```
type arbre_logique = Vide
        |  Feuille_bool of bool
        |  Feuille_var of string
        |  Neg of arbre_logique
        |  Disj of arbre_logique*arbre_logique (* disjonction *)
        |  Conj of arbre_logique*arbre_logique  (* conjonction *)
        |  DisjEx of arbre_logique*arbre_logique (* disjonction exclusive *)
        |  Implication of arbre_logique*arbre_logique (* implication *)
        |  Equiv of arbre_logique*arbre_logique;;(* équivalence logique *)

let supprime_operateurs_auxiliaires =
     let rec sox = function
         |  Implication(arbre_g,arbre_d) -> Disj(Neg(sox arbre_g),sox arbre_d)
         |  DisjEx(arbre_g,arbre_d) ->
               let trans_g = sox arbre_g and trans_d = sox arbre_d in
                    Disj(Conj(Neg(trans_g),trans_d),Conj(trans_g,Neg(trans_d))
         |  Equiv(arbre_g,arbre_d) ->
               let trans_g = sox arbre_g and trans_d = sox arbre_d in
                    Disj(Conj(trans_g,trans_d),Conj(Neg(trans_g),Neg(trans_d))
         |  Disj(arbre_g,arbre_d) -> Disj(sox(arbre_g),sox(arbre_d))
            |  Conj(arbre_g,arbre_d) -> Conj(sox(arbre_g),sox(arbre_d))
         |  Neg(arbre) -> Neg(sox(arbre))
         |  feuille -> feuille
      in sox;;

let rec repousse_non = function
           | Neg(Feuille_bool x) -> Feuille_bool(not x)
           | Neg(Neg(arbre)) -> repousse_non arbre
           | Neg(Disj(arbre_g,arbre_d)) ->
                Conj(repousse_non(Neg(arbre_g)),repousse_non(Neg(arbre_d)))
           | Neg(Conj(arbre_g,arbre_d)) ->
                Disj(repousse_non(Neg(arbre_g)),repousse_non(Neg(arbre_d)))
           | Conj(arbre_g,arbre_d) ->
                Conj(repousse_non(arbre_g),repousse_non(arbre_d))
```

```
          | Disj(arbre_g,arbre_d) ->
                Disj(repousse_non(arbre_g),repousse_non(arbre_d))
          | autre -> autre;;

let distribue_et a =
      let rec dist = function
           | Conj(Disj(arbre_1,arbre_2),arbre_3) ->
                 Disj(dist(Conj(arbre_1,arbre_3)),dist(Conj(arbre_2,arbre_3
           | Conj(arbre_1,Disj(arbre_2,arbre_3)) ->
                 Disj(dist(Conj(arbre_1,arbre_2)),dist(Conj(arbre_1,arbre_3
           | Conj(arbre_g,arbre_d) ->Conj(dist(arbre_g),dist(arbre_d))
           | Disj(arbre_g,arbre_d) ->Disj(dist(arbre_g),dist(arbre_d))
           | arbre -> arbre
      and itere_dist a0 a1 =
           if a0 = a1 then a1 else itere_dist a1 (dist a1)
      in itere_dist a (dist a);;

let forme_disj = function
                a -> distribue_et(repousse_non(
                                      supprime_operateurs_auxiliaires a
                                ));;

let disj_en_liste a =
     let rec gere_disj = function
          |  Disj(arbre_g,arbre_d) -> (gere_disj arbre_g)@(gere_disj arbre_d)
          |  arbre -> [arbre]
     and gere_conj = function
          |  Conj(arbre_g,arbre_d) -> (gere_conj arbre_g)@(gere_conj arbre_d)
          |  arbre -> [arbre]
     in map gere_conj (gere_disj a);;

let supprime_rep =
        let rec sans_rep = function
              |  [] -> []
```

```
                |  h::r ->
                     if mem h r
                        then sans_rep r
                        else h::sans_rep r
        in
        let tri = sort__sort (prefix <) in
        function disj -> sans_rep (map tri (map sans_rep disj));;

let supprime_triv =
        let rec est_triv = function
                |  [] -> false
                |  (Feuille_var s)::r -> (mem (Neg(Feuille_var s)) r) or est_triv
                |  Neg(Feuille_var s)::r -> (mem (Feuille_var s) r) or est_triv r
                |  (Feuille_bool false)::r -> true
                |  _ -> failwith "xxx" (* cas impossible *)
        in map (function x -> if est_triv x then [] else x);;

#let a=analyse_logique (lexeur_logique "(a et b impl (c equiv d)) et a");;
a : arbre_logique =
 Conj
  (Implication
    (Conj (Feuille_var "a", Feuille_var "b"),
     Equiv (Feuille_var "c", Feuille_var "d")),
   Feuille_var "a")
#let d=disj_en_liste(forme_disj a);;
d : arbre_logique list list =
 [[Neg (Feuille_var "a"); Feuille_var "a"];
  [Neg (Feuille_var "b"); Feuille_var "a"];
  [Feuille_var "c"; Feuille_var "d"; Feuille_var "a"];
  [Neg (Feuille_var "c"); Neg (Feuille_var "d"); Feuille_var "a"]]
#supprime_triv(supprime_rep d);;
- : arbre_logique list list =
 [[]; [Feuille_var "a"; Neg (Feuille_var "b")];
  [Feuille_var "a"; Feuille_var "c"; Feuille_var "d"];
```

```
[Feuille_var "a"; Neg (Feuille_var "c"); Neg (Feuille_var "d")]]
```