# Ingeniería Técnica en Informática de Gestión
# Laboratorio de Programación Avanzada

## Curso 2010/2011
## Práctica Funcional N: 5

**Autor: 09047323C - Lucena Pumar, Diego Antonio**

En la Villa de Meco a 24 de Agosto de 2011.
Fecha de última revisión: Meco, 24 de Agosto de 2011.
Editor: Diego Antonio Lucena Pumar.
Maquetado directamente en LaTeX $2_\varepsilon$

*BLA, BLA...*

*"BLA, BLA, BLA..."*

**BLA...**

# Índice general

**1. Planteamiento del problema** **1**
1.1. Condiciones . . . . . 1
1.2. Funciones propuestas . . . . . 1
1.3. Tipos de Sudoku . . . . . 2

**2. Análisis destallado del Código** **3**
2.1. Tableros de Pruebas . . . . . 3
2.1.1. Tablero 1 . . . . . 3
2.1.2. Tablero 2 . . . . . 3
2.1.3. Tablero 3 . . . . . 4
2.1.4. Tablero 4 . . . . . 4
2.2. Lógica para Imprimir Tablero . . . . . 4
2.2.1. cadenahaciaListadeChar . . . . . 4
2.2.2. lstTablero . . . . . 5
2.2.3. imprimeTerciodeLinea . . . . . 5
2.2.4. imprimeLinea . . . . . 6
2.2.5. imprimeCuartodeTablero . . . . . 6
2.2.6. imprimeContenidodeTablero . . . . . 7
2.2.7. imprimirTablero . . . . . 7
2.2.8. mostrarTablero . . . . . 7
2.3. Lógica para Tratar Jugada . . . . . 8
2.3.1. valordeAccion . . . . . 8
2.3.2. substrPreAccion . . . . . 9
2.3.3. substrPostAccion . . . . . 9
2.3.4. jugada . . . . . 9
2.3.5. introDatos . . . . . 10
2.3.6. introJugada . . . . . 10
2.4. Lógica para determinar su el Tablero es Correcto . . . . . 11

2.4.1. fila . . . 11
2.4.2. columna . . . 12
2.4.3. listaFila . . . 12
2.4.4. listaColumna . . . 12
2.4.5. cuadrado . . . 13
2.4.6. listaCuadrado . . . 13
2.4.7. elemento . . . 14
2.4.8. buscarElemento . . . 14
2.4.9. . . . 15
2.4.10. sonlasColumnasCorrectas . . . 15
2.4.11. sonlosCuadradosCorrectos . . . 16
2.4.12. esSudokuCorrecto . . . 16
2.5. Lógica para Resolver el Sudoku de manera Fácil . . . 17
2.5.1. quitarElemento . . . 17
2.5.2. elementosListadeEnteros . . . 17
2.5.3. elementosFila . . . 18
2.5.4. elementosColumna . . . 18
2.5.5. elementosCuadrado . . . 18
2.5.6. filasinElemento ??? . . . 19
2.5.7. elementosparaIndice . . . 19
2.5.8. eliminarBlancos . . . 19
2.5.9. insertar . . . 20
2.5.10. ordenar . . . 20
2.5.11. buscarElemento . . . 21
2.5.12. frontBuscarElemento . . . 21
2.5.13. elementoparaIndice . . . 22
2.5.14. fontTableroconJugada . . . 22
2.5.15. resolverSudokudeManeraFacil . . . 22
2.6. Lógica para Resolver el Sudoku de Manera Díficil (Algoritmo de BACKTRAKING) . . . 23
2.7. Función Principal . . . 23
2.7.1. main . . . 23
2.8. Función de Ejecución . . . 25
2.8.1. main(cadenahaciaListadeChar(tablero), tablero);; . . . 25

**3. Pruebas** **27**

**Bibliografía** **29**

# Capítulo 1

# Planteamiento del problema

**Nota:** Presentación basada em el texto: [Mar10].

## 1.1. Condiciones

La presente prática gira en torno a la creación de una aplicación en base al lenguaje de programación funcional "Caml" en su versión ligera. Dicha aplicación ha de ser un juego "Sudoku Dodeka", versión de popular juego de cifras "Sudoku". Las reglas de este variante se detallan a continuación:

- Las posibles fichas del juego contiene las cifras decimales de: 0 a 9 y las teras A y B.
- Para todos los elementos de una misma fila no se han de repetir dos fichas iguales.
- Para todos los elementos de una misma columna no se han de repetir dos fichas iguales.
- Para todos los elementos de una mismo retángulo (3x4) no se han de repetir dos fichas iguales.
- El juego termina cuando se cumplen las reglas anteriormenete expuestas y no hay ninguna casilla en blanco ('.').

