	Prog 2
AA	$\{INV \wedge B \wedge E = v\} \equiv \{(1 \leq i \leq n+1) \wedge s = \sum_{k=1}^{i-1} A(k) \wedge (i \neq n+1) \wedge n+1-i = v\}$ $\{\varphi_4'\} \equiv \{def(s + A(i)) \wedge (\varphi_3')_s^{\,s+A(i)}\}$ **s : = s + A(i);** $\{\varphi_3'\} \equiv \{n - i < v\}$ **i : = i + 1;** $\{E < v\} \equiv \{n + 1 - i < v\}$

$$\{\varphi_4'\} \equiv \{def(s + A(i)) \wedge (\varphi_3')_s^{\,s+A(i)}\} \equiv$$
$$\equiv \{(1 \leq i \leq n) \wedge n - i < v\}$$

Ahora nos preguntamos ¿$(INV \wedge B \wedge E = v) \rightarrow \varphi_4'$?

$$\underbrace{(1 \leq i \leq n+1)}_{\alpha} \wedge s = \sum_{k=1}^{i-1} A(k) \wedge \underbrace{(i \neq n+1)}_{\beta} \wedge \underbrace{n+1-i = v}_{\delta}$$

$$\downarrow ?$$

$$\underbrace{(1 \leq i \leq n)}_{\text{por } \alpha \text{ y } \beta} \wedge \underbrace{n - i < v}_{\text{por } \delta \text{ ya que por } \delta\ n - i = v - 1}$$

Como se han cumplido las dos implicaciones, $(INV \wedge B) \rightarrow \varphi_4$ y $(INV \wedge B \wedge E = v) \rightarrow \varphi_4'$ ya hemos terminado de derivar el programa. El programa es el siguiente:

```
{φ}
{φ2}
s : = 0;
{φ1}
i : = 1;

while {INV} i ≠ n + 1 loop
      {φ4} {φ4'}
      s : = s + A(i);
      {φ3} {φ3'}
      i : = i + 1;
end loop;

{ψ}
E
```

Sabemos que el programa es **correcto** porque se ha construido siguiendo la regla del while y además el propio método calcula **documentación** del programa, que vendrá dada por $\{\varphi_1\}$, $\{\varphi_2\}$, $\{\varphi_3\}$, $\{\varphi_3'\}$, $\{\varphi_4\}$ y $\{\varphi_4'\}$.

4.2.2. CALCULAR LA SUMA DE LOS ELEMENTOS DE UN VECTOR (EJERCICIO 4)

Teniendo en cuenta la regla del while (RWH) y el axioma de la asignación (AA) del Cálculo de Hoare, se va a derivar un programa que dado un vector A(1..n), calcula en la variable s la suma de los elementos de A(1..n). El programa será correcto con respecto al invariante, a la expresión cota E y a la especificación dada (precondición φ y postcondición ψ).

$\{\varphi\} \equiv \{n \geq 1\}$ ¿Inicializaciones? {INV} **while** {INV} {E} B **loop**   ¿Instrucciones? **end loop;** $\{\psi\} \equiv \{s = \sum_{k=1}^{n} A(k)\}$
$\{INV\} \equiv \{(0 \leq i \leq n) \wedge s = \sum_{k=i+1}^{n} A(k)\}$ $E = i$

Antes de empezar a derivar las instrucciones tenemos que fijarnos en la expresión cota E. Como tiene la forma "variable índice – límite superior", sabemos que el vector se ha de recorrer de derecha a izquierda.

- **Inicializaciones**
  Para obtener las inicializaciones se tiene en cuenta el primer punto de la regla del while: φ ha de implicar el invariante. Si es así, no hace falta inicializar nada, y en caso contrario sí hace falta alguna inicialización. Tras incluir una nueva inicialización se calcula su aserción previa utilizando el Axioma de la Asignación y se comprobará si la nueva fórmula es implicada por φ. El proceso de añadir nuevas inicializaciones continuará hasta obtener una fórmula que sí es implicada por φ.

  ¿$\varphi \rightarrow INV$?

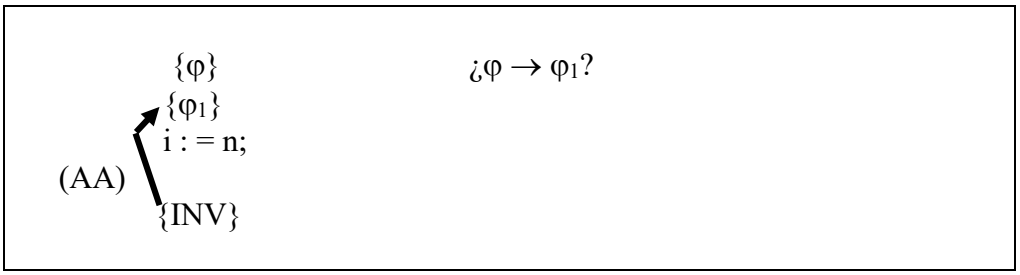
$$\underbrace{n \geq 1}_{\delta} \quad \downarrow ? \quad \underbrace{(0 \leq i \leq n)}_{\alpha} \wedge \underbrace{s = \sum_{k=i+1}^{n} A(k)}_{\beta}$$

  La implicación no se cumple porque en la primera parte de la implicación (δ) no tenemos ninguna información sobre i y s para asegurar que se cumple la segunda

parte de la implicación (α y β). Como el objetivo es que la implicación se cumpla, tenemos que inicializar i y s con valores que hagan que la implicación sea cierta. Como β depende de i y s, hay que empezar fijándose en α, que solo depende de i.

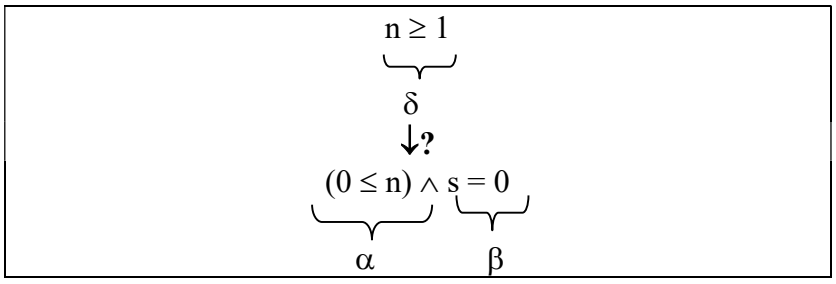
Como sabemos que el vector se ha de recorrer de derecha a izquierda, para que se cumpla α basta con asignar n a la variable i. Por tanto, mirando en α asignamos a i el mayor valor que puede tomar según el invariante.

Ahora calculamos la aserción $\{\varphi_1\}$ utilizando el axioma de la asignación

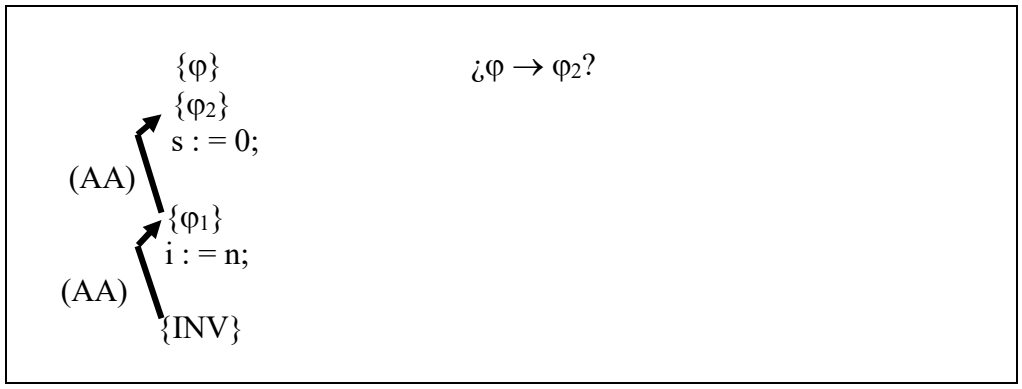


$$\{\varphi_1\} \equiv \{def(n) \wedge (INV)_i^n\} \equiv$$
$$\equiv \{true \wedge (0 \le n \le n) \wedge s = \sum_{k=n+1}^{n} A(k)\} \equiv \text{ simplificación}$$
$$\equiv \{(0 \le n) \wedge s = 0\}$$

- ¿$\varphi \rightarrow \varphi_1$?



El que se cumpla n ≥ 1 sí implica 0 ≤ n pero no implica que se cumpla s = 0, es decir, lo que nos falta es que la variable s valga 0, y esa es la inicialización que nos queda.



$\{\varphi_2\} \equiv \{def(0) \wedge (\varphi_1)_s^0\} \equiv$
$\equiv \{true \wedge (0 \leq n) \wedge 0 = 0\} \equiv$ simplificación
$\equiv true \wedge (1 \leq n + 1) \wedge true \equiv$ simplificación
$\equiv (0 \leq n)$

- ¿$\varphi \rightarrow \varphi_2$?

$n \geq 1$
$\downarrow$?
$(0 \leq n)$

La implicación $\varphi \rightarrow \varphi_2$ sí se cumple y, por tanto, ya hemos terminado con las inicializaciones.

- **Condición del while (B)**

Primero se calcula **¬B** respondiendo a la pregunta **¿Cuándo salimos del while?**

Sabiendo que estamos recorriendo la tabla de derecha a izquierda y que según el invariante el límite superior de i es n + 1, cuando i valga n + 1 hay que salir del while:

$\neg B \equiv i = 0$
Y por tanto, $B \equiv i \neq 0$

Ahora hay que comprobar que realmente la condición B es correcta y para ello se tendrán en cuenta los puntos II, IV y V de la regla del while.

II. ¿INV → def(B)?
¿INV → def(i ≠ 0)?
¿INV → true? Sí porque en la segunda parte de la implicación tenemos true.

IV. ¿(INV ∧ ¬B) → ψ?

$$\underbrace{(0 \le i \le n)}_{\alpha} \wedge \underbrace{s = \sum_{k=i+1}^{n} A(k)}_{\beta} \wedge \underbrace{(i = 0)}_{\delta}$$

↓?

$$\underbrace{s = \sum_{k=1}^{n} A(k)}_{\text{por } \beta \text{ y } \delta}$$

Ya que al ser i = 0, i + 1 es igual a 1

V. ¿INV ∧ B → E > 0?

$$\{\underbrace{(0 \le i \le n)}_{\alpha} \wedge s = \sum_{k=i+1}^{n} A(k)\} \wedge \underbrace{(i \neq 0)}_{\beta}$$

↓?

$$\underbrace{i > 0}_{\delta}$$

δ se cumple por α y β.

- **<u>Instrucciones</u>**

Tenemos que coger los dos programas de los puntos III y VI de la regla del while

Prog 1
{INV ∧ B}
¿Instrucciones?
{INV}

Prog 2
{INV ∧ B ∧ E = v}
¿Instrucciones?
{E < v}

- Nos tenemos que preguntar
  - ✓ ¿(INV ∧ B) → INV?
  - ✓ ¿(INV ∧ B ∧ E = v) → E < v?

La implicación (INV ∧ B) → INV se cumplirá siempre porque si INV y B son ciertos entonces INV será cierto.

En cambio la implicación (INV $\wedge$ B $\wedge$ E = v) $\rightarrow$ E < v no se cumplirá porque si E es igual a v entonces E no puede ser menor que v.

El que la segunda implicación no se cumpla quiere decir que tenemos que añadir una instrucción para conseguir que se cumpla E < v. Como sabemos que estamos recorriendo el vector de derecha a izquierda, para que el valor de E decrezca, tendremos que asignar i – 1 a i y calcularemos la aserción φ_3' utilizando el axioma de la asignación:

	Prog 2
EA ↑	$\{INB \wedge B \wedge E = v\} \equiv \{(0 \leq i \leq n) \wedge s = \sum_{k=i+1}^{n} A(k) \wedge (i \neq 0) \wedge i = v\}$ $\{\varphi_3'\} \equiv \{def(i-1) \wedge (E < v)_i^{i-1}\} \equiv \{true \wedge i - 1 < v\} \equiv \{i - 1 < v\}$ i : = i – 1; {i < v}

Ahora nos preguntamos ¿(INV $\wedge$ B $\wedge$ E = v) $\rightarrow$ φ_1'?

$$(0 \leq i \leq n) \wedge s = \sum_{k=i+1}^{n} A(k) \wedge (i \neq 0) \wedge \underbrace{i = v}_{\alpha}$$

$$\underbrace{i - 1 < v}_{\beta} \quad \downarrow ?$$

La implicación se cumple porque β es cierto por α. Si i = v, entonces i – 1 = v – 1 y v – 1 es menor que v.

Esto quiere decir que Prog 2 ahora es correcto pero como Prog 1 y Prog 2 han de tener las mismas instrucciones tenemos que añadir la asignación i : = i – 1; a Prog 1, luego calcularemos φ_1 utilizando el axioma de la asignación y comprobaremos si se cumple (INV $\wedge$ B) $\rightarrow$ φ_3.

