Ciclos: Ejemplo "Aplanar lista"

Queremos implementar una función que tome como único argumento una lista de listas de enteros, que llamamos `xss`, y que devuelva otra lista con los enteros de las listas de `xss`, en el orden en que aparecen. Por ejemplo, para la lista `[[1,2,3], [], [4], [5,6]]`, debe devolver la lista `[1,2,3,4,5,6]`.

1. Primera versión

Consideremos la función `aplanar` especificada e implementada de la siguiente manera:

```
1  from typing import List
2
3  def aplanar(xss:List[List[int]]) -> List[int]:
4      '''
5      Requiere: nada.
6      Devuelve: la concatenación de las listas de xss (es decir: una lista
7        con los enteros de las listas de xss, en el orden en que aparecen)
8      '''
9      res:List[int] = []
10     i:int = 0
11     # (A)
12     while i < len(xss):
13         # (B)
14         res = res + xss[i]
15         i = i + 1
16         # (C)
17     # (D)
18     return res
```

Se pide:

1. Mostrar que el ciclo termina.
2. Elegir un predicado invariante $\mathcal{I}$, y usarlo para mostrar que cuando termina el ciclo, se verifique la especificación de la función.

1.1. Terminación del ciclo

- Antes del ciclo, la variable `i` se inicializa en 0 (línea 10).
- En cada ejecución del cuerpo del ciclo, `i` se incrementa en 1 (línea 15).
- La lista `xss` no se modifica en el cuerpo del ciclo, por lo que su longitud `len(xss)` nunca cambia.
- Entonces, es inevitable que en algún momento, `i` llegue al valor de `len(xss)`.
- En ese momento, la condición `i<len(xss)` será `False`, por lo que el ciclo terminará. □

1.2. Correctitud del ciclo

Primero identifiquemos qué cosas podemos afirmar sobre las variables del programa, que sean ciertas en los puntos (A), (B), (C) y (D) indicados en el código.

Para entender mejor el código, ejecutemos manualmente esta función para una lista de listas de enteros `xss` que vale `[[1,1],[2,2],[3,3]]`:

1. `i` y `res` empiezan valiendo 0 y `[]`, respectivamente.

2. La condición `i<len(xss)` es verdadera ($0 < 3$), por lo que ingresamos al cuerpo del ciclo.
3. Al terminar la primera iteración del ciclo, `i` vale 1 y `res` vale `[1,1]`.
4. La condición `i<len(xss)` es verdadera ($1 < 3$), por lo que ingresamos al cuerpo del ciclo.
5. Al terminar la segunda iteración del ciclo, `i` vale 2 y `res` vale `[1,1,2,2]`.
6. La condición `i<len(xss)` es verdadera ($2 < 3$), por lo que ingresamos al cuerpo del ciclo.
7. Al terminar la tercera iteración del ciclo, `i` vale 3 y `res` vale `[1,1,2,2,3,3]`.
8. Ahora la condición `i<len(xss)` es **falsa** (no es cierto que $3 < 3$), por lo que esta vez **no** ingresamos al cuerpo del ciclo.
9. Se retorna el valor actual de `res`, que es `[1,1,2,2,3,3]`.

Pasemos a una tabla los valores de las variables `i` y `res`:

`i`	`res`
0	`[] = []`
1	`[1,1] = [] + [1,1]`
2	`[1,1,2,2] = [] + [1,1] + [2,2]`
3	`[1,1,2,2,3,3] = [] + [1,1] + [2,2] + [3,3]`

Viendo esta tabla, podemos afirmar dos cosas que son verdaderas en cada iteración:

- La variable `i` se encuentra entre 0 y `len(xss)` inclusive. Es decir: $0 \leq$ `i` $\leq$ `len(xss)`.
- La variable `res` vale la concatenación de las listas de `xss` entre las posiciones 0 e `i-1` (inclusive). Es decir: `res` $=$ `xss[0]` $+$ `xss[1]` $+ \ldots +$ `xss[i-1]`.

Así, nuestro predicado invariante $\mathcal{I}$ será: **0** $\leq$ **`i`** $\leq$ **`len(xss)`**, y **`res` vale la concatenación de las listas de `xss` entre las posiciones 0 e `i-1` (inclusive)**.

Sigamos ahora el siguiente razonamiento, que nos lleva desde el principio hasta el final de la ejecución de la función `aplanar`:

- $\mathcal{I}$ es cierto en el punto (A), donde `i` vale 0 y `res` vale `[]`. (Notar que la concatenación de un conjunto vacío de listas equivale a la lista vacía `[]`.)
- Supongamos que $\mathcal{I}$ es cierto en el punto (B). Si llamamos ϕ al valor que tiene `i` en ese momento, las siguientes afirmaciones son verdaderas:

  $$\texttt{i} = \phi$$

  $$\texttt{res} = \texttt{xss[0]} + \ldots + \texttt{xss[i-1]} = \texttt{xss[0]} + \ldots + \texttt{xss[}\phi\texttt{-1]}$$

  En el cuerpo del ciclo, la línea 14 le agrega a `res` un término con el valor de `xss[i]` (es decir, `xss[`ϕ`]`). Después, `i` se incrementa en 1 (línea 15). En consecuencia, al terminar el cuerpo del ciclo, en el punto (C), valen las siguientes afirmaciones:

  $$\texttt{i} = \phi + 1$$

  $$\texttt{res} = \texttt{xss[0]} + \ldots + \texttt{xss[}\phi\texttt{-1]} + \texttt{xss[}\phi\texttt{]} = \texttt{xss[0]} + \ldots + \texttt{xss[i-1]}$$