## 1.2. Funciones propuestas

Para el mismo juego se proponen una funciones básicas que deben ser parte del código. Las misma son:

- formarSudoku: Toma como argunmento una lista de carácteres (tablero) y devuelve el Sudoku formateado.
- mostrar tablero: Recibe un tablero como argumento y devuelve el dibujo del mismo.
- elemento casilla tablero: Devuelve el valor que tiene asignado dicha casilla en el tablero.
- posibles casilla tablero: Devuelve una lista con los valores que puede tomar dicha casilla en el tablero.
- resuelto tablero: Devuelve cierto si el Sudoku está resuelto y falso en caso contrario.

## 1.3. Tipos de Sudoku

Existe tres tipos de Sudoku planteados:

- Sudoku fácil: Se trata de un Sudoku sencillo en el que su solución viene dada por eliminación de posibles valores sobre las distintas casillas.
- Sudoku difícil: Se trata de un Sudoku en el que la única forma de obtener su solución se hace con algorítmia compleja (técnica de BackTraking)
- Sudoku imposible: Se trata de un Sudoku en el que por la disposición de las fichas en el tablero es imposible llegar a una solución.

# Capítulo 2

# Análisis destallado del Código

## 2.1. Tableros de Pruebas

### 2.1.1. Tablero 1

**Propósito:** Se trata del tablero de prueba utilizado para confeccionar el juego. Es un tablero ideal para conformar algoritmos para resolución simple y por métodos de "backtraking"sobre el tablero.

**Prototipo:**

**Tablero:**

```
let tablero1 =
".2..06.74....7..3.1.02..81....5.9.........A.342.4.2.1..3.....A36.2.........8.
129.....4..0.A.6.0B1.3.........4.0....8A..10.5.8..9....89.3A..0.";;
```

### 2.1.2. Tablero 2

**Propósito:** Se trata de un tablero correcto para pruebas sobre resolución del mismo añadiendo o eliminando jugadas.

**Prototipo:**

**Tablero:**

```
let tablero2 =
"6193B07A24850AB43528967175281694A03B8B47610235A99610A45382B7235A78B96104B4390A6
517281765928B034AA80243175B9649A15730B862508B29467A1332768BA14950";;
```

### 2.1.3. Tablero 3

**Propósito:** Se trata de un tablero correcto para pruebas sobre resolución del mismo añadiendo o eliminando jugadas.

**Prototipo:**

**Tablero:**

```
let tablero3 =
"6193.07A24850AB43528967175281694A03B8B47610235A99610A45382B7235A78B96104B4390A6
517281765928B034AA80243175B9649A15730B862508B29467A1332768BA14950";;
```

### 2.1.4. Tablero 4

**Propósito:**

**Prototipo:**

**Tablero:**

```
let tablero4 =
".....BA98765432.....
.....";;
```

## 2.2. Lógica para Imprimir Tablero

### 2.2.1. cadenahaciaListadeChar

**Propósito:** Convierte una cadena de "String"hacia una lista de Char.

**Prototipo:** cadenahaciaListadeChar : string -> char list = <fun>

**Prueba 1:** let lstTablero tablero = cadenahaciaListadeChar tablero;;

**Código:**

```
let rec cadenahaciaListadeChar = function str -> let long = string_length str
in if long = 0 then []
else (nth_char str 0) :: cadenahaciaListadeChar(sub_string str 1 (long-1));;
```

### 2.2.2. lstTablero

**Propósito:**

**Prototipo:**

**Prueba 1:**

**Código:**

```
let lstTablero tablero = cadenahaciaListadeChar tablero;;
```

### 2.2.3. imprimeTerciodeLinea

**Propósito:** Imprime cuatro casillas sobre el todal de doce de una línea de tablero.

**Prototipo:** imprimeTerciodeLinea : char list * int -> char list = <fun>

**Prueba 1:**

**Código:**

```
let rec imprimeTerciodeLinea (linea,indice) = if (indice < 5 ) then
 begin
 print_string " ";
 print_char (hd linea);
 print_string " ";
 imprimeTerciodeLinea ((tl linea),indice+1)
 end
                       else linea;;
```

