Programación 2

Lección 13. Aplicación al análisis de la corrección de algoritmos recursivos

1. **Prueba formal de la corrección de un algoritmo recursivo**
2. **Prueba formal de la corrección de la función `factorial(n)`**
3. **Prueba formal de la corrección de la función `factorial(n,resultado)`**
4. **Prueba formal de la corrección de la función `raiz(n)`**

1. Prueba formal de la corrección de un algoritmo recursivo

```
// Pre: P
// Post: Q
tipoDato nombreFunción (lista_de_parámetros) {
   if (C1) {
      A1
   }
   else if (C2) {
      A2
   }
   …
   else if (Cp) {
      Ap
   }
   else {
      Aelse
   }
}
```

1. Se satisfacen las condiciones de dominio de $C_1, C_2, \ldots, C_p$

```
// Pre: P
// Post: Q
tipoDato nombreFunción (lista_de_parámetros) {
   // P  ⇒  Dom(C1)
   if (C1) { A1 }
   // P ∧ ¬C1   ⇒  Dom(C2)
   else if (C2) { A2 }
   ...
   // P ∧ ¬C1 ∧ ... ∧ ¬Cp-1  ⇒  Dom(Cp)
   else if (Cp) { Ap }
   else { Aelse }
}
```

2. Son correctas las acciones A_1, A_2, … , A_p , A_{else}

```
// Pre: P
// Post: Q
tipoDato nombreFunción (lista_de_parámetros) {
   if (C1) {   // P ∧ C1 ⇒ pmd(A1,Q)
      A1
   }
   else if (C2) {   // P ∧ ¬C1 ∧ C2 ⇒ pmd(A2,Q)
      A2
   }
   ...
   else if (Cp) {   // P ∧ ¬C1 ∧ ... ∧ ¬Cp-1 ∧ Cp ⇒ pmd(Ap,Q)
      Ap
   }
   else {   // P ∧ ¬ C1 ∧ ... ∧ ¬ Cp ⇒ pmd(Aelse,Q)
      Aelse
   }
}
```

3. Cualquier secuencia de invocaciones recursivas termina

```
// Pre: P
// Post: Q
tipoDato nombreFunción (lista_de_parámetros) {
   if (C1) { A1 }
   else if (C2) { A2 }
   ...
   else if (Cp) { Ap }
   else { Aelse }
}
```

Método de prueba:

1. Se define una función de cota
2. Se prueba que el valor X que toma la función de cota para la invocación `nombreFunción(p)` es mayor que el valor X' que toma para cualquier invocación recursiva `nombreFunción(p')`: X > X'
3. Se prueba que cualquier secuencia de valores de la función de cota definidos por cualquier sucesión de invocaciones recursivas es finita (por ej.: demostrando que toma valores en $\mathbb{Z}$ y está acotada inferiormente)

2. Prueba formal de la corrección de la función `factorial(n)`

Diseño recursivo de la función `factorial(n)`

```
// Pre: n ≥ 0
// Post: factorial(n) = (Πα∈[1,n]. α)
int factorial (const int n) {
   if (n == 0) {
       return 1;
   }
   else {
       return n * factorial(n-1);
   }
}
```

Demostración de la corrección (tres pruebas , falta probar su terminación):

```
// Pre: n ≥ 0
// Post: factorial(n) = (Πα∈[1,n]. α)
int factorial (const int n) {
   // Pre: n ≥ 0
   // n ≥ 0  ⇒  Dom(n == 0) ≡ cierto                 1º
   if (n == 0) {
       // n = 0  ⇒  1 = (Πα∈[1,n]. α)                2º
       return 1;
   }
   else {
       // n > 0  ⇒  n - 1 ≥ 0
       // n > 0 ∧ factorial(n-1) = (Πα∈[1,n-1]. α)
       // ⇒
       // n * factorial(n-1) = (Πα∈[1,n]. α)         3º
       return n * factorial(n-1);
   }
   // Post: factorial(n) = (Πα∈[1,n]. α)
}
```

Demostración de la terminación de la recursión:

```
// Pre: n ≥ 0
// Post: factorial(n) = (Πα∈[1,n]. α)
int factorial (const int n) {  // f_cota = n
   // n ≥ 0                      // f_cota(inicial) = n        1º
   if (n == 0) { return 1; }
   else { // n > 0
       return n * factorial(n-1);
       // f_cota(invocación_recursiva) = n - 1                  2º
   }
}
```

Prueba formal de la terminación:

1. La función de cota toma valores <u>decrecientes</u>:

   $f_{cota(inicial)} > f_{cota(invocación\_recursiva)}$ ya que **n > n-1**

2. Cualquier secuencia de invocaciones recursivas es finita ya que el valor de la función de cota <u>está acotado inferiormente en Z</u> puesto que, al producirse la invocación recursiva `factorial(n-1)`, se satisface que $n - 1 \geq 0$ y, en consecuencia:

   $f_{cota}(n-1) = n - 1 \geq 0$

3. Prueba formal de la corrección de la función `factorial(n,resultado)`

Diseño recursivo de la función `factorial(n,resultado)`

```
// Pre: n ≥ 0
// Post: resultado = (Πα∈[1,n]. α)
void factorial (const int n, int& resultado) {
   if (n == 0) {
      resultado = 1;
   }
   else {
      factorial(n-1, resultado);
      resultado = n * resultado;
   }
}
```

