Centrale Informatique MP 2014 — Corrigé

Ce corrigé est proposé par Vincent Puyhaubert (Professeur en CPGE) ; il a été relu par Benjamin Monmege (ENS Cachan) et Guillaume Batog (Professeur en CPGE).

L'épreuve d'informatique du concours Centrale s'intéresse cette année au jeu populaire du sudoku. Il permet d'écrire deux fonctions permettant de trouver la solution en pratique de manière quasi-systématique. La construction de ces fonctions passe par le codage d'une occurrence du jeu par une fonction booléene imposante (environ 12 000 clauses, et la bagatelle de 729 variables propositionnelles). On utilise alors essentiellement le cours de logique pour simplifier cette formule et en déduire la solution du sudoku.

- La première partie demande d'écrire quelques fonctions élémentaires de manipulation de listes. Elle ne doit poser aucune difficulté.
- La deuxième partie a pour objectif de traduire les règles du jeu, puis le remplissage initial de la grille, par une formule booléenne sous forme normale conjonctive. La partie théorique consiste simplement à recenser un certain nombre de contraintes. La programmation est franchement plus rébarbative puisqu'il faut parfois imbriquer jusqu'à quatre boucles.
- La dernière partie concerne la résolution du jeu. Elle utilise l'argument simple consistant à dire qu'une conjonction de la forme $x \wedge \mathrm{F}$ avec x un littéral n'est satisfaite que si x est satisfait, et qu'alors F est équivalente à une formule F' où x et son complémentaire ont disparu. On en déduit une première méthode de résolution par simplifications en chaîne, suivie d'une seconde qui utilise un test supplémentaire pour poursuivre lorsque la première méthode ne permet pas de conclure.

L'épreuve dans son ensemble est bien rédigée, l'auteur faisant preuve d'un gros effort de pédagogie. Si vous avez toujours rêvé de faire résoudre vos sudokus par la machine, ce sujet est à travailler sans hésiter. Cependant, on peut regretter que les chapitres abordés dans cette épreuve soient réduits au cours de logique, c'est-à-dire une fraction très restreinte du programme de prépa.

Indications

Partie I

I.A Rédiger une fonction récursive. Faire de même pour les deux questions suivantes.

I.D Commencer par regarder ce qui se passe dans une matrice 3×3 dont les cases sont numérotées dans l'ordre de l'énoncé.

Partie II

II.A.1.a Utiliser une disjonction de 9 littéraux.

II.A.1.b Quantifier la phrase mathématique obtenue à la question précédente.

II.A.1.c Écrire K1 comme une conjonction de clauses de la forme de la question II.A.1.a.

II.A.1.e Utiliser une référence sur une liste complétée au fur et à mesure à l'aide de boucles. On pourra écrire la fonction en deux temps à l'aide d'une première fonction qui construit les clauses de la formule.

II.A.2 Même technique qu'à la question II.A.1.

II.A.3.a Pour la formule B1, utiliser le fait que les variables associées au bloc d'indice b sont indexées par `indice`(b, r) pour $r \in [\![0 ; 8]\!]$.

II.A.3.b Utiliser les mêmes idées qu'à la question II.A.1.e.

II.A.4.a Traduire la propriété par le fait que deux cases différentes de la ligne i ne peuvent contenir la valeur k simultanément.

II.A.4.b Considérer la conjonction de toutes les formules de la question précédente.

II.A.4.d Utiliser quatre boucles for imbriquées.

II.A.5 Adapter la formule obtenue pour L2 à la question II.A.4.b.

II.B.1 Commencer à nouveau par construire une fonction qui crée la conjonction des littéraux définissant la condition initiale : « la case (i, j) contient la valeur k et aucune autre valeur ».

II.B.2.a Définir la fonction réciproque de celle de la question I.D.

II.B.2.b Utiliser une référence sur une liste initialement vide, et ajouter à cette liste une clause $\neg x^p_{(i,j)}$ pour chaque valeur p de `T` non nulle contenue dans la même ligne, la même colonne, et le même bloc que la case d'indice (i, j).

II.B.2.c Concaténer les listes obtenues à l'aide de `interdites_ij` pour les couples (i, j) tels que `T.(i).(j)` est nul.