  En otras palabras, $\mathcal{I}$ vuelve a ser cierto al terminar el cuerpo del ciclo, en el punto (C).
- De esta manera, sabemos que en cada iteración del ciclo, $\mathcal{I}$ empieza y termina siendo cierto. Después de (C) se evalúa la condición `i<len(xss)`, lo cual no modifica el valor de ninguna variable. Así, mientras la condición resulte verdadera, a lo largo de sucesivas iteraciones tendremos que $\mathcal{I}$ vale en (B), vale en (C), vale en (B), vale en (C), vale en (B), vale en (C), ...
- En algún momento el ciclo termina porque la condición resulta falsa. Dado que evaluar la condición `i<len(xss)` no modifica el valor de las variables, $\mathcal{I}$ sigue siendo cierto al llegar al punto (D), y es fácil ver que `i` vale `len(xss)`. Entonces, en este momento estamos en condiciones de afirmar que **`res` vale la concatenación de las listas de `xss` entre las posiciones 0 y `len(xss)-1` (inclusive)**. En otras palabras: **`res` vale la concatenación de las listas de `xss`**. Y eso es precisamente lo que devuelve la función `aplanar` de acuerdo a su especificación. □

2. Segunda versión

Supongamos ahora que no disponemos del operador + para concatenar listas (ni ninguna otra función parecida). En este caso, una opción sería definir y usar nuestra propia función `concatenar`, de esta manera:

```
1  from typing import List
2
3  def aplanar(xss:List[List[int]]) -> List[int]:
4      '''
5      Requiere: nada.
6      Devuelve: la concatenación de las listas de xss (es decir: una lista
7        con los enteros de las listas de xss, en el orden en que aparecen)
8      '''
9      res:List[int] = []
10     i:int = 0
11     while i < len(xss):
12         concatenar(res, xss[i])
13         i = i + 1
14     return res
15
16 def concatenar(xs:List[int], ys:List[int]):
17     '''
18     Requiere: nada. Llamamos XS al valor inicial de xs.
19     Devuelve: nada.
20     Modifica: deja en xs los elementos de XS seguidos de los elementos de ys.
21     '''
22     i:int = 0
23     # (A)
24     while i < len(ys):
25         # (B)
26         xs.append(ys[i])
27         i = i + 1
28         # (C)
29     # (D)
```

Las demostraciones de terminación y correctitud de esta versión de `aplanar` son idénticas a las de la versión anterior. En este caso, como definimos una función auxiliar `concatenar`, debemos también hacer las demostraciones correspondientes a esa función.

2.1. Terminación del ciclo de `concatenar`

Esta demostración es igual a la de la función `aplanar`, reemplazando `xss` por `ys`.

2.2. Correctitud del ciclo de `concatenar`

Consideremos el siguiente predicado invariante $\mathcal{I}$ para el ciclo de `concatenar`: **$0 \leq$ `i` $\leq$ `len(ys)`**, y **`xs` tiene los elementos de `XS` seguidos de los primeros `i` elementos de `ys`**. Sigamos este razonamiento:

- $\mathcal{I}$ es cierto en el punto (A), donde `i` vale 0 y `xs` vale `XS` (el valor que tenía `xs` al comienzo), seguidos de 0 elementos de `ys`.
- Supongamos que $\mathcal{I}$ es cierto en el punto (B). Si llamamos ϕ al valor que tiene `i` en ese momento, las siguientes afirmaciones son verdaderas:

  $$\texttt{i} = \phi$$

  $$\texttt{xs} = \texttt{XS} + \texttt{[ys[0], ..., ys[i-1]]} = \texttt{XS} + \texttt{[ys[0], ..., ys[}\phi\texttt{-1]]}$$

  En el cuerpo del ciclo, se agrega a `xs` el valor de `ys[i]` (es decir, `ys[`ϕ`]`). Después, `i` se incrementa en 1. En consecuencia, al terminar el cuerpo del ciclo, en el punto (C), valen las siguientes afirmaciones:

  $$\texttt{i} = \phi + 1$$

$$\texttt{xs} = \texttt{XS} + \texttt{[ys[0], ..., ys[}\phi\texttt{-1], ys[}\phi\texttt{]]} = \texttt{XS} + \texttt{[ys[0], ..., ys[i-1]]}$$

En otras palabras, $\mathcal{I}$ vuelve a ser cierto al terminar el cuerpo del ciclo, en el punto (C).

- De esta manera, sabemos que en cada iteración del ciclo, $\mathcal{I}$ empieza y termina siendo cierto.
- Cuando el ciclo termina porque la condición resulta falsa, es fácil ver que $\mathcal{I}$ sigue siendo cierto al llegar al punto (D), y también que `i` vale `len(ys)`. Entonces, en este momento podemos afirmar que **`xs` tiene los elementos de `XS` seguidos de los primeros `len(ys)` elementos de `ys`**. En otras palabras: **`xs` tiene los elementos de `XS` seguidos de los elementos de `ys`**, que es lo que indica la especificación de la función `concatenar`. □