### 2.2.4. imprimeLinea

**Propósito:** Imprime una línea del tablero.

**Prototipo:** imprimeLinea : char list * int -> char list = <fun>

**Prueba 1:**

**Código:**

```
let rec imprimeLinea (linea,indice) = if (indice < 4 ) then
begin
print_string " /";
imprimeLinea((imprimeTerciodeLinea (linea,1)), indice+1)
end
              else
         begin
                                                print_string "/";
                                                print_newline();
    linea
         end;;
```

### 2.2.5. imprimeCuartodeTablero

**Propósito:** Imprime tres líneas de tablero. 1/4 del total de casillas.

**Prototipo:** imprimeCuartodeTablero : char list * int -> char list = <fun>

**Prueba 1:**

**Código:**

```
let rec imprimeCuartodeTablero (linea,indice) = if (indice < 37 ) then
imprimeCuartodeTablero((imprimeLinea (linea,1)), indice+12)
else
begin
print_string "----------------------------------------";
print_newline();
linea
```

```
end;;
```

### 2.2.6. imprimeContenidodeTablero

**Propósito:** Imprime las casillas del tablero.

**Prototipo:** imprimeContenidodeTablero : char list * int -> unit = <fun>

**Prueba 1:**

**Código:**

```
let rec imprimeContenidodeTablero (linea,indice) = if (indice < 5 ) then
imprimeContenidodeTablero((imprimeCuartodeTablero(linea,1)), indice+1)
           else
      ();;
```

### 2.2.7. imprimirTablero

**Propósito:** Imprime el tablero completo. Es el "front end"de ïmprimeContenidodeTablero".

**Prototipo:** imprimirTablero : char list -> unit = <fun>

**Prueba 1:**

**Código:**

```
let imprimirTablero tablero = begin
print_newline();
print_string "----------------------------------------";
print_newline();
imprimeContenidodeTablero (tablero,1);
()
end;;
```

### 2.2.8. mostrarTablero

**Propósito:** Muestra el tablero sobre una cadena que convierte a lista de Çhar".

**Prototipo:** mostrarTablero : string -> unit = <fun>

**Prueba 1:**

**Código:**

```
let mostrarTablero tablero = begin
        let lstTablero = cadenahaciaListadeChar tablero in
imprimirTablero lstTablero;
()
end;;
```

## 2.3. Lógica para Tratar Jugada

### 2.3.1. valordeAccion

**Propósito:** Traduce la numeración Ïnt.ª valor sobre tablero, "String". Por ejemplo, si el jugador introduce como jugada el valor 20 se traduce como ficha .ª".

**Prototipo:** valordeAccion : int -> string = <fun>

**Prueba 1:**

**Código:**

```
let valordeAccion accion = if(accion = 10 ) then "0"
           else if (accion = 11) then "1"
   else if (accion = 12) then "2"
   else if (accion = 13) then "3"
   else if (accion = 14) then "4"
   else if (accion = 15) then "5"
   else if (accion = 16) then "6"
   else if (accion = 17) then "7"
                           else if (accion = 18) then "8"
   else if (accion = 19) then "9"
                           else if (accion = 20) then "A"
   else if (accion = 21) then "B"
   else "999";;
```

### 2.3.2. substrPreAccion

**Propósito:** Crea una cadena desde el valor 0 de str hasta el valor 1"sobre el que se cambia la ficha.

**Prototipo:** substrPreAccion : string * int * int -> string = <fun>

**Prueba 1:**

**Código:**

```
let substrPreAccion (str, linea, columna) =  sub_string str 0
((12*((linea)-1)+(columna))-1);;
```

### 2.3.3. substrPostAccion

**Propósito:** Crea una cadena desde el valor .ªcción+1"de str hasta el valor límite de str.

**Prototipo:** substrPostAccion : string * int * int -> string = <fun>

**Prueba 1:**

**Código:**

```
let substrPostAccion (str, linea, columna) = let salida =
((12*((linea)-1)+(columna))) in sub_string str salida ((string_length
str)-salida);;
```

