```
(**************************************************************************)
(*          Tautologies propositionnelles par les IF-expressions          *)
(**************************************************************************)

(* Le type des propositions : *)

type proposition_atomique == string;;

type proposition =
    Atome of proposition_atomique
  | Vrai
  | Faux
  | Neg of proposition
  | Imp of proposition * proposition
  | Et  of proposition * proposition
  | Ou  of proposition * proposition
;;

(* IF-expressions *)

type IFexpr =
    Var of proposition_atomique
  | Vr
  | Fx
  | If of IFexpr * IFexpr * IFexpr
;;

(* Transformation des propositions en IF-expressions *)

let rec prop_if = function
    Atome a   -> Var a
  | Vrai      -> Vr
  | Faux      -> Fx
  | Neg p     -> If (prop_if p, Fx, Vr)
  | Imp (p,q) -> If (prop_if p, prop_if q, Vr)
  | Et  (p,q) -> If (prop_if p, prop_if q, Fx)
  | Ou  (p,q) -> If (prop_if p, Vr, prop_if q)
;;

(* Exemples : *)

let A = Atome "A" and B = Atome "B";;
let ex1 = Imp (Et (A,B), A);;
let ex2 = Imp (A, Et (A,B));;
let ex3 = Imp (A, Imp (Imp (A,B), B));;

let if1 = prop_if ex1;;
let if2 = prop_if ex2;;
let if3 = prop_if ex3;;

(* Une IF-expression est-elle normale ? *)

let rec est_normale = function
    Var _ | Vr | Fx  -> true
  | If (Var _, p, q) -> (est_normale p) & (est_normale q)
  | _                -> false
;;

(* Normalisation des IF-expressions *)

let rec normalise = function
    If (Vr, p, _)         -> normalise p
  | If (Fx, _, q)         -> normalise q
```

```
  | If (Var s, p, q)      -> If (Var s, normalise p, normalise q)
  | If (If (x,y,z), p, q) -> normalise (If (x, If (y,p,q), If (z,p,q)))
  | x -> x
;;

(* Exemples : *)

let ifn1 = normalise if1;;
let ifn2 = normalise if2;;
let ifn3 = normalise if3;;

(* Assignations (partielles) *)

type assignation == (string * bool) list;;

type resultat =
    Tautologie
  | Refutation of assignation
;;

let rec decision_partielle assign = function
    Vr    -> Tautologie

  | Fx    -> Refutation assign

  | Var x ->
      if mem_assoc x assign then
        if assoc x assign then Tautologie else Refutation assign
      else
        Refutation ((x,false)::assign)

  | If (Var x, p, q) ->
      if mem_assoc x assign then
        if assoc x assign then
          decision_partielle assign p
        else
          decision_partielle assign q
      else
        (match decision_partielle ((x,true)::assign) p with
             Tautologie -> decision_partielle ((x,false)::assign) q
           | r -> r)

  | _ -> failwith "IF-expression non en forme normale"
;;

let decision_if f = decision_partielle [] f;;

(* Exemples : *)

decision_if ifn1;;
decision_if ifn2;;
decision_if ifn3;;

(* Decision propositionnelle *)

let decision_prop p =
  let ife = prop_if p in
  let ifn = normalise ife in
  decision_partielle [] ifn
;;

(* L'Aquarium *)
```

```
let MC = Atome "MC"  (* nage en mer chaude *)
and RR = Atome "RR"  (* a des rayures rouges *)
and NB = Atome "NB"  (* a des nageoires bleues *)
and VC = Atome "VC"  (* vit dans le corail *)
and C = Atome "C";;  (* mange des crevettes *)

let r1 = Imp (Neg MC, RR)
and r2 = Ou (NB, Neg RR)
and r3 = Imp (VC, Neg C)
and r4 = Et (Imp (C, MC) , Imp (MC,C))
and r5 = Imp (NB, Et (MC, VC))
and r6 = Imp (MC, NB) ;;

let pas_de_poisson =
  Imp (r1 , Imp (r2, Imp(r3 ,Imp (r4, Imp(r5, Imp (r6, Faux)))))) ;;

(* #decision_prop pas_de_poisson;;
   - : resultat = Tautologie       *)


(* Une mesure justifiant la terminaison de la fonction "normalise" est
   la suivante :

     m(Var _) = m(Vr) = m(Fx) = 1
     m(If(X,Y,Z)) = m(X) . (1 + m(Y) + m(Z))

 *)
```