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.

Solution :

Code 1 – Sudoku - List

```
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
];;

let all_done = [1; 2; 3; 4; 5; 6; 7; 8; 9;
4; 5; 6; 7; 8; 9; 1; 2; 3;
7; 8; 9; 1; 2; 3; 4; 5; 6;
2; 3; 4; 5; 6; 7; 8; 9; 1;
5; 6; 7; 8; 9; 1; 2; 3; 2;
8; 9; 1; 2; 3; 4; 5; 6; 7;
3; 4; 5; 6; 7; 8; 9; 1; 2;
6; 7; 8; 9; 1; 2; 3; 2; 5;
9; 1; 2; 3; 4; 5; 6; 7; 8
];;

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
     ];;


let n = 9;;
let rec_show sudoku =
  let rec aux i b = match b with
   |  [] -> print_string "\n"
   |  v :: t -> if i mod n = 0 then print_newline (); if v = 0 then print_string
        "." else print_int v; print_string " "; aux (i + 1) t
  in aux 0 sudoku;;

let show sudoku =
  let pl i v = if i mod n = 0 then print_newline (); if v = 0 then print_string
      "." else print_int v; print_string " ";
in List.iteri pl sudoku; print_newline ();;

let in_row i number board =
  let row_number = i / n in
  let row = List.filteri (fun index element -> index / n = row_number) board in
  List.mem number row;;

```

```
let in_col i number board =
  let col_number = i mod n in
  let col =  List.filteri (fun index element -> col_number = index mod n ) board
      in
  List.mem number col;;
let in_block i number board =
  let row_number = i / n in
  let col_number = i mod n in
  let row_block =  (row_number / 3) * 3 in
  let col_block =   (col_number / 3) * 3 in
  let check index element = index / n   >= row_block  &&
                            index / n   <  row_block + 3 &&
                            index mod n >= col_block &&
                            index mod n <  col_block +3  in
  let block =  List.filteri check board in
  List.mem number block;;
let is_valid_number number i board = not (in_row i number board) && not (in_col
    i number board) && not (in_block i number board);;

let sudoku board =
  let sol_nb = ref 0 in
  let rec aux index b = match index with
  |  81  -> (incr sol_nb; print_string "Solution #"; print_int !sol_nb; show b)
  |  _   -> if List.nth b index > 0 then aux (index + 1) b
            else (for number = 1 to 9 do
                    if is_valid_number number index b then (aux (index+1) (List.
                        mapi (fun i e -> if index = i then number else e) b))
                  done)
  in aux 0 board;;

sudoku telegram;;
sudoku multiple;;
```

Code 2 – Sudoku - Array

```
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
|];;

let all_done = [|1; 2; 3; 4; 5; 6; 7; 8; 9;
4; 5; 6; 7; 8; 9; 1; 2; 3;
7; 8; 9; 1; 2; 3; 4; 5; 6;
2; 3; 4; 5; 6; 7; 8; 9; 1;
5; 6; 7; 8; 9; 1; 2; 3; 2;
8; 9; 1; 2; 3; 4; 5; 6; 7;
3; 4; 5; 6; 7; 8; 9; 1; 2;
6; 7; 8; 9; 1; 2; 3; 2; 5;
9; 1; 2; 3; 4; 5; 6; 7; 8
```

```
21 |];;
22
23 let multiple = [|8; 0; 9; 0; 0; 0; 0; 0; 7;
24      0; 7; 6; 0; 8; 9; 0; 0; 0;
25      0; 3; 0; 0; 7; 0; 8; 9; 0;
26      0; 2; 0; 7; 0; 8; 0; 1; 0;
27      7; 0; 0; 0; 0; 0; 0; 0; 2;
28      0; 6; 0; 1; 2; 0; 4; 7; 0;
29      6; 9; 7; 8; 0; 4; 2; 3; 1;
30      0; 0; 0; 9; 0; 6; 7; 0; 0;
31      0; 0; 0; 0; 0; 7; 9; 0; 6
32      |];;
33
34
35 let n = 9;;
36 let rec_show sudoku =
37   let rec aux i b = match b with
38    |  [] -> print_string "\n"
39    |  v :: t -> if i mod n = 0 then print_newline (); if v = 0 then print_string
          "." else print_int v; print_string " "; aux (i + 1) t
40   in aux 0 sudoku;;
41
42 let show sudoku =
43   let pl i v = if i mod n = 0 then print_newline (); if v = 0 then print_string
       "." else print_int v; print_string " ";
44 in Array.iteri pl sudoku; print_newline ();;
45 show telegram;;
46
47 let in_row i number board =
48   let row_number = i / n in
49   let row = Array.sub board (row_number * n) n  in
50   Array.mem number row;;
51 in_row 8 7 telegram;;
52
53 let in_col i number board =
54   let col_number = i mod n in
55   let col =  List.filteri (fun index element -> col_number = index mod n ) (
       Array.to_list board) in
56   List.mem number col;;
57 let in_block i number board =
58   let row_number = i / n in
59   let col_number = i mod n in
60   let row_block =  (row_number / 3) * 3 in
61   let col_block =   (col_number / 3) * 3 in
62   let check index element = index / n   >= row_block  &&
63                             index / n   <  row_block + 3 &&
64                             index mod n >= col_block &&
65                             index mod n <  col_block +3  in
66   let block =  List.filteri check (Array.to_list board) in
67   List.mem number block;;
68 let is_valid_number i number board = not (in_row i number board) && not (in_col
     i number board) && not (in_block i number board);;
69
70 let sudoku board =
71   let sol_nb = ref 0 in
```

```
  let rec aux index b = match index with
  |  81  -> (incr sol_nb; print_string "Solution #"; print_int !sol_nb; show b)
  |  _   -> if b.(index) > 0 then aux (index + 1) b
            else (for number = 1 to 9 do
                    if is_valid_number  index  number b
                      then (b.(index) <- number; aux (index + 1) b; b.(index) <-
                           0;)
                  done)
  in aux 0 board;;

sudoku telegram;;
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
     |];;

sudoku multiple;;
```

