CONCOURS CENTRALE•SUPÉLEC

Informatique

MP

4 heures Calculatrices autorisées

2014

Les candidats indiqueront en tête de leur copie le langage de programmation choisi (Pascal ou Caml). Les candidats ayant choisi Caml devront donner le type de chaque fonction écrite. Les candidats travaillant en Pascal pourront écrire des fonctions ou des procédures.

Le sujet comporte trois parties : la partie I est essentiellement consacrée à présenter le problème étudié ; la partie III nécessite d'avoir compris le principe de l'encodage décrit et étudié dans la partie II, mais il n'est pas besoin d'avoir répondu aux questions de la partie II pour l'aborder.

I Présentation du jeu de sudoku

Règles du jeu

Le *sudoku* est un jeu de réflexion très répandu depuis quelques années. Il se présente sous l'aspect d'une *grille* carrée à 81 *cases* (9 *lignes* et 9 *colonnes*), elle-même scindée en 9 sous-grilles de taille 3×3 appelées *blocs* (cf figure 1). Cette grille est initialement partiellement remplie par des chiffres compris entre 1 et 9 : c'est la *grille initiale*. Le but du jeu consiste à compléter entièrement la grille en remplissant les cases initialement vides par des chiffres de 1 à 9, de telle manière qu'une fois la grille remplie, chacun des chiffres compris entre 1 et 9 apparaisse une et une seule fois dans chacune des 9 lignes, chacune des 9 colonnes et chacun des 9 blocs de la grille : on obtient alors une *grille finale* associée à la grille initiale.

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

Exemple de grille initiale

	0	1	2	3	4	5	6	7	8
0	0	9	0	2	0	0	6	0	5
1	3	2	0	0	0	7	0	0	0
2	0	7	0	9	0	5	0	0	8
3	0	1	0	0	0	0	0	0	0
4	0	0	7	0	0	0	0	9	4
5	6	0	0	0	0	0	0	0	0
6	0	0	8	0	0	0	0	0	7
7	0	3	0	4	9	1	5	0	0
8	0	0	0	0	0	3	0	0	0

Représentation de la grille initiale ci-contre (où le bloc numéro 3 est grisé)

Figure 1

On suppose dans tout le sujet que la grille initiale est ainsi faite qu'elle admet *une et une seule* grille finale qui lui soit associée. *Résoudre* une grille de sudoku consiste donc à déterminer la grille finale associée à une grille initiale donnée.

La figure 1 à gauche représente une grille initiale (dont 24 cases sont remplies). Les blocs sont matérialisés par un trait plus épais.

L'objet du problème est d'étudier un algorithme de résolution des grilles de sudoku fondé sur la manipulation de formules logiques. En effet, le principe de la résolution d'une grille de sudoku peut s'énoncer facilement par les cinq règles logiques ci-dessous.

Construire à partir d'une grille initiale I donnée une grille finale F telle que :
(K) toute case de F contient une et une seule fois l'un des chiffres 1 à 9 ;
(L) toute ligne de F contient une et une seule fois chacun des chiffres 1 à 9 ;
(C) toute colonne de F contient une et une seule fois chacun des chiffres 1 à 9 ;
(B) tout bloc de F contient une et une seule fois chacun des chiffres 1 à 9 ;
(I) toute case de I remplie conserve la même valeur dans F.

À ces cinq règles, il convient donc d'ajouter implicitement que la grille F existe et est unique. On peut remarquer que les quatre premières conditions (K), (L), (C) et (B) présentent de la redondance d'un point de vue logique, mais cette redondance s'avère utile pour déduire plus facilement de nouveaux faits permettant de remplir la grille.

Représentation d'une grille de sudoku

Une grille est représentée par un tableau à deux dimensions à 9 lignes et 9 colonnes numérotées chacune de 0 à 8. Elle est découpée en 9 blocs numérotés également de 0 à 8 dans l'ordre de lecture de gauche à droite et de haut en bas et les 9 cases d'un bloc donné sont également numérotées de 0 à 8 selon le même procédé. Ainsi, la même case d'une grille peut être référencée de deux manières différentes : pour $(i,j) \in [\![0,8]\!]^2$, on appelle *case d'indice* (i,j) la case de la grille située à l'intersection de la ligne i et de la colonne j ; pour $(b,r) \in [\![0,8]\!]^2$, on appelle *case de bloc* (b,r) la case de la grille située dans le bloc numéro b ayant le numéro r. Une case non encore remplie est affectée de la valeur 0.