### 2.3.4. jugada

**Propósito:** Comprueba si el valor de jugada es correcto y porteriormente inserta la jugada y reconstruye el tablero.

**Prototipo:** jugada : string * int * int * int -> string = <fun>

**Prueba 1:**

**Código:**

```
let rec jugada (str, linea, columna, accion) = begin
                if (accion > 21) then begin
                  print_string "Las jugada es incorrecta...";
                  jugada(str, linea, columna, accion);
                  ""
                  end
                else concat [(substrPreAccion
(str, linea, columna));(valordeAccion accion);(substrPostAccion (str, linea,
columna))]
                                                    end;;
```

### 2.3.5. introDatos

**Propósito:** Pide los datos necesarios para la jugada y comprueba si esta en el rango correcto.

**Prototipo:** introDatos : string * int * int -> int = <fun>

**Prueba 1:**

**Código:**

```
let rec introDatos (tipo, min, max) = begin
                   print_string "Por favor introduzca ";
   print_string tipo;
   print_string " de (";
   print_int min;
   print_string " a ";
   print_int max;
   print_string "): ";
                   let aux = read_int () in
                      if(aux >= min) \&\& (aux <= max) then aux
      else introDatos (tipo, min, max)
       end;;
```

### 2.3.6. introJugada

**Propósito:** Llama a ïntroDatosçon jugada y rangos. Es el "font end"de ïntroDatos".

**Prototipo:** introJugada : string * int * int * int * int -> string = <fun>

**Prueba 1:**

**Código:**

```
let rec introJugada(tablero, fila, columna, juego, opcion) = if(opcion = 1)
then begin
   let aux = introDatos("Fila", 1, 12) in
   introJugada(tablero, aux, 0, 0, 2)
      end
else if (opcion = 2) then begin
   let aux = introDatos("Columna", 1, 12) in
   introJugada(tablero, fila, aux, 0, 3)
      end
else if (opcion = 3) then begin
   let aux = introDatos("Jugada", 10, 21) in
   introJugada(tablero, fila, columna, aux, 4)
      end
else if (opcion = 4) then
   jugada(tablero, fila, columna, juego)
else "";;
```

## 2.4. Lógica para determinar su el Tablero es Correcto

### 2.4.1. fila

**Propósito:** Funcion que devuelve la fila para una determinada posicion 'p' del tablero.

**Prototipo:** fila : int -> int = <fun>

**Prueba 1:**

**Código:**

```
let fila p = if p mod 12 = 0 then (p / 12)
             else
        (p / 12)+1;;
```

### 2.4.2. columna

**Propósito:** Funcion que devuelve la columna para una determinada posicion 'p' del tablero.

**Prototipo:** columna : int -> int = <fun>

**Prueba 1:**

**Código:**

```
let columna p = if p mod 12 = 0 then 12
        else
           p mod 12;;
```

### 2.4.3. listaFila

**Propósito:** Función que devuelve la lista con los índices que forman la fila en la que está situada la casilla que se pasa como parámetro.

**Prototipo:** listaFila : int -> int list = <fun>

**Prueba 1:**

**Código:**

```
let listaFila = function p -> let n = (fila p)-1 in
[1+n*12;2+n*12;3+n*12;4+n*12;5+n*12;6+n*12;7+n*12;8+n*12;9+n*12;10+n*12;11+n*12;
12+n*12];;
```

### 2.4.4. listaColumna

**Propósito:** Función que devuelve la lista con los índices que forman la columna en la que está situada una determinada casilla.

**Prototipo:** listaColumna : int -> int list = <fun>

**Prueba 1:**

**Código:**

```
let listaColumna = function p -> let n = columna p in
[n;n+12;n+24;n+36;n+48;n+60;n+72;n+84;n+96;n+108;n+120;n+132];;
```

### 2.4.5. cuadrado

**Propósito:** Función para obtener el cuadrado en el que se encuentra una posición.

**Prototipo:** cuadrado : int -> int = <fun>

**Prueba 1:**

**Código:**

```
let cuadrado p = if (p > 0) \& ((fila p)-1)/3 = 0 then
                               (((columna p)-1)/4) + 1
    else if ((fila p)-1)/3 = 1 then
                         (((columna p)-1)/4) + 4
    else if ((fila p)-1)/3 = 2 then
                               (((columna p)-1)/4) + 7
    else if ((fila p)-1)/3 = 3 then
                               (((columna p)-1)/4) + 10
            else
            failwith "La posicion no se encuentra en el tablero...";;
```

