Pendant ce projet nous allons voir comment résoudre des problèmes à priori difficiles par essai/erreur en utilisant des exceptions.

1 Modélisation du problème

J'imagine que tout le monde connaît les règles du sudoku, voici tout de même un petit rappel du jeu. Le sudoku est un jeu sur une grille de 9x9 cases, découpée en 9 sous carrés de 3x3 cases. Certaines cases contiennent déjà un entier de 1 à 9 le but est de remplir toutes les cases avec de tels entiers en vérifiant les contraintes que chaque entier doit apparaître une unique fois sur chaque ligne, sur chaque colonne et dans chaque sous-carré.

Nous allons ici fabriquer un programme qui résout toutes les grilles de sudoku, pour cela on va modéliser chaque case par une liste des valeurs possibles de cette case, on a donc le type case = int list. Une grille est alors définie comme une liste de liste de cases : type grille = case list list.

Question 1 [4] Écrire une fonction sure: case -> int option qui renvoie Some a si la case ne contient qu'un entier a on est alors sûr de la valeur de cette case et None dans les autres cas.

Question 2 Cherché une grille de sudoku facile sur internet (ou ailleurs). Écrire une valeur grille_test : grille qui encode cette grille de sudoku.

Question 3 [8] On va tout d'abord écrire trois fonctions qui vont nous permettre de manipuler plus facilement les lignes colonnes et sous carrés. Les trois fonctions ont le même type :

int*int -> int -> int * int les deux premiers arguments sont un couple (i, j) désignant une case de la grille le troisième argument est un entier entre 1 et 9.

La fonction iter_colonne (i,j) est une bijection de [1;9] vers les cases de la colonne contenant la case (i,j).

La fonction iter_ligne (i,j) est une bijection de [1;9] vers les cases de la ligne contenant la case (i,j).

La fonction iter_carre (i,j) est une bijection de [1;9] vers les cases du sous carré contenant la case (i,j).

Question 4 [14] Écrire une fonction qui imprime une grille à l'écran. Vous pourrez utiliser les fonctions print_string: string -> unit; print_newline : unit -> unit; print_int: int -> unit;

2 Résolution de contrainte simple

Question 5 Écrire la fonction : map_grille : (int -> int -> case -> case) -> grille -> grille tel que map_grille f g applique la fonction f à toutes les cases de la grille g. Les 2 premiers paramètres de f sont les coordonnées en colonne et ligne de la case numérotées de 0 à 8. Cette fonction nécessite des fonctions auxiliaires récursives.

Au début de la résolution, chaque case d'une grille contient soit une seule valeur, soit toutes les valeurs (c.-à-d. la liste [1;2;3;4;5;6;7;8;9]). Dans cette partie nous allons supprimer au fur et à

mesure des éléments dans les listes en applicant directement les règles du sudoku. Les chiffres entre crochets donnent une idée du nombre de lignes nécessaire pour écrire la fonction.

Pour simplifier les tests, je vous fournis le code suivant, la valeur grille_test est valide pour commencer la résolution.

```
let build_grille = fun g ->
  map_grille (fun _ -> fun _ -> fun c ->
     match c with
            [] \rightarrow [1;2;3;4;5;6;7;8;9]
           [x] \rightarrow [x]
            _ -> failwith "Grille incorrect"
    );;
let grille_test_incomplete [
  [[1]; []; [];
                     [8]; [3]; [];
                                        [];[];[2]];
  [[5]; [7]; [];
                     [];[];
                               [1];
                                        [];[];;
                                        [];
  [[]; []; [];
                     [5]; [];
                               [9];
                                             [6]; [4]];
  [[7]; []; [4];
                     []; []; [8];
                                        [5]; [9]; []];
  [[]; []; [3];
                     []; [1]; [];
                                        [4]; [ ]; [ ]];
  [[]; [5]; [1];
                     [4]; []; [];
                                        [3]; [ ]; [6]];
                                          ]; []; []];
                     [7]; [ ];
[6]; [ ];
  [[3]; [6]; [ ];
[[ ]; [ ]; [ ];
                                 [4];
                                 [];
                                        []; [7]; [9]];
  [[8]; []; [];
                     []; [5];
                                [2];
                                        [];[];[3]]]
];;
let grille_test = build_grille grille_test_incomplete;;
```

Dans la suite le but est d'enlever le plus de valeur possible dans chaque case jusqu'à ce que la valeur devienne sure. La plupart des fonctions ont un argument acc de type case qui contient la liste des valeurs possible pour une case. Pour chacune des cases, il faut : regarder pour toutes les cases qui sont dans la même colonne, ligne et carré si la valeur est sure l'enlever des possibilités. Les questions suivantes découpent ce problème, chaque question est assez courte. pour simplifier la lecture des types, je définis les abréviations de types suivantes :

```
type coord = int*int
type iter = int -> int -> coord
```

Les fonctions iter_column, iter_ligne, iter_carre sont de type iter.

