

# **Rapport de Projet d'Analyse d'algorithmes et validation de programmes**

Encadré par : Devan Sohier

Institut en Sciences et Techniques des Yvelines (ISTY)

Thème : Problème des N-Reines (N-Queens Problem)

Étudiant : Amayes DJERMOUNE

Classe : IATIC 4

March 10, 2024

# Sommaire

Amayes Djermoune

March 10, 2024

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Présentation du Problème . . . . .	2
1.2	Motivation . . . . .	2
1.3	Problématique et Contraintes . . . . .	2
1.4	Description des types de données manipulées . . . . .	3
<b>2</b>	<b>Implémentation de solution au Problème</b>	<b>4</b>
2.1	Solution Récursive . . . . .	4
2.2	Dérécursivation . . . . .	5
2.3	Solution Itérative . . . . .	6
2.4	Validation et Scénarios de tests . . . . .	7
<b>3</b>	<b>Analyse de complexité algorithmique</b>	<b>8</b>
3.1	Solution Récursive . . . . .	8
3.2	Solution Itérative . . . . .	9
3.3	Complexité en Moyenne . . . . .	9
<b>4</b>	<b>Preuve de programme</b>	<b>9</b>
4.1	Spécifications du problème . . . . .	9
4.2	Invariant de Boucle . . . . .	10
4.3	Démonstration . . . . .	10
<b>5</b>	<b>Conclusion</b>	<b>10</b>
<b>6</b>	<b>Références Bibliographiques</b>	<b>10</b>
<b>7</b>	<b>Annexe : Démonstration</b>	<b>11</b>

# 1 Introduction

## 1.1 Présentation du Problème

Pour mettre un peu de contexte, le problème des N-Reines fait partie des problèmes NP-complet les plus connus dans le domaine de l'informatique. Posé par le joueur d'échecs allemand Max Bezzel en 1848 pour 8 Reines, le problème consiste à placer N reines sur un échiquier de taille  $N \times N$  de sorte à ce qu'aucune reine ne puisse attaquer une autre en un seul coup.

## 1.2 Motivation

Mettant de coté ma passion pour les jeux d'échecs, mon choix d'étude du problème des N-Reines est motivé par divers aspects fascinants et applicatifs, à savoir :

- Représentation modeste et puissante : Le prestige principal qu'offre ce problème est la simplicité et la richesse de sa représentation algorithmique, contribuant explicitement au développement d'un nouveau terrain d'exploration de divers techniques de résolution algorithmique.
- Application en Intelligence Artificielle : Les techniques utilisées pour résoudre le problème des N-queens, telles que le backtracking et les algorithmes de recherche, sont pertinentes dans le domaine de l'intelligence artificielle. La résolution de problèmes similaires peut être rencontrée dans la planification automatique et d'autres domaines de l'IA.
- Problèmes NP-complets : Le problème en lui même fait partie de la famille des problèmes NP-complet. Ainsi, étudier ce problème projetera la lumière sur les défis et techniques utilisés pour la résolution de ce genre de problèmes.
- Algorithmes de recherche et backtracking : À vue d'oeil, ce genre de problèmes demande l'implémentation d'algorithmes principalement basées sur la recherche, voir du Backtracking, ce qui est en soit idéal pour aiguiser ses capacités de raisonnement et résolution de problèmes informatiques .

## 1.3 Problématique et Contraintes

- Entrée : Un échiquier de taille  $N \times N$  et N Reines.
- Sortie : Un arrangement des reines sur l'échiquier de façon à ce qu'aucune reine ne puisse attaquer une autre.

Il est à garder en tête qu'il existe des contraintes strictes à respecter. Aucune paire de reines ne doit se trouver sur la même ligne, colonne ou diagonale, ce qui est en soi un défi loin d'être réservé à des enfants de choeur :

- Recherche d'une solution : Trouver une solution valide pour un  $N$  donné est loin d'être un cadeau. L'idée la plus pertinente consiste à exploiter toutes les possibilités et combinaisons , tout en considérant les marche arrières (Backtracking ).
- Optimisation : Pour des valeurs élevées de  $N$ , la solution optimale devient plus difficile à retrouver. L'objectif est souvent de trouver une solution valide plutôt que la seule solution possible, surtout pour des tailles d'échiquier plus grandes.
- Complexité algorithmique : Le problème des  $N$ -queens est classé comme NP-complet. Cela signifie par extension qu'il n'existe pas (encore) d'algorithme polynomial connu pour le résoudre de manière générale, bien que des solutions puissent être trouvées efficacement pour des valeurs spécifiques de  $N$ .

