▶ **Lycée Faidherbes – MPSI – Option Informatique**
Cinquième TP Caml
Vendredis 21 et 28 mars 2008

# Sudokus

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

*Lors de ce cinquième TP, nous allons essayer de résoudre des grilles de sudoku.*

## 1 Structure de données

Il y a plusieurs façon de représenter la grille de sudoku dans la mémoire de l'ordinateur. Par exemple : une liste de listes d'entiers (`int list list`), un tableau de tableaux d'entiers (`int array array`), un tableau d'entiers (`int array`)...
J'ai choisi d'utiliser des tableaux de tableaux, vous pouvez utiliser la représentation qui vous conviendra le mieux, mais vous devrez écrire des fonction de conversion si vous souhaitez utiliser mes exemples.
Nous allons également avoir besoin de représenter des cases vides. Une solution courante consiste à considérer que la valeur 0 correspond à une case vide. Cette méthode n'est cependant pas la plus élégante et Caml nous offre des mécanismes bien plus intéressant. Par exemple il nous permet de définir nos propres types, soit en combinant des types existants :

```
(* ce type correspond a une liste d'entiers *)
type liste_d_entiers = int list;;
```

soit en construisant des types à l'aide de mots clefs :

```
(* les jours de la semaine *)
type jour_semaine = Lundi | Mardi | Mercredi ...;;
(* Notez que les mots clefs commencent
   toujours par une majuscule *)
```

on peut enfin combiner les deux : utiliser à la foi les mots clefs et des types existants :

```
(* les nombre, entiers ou flotants *)
type nombre = Entier of int | Reel of int,,
```

En ce qui nous concerne, pour représenter l'état de nos cases, on peut utiliser le type suivant :

```
type case = Vide | Pleine of int;;
```

Mais Caml nous propose déjà un type presque identique, le type `option`. Ce type est défini de la façon suivante :

```
type 'a option = None | Some of 'a;;
```

On peut donc utiliser une type `int option` pour représenter nos cases. On n'utilisera donc pas des tableaux ou des listes d'entiers mais des des tableaux ou des listes d'entiers optionnels (`int option array` ou `int option list`).
Dans mon cas, une grille de sudoku est donc représentée par un objet du type `int option array array`. Dorénavant, j'utiliserai `sudoku` pour désigner le type d'une grille, quelque soit la représentation choisie.

⋙ **Question 1** *Écrire une fonction qui crée une grille vide. (type :* `init_sudoku : unit -> sudoku`*)*

Malheureusement, Caml ne nous permet pas de fixer des règles sur la taille des tableaux : il nous faut une fonction qui vérifie qu'un élément du type sudoku est bien une grille de sudoku (une matrice carrée de taille 9)

⋙ **Question 2** *Écrire une fonction qui vérifie la taille de la grille et qui lève une exception si la grille est mauvaise. (type :* check1 : sudoku −> unit*)*

⋙ **Question 3** *Écrire une fonction*

case : sudoku −> int −> int −> int option

*qui renvoie le contenu d'une case en fonction de ses numéros de ligne et de colonne.*

## 2 Règles du jeu

Je rappelle les règles du jeu : pour pouvoir insérer un chiffre dans une case, il faut que ce chiffre n'apparaisse pas déjà :

1. dans la colonne
2. dans la ligne
3. dans la sous-grille

⋙ **Question 4** *Écrire une fonction :*

verif_case : sudoku −> int −> int −> int −> unit

*qui vérifie, étant donnés une grille, un numéro de ligne, un numéro de colonne et un chiffre, s'il est licite d'insérer le chiffre dans la case correspondant à la ligne et la colonne dans la grille.*

## 3 Grille vide ?

On va essayer dans cette partie de remplir une grille vierge. Pour cela, on va utiliser une solution canonique : On rempli la première ligne avec les chiffres dans l'ordre, puis la quatrième en les décalants une fois, la septième en les décalant à nouveau, on retourne à la seconde, etc. Cela devrait donner la grille suivante :

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

⋙ **Question 5** *Pour remplir cette grille, on commence par la ligne 1 puis la 4 puis la 7 puis la 2 puis la 3.... l'ordre des lignes est en fait le suivant :*

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

*Construire une boucle qui affiche ces indices dans l'ordre.*

⋙ **Question 6** *À l'aide de cette boucle, écrivez une fonction qui remplis correctement une grille vide.*

## 4 Backtracking

À présent nous allons résoudre des grilles avec contraintes, c'est à dire partiellement remplies. Pour cela on utilisera une méthode assez naïve qui essayer toutes les solutions possible jusqu'à en trouver une bonne. on appelle cette méthode "backtracking" car on revient en arrière quand on tombe sur un cul-de-sac.
On va tenter d'insérer un 1 dans la première case vide. Si c'est possible alors on passe à la case suivante, si ça ne l'est pas, ou si on a pas trouvé de solution avec 1 on essaie 2 puis 3, etc. Si on arrive à 9 sans trouver une solution, on retourne à la case précédente pour y essayer le nombre suivant etc.

⋙ **Question 7** *Écrivez une fonction qui résout une grille par backtracking.*