	Prog 1
AA	$\{INV \wedge B\} \equiv \{(0 \le i \le n) \wedge s = \sum_{k=i+1}^{n} A(k) \wedge (i \ne 0)\}$ $\{\varphi_3\} \equiv \{def(i-1) \wedge (INV)_i^{i-1}\}$ i : = i – 1; $\{INV\} \equiv \{(0 \le i \le n) \wedge s = \sum_{k=i+1}^{n} A(k)\}$

$$\{\varphi_3\} \equiv \{def(i-1) \wedge (INV)_i^{i-1}\} \equiv$$
$$\equiv \{true \wedge (0 \le i-1 \le n) \wedge s = \sum_{k=i-1+1}^{n} A(k)\} \equiv$$
$$\equiv \{(1 \le i \le n+1) \wedge s = \sum_{k=i}^{n} A(k)\}$$

Ahora nos preguntamos ¿$(INV \wedge B) \rightarrow \varphi_3$?

$$\underbrace{(0 \le i \le n)}_{\alpha} \wedge \underbrace{s = \sum_{k=i+1}^{n} A(k)}_{\beta} \wedge \underbrace{(i \ne 0)}_{\delta}$$
$$\downarrow ?$$
$$\underbrace{(1 \le i \le n+1)}_{\lambda} \wedge \underbrace{s = \sum_{k=i}^{n} A(k)}_{\pi}$$

- λ se cumple por α y δ.
- Por β sabemos que en s tenemos la suma de A(i + 1..n) pero en π se nos pregunta si em s tenemos la suma de A(i..n) y como la respuesta es que no, la implicación $(INV \wedge B) \rightarrow \varphi_3$ no se cumple. Como el objetivo en este momento es que φ_3 se cumpla, es decir, el objetivo es que en s se tenga la suma de los elementos de A(i..n), habrá que sumar A(i) a s.

  s : = s + A(i);

  luego calcularemos φ_4 comprobaremos si se cumple $(INV \wedge B) \rightarrow \varphi_4$

	Prog 1
AA	$\{INV \wedge B\} \equiv \{(0 \le i \le n) \wedge s = \sum_{k=i+1}^{n} A(k) \wedge (i \neq 0)\}$ $\{\varphi_4\} \equiv \{def(s + A(i)) \wedge (\varphi_3)_s^{\,s + A(i)}\}$ **s : = s + A(i);** $\{\varphi_3\} \equiv \{(1 \le i \le n + 1) \wedge s = \sum_{k=i}^{n} A(k)\}$ i : = i – 1; $\{INV\} \equiv \{(0 \le i \le n) \wedge s = \sum_{k=i+1}^{n} A(k)\}$

$$\{\varphi_4\} \equiv \{def(s + A(i)) \wedge (\varphi_3)_s^{\,s + A(i)}\} \equiv$$
$$\equiv \{(1 \le i \le n) \wedge (1 \le i \le n + 1) \wedge s + A(i) = \sum_{k=i}^{n} A(k)\} \equiv$$
$$\equiv \{(1 \le i \le n) \wedge s + A(i) = \sum_{k=i}^{n} A(k)\}$$

Ahora nos preguntamos ¿$(INV \wedge B) \rightarrow \varphi_4$?

$$\underbrace{(0 \le i \le n)}_{\alpha} \wedge \underbrace{s = \sum_{k=i+1}^{n} A(k)}_{\beta} \wedge \underbrace{(i \neq 0)}_{\delta}$$
$$\downarrow ?$$
$$\underbrace{(1 \le i \le n)}_{\lambda} \wedge \underbrace{s + A(i) = \sum_{k=i}^{n} A(k)}_{\pi}$$

- λ se cumple por α y δ.
- π se cumple por β.
- Por tanto la implicación $(INV \wedge B) \rightarrow \varphi_4$ se cumple.

Prog 1 ahora es correcto. Pero como Prog 1 y Prog 2 han de tener las mismas instrucciones tenemos que añadir la asignación **s : = s + A(i);** a Prog 2, luego calcular φ_4' utilizando el axioma de la asignación y comprobar si se cumple $(INV \wedge B \wedge E = v) \rightarrow \varphi_4'$.

	Prog 2
AA	$\{INV \wedge B \wedge E = v\} \equiv \{(0 \le i \le n) \wedge s = \sum_{k=i+1}^{n} A(k) \wedge (i \neq 0) \wedge i = v\}$ $\{\varphi_4'\} \equiv \{def(s + A(i)) \wedge (\varphi_3')_s^{\,s + A(i)}\}$ **s : = s + A(i);** $\{\varphi_3'\} \equiv \{i - 1 < v\}$ **i : = i – 1;** $\{E < v\} \equiv \{i < v\}$

$$\{\varphi_4'\} \equiv \{def(s + A(i)) \wedge (\varphi_3')_s^{\,s + A(i)}\} \equiv$$
$$\equiv \{(1 \le i \le n) \wedge i - 1 < v\}$$

Ahora nos preguntamos ¿$(INV \wedge B \wedge E = v) \rightarrow \varphi_4'$?

$$\underbrace{(0 \le i \le n)}_{\alpha} \wedge s = \sum_{k=i+1}^{n} A(k) \ \underbrace{\wedge (i \neq 0)}_{\beta} \wedge \underbrace{i = v}_{\delta}$$

$$\downarrow ?$$

$$\underbrace{(1 \le i \le n)}_{\text{por } \alpha \text{ y } \beta} \wedge \underbrace{i - 1 < v}_{\text{por } \delta, \text{ ya que por } \delta\ i - 1 = v - 1 \text{ y } v - 1 \text{ es menor que } v}$$

Como se han cumplido las dos implicaciones, $(INV \wedge B) \rightarrow \varphi_4$ y $(INV \wedge B \wedge E = v) \rightarrow \varphi_4'$ ya hemos terminado de derivar el programa. El programa es el siguiente:

```
{φ}
{φ2}
s : = 0;
{φ1}
i : = n;

while {INV} i ≠ 0 loop
      {φ4} {φ4'}
      s : = s + A(i);
      {φ3} {φ3'}
      i : = i – 1;
end loop;

{ψ}
E
```

Sabemos que el programa es **correcto** porque se ha construido siguiendo la regla del while y además el propio método calcula **documentación** del programa, que vendrá dada por $\{\varphi_1\}$, $\{\varphi_2\}$, $\{\varphi_3\}$, $\{\varphi_3'\}$, $\{\varphi_4\}$ y $\{\varphi_4'\}$.

4.2.3. DEVOLVER 0 EN C SI Y SOLAMENTE SI A(1..N) CONTIENE SÓLO CEROS (EJERCICIO 7)

Teniendo en cuenta la regla del while (RWH) y el axioma de la asignación (AA) del Cálculo de Hoare, se va a derivar un programa que dado un vector A(1..n) que sólo contiene números no negativos, devuelve un 0 en la variable entera *c* si A(1..n) contiene sólo ceros y devuelve algo distinto de 0 si en A(1..n) no todos son ceros. El programa será correcto con respecto al invariante, a la expresión cota E y a la especificación dada (precondición φ y postcondición ψ).

$\{\varphi\} \equiv \{n \geq 1 \wedge \text{noneg}(A(1..n))\}$ ¿Inicializaciones? {INV} **while** {INV} {E} ¿B? **loop**   ¿Instrucciones? **end loop;** $\{\psi\} \equiv \{(c = 0) \leftrightarrow \forall k(1 \leq k \leq n \rightarrow A(k) = 0)\}$
$\{\text{INV}\} \equiv \{\text{noneg}(A(1..n)) \wedge (0 \leq i \leq n) \wedge c = \sum_{k=1}^{i} A(k)\}$ $E = n - i$ $\text{noneg}(A(1..n)) \equiv \{\forall k(1 \leq k \leq n \rightarrow A(k) \geq 0)\}$

Antes de empezar a derivar las instrucciones tenemos que fijarnos en la expresión cota E. Como tiene la forma "límite superior – variable índice", sabemos que el vector se ha de recorrer de izquierda a derecha.

- **Inicializaciones**

  Para obtener las inicializaciones se tiene en cuenta el primer punto de la regla del while: la precondición φ ha de implicar el invariante. Si es así, no hace falta inicializar nada, y en caso contrario sí hace falta alguna inicialización. Tras incluir una nueva inicialización se calcula su aserción previa utilizando el Axioma de la Asignación y se comprobará si la nueva fórmula es implicada por φ. El proceso de añadir nuevas inicializaciones continuará hasta obtener una fórmula que sí es implicada por φ.

¿$\varphi \rightarrow$ INV?

$$\underbrace{n \geq 1}_{\alpha} \wedge \underbrace{noneg(A(1..n))}_{\beta}$$

$$\downarrow ?$$

$$\underbrace{noneg(A(1..n))}_{\text{sí por } \beta} \wedge \underbrace{\underbrace{(0 \leq i \leq n)}_{\delta_1} \wedge \underbrace{c = \sum_{k=1}^{i} A(k)}_{\delta_2}}_{\delta}$$

No se cumple

La implicación no se cumple porque en la primera parte de la implicación (φ) no tenemos ninguna información sobre i y c para asegurar que se cumple δ. Como el objetivo es que la implicación se cumpla, tenemos que inicializar i y c con valores que hagan que la implicación sea cierta. Como δ_2 depende de i y c, hay que empezar fijándose en δ_1, que sólo depende de i.

Como sabemos que el vector se ha de recorrer de izquierda a derecha, para que se cumpla δ_1 basta con asignar 0 a la variable i. Por tanto mirando en δ_1 asignamos a i el valor más pequeño que puede tomar según el invariante.

Ahora calculamos la aserción $\{\varphi_1\}$ utilizando el axioma de la asignación

$\{\varphi\}$ ¿$\varphi \rightarrow \varphi_1$?

(AA) $\{\varphi_1\}$ i : = 0; $\{INV\}$

$$\{\varphi_1\} \equiv \{def(0) \wedge (INV)_i^0\} \equiv$$

$$\equiv \{true \wedge noneg(A(1..n)) \wedge (0 \leq 0 \leq n) \wedge c = \sum_{k=1}^{0} A(k)\} \equiv^{\text{simplificación}}$$

$$\equiv \{noneg(A(1..n)) \wedge (0 \leq n) \wedge c = \sum_{k=1}^{0} A(k)\} \equiv^{\text{simplificación}}$$

$$\equiv \{noneg(A(1..n)) \wedge (0 \leq n) \wedge c = 0\}$$

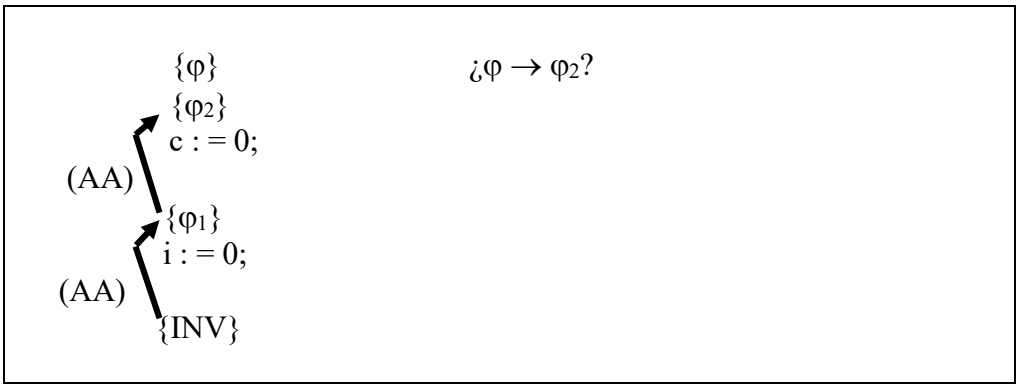
- ¿$\varphi \rightarrow \varphi_1$?

$$\underbrace{n \geq 1}_{\alpha} \wedge \underbrace{noneg(A(1..n))}_{\beta}$$

$$\downarrow ?$$

$$\underbrace{noneg(A(1..n))}_{\text{sí por } \beta} \wedge \underbrace{(0 \leq n)}_{\text{sí por } \alpha} \wedge \underbrace{c = 0}_{\text{No se cumple}}$$

El que se cumplan α y β no implica que se cumpla $c = 0$, es decir, lo que nos falta es que la variable c valga 0, y esa es la inicialización que nos queda.



$$\{\varphi_2\} \equiv \{def(0) \wedge (\varphi_1)_c^0\} \equiv$$
$$\equiv \{true \wedge noneg(A(1..n)) \wedge (0 \leq n) \wedge 0 = 0\} \equiv^{\text{simplificación}}$$

$$\equiv \{true \wedge noneg(A(1..n)) \wedge (0 \leq n) \wedge true\} \equiv^{\text{simplificación}}$$
$$\equiv \{noneg(A(1..n)) \wedge (0 \leq n)\}$$

- ¿$\varphi \rightarrow \varphi_2$?

$$\underbrace{n \geq 1}_{\alpha} \wedge \underbrace{noneg(A(1..n))}_{\beta}$$

$$\downarrow ?$$

$$\underbrace{noneg(A(1..n))}_{\text{sí por } \beta} \wedge \underbrace{(0 \leq n)}_{\text{sí por } \alpha}$$

La implicación $\varphi \rightarrow \varphi_2$ sí se cumple y, por tanto, ya hemos terminado con las inicializaciones.