#### 1.4 Description des types de données manipulées

- **board (échiquier)** : Une matrice de taille  $N \times N$  ( $N$  lignes,  $N$  colonnes) pour représenter l'échiquier. Chaque élément de la matrice peut prendre les valeurs 0 (case vide) ou 1 (présence d'une reine).
- **row (ligne)** : Variable entière représentant l'indice de ligne actuel lors de la recherche de solutions.
- **col (colonne)** : Variable entière représentant l'indice de colonne actuel lors de la recherche de solutions.
- **$N$**  : Variable entière représentant la taille de l'échiquier (nombre de lignes et de colonnes).
- **is\_safe(board, row, col, N)** : Fonction qui vérifie si une reine peut être placée en position (row, col) sur l'échiquier sans menacer les autres reines.
- **solve\_nqueens\_util(board, col, N)** : Fonction récursive qui résout le problème en plaçant les reines de manière récursive tout en respectant les règles du jeu.
- **solve\_nqueens\_iterative(N)** : Fonction itérative qui résout le problème en utilisant une approche basée sur une pile pour gérer l'exploration des options et le backtracking.
- **print\_solution(board)** : Fonction qui affiche une solution trouvée sur l'échiquier.

En terme de structures de données manipulées, les éventuelles structures utilisés dans notre problèmes sont piochées de sorte à proposer une implémentation modeste mais puissante en terme de recherche de solutions, à savoir : Les vecteurs, matrices, tuples et bien sur les piles.

## 2 Implémentation de solution au Problème

### 2.1 Solution Récursive

L'idée principale de la solution récursive est de placer une reine dans chaque colonne, en vérifiant si elle peut être placée en respectant les contraintes du problème.

- Cas de Base : Si je réussis à placer toutes les reines avec succès dans les colonnes précédentes (i.e., Après exploration de toutes les lignes pour chaque colonne), un configuration valide voit le jour (Approuvée par Garry Kasparov : ) , et nous pouvons donc renvoyer cette solution.
- Pour chaque ligne dans la colonne actuelle : Je vérifie si une reine peut être placée à la position courante (ligne, colonne).
- Si la position est sûre, je place la reine et je passe à la colonne suivante avec un appel récursif.
- Si l'appel récursif retourne une plutôt solution valide, on peut renvoyer cette solution.
- Si la position n'est pas sûre ou si l'appel récursif ne trouve pas de solution, je retire la reine de la position et j'explore d'autres positions en continuant le processus de backtracking.
- Si aucune solution n'est trouvée après avoir exploré toutes les possibilités, renvoyer qu'aucune solution n'existe (-1 ).
- Bien sur, il faut s'assurer qu'en sortie, on a bien et bel une configuration qui respecte les contraintes du problème, lorsque toutes les reines ont été posées dans l'échiquier en question.

```
fonction solve_nqueens(echiquier, col, N) -> configuration :
    si col >= N:
        renvoyer une solution valide
    pour chaque ligne dans la colonne actuelle:
        si je peux placer la reine a la position (ligne, col):
            je place la reine dans la pos (ligne, col)
            si solve_nqueens(board, col + 1, N) est valide :
                renvoyer cette solution
            retirer la reine de la position (ligne, col)
    renvoyer -1
```

## 2.2 Dérécursivation

L'idée principale derrière le processus de dérécursivation est de transformer notre algorithme récursif à un algorithme dit itératif. Divers raisons se cachent derrière cette transition. Dans un premier temps, c'est pas tous les langages de programmations qui offrent ce prestige de récursivité. Autre chose, bien que cette dernière soit mieux intuitive et facile à comprendre, elle est en conséquence assez coûteuses en terme de complexité algorithmique (ce qui n'est pas agréable à voir).

