Aplicaciones de la Recursividad a la Programación

La Programación como Composición

- Un programa consta de *sentencias ejecutables*, *declaraciones*, *comentarios*, etc.
- Las *sentencias ejecutables* alteran el valor de variables – **El estado del programa**.
- Una *sentencia ejecutable* es una función: Toma el estado inicial y lo transforma en un estado final (si se excluyen loops sin fin y excepciones).

Ejemplos

- Sean las sentencias:

$$x = 0;$$

$$y = x+2;$$

- Identificar el estado del programa en cada paso

```
% Ejemplo Prolog
?- X is 0, Y is X+2.
X = 0,
Y = 2.
```

```
# Ejemplo Python
>>> x=0
>>> y=x+2
>>> x
0
>>> y
2
>>>
```

Sentencias como composiciones

- Llamemos x al estado inicial del programa (x representa los valores de todas las variables en el programa).
- La ejecución de una primera sentencia modifica el estado produciendo A(x).
- La ejecución de una segunda sentencia B, toma como entrada el estado y, resultado de la ejecución de A, por lo tanto obtiene B(A(x))
- También se representa

$$A; B$$

$$B \circ A$$

Ejemplos

- Sean las sentencias:

  a = a+3; a=a*2;

- Calcular
  - (a=a+3)(x)
  - (a=a*2)(y)
  - (a=a+3; a=a*2)(x)

```
# Ejemplo Python
>>> f1 = lambda a: a+3
>>> f2 = lambda a: a*2
>>> x=1
>>> f2(f1(x))
8
```

Notas sobre la composición

- Si se tiene la secuencia de sentencias ejecutables A; B; C;, el resultado es C(B(A(x))))
- Observemos que agrupaciones de sentencias, tales como (A; B); C; ó A; (B; C); producen el mismo resultado.

La composición de funciones se comporta como un operador asociativo

No son necesarios los paréntesis

$$C \circ B \circ A = (C \circ B) \circ A = C \circ (B \circ A)$$

La función identidad

- No altera el estado del programa
- Se representa

$$I(x) = x$$

```
% Ejemplo Prolog
?- X is 0, write(X), Y is X+2, tab(5), write(Y).
0     2
X = 0,
Y = 2.
```

```
# Ejemplo Python
>>> x=0; print(x); y=x+2; print(y)
0
2
```

Sentencias condicionales como funciones

La sentencia

```
if g(x) then A;
else B;
```

se representa mediante la función

H = (if g(x) then A else B)

```
% Ejemplo Prolog
% par_impar/1 : Imprime en pantalla si un entero es par o impar
par_impar(X) :- X mod 2 =:= 0, !, write("Es par"); write("Es impar").
```

```
# Ejemplo Python
>>> x=3
>>> print("Es par") if x % 2 == 0 else print("Es impar")
Es impar
```

Ciclos como funciones recursivas

El ciclo

```
while g(x) A;
```

Se puede representar como la función

$$W = (\text{while } g(x)\ A)$$

o haciendo explícita la recursión

$$W(x) = (\text{if } g(x) \text{ then } (A; W)(x))$$

En general todo programa puede traducirse como la composición de un conjunto de funciones primitivas

Ejemplo:
Ciclos como funciones recursivas

```
% Ejemplos Prolog

% Ciclo descendiente de M hasta 0
cicloDescendiente(M) :- M=0, write(M), !.
cicloDescendiente(M) :- write(M), NuevaM is M-1, cicloDescendiente(NuevaM).

% Ciclo ascendiente desde A hasta B
cicloAscendiente(A,B) :- cicloAscendiente(A,B,A).
cicloAscendiente(A,B,I) :- I>B, !.
cicloAscendiente(A,B,I) :- write(I), nl, J is I+1, cicloAscendiente(A,B,J).

% Sucesion aritmetica con terminos constantes A,R. Imprimir hasta el N-esimo termino
serieAritmetica(A,R,N) :- serieAritmetica(A,R,N,0).
serieAritmetica(A,R,N,I) :- I>=N, !.
serieAritmetica(A,R,N,I) :- T is A+R*I, write(T), nl, J is I+1, serieAritmetica(A,R,N,J).
```

Ejercicios:

1.Obtener la serie de los n primeros términos

2.Hacer la sucesión geométrica con constantes A,R e imprimir los primeros N términos

Ejemplo:
Ciclos como funciones iterativas

```
# Ejemplos Python

# Ciclo ascendiente desde A hasta B
>>> a=3; b=10
>>> for i in range(a,b+1):
...     print(i)
...

# Ciclo descendiente desde M hasta 0
>>> m=7
>>> for i in range(m,-1,-1):
...     print(i)
...

# Sucesion aritmetica con terminos constantes A,R. Imprimir hasta el N-esimo termino
>>> a=2; r=3; n=10
>>> for i in range(n):
...     t = a+r*i
...     print(t)
```

