Argumentación de corrección de programas

Juan Francisco Díaz Frias

Mayo 2022

1. Argumentando sobre corrección de programas recursivos

Sea $f : A \to B$ una función, y A un conjunto definido recursivamente (recordar definición de matemáticas discretas I), como por ejemplo los naturales o las listas.

Sea P_f un programa recursivo (lineal o en árbol) desarrollado en Scala (o en cualquier lenguaje de programación) hecho para calcular f :

```
def Pf(a:A):B = {                    // Pf recibe a de tipo A, y devuelve f(a) de tipo B
...
}
```

¿Cómo argumentar que $P_f(a)$ siempre devuelve $f(a)$ como respuesta? Es decir, ¿Cómo argumentar que P_f es correcto con respecto a su especificación?

La respuesta es sencilla, demostrando el siguiente teorema:

$$\forall a \in A : P_f(a) == f(a)$$

Si recuerdan las matemáticas discretas, cuando uno tiene que demostrar que algo se cumple para todos los elementos de un conjunto definido recursivamente (como en este caso A), es natural usar **inducción estructural**. En términos prácticos esto significa demostrar que:

- Para cada valor básico a de A, se tiene que:

  $$P_f(a) == f(a)$$

- Para cada valor $a \in A$ construido recursivamente a partir de otro(s) valor(es) $a' \in A$ se tiene que

  $$P_f(a') == f(a') \to P_f(a) == f(a)$$

  Nótese que $P_f(a') == f(a')$ es lo que llamamos la hipótesis de inducción.

2. Ejemplo: Factorial Recursivo

Por ejemplo, sea $f : \mathbb{N} \to \mathbb{N}$ la función que calcula el factorial de un número natural, es decir, $f(n) = n!$.

Y sea P_f el siguiente programa en Scala:

```
def Pf(n:Int):Int = {                    // Pf recibe  n de tipo Int, y devuelve n! de tipo Int
   if (n==0) 1 else n*Pf(n-1)
}
```

Vamos a demostrar que

$$\forall n \in \mathbb{N} : Pf(n) == n!$$

- Caso base: $n = 0$ Vamos a calcular qué devuelve Pf usando el modelo de substitución:

  $$Pf(0) \to if\ (0 == 0) 1\ else\ 0 * Pf(0-1) \to 1$$

  Por otro lado, $f(0) = 0! = 1$.

  Podemos concluir que

  $$Pf(0) == f(0)$$

- Caso de inducción: $n = k + 1, k \geq 0$. Hay que demostrar:

  $$Pf(k) == f(k) \to Pf(k+1) == f(k+1)$$

  Empecemos por calcular qué devuelve Pf usando el modelo de substitución:

  $$Pf(k{+}1) \to if\ (k{+}1 == 0) 1\ else\ (k{+}1){*}Pf((k{+}1){-}1) \to (k{+}1){*}Pf(k) \overbrace{\to}^{HI} (k{+}1){*}k! = (k{+}1)!$$

  Por otro lado, $f(k+1) = (k+1)!$.

  Podemos concluir que

  $$Pf(k+1) == f(k+1)$$

- Podemos concluir entonces por inducción que:

  $$\forall n \in \mathbb{N} : Pf(n) == n!$$

Para reflexionar: ¿Qué parte de la prueba no funciona si en Pf en lugar de $(n == 0)$ ponemos $(n == 1)$?

¿Qué parte de la prueba no funciona si en Pf en lugar de $n * Pf(n-1)$ ponemos $(n+1) * Pf(n)$?

3. Argumentando sobre corrección de programas iterativos

En el ejemplo anterior, el programa Pf implementaba un proceso recursivo lineal. De igual manera se hace si el proceso es recursivo de árbol. Pero, si el proceso es iterativo, ¿cómo se demuestra corrección? En ese caso la argumentación es diferente. Lo que se debe formalizar es cómo es la iteración, es decir:

- Cómo se representa un estado de la iteración s
- Cuál es el estado inicial, s_0
- Cuál es el estado final (o cómo se reconoce que un estado es final): s_f
- Qué condición (o predicado) cumple todo estado: $Inv(s)$ (a esa condición se le llama una condición **invariante** porque no importa cuál sea el estado de la iteración, esa condición se cumple).
- Cuál es el mecanismo para pasar de un estado al siguiente: $transformar(s)$. Si s_i es el estado i, $transformar(s_i) = s_{i+1}$

