

# Teoría de Autómatas y Lenguajes Formales (2009/10)

## Práctica 0

### (Repaso de Ocaml)

#### Enunciado

---

#### Nota

Esta práctica consta de una serie de ejercicios cuya resolución tiene como propósito familiarizarse de nuevo con el lenguaje de programación Ocaml. Dejaremos un plazo de dos semanas para alcanzar las soluciones.

---

#### Ejercicio 1

La función `mapdoble` recibe como argumentos dos funciones y una lista, y devuelve una lista resultado de aplicar la primera función a los elementos de la lista que ocupan posición impar, y la segunda a los que ocupan posición par.

Por ejemplo:

```
# mapdoble (function x -> x) (function x -> -x) [1;1;1;1;1];;  
- : int list = [1; -1; 1; -1; 1]
```

Se pide:

- Implemente la función `mapdoble`.
- Indique el tipo de la función `mapdoble`.
- Indique el valor de:

```
mapdoble (function x -> x*2) (function x -> "x") [1;2;3;4;5];;
```

- Indique el tipo de:

```
let y = function x -> 5 in mapdoble y;;
```

---

#### Ejercicio 2

- Defina una función `primero_que_cumple`, que dado un predicado (es decir, una función de tipo `'a -> bool`) y una lista, devuelva el primer elemento de la lista que verifica dicho predicado.
- Indique el tipo de la función `primero_que_cumple`.
- Utilizando la función `primero_que_cumple`, defina una función `existe`, que dado un predicado y una lista devuelva `true` si en la lista hay algún elemento que verifica el predicado, y `false` en caso contrario.
- Se quiere mantener un conjunto de valores etiquetados de cualquier tipo, mediante una lista de pares `'a * 'b`, donde la primera componente del par es la etiqueta o clave, y la segunda es el valor asociado a esa clave en dicho conjunto. Utilizando la función `primero_que_cumple`, defina una función `asociado : 'a -> ('a * 'b) list -> 'b`, que dado un conjunto y una clave, devuelva

su valor asociado.

---

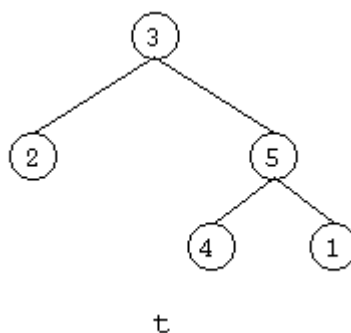
### Ejercicio 3

Se ha construido el siguiente tipo de dato con el fin de poder representar árboles binarios donde la información que aparece en cada nodo puede ser de cualquier tipo:

```
type 'a arbol_binario =
  Vacio
  | Nodo of 'a * 'a arbol_binario * 'a arbol_binario;;
```

Se pide:

- Construya las funciones `in_orden`, `pre_orden`, `post_orden` y `anchura`, todas ellas de tipo `'a arbol_binario -> 'a list`, que devuelvan los correspondientes recorridos sobre un árbol binario dado, tal y como se muestra en los siguientes ejemplos:



```
# in_orden t;;
- : int list = [2; 3; 4; 5; 1]

# pre_orden t;;
- : int list = [3; 2; 5; 4; 1]

# post_orden t;;
- : int list = [2; 4; 1; 5; 3]

# anchura t;;
- : int list = [3; 2; 5; 4; 1]
```

---

### Ejercicio 4

Consideremos el siguiente tipo de dato para una representación de conjuntos basada en listas sin elementos repetidos:

```
type 'a conjunto = Conjunto of 'a list;;
```

Por ejemplo, el conjunto vacío se podría representar mediante el siguiente valor:

```
let conjunto_vacio = Conjunto [];;
```

Se pide implementar las siguientes funciones:

- `es_vacio : 'a conjunto -> bool`
- `pertenece : 'a -> 'a conjunto -> bool`
- `agregar : 'a -> 'a conjunto -> 'a conjunto`
- `conjunto_of_list : 'a list -> 'a conjunto`
- `suprimir : 'a -> 'a conjunto -> 'a conjunto`
- `cardinal : 'a conjunto -> int`

- union : 'a conjunto -> 'a conjunto -> 'a conjunto
  - interseccion : 'a conjunto -> 'a conjunto -> 'a conjunto
  - diferencia : 'a conjunto -> 'a conjunto -> 'a conjunto
  - incluido : 'a conjunto -> 'a conjunto -> bool
  - igual : 'a conjunto -> 'a conjunto -> bool
  - list\_of\_conjunto : 'a conjunto -> 'a list
  - producto\_cartesiano : 'a conjunto -> 'b conjunto -> ('a \* 'b) conjunto
-