Sudoku

Introduction

Il est strictement INTERDIT d'utiliser l'opérateur `@` ainsi que le module `List` (et les autres).

Le Sudoku

> *"Le Sudoku (prononcé soudokou en français, est un jeu en forme de grille défini en 1979 par l'Américain Howard Garns.*
> *Le but du jeu est de remplir la grille avec une série de chiffres (ou de lettres ou de symboles) tous différents, qui ne se trouvent jamais plus d'une fois sur une même ligne, dans une même colonne ou dans une même sous-grille. La plupart du temps, les symboles sont des chiffres allant de 1 à 9, les sous-grilles étant alors des carrés de 3 × 3. Quelques symboles sont déjà disposés dans la grille, ce qui autorise une résolution progressive du problème complet."*
> Source : Wikipédia.

Pendant ce TP nous allons vous demander d'écrire un algorithme qui nous permettra de résoudre des grilles de Sudoku simple (pas d'ambiguïté sur les valeurs pendant la résolution).
Afin de rendre les choses plus faciles (ou pas) le type de la grille vous est imposé !
On considérera une grille permettant de stocker les valeurs de `1` au nombre de valeurs, on utilisera la valeur `null` pour dire que la case est vide.

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

Références :

```
# let null = 0
  and values = [1;2;3;4;5;6;7;8;9]
  and grid_sample =
    [[5; 3; 0; 0; 7; 0; 0; 0; 0];
     [6; 0; 0; 1; 9; 5; 0; 0; 0];
     [0; 9; 8; 0; 0; 0; 0; 6; 0];
     [8; 0; 0; 0; 6; 0; 0; 0; 3];
     [4; 0; 0; 8; 0; 3; 0; 0; 1];
     [7; 0; 0; 0; 2; 0; 0; 0; 6];
     [0; 6; 0; 0; 0; 0; 2; 8; 0];
     [0; 0; 0; 4; 1; 9; 0; 0; 5];
     [0; 0; 0; 0; 8; 0; 0; 7; 9]];;
```

Level 0 : Les Must-do

0.1 Longueur

Écrire la fonction `length` qui retourne la longueur d'une liste.

```
val length : 'a list -> int = <fun>
```

0.2 Aplatir

Écrire la fonction `flatten` qui à partir d'une liste de liste, retourne une liste.

```
val flatten : 'a list list -> 'a list = <fun>
# flatten [[1;2;3];[4;5]];;
- : int list = [1; 2; 3; 4; 5]
```

0.3 Présence

Écrire la fonction `check` qui nous indique si un élément présent dans la liste vérifie la fonction de comparaison passée en paramètre avec comme premier argument la valeur passée en second argument.

```
val check : ('a -> 'b -> bool) -> 'b -> 'a list -> bool = <fun>
```

0.4 Suppression

Écrire la fonction `remove` qui retire la première occurrence d'un élément dans la liste en utilisant la fonction de comparaison passée en paramètre avec comme premier argument la valeur passée en second argument.

```
val remove : ('a -> 'b -> bool) -> 'b -> 'a list -> 'a list = <fun>
```

0.5 Unicité

Écrire la fonction `list_uniq` qui va retirer les doublons (trouvés via la fonction de comparaison) de tous les élements et va retourner la liste filtrée. L'ordre des éléments n'a pas d'importance.

```
val list_uniq : ('a -> 'a -> bool) -> 'a list -> 'a list = <fun>
# list_uniq (=) [1;2;3;5;6;8;0;0;0;2;0;3;0;5;6;0;1;4];;
- : int list = [8; 2; 3; 5; 6; 0; 1; 4]
```

0.6 Unicité - bis

Écrire la fonction `list_match` qui va retourner la liste des éléments (une seule occurence de chaque élément, trouvés via la fonction de comparaison) présent dans les deux listes passées en paramètres. L'ordre des éléments n'a pas d'importance.

```
val list_match : ('a -> 'a -> bool) -> 'a list -> 'a list -> 'a list = <fun>
# list_match (=) [1;2;3;5;6;8;0;0;0] [2;0;3;0;5;6;0;1;4];;
- : int list = [8; 2; 3; 5; 6; 0; 1; 4]
```

1 Level 1 : Les Matrices

1.1 Rectangle

Écrire la fonction `grid_make_rectangle` qui retourne une liste contenant y listes contenant x valeurs nulles (spécifiée en paramètre).

```
val grid_make_rectangle : int -> int -> 'a -> 'a list list = <fun>
```

