Code

Test arbre binaire de recherche

Pour vérifier si un arbre binaire est bien " de recherche", il faut vérifier **pour tout noeud** que l'étiquette de celui-ci est plus grande (strictement) que celles des noeuds du sous-arbre gauche, et [etc.], et que chaque sous-arbre est également un ABR. C'est ici fait récursivement. Un noeud dans ce cadre est soit une feuille "vide", soit contient une étiquette et possède deux fils. Ceci permet de traiter l'ensemble des arbres binaires, même non stricts (par l'application qui retire ses feuilles à un arbre)

```
type arbre = Nil | Noeud of int*arbre*arbre;;

let testABR arbre =
let rec aux a = match a with
Nil->(0,0,true)
|Noeud(v,Nil, Nil)->(v,v,true)
|Noeud(v,Nil,a)->let (min,max,test)= aux a in (v,max,test&&(min>=v ))
|Noeud(v,a,Nil)->let (min,max,test)= aux a in (min,v,test&&(max<v))
|Noeud(v,a,b)->let (min_g,max_g,test_g)= aux a and (min_d,max_d,test_d) = aux b in
(min_g,max_d, test_g&&test_d&&(max_g<v)&&(min_d>=v))
in
let (min, max,test) = aux arbre in test;;
```

Génération code Gray

J'ai choisi de le faire sous forme de liste de listes pour la clarté et la concision du code demandé ; un autre choix est bien sûr possible. Représenter une valuation sous la forme de vecteur serait plus confortable pour la suite.

```
let rec ajout i l = match l with
[]->[]
|a::s-> (i::a)::(ajout i s);;

let rec miroir liste = match liste with
[]->[]
|a::s-> (miroir s)@[a];;

let rec enum n =
if n==1 then [[0];[1]]
else let liste = enum(n-1) in
(ajout 0 liste)@(ajout 1 (miroir liste));;

# enum 4;;
- : int list list =
[[0; 0; 0; 0]; [0; 0; 0; 1]; [0; 0; 1; 1]; [0; 0; 1; 0]; [0; 1; 1; 0];
 [0; 1; 1; 1]; [0; 1; 0; 1]; [0; 1; 0; 0]; [1; 1; 0; 0]; [1; 1; 0; 1];
 [1; 1; 1; 1]; [1; 1; 1; 0]; [1; 0; 1; 0]; [1; 0; 1; 1]; [1; 0; 0; 1]; [1; 0; 0; 0]]
```