LENGUAJES, TECNOLOGÍAS Y PARADIGMAS DE PROGRAMACIÓN

TEMA 3:

PROGRAMACIÓN FUNCIONAL

PARTEs II y III

(EJERCICIOS DE AULA)

Salvador Lucas,
Javier Piris, María José Ramírez, María José Vicent y Germán Vidal

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática etsinf

PARTE II: MODELO COMPUTACIONAL

1. Indica cuál de las siguientes afirmaciones referentes a la función

$$\texttt{tres} :: \texttt{Int} \to \texttt{Int}$$
$$\texttt{tres x} = 3$$

es **cierta**.

- A La evaluación perezosa y voraz de cualquier expresión de la forma `tres e`, siendo `e` una expresión de tipo `Int`, devuelve el valor `3`
- B Se trata de una definición local
- C Se trata de una definición parcial
- D La evaluación perezosa de la expresión `tres 4` llega a la forma normal `3`.

2. Dada la siguiente definición

$$\texttt{ejFun x y} = \texttt{x} + 12$$

Indica cuál de las siguientes secuencias de reducciones corresponde a la evaluación perezosa de la expresión "`ejFun (9-3) (ejFun 34 3)`":

- A `ejFun (9-3) (ejFun 34 3)` → `ejFun (9-3) (34 + 12)` → `ejFun (9-3) 46` → `ejFun 6 46` → `6 + 12` → `18`
- B `ejFun (9-3) (ejFun 34 3)` → `ejFun (9-3) (34 + 12)` → `ejFun (9-3) 46` → `6 + 12` → `18`
- C `ejFun (9-3) (ejFun 34 3)` → `ejFun 6 (34 + 12)` → `6 + 12` → `18`
- D `ejFun (9-3) (ejFun 34 3)` → `(9-3) + 12` → `6 + 12` → `18`

3. Indica cuál de los siguientes opciones representa el redex que selecciona la **estrategia voraz** para efectuar un paso de reducción de la expresión "`ejFun2 (9-3) (ejFun2 34 3)`" con respecto al programa:

$$\texttt{ejFun2 x \_} = \texttt{x} + \texttt{x}$$

- A $(9 - 3)$
- B `ejFun2 34 3`
- C `ejFun2` $(9 - 3)$`(ejFun2 34 3)`
- D la expresión no contiene redexes ya que está en forma normal.

4. Considerando las funciones `if cond then x else z` y `filter` de Haskell definidas por las ecuaciones

```
if True then x else y = x
if False then x else y = y

filter p [] = []
filter p (x:xs) = if p x  then x:(filter p xs)
                          else filter p xs
```

indica cuál de las siguientes secuencias corresponde a la evaluación perezosa de la expresión `f(filter (/=0) [6,0,1,2])` con respecto al siguiente programa funcional:

```
f :: [Int] -> Int
f [ ] = 0
f (x:xs) = x + 3
```

A `f (filter (/=0) [6,0,1,2])` $\rightarrow$ `f (if (/=0) 6 then 6:(filter (/=0) [0,1,2]) else filter (/=0) [0,1,2])` $\rightarrow$ `f (if True then 6:(filter (/=0) [0,1,2]) else filter (/=0) [0,1,2])` $\rightarrow$ `f (6:filter (/=0) [0,1,2])` $\rightarrow$ `f (6: (if (/=0) 0 then 0:(filter (/=0) [1,2]) else filter (/=0) [1,2]))` $\rightarrow$ `f (6: (if False then 0:(filter (/=0) [1,2]) else filter (/=0) [1,2]))` $\rightarrow$ `f (6:filter (/=0) [1,2])` $\rightarrow$ `f (6: (if (/=0) 1 then 1:(filter (/=0) [2]) else filter (/=0) [2]))` $\rightarrow$ `f (6: (if True then 1:(filter (/=0) [2]) else filter (/=0) [2]))` $\rightarrow$ `f (6:1:filter (/=0) [2])` $\rightarrow$ `f (6: 1 (if (/=0) 2 then 2:(filter (/=0) []) else filter (/=0) []))` $\rightarrow$ `f (6: 1 (if True then 2:(filter (/=0) []) else filter (/=0) []))` $\rightarrow$ `f (6:1:2:filter (/=0) [])` $\rightarrow$ `f (6:1:2:[])` $\rightarrow$ `6 + 3` $\rightarrow$ `9`