Code 3 – Sudoku - Python

```
# coding: utf8
# nb solution --> https://oeis.org/A000170/list
from random import randrange, sample, seed

nu = 3
N = nu * nu
solutions_number = 0
seed(66)

telegram = [8, None, 9, None, None, None, None, None, 7,
            None, None, None, 5, None, 9, None, None, None,
            None, 3, 5, None, 7, None, 8, 9, None,
            None, 2, None, 7, None, 8, None, 1, None,
            7, None, None, None, None, None, None, None, 2,
            None, 6, None, 1, None, 5, 4, 7, None,
            None, 9, 7, 8, 5, 4, 2, 3, None,
            None, None, None, 9, None, 6, 7, None, None,
            5, None, None, None, None, None, None, None, 6
            ]

all_done = [1, 2, 3, 4, 5, 6, 7, 8, 9,
            4, 5, 6, 7, 8, 9, 1, 2, 3,
            7, 8, 9, 1, 2, 3, 4, 5, 6,
            2, 3, 4, 5, 6, 7, 8, 9, 1,
            5, 6, 7, 8, 9, 1, 2, 3, 2,
            8, 9, 1, 2, 3, 4, 5, 6, 7,
            3, 4, 5, 6, 7, 8, 9, 1, 2,
            6, 7, 8, 9, 1, 2, 3, 2, 5,
            9, 1, 2, 3, 4, 5, 6, 7, 8
```

```
            ]

multiple = [8, None, 9, None, None, None, None, None, 7,
            None, 7, 6, None, 8, 9, None, None, None,
            None, 3, None, None, 7, None, 8, 9, None,
            None, 2, None, 7, None, 8, None, 1, None,
            7, None, None, None, None, None, None, None, 2,
            None, 6, None, 1, 2, None, 4, 7, None,
            6, 9, 7, 8, None, 4, 2, 3, 1,
            None, None, None, 9, None, 6, 7, None, None,
            None, None, None, None, None, 7, 9, None, 6
            ]



def show(board):
    for c in range(N * N):
        if board[c]:
            print(board[c], end=" ")
        else:
            print('.', end=" ")
        # print("L :", c // N, "C: ", c % N)
        if (c % N) % nu == nu - 1:
            print(' ', end=" ")
        if c % N == N - 1:
            print()
            if (c // N) % nu == nu - 1:
                print()
    print("--")


def in_row(board, c, n):
    row = c // N
    return n in board[row * N:row * N + N]


def in_col(board, c, n):
    col = c % N
    return n in board[col:col + N * N:N]


def in_block(board, c, n):
    row = c // N
    col = c % N
    r = (row // nu) * nu
    c = (col // nu) * nu
    block_given = set()
    for i in range(r, r + nu):
        for j in range(c, c + nu):
            if board[i * N + j]:
                block_given.add(board[i * N + j])
    return n in block_given


def is_valid(board, c, n):
```

```
85      return not in_row(board, c, n) and not in_col(board, c, n) and not in_block(
            board, c, n)
86
87
88  def sudoku(board, c=0):
89      global solutions_number
90      if c == N * N or None not in board:  # last place set or full board
91          solutions_number += 1
92          print("New solution !", solutions_number)
93          show(board)
94          return True
95      else:
96          if board[c] is None:
97              for n in range(1, N + 1):
98                  if is_valid(board, c, n):
99                      board[c] = n
100                     sudoku(board, c + 1)
101                     board[c] = None  # Backtrack
102         else:
103             sudoku(board, c + 1)
104     return False
105
106
107 if __name__ == "__main__":
108     show(telegram)
109     #sudoku(telegram)
110     sudoku(multiple)
```