Devoir Maison n°02

4 mai 2024

L'exercice 1 est à faire en remplissant un fichier sudoku.ml, disponible sur le Drive. Celui-ci est partiellement rempli. Les questions de programmation sont précédées d'un P encadré.

Les questions de programmation sont à faire en remplissant directement le fichier sudoku.ml, qui devra être renvoyé complété par mail, avec comme nom de fichier sudoku_nom_prenom.ml. L'exécution du fichier complété correctement doit créer trois fichiers textes dont le contenu correspond à des grilles spécifiques complétées.

Les autres questions sont à rédiger, soit dans un fichier texte à renvoyer par mail en même temps que le programme, soit écrit sur papier.

Une attention particulière sera portée à la rédaction et à la propreté du code (mise en forme, commentaires...)

Exercice 1

On considère une variante du Sudoku avec des grilles de taille 4×4. la grille est partiellement remplie par des valeurs entre 1 et 4, et l'objectif est de remplir la grille de Sudoku de telle sorte que chaque nombre entre 1 et 4 soit présent au plus une fois :

- Dans chacun des quatre carrés 2×2 ;
- Dans chaque ligne ;
- Dans chaque colonne.

On cherche ici à effectuer la résolution automatique une grille de Sudoku représentée sous la forme d'une tableau en deux dimensions en OCaml, en utilisant la logique propositionnelle. On travaillera sur des formules du calcul propositionnel sur des variables libres $x_{i,j,k}$ avec $0 \leq i, j \leq 3$ et $1 \leq k \leq 4$, où $x_{i,j,k}$ est la variable propositionnelle correspondant à l'affirmation : "la case à la ligne i et à la colonne j est remplie par la valeur k".

Les morceaux de code sont à remplir aux endroits adéquats dans le fichier `sudoku.ml`, qui contient également des portions de code et fonctions (avec leurs spécifications) qui pourront être utilisées.

1) Combien de variables propositionnelles a-t-on ? Combien a-t-on de valuations possibles pour cet ensemble de variables ?

2) **P** On va représenter les formules du calcul propositionnel sur cet ensemble de variables avec le type `formule` défini ainsi :

```
type formule = Vrai
             | Faux
             | Var of int*int*int
             | Neg of formule
             | Et of formule list
             | Ou of formule list ;;
```

Une variable $x_{i,j,k}$ est représentée par `Var(i,j,k)`. Par associativité de la conjonction, on peut représenter une conjonction de plusieurs formules par la liste de ces formules, mise en argument d'un constructeur `Et` (pareil pour les disjonctions avec le constructeur `Ou`).

A partir d'une grille initiale partiellement remplie, on cherche à construire une formule représentant la donnée du remplissage initial d'une grille, ainsi que les règles du Sudoku. Cette formule est satisfiable si et seulement s'il existe une solution à la grille de Sudoku. Une valuation satisfaisant la formule permettra de déduire une solution pour la grille.

La formule à satisfaire représentant une grille de Sudoku remplie correctement peut s'écrire sous la forme d'une conjonction de formules sur l'ensemble des variables propositionnelles $x_{i,j,k}$. Cette conjonction contient une formule informant sur les valeurs contenues dans les cases déjà remplies initialement, ainsi que des formules correspondant aux affirmations suivantes :

(a) Chaque case contient au moins un nombre (autrement dit : pour toute case (i, j), on a soit $x_{i,j,1}$, soit $x_{i,j,2}$, soit $x_{i,j,3}$, soit $x_{i,j,4}$)

(b) Chaque case contient au plus un nombre (autrement dit : pour toute case (i, j), pour toutes valeurs $1 \leq k_1 < k_2 \leq 4$, on n'a pas en même temps x_{i,j,k_1} et x_{i,j,k_2})

(c) Chaque ligne contient une valeur au plus une fois (autrement dit : pour tout ligne i, pour toute valeur k entre 1 et 4, pour tous $0 \leq j_1 < j_2 \leq 3$, on n'a pas $x_{i,j_1,k}$ et $x_{i,j_2,k}$)

(d) Chaque colonne contient une valeur au plus une fois (autrement dit : pour tout colonne j, pour toute valeur k entre 1 et 4, pour tous $0 \leq i_1 < i_2 \leq 3$, on n'a pas $x_{i_1,j,k}$ et $x_{i_2,j,k}$)

(e) Chaque carré 2×2 contient une valeur au plus une fois (autrement dit : pour tous $0 \leq i_1, j_1, i_2, j_2 \leq 3$ tels que (i_1, j_1) et (i_2, j_2) sont dans le même carré 2×2 et $(i_1, j_1) \neq (i_2, j_2)$, pour toute valeur k, on n'a pas $x_{i_1,j_1,k}$ et $x_{i_2,j_2,k}$)

Créer des variables :

— `grille_complete`
— `un_par_case`
— `un_par_ligne`
— `un_par_colonne`
— `un_par_carre`

de type `formule`, représentant des formules correspondant respectivement aux affirmations (a), (b), (c), (d) et (e).

