Tecnología Digital 1: Introducción a la Programación – TP1 – Parte B

Autores: Sosa, Luisina
Condorpocco, Vanina
Texeira, Milagros

Función simBP

Terminación del ciclo:
Para demostrar que termina, debemos entender cuándo termina el ciclo, según la guarda. En este caso:

“i<longitud_binario_n and i<longitud_binario_m”

Por lo tanto, nuestro ciclo dejará de ejecutarse en tanto nuestra variable **i** sea igual o más grande que **longitud_binario_n** o **longitud_binario_m**, esto por lo que nos dice la regla del and que si una de las condiciones es false toda la condición conjunta es false. Para ver que esto sucede, podemos seguir la siguiente línea de pensamiento:

- Nuestra variable **i** se inicializa en 0.
- Como en el *Requiere* nos especifican que **n>0** y **m>0**, **longitud_binario_n** y **longitud_binario_m** siempre serán mayores a 0, por lo tanto el ciclo siempre se va a ejecutar al menos una vez. Veamos qué pasa al ejecutarse el ciclo.
- La variable **i** se va incrementando en 1 cada vez que el ciclo se corre. Además, hay un caso donde **i** vale más que **longitud_binario_n** y es cuando no se cumple la condición **if(num_binario_n[i]==num_binario_m[i])** esto hace que **i** valga **longitud_binario_n + 1** al llegar al final del ciclo. Cuando esto suceda, **i** no es menor a **longitud_binario_n** por lo que la condición para que se ejecute el ciclo (**i<longitud_binario_n and i<longitud_binario_m**) será false, ocasionando que este termine. Veamos qué pasa si esa condición **if** siempre se cumple.
- Las variables **longitud_binario_n** y **longitud_binario_m** no se modifican mientras el ciclo se ejecuta, por lo que son constantes.
- Por lo tanto, es inevitable que en alguna iteración, i alcance el valor de **longitud_binario_n** o **longitud_binario_m.**
- Cuando esto sucede, **i** no es menor a **longitud_binario_n o longitud_binario_m**, por lo que la condición (**i<longitud_binario_n and i<longitud_binario_m**) cuando se ejecute el ciclo será false, ocasionando que este termine.

Predicado Invariante
Para cualquier valor de n y m (siguiendo la especificación) :

Inv**:** res vale la longitud del prefijo común más largo que hay en los primeros i caracteres entre las representaciones binarias de n (num_binario_n) y m (num_binario_m).

Correctitud:
Este predicado, vale para cada iteración del ciclo, por lo que podemos sacar la siguiente conclusión:

El ciclo termina cuando i >= longitud_binario_n o si i>= longitud_binario_m.

Por lo tanto, podemos reemplazar i por longitud_binario_n o longitud_binario_m en nuestro invariante.

Si hacemos esto, nuestro invariante nos asegura que “res vale la longitud del prefijo común más largo que hay en los primeros longitud_binario_n o longitud_binario_m caracteres entre las representaciones binarias de n (num_binario_n) y m (num_binario_m)”.

Pero, decir la longitud del prefijo común más largo que hay en los primeros longitud_binario_n o longitud_binario_m caracteres entre las representaciones binarias de n (num_binario_n) y m (num_binario_m) es lo mismo que decir, la similitud binaria de prefijo entre n y m.

Entonces: al terminar el ciclo, “res vale la similitud binaria de prefijo entre n y m”.
Y esa es la conclusión a la que queríamos llegar.

Función cantidad_con_simBP_en_intervalo

Terminación del ciclo:
Para demostrar que termina, debemos entender cuándo termina el ciclo, según la guarda.En este caso:

“i<=diferencia_ab”

Por lo tanto, nuestro ciclo dejará de ejecutarse en tanto nuestra variable **i** sea más grande que **diferencia_ab**. Para ver que esto sucede, podemos seguir la siguiente línea de pensamiento:

- Nuestra variable **i** se inicializa en 0.
- Por la cláusula *Requiere* de la especificación, sabemos que a>0, b>0 y a<=b, por lo cual **diferencia_ab** es si o si >=0, es decir, el ciclo por lo menos se ejecuta una vez. Veamos qué pasa al ejecutarse el ciclo.
- Hay que hacer una mención especial a que dentro del ciclo usamos una función auxiliar llamada simBP() que contiene un ciclo, aclarado, continuamos.
- La variable **i** se va incrementando en 1 cada vez que el ciclo se corre.
- La variable **diferencia_ab** no se modifica mientras el ciclo se ejecuta, por lo que es una constante.
- Por lo tanto, es inevitable que en alguna iteración, **i** alcance un valor mayor a **diferencia_ab**.
- Cuando esto suceda, **i** no es menor o igual a **longitud_mayor_ab**, por lo que la condición para que se ejecute el ciclo (**i<=diferencia_ab**) será false, ocasionando que este termine.