Voici les étapes à prendre en considération :

- **Initialisation de la pile :** On crée une pile pour stocker les informations d'appel de fonction. Dans ce cas, chaque élément de la pile représente une structure `Appel` contenant les informations nécessaires (ligne, colonne, étape).
- **Premier appel :** On empile le premier appel initial avec les paramètres appropriés (ici, ligne=0, colonne=0, étape=0).
- **Boucle principale :** On utilise une boucle `tant que` pour simuler l'exécution de la fonction récursive. La condition de sortie de la boucle est lorsque la pile est vide (Logique).
- **Récupération de l'appel actuel :** À chaque itération de la boucle, je récupère l'élément en haut de la pile pour obtenir les informations sur l'appel actuel.
- **Traitements en fonction de l'étape :** On examine la valeur de l'étape de l'appel actuel pour déterminer la prochaine action à effectuer.
  - Si notre étape est 0, cela signifie que l'algorithme est dans la phase de placement de la reine.
    - \* Si la colonne dépasse la taille N, On affiche la solution, je marque le label de reprise de programme et je dépile.
    - \* Sinon, on vérifie les conditions pour placer la reine, on mets notre pile à jour, on marque le label de reprise pour passer à la colonne suivante ou à la ligne suivante si nécessaire.
  - Si notre étape est 1, cela signifie que l'algorithme est dans la phase de retrait de la reine. Je retire ma reine, je marque le label de reprise, et je dépile.
- **Affichage des solutions :** À la fin de la boucle principale, toutes les solutions ont été explorées, et les résultats peuvent être affichés.

```

structure Appel:
    ligne
    col
    etape # Pour suivre l'etape actuelle

fonction solve_nqueens_iteratif(N):
    pile = creer_pile()
    empiler(pile, Appel(0, 0, 0)) # Initialiser la pile

    tant_QUE non est_vide(pile):
        appel = sommet(pile)

        si appel.etape == 0:
            si appel.col >= N:
                afficher_solution(appel.board)
                depiler(pile)
            sinon:
                si appel.ligne < N:
                    si position_valide(appel.board, appel.ligne, appel.col):
                        placer_reine(appel.board, appel.ligne, appel.col)
                        empiler(pile, Appel(appel.ligne + 1, appel.col, 0))
# Aller a la colonne suivante
                sinon:
                    empiler(pile, Appel(appel.ligne + 1, appel.col, 0))
# Aller a la ligne suivante
                sinon:
                    retirer_reine(appel.board, appel.ligne, appel.col)
                    depiler(pile)
                sinon:
                    retirer_reine(appel.board, appel.ligne, appel.col)
                    depiler(pile)

# Toutes les solutions ont ete explorees.

```

### 2.3 Solution Itérative

En dérécurvant d'avantage l'algorithme récursive (Suppression des instructions inutiles.....), on obtient une solution itérative, ou L'idée est d'utiliser une approche basée sur une pile pour explorer de manière itérative toutes les configurations possibles du problème des N-Queens.

- Initialisation : On pense à initialiser une pile pour stocker les positions des reines (ligne, col). Tout en mettant les indices de colonne (col) et de ligne (ligne) à 0.
- Boucle Principale :

Commencez une boucle infinie pour explorer continuellement les configurations du plateau. Vérifiez si col est supérieur ou égal à N. Si vrai, cela signifie qu'une configuration valide a été trouvée. Affichez la configuration actuelle et effectuez le backtracking.

- Validation de Position :

Vérifiez si la position actuelle est valide par rapport aux positions déjà placées dans la pile.

- Empilement ou Backtracking :

Si la position est valide, empilez la position sur la pile, passez à la colonne suivante ( $col += 1$ ), et initialisez la ligne (row) à 0. Si la position n'est pas valide, passez à la ligne suivante ( $row += 1$ ).

- Backtracking :

Si la pile n'est pas vide et qu'aucune solution n'a été trouvée, dépilez la position de la reine, décrémentez col, incrémentez row, et continuez la recherche.