La figure 1 à droite représente le tableau correspondant à la grille initiale donnée à gauche, dans laquelle le bloc numéro 3 est grisé. Les éléments du bloc grisé de numéros 0, 1, 2, 3, 4, 5, 6, 7 et 8 sont donc respectivement affectés des valeurs 0, 1, 0, 0, 0, 7, 6, 0 et 0. La case d'indice $(5,0)$ est aussi la case de bloc $(3,6)$: elle est grisée de manière plus foncée.

Caml

Une grille de sudoku est représentée par un tableau défini par `make_matrix 9 9 0` ; pour $(i,j) \in [\![0,8]\!]^2$, l'accès à la case d'indice (i,j) d'un tel tableau `T` se fait par l'expression `T.(i).(j)`.

Pascal

Une grille de sudoku est représentée par un tableau rectangulaire `array [0..8,0..8] of integer` ; pour $(i,j) \in [\![0,8]\!]^2$, l'accès à la case d'indice (i,j) d'un tel tableau `T` se fait par l'expression `T[i,j]`.

Préliminaires

On commence par écrire quelques fonctions qui seront utiles dans les parties suivantes.

I.A – Écrire une fonction `appartient` d'appartenance d'un élément à une liste.

I.B – Écrire une fonction `supprime` de suppression de toutes les occurrences d'un élément dans une liste.

I.C – Écrire une fonction `ajoute` d'ajout d'un élément dans une liste sans redondance (si l'élément appartient déjà à la liste, on renvoie la liste sans la modifier).

I.D – Écrire une fonction/procédure `indice` qui, étant donné un couple $(b,r) \in [\![0,8]\!]^2$ correspondant à la case de bloc (b,r) dans une grille, renvoie l'indice (i,j) de cette case dans la grille. Cette fonction pourra utiliser judicieusement les quotients et restes de divisions euclidiennes par 3 et on justifiera les formules utilisées. Par exemple, la case de bloc $(3,6)$ est la case d'indice $(5,0)$ (c'est la case grisée en plus foncé dans la figure 1) : `indice` appliqué au couple $(3,6)$ doit donc renvoyer le couple $(5,0)$.

Caml

La fonction aura pour signature

```
indice : int * int -> int * int = < fun >
```

On rappelle que pour deux entiers `a` et `b`, les expressions `a quo b` et `a mod b` calculent respectivement leur quotient et leur reste dans la division euclidienne.

Pascal

La procédure aura pour en-tête

```
procedure indice(b, r : integer ; var i, j : integer) ;
```

et modifiera la valeur des entiers `i` et `j`.
On rappelle que pour deux entiers `a` et `b`, les expressions `a DIV b` et `a MOD b` calculent respectivement leur quotient et leur reste dans la division euclidienne.

II Codage de la formule initiale

Représentation des formules logiques

Dans tout le problème, $\wedge$, $\vee$ et $\neg$ dénotent respectivement les connecteurs de *conjonction*, de *disjonction* et de *négation*. Étant donné un ensemble $\mathcal{V}$ de *variables propositionnelles*, on considère l'ensemble $\mathcal{F}$ des *formules logiques* construites à partir de $\mathcal{V}$ et de l'ensemble de connecteurs $\{\wedge, \vee, \neg\}$.

Une *valuation* sur $\mathcal{V}$ est une application $\sigma : \mathcal{V} \to \mathbb{B}$ (où $\mathbb{B} = \{\text{vrai}, \text{faux}\}$ désigne l'ensemble des booléens) ; cette application σ s'étend en une application $\text{Ev}_\sigma : \mathcal{F} \to \mathbb{B}$, attribuant une *valeur de vérité* à toute formule $F \in \mathcal{F}$ sous la valuation σ. Une formule logique F est *satisfiable* s'il existe une valuation σ telle que $\text{Ev}_\sigma(F) = \text{vrai}$. Deux formules logiques F et G sont *équivalentes* si $\text{Ev}_\sigma(F) = \text{Ev}_\sigma(G)$ pour toute valuation σ, et on note alors $F \equiv G$.

