# Backtracking - sudoku

MPSI/MP OPTION INFORMATIQUE - TP n° 3.2 - Olivier Reynet

**À la fin de ce chapitre, je sais :**

- ☞ effectuer un filtrage de motif (pattern matching)
- ☞ coder une expression conditionnelle `if ... then (...)else (...)`
- ☞ coder une boucle pour en OCaml `for i=0 to n-1 do (...)done`
- ☞ garantir qu'une fonction retourne toujours le même type et éventuellement `unit ()`.
- ☞ coder une fonction sur une liste de manière récursive
- ☞ utiliser l'API List pour coder une fonction sur une liste (map, mapi, filter, fold)

## A Sudoku ?

Le sudoku est une grille carrée de 9x9 qu'il faut compléter à l'aide de nombres entiers compris entre 1 et 9. La grille est divisée en neuf lignes, neuf colonnes et neufs blocs comme indiqué sur la figure 1. Les blocs sont les carrés de 3x3 délimités par des traits en gras.

Les règles de complétion sont les suivantes : **on ne peut pas placer deux fois le même chiffre**

- sur une même ligne,
- sur une même colonne,
- sur un même bloc.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 8 | · | 9 | · | · | · | · | · | 7 |
| · | · | · | 5 | · | 9 | · | · | · |
| · | 3 | 5 | · | 7 | · | 8 | 9 | · |
| · | 2 | · | 7 | · | 8 | · | 1 | · |
| 7 | · | · | · | · | · | · | · | 2 |
| · | 6 | · | 1 | · | 5 | 4 | 7 | · |
| · | 9 | 7 | 8 | 5 | 4 | 2 | 3 | · |
| · | · | · | 9 | · | 6 | 7 | · | · |
| 5 | · | · | · | · | · | · | · | 6 |

FIGURE 1 – Exemple de sudoku. Un bloc est délimité par des traits en gras.

```
8 · 9 · · · · · 7
· · · 5 · 9 · · ·
· 3 5 · 7 · 8 9 ·
· 2 · 7 · 8 · 1 ·
7 · · · · · · · 2
· 6 · 1 · 5 4 7 ·
· 9 7 8 5 4 2 3 ·
· · · 9 · 6 7 · ·
5 · · · · · · 6
```

FIGURE 2 – Résultat sur la console de la fonction `rec_show board` avec sudoku `telegram`

## B Modélisation du sudoku

On choisit de représentere simplement le sudoku par une liste de valeur avec la convention qu'on lit la grille ligne par ligne. Ainsi, le sudoku de la figure 1 est modélisé par la liste :

```
let telegram = [ 8; 0; 9; 0; 0; 0; 0; 0; 7;
                 0; 0; 0; 5; 0; 9; 0; 0; 0;
                 0; 3; 5; 0; 7; 0; 8; 9; 0;
                 0; 2; 0; 7; 0; 8; 0; 1; 0;
                 7; 0; 0; 0; 0; 0; 0; 0; 2;
                 0; 6; 0; 1; 0; 5; 4; 7; 0;
                 0; 9; 7; 8; 5; 4; 2; 3; 0;
                 0; 0; 0; 9; 0; 6; 7; 0; 0;
                 5; 0; 0; 0; 0; 0; 0; 0; 6
];;
```

Le zéro représente l'absence de chiffre dans une case.

Si on utilise cette modélisation, le 36$^{e}$ élément de `telegram` est le dernier élément de la cinquième ligne et vaut 2. Son indice dans la liste vaut 35…

Dans tout ce qui suit, on considère que $n$ est une variable globale qui vaut 9.

B1. Écrire une fonction récursive de signature `rec_show : int list -> unit` qui affiche le sudoku sur la console comme indiqué sur la figure 2.

B2. Écrire une fonction de signature `show : int list -> unit` qui affiche le sudoku sur la console comme indiqué sur la figure 2. On utilisera les fonctions du module List.

B3. Écrire une fonction de signature `in_row int -> int -> int list -> bool` qui teste la présence d'un chiffre sur une ligne. Elle renvoie `true` si le chiffre est déjà présent sur la ligne. **Le chiffre est repéré par son indice dans la liste!**

B4. Écrire une fonction de signature `in_col int -> int -> int list -> bool` qui teste la présence d'un chiffre sur une colonne. Elle renvoie `true` si le chiffre est déjà présent sur la colonne.

B5. Écrire une fonction de signature `in_block int -> int -> int list -> bool` qui teste la présence d'un chiffre sur un bloc. Elle renvoie `true` si le chiffre est déjà présent sur le bloc.

B6. Écrire une fonction de signature `is_valid_number : int -> int -> int list -> bool` teste la validité d'un chiffre $c$ sur la grille à l'index $i$. Elle renvoie `true` si le chiffre $c$ peut-être écrit dans la case $i$.

## C Résolution du sudoku par retour sur trace

L'algorithme de retour sur trace 1 construit au fur et à mesure les solutions partielles du problème et les rejette dès qu'il découvre une impossibilité.

**Algorithme 1** Algorithme de retour sur trace

```
1 : Fonction RETOUR_SUR_TRACE(v)                    ▷ v est un nøeud de l'arbre de recherche
2 :     si v est une feuille alors
3 :         renvoyer Vrai
4 :     sinon
5 :         pour chaque fils u de v répéter
6 :             si u peut compléter une solution partielle au problème 𝒫 alors
7 :                 RETOUR_SUR_TRACE(u)
8 :     renvoyer Faux
```

On utilise la modélisation du sudoku précédente, c'est à dire qu'on construit la liste des cases occupées au fur et à mesure. On procède par ligne puis par colonne en positionnant d'abord un chiffre dans la première case vide puis une autre sur la deuxième case libre.... Ainsi de suite la liste augmente de taille jusqu'à atteindre la taille 81.

La différence par rapport au problème des n reines et que l'on commence avec une solution partielle.

C1. Comment coder «$v$ est une feuille» en OCaml?

C2. Comment coder «$u$ peut compléter une solution partielle au problème $\mathcal{P}$»?

C3. Implémenter un algorithme de retour sur trace pour le problème du sudoku pour résoudre le sudoku `telegram`.

On dispose également d'un autre sudoku `multiple` :

```
let multiple = [8; 0; 9; 0; 0; 0; 0; 0; 7;
                0; 7; 6; 0; 8; 9; 0; 0; 0;
                0; 3; 0; 0; 7; 0; 8; 9; 0;
                0; 2; 0; 7; 0; 8; 0; 1; 0;
                7; 0; 0; 0; 0; 0; 0; 0; 2;
                0; 6; 0; 1; 2; 0; 4; 7; 0;
                6; 9; 7; 8; 0; 4; 2; 3; 1;
                0; 0; 0; 9; 0; 6; 7; 0; 0;
                0; 0; 0; 0; 0; 7; 9; 0; 6
    ];;
```

C4. Tester le programme de résolution sur `multiple`. Combien y-a-t-il de solutions?

C5. Proposer une version du programme dont le modèle de sudoku est un `Array`. Quelles différences y-a-t-il avec l'implémentation sous forme de `List`?

C6. Implémenter cet algorithme en Python.