Question 6 [2] Écrire la fonction get : grille -> int -> int -> case tel que get g i j renvoit le contenu de la case à la ligne i et à la colonne j de la grille g. On pourra utiliser la fonction List.nth

Question 7 [4] Écrire la fonction retire_valeur_case : grille -> coord -> case -> case telle que retire_case grille acc (1,m) regarde si la valeur de la case au coordonné (1,m) est sure. Si elle l'est, la fonction renvoie la liste des possibilités acc sans cette valeur. Sinon acc est renvoyé inchangé. On pourra utiliser la fonction List.filter

Question 8 [3] Écrire la fonction retire_valeur_case_it : grille -> coord -> case -> iter -> int -> case tel que retire_valeur_case_it g (i,j) acc it k calcule les coordonnées (1,m) avec l'itérateur it appliqué aux coordonnées (i,j) et l'entier k. Si $(i,j) \neq (k,l)$ la fonction applique retire_valeur_case sinon elle renvoie acc.

Question 9 [4] Écrire la fonction retire_valeur_case_it_app : grille -> coord -> case -> it -> case qui applique la fonction précédente pour toutes les valeurs dans [0;1;2;3;4;5;6;7;8]. Vous pouvez utiliser la fonction List.fold_left ou utiliser une fonction auxiliaire récursive.

Question 10 [5] Écrire la fonction iter_all: grille -> coord -> case -> case qui applique la fonction précédente sur les trois itérateurs.

Question 11 [4] Écrire la fonction enleve_sur : grille -> grille qui applique la fonction précédente sur chaque case de la grille.

Faites des tests sur la grille grille_test à chaque application de enleve_sur le nombre de possibilités diminue, au bout de 5 applications la grille doit être résolue. Faites des tests avec d'autres grilles.