- Sortie de la Boucle :

Si la pile est vide et qu'aucune solution n'a été trouvée, sortez de la boucle principale.

```
fonction solve_nqueens_iterative(N) -> configuration :
```

```

pile = []
ligne = col = 0
tant que vrai:
    si col >= N:
        afficher_configuration_actuelle(pile)
        (ligne, col) = pile.pop() si pile sinon sortir de la boucle
        col -- 1
        ligne += 1
    si est_position_valide(pile, ligne, col):
        empiler (ligne, col) sur la pile
        col += 1
        ligne = 0
    sinon:
        ligne += 1

```

## 2.4 Validation et Scénarios de tests

Pour tester le programme, nous pouvons utiliser plusieurs jeux de tests, en particulier pour les cas suivants :

1. Cas de Base :

- $N = 1$  : Un échiquier 1x1, le plus petit possible.
- $N = 2$  : Un échiquier 2x2, où aucune solution ne devrait être trouvée.

## 2. Cas Généraux :

- $N = 4$  : Un échiquier 4x4, où plusieurs solutions existent.
- $N = 8$  : Un échiquier 8x8, classique pour les problèmes des N-Queens.

## 3. Cas Limite :

- $N = 15$  : Un échiquier 15x15, proche de la limite pratique pour les algorithmes naïfs.

## 3 Analyse de complexité algorithmique

### 3.1 Solution Récursive

**Complexité Temporelle** : La complexité en temps de la solution récursive pour le problème des N-Queens est généralement exprimée en termes de factorielle,  $O(N!)$ . Cela est dû au fait que dans le pire des cas, toutes les configurations possibles doivent être explorées.

La relation de récurrence peut être exprimée comme suit :

$$T(N) = N \cdot T(N - 1)$$

où  $T(N)$  représente le temps nécessaire pour résoudre le problème pour une taille  $N$ .

Du coup, si on développe notre formule de récurrence, on aura :

$$T(N - 1) = N - 1 \cdot T(N - 2)$$

$$T(N - 2) = N - 2 \cdot T(N - 3)$$

Sachant que le cas de base est :

$$T(0) = 1$$

Du coup, en développant notre formule, on obtient :

$$T(N) = N \cdot (N - 1) \cdot (N - 2) \dots$$

Ce qui représente en effet la fonction factorielle, et ainsi :

$$T(N) = N!$$

### Complexité Spatiale :

- La complexité en espace est principalement due à la pile d'appels récursifs.
- Dans le pire cas, la profondeur de la pile d'appels est égale à  $N$  (nombre de colonnes).
- La complexité en espace est donc de l'ordre de  $O(N)$ .

## 3.2 Solution Itérative

**Complexité Temporelle :**

- La complexité temporelle est de l'ordre de  $O(N!)$ , où  $N$  est la taille de l'échiquier.
- Cette estimation est basée sur le fait que l'algorithme explore toutes les permutations possibles des reines sur l'échiquier (similairement à la version récursive).
- Chaque colonne a  $N$  options possibles, et la profondeur maximale de l'exploration est  $N$ .
- Par conséquent, la complexité est exponentielle en fonction de la taille de l'échiquier.

**Complexité Spatiale :**

- La complexité spatiale dépend de la taille de l'échiquier.
- L'algorithme utilise une matrice  $N \times N$  pour représenter l'échiquier.
- Il utilise également une pile qui peut atteindre une profondeur maximale de  $N$  lors de l'exploration.
- Par conséquent, la complexité spatiale est de l'ordre de  $O(N^2)$ .

## 3.3 Complexité en Moyenne

Dans notre problème, on ne peut pas explicitement parler de Complexité en Moyenne, étant donné que dans le cas récursif, tout comme le cas itératif, le mécanisme et mode de fonctionnement de l'algorithme reste inchangé : Je mets ma reine dans une position bien définie, puis j'explore les autres possibilités et je fais ainsi pour tout le reste de mon échiquier. Ce qui nous amène à la conclusion que la Complexité en Moyenne du Problème des N-Reines est de l'ordre  $O(N!)$  pour les versions récursives et itératives.

# 4 Preuve de programme

## 4.1 Spécifications du problème

- But : Trouver toutes les configurations valides au problème des N-queens.
- Entrée : Un entier  $N$  représentant le nombre de reines .
- Sortie : Afficher ou retourner toutes les configurations d'échiquier valides (respectant les contraintes du problème).
- Préconditions :  $N > 0$  (l'échiquier doit avoir une taille valide).

- Postconditions : Aucune reine ne menace une autre reine sur l'échiquier et toutes les configurations valides sont explorées.