B `f (filter (/=0) [6,0,1,2])` $\rightarrow$ `f (6:1:2:[])` $\rightarrow$ `6 + 3` $\rightarrow$ `9`

C `f (filter (/=0) [6,0,1,2])` $\rightarrow$ `6 + 3` $\rightarrow$ `9`

D `f (filter (/=0) [6,0,1,2])` $\rightarrow$ `f (if (/=0) 6 then 6:(filter (/=0) [0,1,2]) else filter (/=0) [0,1,2])` $\rightarrow$ `f (if True then 6:(filter (/=0) [0,1,2]) else filter (/=0) [0,1,2])` $\rightarrow$ `f (6: filter (/=0) [0,1,2])` $\rightarrow$ `6 + 3` $\rightarrow$ `9`

5. Indica cuál de las siguientes expresiones **ES** un redex de la expresión `map snd (zip [Cero,Cero+(S Cero),Cero+(S Cero)+(S(S Cero))] [0])` en el siguiente programa funcional:

```
data Nat = Cero | S Nat deriving Show
instance Num Nat where
    Cero + x = x
    (S x) + y = S(x+y)
```

map función lista -> no hay lista aún
snd tupla -> snd (x,y) -> no hay tupla aún

A `Cero`

B `zip [Cero,Cero+(S Cero),Cero+(S Cero)+(S(S Cero))] [0]`

C `map snd (zip [Cero,Cero+(S Cero),Cero+(S Cero)+(S(S Cero))] [0])`

D `snd (zip [Cero,Cero+(S Cero),Cero+(S Cero)+(S(S Cero))] [0])`

6. Indica cuál **ES** la forma normal de la expresión `map snd (zip [Cero,Cero+(S Cero),Cero+(S Cero)+(S(S Cero))] [0..])` de la pregunta anterior:

A `[Cero, S Cero, S (S Cero)]`

B `[0,1,2]`

C `[(Cero,0), (S Cero,1), (S (S Cero),2)]`

D la expresión no tiene forma normal porque, bajo cualquier estrategia de evaluación, la computación no termina

7. Indica cuál **ES** la forma normal de la expresión `(cuadrado Cero)` con respecto al siguiente programa:

```
data Nat = Cero | S Nat deriving Show
instance Num Nat where
   Cero + x = x
   (S x) + y = S(x+y)

cuadrado::Nat -> Nat
cuadrado (S Cero) = S Cero
cuadrado (S x) = cuadrado x + x + (S x)
```

A `Cero`

B `S Cero`

C No tiene forma normal ya que da un error de ejecución.

D `cuadrado Cero`

8. La evaluación de la expresión `(cuadrado Cero)` de la pregunta anterior es:

   A incompleta.

   B incorrecta.

   C de éxito.

   D de fallo.

9. Indica cuál de las siguientes expresiones **NO** es un redex respecto del programa:

```
pair x y = (x,y)

mapflip f (x,y) = (f y, f x)
```

   A `mapflip reverse (pair ‘‘Hola’’ ‘‘mundo’’)`

   B `mapflip (2*) (1,10)`

   C `pair ‘‘Hola’’ ‘‘mundo’’`

   D `pair (length ‘‘Hola’’) (length ‘‘mundo’’)`

10. Indica cuál de los siguientes opciones representa el redex que selecciona la **estrategia perezosa** para efectuar un paso de reducción de la expresión `surface(nudge (Circle (Point 0 0) 2) (fst (centerScreen 30 20)) (snd (centerScreen 30 20)))`
con respecto al siguiente programa funcional:

```
data Point = Point Float Float deriving (Show)
data Shape = Circle Point Float | Rectangle Point Point deriving (Show)

nudge :: Shape -> Float -> Float -> Shape
nudge (Circle (Point x y) r) a b = Circle (Point (x+a) (y+b)) r
nudge (Rectangle (Point x1 y1) (Point x2 y2)) a b =
            Rectangle (Point (x1+a) (y1+b)) (Point (x2+a) (y2+b))

surface :: Shape -> Float
surface (Circle _ r) = 3.14 * r ^ 2
surface (Rectangle (Point x1 y1) (Point x2 y2)) =
              (abs (x2 - x1)) * (abs(y2 - y1))

centerScreen :: Float -> Float -> (Float,Float)
centerScreen x y = ((x/2), (y/2))
```