3) On représente en OCaml une grille de Sudoku partiellement remplie par un tableau en deux dimensions de taille 4×4. On considère que la case (i, j) de la grille est remplie par la valeur k si `t.(i).(j)` prend une valeur k entière entre 1 et 4, et qu'elle est vide sinon.

Donner la représentation de la grille suivante sous forme d'un tableau en deux dimensions en OCaml :

2	3	1	
		2	
3			
	4		

4) Donner une formule (sous la forme d'une conjonction de variables propositionnelles) représentant l'information du remplissage initial de la grille dans la question précédente.

5) **P** Ecrire une fonction `formule_grille t` qui prend en entrée un tableau `t` en deux dimensions représentant une grille de Sudoku partiellement remplie, et renvoie la formule correspondant à l'information de remplissage initial de cette grille.

6) Soit `t` un tableau à deux dimensions représentant une grille de Sudoku partiellement remplie. Exprimer en fonction des objets définis précédemment une formule qui est satisfiable si et seulement si la grille de Sudoku admet une solution, et pour laquelle on peut, à partir d'une valuation des variables qui satisfait la formule, déduire une solution pour la grille.

7) **P** Dans la suite du programme, les fonctions suivantes ont été définies :

- `substitution_sudoku`
- `simplif_quine_sudoku`
- `variables`

analogues aux fonctions `substitution`, `simplif_quine` et `variables` de l'exercice 2 du TP 13, adaptées pour le type `formule` tel qu'il a été défini ici.

Ecrire les fonctions suivantes, qui sont utilisées par la fonction `simplif_quine_sudoku` et dont les spécifications sont les suivantes :

(a) `simplif_list_et l`, qui prend en entrée une liste de formules `l` (de type `formule list`) et supprime les `Vrai` de la liste ;

(b) `simplif_list_ou l`, qui prend en entrée une liste de formules `l` et supprime les `Faux` de la liste ;

(c) `vrai_in l`, qui prend en entrée une liste de formules `l` et renvoie `true` si la liste contient `Vrai` et `false` sinon ;

(d) `faux_in l`, qui prend en entrée une liste de formules `l` et renvoie `true` si la liste contient `Faux` et `false`.

8) **P** Le programme contient également un type `arbre_quine_sudoku` et une fonction `arbre`, analogues au type `arbre_quine` et à la fonction `arbre` de l'exercice 2 du TP 13. La fonction `arbre` prend en entrée une formule et renvoie son arbre de Quine (de type `arbre_quine_sudoku`). Le constructeur `Noeud` prend un argument de type `int * int * int * arbre_quine_sudoku * arbre_quine_sudoku` : les deux sous-arbres et l'étiquette constituée des trois entiers i, j, k de la variable $x_{i,j,k}$ sur laquelle est faite la substitution.

Ecrire une fonction `valuation_sat_arbre a` qui prend en entrée un arbre de Quine associé à une formule, et renvoie un couple composé d'une liste de formules et d'un booléen, où :

- Si la formule est satisfiable, la liste de formules est une liste de variables satisfaites par une valuation qui satisfait la formule, et le booléen vaut `true` ;

— Si la formule n'est pas satisfiable, la liste de formules est vide et la booléen vaut `false`.

9) **P** Ecrire une fonction `liste_to_val` qui prend en entrée une liste de formules égales à des variables propositionnelles, et renvoie un tableau en deux dimensions de taille 4×4 avec à la case (i, j) :

— La valeur k si une variable de la forme `Var(i,j,k)` est présente dans la liste

— La valeur 0 s'il n'y a pas de variable de la forme `Var(i,j,k)` dans la liste.

10) **P** A l'aide des fonctions précédentes, écrire une fonction `valuation_sat` qui prend en entrée une formule du calcul propositionnel, et renvoie le tableau en deux dimensions de taille 4×4 en sortie de `liste_to_val` correspondant à une liste de variables à satisfaire pour satisfaire la formule en entrée. La fonction renvoie une exception avec un message d'erreur si la formule n'est pas satisfiable.

11) **P** A l'aide des questions précédentes, écrire une fonction `solution_grille` qui prend en entrée une grille de Sudoku sous la forme d'un tableau en deux dimensions de taille 4×4.

12) **P** Ecrire une fonction `fichier_solution` qui prend en entrée un nom de fichier et un tableau correspondant à une grille de Sudoku partiellement remplie, et stocke dans un fichier texte une solution à la grille de Sudoku. Par exemple, pour la grille complétée suivante :

1	2	4	3
3	4	1	2
4	3	2	1
2	1	3	4

le texte à l'intérieur du fichier correspondant sera le suivant :

```
1 2 4 3

3 4 1 2

4 3 2 1

2 1 3 4
```

Lorsque le code du programme est complété, les dernières lignes du programme permettent, lors de son exécution, de stocker dans des fichiers `grille_1.txt`, `grille_2.txt` et `grille_3.txt` les solutions des grilles suivantes :

3	0	2	0
2	4	1	0
0	0	0	0
1	0	0	0

1	0	3	4
0	4	0	0
0	0	2	0
0	0	0	0

2	0	0	0
0	1	2	0
3	0	0	1
0	0	0	0