### 2.4.6. listaCuadrado

**Propósito:** Función que devuelve una lista con los índices del cuadrado al que pertenece un cierto elemento.

**Prototipo:** listaCuadrado : int -> int list = <fun>

**Prueba 1:**

**Código:**

```
let listaCuadrado p = let n = cuadrado p in [1+36*((n-1)/3)+4*((n-1) mod 3);
    2+36*((n-1)/3)+4*((n-1) mod 3);
    3+36*((n-1)/3)+4*((n-1) mod 3);
    4+36*((n-1)/3)+4*((n-1) mod 3);
    3+36*((n-1)/3)+4*((n-1) mod 3);
    14+36*((n-1)/3)+4*((n-1) mod 3);
    15+36*((n-1)/3)+4*((n-1) mod 3);
    16+36*((n-1)/3)+4*((n-1) mod 3);
    25+36*((n-1)/3)+4*((n-1) mod 3);
    26+36*((n-1)/3)+4*((n-1) mod 3);
    27+36*((n-1)/3)+4*((n-1) mod 3);
    28+36*((n-1)/3)+4*((n-1) mod 3)];;
```

### 2.4.7. elemento

**Propósito:** Función que devuelve un elemento a partir de una lista de elementos dada y su posición.

**Prototipo:** elemento : int * int * 'a list -> 'a = <fun>

**Prueba 1:**

**Código:**

```
let rec elemento (indice,recorrido,tabla) = if(indice = recorrido) then hd tabla
            else elemento (indice,(recorrido+1), tl tabla);;
```

### 2.4.8. buscarElemento

**Propósito:** Se trata de un "front"de .elemento", con algunas particularidades. En el caso de encontrar agua devuelve el valor uno sino devuelve el indice del recorrido, e igualmente comprueba que los elementos no se repitan.

**Prototipo:** buscarElemento : char * char list * int -> int list -> int = <fun>

**Prueba 1:**

**Código:**

```
let rec buscarElemento (elementoIndice,tablero,contador) = function [] -> contador
 |h::f -> if(elementoIndice = '.') then 1
  else if(elemento(h,1,tablero) = elementoIndice) then buscarElemento
(elementoIndice,tablero,(contador+1)) f
  else buscarElemento (elementoIndice,tablero,contador) f;;
```

### 2.4.9.

**Propósito:** Función que usa "buscarElemento"para dado un índce de reccorrido 1 y contador 0 comprobar la condición de que no existe agua y todos los elementos de cada fila son distintos.

**Prototipo:** sonlasFilaCorrectas : int * char list -> bool = <fun>

**Prueba 1:**

**Código:**

```
let rec sonlasFilaCorrectas (indice,tablero) = if indice > 144 then true
       else if (buscarElemento
(elemento(indice,1,tablero),tablero,0) (listaFila indice)) > 1 then false
       else sonlasFilaCorrectas
       ((indice+1),tablero);;
```

### 2.4.10. sonlasColumnasCorrectas

**Propósito:** Función que usa "buscarElemento"para dado un índce de reccorrido 1 y contador 0 comprobar la condición de que no existe agua y todos los elementos de cada columna son distintos.

**Prototipo:** sonlasColumnasCorrectas : int * char list -> bool = <fun>

**Prueba 1:**

**Código:**

```
let rec sonlasColumnasCorrectas (indice,tablero) = if indice > 144 then true
          else if (buscarElemento
(elemento(indice,1,tablero),tablero,0) (listaColumna indice)) > 1 then false
          else sonlasColumnasCorrectas
                                    ((indice+1),tablero);;
```

### 2.4.11. sonlosCuadradosCorrectos

**Propósito:** Función que usa "buscarElemento"para dado un índce de reccorrido 1 y contador 0 comprobar la condición de que no existe agua y todos los elementos de cada cuadrado son distintos.

**Prototipo:** sonlosCuadradosCorrectos : int * char list -> bool = <fun>

**Prueba 1:**

**Código:**

```
let rec sonlosCuadradosCorrectos (indice,tablero) = if indice > 144 then true
           else if (buscarElemento
(elemento(indice,1,tablero),tablero,0) (listaCuadrado indice)) > 1 then false
           else
sonlosCuadradosCorrectos ((indice+1),tablero);;
```

### 2.4.12. esSudokuCorrecto

**Propósito:** Es el "front"para las tres funciones anteriores. Devuelve "true.º "false"sobre si el tablero cumple la condición de que no existen elementos repetidos en: cada fila, cada columna y cada cuarto de tablero.