- **Condición del while (B)**

  Primero se calcula **¬B** respondiendo a la pregunta **¿Cuándo salimos del while?**

  Sabiendo que estamos recorriendo la tabla de derecha a izquierda y que según el invariante el límite superior de i es n, cuando i valga n hay que salir del while:

  $\neg B \equiv i = n$
  Y por tanto, $B \equiv i \neq n$

  Ahora hay que comprobar que realmente la condición B es correcta y para ello se tendrán en cuenta los puntos II, IV y V de la regla del while.

  II. ¿INV → def(B)?
  ¿INV → def(i ≠ n)?
  ¿INV → true? Sí, porque en la segunda parte de la implicación tenemos true.

  IV. ¿(INV ∧ ¬B) → ψ?

$$\underbrace{\text{noneg}(A(1..n))}_{\alpha} \wedge \underbrace{(0 \le i \le n)}_{\beta} \wedge \underbrace{c = \sum_{k=1}^{i} A(k)}_{\delta} \wedge \underbrace{(i = n)}_{\gamma}$$

$$\downarrow ?$$

$$\underbrace{(c = 0) \leftrightarrow \forall k(1 \le k \le n \rightarrow A(k) = 0)}_{\text{por } \alpha, \delta \text{ y } \gamma}$$

  Como por δ y γ la variable c es igual a la suma de los elementos de A(1..n) y como por α solo hay ceros y unos, c será igual a 0 si y solamente si todos los elementos de A(1..n) son ceros.

  V. ¿INV ∧ B → E > 0?

$$\text{noneg}(A(1..n)) \wedge \underbrace{(0 \le i \le n)}_{\alpha} \wedge c = \sum_{k=1}^{i} A(k) \wedge \underbrace{(i \neq n)}_{\beta}$$

$$\downarrow ?$$

$$\underbrace{n - i > 0}_{\delta}$$

  δ se cumple por α y β.

- **Instrucciones**

Tenemos que coger los dos programas de los puntos III y VI de la regla del while

Prog 1
{INV ∧ B} Instrucciones {INV}

Prog 2
{INV ∧ B ∧ E = v} Instrucciones {E < v}

- Nos tenemos que preguntar
  - ✓ ¿(INV ∧ B) → INV?
  - ✓ ¿(INV ∧ B ∧ E = v) → E < v?

La implicación (INV ∧ B) → INV se cumplirá siempre porque si INV y B son ciertos entonces INV será cierto.

En cambio la implicación (INV ∧ B ∧ E = v) → E < v no se cumplirá porque si E es igual a v entonces E no puede ser menor que v.

El que la segunda implicación no se cumpla quiere decir que tenemos que añadir una instrucción para conseguir que se cumpla E < v. Como sabemos que estamos recorriendo el vector de izquierda a derecha, para que el valor de E decrezca, tendremos que asignar i + 1 a i y calcularemos la aserción φ_3' utilizando el axioma de la asignación

	Prog 2
AA	{INV ∧ B ∧ E = v} ≡ ≡ {noneg(A(1..n)) ∧ (0 ≤ i ≤ n) ∧ c = $\sum_{k=1}^{i} A(k)$ ∧ (i ≠ n + 1) ∧ n – i = v} {φ_3'} ≡ {def(i + 1) ∧ $(E < v)_i^{i+1}$} ≡ {true ∧ n – (i + 1) < v} ≡ {n – i – 1 < v} **i : = i + 1;** {n – i < v}

Ahora nos preguntamos ¿(INV ∧ B ∧ E = v) → φ_3'?

$$\text{noneg}(A(1..n)) \wedge (0 \le i \le n) \wedge c = \sum_{k=1}^{i} A(k) \wedge (i \ne n) \underbrace{\wedge\ n - i = v}_{\alpha}$$

$$\underbrace{\downarrow ?\quad n - i - 1 < v}_{\beta}$$

La implicación se cumple porque β es cierto por α. Si n – i = v, entonces n – i – 1 = v – 1 y v – 1 es menor que v.

Esto quiere decir que Prog 2 ahora es correcto pero como Prog 1 y Prog 2 han de tener las mismas instrucciones tenemos que añadir la asignación i : = i + 1; a Prog 1, luego calcularemos φ_3 utilizando el axioma de la asignación y comprobaremos si se cumple $(INV \wedge B) \rightarrow \varphi_3$.

	Prog 1
AA	$\{INV \wedge B\} \equiv \{noneg(A(1..n)) \wedge (0 \le i \le n) \wedge c = \sum_{k=1}^{i} A(k) \wedge (i \ne n)\}$ $\{\varphi_3\} \equiv \{def(i+1) \wedge (INV)_i^{i+1}\}$ **i : = i + 1;** $\{INV\} \equiv \{noneg(A(1..n)) \wedge (0 \le i \le n) \wedge c = \sum_{k=1}^{i} A(k)\}$

$$\{\varphi_3\} \equiv \{def(i+1) \wedge (INV)_i^{i+1}\} \equiv$$

$$\equiv \{true \wedge noneg(A(1..n)) \wedge (0 \le i+1 \le n) \wedge c = \sum_{k=1}^{i+1} A(k)\} \equiv \text{simplificación}$$

$$\equiv \{noneg(A(1..n)) \wedge (0 \le i+1 \le n) \wedge c = \sum_{k=1}^{i+1} A(k)\} \equiv \text{simplificación}$$

$$\equiv \{noneg(A(1..n)) \wedge (-1 \le i \le n-1) \wedge c = \sum_{k=1}^{i+1} A(k)\}$$

Ahora nos preguntamos ¿$(INV \wedge B) \rightarrow \varphi_3$?

$$\underbrace{noneg(A(1..n))}_{\alpha} \wedge \underbrace{(0 \le i \le n)}_{\beta} \wedge \underbrace{c = \sum_{k=1}^{i} A(k)}_{\delta} \wedge \underbrace{(i \ne n)}_{\gamma}$$

$$\downarrow ?$$

$$\underbrace{noneg(A(1..n))}_{\text{por } \alpha} \wedge (\underbrace{-1 \le i \le n-1}_{\text{por } \beta \text{ y } \gamma}) \wedge \underbrace{c = \sum_{k=1}^{i+1} A(k)}_{\pi \text{ (No se cumple)}}$$

Por β sabemos que en c tenemos la suma de A(1..i) pero en π se nos pregunta si en c tenemos la suma de A(1..i + 1) y como la respuesta es que no, la implicación $(INV \wedge B) \rightarrow \varphi_3$ no se cumple. Como el objetivo en este momento es que φ_3 se cumpla, es decir, el objetivo es que en c se tenga la suma de los elementos de A(1..i + 1), habrá que sumar A(i + 1) a c.

$$c := c + A(i + 1);$$

luego calcularemos φ_4 y comprobaremos si se cumple $(INV \wedge B) \rightarrow \varphi_4$

	Prog 1
AA	$\{INV \wedge B\} \equiv \{noneg(A(1..n)) \wedge (0 \leq i \leq n) \wedge c = \sum_{k=1}^{i} A(k) \wedge (i \neq n)\}$ $\{\varphi_4\} \equiv \{def(c + A(i + 1)) \wedge (\varphi_3)_c^{\,c + A(i + 1)}\}$ **c : = c + A(i + 1);** $\{\varphi_3\} \equiv \{noneg(A(1..n)) \wedge (-1 \leq i \leq n - 1) \wedge c = \sum_{k=1}^{i+1} A(k)\}$ i : = i + 1; $\{INV\} \equiv \{noneg(A(1..n)) \wedge (0 \leq i \leq n) \wedge c = \sum_{k=1}^{i} A(k)\}$

$$\{\varphi_4\} \equiv \{def(c + A(i + 1)) \wedge (\varphi_3)_c^{\,c + A(i + 1)}\} \equiv$$
$$\equiv \{(1 \leq i + 1 \leq n) \wedge noneg(A(1..n)) \wedge (-1 \leq i \leq n - 1) \wedge$$
$$c + A(i + 1) = \sum_{k=1}^{i+1} A(k)\} \equiv \text{simplificación}$$
$$\equiv \{(0 \leq i \leq n - 1) \wedge noneg(A(1..n)) \wedge (-1 \leq i \leq n - 1) \wedge$$
$$c + A(i + 1) = \sum_{k=1}^{i+1} A(k)\} \equiv \text{simplificación}$$
$$\equiv \{(0 \leq i \leq n - 1) \wedge noneg(A(1..n)) \wedge c + A(i + 1) = \sum_{k=1}^{i+1} A(k)\}$$

Ahora nos preguntamos ¿$(INV \wedge B) \rightarrow \varphi_4$?

$$\underbrace{noneg(A(1..n))}_{\alpha} \wedge \underbrace{(0 \leq i \leq n)}_{\beta} \wedge \underbrace{c = \sum_{k=1}^{i} A(k)}_{\delta} \wedge \underbrace{(i \neq n)}_{\gamma}$$

$$\downarrow ?$$

$$\underbrace{(0 \leq i \leq n - 1)}_{\text{por } \beta \text{ y } \gamma} \wedge \underbrace{noneg(A(1..n))}_{\text{por } \alpha} \wedge \underbrace{c + A(i + 1) = \sum_{k=1}^{i+1} A(k)}_{\text{por } \delta}$$

Prog 1 ahora es correcto. Pero como Prog 1 y Prog 2 han de tener las mismas instrucciones tenemos que añadir la asignación **c : = c + A(i + 1);** a Prog 2, luego hay que calcular φ_4' utilizando el axioma de la asignación y comprobar si se cumple $(INV \wedge B \wedge E = v) \rightarrow \varphi_4'$.

	Prog 2
AA	$\{INV \wedge B \wedge E = v\} \equiv$ $\equiv \{noneg(A(1..n)) \wedge (0 \le i \le n) \wedge c = \sum_{k=1}^{i} A(k) \wedge (i \neq n) \wedge n - i = v\}$ $\{\varphi_4'\} \equiv \{def(s + A(i+1)) \wedge (\varphi_3')_s^{\,s + A(i+1)}\}$ **c : = c + A(i + 1);** $\{\varphi_3'\} \equiv \{n - i - 1 < v\}$ **i : = i – 1;** $\{E < v\} \equiv \{n - i < v\}$

$$\{\varphi_4'\} \equiv \{def(c + A(i+1)) \wedge (\varphi_3')_s^{\,s + A(i+1)}\} \equiv^{\text{simplificación}}$$
$$\equiv \{(1 \le i + 1 \le n) \wedge n - i - 1 < v\} \equiv^{\text{simplificación}}$$
$$\equiv \{(0 \le i \le n - 1) \wedge n - i - 1 < v\}$$

Ahora nos preguntamos ¿$(INV \wedge B \wedge E = v) \rightarrow \varphi_4'$?

$$\underbrace{noneg(A(1..n)) \wedge (0 \le i \le n)}_{\alpha} \wedge c = \sum_{k=1}^{i} A(k) \underbrace{\wedge (i \neq n)}_{\beta} \underbrace{\wedge n - i = v}_{\delta}$$

$$\downarrow ?$$

$$\underbrace{(0 \le i \le n - 1)}_{\text{por } \alpha \text{ y } \beta} \wedge \underbrace{n - i - 1 < v}_{\text{por } \delta \text{ ya que por } \delta\ n - i - 1 = v - 1 \text{ y } v - 1 \text{ es menor que } v}$$

Como se han cumplido las dos implicaciones, $(INV \wedge B) \rightarrow \varphi_4$ y $(INV \wedge B \wedge E = v) \rightarrow \varphi_4'$ ya hemos terminado de derivar el programa. El programa es el siguiente:

```
{φ}
{φ2}
c : = 0;
{φ1}
i : = 1;

while {INV} i ≠ n loop
      {φ4} {φ4'}
      c : = c + A(i + 1);
      {φ3} {φ3'}
      i : = i + 1;
end loop;

{ψ}
E
```

Sabemos que el programa es **correcto** porque se ha construido siguiendo la regla del while y además el propio método calcula la **documentación** del programa, que vendrá dada por $\{\varphi_1\}$, $\{\varphi_2\}$, $\{\varphi_3\}$, $\{\varphi_3'\}$, $\{\varphi_4\}$ y $\{\varphi_4'\}$.

4.2.4. DEVOLVER 0 EN C SI Y SOLAMENTE SI A(1..N) CONTIENE AL MENOS UN 0 (EJERCICIO 11)

Teniendo en cuenta la regla del while (RWH) y el axioma de la asignación (AA) del Cálculo de Hoare, se va a derivar un programa que dado un vector A(1..n) de enteros, devuelve un 0 en la variable entera *c* si A(1..n) contiene sólo ceros y devuelve algo distinto de 0 si en A(1..n) no todos son ceros. El programa será correcto con respecto al invariante, a la expresión cota E y a la especificación dada (precondición φ y postcondición ψ).