A `centerScreen 30 20`

B `fst(centerScreen 30 20)`

C `nudge(Circle(Point 0 0) 2)(fst(centerScreen 30 20))(snd(centerScreen 30 20))`

D `surface(nudge(Circle(Point 0 0) 2)(fst(centerScreen 30 20))`
`(snd(centerScreen 30 20)))`

11. Indica cuál de las siguientes afirmaciones es **CIERTA**:

A No importa la estrategia (voraz o perezosa) que se use para evaluar una expresión ya que ambas generan la misma secuencia de pasos de reducción.

B Para algunas expresiones la estrategia perezosa computa la forma normal mientras que la voraz cae en bucles infinitos.

C Para algunas expresiones la estrategia voraz computa la forma normal mientras que la perezosa cae en bucles infinitos.

D He de indicar en Haskell la estrategia a ser utilizada para evaluar una expresión.

12. Dado el siguiente programa funcional:

```
inc x = x+1
head (x:xs) = x
from x = x : from (x+1)
zip [] _ = []
zip (_:_) [] = []
zip (x:xs) (y:ys) = (x,y) : zip xs ys
```

Indica cuál es la forma normal de:

```
zip ((inc 0) : 2 : []) ((head (from 1)) : 2 : [])
```

A No tiene.

B `[(1,1),(2,2)]`

C `[1,2]`

D `[(1,2),(1,2)]`

13. Dado el siguiente programa funcional, marca la respuesta **CORRECTA**:

```
type Nombre = String
type Apellido = String
data Autor = A Nombre Apellido

richardBird = A "Richard" "Bird"
getName (A fname name) = name
```

A La expresión `richardBird` no es un redex.

B La expresión `getName richardBird` es un redex de la última ecuación del programa.

C La expresión `getName richardBird` contiene un redex. = name = "Bird"

D La expresión `getName (A '''' '''')` no es un redex.

14. ¿Qué devuelve la evaluación perezosa de la expresión `f 3`?

```
f n = take  n [(x,y) | x<-[1..], y<-[1..]]
```

backtracking

A `[(1,1),(1,2),(1,3)]` <-

B `[(1,1),(2,1),(3,1)]`

C la computación no termina.

D `[(1,1),(2,2),(3,3)]`

15. Indica la forma normal de la expresión `zip [1..] [1..]`:

A `(1,1): zip [2..] [2..]`

B `zip [1..] [1..]` ya está en forma normal.

C `([1..], [1..])`

D no existe (la computación no termina)

16. ¿Cuál de las siguientes trazas corresponde a la evaluación perezosa de la expresión `select [0,1,0] [(3+2),tres,5]` con respecto al siguiente programa?

```
tres = 3
select [] _ = []
select (1:xs) (y:ys) = y:(select xs ys)
select (0:xs) (y:ys) = select xs ys
```

A `select [0,1,0] [(3+2),tres,5]` → `select [1,0] [tres,5]` → `tres:(select [0] [5])` → `3:(select [0] [5])` →`3:(select [] [])` → `3:[]`

B `select [0,1,0] [(3+2),tres,5]` → `select [1,0] [tres,5]` → `tres:(select [0] [5])` → `tres:[]` → `3:[]`

C `select [0,1,0] [(3+2),tres,5]` → `select [0,1,0] [5,tres,5]` → `select [0,1,0] [5,3,5]` → `select [1,0] [3,5]` →`3:(select [0] [5])` → `3:[]`

D `select [0,1,0] [(3+2),tres,5]` → `select [1,0] [tres,5]` → `3:[]`

17. Dado un programa funcional, un *redex* es:

A Una expresión que no puede reducirse.

B Una expresión que es una instancia de la parte izquierda de una ecuación del programa.

C El resultado final del proceso de evaluación de un programa funcional.

D Un tipo de estrategia de evaluación que siempre obtiene la forma normal de una expresión inicial (si ésta existe).

