Corrigé feuille d'exercices n°16 de l'option d'informatique.

Exercice n°1

```
type constante = Vrai | Faux;;
type expression =
  const of constante
| var of string
| Et of expression * expression
| Ou of expression * expression
| Non of expression
| Implique of expression * expression
| Equiv of expression * expression
| Nor of expression * expression
| Xor of expression * expression;;
let expr1 = Implique(var "a",Equiv(const Vrai,var "b"));;

let rec transforme = function
  const(c) -> const(c)
| var(v) -> var(v)
| Et(a,b) -> Et(transforme(a),transforme(b))
| Ou(a,b) -> Ou(transforme(a),transforme(b))
| Non(a) -> Non(transforme(a))
| Implique(a,b) -> Ou(Non(transforme a),(transforme b))
| Equiv(a,b) -> let a' = transforme a and b' = transforme b in
                 Ou(Et(a',b'),Et(Non(a'),Non(b')))
| Nor(a,b) -> let a' = transforme a and b' = transforme b in
                  Non(Ou(a',b'))
| Xor(a,b) -> let a' = transforme a and b' = transforme b in
                  Ou(Et(a',Non(b')),Et(Non(a'),b'));;
```

Exercice n°2

```
let rec simplifie = function
 Et(a,b) -> (let a' = simplifie a and b' = simplifie b in
            match (a',b') with
             (const(Faux),_) -> const(Faux)
            | (_,const(Faux)) -> const(Faux)
            | (const(Vrai),b') -> b'
            | (a',const(Vrai)) -> a'
            | (a',b') when a'=b' -> a'
            | (a',Non(b'')) when a'=b'' -> const(Faux)
            | (Non(a''),b') when a''=b' -> const(Faux)
            | _ -> Et(a',b'))
| Ou(a,b) -> (let a' = simplifie a and b' = simplifie b in
            match (a',b') with
             (const(Faux),b') -> b'
            | (a',const(Faux)) -> a'
            | (const(Vrai),b') -> const(Vrai)
            | (a',const(Vrai)) -> const(Vrai)
            | (a',b') when a'=b' -> a'
            | (a',Non(b'')) when a'=b'' -> const(Vrai)
            | (Non(a''),b') when a''=b' -> const(Vrai)
            | _ -> Et(a',b'))
```

```
        | Non(a) -> (let a' = simplifie(a) in
50                    match a' with
                       const(Faux) -> const(Vrai)
52                    | const(Vrai) -> const(Faux)
                      |  _ -> Non(a'))
54      | const(c) -> const(c)
        | var(c) -> var(c)
56      | _ -> failwith "Cas non traité";;
```

Exercice n°3

```
        let rec Nor_of_expr expr =
58      let transf = function
          Non(a) -> Nor(Nor_of_expr(a),const(Faux))
60      | Et(a,b) -> Nor(Nor(Nor_of_expr(a),const(Faux)),Nor(Nor_of_expr(b),const(Faux)))
        | Ou(a,b) -> Nor(Nor(Nor_of_expr(a),Nor_of_expr(b)),const(Faux))
62      | const(c) -> const(c)
        | var(v) -> var(v)
64      | _ -> failwith "Nor_of_expr : Cas non traité" in
        transf(transforme expr);;
66
        Nor_of_expr(Implique(var("a"),var("b")));;
68
        let rec union = fun
70        [] l2 -> l2
        | l1 [] -> l1
72      | (x::xs) (y::ys) when x=y -> (* cas où les 2 têtes de listes sont égales *)
                                        x::(union xs ys)
74      | (x::xs as l1) (y::ys as l2) -> if x < y then x::(union xs l2)
                                                  else y::(union l1 ys);;
```

Exercice n°4

```
76      let rec variable = function
          const(_) -> []
78      | var(a) -> [a]
        | Et(a,b) -> let l1 = variable a and l2 = variable b in
80                     union l1 l2
        | Ou(a,b) -> let l1 = variable a and l2 = variable b in
82                     union l1 l2
        | Non(a) -> variable a
84      | Implique(a,b) -> let l1 = variable a and l2 = variable b in
                       union l1 l2
86      | Equiv(a,b) -> let l1 = variable a and l2 = variable b in
                       union l1 l2
88      | Xor(a,b) -> let l1 = variable a and l2 = variable b in
                       union l1 l2
90      | Nor(a,b) -> let l1 = variable a and l2 = variable b in
                       union l1 l2;;
```

Exercice n°5

```
92      let rec eval = function
         const(Vrai) -> true
94      | const(Faux) -> false
        | var(c) -> failwith "eval : impossibilité d'évaluer l'expression"
96      | Et(a,b) -> (eval a) && (eval b)
        | Ou(a,b) -> (eval a) || (eval b)
98      | Non(a) -> not(eval(a))
        | Implique(a,b) -> not(eval a) || (eval b)
100     | Equiv(a,b) -> let a' = eval a and b' = eval b in
                          (a' && b') || (not(a') && not(b'))
102     | Xor(a,b) -> let a' = eval a and b' = eval b in
                          (a' && (not b')) || ((not a') && b')
104     | Nor(a,b) -> let a' = eval a and b' = eval b in
                          (not a') && (not b');;
```

La fonction "`remplace`" permet de remplacer dans une expression "`expr`" toutes les variables de nom "`vari`" par la constante booléenne "`valeur`".

```
106   let rec remplace expr vari valeur =
        match expr with
108      const(c) -> const(c)
        | var(c) when c = vari -> const(valeur)
110     | var(c) -> var(c)
        | Et(a,b) -> Et(remplace a vari valeur, remplace b vari valeur)
112     | Ou(a,b) -> Ou(remplace a vari valeur, remplace b vari valeur)
        | Non(a) -> Non(remplace a vari valeur)
114     | Implique(a,b) ->  Implique(remplace a vari valeur, remplace b vari valeur)
        | Equiv(a,b) ->  Equiv(remplace a vari valeur, remplace b vari valeur)
116     | Xor(a,b) ->  Xor(remplace a vari valeur, remplace b vari valeur)
        | Nor(a,b) ->  Nor(remplace a vari valeur, remplace b vari valeur);;
```

Exercice n°6

```
118   let tautologie expr =
        let rec tautol vars expr
120       match vars with
         (* On remplace "x" par vrai et par faux dans l'arbre et on regarde si on
122            a une tautologie dans les deux cas *)
             x::xs -> let a1 = remplace expr x Vrai
124                       and a2 = remplace expr x Faux in
                     (tautol xs a1) & (tautol xs a2)
126          (* Quand il n'y a plus de variable, on évalue l'arbre logique obtenu *)
             |  [] -> eval expr in
128     let vars = variable expr in
             tautol vars expr;;
```