$\{\varphi\} \equiv \{n \geq 1\}$ ¿Inicializaciones? {INV} **while** {INV} {E} ¿B? **loop**   ¿Instrucciones? **end loop;** $\{\psi\} \equiv \{(c = 0) \leftrightarrow \exists k(1 \leq k \leq n \wedge A(k) = 0)\}$
$\{INV\} \equiv \{(1 \leq i \leq n) \wedge c = \prod_{k=1}^{i} A(k)\}$ $E = n - i$

Antes de empezar a derivar las instrucciones tenemos que fijarnos en la expresión cota E. Como tiene la forma "límite superior – variable índice", sabemos que el vector se ha de recorrer de izquierda a derecha.

- **Inicializaciones**

  Para obtener las inicializaciones se tiene en cuenta el primer punto de la regla del while: la precondición φ ha de implicar el invariante. Si es así, no hace falta inicializar nada, y en caso contrario sí hace falta alguna inicialización. Tras incluir una nueva inicialización se calcula su aserción previa utilizando el Axioma de la Asignación y se comprobará si la nueva fórmula es implicada por φ. El proceso de añadir nuevas inicializaciones continuará hasta obtener una fórmula que sí es implicada por φ.

¿$\varphi \rightarrow$ INV?

$$\underbrace{n \geq 1}_{\alpha} \quad \downarrow ? \quad \underbrace{\underbrace{(1 \leq i \leq n)}_{\delta_1} \wedge \underbrace{c = \prod_{k=1}^{i} A(k)}_{\delta_2}}_{\delta}$$

No se cumple

La implicación no se cumple porque en la primera parte de la implicación (α) no tenemos ninguna información sobre i y c para asegurar que se cumple δ. Como el objetivo es que la implicación se cumpla, tenemos que inicializar i y c con valores que hagan que la implicación sea cierta. Como δ_2 depende de i y c, hay que empezar fijándose en δ_1, que sólo depende de i.

Como sabemos que el vector se ha de recorrer de izquierda a derecha, para que se cumpla δ_1 basta con asignar 1 a la variable i. Por tanto mirando en δ_1 asignamos a i el valor más pequeño que puede tomar según el invariante.

Ahora calculamos la aserción $\{\varphi_1\}$ utilizando el axioma de la asignación

$\{\varphi\}$ ¿$\varphi \rightarrow \varphi_1$?

$\{\varphi_1\}$

i : = 1;

(AA)

$\{INV\}$

$$\{\varphi_1\} \equiv \{def(1) \wedge (INV)_i^1\} \equiv$$

$$\equiv \{true \wedge (1 \leq 1 \leq n) \wedge c = \prod_{k=1}^{1} A(k)\} \equiv^{\text{simplificación}}$$

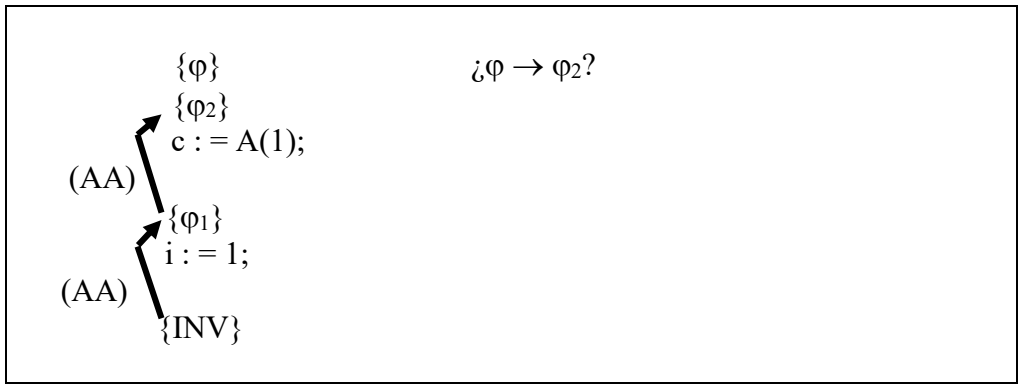
$$\equiv \{(1 \leq n) \wedge c = \prod_{k=1}^{1} A(k)\} \equiv^{\text{simplificación}}$$

$$\equiv \{(1 \leq n) \wedge c = A(1)\}$$

- ¿$\varphi \rightarrow \varphi_1$?

$$\underbrace{n \geq 1}_{\alpha} \downarrow ? \; \underbrace{(1 \leq n)}_{\text{sí por } \alpha} \wedge \underbrace{c = A(1)}_{\text{No se cumple}}$$

El que se cumpla α no implica que se cumpla $c = A(1)$, es decir, lo que nos falta es que la variable c valga A(1), y ésa es la inicialización que nos queda.



$$\begin{aligned}\{\varphi_2\} &\equiv \{def(A(1)) \wedge (\varphi_1)_c^{A(1)}\} \equiv \\ &\equiv \{(1 \leq 1 \leq n) \wedge (1 \leq n) \wedge A(1) = A(1)\} \equiv \\ &\equiv \{(1 \leq n) \wedge (1 \leq n) \wedge true\} \equiv \\ &\equiv \{(1 \leq n)\}\end{aligned}$$

- ¿$\varphi \rightarrow \varphi_2$?

$$\underbrace{n \geq 1}_{\alpha} \downarrow ? \; \underbrace{(1 \leq n)}_{\text{sí por } \alpha}$$

La implicación $\varphi \rightarrow \varphi_2$ sí se cumple y, por tanto, ya hemos terminado con las inicializaciones.

- **<u>Condición del while (B)</u>**

  Primero se calcula **¬B** respondiendo a la pregunta **¿Cuándo salimos del while?**

  Sabiendo que estamos recorriendo la tabla de izquierda a derecha y que según el invariante el límite superior de i es n, cuando i valga n hay que salir del while:

  $\neg B \equiv i = n$
  Y por tanto, $B \equiv i \neq n$

  Ahora hay que comprobar que realmente la condición B es correcta y para ello se tendrán en cuenta los puntos II, IV y V de la regla del while.

  II. ¿INV → def(B)?
  ¿INV → def(i ≠ n)?
  ¿INV → true? Sí porque en la segunda parte de la implicación tenemos true.

  IV. ¿(INV ∧ ¬B) → ψ?

$$\underbrace{(1 \le i \le n)}_{\beta} \wedge \underbrace{c = \prod_{k=1}^{i} A(k)}_{\delta} \wedge \underbrace{(i = n)}_{\gamma}$$

$$\downarrow ?$$

$$\underbrace{(c = 0) \leftrightarrow \exists k(1 \le k \le n \wedge A(k) = 0)}_{\text{por } \delta \text{ y } \gamma}$$

  Como por δ y γ la variable c es igual al producto de los elementos de A(1..n) y c será igual a 0 si y solamente existe al menos una posición de A(1..n) que contenga un cero.

  V. ¿INV ∧ B → E > 0?

$$\underbrace{(1 \le i \le n)}_{\alpha} \wedge c = \prod_{k=1}^{i} A(k) \wedge \underbrace{(i \neq n)}_{\beta}$$

$$\downarrow ?$$

$$\underbrace{n - i > 0}_{\delta}$$

  δ se cumple por α y β.

- **Instrucciones**

Tenemos que coger los dos programas de los puntos III y VI de la regla del while

Prog 1
{INV ∧ B} Instrucciones {INV}

Prog 2
{INV ∧ B ∧ E = v} Instrucciones {E < v}

- Nos tenemos que preguntar
  - ✓ ¿(INV ∧ B) → INV?
  - ✓ ¿(INV ∧ B ∧ E = v) → E < v?

La implicación (INV ∧ B) → INV se cumplirá siempre porque si INV y B son ciertos entonces INV será cierto.

En cambio la implicación (INV ∧ B ∧ E = v) → E < v no se cumplirá porque si E es igual a v entonces E no puede ser menor que v.

El que la segunda implicación no se cumpla quiere decir que tenemos que añadir una instrucción para conseguir que se cumpla E < v. Como sabemos que estamos recorriendo el vector de izquierda a derecha, para que el valor de E decrezca, tendremos que asignar i + 1 a i y calcularemos la aserción φ_3' utilizando el axioma de la asignación

	Prog 2
AA	{INV ∧ B ∧ E = v} ≡ $\equiv \{(1 \le i \le n) \wedge c = \prod_{k=1}^{i} A(k) \wedge (i \ne n) \wedge n - i = v\}$ $\{\varphi_3'\} \equiv \{def(i+1) \wedge (E < v)_i^{i+1}\} \equiv \{true \wedge n - (i+1) < v\} \equiv \{n - i - 1 < v\}$ **i : = i + 1;** {n – i < v}

Ahora nos preguntamos ¿(INV ∧ B ∧ E = v) → φ_3'?

$$(1 \le i \le n) \wedge c = \prod_{k=1}^{i} A(k) \wedge (i \ne n) \wedge \underbrace{n - i = v}_{\alpha}$$

$$\downarrow ?$$

$$\underbrace{n - i - 1 < v}_{\beta}$$

La implicación se cumple porque β es cierto por α. Si n – i = v, entonces n – i – 1 = v – 1 y v – 1 es menor que v.

Esto quiere decir que Prog 2 ahora es correcto pero como Prog 1 y Prog 2 han de tener las mismas instrucciones tenemos que añadir la asignación i : = i + 1; a Prog 1, luego calcularemos φ_3 utilizando el axioma de la asignación y comprobaremos si se cumple $(INV \wedge B) \rightarrow \varphi_3$.

	Prog 1
AA	$\{INV \wedge B\} \equiv \{(1 \leq i \leq n) \wedge c = \prod_{k=1}^{i} A(k) \wedge (i \neq n)\}$ $\{\varphi_3\} \equiv \{def(i+1) \wedge (INV)_i^{i+1}\}$ **i : = i + 1;** $\{INV\} \equiv \{(1 \leq i \leq n) \wedge c = \prod_{k=1}^{i} A(k)\}$

$$\{\varphi_1\} \equiv \{def(i+1) \wedge (INV)_i^{i+1}\} \equiv$$

$$\equiv \{true \wedge (1 \leq i+1 \leq n) \wedge c = \prod_{k=1}^{i+1} A(k)\} \equiv^{\text{simplificación}}$$

$$\equiv \{(1 \leq i+1 \leq n) \wedge c = \prod_{k=1}^{i+1} A(k)\} \equiv^{\text{simplificación}}$$

$$\equiv \{(0 \leq i \leq n-1) \wedge c = \prod_{k=1}^{i+1} A(k)\}$$

Ahora nos preguntamos ¿$(INV \wedge B) \rightarrow \varphi_3$?

$$\underbrace{(1 \leq i \leq n)}_{\beta} \wedge \underbrace{c = \prod_{k=1}^{i} A(k)}_{\delta} \wedge \underbrace{(i \neq n)}_{\gamma}$$

$$\downarrow ?$$

$$\underbrace{(0 \leq i \leq n-1)}_{\text{por } \beta \text{ y } \gamma} \wedge \underbrace{c = \prod_{k=1}^{i+1} A(k)}_{\pi \text{ (No se cumple)}}$$

Por β sabemos que en c tenemos el producto de A(1..i) pero en π se nos pregunta si en c tenemos el producto de A(1..i + 1) y como la respuesta es que no, la implicación $(INV \wedge B) \rightarrow \varphi_3$ no se cumple. Como el objetivo en este momento es que φ_3 se cumpla, es decir, el objetivo es que en c se tenga el producto de los elementos de A(1..i + 1), habrá que multiplicar A(i + 1) y c.

$$c := c * A(i + 1);$$

luego calcularemos φ_4 y comprobaremos si se cumple $(INV \wedge B) \rightarrow \varphi_4$

	Prog 1
AA	$\{INV \wedge B\} \equiv \{(1 \leq i \leq n) \wedge c = \prod_{k=1}^{i} A(k) \wedge (i \neq n)\}$ $\{\varphi_4\} \equiv \{def(c * A(i + 1)) \wedge (\varphi_3)_c{}^{c * A(i + 1)}\}$ **c : = c * A(i + 1)** $\{\varphi_3\} \equiv \{(0 \leq i \leq n - 1) \wedge c = \prod_{k=1}^{i+1} A(k)\}$ i : = i + 1; $\{INV\} \equiv \{(1 \leq i \leq n) \wedge c = \prod_{k=1}^{i} A(k)\}$

$$\{\varphi_4\} \equiv \{def(c * A(i + 1)) \wedge (\varphi_3)_c{}^{c * A(i + 1)}\} \equiv$$
$$\equiv \{(1 \leq i + 1 \leq n) \wedge (0 \leq i \leq n - 1) \wedge$$
$$c * A(i + 1) = \prod_{k=1}^{i+1} A(k)\} \equiv \text{simplificación}$$
$$\equiv \{(0 \leq i \leq n - 1) \wedge (0 \leq i \leq n - 1) \wedge$$
$$c * A(i + 1) = \prod_{k=1}^{i+1} A(k)\} \equiv \text{simplificación}$$
$$\equiv \{(0 \leq i \leq n - 1) \wedge c * A(i + 1) = \prod_{k=1}^{i+1} A(k)\}$$

Ahora nos preguntamos ¿$(INV \wedge B) \rightarrow \varphi_4$?

$$\underbrace{(1 \leq i \leq n)}_{\beta} \wedge \underbrace{c = \prod_{k=1}^{i} A(k)}_{\delta} \wedge \underbrace{(i \neq n)}_{\gamma}$$

$$\downarrow ?$$

$$\underbrace{(0 \leq i \leq n - 1)}_{\text{por } \beta \text{ y } \gamma} \wedge \underbrace{c * A(i + 1) = \prod_{k=1}^{i+1} A(k)}_{\text{por } \delta}$$

Prog 1 ahora es correcto. Pero como Prog 1 y Prog 2 han de tener las mismas instrucciones tenemos que añadir la asignación **c : = c * A(i + 1);** a Prog 2, luego hay que calcular φ_4' utilizando el axioma de la asignación y comprobar si se cumple $(INV \wedge B \wedge E = v) \rightarrow \varphi_4'$.

