

Option Informatique

TP 6: Sudoku

David Baelde

david.baelde@ens-lyon.org

Vous commencerez ce TP en récupérant quelques fonctions préféfinies pour le second exercice. Pour cela insérez (menu File > Insert file) le fichier `~/Pub/tp6defs.ml`.

1 Étirements

Pour commencer, nous allons pratiquer un peu les références, dans l'exercice inverse du préliminaire du TP4.

Mémo

```

List.mem e [x1; ...; xn] ≡ e ∈ [x1; ...; xn]
List.map f [x1; ...; xn] ≡ [f x1; ...; f xn]
List.filter f [x1; ...; xn] ≡ [xi | f xi]
List.length [x1; ...; xn] ≡ n
List.rev [x1; ...; xn] ≡ [xn; ...; x1]
List.iter f [x1; ...; xn] ≡ f x1; ...; f xn
List.fold_left f i [x1; x2; ...; xn] ≡ f (... (f i x1) x2) ... xn

```

► Réécrivez toutes les fonctions du mémo en n'utilisant que `List.iter`! Bien sûr pas le droit de faire les récursions à la main... Si vous n'êtes pas trop rouillés, faites directement `List.fold_left`, et passez à la suite.

2 Sudoku

La plupart d'entre vous connaissent déjà le jeu de Sudoku. Pour les rebelles, j'en rappelle rapidement les règles. On joue sur une grille 9×9 . On y distingue des *groupements* de trois types : les lignes, les colonnes, et les neuf carrés disjoints 3×3 . Le but du jeu est de compléter une grille, en remplissant les cases par un chiffre entre 0 et 9, de sorte que chaque groupement comporte tous les chiffres.

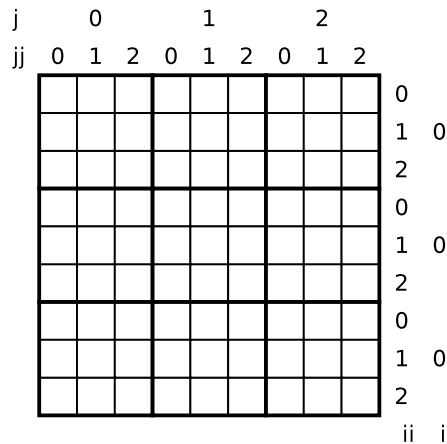
Une première technique de résolution consiste à choisir une case, et pour chaque choix de chiffre possible pour cette case essayer de résoudre la grille. Nous nous intéresserons pas à cette technique pour deux raisons. D'abord c'est assez similaire à ce que nous avons fait pour les reines. Mais surtout, il y a plus intelligent dans ce cas.

Nous allons exploiter une propriété supplémentaire des jeux de Sudoku : ils sont *faisables dans le métro*. Cette propriété mathématique compliquée veut dire qu'il est possible de résoudre une grille de Sudoku avec un stylo, sans gomme, sans faire des choix : il suffit de raisonner par condition nécessaire. Dans ce TP nous allons implémenter deux schémas de déduction qui permettent de résoudre un grand nombre de grilles.

Nous calculerons pour chaque case une majoration de l'ensemble des chiffres encore possibles. On parlera des possibilités associées à une case, et on dira qu'une possibilité est choisie sur une case si la case ne présente plus que cette possibilité. Les deux schémas de déduction permettant de raffiner nos approximations sont les suivants :

- Si une possibilité est choisie sur une case d'un groupement, l'enlever des autres cases du groupement.
- Si une possibilité n'est plus proposée que par une case d'un groupement, la choisir pour cette case.

Encore une fois, la représentation des données va être cruciale pour résoudre notre problème. Les cases seront simplement représentées par des `int list`. Nous adopterons par contre un système de coordonnées astucieux pour représenter facilement les groupements. Au lieu d'une grille 9×9 nous aurons une grille $3 \times 3 \times 3 \times 3$, soit un tableau `((int list) array array array array)`.



- Programmez les fonctions calculant l'intersection de deux listes de possibilités, puis le complémentaire d'une liste de possibilités.
- Pour chaque type de groupement (`line`, `col`, `block`) écrivez un itérateur de type `(int*int) -> (int -> int -> int -> int -> unit) -> unit`. Par exemple `(iter_col (1,1) f)` appelle `(f i ii 1 1)` pour chaque case $(i, ii, 1, 1)$ de la colonne du milieu.

- ▶ Programmez les deux étapes de déduction comme des fonctions prenant un groupement (représenté par son itérateur) et une grille, qui raffinent si possible les ensembles de possibilités sur le groupement, et renvoient un booléen indiquant si une amélioration a eu lieu.
- ▶ Assemblez le tout et résolvez le problème fourni : appliquer les deux étapes de déduction à tous les groupements jusqu'à ce qu'aucune amélioration ne soit apportée.

Pour aller plus loin, on pourra trouver d'autres méthodes de déduction. On peut par exemple appliquer des principes similaires à nos deux étapes mais prenant en compte des paires de cases d'un groupement : s'il ne reste que deux cases à posséder deux possibilités, on peut éliminer les autres possibilités pour ces cases ; inversement, si deux cases n'offrent plus que les deux mêmes possibilités alors ces possibilités peuvent être éliminées pour les autres cases du groupement.

Une fois qu'on sait résoudre une grille par condition nécessaire, on peut générer des grilles *faisables dans le métro*. Pour cela on remplit petit à petit la grille, et on peut s'arrêter dès qu'elle est complètement résoluble par notre algorithme. Pour obtenir une jolie grille on la remplira en suivant un motif régulier, etc.