II.B.3 Remarquer que $\mathrm{F}_{\text{grille}}$ ne contient jamais un littéral et son complémenté.

Partie III

III.A.1 Comparer le nombre de valuations satisfaisant F_{initiale} et le nombre de solutions au sudoku.

III.A.2 Déterminer le nombre de valuations de n variables propositionnelles.

III.A.3 Remarquer qu'une valuation satisfaisant F doit nécessairement attribuer une valeur à p qui satisfait le littéral ℓ.

III.A.5.a Déterminer une valeur parmi $\{1, 2, 3, 4, 7, 9\}$ ne pouvant se trouver nulle part ailleurs qu'en case $(0, 7)$.

III.A.6 Écrire un programme qui renvoie le littéral contenu dans la première clause trouvée à un seul élément dans F.

III.A.7 Renvoyer à partir de la liste des clauses de F une liste obtenue récursivement en supprimant (resp. modifiant) celles qui contiennent ℓ (resp. $\neg\,\ell$).

III.A.8 Appliquer récursivement les simplifications jusqu'à obtenir une formule sans littéral isolé.

III.A.10 Vérifier que les fonctions `nouveau_lit_isole` et `simplification` parcourent un nombre borné de fois chaque clause.

III.B.1 Remarquer qu'une formule contenant une clause vide n'est pas satisfaisable, et que par ailleurs, $F \equiv (F \wedge x) \vee (F \wedge (\neg\, x))$ pour toute formule F et tout littéral x.

III.B.2 Utiliser la fonction `flatten` pour récupérer les littéraux d'une formule (et non pas les variables comme le demande l'énoncé).

III.B.3 Ajouter x à la formule F et simplifier cette nouvelle formule (penser à utiliser une copie de T). Vérifier ensuite si la liste vide appartient au résultat.

III.B.4 Définir une fonction qui cherche parmi les littéraux apparaissant dans F si l'un d'entre eux réussit le test de la règle du littéral infructueux.

I. Présentation du jeu de sudoku

I.A La structure de liste se prête naturellement à une fonction récursive, qui peut s'écrire de la manière suivante :

```
let rec appartient x l =
  match l with
  [] -> false
  |t::q -> (t=x) || appartient x q ;;
```

> La fonction `appartient` est déjà prédéfinie en Caml sous le nom `mem`. Pour autant, l'énoncé demande clairement de la redéfinir. On notera que la signature de cette fonction est `'a -> 'a list -> bool`. La fonction est dite polymorphe, ce qui signifie qu'elle peut s'appliquer à n'importe quelle liste, indépendamment du type des éléments qu'elle contient.

I.B Il suffit à nouveau d'écrire une fonction récursive.

```
let rec supprime x l =
  match l with
  [] -> []
  |t::q -> if t=x then supprime x q
                  else t::supprime x q;;
```

I.C La fonction `appartient` permet de tester si l'élément `x` que l'on veut ajouter à la liste `l` s'y trouve déjà. Il n'y a plus qu'à renvoyer la bonne liste en fonction du résultat du test.

```
let ajoute x l =
  if appartient x l then l
  else x::l;;
```

I.D Commençons par regarder ce qu'il se passe au sein d'une matrice de taille 3×3 dont les éléments sont numérotées selon l'ordre de l'énoncé :

	0	1	2
0	0	1	2
1	3	4	5
2	6	7	8

Si r est le numéro d'une case, on constate alors que ses indices de ligne et de colonne sont respectivement donnés par le quotient et le reste de la division euclidienne de r par 3. On en déduit les formules suivantes : la case de bloc (b, r) dans une grille est celle d'indice (i, j) avec

$$i = 3 \cdot b_1 + r_1 \qquad \text{et} \qquad j = 3 \cdot b_2 + r_2$$

avec (b_1, b_2) (respectivement (r_1, r_2)) les quotients et restes de la division euclidienne de b (respectivement r) par 3. Cela permet d'en déduire la fonction suivante :

```
let indice (b,r) =
  (3*(b quo 3)+(r quo 3), 3*(b mod 3)+(r mod 3));;
```