En este caso el programa iterativo tiene la siguiente forma:

```
def Pf(a:A):B = {                    // Pf recibe  a de tipo A, y devuelve f(a) de tipo B
   def Pfiter(s:Estado): B
      if (esFinal(Estado))  respuesta(s)  else Pfiter(transformar(s))
   Pfiter(s0)
}
```

Como $Pf(a) \rightarrow Pfiter(s0)$ lo que hay que demostrar es que

$$Pfiter(s0) == f(a)$$

Para ello se demuestra lo siguiente:

1. $Inv(s_0)$: el estado inicial cumple la condición invariante.
2. $(s_i \neq s_f \wedge Inv(s_i)) \rightarrow Inv(transformar(s_i))$: el nuevo estado cumple la condición invariante, si el anterior estado la cumplía.
3. Nótese que de los dos anteriores, se puede concluir $Inv(s_f)$, es decir, el estado final cumple la condición invariante. En este paso lo que vamos a demostrar es

$$Inv(s_f) \rightarrow respuesta(s_f) == f(a)$$

4. Por último, hay que demostrar que siempre se llega al estado final s_f. Esto implica que

$$Pf(a) == Pfiter(s0) == respuesta(s_f) == f(a)$$

4. Ejemplo: Factorial Iterativo

Considere el siguiente programa iterativo en Scala para calcular la función factorial:

```
def Pf(n:Int):Int = {                // Pf recibe  n de tipo Int, y devuelve n! de tipo Int
   def Pfiter(i:Int, n:Int, ac:Int): Int
      if (i>n) ac else Pfiter(i+1, n, i*ac)
   Pfiter(1,n, 1)
}
```

Este programa implementa el siguiente proceso iterativo:

- Un estado $s = (i, n, ac)$
- El estado inicial es $s_0 = (1, n, 1)$
- (i, n, ac) es final si $i > n$ o lo que es lo mismo si $i == n + 1$
- $Inv(i, n, ac) \equiv i \leq n + 1 \wedge ac = (i - 1)!$
- $transformar((i, n, ac)) = (i + 1, n, i * ac)$

Ahora vamos a demostrar los 4 puntos enunciados en la sección anterior:

1. $Inv(s_0)$: el estado inicial cumple la condición invariante. Como $s_0 = (1, n, 1)$ y $1 \leq n + 1 \wedge 1 = (0)!$ podemos concluir que $Inv(s_0)$.

2. $(s_i \neq s_f \wedge Inv(s_i)) \rightarrow Inv(transformar(s_i))$: el nuevo estado cumple la condición invariante, si el anterior estado la cumplía.

$$(s_i \neq s_f \wedge Inv(s_i)) \rightarrow Inv(transformar(s_i))$$

$$\equiv$$

$$(i \neq (n + 1) \wedge i \leq n + 1 \wedge ac = (i - 1)!) \rightarrow i + 1 \leq n + 1 \wedge i * ac = ((i + 1) - 1)!$$

Nótese que

$$(i \neq (n + 1) \wedge i \leq n + 1) \rightarrow i + 1 \leq n + 1$$

y que

$$ac = (i - 1)! \rightarrow i * ac = i * (i - 1)! \rightarrow i * ac = i! \rightarrow i * ac = ((i + 1) - 1)!$$

Por tanto,

$$(s_i \neq s_f \wedge Inv(s_i)) \rightarrow Inv(transformar(s_i))$$

3.

$$Inv(s_f) \rightarrow respuesta(s_f) == f(a)$$

$$\equiv$$

$$Inv((n+1, n, ac) \rightarrow ac == n!$$

$$\equiv$$

$$((n+1 \leq n+1) \wedge ac = ((n+1)-1)!) \rightarrow ac == n!$$

$$\equiv$$

$$(true \wedge ac = n!) \rightarrow ac == n!$$

$$\equiv$$

$$ac = n! \rightarrow ac == n!$$

$$\equiv$$

$$true$$

4. En cada paso la componente i del estado es uno más. Por tanto, está cada vez más cerca de $n+1$. En consecuencia, después de n iteraciones llega a $n+1$.

   Esto implica que

$$Pf(n) == Pfiter(1, n, 1) == ac == n!$$