Ejercicios:
Los mismos pero en Python

Recursividad en Programación

- Similar a las definiciones recursivas, los procedimientos recursivos se invocan a si mismos, salvo para los casos base.
- Un procedimiento recursivo debe resolver uno o más casos base de forma no recursiva y en los demás casos hacer uso de la llamada recursiva.
- La técnica de demostración por recursividad nos permite probar el correcto funcionamiento de estos procedimientos.

Ejemplos de funciones definidas recursivamente

```
% Ejemplo Prolog

% factorial/2: Calcula el factorial de N
% arg1 : N
% arg2 : resultado N!
factorial(0,1) :- !.
factorial(N,X) :- N>0, NN is N-1, factorial(NN,Y), X is Y*N.
```

```
# Ejemplo Python, version recursiva
def factorial(n):
    if n==0:
        return 1
    else:
        if n>0:
            return n*factorial(n-1)
        else:
            print("Valor de N no valido")

print(factorial(3))
print(factorial(5))
```

```
# Ejemplo Python, version iterativa
def factorialIterativo(n):
    producto = 1
    for i in range(1,n+1):
        producto *= i
    return producto

print(factorialIterativo(3))
print(factorialIterativo(5))
```

Mas ejercicios

- Obtener los factoriales de los enteros entre M y 0 (en orden descendiente).
- Obtener los factoriales de los enteros desde A hasta B (en orden ascendente).
- Obtener la suma de los factoriales desde A hasta B.

Aserciones

- Son predicados sobre el estado del programa.
- Se utilizan en la definición de especificaciones de software:
  - Pre-condiciones
  - Post-condiciones
- Ejemplos

```
Java:    assert(<expresión booleana>);
Python:  assert <expresión booleana>
Node.js: assert(<valor>[, mensaje]);
```

Ejemplos
Verificar tipo de datos

```
% Ejemplos Prolog

% Predicados para verificar el tipo de dato
?- integer(5), float(3.333), number(7.456).
true.

?- atom(hola), \+atom(123), \+atom("hola"), \+atom([1,2,3]).
true.

?- string("hola"), \+string('hola'), \+string(123).
true.
```

```
# Ejemplo Python

>>> a=123
>>> b=4.333
>>> c="Hola mundo"
>>> type(a)
<class 'int'>
>>> type(b)
<class 'float'>
>>> type(c)
<class 'str'>
>>> type(a) == int
True
>>> type(b) == float
True
>>> type(c) == str
True
```

Invariantes

- Predicados en el estado del programa que deben ser verdaderos para todos los estados válidos de su ejecución.

Pre-condiciones y Post-condiciones

- Una aserción sobre el estado inicial de un segmento de código se llama una **pre-condición**.
- Una aserción sobre el estado final de un segmento de código se llama una **post-condición**.

Especificaciones en Ing. de Software

- Las pre-condiciones establecen las condiciones que se deben cumplir antes del inicio del programa (o función o método).
- Las post-condiciones establecen las condiciones que deben cumplirse una vez concluye el programa.
- El diseño e implementación del programa consisten en plantear y codificar un algoritmo que dadas las pre-condiciones, siempre satisfaga las post-condiciones.

Ejemplo

Especificaciones para un programa para obtener las raíces de una ecuación cuadrática:

$$a*X^2 + b*X + c = 0$$

- Precondiciones:
  - Se tienen como entradas los valores reales a,b,c
  - $b^2-4*a*c >= 0$
- Postcondiciones:
  - Se obtienen dos raíces reales X_1 y X_2.
  - $a*X_1^2 + b*X_1 + c = 0$
  - $a*X_2^2 + b*X_2 + c = 0$

Implementación de la solución

```
% Ejemplo Prolog

% cuadratica/5 : Encuentra las raices de la ecuacion cuadratica
% arg1 = A
% arg2 = B
% arg3 = C
% arg4 = X1
% arg5 = X2
cuadratica(A,B,C,X1,X2) :-
    B^2-4*A*C>=0,
    X1 is (-B+sqrt(B^2-4*A*C))/(2*A),
    X2 is (-B-sqrt(B^2-4*A*C))/(2*A).
```

Implementación en Python

```
# Ejemplo Python, version iterativa

import math

a = float(input("Ingrese a: "))
b = float(input("Ingrese b: "))
c = float(input("Ingrese c: "))

if b**2-4*a*c>=0:
    x1 = (-b+math.sqrt(b**2-4*a*c))/(2*a)
    x2 = (-b-math.sqrt(b**2-4*a*c))/(2*a)

    print('Primera raiz ', x1)
    print('Segunda raiz ', x2)

else:
    print('Sin raices reales')
```

Corrección parcial y total

- Un segmento de código es ***parcialmente correcto*** con respecto a la pre-condición P y la post-condición Q, si el estado final siempre satisface Q, siempre y cuando empiece en P y concluya.
- Un segmento de código es ***totalmente correcto*** si siempre que comienza en P termina y el estado final satisface Q.