	Prog 2
AA	$\{INV \wedge B \wedge E = v\} \equiv$ $\equiv \{(1 \le i \le n) \wedge c = \prod_{k=1}^{i} A(k) \wedge (i \ne n) \wedge n - i = v\}$ $\{\varphi_4'\} \equiv \{def(s * A(i+1)) \wedge (\varphi_1')_s^{\,s * A(i+1)}\}$ **c : = c * A(i + 1);** $\{\varphi_3'\} \equiv \{n - i - 1 < v\}$ **i : = i + 1;** $\{E < v\} \equiv \{n - i < v\}$

$$\{\varphi_4'\} \equiv \{def(c * A(i+1)) \wedge (\varphi_3')_s^{\,s * A(i+1)}\} \equiv$$
$$\equiv \{(1 \le i + 1 \le n) \wedge n - i - 1 < v\} \equiv^{\text{simplificación}}$$
$$\equiv \{(0 \le i \le n - 1) \wedge n - i - 1 < v\}$$

Ahora nos preguntamos ¿$(INV \wedge B \wedge E = v) \rightarrow \varphi_4'$?

$$\underbrace{(1 \le i \le n)}_{\alpha} \wedge c = \prod_{k=1}^{i} A(k) \wedge \underbrace{(i \ne n)}_{\beta} \wedge \underbrace{n - i = v}_{\delta}$$

$$\downarrow ?$$

$$\underbrace{(0 \le i \le n - 1)}_{\text{por } \alpha \text{ y } \beta} \wedge \underbrace{n - i - 1 < v}_{\text{por } \delta \text{ ya que por } \delta\ n - i - 1 = v - 1 \text{ y } v - 1 \text{ es menor que } v}$$

Como se han cumplido las dos implicaciones, $(INV \wedge B) \rightarrow \varphi_4$ y $(INV \wedge B \wedge E = v) \rightarrow \varphi_4'$ ya hemos terminado de derivar el programa. El programa es el siguiente:

```
{φ}
{φ2}
c : = A(1);
{φ1}
i : = 1;

while {INV} i ≠ n loop
      {φ4} {φ4'}
      c : = c * A(i + 1);
      {φ3} {φ3'}
      i : = i + 1;
end loop;

{ψ}
E
```

Sabemos que el programa es **correcto** porque se ha construido siguiendo la regla del while y además el propio método calcula la **documentación** del programa, que vendrá dada por $\{\varphi_1\}$, $\{\varphi_2\}$, $\{\varphi_3\}$, $\{\varphi_3'\}$, $\{\varphi_4\}$ y $\{\varphi_4'\}$.

4.2.5. DECIDIR SI DOS VECTORES SON IGUALES (EJERCICIO 12)

Teniendo en cuenta la regla del while (RWH) y el axioma de la asignación (AA) del Cálculo de Hoare, se va a derivar un programa que dados dos vectores A(1..n) y B(1..n), decide en la variable booleana *c* si A(1..n) y B(1..n) son iguales. El programa derivado deberá ser eficiente en el sentido de que al recorrer los vectores, si en un momento se detecta que la respuesta va a ser negativa el while deberá terminar sin analizar las posiciones restantes. El programa derivado ha de ser correcto con respecto a φ, ψ, INV y E.

$\{\varphi\} \equiv \{n \geq 1\}$ ¿Inicializaciones? {INV} **while** {INV} {E} B **loop**   ¿Instrucciones? **end loop;** $\{\psi\} \equiv \{c \leftrightarrow \forall k(1 \leq k \leq n \rightarrow A(k) = B(k))\}$
$\{INV\} \equiv \{(1 \leq i \leq n + 1) \wedge (c \leftrightarrow \forall k(1 \leq k \leq i - 1 \rightarrow A(k) = B(k)))\}$ $E = n + 1 - i$

Antes de empezar a derivar las instrucciones tenemos que fijarnos en la expresión cota E. Como tiene la forma "límite superior – variable índice", sabemos que el vector se ha de recorrer de izquierda a derecha.

- **Inicializaciones**
  Para obtener las inicializaciones se tiene en cuenta el primer punto de la regla del while: φ ha de implicar el invariante. Si es así, no hace falta inicializar nada, y en caso contrario sí hace falta alguna inicialización. Tras incluir una nueva inicialización se calcula su aserción previa utilizando el Axioma de la Asignación y se comprobará si la nueva fórmula es implicada por φ. El proceso de añadir nuevas inicializaciones continuará hasta obtener una fórmula que sí es implicada por φ.

  ¿$\varphi \rightarrow INV$?

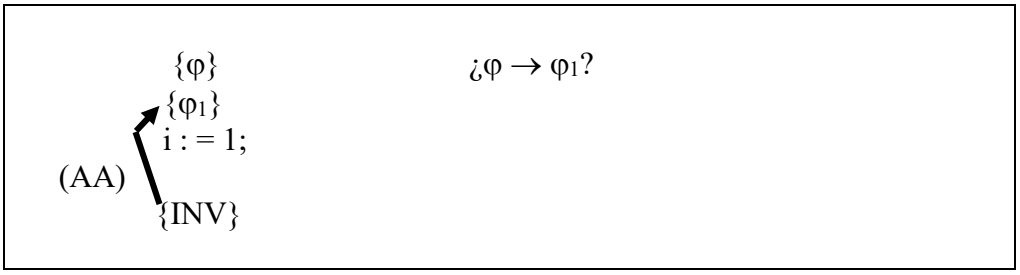
$$\underbrace{n \geq 1}_{\delta} \quad \downarrow ? \quad \underbrace{(1 \leq i \leq n + 1)}_{\alpha} \wedge \underbrace{(c \leftrightarrow \forall k(1 \leq k \leq i - 1 \rightarrow A(k) = B(k)))}_{\beta}$$

  La implicación no se cumple porque en la primera parte de la implicación (δ) no tenemos ninguna información sobre i y c para asegurar que se cumple la segunda parte de la implicación (α y β). Como el objetivo es que la implicación se cumpla,

tenemos que inicializar i y c con valores que hagan que la implicación sea cierta. Como β depende de i y c, hay que empezar fijándose en α, que solo depende de i.

Como sabemos que el vector se ha de recorrer de izquierda a derecha, para que se cumpla α basta con asignar 1 a la variable i. Por tanto mirando en α asignamos a i el valor más pequeño que puede tomar según el invariante.

Ahora calculamos la aserción $\{\varphi_1\}$ utilizando el axioma de la asignación



$\{\varphi_1\} \equiv \{def(1) \wedge (INV)_i^1\} \equiv$

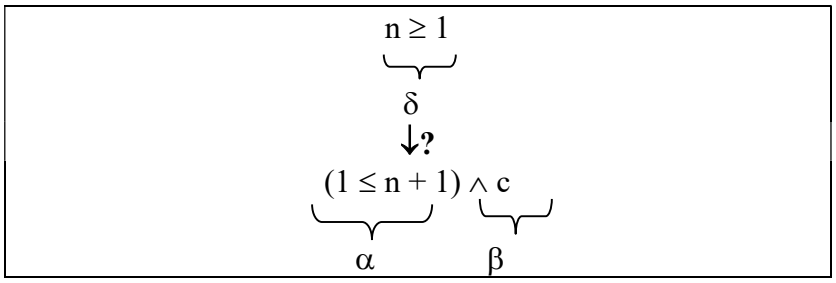
$\equiv \{true \wedge (1 \leq 1 \leq n + 1) \wedge (c \leftrightarrow \forall k(1 \leq k \leq 1 - 1 \rightarrow A(k) = B(k)))\} \equiv$ simplificación

$\equiv \{(1 \leq n + 1) \wedge (c \leftrightarrow \forall k(1 \leq k \leq 0 \rightarrow A(k) = B(k)))\} \equiv$ simplificación

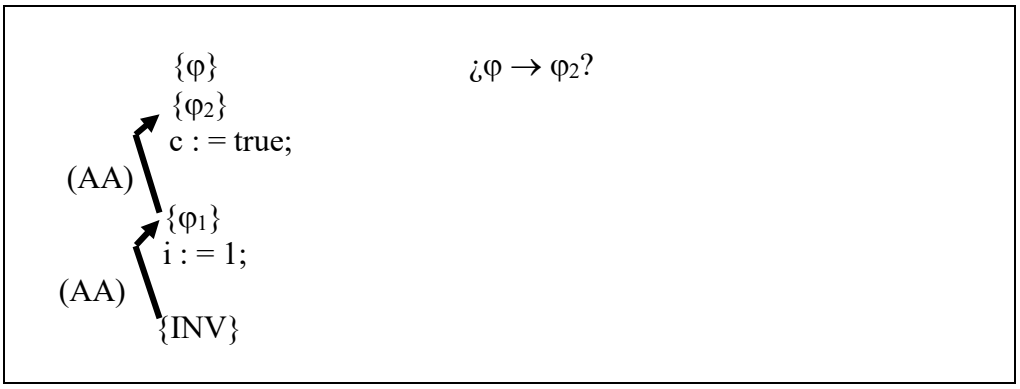
$\equiv \{(1 \leq n + 1) \wedge c \leftrightarrow true\} \equiv$ simplificación

$\equiv \{(1 \leq n + 1) \wedge c\}$

- ¿$\varphi \rightarrow \varphi_1$?



El que se cumpla $n \geq 1$ sí implica $1 \leq n + 1$ pero no implica que se cumpla c sea true, es decir, lo que nos falta es que la variable c valga true, y esa es la inicialización que nos queda.



$\{\varphi_2\} \equiv \{def(true) \wedge (\varphi_1)_c^{true}\} \equiv$
$\equiv \{true \wedge (1 \leq n + 1) \wedge true\} \equiv$ simplificación
$\equiv (1 \leq n + 1)$

- ¿$\varphi \rightarrow \varphi_2$?

$$n \geq 1$$
$$\downarrow ?$$
$$(1 \leq n + 1)$$

La implicación $\varphi \rightarrow \varphi_2$ sí se cumple y por tanto ya hemos terminado con las inicializaciones.

- **Condición del while (B)**

Primero se calcula **¬B** respondiendo a la pregunta **¿Cuándo salimos del while?**

Sabiendo que estamos recorriendo la tabla de derecha a izquierda y que según el invariante el límite superior de i es n + 1, cuando i valga n + 1 hay que salir del while pero además el programa ha de ser eficiente en el sentido de que si nos damos cuenta de que los vectores no son iguales, se ha de parar sin comparar las posiciones restantes, se saldrá del while también si c vale false:

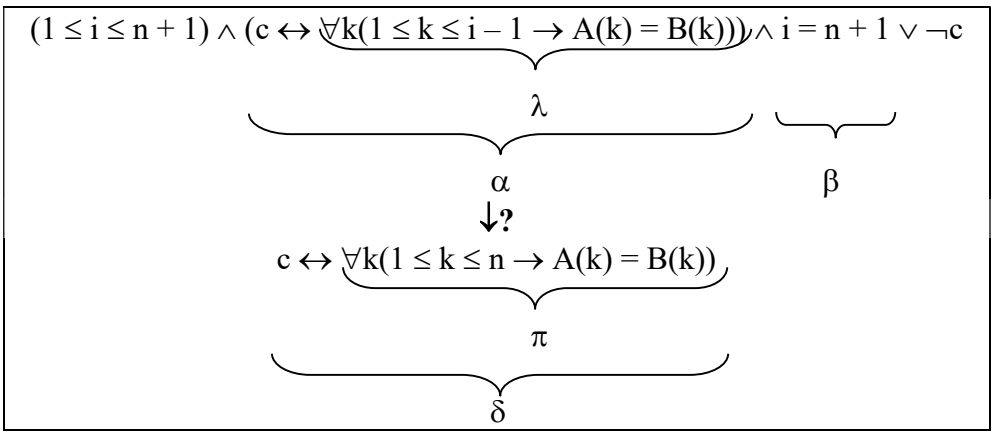
$\neg B \equiv i = n + 1 \vee \neg c$
Y por tanto, $B \equiv \neg(i = n + 1 \vee \neg c) \equiv$
$\equiv i \neq n + 1 \wedge c$

Ahora hay que comprobar que realmente la condición B es correcta y para ello se tendrán en cuenta los puntos II, IV y V de la regla del while.