PARTE III: CARACTERISTICAS AVANZADAS

18. Dadas dos funciones `f` y `g` con definiciones de tipo `f :: b → c` y `g :: a → b`, indica cuál de las siguientes expresiones será el tipo de la composición de funciones $(\cdot)$, sabiendo que $(\cdot)$ se define por la ecuación $(\mathtt{f} \cdot \mathtt{g})\ \mathtt{x} = \mathtt{f}(\mathtt{g}\ \mathtt{x})$.

Primero se hace g, por lo que f debe tener la misma entrada que la salida que g. Después es: entrada de g -> salida de f

- [A] $(\cdot) :: \mathtt{a} \to \mathtt{c}$
- [B] $(\cdot) :: (\mathtt{b} \to \mathtt{c}) \to (\mathtt{a} \to \mathtt{b})$
- [C] $(\cdot) :: (\mathtt{b} \to \mathtt{c}) \to (\mathtt{a} \to \mathtt{b}) \to \mathtt{a} \to \mathtt{c}$ <-
- [D] $(\cdot) :: (\mathtt{b} \to \mathtt{c}) \to (\mathtt{a} \to \mathtt{b}) \to \mathtt{c}$

19. Dada la función $pair\, x\, y = (x, y)$, indica cuál de las siguientes ecuaciones define la función estándar `zip` usando la siguiente función `zipWith`:

-> aplica una función a las listas a concatenar

```
zipWith f (a : as) (b : bs) = f a b : zipWith f as bs
zipWith = [ ]
```

- [A] `zip xs ys = pair (zipWith xs ys)`
- [B] `zip = zipWith · pair`
- [C] `zip = zipWith pair`
- [D] ninguna de las anteriores

20. Indica qué computa la siguiente función:

```
pippo xs = foldl (*) 1 xs
```

pippo [1,2,5] = 1* 1*2*5 = 10

- [A] suma los elementos de la lista de entrada y multiplica por 1 el resultado
- [B] devuelve la lista de entrada (ya que multiplica por 1 cada elemento)
- [C] calcula el producto de los elementos de una lista <-
- [D] calcula el producto de los elementos de una lista, excepto el primero

21. Indica cuál de las siguientes expresiones **no** es de orden superior

   A `dosVeces f x = f(f x)`

   B $(f \cdot g)x = f(g\ x)$

   C `aplicar f x = f * x`

   D $dosVeces\ f = f \cdot f$

22. Una lista de enteros $[x_1, x_2, \ldots, x_{n-1}, x_n]$ está ordenada si y sólo si $(x_1 \leq x_2 \wedge \ldots \wedge x_{n-1} \leq x_n) = True$. La siguiente definición se basa en la idea anterior

```
esOrdenada xs =  and [x ≤ y|(x,y) ← zip xs (tail xs)]
                 where and...
```

   Indica cuál de las siguientes ecuaciones completa la definición de la función and:

   A `and (x:xs)= (&&) x xs`

   B `and xs= (&&) True xs`

   C `and xs= foldr (&&) True xs`

   D `and xs= (&&) xs xs`

23. Asumiendo definida la función `iSort`, que ordena en orden creciente una lista de enteros, completa la siguiente función para calcular el elemento menor de una lista:

```
minimo :: [Int] ->Int
minimo = f . iSort
```

   A `f = init` -> Devuelve todos menos el último de la cola

   B `f = head` <-

   C `f = (<)`

   D `f = (:)`

24. Indica cuál de las siguientes funciones `f1, f2, f3` y `f4` no es equivalente a las otras (es decir, no computa los mismo valores cuando dicha función se aplica a un entero y una lista de entrada no vacía):

   A
```
f1 x alist = map (+ x) alist
```

   B
```
f2 x alist = [x+y | y <- alist]
```

   C
```
f3 _ [ ] = [ ]
f3 x (y:ys) = (x+y):f3 x ys
```

   D
```
f4 x = map . (+ x)
```

25. ¿Para qué argumentos la siguiente función devuelve el valor `True`?

$$3 > \texttt{mod x 2} = (3 >) \cdot (\text{`}\texttt{mod}\text{`}2)$$

Primero hace mod 2 y después comprueba si es menor que tres

- [A] cualquier número entero.
- [B] sólo para los números pares.
- [C] sólo para los números impares.
- [D] sólo para los múltiplos de 3.