## 4.2 Invariant de Boucle

Notre algorithme utilise une sorte de boucle récursive dans le but d'explorer toutes les possibilités. Dans un premier temps, déterminons les points clés de notre algorithme :

- **Cas de Base** : Si  $col \geq N$ , la fonction retourne True, indiquant que toutes les reines sont placées dans l'échiquier.
- **Boucle For** : On parcours toutes les lignes ( $i$ ) de la colonne courante ( $col$ ), vérifiant si je peux placer une reine dans la position.
- **Récursion** : La fonction s'appelle récursivement pour la colonne suivante ( $col + 1$ ) si je peux placer une autre reine dans dans position actuelle ( $board[i][col] = 1$ ).

Invariant de Boucle :  $(\forall j < col, echequier[ligne][j] = 0 \wedge \forall k < ligne, echequier[k][col] = 0)$   
De sorte à ce que la colonne j ne contiennet pas de reines et la ligne k aussi .

## 4.3 Démonstration

(voir Annexe )

## 5 Conclusion

Bien que les algorithmes récursives et itératives proposent des solutions plutot valides au problème des N-Reines, mais ces solutions sont loin d'etre optimale. Étant donné que peu importe la solution proposée, le mécanisme de recherche reste par extension le même , qui consiste à exploiter toutes les cases dans tous les cas. Aujourd'hui, le défi principal des ingénieurs de recherche est d'optimiser le code et pousser les limites de l'informatique et sortir de l'ordinaire. En ce moment, aucune solution optimale n'a été propsée pour la résolution du problème des N-Reines. Cependant, je reste optimiste à la vision que ce n'est qu'une question de temps avant que l'humain exploite les ressources informatiques de façon plus optimale.

## 6 Références Bibliographiques

### Sites Web et Tutoriels

- Google Optimization - N Queens Problem:  
[https://developers.google.com/optimization/cp/queens?hl=fr#c++\\_6](https://developers.google.com/optimization/cp/queens?hl=fr#c++_6)

## **Livres**

- ”N Queens Perimeter Method: A New Approach To The N Queens Puzzle”  
Auteur: Candida Bowtell  
Date de publication: June 19, 2020

## **Thèse Universitaire**

- ”The n-queens problem”  
Auteur: Candida Bowtell  
Université: St Anne’s College, University of Oxford  
Type de document: Thesis submitted for the degree of Doctor of Philosophy  
Date: March 2022

## **7 Annexe : Démonstration**

fonction solve-nqueens (échiquier, col, N) → échiquier[N][N]

col ← 0

tant que (col < N) faire

    ligne ← 0

    tant que (ligne < N) faire

        si échiquier[ligne][col] = 0 alors

            échiquier[ligne][col] = 1

            si solve-nqueens (échiquier, col+1, N) alors

                continuer

            sinon

                échiquier[ligne][col] = 0

            finsi;

            sinon

                ligne ← ligne + 1;

            finsi;

        fintantque;

        col ← col + 1;

    fintantque;

    renvoyer échiquier;

fin.

$$r = (\forall j < \text{col}, \text{échiquier}[\text{ligne}][j] = 0) \wedge (\forall k < \text{ligne}, \\ \text{échiquier}[k][\text{col}] = 0)$$

Passons à la démo :

1ère itération, nous avons :

$$\left( \forall j < \text{col} + 1, \text{échiquier}[\text{ligne}][j] \right) \wedge \left( \forall k < \text{ligne}, \text{échiquier}[k][\text{col}] = 0 \right)$$

$\text{col} \leq \text{col} + 1$

$$\left( \forall j < \text{col}, \text{échiquier}[\text{ligne}][j] = 0 \right) \wedge \left( \forall k < \text{ligne}, \text{échiquier}[k][\text{col}] = 0 \right)$$

( ① )

d'après les axiomes de la logique d'Hoare :

Si on a  $\{P \wedge B\} \rightarrow \{P'\}$  alors  $\{P'\} \text{tantque } B \text{ faire } S$   
Fin Tantque  $\{\neg B \wedge P\}$ .