**Prototipo:** esSudokuCorrecto : char list -> bool = <fun>

**Prueba 1:**

**Código:**

```
let esSudokuCorrecto lstTablero = sonlasFilaCorrectas (1, lstTablero) \&\&
sonlasColumnasCorrectas (1, lstTablero) \&\& sonlosCuadradosCorrectos (1,
lstTablero);;
```

## 2.5. Lógica para Resolver el Sudoku de manera Fácil

### 2.5.1. quitarElemento

**Propósito:** Función que elimina un elemento de una lista dada.

**Prototipo:** quitarElemento : 'a * 'a list -> 'a list -> 'a list = <fun>

**Prueba 1:** quitarElemento(10,[]) (listaFila 5);;

**Código:**

```
let rec quitarElemento (elemento, cabezaLista) = function [] -> []
 |h::f-> if (elemento = h) then cabezaLista@f
 else
quitarElemento (elemento, cabezaLista@[h]) f;;
```

### 2.5.2. elementosListadeEnteros

**Propósito:** Función que extrae una lista genérica para lista de enteros dada.

**Prototipo:** elementosListadeEnteros : 'a list * 'a list -> int list -> 'a list = <fun>

**Prueba 1:** elementosListadeEnteros ([],(lstTablero tablero1)) (listaFila 5);;

**Código:**

```
let rec elementosListadeEnteros (listaFinal,tablero) = function [] -> listaFinal
| h::f ->
elementosListadeEnteros ((listaFinal@[(elemento(h, 1, tablero))]),tablero) f;;
```

### 2.5.3. elementosFila

**Propósito:** Es el "front"para obtener la lista genérica de elementos de una fila a partir de un índice.

**Prototipo:** elementosFila : int * 'a list -> 'a list = <fun>

**Prueba 1:** elementosFila (5, (lstTablero tablero1));;

**Código:**

```
let elementosFila (elemento,tablero) = elementosListadeEnteros ([], tablero)
(listaFila elemento);;
```

### 2.5.4. elementosColumna

**Propósito:** Es el "front"para obtener la lista genérica de elementos de una columna a partir de un índice.

**Prototipo:** elementosColumna : int * 'a list -> 'a list = <fun>

**Prueba 1:** elementosColumna (5, (lstTablero tablero1));;

**Código:**

```
let elementosColumna (elemento,tablero) = elementosListadeEnteros ([], tablero)
(listaColumna elemento);;
```

### 2.5.5. elementosCuadrado

**Propósito:** Es el "front"para obtener la lista genérica de elementos de un cuadrado a partir de un índice.

**Prototipo:** elementosCuadrado : int * 'a list -> 'a list = <fun>

**Prueba 1:** elementosCuadrado (5, (lstTablero tablero1));;

**Código:**

```
let elementosCuadrado (elemento,tablero) = elementosListadeEnteros ([], tablero)
(listaCuadrado elemento);;
```

### 2.5.6. filasinElemento ???

**Propósito:** Función que elimina de una lista el elemeto dado como parámentro.

**Prototipo:** filasinElemento : int * int list -> int list = <fun>

**Prueba 1:**

**Código:**

```
let filasinElemento (elemento, tablero) = quitarElemento (elemento,[])
(elementosFila(elemento, tablero));;
```

### 2.5.7. elementosparaIndice

**Propósito:** Función que obtiene la lista de elementos para un índice dado.

**Prototipo:** elementosparaIndice : int * 'a list -> 'a list = <fun>

**Prueba 1:** elementosparaIndice (5, (lstTablero tablero1));;

**Código:**

```
let elementosparaIndice (elemento, tablero) = (elementosFila
(elemento,tablero))@(elementosColumna (elemento,tablero))@(elementosCuadrado
(elemento,tablero));;
```

### 2.5.8. eliminarBlancos

**Propósito:** Función que elimina el carácter '.' de una lista dada.

**Prototipo:** eliminarBlancos : char list -> char list -> char list = <fun>

**Prueba 1:** eliminarBlancos ([]) (elementosparaIndice (5, (lstTablero tablero1)));;

**Código:**

```
let rec eliminarBlancos (listaLimpia) = function [] -> listaLimpia
|h::f -> if (h = '.') then eliminarBlancos (listaLimpia) f
 else eliminarBlancos (listaLimpia@[h]) f;;
```

### 2.5.9. insertar

**Autor:** Luis Bengochea Martínez.

**Licencia de Código:** (C) Luis Bengochea Martínez. Departamento de Ciencias de la Computación. Universidad de Alcalá (UAH). Todos los derechos reservados.