II. ¿INV → def(B)?
¿INV → def($i \neq n + 1 \wedge c$)?
¿INV → true? Sí porque en la segunda parte de la implicación

tenemos true.

IV. ¿(INV ∧ ¬B) → ψ?



Como ¬B es uma disyunción, hay tres opciones para que ¬B se cumpla y hay que analizar las tres:

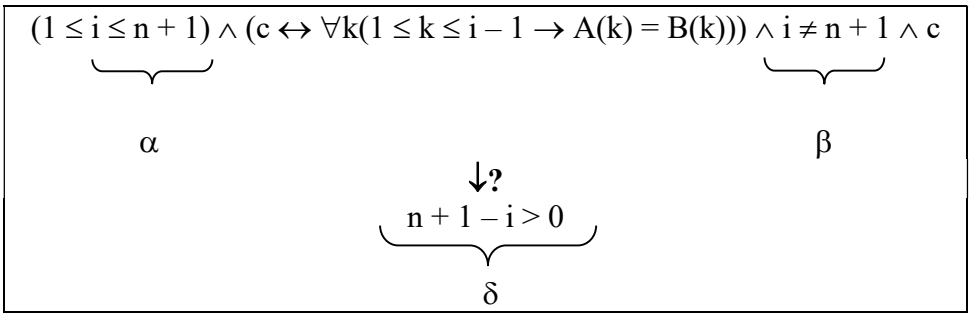
i = n + 1	¬c
True	True
True	False
False	True

Como em los dos primeros casos se cumple i = n + 1, el valor de i – 1 es igual a n y la fórmula δ es igual a la fórmula α y por tanto δ se cumple.

Em el tercer caso se cumple i ≠ n + 1 y ¬c es True, y por tanto c es False. Como α es true y c es False, λ es también false, ya que para que la implicación doble sea cierta c y λ han de ser las dos true o las dos false. Si λ es False, en el intervalo 1 .. i – 1 de los vectores A y B se tiene alguna posición en la cual A y B difieren. Y ahora la pregunta es, ¿se cumple δ? Como δ es una implicación doble, y c es false, la fórmula π debería ser false para que δ sea true. ¿Es π false? Sí, puesto que λ es false, sabemos que A y B difieren en al menos una posición y por tanto π es false y δ es true.

Por tanto la implicación (INV ∧ ¬B) → ψ se cumple.

V. ¿INV ∧ B → E > 0?



δ se cumple por α y β, ya que por α y β tenemos n + 1 > i y por tanto se cumple n + 1 – i > i – i, es decir, n + 1 – i > 0.

- **<u>Instrucciones</u>**

Tenemos que coger los dos programas de los puntos III y VI de la regla del while

Prog 1
{INV ∧ B} ¿Instrucciones? {INV}

Prog 2
{INV ∧ B ∧ E = v} ¿Instrucciones? {E < v}

- Nos tenemos que preguntar
  - ✓ ¿(INV ∧ B) → INV?
  - ✓ ¿(INV ∧ B ∧ E = v) → E < v?

La implicación (INV ∧ B) → INV se cumplirá siempre porque si INV y B son ciertos entonces INV será cierto.
En cambio la implicación (INV ∧ B ∧ E = v) → E < v no se cumplirá porque si E es igual a v entonces E no puede ser menor que v.

El que la segunda implicación no se cumpla quiere decir que tenemos que añadir una instrucción para conseguir que se cumpla E < v. Como sabemos que estamos recorriendo el vector de izquierda a derecha, para que el valor de E decrezca, tendremos que asignar i + 1 a i y calcularemos la aserción φ_3' utilizando el axioma de la asignación:

	Prog 2
AA	$\{INV \wedge B \wedge E = v\} \equiv$ $\{(1 \leq i \leq n + 1) \wedge (c \leftrightarrow \forall k(1 \leq k \leq i - 1 \rightarrow A(k) = B(k))) \wedge i \neq n + 1 \wedge c \wedge n + 1 - i = v\}$ $\{\varphi_3'\} \equiv \{def(i + 1) \wedge (E < v)_i^{i+1}\} \equiv \{true \wedge n + 1 - (i + 1) < v\} \equiv \{n - i < v\}$ i : = i + 1; $\{n + 1 - i < v\}$

Ahora nos preguntamos ¿$(INV \wedge B \wedge E = v) \rightarrow \varphi_3'$?

$$(1 \leq i \leq n + 1) \wedge (c \leftrightarrow \forall k(1 \leq k \leq i - 1 \rightarrow A(k) = B(k))) \wedge i \neq n + 1 \wedge c \wedge \underbrace{n + 1 - i = v}_{\alpha}$$

$$\downarrow ?$$

$$\underbrace{n - i < v}_{\beta}$$

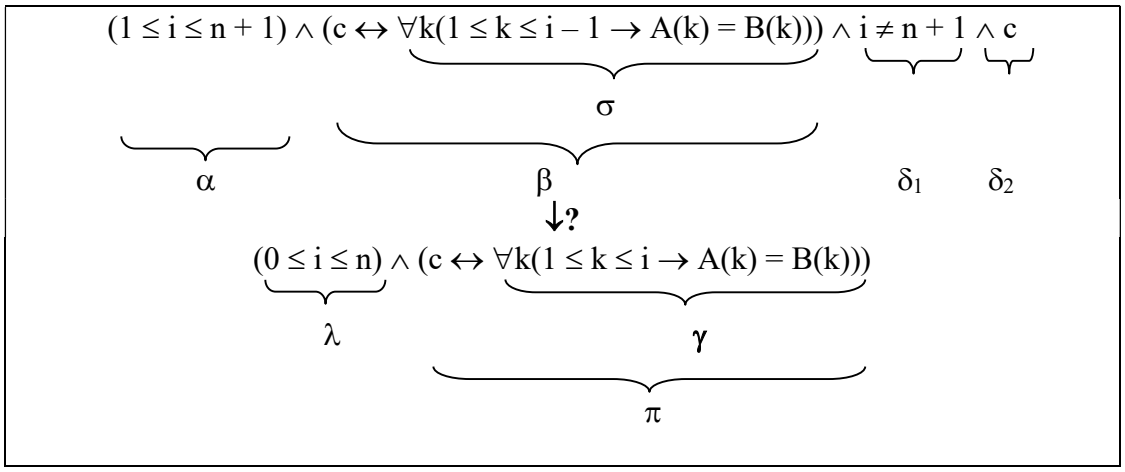
La implicación se cumple porque β es cierto por α. Si $n + 1 - i = v$, entonces $n - i = v - 1$ y $v - 1$ es menor que v.

Esto quiere decir que Prog 2 ahora es correcto pero como Prog 1 y Prog 2 han de tener las mismas instrucciones tenemos que añadir la asignación i : = i + 1; a Prog 1, luego calcularemos φ_3 utilizando el axioma de la asignación y comprobaremos si se cumple $(INV \wedge B) \rightarrow \varphi_3$.

	Prog 1
AA	$\{INV \wedge B\} \equiv$ $\equiv \{(1 \leq i \leq n + 1) \wedge (c \leftrightarrow \forall k(1 \leq k \leq i - 1 \rightarrow A(k) = B(k))) \wedge i \neq n + 1 \wedge c\}$ $\{\varphi_3\} \equiv \{def(i + 1) \wedge (INV)_i^{i+1}\}$ i : = i + 1; $\{INV\} \equiv \{(1 \leq i \leq n + 1) \wedge (c \leftrightarrow \forall k(1 \leq k \leq i - 1 \rightarrow A(k) = B(k)))\}$

$\{\varphi_3\} \equiv \{def(i + 1) \wedge (INV)_i^{i+1}\} \equiv$
$\equiv \{true \wedge (1 \leq i + 1 \leq n + 1) \wedge (c \leftrightarrow \forall k(1 \leq k \leq i + 1 - 1 \rightarrow A(k) = B(k)))\}$
$\equiv \{(0 \leq i \leq n) \wedge (c \leftrightarrow \forall k(1 \leq k \leq i \rightarrow A(k) = B(k)))\}$

Ahora nos preguntamos ¿$(INV \wedge B) \rightarrow \varphi_3$?



- λ se cumple por α y δ_1.
- Por δ_2 sabemos que c es true y como β es también true, σ tiene que ser true (por tanto σ es true por δ_2 y β). La fórmula σ dice que los vectores A y B son iguales en el intervalo 1..i – 1. Por otro lado, como c es true, para que la fórmula π sea true la fórmula γ debería ser true, pero con la información con la que se dispone no se puede afirmar que γ sea true. Por tanto, no se cumple la implicación (INB $\wedge$ B) $\rightarrow \varphi_3$. Como el objetivo es que φ_3 se cumpla, es decir, como el objetivo es que π sea cierta, la variable booleana c debería valer true si los vectores A y B son iguales en el intervalo 1..i y debería valer false si los vectores A y B no son iguales en el intervalo 1..i. Por δ_2 y β sabemos que c es true y que los vectores A y B son iguales en el intervalo 1..i – 1. Por tanto, para que se cumpla π la variable c debería seguir valiendo true si A(i) y B(i) son iguales y c debería pasar a valer false si A(i) y B(i) no son iguales. Eso se consigue con la siguiente asignación:

  c : = (A(i) = B(i));

  luego calcularemos φ_4 y comprobaremos si se cumple (INV $\wedge$ B) $\rightarrow$ φ_4

	Prog 1
AA	$\{INV \wedge B\} \equiv$ $\equiv \{(1 \le i \le n+1) \wedge (c \leftrightarrow \forall k(1 \le k \le i-1 \rightarrow A(k) = B(k))) \wedge i \ne n+1 \wedge c\}$ $\{\varphi_4\} \equiv \{def(A(i) = B(i)) \wedge (\varphi_3)_c^{(A(i) = B(i))}\}$ **c : = (A(i) = B(i));** $\{\varphi_3\} \equiv \{(0 \le i \le n) \wedge (c \leftrightarrow \forall k(1 \le k \le i \rightarrow A(k) = B(k)))\}$ i : = i + 1; $\{INV\} \equiv \{(1 \le i \le n+1) \wedge (c \leftrightarrow \forall k(1 \le k \le i-1 \rightarrow A(k) = B(k)))\}$

$$\{\varphi_4\} \equiv \{def(A(i) = B(i)) \wedge (\varphi_3)_c^{(A(i) = B(i))}\} \equiv$$
$$\equiv \{(1 \le i \le n) \wedge (0 \le i \le n) \wedge ((A(i) = B(i)) \leftrightarrow \forall k(1 \le k \le i \rightarrow A(k) = B(k)))\} \equiv$$
$$\equiv \{(1 \le i \le n) \wedge ((A(i) = B(i)) \leftrightarrow \forall k(1 \le k \le i \rightarrow A(k) = B(k)))\}$$

Ahora nos preguntamos ¿$(INV \wedge B) \rightarrow \varphi_4$?

$$\underbrace{(1 \le i \le n+1)}_{\alpha} \wedge \underbrace{(c \leftrightarrow \underbrace{\forall k(1 \le k \le i-1 \rightarrow A(k) = B(k))}_{\sigma})}_{\beta} \wedge \underbrace{i \ne n+1}_{\delta_1} \wedge \underbrace{c}_{\delta_2}$$

$$\beta \downarrow ?$$

$$\underbrace{(1 \le i \le n)}_{\lambda} \wedge \underbrace{((A(i) = B(i)) \leftrightarrow \underbrace{\forall k(1 \le k \le i \rightarrow A(k) = B(k)))}_{\gamma_2}}_{\pi}$$

- λ se cumple por α y δ_1.
- Para que π se cumpla las fórmulas γ_1 y γ_2 han de ser las dos true o las dos false. Por δ_2 sabemos que c es true. Como β es true, la fórmula σ es también true, A y B son iguales en el intervalo 1..i – 1. Sabiendo eso, si γ_1 es true también γ_2 será true y por tanto π también será true ya que tendremos true $\leftrightarrow$ true. Si γ_1 es false entonces γ_2 también será false ya que no será verdad que A y B son iguales en el intervalo 1..i y por tanto π será true ya que se tendrá false $\leftrightarrow$ false.
- Por tanto la implicación $(INV \wedge B) \rightarrow \varphi_4$ se cumple.

Prog 1 ahora es correcto. Pero como Prog 1 y Prog 2 han de tener las mismas instrucciones tenemos que añadir la asignación **c : = (A(i) = B(i));** a Prog 2, luego calcular φ_4' utilizando el axioma de la asignación y comprobar si se cumple $(INV \wedge B \wedge E = v) \rightarrow \varphi_4'$.