en gros, on doit démontrer cet invariant de boucle :

$$\left\{ \left( \forall j < \text{col}, \text{échiquier}[\text{ligne}][j] = 0 \right) \wedge \left( \forall k < \text{ligne}, \text{échiquier}[k][\text{col}] = 0 \right) \right\} \wedge \left( \text{col} < N \right)$$

tantque ( $\text{col} < N$ ) faire

$\begin{array}{c} \text{---} \\ | \\ S \end{array}$

Fin tantque :

$$\left\{ \left( \forall j < \text{col}, \text{échiquier}[\text{ligne}][j] = 0 \right) \wedge \left( \forall k < \text{ligne}, \text{échiquier}[k][\text{col}] = 0 \right) \right\} \wedge \left( \text{col} \leq N \right)$$

Démontrons maintenant  $S'$  :

après avoir fait ①  $\Rightarrow$  on entre dans une autre sous-boucle  
tantque avec l'invariant de boucle

$\left( \forall k < \text{ligne}, \text{échiquier}[k][\text{col}] = 0 \right)$ . qu'on doit aussi le  
démontrer.

( ② )

en gros, démontrer aussi que :

$$\{ (\forall k < \text{ligne}, \text{echequier}[k][\text{col}] = 0) \wedge (\text{ligne} < N) \}$$

tantque ligne < N faire

S'

Fin tantque

$$\{ (\text{ligne} \geq N) \wedge (\forall k < \text{ligne}, \text{echequier}[k][\text{col}] = 0) \}.$$

en rentrant dans notre sous-boucle, nous avons :

$$\{ \forall k < \text{ligne} + 1, \text{echequier}[k][\text{col}] = 0 \} \quad \text{ligne} \leftarrow \text{ligne} + 1; \quad \{ (\forall k < \text{ligne}), \text{echequier}[k][\text{col}] = 0 \}$$

Puis nous avons une structure conditionnelle à démontrer :

o En gros :

$$\{ (\forall k < \text{ligne}, \text{echequier}[k][\text{col}] = 0) \wedge (\text{echequier}[\text{ligne}][\text{col}] = 0) \}$$

Si  $\text{echequier}[\text{ligne}][\text{col}] = 0$  alors

S'

Fin Si;

(3)

$$\{ (\forall k < \text{ligne}, \text{echequier}[k][\text{col}] = 0) \} S$$

et bien sûr un autre sinon..

$$(\forall k < \text{ligne}, \text{echequier}[k][\text{col}] = 0) \wedge (\text{echequier}[\text{ligne}][\text{col}] = 0)$$

Si  $\text{echequier}[\text{ligne}][\text{col}] \neq 0$  alors

$\text{echequier}[\text{ligne}][\text{col}] = 0$

Fin Si;

(2)

$$(\forall k < \text{ligne}, \text{echequier}[k][\text{col}] = 0).$$

(3)

commençons par ②

I)

nous avons :

$$(TB1P) \Rightarrow (\forall k < \text{ligne}, \text{échiquier}[k][\text{col}] = 0 \wedge \text{échiquier}[\text{ligne}][\text{col}] = 1)$$

$$(TB1P) \Rightarrow (\forall k < \text{ligne}, \text{échiquier}[k][\text{col}] = 0)$$

(TB1P)  $\Rightarrow \varphi$  ----- vérifiée.

II)

$$(\forall k < \text{ligne}, \text{échiquier}[k][\text{col}] = 0 \wedge \text{échiquier}[\text{ligne}][\text{col}] = 0)$$



$$(\forall k < \text{ligne}, \text{échiquier}[k][\text{col}] = 0).$$

Du coup d'après l'axiome d'Hoare :

$$\{(\forall k < \text{ligne}, \text{échiquier}[k][\text{col}] = 0)\}$$

Si  $\text{échiquier}[\text{ligne}][\text{col}] = 0$  alors

$$\text{échiquier}[\text{ligne}][\text{col}] = 1$$

Fin si

$$\{(\forall k < \text{ligne}, \text{échiquier}[k][\text{col}] = 0)\}$$

D'où : ② est vrai.

$\{ (\forall k < \text{ligne}, \text{échiquier}[k][\text{col}] = 0) \}$

Si  $\text{solve-nqueens}(\text{échiquier}, \text{col}+1, N)$  alors

    └ Continue

Finsi;

$\{ (\forall k < \text{ligne}, \text{échiquier}[k][\text{col}] = 0) \}$

donc  $\textcircled{3}$  est valide.