Predicado Invariante
Para cualquier valor de n,k,a,b (siguiendo la especificación):

Inv**:** res vale la cantidad de números entre b y b-i (inclusive) cuya simBP con n es igual a k.

Correctitud:
Este predicado, vale para cada iteración del ciclo, por lo que podemos sacar la siguiente conclusión.

El ciclo termina cuando i > diferencia_ab es decir, i=diferencia_ab +1.

Por lo tanto, podemos reemplazar i por diferencia_ab+1 en nuestro invariante, ya que en algún momento llegaremos a esa iteración y el invariante vale para cualquier iteración.

Si hacemos esto, nuestro invariante nos asegura que “res vale la cantidad de números entre b y b-diferencia_ab+1 (inclusive) cuya simBP con n es igual a k”.

Pero decir la cantidad de números entre b y b-diferencia_ab+1 (inclusive) cuya simBP con n es igual a k es lo mismo que decir la cantidad de números entre b y a (inclusive) cuya simBP con n es igual a k, porque diferencia_ab=b-a.

Entonces: al terminar el ciclo, “res vale la cantidad de números entre b y a (inclusive) cuya simBP con n es igual a k”.
Y esa es la conclusión a la que queríamos llegar.

Función existe_con_simBP_en_intervalo

En la función **existe_con_simBP_en_intervalo** no hay ciclos; se llama a una función auxiliar llamada cantidad_con_simBP_en_intervalo, pidiendo que n, a, b, k sean siempre distintos de cero y a<=b. Esta (función auxiliar) contiene un ciclo; la función fue justificada anteriormente.

La correctitud y terminación de esta función son las de la función auxiliar cantidad_con_simBP_en_intervalo; entonces no sería necesario definir un predicado invariante ni terminación para la función **exite_simBP_en_intervalo** ya que está en ningún momento tiene alguna iteración; todo se lo deja a la otra función.

Función numero_con_mayor_simBP_en_intervalo

Terminación del ciclo:
Para demostrar que termina, debemos entender cuándo termina el ciclo, según la guarda. En este caso:

“i<=diferencia_ab”

Por lo tanto, nuestro ciclo dejará de ejecutarse en tanto nuestra variable **i** sea más grande que **diferencia_ab**. Para ver que esto sucede, podemos seguir la siguiente línea de pensamiento:

- Nuestra variable **i** se inicializa en 0.
- Por la cláusula *Requiere* de la especificación, sabemos que a>0, b>0 y a<=b, por lo cual **diferencia_ab** es sí o sí >=0, es decir, el ciclo por lo menos se ejecuta una vez. Veamos qué pasa al ejecutarse el ciclo.
- Hay que hacer una mención especial a que dentro del ciclo usamos una función auxiliar llamada simBP() que contiene un ciclo, aclarado, continuamos.
- La variable **i** se va incrementando en 1 cada vez que el ciclo se corre.
- La variable **diferencia_ab** no se modifica mientras el ciclo se ejecuta, por lo que es una constante.
- Por lo tanto, es inevitable que en alguna iteración, **i** alcance un valor mayor a **diferencia_ab**.
- Cuando esto suceda, **i** no es menor o igual a **diferencia_ab**, por lo que la condición para que se ejecute el ciclo (**i<=diferencia_ab**) será false, ocasionando que este termine.

Predicado Invariante
Para cualquier valor de n,a,b (siguiendo la especificación):

Inv**:** res vale el menor número entre los primeros i números recorridos desde b hacia a (inclusive) con mayor simBP con n.

Correctitud:
Este predicado, vale para cada iteración del ciclo, por lo que podemos sacar la siguiente conclusión:

El ciclo termina cuando i > diferencia_ab, es decir, i=diferencia_ab +1.

Por lo tanto podemos reemplazar i por diferencia_ab+1 en nuestro invariante.

Si hacemos esto, nuestro invariante nos asegura que “res vale el menor número entre los primeros diferencia_ab + 1 números recorridos desde b hacia a (inclusive) con mayor simBP con n”.

Pero decir el menor número entre los primeros diferencia_ab +1 números recorridos desde b hacia a (inclusive) con mayor simBP con n es lo mismo que decir el menor número entre a y b (inclusive) con mayor simBP con n.

Entonces: al terminar el ciclo, “res vale el menor número entre a y b (inclusive) con mayor simBP con n”.
Y esa es la conclusión a la que queríamos llegar.