Un *littéral* est une formule logique réduite à p ou à $\neg p$, où p est une variable propositionnelle. Une *clause* est une formule logique écrite sous la forme d'une disjonction de littéraux. La *clause vide* est la clause ne contenant aucun littéral, elle est notée $\bot$ et est considérée comme non satisfiable. Une *clause unitaire* est une clause réduite à un seul littéral ; une clause unitaire *positive* (respectivement *négative*) est une formule logique réduite à p (respectivement à $\neg p$), où p est une variable propositionnelle. Une formule en *forme normale conjonctive* est une formule logique écrite sous la forme d'une conjonction de clauses. La *formule vide* est la formule ne contenant aucune clause, elle est notée $\square$ et est considérée comme satisfiable.

Dans tout le problème, les variables propositionnelles sont indexées par un triplet d'entiers (i, j, k) et notées $x^k_{(i,j)}$, avec la sémantique suivante : pour un triplet $(i, j, k) \in [\![0, 8]\!] \times [\![0, 8]\!] \times [\![1, 9]\!]$, une valuation σ assigne la valeur vrai à $x^k_{(i,j)}$ lorsque la case d'indice (i, j) de la grille de sudoku contient la valeur k. L'ensemble $\mathcal{V}$ utilisé dans le problème est donc constitué des $9^3 = 729$ variables propositionnelles $x^k_{(i,j)}$, où (i, j, k) parcourt $[\![0, 8]\!]^2 \times [\![1, 9]\!]$.

À titre d'exemples :

- la clause $x^5_{(0,0)} \vee x^5_{(0,8)} \vee x^5_{(8,0)} \vee x^5_{(8,8)}$ exprime qu'une au moins des quatre cases situées aux coins de la grille est occupée par un 5 ;
- la formule sous forme normale conjonctive $\bigwedge_{i=0}^{8} \left(\bigvee_{k=1}^{9} x^k_{(i,7)} \right)$ exprime que chacune des valeurs de la colonne 7 est constituée de chiffres compris entre 1 et 9 ;
- la clause $\bigvee_{j=0}^{8} x^6_{\texttt{indice}(3,j)}$ exprime que l'une des cases du bloc numéro 3 de la grille contient le chiffre 6 (où `indice` est la fonction/procédure de la question I.D).

Pour la programmation, les clauses sont des listes de littéraux et les formules logiques (qui seront écrites exclusivement en forme normale conjonctive) des listes de clauses.

Ainsi, en notant les listes entre $\langle\ \rangle$, la formule $x^7_{(1,8)} \wedge (x^3_{(0,4)} \vee \neg\, x^5_{(2,7)})$ est représentée formellement par la liste de listes $\langle\ \langle\ (1, 8, 7)\ \rangle\ ;\ \langle\ (0, 4, 3)\ ;\ \overline{(2, 7, 5)}\ \rangle\ \rangle$ (où un triplet surligné désigne un littéral négatif). Plus précisément :

Caml

On déclare le type `litteral` comme suit :

```
type litteral = X of int * int * int | NonX of int * int * int ; ;
```

Dans un triplet d'entiers de type `int * int * int`, le premier élément correspond au numéro de ligne, le second au numéro de colonne et le troisième à la valeur comprise entre 1 et 9. Une clause est de type `litteral list` et une formule logique (en forme normale conjonctive) de type `litteral list list`.

Pascal

On déclare le type `litteral` comme suit :

```
type litteral = record signe : boolean ; i : integer ; j : integer ; k : integer end ;
```

On dispose d'une fonction

```
litt(signe : boolean ; i : integer ; j : integer ; k : integer) : litteral
```

qui permet de créer des littéraux. Par exemple `litt(true, 0, 0, 9)` renvoie le littéral $x^9_{(0,0)}$ et `litt(false, 1, 4, 2)` renvoie le littéral $\neg\, x^2_{(1,4)}$. On suppose qu'on a défini les types `clause` et `formule` comme des listes pour lesquelles on dispose des primitives usuelles suivantes :

- `Nil` qui est la liste vide ; on peut tester si une liste `l` est vide avec l'expression `l=Nil` ;
- `tete(c : clause) : litteral` et `tete(f : formule) : clause` qui permettent d'obtenir le premier élement d'une liste ;
- `queue(c : clause) : clause` et `queue(f : formule) : formule` qui permettent d'obtenir la queue d'une liste ;
- `ajout_en_tete(l : litteral ; c : clause) : clause` et `ajout_en_tete(c : clause ; f : formule) : formule` qui permettent d'ajouter un élément en tête d'une liste ;
- `concatener(f1 : formule ; f2 : formule) : formule` qui permet de concaténer deux listes, cette fonction a un coût linéaire en la taille de la première liste.