Passeons à  $\textcircled{4}$ :

$(7B1P) : (\forall k < \text{ligne}, \text{échiquier}[k][\text{col}] = 0) \wedge (\text{solve-nqueens}(\text{échiquier}, \text{col}+1, N))$

$\Rightarrow (\forall k < \text{ligne}, \text{échiquier}[k][\text{col}] = 0)$

$\Rightarrow P$

et  $P \Leftrightarrow Q$

donc: d'après l'axiome d'Hoare:

$\{ (\forall k < \text{ligne}, \text{échiquier}[k][\text{col}] = 0) \}$

Si  $\neg \text{solve-nqueens}(\text{échiquier}, \text{col}+1, N)$  alors

    └  $\text{échiquier}[\text{ligne}][\text{col}] = 0$

Finsi;

$\{ (\forall k < \text{ligne}), \text{échiquier}[k][\text{col}] = 0 \}$

donc  $\textcircled{4}$  est vrai

Pensons à ① Désormais :

dans le bloc S, on a encore une autre instruction à démontrer :

$$(\forall k < \text{ligne}, \text{échiquier}[k][\text{col}] = 0) \wedge (\text{solve\_nqueens}(\text{echquier})^{\text{col}+1, N})$$

③

Si  $\text{solve\_nqueens}(\text{echquier}, \text{col}+1, N)$  alors  
    | Continue  
Finsi;

$$(\forall k < \text{ligne}, \text{échiquier}[k][\text{col}] = 0)$$

et l'autre :

④  $(\forall k < \text{ligne}, \text{échiquier}[k][\text{col}] = 0) \wedge (\neg \text{solve\_nqueens}(\text{echquier}, \text{col}+1, N))$

Si  $\neg \text{solve\_nqueens}(\text{echquier}, \text{col}+1, N)$  alors  
    |  $\text{échquier}[\text{ligne}][\text{col}] = 0$   
Finsi;

$$(\forall k < \text{ligne}, \text{échiquier}[k][\text{col}] = 0)$$

commençons par ③ :

$$(GB1P) : (\forall k < \text{ligne}, \text{échiquier}[k][\text{col}] = 0) \wedge (\neg \text{solve}(\text{échquier}, \text{col}+1, N))$$

$$\Rightarrow (\forall k < \text{ligne}, \text{échiquier}[k][\text{col}] = 0)$$

$$\Rightarrow Q \quad \text{et } (GB1P) \Rightarrow P$$

et donc d'après l'axiome d'Hoare :

⑥

et donc, par extension :

$$\{(\forall k < \text{ligne}, \text{échiquier}[k][\text{col}] = 0)\} \text{ échiquier}[\text{ligne}][\text{col}] = 1 \}$$

$$\begin{aligned} &\forall k < \text{ligne}, \text{échiquier} \\ &[k][\text{col}] = 0 \end{aligned}$$

En utilisant l'axiome de transitivité d'Hoare

$$\{\text{si } m \in \{P\} \wedge \{Q\} \text{ et } \{Q\} \wedge \{R\} \Rightarrow \{P\} \wedge \{R\}\}$$

nous aurons :

$$\{(\forall k < \text{ligne}), \text{échiquier}[k][\text{col}] = 0\}$$

Si  $\text{échiquier}[\text{ligne}][\text{col}] = 0$  alors

$$\text{échiquier}[\text{ligne}][\text{col}] = 1$$

et : ~~Solve-queens~~  $\text{échiquier}, \text{col}+1, N$  alors

Continuer

Sinon

//Solve-queens(

$$\text{échiquier}[\text{ligne}][\text{col}] = 0$$

Fin si j

Sinon

$$\text{ligne} \leftarrow \text{ligne} + 1;$$

Fin si j

$$\{(\forall k < \text{ligne}, \text{échiquier}[k][\text{col}] = 0)\}$$

d'où : notre structure conditionnelle est vraie.

revenons à notre invariant de Boucle  
de la deuxième boucle interne :

• nous avons ..