Demostración de la corrección (tres pruebas, falta probar la terminación):

```
// Pre: n ≥ 0
// Post: resultado = (Πα∈[1,n]. α)
void factorial (const int n , int& resultado) {
   // n ≥ 0
   // n ≥ 0 ⇒ Dom(n == 0) ≡ cierto                 1º
   if (n == 0) {
       // n = 0 ⇒ 1 = (Πα∈[1,n]. α)               2º
       resultado = 1;
   }
   else {
       // n > 0 ⇒ n - 1 ≥ 0
       factorial(n-1, resultado);
       // n > 0 ∧ resultado = (Πα∈[1,n-1]. α)
       // ⇒
       // n * resultado = (Πα∈[1,n]. α)             3º
       resultado = n * resultado;
   }
   // resultado = (Πα∈[1,n]. α)
}
```

Demostración de la terminación de la recursión:

```
// Pre: n ≥ 0
// Post: resultado = (Πα∈[1,n]. α)
void factorial (const int n, int& resultado) {  // f_cota = n
   // n ≥ 0                                  // f_cota(inicial) = n        1º
   if (n == 0) { resultado = 1; }
   else {  // n > 0
       factorial(n-1, resultado);
       // f_cota(invocación_recursiva) = n - 1                             2º
       resultado = n * resultado;
   }
}
```

Prueba formal de la terminación:

1. La función de cota toma valores <u>decrecientes</u>:

   $f_{cota(inicial)} > f_{cota(invocación\_recursiva)}$ ya que **n > n-1**

2. Cualquier secuencia de invocaciones recursivas es finita ya que el valor de la función de cota <u>está acotado inferiormente en *Z*</u> puesto que, al producirse la invocación `factorial(n-1,resultado)` se satisface que `n > 0` y, en consecuencia:

   $f_{cota}(n-1) = n - 1 \geq 0$

4. Prueba formal de la corrección de la función raiz(n)

```
// Pre: n ≥ 0
// Post: raiz(n)² ≤ n ∧ (raiz(n) + 1)² > n
int raiz (const int n) {
   return raiz(n,0);
}

// Pre: n ≥ 0 ∧ cand² ≤ n
// Post: raiz(n,cand)² ≤ n ∧ (raiz(n,cand) + 1)² > n
int raiz (const int n, const int cand) {
   if ((cand + 1)*(cand + 1)) > n) {
      return cand;
   }
   else {
      return raiz(n, cand + 1);
   }
}
```

Corrección de la función **raiz(n)**

```
// Pre: n ≥ 0
// Post: raiz(n)² ≤ n ∧ (raiz(n) + 1)² > n
int raiz (const int n) {
   // n ≥ 0 ⇒ n ≥ 0 ∧ 0² ≤ n
   // n ≥ 0 ∧ raiz(n,0)² ≤ n ∧ (raiz(n,0) + 1)² > n
   // ⇒
   // raiz(n,0)² ≤ n ∧ (raiz(n,0) + 1)² > n          1º
   return raiz(n,0);
   // raiz(n)² ≤ n ∧ (raiz(n) + 1)² > n
}

// Pre: n ≥ 0 ∧ cand² ≤ n
// Post: raiz(n,cand)² ≤ n ∧ (raiz(n,cand) + 1)² > n
int raiz (const int n, const int cand);
```

Corrección de **raiz(n,cand)** (tres pruebas, falta terminación):

```
// Pre: n ≥ 0 ∧ cand² ≤ n
// Post: raiz(n,cand)² ≤ n ∧ (raiz(n,cand) + 1)² > n
int raiz (const int n, const int cand) {
   // n ≥ 0 ∧ cand² ≤ n ⇒ Dom((cand+1)² > n) ≡ cierto                 ✓
   if ((cand + 1)*(cand + 1)) > n)
      // n ≥ 0 ∧ cand² ≤ n ∧ (cand + 1)² > n
      // ⇒ cand² ≤ n ∧ (cand + 1)² > n                    2º         ✓
      return cand;
   else
      // n ≥ 0 ∧ cand² ≤ n ∧ (cand+1)² ≤ n
      // ⇒ n ≥ 0 ∧ (cand+1)² ≤ n
      // n ≥ 0 ∧ cand² ≤ n ∧ (cand+1)² ≤ n ∧
      // raiz(n,cand+1)² ≤ n ∧ (raiz(n,cand+1) + 1)² > n
      // ⇒ raiz(n,cand+1)² ≤ n ∧ (raiz(n,cand+1)+1)² > n   3º       ✓
      return raiz(n,cand + 1);
   // raiz(n,cand)² ≤ n ∧ (raiz(n,cand) + 1)² > n
}
```

Demostración de la terminación:

```
// Pre: n ≥ 0 ∧ cand² ≤ n
// Post: raiz(n,cand)² ≤ n ∧ (raiz(n,cand) + 1)² > n
int raiz (const int n, const int cand) {  // f_cota = n - cand²
    // n ≥ 0 ∧ cand² ≤ n          // f_cota(inicial) = n - cand²      4º
    if ((cand+1)*(cand+1)) > n) { return cand; }
    else {
        return raiz(n,cand + 1);
        // f_cota(invoc._recursiva) = n – (cand+1)²                  5º
    }
}
```

Prueba formal de la terminación:

1. La función de cota toma valores <u>decrecientes</u>:

   $f_{cota(inicial)} > f_{cota(invoc.\_recursiva)}$ ya que $n - cand^2 > n - (cand+1)^2$

2. Cualquier secuencia de invocaciones recursivas es finita ya que el valor de la función de cota <u>toma valores en el conjunto de los enteros</u> y <u>está limitada inferiormente</u>:

   $n \geq 0 \wedge cand^2 \leq n \quad \rightarrow \quad f_{cota} = n - cand^2 \geq 0$