On suppose que l'on peut tester l'égalité de deux littéraux `l1` et l2 à l'aide de l'expression `l1 = l2`. Cette opération n'est pas disponible pour les clauses et les formules.

Avant de décrire des stratégies de résolution d'une grille de sudoku, on va coder l'information contenue dans la grille initiale par une formule logique en forme normale conjonctive. Ce codage se fait en deux temps : on code déjà la règle générale du jeu, représentée par les quatre règles logiques (K), (L), (C) et (B) ; puis on code l'information propre à la grille initiale fournie, qui repose sur la cinquième règle (I).

II.A – Formule logique décrivant la règle du jeu

On s'intéresse dans cette partie à coder par une formule logique en forme normale conjonctive l'information fournie par la règle du jeu.

Chacune des quatre règles logiques (K), (L), (C) et (B) peut être scindée en deux sous-règles :
(K1) toute case de F contient au moins une fois l'un des chiffres 1 à 9 ;
(L1) toute ligne de F contient au moins une fois chacun des chiffres 1 à 9 ;
(C1) toute colonne de F contient au moins une fois chacun des chiffres 1 à 9 ;
(B1) tout bloc de F contient au moins une fois chacun des chiffres 1 à 9 ;
(K2) toute case de F contient au plus une fois l'un des chiffres 1 à 9 ;
(L2) toute ligne de F contient au plus une fois chacun des chiffres 1 à 9 ;
(C2) toute colonne de F contient au plus une fois chacun des chiffres 1 à 9 ;
(B2) tout bloc de F contient au plus une fois chacun des chiffres 1 à 9.

II.A.1)

a) Pour $(i,j) \in [\![0,8]\!]^2$, écrire une clause qui traduit la phrase mathématique : $\exists\, k \in [\![1,9]\!], x^k_{(i,j)}$.

b) Traduire la condition (K1) par une phrase mathématique.

c) En déduire une formule $K1$ en forme normale conjonctive qui exprime la condition (K1).

d) Déterminer le nombre de clauses que contient $K1$.

e) Écrire une fonction `case1` qui ne prend pas d'argument et renvoie la liste qui représente la formule logique $K1$.

II.A.2) Traduire la condition (L1) par une phrase mathématique et en déduire une formule $L1$ en forme normale conjonctive qui exprime (L1).

II.A.3)

a) Obtenir de même des formules logiques $C1$ et $B1$ exprimant les conditions (C1) et (B1).

b) Écrire une fonction `bloc1` qui renvoie la liste qui représente la formule logique $B1$.

II.A.4)

a) Pour $(i,k) \in [\![0,8]\!] \times [\![1,9]\!]$, exprimer mathématiquement le fait que la ligne de numéro i de la grille contient au plus une fois la valeur k et écrire une formule en forme normale conjonctive qui traduit cette condition.

b) En déduire une phrase mathématique qui traduit la condition (L2) et une formule $L2$ en forme normale conjonctive qui exprime (L2).

c) Déterminer le nombre de clauses que contient $L2$.

d) Écrire une fonction `lig2` qui renvoie la liste qui représente la formule logique $L2$.

II.A.5) Obtenir de même des formules logiques $C2$, $B2$ et $K2$ exprimant les conditions (C2), (B2) et (K2).

On suppose avoir écrit des fonctions `lig1`, `col1`, `case2`, `col2` et `bloc2` qui renvoient respectivement les listes qui représentent les formules logiques $L1$, $C1$, $K2$, $C2$ et $B2$.

On pose alors $F_{\text{règle}} = K1 \wedge L1 \wedge C1 \wedge B1 \wedge K2 \wedge L2 \wedge C2 \wedge B2$: $F_{\text{règle}}$ est la formule logique en forme normale conjonctive décrivant la règle du jeu. Elle contient 11988 clauses.