26. Indica cuál de las siguientes ecuaciones **NO** define una función `paraTodo` que dado un predicado `p` y una lista `xs` devuelva `True` si todos sus elementos satisfacen `p` y `False` en caso contrario:

- [A]
```
paraTodo p xs = and [p x | x <- xs]
```
- [B]
```
paraTodo p xs = foldr f True xs
                where f x y = (p x) && y
```
- [C]
```
paraTodo p xs = null (filter p xs)
```
- [D]
```
paraTodo p [ ] = True
paraTodo p (x:xx) = (p x) && (paraTodo p xx)
```

27. Indica qué computa la siguiente función, siendo `max` la función que devuelve el mayor de dos números:

```
pippo (x:xs) = foldr max x xs
```

- [A] devuelve `true` si el primer elemento de la lista es el mayor de ésta
- [B] devuelve el primer elemento de la lista
- [C] computa el máximo de la lista
- [D] computa la lista de los elementos más grandes, uno por cada pareja de elementos de la lista

28. Indica cuál de las siguientes afirmaciones es **FALSA**, en relación con el siguiente programa:

```
any p xs = or (map p xs)
or xs = foldr (||) False xs
```

- [A] aplicada a un predicado y una lista, la función `any` devuelve `True` si cualquiera de los elementos de la lista satisface el predicado, y `Falso` en caso contrario.
- [B] la exprexión `any (<11) [10..20]` se evalúa a `True`
- [C] el tipo de la función `or` es :: `Bool` → `Bool` → `Bool`
- [D] el tipo de la función `any` es :: (`a` → `Bool`) → [`a`] → `Bool`

29. ¿Cuál de las siguientes definiciones de función es **INCORRECTA**?

A `func1 x y = [x e | e<-y]` func1 :: (t -> a) -> [t] -> [a]

B `func2 x y = map x y` func1 (2+) [1,2,3] = [2,4,5]

C `func3 x y = x y`

D `func4 x y = take (length x) [x e | y <- e]`

30. Indica qué calcula la función `f n = pro [1..n]`, donde `pro = foldr (*) 1`.

A el fibonacci de `n`

B el factorial de `n`

C el promedio de los valores entre `1` y `n`

D la mediana de los valores entre `1` y `n`

31. Indica cuál de las siguientes funciones **NO** es de orden superior:

A
```
takeWhile :: (a -> Bool) -> [a] -> [a]
takeWhile _ [] = []
takeWhile p (x:xs)
      | p x = x : takeWhile' p xs
| otherwise = []
```

B
```
initialUpperCase :: String -> String
initialUpperCase [] = []
initialUpperCase (x:xs) = toUpper x : [toLower x| x<-xs]
```

C
```
productoP :: Num a => (a -> Bool) -> [a] -> a
productoP p = foldr (\x y -> if p x then x*y else y) 1
```

D
```
anotherSum :: Num b => (a -> b) -> [a] -> b
anotherSum f [] = 0
anotherSum f (x:xs) = f x + anotherSum f xs
```

32. Indica cuál de las siguientes opciones define el significado de la función `succ = toEnum . (1+) . fromEnum`

(donde se recuerda que la función `fromEnum` convierte a entero un valor de un tipo enumerado mientras que `toEnum` hace la conversión contraria):

A Calcula el sucesor de un número entero dado.

B Calcula el valor que sigue, en la enumeración, a un valor dado. Por ejemplo, para el caracter `'a'` tendríamos que `succ 'a' = 'b'`.

C Es la función identidad.

D Calcula lo mismo que esta función: `succ' v = succ' (fromEnum v)`.

33. Dada la definición habitual de `foldr`:

```
foldr f z []     =  z
foldr f z (x:xs) =  f x (foldr f z xs)
```

¿Qué hace la siguiente función?

```
foo xs = foldr dup [] xs
   where dup x y = x:x:y
```

A Eleva al cuadrado los elementos de una lista.

B Toma una lista de listas y las concatena.

C Toma una lista y devuelve una nueva lista en la que cada elemento de la lista original aparece duplicado, por ej., `foo [1,2,3] -> [1,1,2,2,3,3]`.

D Toma una lista y devuelve una nueva lista en la que cada elemento de la lista original es reemplazado por un par de listas con dicho elemento, por ejemplo, `foo [1,2,3] -> [[1],[1],[2],[2],[3],[3]]`.