1.2 Carré

Écrire la fonction `grid_make_square` qui retourne une liste contenant x listes contenant x valeurs nulles (spécifiée en paramètre).

```
val grid_make_square : int -> 'a -> 'a list list = <fun>
```

1.3 Grille

Écrire la fonction `grid_make` qui retourne une liste de liste pouvant contenir la liste des valeurs de notre jeu de Sudoku (passée en paramètre), remplie de la valeur nulle (spécifiée en paramètre).

```
val grid_make : 'a list -> 'b -> 'b list list = <fun>
```

2 Level 2 : Les Extractions

2.1 Ligne

Écrire la fonction `extract_row` qui retourne la ligne numéro `n`. Les lignes sont numérotées de `0` au nombre de valeurs possible - 1 (`length values - 1`), de haut en bas. La valeur de retour de la fonction doit être une liste de valeurs simples.

```
val extract_row : 'a list list -> int -> 'b list -> 'a list = <fun>

# extract_row grid_sample 5 values;;
 - : int list = [7; 0; 0; 0; 2; 0; 0; 0; 6]
```

2.2 Colonne

Écrire la fonction `extract_column` qui retourne la colonne numéro `n`. Les colonnes sont numérotées de `0` au nombre de valeurs possible - 1 (`length values - 1`, de gauche à droite. La valeur de retour de la fonction doit être une liste de valeurs simples.

```
val extract_column : 'a list list -> int -> 'b list -> 'a list = <fun>

# extract_column grid_sample 6 values;;
 - : int list = [0; 0; 0; 0; 0; 0; 2; 0; 0]
```

2.3 Carré

Écrire la fonction `extract_square` qui retourne la sous-grille numéro `n`. Les sous grilles sont numérotées de `0` au nombre de valeurs possible - 1 (`length values - 1`), de gauche à droite et de haut en bas. La valeur de retour de la fonction doit être une liste de valeurs simples. La fonction prendra en paramètre le plateau de jeu, le numéro de la sous-grille, ainsi que la liste des valeurs possible pour notre jeu.

```
val extract_square : 'a list list -> int -> 'b list -> 'a list = <fun>

# extract_square grid_sample 8 values;;
 - : int list = [2; 8; 0; 0; 0; 5; 0; 7; 9]
```

2.4 Affichage

Écrire la fonction `grid_print` qui affiche une grille sur la sortie standard. La fonction prend en paramètre la fonction d'affichage d'un élément.

```
 val grid_print : ('a -> 'b) -> 'a list list -> unit = <fun>

# grid_print (print_int) grid_sample;;
5 3 0 0 7 0 0 0 0
6 0 0 1 9 5 0 0 0
0 9 8 0 0 0 0 6 0
8 0 0 0 6 0 0 0 3
4 0 0 8 0 3 0 0 1
7 0 0 0 2 0 0 0 6
0 6 0 0 0 0 2 8 0
0 0 0 4 1 9 0 0 5
0 0 0 0 8 0 0 7 9
 - : unit = ()
```

3 Level 3 : Les vérifications

3.1 Présence et Unicité

Écrire la fonction `list_validate` qui permet de vérifier que les éléments de notre liste de valeurs ne sont présents dans une autre liste qu'une seule et unique fois (via la méthode de comparaison spécifiée en paramètre. Attention au cas particulier de la valeur nulle (spécifiée en paramètre également).

```
 val list_validate : ('a -> 'b -> bool) -> 'a list -> 'a -> 'b list -> bool = <fun>

# list_validate (=) values null [1;2;3;4;5;6;7;8;9];;
 - : bool = true

# list_validate (=) values null [0;1;3;4;0;0;0;1;9];;
 - : bool = false

# list_validate (=) values null [1;3;5;0;4;6;0;0;2];;
 - : bool = true
```

3.2 Grille valide

Écrire la fonction `grid_validate` qui permet de vérifier qu'une grille respecte bien les règles du Sudoku. C'est à dire pas deux éléments de même valeur sur une ligne ou colonne ou sous-grille. Attention toujours aux valeurs nulles.

```
 val grid_validate : ('a -> 'b -> bool) -> 'b list list -> 'a list -> 'a -> bool = <fun>

# grid_validate (=) grid_sample values null;;
 - : bool = true
```

