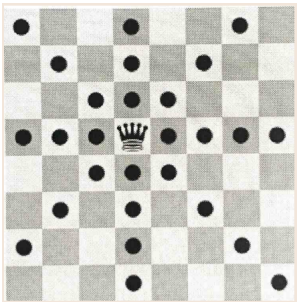
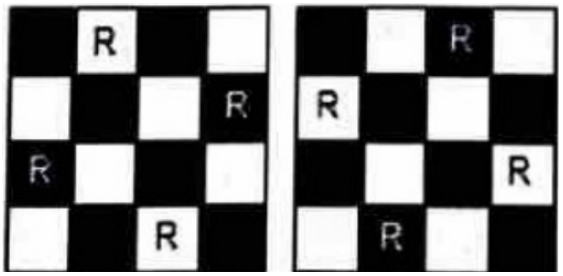
Back Tracking : solution problème des N reines

Conseil : relire polycopié PROJET BACK TRACKING (Problème des n reines)

Projet réalisé par : Iba Kengaye, Arthur Durand, Bastien Dufour, Antoine Renouard



Les cases mises en danger par une reine

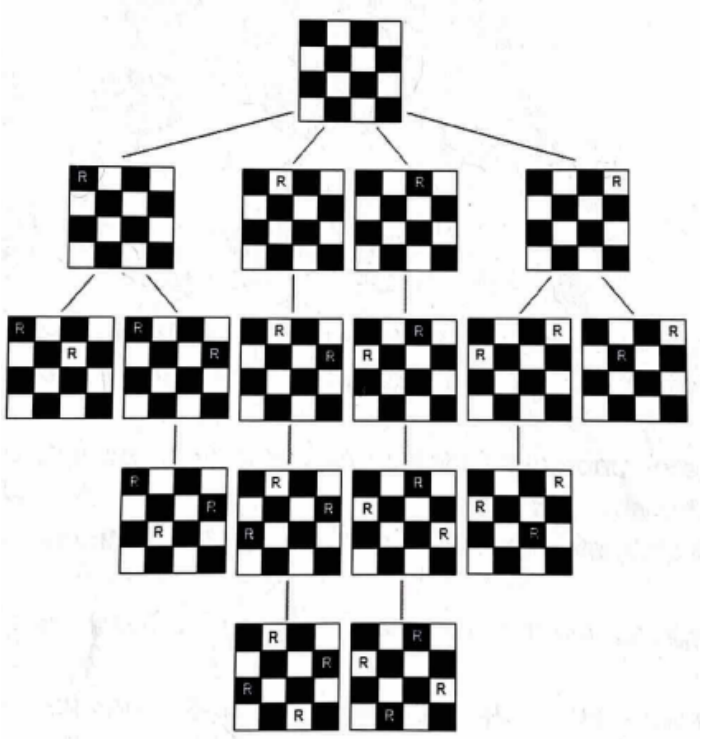


Exemple des solutions avec un échiquier 4x4

Objectifs:

- déterminer si à partir d'un échiquier partiellement rempli, il existe des solutions au problème
- déterminer le nombre de solutions en fonctions de la taille de l'échiquier

Exemple pour n = 4, le nombre de solutions est 2 :



Utilisation du langage Ocaml :

Pourquoi ? → algorithme récursif donc adapté au langage.

Pour déterminer si à partir d'un échiquier donné, il existe des solutions au problème, on met en place une structure de type solution:

```
type solution = {reines_a_poser: int; reines_posees: int array};;
(* reines_a_poser : nombre de reines à poser sur l'échiquier = taille de l'échiquier
   reines_posées : colonne sur laquelle la i-ème reine est posée*)
```

Dès lors, il convient d'initialiser cette structure pour l'appliquer au problème

```
let initialiser n =
   (*on crée la structure qui nous donne la taille de l'échiquier et les solutions*)
   let structure = {reines_a_poser = n ; reines_posees = Array.make n (-1)} in
   structure;;
```

:
ic si n = 4 ,
initialiser 4 :
structure = { 4 ; [| -1; -1; -1; -1 |] }

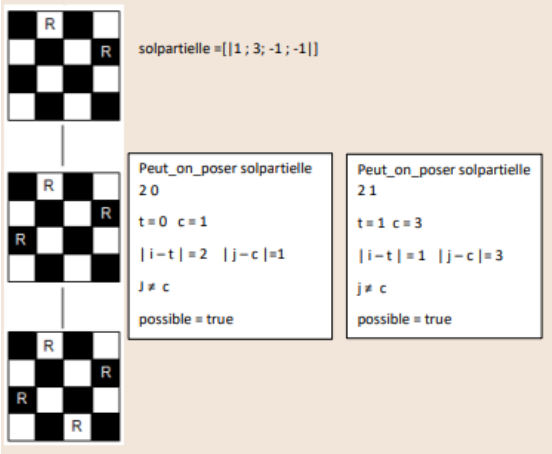
Ensuite, nous voulons créer une fonction testant s'il est possible de poser une reine en une certaine case de coordonnées x y.
Pour ce faire, nous avons besoin d'une fonction valeur absolue qui permettra de contrôler si une case est menacée par une reine située dans les deux diagonales:

```
let abs_val value =
  if value < 0 then
    -value
  else
    value;;
```

et donc la fonction peut_on_poser :