II.B – Formule logique décrivant la grille initiale

On s'intéresse dans cette partie à coder par une formule logique en forme normale conjonctive l'information fournie par la grille initiale à compléter.

On pourrait pour cela se contenter de considérer une formule logique F construite à partir de la cinquième règle (I) donnée au début de l'énoncé : pour toute case de la grille d'indice (i,j) initialement remplie par la valeur $k \in [\![1,9]\!]$, on considère la clause réduite au littéral positif $x^k_{(i,j)}$ et, si r est le nombre de cases remplies dans la grille initiale, on pose F comme étant la conjonction des r clauses ainsi obtenues.

Mais, pour accélérer la phase d'inférences logiques qui sera menée dans la partie III, on n'hésite pas à introduire à nouveau de la redondance en déduisant d'emblée un certain nombre de faits que l'on ajoute à la formule F liée à la grille initiale. Ces nouveaux faits sont de deux ordres :

— si dans la grille initiale, la case d'indice (i,j) est déjà remplie par la valeur $k \in [\![1,9]\!]$, on peut considérer d'emblée, en plus de la clause unitaire positive $x^k_{(i,j)}$ (déjà prise en compte par la formule F ci-dessus), les 8 clauses unitaires négatives $\neg\, x^l_{(i,j)}$ avec $l \in [\![1,9]\!] \setminus \{k\}$; on enrichit ainsi la formule F en une formule F_1

en forme normale conjonctive constituée de r clauses unitaires positives (celles de F) et $8r$ négatives (où r est le nombre de cases remplies dans la grille initiale) ;
- si, dans une grille initiale, une case d'indice (i,j) n'est pas remplie (c'est-à-dire est occupée par la valeur 0), alors on sait que cette case ne peut être remplie par aucune valeur $k \in [\![1,9]\!]$ apparaissant déjà dans la ligne i, ou dans la colonne j, ou dans le bloc auquel appartient la case d'indice (i,j) ; puisqu'une telle valeur k est interdite pour la case d'indice (i,j), on peut donc considérer la clause unitaire négative $\neg\, x^k_{(i,j)}$; ainsi, pour la case d'indice $(1,3)$ de l'exemple de la figure 1 à droite, il n'est pas question de remplacer la valeur 0 par aucune des valeurs appartenant à $\{2,3,4,5,7,9\}$: on peut donc ajouter les clauses $\neg\, x^2_{(1,3)}$, $\neg\, x^3_{(1,3)}$, $\neg\, x^4_{(1,3)}$, $\neg\, x^5_{(1,3)}$, $\neg\, x^7_{(1,3)}$ et $\neg\, x^9_{(1,3)}$; on note alors F_2 la formule en forme normale conjonctive constituée de la conjonction de toutes ces clauses unitaires négatives obtenues en parcourant toutes les cases non remplies de la grille initiale.

La formule décrivant la grille initiale est donc $F_{\text{grille}} = F_1 \wedge F_2$.

II.B.1) Écrire une fonction `donnees` qui, à partir d'une grille de sudoku initiale (donnée sous la forme d'un tableau), renvoie la formule F_1, toujours sous la forme d'une liste de listes de littéraux.

II.B.2)

a) Étant donnée une case d'indice $(i,j) \in [\![0,8]\!]^2$, exprimer en fonction de i et j le numéro de bloc $b \in [\![0,8]\!]$ auquel cette case appartient.

b) Écrire une fonction `interdites_ij` qui, étant données une grille de sudoku initiale et une case d'indice (i,j) non remplie de la grille initiale, renvoie la conjonction des clauses unitaires négatives correspondant aux valeurs interdites pour la case d'indice (i,j).

c) Écrire une fonction `interdites` qui, à partir d'une grille de sudoku initiale, renvoie la formule F_2.

II.B.3) Montrer que le nombre de clauses de la formule F_{grille} est majoré par 729.

Formule initiale complète

D'après II.A et II.B, la formule logique à associer à une grille initiale qui code à la fois les informations qu'elle contient et qui peut permettre de la résoudre est donc $F_{\text{initiale}} = F_{\text{grille}} \wedge F_{\text{règle}}$.

On supposera dans la partie III avoir écrit une fonction `formule_initiale` qui, à partir d'une grille initiale, renvoie la formule F_{initiale}.