Ejemplo: Función factorial

- Si se definen:
  - Pre-condición: n es natural
  - Post-condición: el valor retornado es n!
  - La función factorial es totalmente correcta.
- Si se definen:
  - Pre-condición: n es un entero
  - Post-condición: el valor retornado es n!
  - La función factorial es parcialmente correcta.
- Implementación:
  - La misma función factorial implementada en Java falla para n grande.

Y será totalmente correcta la implementación en Java?

```
public class Factorial {

    public static int factorial(int n) {
        int f = 1;
        for(int i=n; i>1; i--) f*=i;
        return f;
    }


    public static void main(String[] args) {
        int f1 = factorial(5);
        System.out.println("Factorial de 5: "+f1);
        int f2 = factorial(40);
        System.out.println("Factorial de 5: "+f2);
    }

}
```

Demostración de corrección por recursividad

- Si se demuestran los casos base y el paso inductivo, el segmento de código es parcialmente correcto.
- Si además se demuestra que el dominio es bien fundado, el programa es totalmente correcto.

Ejemplo

- Buscar un término en una lista
- Precondiciones:
  - Se tienen como entradas una lista y un término
- Postcondiciones:
  - Si el término está presente en la lista retorna verdadero, en caso contrario falso.

Implementación

```
% miembroDeLista/2: Determina si un término esta en la lista
% arg1: la lista
% arg2: término a buscar

miembroDeLista([H|_], H) :- !.
miembroDeLista([_|T], H) :- miembroDeLista(T,H).
```

Demostrar que miembroDeLista es totalmente correcto

1) Dominio bien fundado
2) Caso base
3) Hipótesis inductiva
4) Paso inductivo

Ejemplo: Búsqueda binaria

Dados:

- Un vector ordenado ascendentemente,
- un intervalo definidos por las posiciones primero-último donde se hace la búsqueda,
- una clave a buscar

Devolver

- encontrado = V/F si la clave fue encontrada en el vector
- posición, el índice del vector donde fue encontrada la clave.

Procedimiento recursivo

- Si primero>último, entonces encontrado=falso.
- El arreglo se parte en 2 mitades a partir del punto:

$$medio = (primero+ultimo) \div 2$$

- si vector[medio]<clave, buscar en el subintervalo derecho,
- si vector[medio]>clave, buscar en el subintervalo izquierdo,
- de lo contrario, el dato o está en medio o no está presente.

Pseudo-código

```
funcion busquedaBinaria(primero, ultimo, clave, vector)
if primero>ultimo,
   devolver falso
else
   medio ← (primero+ultimo)÷2
   if clave<vector[medio]
      devolver busquedaBinaria(primero, medio-1, clave, vector)
   else if clave>vector[medio]
      devolver busquedaBinaria(medio+1, ultimo, clave, vector)
   else
      posicion ← medio
      encontrado ← verdadero
      devolver posicion
```

Implementación de la búsqueda binaria: Java, Python

Análisis de corrección (I)

- **Base inductiva**: El intervalo vacío, correspondiente a primero>ultimo. Retorna falso, lo cual siempre es correcto.
- **Hipótesis inductiva**: Dado un intervalo [primero,ultimo], suponemos que el procedimiento siempre es correcto para cualquier sub-intervalo.
- **Paso inductivo**: 3 casos
  - clave<vector[medio]: Correcto por hipótesis inductiva
  - clave>vector[medio]: Correcto por hipótesis inductiva
  - clave=vector[medio]: El programa devuelve posición=medio si el dato está lo cual es correcto; o devuelve que no el dato no está.

Análisis de corrección (II)

- **Dominio bien fundado**: El dominio es bien fundado porque en todas las secuencias de ejecución el intervalo siguiente es un sub-intervalo propio del elemento precedente.
  (Alternativamente: Dar una definición recursiva de intervalos propios de un vector ordenado)
- **Conclusión**: El procedimiento funciona correctamente para todos los casos que cumplen la pre-condición (vector ordenado ascendentemente).