3.3 Grille pleine

Écrire la fonction `grid_isfull` qui permet de vérifier qu'une grille est pleine ou pas. C'est à dire qu'elle ne contient plus de valeurs nulles.

```
 val grid_isfull : ('a -> 'b -> bool) -> 'b list list -> 'a -> bool = <fun>

# grid_isfull (=) grid_sample null;;
 - : bool = false
```

4 Level 4 : La résolution

4.1 Élement manquant

Écrire la fonction `find_missing` qui retourne la liste des élements de notre liste de valeurs non présents dans la deuxième.

```
 val find_missing : ('a -> 'b -> bool) -> 'b list -> 'a list -> 'b list = <fun>

# find_missing (=) values [8;2;3;5;6;0;1;4];;
 - int list = [7; 9]
```

4.2 Les choix

Écrire la fonction `grid_find` qui liste les possibilités de valeurs pour une case de notre grille de Sudoku.

```
 val grid_find : ('a -> 'a -> bool) -> 'a list list -> int -> int -> 'a list -> 'a list = <fun>

# grid_find (=) grid_sample 8 0 values;;
  - : int list = [2; 4; 8]

# grid_find (=) grid_sample 0 8 values;;
  - : int list = [1; 2; 3]

# grid_find (=) grid_sample 0 6 values;;
  - : int list = [1; 3; 9]
```

4.3 La solution au rang suivant

Écrire la fonction `grid_nsolve` qui donne la grille suivante dans le processus de résolution :
On se propose de parcourir toute la grille et pour chaque case :

- si la case contient une valeur non nulle, on garde cette valeur dans la nouvelle grille.
- si la case contient une valeur nulle, on va regarder la liste des solutions possibles pour cette case
  - si la liste des solutions contient plus d'une possibilité, on ignore la case et on passe a la suivante.
  - si la liste des solutions ne contient qu'une seule valeur, on met cette valeur dans la nouvelle grille.

Une fois l'ensemble des cases vues, on retourne la nouvelle grille.

```
 val grid_nsolve : ('a -> 'a -> bool) -> 'a list list -> 'a -> 'a list -> 'a list list = <fun>

# grid_print (print_int) (grid_nsolve (=) grid_sample null values)
5 3 0 0 7 0 0 0 0
6 0 0 1 9 5 0 0 0
0 9 8 0 0 0 0 6 0
8 0 0 0 6 0 0 0 3
4 0 0 8 5 3 0 0 1
7 0 0 0 2 0 0 0 6
0 6 0 0 0 7 2 8 4
0 0 0 4 1 9 0 3 5
0 0 0 0 8 0 0 7 9
 - : unit = ()
```

5 Level 5 : Bonus - obligatoire ou presque :)

5.1 Résolution complète

Écrire la fonction `solve` qui donne la solution pour une grille de Sudoku.

```
val solve : 'a list list -> ('a -> 'a -> bool) -> 'a list -> 'a -> 'a list list = <fun>

# grid_print (print_int) (solve grid_sample (=) values null;;
5 3 4 6 7 8 9 1 2
6 7 2 1 9 5 3 4 8
1 9 8 3 4 2 5 6 7
8 5 9 7 6 1 4 2 3
4 2 6 8 5 3 7 9 1
7 1 3 9 2 4 8 5 6
9 6 1 5 3 7 2 8 4
2 8 7 4 1 9 6 3 5
3 4 5 2 8 6 1 7 9
- : unit = ()
```

6 Level 6 : Les bonus

6.1 La vraie résolution complète

Dans la partie obligatoire, on ne propose de résoudre que des grilles simples, ou il n'y a jamais d'ambiguïté sur les valeurs à choisir. Cependant ce cas n'est pas représentatif de l'ensemble des grilles de Sudoku. On se propose de gérer dans cette partie la résolution des grilles avec ambiguïté sur le choix des valeurs. Attention certains choix vous emmènerons donc sur des grilles non valides.

6.2 La génération de grille

Maintenant que nous sommes capables de résoudre toutes les grilles, nous allons donc passer à la création de celles-ci. Il nous suffit de prendre une grille complète et valide, et de remplacer certaines valeurs par la valeur nulle. La complexité de la grille dépendra des positions et quantités de valeurs que vous allez retirer.

7 Level 7 : Bonus *j'ai vraiment rien d'autre a faire*

Puisque apparemment le reste était tellement trivial (et je vous comprends) que vous vous êtes ennuyés sur le reste du tp, je vous invite à aller sur wikipedia et d'implémenter les autres types de grille de Sudoku :

- Nonomino
- Killer Sudoku
- Hyper Sudoku
- ...