Caml

Cette fonction a pour signature :

```
formule_initiale : int vect vect -> litteral list list = < fun >
```

Pascal

Cette fonction a pour en-tête :

```
function formule_initiale (t : array of integer) : formule ;
```

III Résolution d'une grille de sudoku

III.A – Règle de propagation unitaire

Nous supposons désormais que nous disposons d'une formule F_{initiale} en forme normale conjonctive encodant un sudoku. Pour le résoudre on cherche à satisfaire F_{initiale}.

III.A.1) Combien existe-t-il de valuations satisfaisant F_{initiale} ?

III.A.2) Déterminer le nombre de lignes de la table de vérité de F_{initiale}.

Il semble donc illusoire de construire une telle table de vérité en un temps raisonnable. Nous proposons donc une méthode qui n'est pas assurée de conclure dans tous les cas mais qui utilise un nombre polynomial d'opérations.

Soit F une formule en forme normale conjonctive. On suppose que F contient une clause unitaire réduite au littéral l. Un tel littéral est appelé *littéral isolé*. Nous pouvons alors *simplifier* la formule F de la manière suivante :
- en supprimant toutes clauses de F qui contiennent l ;
- en supprimant $\neg\, l$ de toutes les clauses de F (ainsi si une clause ne contient que $\neg\, l$ elle est remplacée par la clause vide $\bot$)

Par exemple la formule $F = (\neg\, x^3_{(0,2)}) \wedge (\neg\, x^3_{(0,2)} \vee x^6_{(2,5)}) \wedge (x^3_{(0,2)} \vee \neg\, x^7_{(1,4)})$ contient un unique littéral isolé $\neg\, x^3_{(0,2)}$. En simplifiant F par ce littéral, on obtient la formule $F' = \neg\, x^7_{(1,4)}$.

III.A.3) On justifie formellement cette simplification. Soit F une formule en forme normale conjonctive contenant un littéral isolé l (associé à la variable propositionnelle p : on a donc $l = p$ ou $l = \neg p$). F peut s'écrire alors sous la forme $F = l \wedge F_1 \wedge F_2 \wedge F_3$, où :
- F_1 est une formule constituée de clauses contenant chacune le littéral l ;
- F_2 est une formule constituée de clauses contenant chacune le littéral $\neg\, l$ et ne contenant pas le littéral l ;
- F_3 est une formule constituée de clauses ne contenant chacune ni le littéral l, ni le littéral $\neg\, l$.

Remarquons que chacune des formules F_1, F_2 et F_3 peut être réduite à la formule vide, satisfiable.

La formule simplifiée de F par le littéral l est alors la formule $F' = F'_2 \wedge F_3$, où F'_2 s'obtient à partir de F_2 en supprimant toutes les occurrences du littéral $\neg\, l$ dans chacune des clauses de F_2.

Soit σ une valuation de $\mathcal{V} \setminus \{p\}$. Montrer que σ satisfait F' si et seulement s'il existe une valuation $\overline{\sigma}$ de $\mathcal{V}$ prenant les mêmes valeurs que σ sur $\mathcal{V} \setminus \{p\}$ qui satisfait F. On précisera la valeur de $\overline{\sigma}(p)$.

L'algorithme de résolution du sudoku que nous proposons est appelé *algorithme de propagation unitaire* et consiste à appliquer la simplification présentée ci-dessus de manière répétée tant que l'on peut déduire de nouveaux littéraux isolés.

III.A.4) Appliquer l'algorithme de propagation unitaire à la formule

$$F = x^1_{(0,0)} \wedge \left(x^4_{(2,2)} \vee x^6_{(3,6)} \vee x^7_{(7,7)}\right) \wedge \left(\neg\, x^1_{(0,0)} \vee \neg\, x^6_{(3,6)}\right)$$

III.A.5)

a) Sur l'exemple de la figure 1, déterminer la valeur k_0 de la case d'indice $(0, 7)$ dans la grille finale en raisonnant à la manière d'un joueur de sudoku (le raisonnement suivi sera décrit).

b) Retrouver le résultat k_0 de la question précédente en appliquant l'algorithme de propagation unitaire, c'est-à-dire montrer que l'on peut, au terme d'un certain nombre de simplifications à partir de la formule F_{initiale} associée à cette grille initiale, déduire le littéral isolé $x^{k_0}_{(0,7)}$. On ne sélectionnera dans la formule F_{initiale} que des clauses utiles à l'obtention du littéral isolé $x^{k_0}_{(0,7)}$.