	Prog 2
AA	$\{INV \wedge B \wedge E = v\} \equiv$ $\{(1 \le i \le n+1) \wedge (c \leftrightarrow \forall k(1 \le k \le i-1 \rightarrow A(k) = B(k))) \wedge i \ne n+1 \wedge c \wedge n+1-i = v\}$ $\{\varphi_4'\} \equiv \{def((A(i) = B(i))) \wedge (\varphi_3')_c^{(A(i) = B(i))}\}$ **c : = (A(i) = B(i));** $\{\varphi_3'\} \equiv \{n - i < v\}$ **i : = i + 1;** $\{E < v\} \equiv \{n + 1 - i < v\}$

$$\{\varphi_4'\} \equiv \{def((A(i) = B(i))) \wedge (\varphi_3')_c^{(A(i) = B(i))}\} \equiv$$
$$\equiv \{(1 \le i \le n) \wedge n - i < v\}$$

Ahora nos preguntamos ¿$(INV \wedge B \wedge E = v) \rightarrow \varphi_4'$?

$$\underbrace{(1 \le i \le n+1)}_{\alpha} \wedge (c \leftrightarrow \forall k(1 \le k \le i-1 \rightarrow A(k) = B(k))) \underbrace{\wedge\ i \ne n+1}_{\beta} \wedge c \wedge \underbrace{n+1-i = v}_{\delta}$$

$$\downarrow ?$$

$$\underbrace{(1 \le i \le n)}_{\text{Por } \alpha \text{ y } \beta} \wedge \underbrace{n - i < v}_{\text{por } \delta, \text{ ya que por } \delta \text{ tenemos } n - i = v - 1 \text{ y } v - 1 \text{ es menor que } v}$$

Como se han cumplido las dos implicaciones, $(INV \wedge B) \rightarrow \varphi_4$ y $(INV \wedge B \wedge E = v) \rightarrow \varphi_4'$ ya hemos terminado de derivar el programa. El programa es el siguiente:

```
{φ}
{φ2}
c : = true;
{φ1}
i : = 1;

while {INV} i ≠ n + 1 and c loop
      {φ4} {φ4'}
      c : = (A(i) = B(i));
      {φ3} {φ3'}
      i : = i + 1;
end loop;

{ψ}
E
```

Sabemos que el programa es **correcto** porque se ha construido siguiendo la regla del while y además el propio método calcula **documentación** del programa, que vendrá dada por $\{\varphi_1\}$, $\{\varphi_2\}$, $\{\varphi_3\}$, $\{\varphi_3'\}$, $\{\varphi_4\}$ y $\{\varphi_4'\}$.

4.2.6. DECIDIR SI X ESTÁ EN A(1..N) (EJERCICIO 18)

Teniendo en cuenta la regla del while (RWH) y el axioma de la asignación (AA) del Cálculo de Hoare, se va a derivar un programa que dados un número x y un vector A(1..n), decide en la variable booleana *c* si x aparece en A(1..n). El programa derivado deberá ser eficiente en el sentido de que al recorrer el vector, si en un momento se detecta que la respuesta va a ser afirmativa el while deberá terminar sin analizar las posiciones restantes. El programa derivado ha de ser correcto con respecto a φ, ψ, INV y E.

$\{\varphi\} \equiv \{n \geq 1\}$ ¿Inicializaciones? {INV} **while** {INV} {E} ¿B? **loop**   ¿Instrucciones? **end loop;** $\{\psi\} \equiv \{c \leftrightarrow \exists k(1 \leq k \leq n \wedge A(k) = x)\}$
$\{INV\} \equiv \{(0 \leq i \leq n) \wedge (c \leftrightarrow \exists k(1 \leq k \leq i \wedge A(k) = x))\}$ $E = n - i$

Antes de empezar a derivar las instrucciones tenemos que fijarnos en la expresión cota E. Como tiene la forma "límite superior – variable índice", sabemos que el vector se ha de recorrer de izquierda a derecha.

- **Inicializaciones**
  Para obtener las inicializaciones se tiene en cuenta el primer punto de la regla del while: φ ha de implicar el invariante. Si es así, no hace falta inicializar nada, y en caso contrario sí hace falta alguna inicialización. Tras incluir una nueva inicialización se calcula su aserción previa utilizando el Axioma de la Asignación y se comprobará si la nueva fórmula es implicada por φ. El proceso de añadir nuevas inicializaciones continuará hasta obtener una fórmula que sí es implicada por φ.

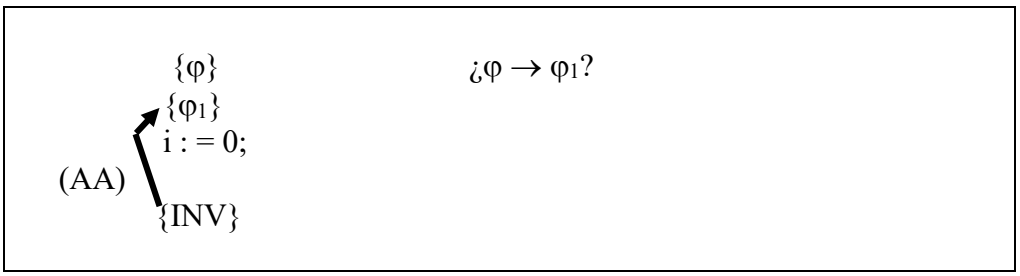
  ¿$\varphi \rightarrow INV$?

$$\underbrace{n \geq 1}_{\delta} \;\downarrow ?\; \underbrace{(0 \leq i \leq n)}_{\alpha} \wedge \underbrace{(c \leftrightarrow \exists k(1 \leq k \leq i \wedge A(k) = x))}_{\beta}$$

La implicación no se cumple porque en la primera parte de la implicación (δ) no tenemos ninguna información sobre i y c para asegurar que se cumple la segunda parte de la implicación (α y β). Como el objetivo es que la implicación se cumpla, tenemos que inicializar i y c con valores que hagan que la implicación sea cierta. Como β depende de i y c, hay que empezar fijándose en α, que sólo depende de i.

Como sabemos que el vector se ha de recorrer de izquierda a derecha, para que se cumpla α basta con asignar 0 a la variable i. Por tanto mirando en α asignamos a i el valor más pequeño que puede tomar según el invariante.

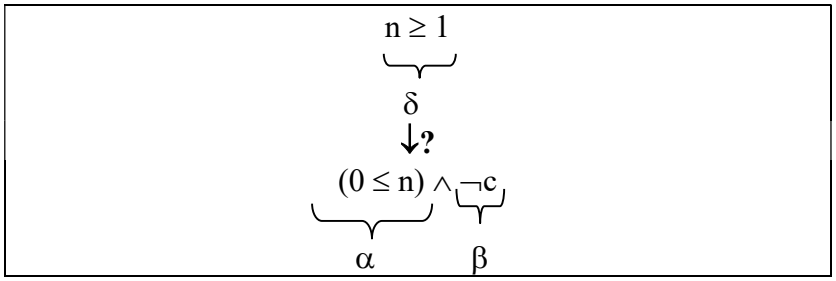
Ahora calculamos la aserción $\{\varphi_1\}$ utilizando el axioma de la asignación



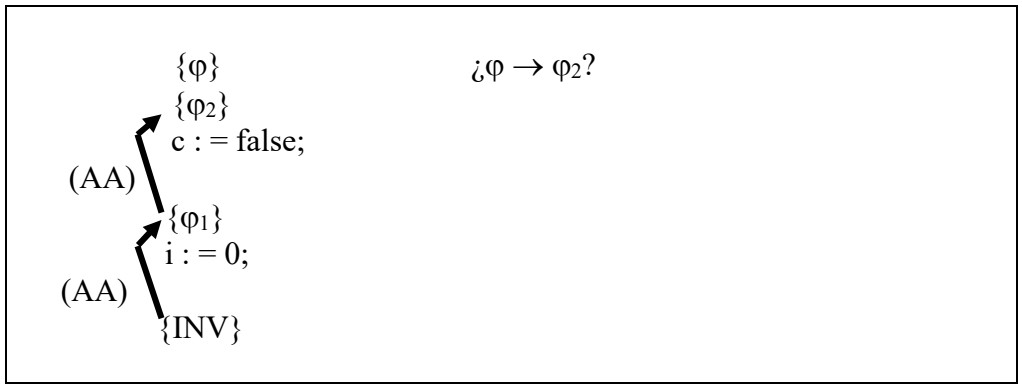
$$\{\varphi_1\} \equiv \{def(0) \wedge (INV)_i^0\} \equiv$$
$$\equiv \{true \wedge (0 \le 0 \le n) \wedge (c \leftrightarrow \exists k(1 \le k \le 0 \wedge A(k) = x))\} \equiv^{\text{simplificación}}$$
$$\equiv \{(0 \le n) \wedge (c \leftrightarrow \exists k(1 \le k \le 0 \wedge A(k) = x))\} \equiv^{\text{simplificación}}$$
$$\equiv \{(0 \le n) \wedge c \leftrightarrow false\} \equiv^{\text{simplificación}}$$
$$\equiv \{(0 \le n) \wedge \neg c\}$$

$\exists k(1 \le k \le 0 \wedge A(k) = x))$ es false porque el intervalo $1 \le k \le 0$ es vacío.

- ¿φ → φ1?



El que se cumpla n ≥ 1 sí implica 0 ≤ n pero no implica que se cumpla que c sea false, es decir, lo que nos falta es que la variable c valga false, y esa es la inicialización que nos queda.



$\{\varphi_2\} \equiv \{def(false) \wedge (\varphi_1)_c^{false}\} \equiv$
$\equiv \{true \wedge (0 \leq n) \wedge \neg false\} \equiv$ simplificación
$\equiv \{true \wedge (0 \leq n) \wedge true\} \equiv$ simplificación
$\equiv (0 \leq n)$

- ¿$\varphi \rightarrow \varphi_2$?

$$n \geq 1$$
$$\downarrow ?$$
$$(0 \leq n)$$

La implicación $\varphi \rightarrow \varphi_2$ sí se cumple y, por tanto, ya hemos terminado con las inicializaciones.

- **Condición del while (B)**

Primero se calcula **¬B** respondiendo a la pregunta **¿Cuándo salimos del while?**

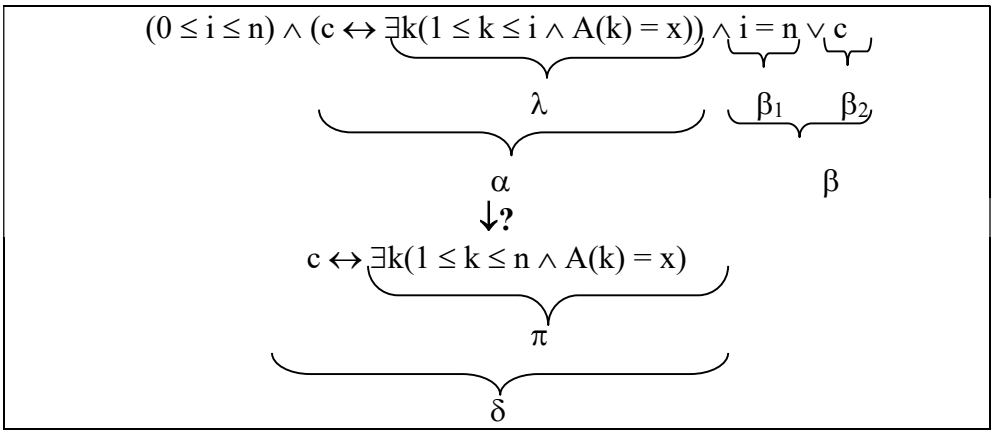
Sabiendo que estamos recorriendo la tabla de derecha a izquierda y que según el invariante el límite superior de i es n, cuando i valga n hay que salir del while pero además el programa ha de ser eficiente en el sentido de que si nos damos cuenta de que x aparece en el vector, se ha de parar sin comparar las posiciones restantes, se saldrá del while también si c vale true:

$\neg B \equiv i = n \vee c$
Y por tanto, $B \equiv \neg(i = n \vee c) \equiv$
$\equiv i \neq n \wedge \neg c$

Ahora hay que comprobar que realmente la condición B es correcta y para ello se tendrán en cuenta los puntos II, IV y V de la regla del while.

II. ¿INV → def(B)?
¿INV → def($i \neq n \wedge \neg c$)?
¿INV → true? Sí porque en la segunda parte de la implicación tenemos true.

IV. ¿(INV ∧ ¬B) → ψ?



Como ¬B es uma disyunción, hay tres opciones para que ¬B se cumpla y hay que analizar las tres:

i = n	c
True	True
True	False
False	True

Como em los dos primeros casos se cumple i = n, la fórmula δ es igual a la fórmula α y por tanto δ se cumple.

Em el tercer caso se cumple i ≠ n y c es True. Como α es true y c es True, λ es también true, ya que para que la implicación doble sea cierta c y λ han de ser las dos true o las dos false. Si λ es True, x aparece en el intervalo 1 .. i del vector A. Y ahora la pregunta es, ¿se cumple δ? Como δ es una implicación doble, y c es true, la fórmula π debería ser true para que δ sea true. ¿Es π true? Sí, puesto que λ es true, sabemos que x aparece en el intervalo 1..i de A y por tanto x aparece en A(1..n) y por tanto π es true y δ es true.

