Resolución de Sistemas de Restricciones de Tipos (Algoritmo de Unificación de Robinson)

Programación Funcional

1. Representación de Tipos

```
type variable = string;;

type tipo = Bool
          | Flecha of tipo * tipo
          | Var_tipo of variable;;
```

2. Variable en Tipo

Determina si una variable está contenida en un tipo (`variable -> tipo -> bool`)

```
let rec variable_en_tipo x = function
    Bool                -> false
  | Var_tipo y when x=y -> true
  | Var_tipo _          -> false
  | Flecha (s,t)        -> (variable_en_tipo x s) or
                           (variable_en_tipo x t);;
```

3. Representación de Sustituciones

```
type sustitucion = (variable * tipo) list;;

let sustitucion_vacia = [];;
```

Las sustituciones son listas de pares (x_i, t_i): `variable * tipo`, verificando las siguientes condiciones:

- Todos los x_i son distintos
- Ningún t_i hace referencia a algún x_i, es decir, no existen x_i y t_j tal que se cumpla `variable_en_tipo` x_i t_j

4. Sustitución de Variables

```
let rec sust_var sigma x =
  match sigma with
     []                  -> Var_tipo x
   | (y,t) :: _ when x=y -> t
   | _ :: sigma'         -> sust_var sigma' x;;
```

5. Sustitución de Tipos

```
let rec sust_tipo sigma = function
    Bool           -> Bool
  | Flecha (t1,t2) -> Flecha (sust_tipo sigma t1, sust_tipo sigma t2)
  | Var_tipo x     -> sust_var sigma x;;
```

6. Extensión de una Sustitución

Implementa $\{y \mapsto t\} \circ \sigma$

1. Se sustituyen las variables x_i de σ en t (t')
2. Se comprueba que en t' no aparezca y
   - Si $y = t'$ entonces no es necesario añadir $\{y \mapsto y\}$
   - Si $y \neq t'$ entonces se trata de una sustitución incorrecta (ciclo, ooops)
3. la nueva sustitución es la sustitución original, en la que se cambian todas las y de t_i por t', más el par (y, t')

```
exception Sust_recursiva;;

let rec ext_sust (y,t) sigma =
  let t' = sust_tipo sigma t
  in if t' = Var_tipo y
       then sigma                          (* trivial *)
       else if variable_en_tipo y t'
              then raise Sust_recursiva    (* ciclo *)
              else (y,t') :: List.map
                               (fun (xi,ti) -> (xi, sust_tipo [(y,t')] ti))
                               sigma;;
```

7. Unificación

Partiendo de una sustitución inicialmente vacía, `unif : sustitucion ->tipo * tipo ->sustitucion` determina la sustitución más general que unifica dos tipos.

```
exception Unificacion;;

let rec unif sigma = function
    (Bool,Bool)     -> sigma
  | (Flecha(t1,t2), Flecha(s1,s2)) -> unif (unif sigma (t1,s1)) (t2,s2)
  | (Var_tipo x, t) ->
        (match (sust_var sigma x) with
          Var_tipo y -> ext_sust (y,t) sigma
        | s          -> unif sigma (s, t))
  | (t, Var_tipo x) -> unif sigma (Var_tipo x, t)
  | _               -> raise Unificacion;;
```

8. Resolución Sistema de Restricciones de Tipos

```
let rec resolver_sistema_restricciones = function
     []    -> sustitucion_vacia
  |  e::es -> unif (resolver_sistema_restricciones es) e;;
```

9. Ejemplos

9.1.

```
let e1 = (Var_tipo "X", Bool);;
let e2 = (Var_tipo "Y", Flecha (Var_tipo "X", Var_tipo "X"));;

let s1 = resolver_sistema_restricciones [e1;e2];;
val s1 : sustitucion = ["X", Bool; "Y", Flecha (Bool, Bool)]

sust_tipo s1 (Var_tipo "X");;
- : tipo = Bool

sust_tipo s1 Bool;;
- : tipo = Bool

sust_tipo s1 (Var_tipo "Y");;
- : tipo = Flecha (Bool, Bool)

sust_tipo s1 (Flecha (Var_tipo "X", Var_tipo "X"));;
- : tipo = Flecha (Bool, Bool)
```

9.2.

```
let e3 = (Flecha(Bool,Bool), Flecha(Var_tipo "X", Var_tipo "Y"));;

let s2 = resolver_sistema_restricciones [e3];;
```

```
val s2 : sustitucion = ["X", Bool; "Y", Bool]

sust_tipo s2 (Flecha(Bool,Bool));;
- : tipo = Flecha (Bool, Bool)

sust_tipo s2 (Flecha(Var_tipo "X", Var_tipo "Y"));;
- : tipo = Flecha (Bool, Bool)
```

9.3.

```
let e5 = (Bool, Flecha(Bool, Var_tipo "Y"));;

let s4 = resolver_sistema_restricciones [e5];;
Uncaught exception: Unificacion
```

9.4.

```
let e6 = (Var_tipo "Y", Flecha(Bool, Var_tipo "Y"));;

let s5 = resolver_sistema_restricciones [e6];;
Uncaught exception: Sust_recursiva
```

9.5.

```
let e4 = (Flecha(Var_tipo "X", Var_tipo "Y"),
          Flecha(Var_tipo "Y", Var_tipo "Z"));;
let e5 = (Var_tipo "Z", Flecha (Var_tipo "U", Var_tipo "W"));;

let s3 = resolver_sistema_restricciones [e4;e5];;
val s3 : sustitucion =
  ["Y", Flecha (Var_tipo "U", Var_tipo "W");
   "X", Flecha (Var_tipo "U", Var_tipo "W");
   "Z", Flecha (Var_tipo "U", Var_tipo "W")]

sust_tipo s3 (Flecha(Var_tipo "X", Var_tipo "Y"));;
- : tipo =
   Flecha (Flecha (Var_tipo "U", Var_tipo "W"),
           Flecha (Var_tipo "U", Var_tipo "W"))

sust_tipo s3 (Flecha(Var_tipo "Y", Var_tipo "Z"));;
- : tipo =
   Flecha (Flecha (Var_tipo "U", Var_tipo "W"),
           Flecha (Var_tipo "U", Var_tipo "W"))
```

9.6.

```
let e7 = (Var_tipo "X", Flecha (Var_tipo "Z", Var_tipo "X2"));;
let e8 = (Var_tipo "Y", Flecha (Var_tipo "Z", Var_tipo "X3"));;
let e9 = (Var_tipo "X2", Flecha (Var_tipo "X3", Var_tipo "X1"));;

let s6 = resolver_sistema_restricciones [e7;e8;e9];;
val s6 : sustitucion =
  ["X", Flecha (Var_tipo "Z", Flecha (Var_tipo "X3", Var_tipo "X1"));
   "Y", Flecha (Var_tipo "Z", Var_tipo "X3");
   "X2", Flecha (Var_tipo "X3", Var_tipo "X1")]

List.map (fun (t,s) -> sust_tipo s6 t = sust_tipo s6 s) [e7;e8;e9];;
- : bool list = [true; true; true]
```