$$\{P \wedge B\} = \{(\forall k < \text{ligne}, \text{échiquier}[k][\text{col}] = 0) \wedge (\text{ligne} \geq N)\}$$
$$= \{(\forall k \leq \text{ligne}, \text{échiquier}[k][\text{col}] = 0)\}$$
$$\Rightarrow \{Q\}.$$

et d'après l'axiome de transitivité d'Hoare et  
l'axiome de la Boucle :

$$\{(\forall k < \text{ligne}, \text{échiquier}[k][\text{col}] = 0)\}$$

tant que ( $\text{ligne} < N$ ) faire

    [ Si  $\text{échiquier}[\text{ligne}][\text{col}] = 1$  alors

$\text{échiquier}[\text{ligne}][\text{col}] = 0$

    [ Si  $\text{solve\_nqueens}(\text{échiquier}, \text{col} + 1, N)$  alors

        continuer

    [ Sinon

$\text{échiquier}[\text{ligne}][\text{col}] = 0$

    [ Fin si;

    [ Sinon

$\text{ligne} \leftarrow \text{ligne} + 1;$

    [ Fin tantque;

$$\{(\forall k \leq \text{ligne}, \text{échiquier}[k][\text{col}] = 0)\}.$$

du coup  $\Rightarrow$  notre invariant de boucle est valide

Passons à notre invariant de boucle principal :

Par extension, nous avons :

$$\left\{ \begin{array}{l} \forall k < \text{col}, \text{échiquier}[0][k] = 0 \\ \exists 1 \leq k < \text{col}, \\ \text{échiquier}[0][k] = 1 \end{array} \right\} \text{ligne} = 0 \quad \left\{ \begin{array}{l} \forall j < \text{col}, \text{échiquier}[\text{ligne}][j] = 0 \\ \exists 1 \leq j < \text{col}, \\ \text{échiquier}[\text{ligne}][j] = 1 \end{array} \right\}$$

et donc selon l'axiome de transitivité et de boucle d'Hoare :

$$\left\{ \forall k < \text{ligne}, \text{échiquier}[k][\text{col}] = 0 \quad \exists 1 \leq j < \text{col}, \text{échiquier}[\text{ligne}][j] = 1 \right\}$$

tantque ( $\text{col} < N$ ) faire

    ligne  $\leftarrow 0$

    tantque ligne  $< N$  faire

        Si échiquier[ligne][col] = 0 alors

            échiquier[ligne][col] = 1

        Si solve-nqueens(échiquier, col+1, N) alors  
            Continue

        Sinon  
            (Solve-nqueens)

            échiquier[ligne][col] = 0

    Finsi;

    Sinon

        ligne  $\leftarrow$  ligne + 1;

    Finsi;

Fin tantque;

    col  $\leftarrow$  col + 1;

Fin tantque

$$\left\{ \forall k < \text{ligne}, \text{échiquier}[k][\text{col}] = 0 \quad \forall j \leq \text{col}, \text{échiquier}[\text{ligne}][j] = 0 \right\}$$

(9)

Du coup, notre invariant de boucle est correcte ✓.

Par affectation et extension ..

$$\left\{ \begin{array}{l} \forall k \leq 0, \text{échiquier} \\ [0][0] = 0 \end{array} \right. \quad \left\{ \begin{array}{l} \forall j \leq 0, \text{échiquier} \\ [0][0] = 0 \end{array} \right. \quad \left\{ \begin{array}{l} \forall k \leq 0, \text{échiquier}[0][0] = 0 \\ \forall j < \text{col}, \text{échiquier}[0][j] = 0 \end{array} \right. \quad \left\{ \begin{array}{l} \forall k \leq 0, \text{échiquier}[0][\text{col}] = 0 \\ \forall j < \text{col}, \text{échiquier}[0][j] = 0 \end{array} \right.$$

$\downarrow \text{col} \leftarrow 0$

Ce qui est correct, vu qu'au départ, la case de l'échiquier est vide, tout comme tous les cases

du coup: notre programme récursif est en effet correct.

NB: Il est à noter que si notre algorithme itératif fonctionne exactement de la même façon que l'algorithme récursive  $\Rightarrow$  la preuve de l'algorithme récursif suffit amplement pour démontrer que même l'algorithme itératif est valide.