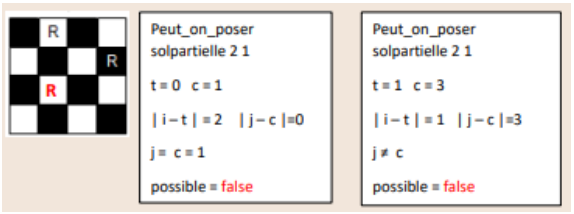
```
let  peut_on_poser solpartielle i j =
(*la fonction renvoie true si la case (i,j) n'est pas menacée, false sinon*)
(* Entrées:
   solpartielle : un tableau d'entiers contenant les positions des reines
   i : ligne sur laquelle on veut poser la reine
   j : colonne sur laquelle on veut poser la reine
   Sortie : Booléen
   *)
   let  possible = ref true in
   for t = 0 to (i-1) do
      let c = solpartielle .(t) in
      (*on vérifie que l'on ne place pas dans la diagnonale ou la colonne d'une reine déja placée*)
      if (abs_val (i-t) = abs_val (j-c) )(*diagonale*) || (j = c)(*colonne*) then begin
      (*si une des conditions est vérifiée : la case sur laquelle on veut poser une reine est menacée*)
      (*si on ne prend pas la valeur absolue de i-t et j-c on risque d'avoir des nombres négatifs
      donc hors de l'échiquier *)
         possible := false
         end;
      done;
   possible ;;
```

Explication peut_on_poser : échiquier de taille 4 x 4 déjà rempli à moitié

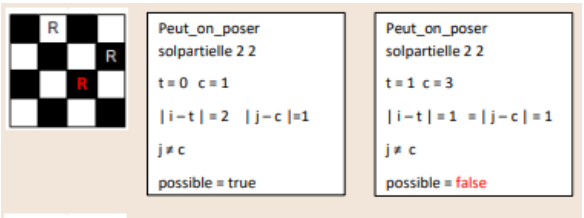


Test sur la 2e case à la 3e ligne (2 ;1)

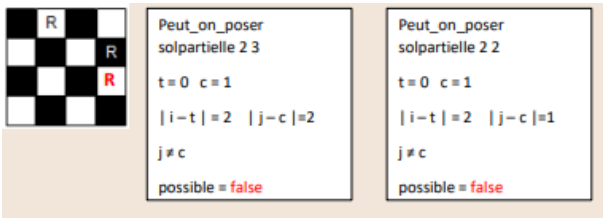
↓



Test sur la 3e case à la 3e ligne (2 ;2)



Test sur la 4e case à la 3e ligne (2 ;3)



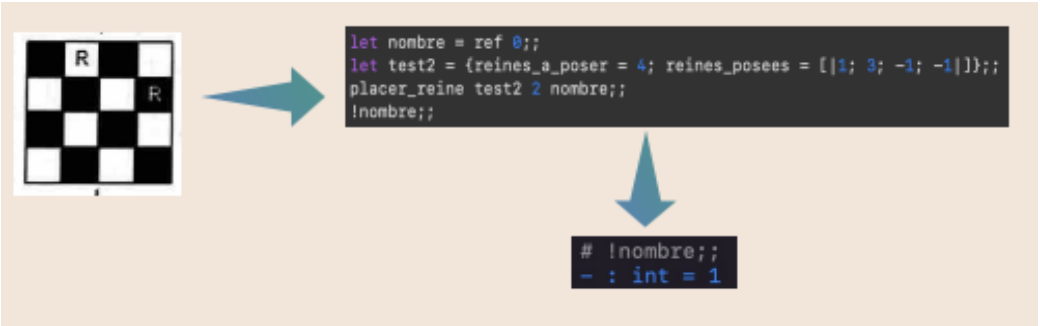
Ici seul le premier cas a permis le placement d'une reine, on observe d'ailleurs que ce placement entraine une solution au problème des n reines où n = 4.

Enfin il nous reste la fonction placer_reine qui va employer la méthode du backtracking :

```
let rec placer_reine structure ligne solution =
(*la fonction incrémente le paramètre solution qui correspond aux nombres de solutions du problème*)
(* Entrées:
   structure : une variable de type solution qui est une solution partielle du problème
   ligne: ligne sur laquelle on veut poser la reine
   solution : un int ref qui donne le nombre de solutions finales
   *)
   match ligne with
   (*on est arrivé à une feuille donc on à trouvé une solution*)
   |n when ligne >= structure.reines_a_poser -> incr solution;
                                                for i = 0 to ((Array.length structure.reines_posees) - 1) do
                                                   print_int structure.reines_posees.(i);
                                                done;
                                                print_newline ();
                                                print_newline ();

   |_ -> for colonne = 0 to  (structure.reines_a_poser - 1) do

         let avancer = peut_on_poser structure.reines_posees ligne colonne in
         (*la fonction peut_on_poser renvoie un ref booléen*)

         if !avancer = true then begin
            (*il est possible de placer une reine donc on continue dans l'arbre*)
            structure.reines_posees.(ligne) <- colonne;
            (*appel récursif*)
            placer_reine structure (ligne + 1) solution;
            (*on remonte dans l'arbre (principe du backtracking)*)
            structure.reines_posees.(ligne) <- (-1);
         end;

      done;;
```

L'invariant proposé est : À l'appel p il y a p reines posées (démonstration triviale)

Observation :



ce qui correspond bien à ce que nous avions trouvé lorsque nous expliquions peut_on_poser .

Complexité :

$$T(n) \leq \begin{cases} \Theta(1) \text{ si } n=1 \\ \Theta(1)+n(\Theta(1)+T(n-1)) \text{ sinon} \end{cases}$$

$$T(n) \leq \begin{cases} \Theta(1) \text{ si } n=1 \\ \Theta(n)+nT(n-1) \text{ sinon} \end{cases}$$

$$T(n) \leq n!T(1)+\sum(n!/(n-k)!)\Theta(n-k) \leq \Theta(n!)$$

$$T(n)=O(n!)$$