Por tanto la implicación (INV ∧ ¬B) → ψ se cumple.

V. ¿INV ∧ B → E > 0?

$$(0 \leq \underbrace{i \leq n}_{\alpha}) \wedge (c \leftrightarrow \exists k(1 \leq k \leq i \wedge A(k) = x)) \wedge \underbrace{i \neq n}_{\beta} \wedge \neg c$$

$$\downarrow ?$$

$$\underbrace{n - i > 0}_{\delta}$$

δ se cumple por α y β, ya que por α y β tenemos n > i y por tanto se cumple n – i > i – i, es decir, n – i > 0.

- **<u>Instrucciones</u>**

Tenemos que coger los dos programas de los puntos III y VI de la regla del while

Prog 1
{INV ∧ B} ¿Instrucciones? {INV}

Prog 2
{INV ∧ B ∧ E = v} ¿Instrucciones? {E < v}

- Nos tenemos que preguntar
  - ✓ ¿(INV ∧ B) → INV?
  - ✓ ¿(INV ∧ B ∧ E = v) → E < v?

La implicación (INV ∧ B) → INV se cumplirá siempre porque si INV y B son ciertos entonces INV será cierto.
En cambio la implicación (INV ∧ B ∧ E = v) → E < v no se cumplirá porque si E es igual a v entonces E no puede ser menor que v.

El que la segunda implicación no se cumpla quiere decir que tenemos que añadir una instrucción para conseguir que se cumpla E < v. Como sabemos que estamos recorriendo el vector de izquierda a derecha, para que el valor de E decrezca, tendremos que asignar i + 1 a i y calcularemos la aserción φ_3' utilizando el axioma de la asignación:

	Prog 2
AA	$\{INV \wedge B \wedge E = v\} \equiv$ $\{(0 \leq i \leq n) \wedge (c \leftrightarrow \exists k(1 \leq k \leq i \wedge A(k) = x)) \wedge i \neq n \wedge \neg c \wedge n - i = v\}$ $\{\varphi_3'\} \equiv \{def(i + 1) \wedge (E < v)_i^{i+1}\} \equiv \{true \wedge n - (i + 1) < v\} \equiv \{n - i - 1 < v\}$ i : = i + 1; $\{n - i < v\}$

Ahora nos preguntamos ¿$(INV \wedge B \wedge E = v) \rightarrow \varphi_3'$?

$$(0 \leq i \leq n) \wedge (c \leftrightarrow \exists k(1 \leq k \leq i \wedge A(k) = x)) \wedge i \neq n \wedge \neg c \wedge \underbrace{n - i = v}_{\alpha}$$

$$\downarrow ?$$

$$\underbrace{n - i - 1 < v}_{\beta}$$

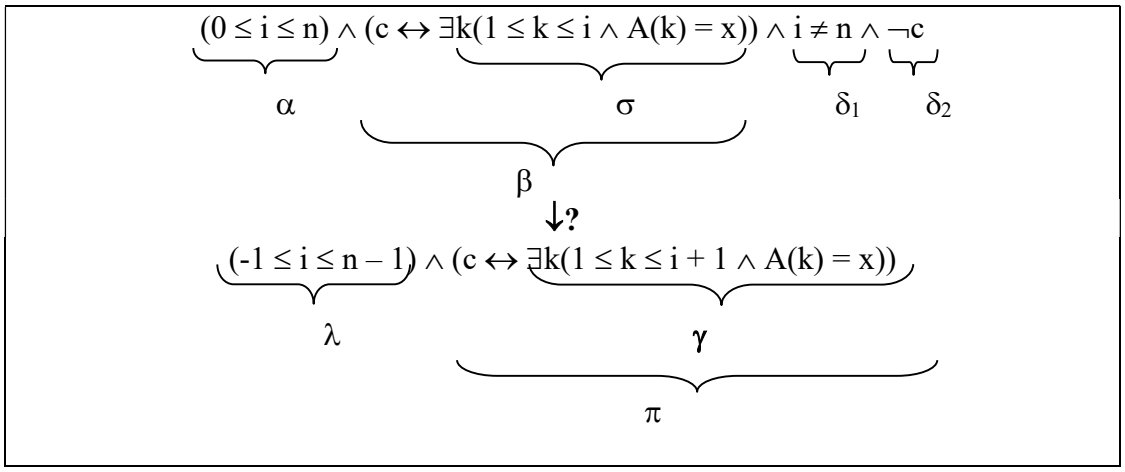
La implicación se cumple porque β es cierto por α. Si $n - i = v$, entonces $n - i - 1 = v - 1$ y $v - 1$ es menor que v.

Esto quiere decir que Prog 2 ahora es correcto pero como Prog 1 y Prog 2 han de tener las mismas instrucciones tenemos que añadir la asignación i : = i + 1; a Prog 1, luego calcularemos φ_3 utilizando el axioma de la asignación y comprobaremos si se cumple $(INV \wedge B) \rightarrow \varphi_3$.

	Prog 1
AA	$\{INV \wedge B\} \equiv$ $\equiv \{(0 \leq i \leq n) \wedge (c \leftrightarrow \exists k(1 \leq k \leq i \wedge A(k) = x)) \wedge i \neq n \wedge \neg c\}$ $\{\varphi_3\} \equiv \{def(i + 1) \wedge (INV)_i^{i+1}\}$ i : = i + 1; $\{INV\} \equiv \{(0 \leq i \leq n) \wedge (c \leftrightarrow \exists k(1 \leq k \leq i \wedge A(k) = x))\}$

$\{\varphi_3\} \equiv \{def(i + 1) \wedge (INV)_i^{i+1}\} \equiv$
$\equiv \{true \wedge (0 \leq i + 1 \leq n) \wedge (c \leftrightarrow \exists k(1 \leq k \leq i + 1 \wedge A(k) = x))\}$
$\equiv \{(-1 \leq i \leq n - 1) \wedge (c \leftrightarrow \exists k(1 \leq k \leq i + 1 \wedge A(k) = x))\}$

Ahora nos preguntamos ¿$(INV \wedge B) \rightarrow \varphi_3$?



- λ se cumple por α y δ_1.
- Por δ_2 sabemos que c es false y como β es true, σ tiene que ser false (por tanto σ es false por δ_2 y β). Como la fórmula σ es false, x no aparece en el intervalo 1..i. Por otro lado, como c es false, para que la fórmula π sea true la fórmula γ debería ser false, pero con la información con la que se dispone no se puede afirmar que γ sea false ya que no sabemosa si en la posición i + 1 de A aparece x o no. Por tanto, no se cumple la implicación (INB $\wedge$ B) $\rightarrow$ φ_3. Como el objetivo es que φ_3 se cumpla, es decir, como el objetivo es que π sea cierta, la variable booleana c debería valer true si x aparece en el intervalo 1..i + 1 de A y debería valer false si x no aparece en el intervalo 1..i + 1 de A los vectores A. Por δ_2 y β sabemos que c es false y que x no aparece en el intervalo 1..i del vector A. Por tanto, para que se cumpla π la variable c debería seguir valiendo false si en la posición i + 1 de A no aparece x y c debería pasar a valer true si en la posición i + 1 de A aparece x. Eso se consigue con la siguiente asignación:

  c : = (A(i) = x);

  luego calcularemos φ_4 y comprobaremos si se cumple (INV $\wedge$ B) $\rightarrow$ φ_4

	Prog 1
AA	{INV ∧ B} ≡ ≡ {(0 ≤ i ≤ n) ∧ (c ↔ ∃k(1 ≤ k ≤ i ∧ A(k) = x)) ∧ i ≠ n ∧ ¬c} {φ4} ≡ {def(A(i) = x) ∧ $(\varphi_3)_c^{(A(i) = x)}$} **c : = (A(i) = x);** {φ3} ≡ {(-1 ≤ i ≤ n – 1) ∧ (c ↔ ∃k(1 ≤ k ≤ i + 1 ∧ A(k) = x))} i : = i + 1; {INV} ≡ {(0 ≤ i ≤ n) ∧ (c ↔ ∃k(1 ≤ k ≤ i ∧ A(k) = x))}

$$\{\varphi_4\} \equiv \{\text{def}(A(i) = x) \wedge (\varphi_3)_c^{(A(i) = x)}\} \equiv$$
$$\equiv \{(1 \le i \le n) \wedge (-1 \le i \le n - 1) \wedge ((A(i) = x) \leftrightarrow \exists k(1 \le k \le i + 1 \wedge A(k) = x))\} \equiv$$
$$\{(1 \le i \le n) \wedge (-1 \le i \le n - 1) \wedge ((A(i) = x) \leftrightarrow \exists k(1 \le k \le i + 1 \wedge A(k) = x))\}$$
$$\equiv \{(1 \le i \le n - 1) \wedge ((A(i) = x) \leftrightarrow \exists k(1 \le k \le i + 1 \wedge A(k) = x))\}$$

Ahora nos preguntamos ¿(INV ∧ B) → φ4?

$$\underbrace{(0 \le i \le n)}_{\alpha} \wedge \underbrace{(c \leftrightarrow \underbrace{\exists k(1 \le k \le i \wedge A(k) = x)}_{\sigma})}_{\beta} \wedge \underbrace{i \ne n}_{\delta_1} \wedge \underbrace{\neg c}_{\delta_2}$$

$$\beta \downarrow ?$$

$$\underbrace{(1 \le i \le n - 1)}_{\lambda} \wedge \underbrace{((A(i) = x)}_{\gamma_1} \leftrightarrow \underbrace{\exists k(1 \le k \le i + 1 \wedge A(k) = x))}_{\gamma_2}$$

$$\pi$$

- λ se cumple por α y δ_1.
- Para que π se cumpla las fórmulas γ_1 y γ_2 han de ser las dos true o las dos false. Por δ_2 sabemos que c es false. Como β es true, la fórmula σ es también false, es decir x no aparece en el intervalo 1..i de A. Sabiendo eso, si γ_1 es true también γ_2 será true y por tanto π también será true ya que tendremos true ↔ true. Si γ_1 es false entonces γ_2 también será false ya que no será verdad que x aparece en el intervalo 1..i + 1 y por tanto π será true ya que se tendrá false ↔ false.
- Por tanto la implicación (INV ∧ B) → φ4 se cumple.

Prog 1 ahora es correcto. Pero como Prog 1 y Prog 2 han de tener las mismas instrucciones tenemos que añadir la asignación **c : = (A(i) = x);** a Prog 2, luego calcular φ4' utilizando el axioma de la asignación y comprobar si se cumple (INV ∧ B ∧ E = v) → φ4'.

	Prog 2
AA	$\{INV \wedge B \wedge E = v\} \equiv$ $\{(0 \le i \le n) \wedge (c \leftrightarrow \exists k(1 \le k \le i \wedge A(k) = x)) \wedge i \ne n \wedge \neg c \wedge n - i = v\}$ $\{\varphi_4'\} \equiv \{def((A(i) = x)) \wedge (\varphi_3')_c^{(A(i) = x)}\}$ **c : = (A(i) = x);** $\{\varphi_3'\} \equiv \{n - i < v\}$ **i : = i + 1;** $\{E < v\} \equiv \{n + 1 - i < v\}$

$$\{\varphi_4'\} \equiv \{def((A(i) = x)) \wedge (\varphi_3')_c^{(A(i) = x)}\} \equiv$$
$$\equiv \{(1 \le i \le n) \wedge n - i < v\}$$

Ahora nos preguntamos ¿$(INV \wedge B \wedge E = v) \rightarrow \varphi_4'$?

$$\underbrace{(0 \le i \le n)}_{\alpha} \wedge (c \leftrightarrow \exists k(1 \le k \le i \wedge A(k) = x)) \wedge \underbrace{i \ne n}_{\beta} \wedge \neg c \wedge \underbrace{n - i = v}_{\delta}$$

$$\downarrow ?$$

$$\underbrace{(1 \le i \le n)}_{\text{Por } \alpha \text{ y } \beta} \wedge \underbrace{n - i < v}_{\text{por } \delta, \text{ ya que por } \delta \text{ tenemos } n - i = v - 1 \text{ y } v - 1 \text{ es menor que } v}$$

Como se han cumplido las dos implicaciones, $(INV \wedge B) \rightarrow \varphi_4$ y $(INV \wedge B \wedge E = v) \rightarrow \varphi_4'$ ya hemos terminado de derivar el programa. El programa es el siguiente:

```
{φ}
{φ2}
c : = false;
{φ1}
i : = 0;

while {INV} i ≠ n and not c loop
      {φ4} {φ4'}
      c : = (A(i) = x);
      {φ3} {φ3'}
      i : = i + 1;
end loop;

{ψ}
E
```

Sabemos que el programa es **correcto** porque se ha construido siguiendo la regla del while y además el propio método calcula **documentación** del programa, que vendrá dada por $\{\varphi_1\}$, $\{\varphi_2\}$, $\{\varphi_3\}$, $\{\varphi_3'\}$, $\{\varphi_4\}$ y $\{\varphi_4'\}$.