**Propósito:** Función que inserta un elemento en una lista.

**Prototipo:** insertar : 'a -> 'a list -> 'a list = <fun>

**Código:**

```
let rec insertar n = function [] -> [n]
                            |x::l -> if x=n then x::l
                                     else if x>n then n::x::l
                                     else x::(insertar n l);;
```

### 2.5.10. ordenar

**Autor:** Luis Bengochea Martínez.

**Licencia de Código:** (C) Luis Bengochea Martínez. Departamento de Ciencias de la Computación. Universidad de Alcalá (UAH). Todos los derechos reservados.

**Propósito:** Función que ordena una lista genérica de elementos.

**Prototipo:** ordenar : 'a list -> 'a list = <fun>

**Prueba 1:** ordenar (eliminarBlancos ([]) (elementosparaIndice (5, (lstTablero tablero1))));;

**Prueba 2:** ordenar [‘5‘;‘1‘;‘B‘;‘A‘;‘3‘;‘7‘;‘1‘];;

**Código:**

```
let rec ordenar = function [] -> []
                        | x::l -> insertar x (ordenar l);;
```

### 2.5.11. buscarElemento

**Propósito:** Función que busca partiendo en una lista el elemento que no se encuentra en la otra.

**Prototipo:** buscarElemento : ’a list -> ’a list -> ’a = <fun>

**Prueba 1:** buscarElemento [‘0‘;‘1‘;‘2‘;‘3‘;‘4‘;‘5‘;‘6‘;‘7‘;‘8‘;‘9‘;‘A‘;‘B‘;‘.‘] (ordenar (eliminarBlancos ([]) (elementosparaIndice (5, (lstTablero tablero1)))));;

**Código:**

```
let rec buscarElemento listaFormal = function [] -> hd(listaFormal)
|h::f -> if (h = hd(listaFormal)) then buscarElemento (tl(listaFormal)) f
 else hd(listaFormal);;
```

### 2.5.12. frontBuscarElemento

**Propósito:** Se trata de un ”front”para ”buscarElemento”, con el objetivo de simplificar el código.

**Prototipo:** frontBuscarElemento : int -> char list -> char = <fun>

**Prueba 1:** frontBuscarElemento 5 (lstTablero tablero1);;

**Código:**

```
let rec frontBuscarElemento indice tablero = buscarElemento
[‘0‘;‘1‘;‘2‘;‘3‘;‘4‘;‘5‘;‘6‘;‘7‘;‘8‘;‘9‘;‘A‘;‘B‘;‘.‘] (ordenar (eliminarBlancos
([]) (elementosparaIndice (indice, tablero))));;
```

### 2.5.13. elementoparaIndice

**Propósito:** Función que devuelve un elemento para un índice dado.

**Prototipo:** elementoparaIndice : int -> int -> 'a list -> 'a = <fun>

**Prueba 1:** elementoparaIndice 1 5 (lstTablero tablero1);;

**Código:**

```
let rec elementoparaIndice indiceRecorrido indiceInicial lista =
   if (indiceInicial = indiceRecorrido) then (hd(lista))
   else elementoparaIndice(indiceRecorrido+1) indiceInicial (tl(lista));;
```

### 2.5.14. fontTableroconJugada

**Propósito:** Función que inserta en un tablero una jugada y reconstruye el mismo.

**Prototipo:** fontTableroconJugada : int * string * string -> string = <fun>

**Código:**

```
let fontTableroconJugada (indice, jugada, strTablero) = (concat
[(substrPreAccion (strTablero, (fila indice), (columna
indice)));jugada;(substrPostAccion (strTablero, (fila indice), (columna
indice)))]);;
```