1	123456789	123456789	8	3	123456789	123456789	123456789	2
5	7	123456789 123456789	123456789	123456789 123456789	1	123456789 123456789	123456789	12345678 4
7	123456789		123456789			5	9	12345678
123456789 123456789	123456789 5		123456789 4 +		123456789 123456789		123456789 123456789	
3 123456789 8		123456789 123456789 123456789	16		123456789	123456789 123456789 123456789	7	12345678 9 3
nleve_sur	e grille_te	 est :	+			+		
1	49		18	3	67	79	5	2
5	7		12	246		189	38	8
2	238	28	5 +	27	9	178 +	6	4
7	2		123	26		15	9	1
269 29	289 5		29 4	1 279		4 3	28 28	78 6
			÷			+		
3	6		17	89		1128	1258	158
24	124 149		6 19	8 5		128 16	7 14	9
		sure grille	+		<u>.</u>	+		
			+			+		
1 5	49 7	69 69	8 2	3 46		79 9	5 3	2
2	38		15	7		117	6	4
7	2	4	+ 3	6	8	+ 5	9	1
69	89		13	1		15 14	28	7
9	5		14	29	7	3	28	6
3	6	259	+ 7	9		+ 128	128	5
4	14	259	16	8		1128	7	9
8	149	79	119	5		116	14	3
nleve_sur	e (enleve_	sure (enlev	+ e_sure gri	lle_test))	:	+		
1	49	9	+ 8	3	6	+ 7	5	2
5	7	6	12	4	1	19	3	8
2	3	8	5 +	7	9	1	6	4
7	2	4	3	6	8	† 5	9	1
				1		14	28	7
6	8		19			1 -		,
		3	9 4 +	2		3	28	6
6	8	3 1 		9	7		28 128	
6 9 3 4	8 5 6 1	3 1 2 25	4 +	9	7 	3 + 128 12	128 7	6 5 9
6 9 3	8 5 6	3 1 2 25	4 + 7	9	7 	3 + 128	128	6 5
6 9 3 4 8	8 5 6 1 19	3 1 2 25	4 +	9 8 5	7 4 3 2	3 + 128 12 16 +	128 7	6 5 9
6 9 3 4 8	8 5 	3 1 2 25 79 	4 +	9 8 5	7 4 3 2 grille_test	3 + 128 12 16 +	128 7 14 	6 5 9 3
6 9	8 5 	3 1 2 2 25 79 	4 +	9 8 5 Leve_sure (7 4 3 2 zrille_test 6 1	3 +	128 7 14 5 3	6 5 9 3
6 9 3 4 4 8 1	8 5 	3 1 2 2 25 79 	4 +	9 8 5 Leve_sure {	7 4 3 2 zrille_test 6 1	3 +	128 7 14 	6 5 9 3
6 9 9 3 4 8 8 1 1 5 5 2 2 7	8 5 6 1 19 e (enleve_: 4 7 3	3 1 2 25 79 	4 +	9 8 5 Leve_sure { 3 4 7	7 4 3 2 grille_test 6 1 9	3 +	128 7 14 5 3 6	6 5 9 3 2 8 4
6 9 9 3 4 8 8 8 1 1 5 5 5 2 7 7 6 6	8 5 5 6 1 19 e (enleve_: 4 7 3 3 2 8	3 1 2 25 79 sure (enlev 9 6 8	4 +	9 8 5 Leve_sure 2 4 7	7 	3 +	128 7 14 5 3 6	6 5 9 3
6 9 9 3 4 8 8 1 1 5 5 2 2 7	8 5 6 1 19 e (enleve_: 4 7 3	3 1 2 25 79 sure (enlev 9 6 8	4 +	9 8 5 Leve_sure { 3 4 7	7 	3 +	128 7 14 5 3 6	6 5 9 3 2 8 4
6 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9	8 5 5 6 1 1 19 4 7 3 2 2 8 5 5 6 6	3 1 2 2 25 79 9 6 8 4 3 1 1 2 2	4 +	9 8 5 5 Leve_sure { 3 4 7 7 6 6 1 2 9 9	7 4 3 2 rrille_test 6 1 9 8 5 7	3 +	128 7 14 	6 5 9 3 2 8 4 1 7 6
6 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9	8 5 6 1 19 4 7 3 2 8 5 6 1	3 1 2 25 79 sure (enlev 9 6 8 4 3 1	4 +	9 8 5 5 eve_sure 1 3 4 7 6 1 2 2 9 8 8	7 	3 +	128 7 14 5 3 6 9 2 8 8 18 7	6 5 9 3 2 8 4 1 7 6 5 9
6 9 9 3 4 8 8 7 7 6 6 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9	8 5 5 6 1 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9	3 1 2 25 79 sure (enlev 9 6 8 3 1 1 2 2 5 7 7 - 2 5 7 7	4 +	9 8 5 5 Leve_sure 1 3 4 7 7 6 6 1 2 9 8 5 5	7 4 3 2 grille_test 6 1 9 8 5 7 4 3 2	3 +	128 7 14 5 3 6 9 2 8 118 7	6 5 9 3 2 8 4 1 7 6
6 9 9 3 4 8 5 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9	8 5 6 1 9 e (enleve_i	3 1 2 25 79 sure (enlev 9 6 8 3 1 1 2 2 5 7 7 sure (enlev conservation)	4 +	9 8 5 5	7 4 3 2 grille_test 6 1 9 8 5 7 4 4 3 2 2 cenleve_sur.	3 +	128 7 14 5 3 6 9 2 8 118 7 4	6 5 9 3 2 8 4 1 7 6 5 9 3
6 9 9 3 4 8 8 5 5 5 5 5 5 6 6 9 9 9 9 9 9 9 9 9 9 9 9	8 5 6 1 19 e (enleve : 4 7 3 3 2 8 5 6 6 1 9 e (enleve : 4 4	3 1 2 2 25 79 sure (enlev 9 6 8 8 4 3 1 1 2 2 5 7 7 sure (enlev 9 9 6 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9	4	9 8 5 5 eve_sure 1 3 4 7 7 6 1 1 2 2 9 8 8 5 5 Eve_sure 3 3	7 4 3 2 2 Srille_test 6 1 9 8 5 7 7 4 4 3 2 2 (enleve_sure 6 6	3 +	128 7 14 	6 5 9 3
6 9	8 5 6 1 1 9 2 8 5 6 1 9 9 e (enleve_: 4 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7	3 1 2 25 79 sure (enlev 9 6 8 8 1 2 5 7 7 sure (enlev 9 6 6	4 +	9 8 5 5	7 4 3 2 2	3 +	128 77 114 53 6 9 9 2 8 118 7 4 4 sst)))) :	6 5 9 3 2 2 8 4 1 7 6 5 9 3
6 9	8 5 6 1 9 2 8 5 6 1 9 9 e (enleve_i 4 7 3 3 3 3 9 9	3 1 2 25 79 sure (enlev 9 6 8 3 1 1 2 5 7 7 sure (enlev 9 6 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8	4 +	9 8 5 5 4 7 7 6 6 1 2 9 8 5 5 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6	7 4 3 2 2	3 +	128 7 114 5 3 6 9 2 8 8 118 7 4 4 	6 5 9 3 2 8 4 1 7 6 5 9 3
6 9	8 5 6 1 1 9 2 8 5 6 1 9 9 e (enleve_: 4 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7	3 1 2 25 79 sure (enlev 9 6 8 3 1 2 5 7 7 sure (enlev 9 6 8 8 4 4 3 4 5 5 7 7 5 5 5 5 6 6 8 8 4 4 4 4 4 4	4 +	9 8 5 5	7 4 3 2 rrille_test 6 1 9 8 5 7 4 3 2 cenleve_sure 6 1 9	3 +	128 77 114 53 6 9 9 2 8 118 7 4 4 sst)))) :	6 5 9 3 2 2 8 4 1 7 6 5 9 3

Projet OCaml – Partie I et II				Résolution du Sudoku					Benoît Barbot – 2020		
13	6	2	17	9	4	18	18	5	I		
14	1	5	16	8	3	12	7	9	I		
Ĭ8	9	7	<u> </u> 1	5	2	<u> 1</u> 6	4	3	1		