TEMA 4: DISEÑO DE ALGORITMOS RECURSIVOS

6. Inmersión de especificaciones

- La inmersión de programas se puede realizar también directamente sobre la especificación, sin conocer el programa.
  - Nos ahorramos escribir dos veces la función.
  - En ocasiones es muy difícil o imposible escribir el programa recursivo sin la inmersión.
- Ejemplo: Suma de un vector.

```
func suma(a: Vector) dev s: Entero
{Pre: cierto}
{Post: s = Σα: 1 <= α <= N: a[α]}
```

Es imposible realizarla recursivamente.

6. Inmersión de especificaciones

- Realizamos una inmersión de parámetros sobre la especificación:

```
func suma(a: Vector; i: Natural) dev s: Entero
{Pre: i <= N}
{Post: s = Σα: 1 <= α <= i: a[α]}
```

Ahora si que se puede implementar recursivamente.

- Para realizar una inmersión de una especificación existen dos posibilidades:
  - **Debilitar la postcondición.**
  - **Reforzar la precondición.**

6.1. Debilitamiento de asertos

- Lo primero es aprender a debilitar un aserto:
  - Si tenemos un aserto con conjunciones, se puede eliminar una de ellas:

    $A1 \wedge A2 \wedge A3 \quad \text{-->} \quad A1 \wedge A2$

  - Podemos añadir un nuevo aserto mediante una disyunción:

    $A1 \quad \text{-->} \quad A1 \vee A2$

  - Cuando el aserto no contiene conjunciones, podemos añadir una nueva sustituyendo constantes por variables.

6.1. Debilitamiento de asertos

- Ejemplo: Suma de un vector

```
s = Σα: 1 <= α <= N: a[α]
```

  - Varias posibilidades:

```
s = Σα: 1 <= α <= i: a[α] ∧ i = N
s = Σα: i <= α <= N: a[α] ∧ i = 1
s = Σα: 1 + i <= α <= N: a[α] ∧ i = 0
```

  - Ahora podemos debilitar el aserto eliminando la segunda parte:

```
s = Σα: 1 <= α <= i: a[α] ∧ i <= N
```

  - Normalmente, al debilitar el aserto, tendremos que añadir condiciones de dominio a la precondición:

```
i <= N
```

6.2. Debilitar postcondición

- Inmersión fácil y aplicable con recursividad múltiple.
- No se obtiene recursividad final.
- Consiste en debilitar la postcondición sustituyendo constantes por variables y colocando las variables como parámetros de entrada.

Ejemplo: Suma Vector

```
func suma(a: Vector) dev s: Entero
{Pre: cierto}
{Post: s = Σα: 1 <= α <= N: a[α]}
```

```
func i_suma(a: Vector; i: Natural) dev s: Entero
{Pre: i <= N}
{Post: s = Σα: 1 <= α <= i: a[α]}
```

6.2. Debilitar postcondición

Suma vector completo:

```
func i_suma(a: Vector; i: Natural) dev s: Entero
{Pre: i <= N; Dec: i}
[i = 0 -> s := 0
 i > 0 -> s := i_suma(a, i-1);
          s := s + a[i]
]
{Post: s = Σα: 1 <= α <= i: a[α]}

func suma(a: Vector) dev s: Entero
{Pre: cierto}
    s := i_suma(a, N)
{Post: s = Σα: 1 <= α <= N: a[α]}
```

6.2. Debilitar postcondición

Suma vector utilizando otro debilitamiento:

```
func i_suma2(a: Vector; i: Natural) dev s: Entero
{Pre: 0 < i <= N; Dec: N-i+1}
[i = N+1 -> s := 0
 i < N+1 -> s := i_suma2(a, i+1);
            s := s + a[i]
]
{Post: s = Σα: i <= α <= N: a[α]}

func suma(a: Vector) dev s: Entero
{Pre: cierto}
    s := i_suma(a, 1)
{Post: s = Σα: 1 <= α <= N: a[α]}
```

6.3. Reforzar precondición

- Inmersión más complicada y no se puede aplicar con recursividad múltiple.
- Se obtiene recursividad final.
- Consiste en reforzar la precondición añadiéndole la postcondición debilitada. Las variables que jugaban el papel de resultado serán ahora resultados parciales y habrá que renombrarlas y ponerlas como parámetros de entrada.

```
func suma(a: Vector) dev s: Entero
{Pre: cierto}
{Post: s = Σα: 1 <= α <= N: a[α]}

func i2_suma(a: Vector; i: Natural; ss: Ent) dev s: Ent
{Pre: s = Σα: 1 <= α <= i: a[α] ∧ i <= N}
{Post: s = Σα: 1 <= α <= N: a[α]}
```

6.3. Reforzar precondición

Suma vector completo

```
func i2_suma(a: Vector; i: Natural; ss: Ent) dev s: Ent
{Pre: s = Σα: 1 <= α <= i: a[α] ∧ i <= N; Dec: N-i}
[i = N -> s := ss
 i < N -> s := i2_suma(a, i+1, ss+a[i+1]);
]
{Post: s = Σα: 1 <= α <= N: a[α]}

func suma(a: Vector) dev s: Entero
{Pre: cierto}
    s := i2_suma(a, 0, 0)
{Post: s = Σα: 1 <= α <= N: a[α]}
```