### 2.5.15. resolverSudokudeManeraFacil

**Propósito:** Función que resuelve el Sudoku por eliminación simple de posibilidades.

**Prototipo:** resolverSudokudeManeraFacil : int * string * string -> string = <fun>

**Tablero de Prueba Original:**

```
"6193B07A24850AB43528967175281694A03B8B47610235A99610A45382B7235A78B96104B4390A6
517281765928B034AA80243175B9649A15730B862508B29467A1332768BA14950"
```

**tableroPruebaFacil1:**

```
let tableroPruebaFacil1 =
"6193.07A24850AB435289671752816 94A03B8B47610235A99610A45382B7235A78B96104B4390A6
517281765928B034AA80243175B9649A15730B862508B29467A1332768BA1....";;
```

**tableroPruebaFacil2:**

```
let tableroPruebaFacil2 =
".193B07A2485.AB435289671.5281694A03B.B47610235A99610A45382B7235A78B96104B4390A6
517281765928B034AA80243175B9649A15730B862508B29467A1332768BA14950";;
```

**Prueba 1:** resolverSudokudeManeraFacil (1, , tableroPruebaFacil1);;

**Prueba 2:** resolverSudokudeManeraFacil (1, , tableroPruebaFacil2);;

**Código:**

```
let rec resolverSudokudeManeraFacil (indice, strTableroFinal, strTableroInicial) =
if (indice > 144) then strTableroFinal
else if ((elementoparaIndice 1 indice (cadenahaciaListadeChar strTableroInicial))
= '.') then  let jugada = (frontBuscarElemento indice
 (cadenahaciaListadeChar strTableroInicial)) in  resolverSudokudeManeraFacil
 ((indice+1), strTableroFinal\^(char_for_read jugada),
 (fontTableroconJugada(indice, (char_for_read jugada), strTableroInicial)))
else resolverSudokudeManeraFacil ((indice+1),
 strTableroFinal\^(char_for_read(elementoparaIndice 1 indice
 (cadenahaciaListadeChar strTableroInicial))),strTableroInicial);;
```

## 2.6. Lógica para Resolver el Sudoku de Manera Díficil (Algoritmo de BACKTRAKING)

## 2.7. Función Principal

### 2.7.1. main

**Propósito:** Tratar todo el conjunto de "sub-funciones", para hacer "jugable.[el] Sudoku.

**Prototipo:** main : char list * string -> unit = <fun>

**Prueba 1:**

**Código:**

```
let rec main (lstTablero, strTablero) =   begin
        imprimirTablero lstTablero;
print_string "Estado del Juego: ";
if(sonlasFilaCorrectas (1,lstTablero) \&\& sonlasColumnasCorrectas (1, lstTablero)
&& sonlosCuadradosCorrectos (1, lstTablero)) = true then print_string "Correcto"
else print_string "Incorrecto";
print_newline();
print_string "1. Jugar";
print_newline();
                print_string "2. Resolver de manera fácil";
print_newline();
print_string "3. Resolver de manera difícil";
print_newline();
print_string "4. Salir";
print_newline();
                print_string "Por favor selecciona una opcion: ";
                   let opcion = read_int () in
   if (opcion = 1) then begin
   let tableroAux =
introJugada (strTablero,0,0,0,1) in main((cadenahaciaListadeChar tableroAux),
tableroAux);
        end
    else if (opcion = 2) then begin
let tableroFacil = resolverSudokudeManeraFacil (1, "", strTablero) in
main(cadenahaciaListadeChar(tableroFacil), tableroFacil)
              end
      else if (opcion = 3) then begin
    ()
              end
    else if (opcion = 4) then
                    ()
          end;;
```

## 2.8. Función de Ejecución

### 2.8.1. main(cadenahaciaListadeChar(tablero), tablero);;

**Propósito:** Ejecutar el Sudoku.

**Prototipo:**

**Prueba 1:**

**Código:**

```
(*
main(cadenahaciaListadeChar(tablero1), tablero1);;

main(cadenahaciaListadeChar(tablero2), tablero2);;

main(cadenahaciaListadeChar(tablero3), tablero3);;

main(cadenahaciaListadeChar(tablero4), tablero4);;
*)
```

# Capítulo 3

# Pruebas

# Bibliografía

[Mar10] Luis Bengochea Martínez. *Apuntes para la asignatura: Laboratorio de Programación Avanzada.* UAH, 2010.