III.A.6) Écrire une fonction `nouveau_lit_isole` qui, à partir d'une formule F, renvoie un littéral isolé de F. Si F ne contient pas de tel littéral, la fonction renverra le littéral $x^{-1}_{(-1,-1)}$ qui n'est pas utilisé comme variable propositionnelle par ailleurs.

III.A.7) Écrire une fonction `simplification` qui, à partir d'un littéral l et d'une formule F, renvoie la formule simplifiée F' après propagation du littéral l dans F.

III.A.8) Écrire une fonction `propagation` qui, à partir d'un tableau t représentant le sudoku et d'une formule F représentant les contraintes du sudoku, renvoie la formule obtenue par propagation unitaire. Cette fonction modifiera le tableau t quand de nouveaux littéraux isolés découverts au cours de l'algorithme permettent de déduire la valeur d'une case du sudoku.

III.A.9) Que peut-on dire sur le tableau modifié t et la formule renvoyée par la fonction `propagation` dans le cas où l'algorithme de propagation unitaire permet de résoudre le sudoku ? Et dans le cas où il ne permet pas de le résoudre ?

III.A.10) On appelle taille d'une formule F le nombre total de littéraux apparaissant dans ses clauses. Évaluer le nombre d'opérations effectuées par les fonctions `nouveau_lit_isole`, `simplification`, puis par la fonction `propagation` quand elles s'appliquent à une formule de taille n. On donnera une estimation de la forme $O(f(n))$ que l'on justifiera.

III.B – Règle du littéral infructueux

On décrit maintenant une autre méthode de déduction plus puissante combinant la propagation unitaire et une opération appelée *règle du littéral infructueux* décrite ci-dessous.

Étant donné une formule F en forme normale conjonctive et une variable propositionnelle x,
- si l'algorithme de propagation unitaire appliqué à la formule $F \wedge \neg\, x$ permet de déduire la clause vide alors on ajoute la clause x à F ;
- si l'algorithme de propagation unitaire appliqué à la formule $F \wedge x$ permet de déduire la clause vide alors on ajoute la clause $\neg\, x$ à F.

III.B.1) Justifier formellement que si l'on peut déduire la clause vide à partir de la formule $F \wedge \neg\, x$ alors $F \equiv F \wedge x$.

III.B.2) Écrire une fonction `variables` qui, à partir d'une formule, renvoie la liste de ses variables sans doublons.

On supposera qu'on dispose d'une fonction `flatten` qui à partir d'une formule f renvoie la clause formée de tous les éléments des sous-listes de f. Par exemple `flatten` appliqué à $\langle\ \langle\ (1,8,7)\ \rangle\ ;\ \langle\ \overline{(2,7,5)}\ ;\ (1,8,7)\ \rangle\ \rangle$ renvoie $\langle\ (1,8,7)\ ;\ \overline{(2,7,5)}\ ;\ (1,8,7)\ \rangle$.

III.B.3) Écrire une fonction `deduction` qui, à partir d'un tableau, d'une variable x et d'une formule F, renvoie 1 si la règle du littéral infructueux permet d'ajouter la clause x à F, -1 si elle permet d'ajouter la clause $\neg\, x$ à F et 0 sinon.

On supposera qu'on dispose d'une fonction `copier_matrice` qui à partir d'un tableau renvoie un autre tableau distinct contenant les mêmes valeurs.

III.B.4) Nous proposons un deuxième algorithme de propagation basé sur la règle du littéral infructueux. Celui-ci consiste à appliquer la propagation unitaire et, quand celle-ci ne permet plus de déduire de nouvelles clauses, à appliquer la règle du littéral infructueux pour obtenir une nouvelle clause unitaire de la forme x ou $\neg\, x$. Dès lors on peut reprendre la propagation unitaire. Le processus s'arrête lorsque ni la propagation unitaire ni la règle du littéral infructueux ne permettent de déduire de nouvelles clauses.

Écrire une fonction `propagation2` qui, à partir d'un tableau t représentant le sudoku et d'une formule F représentant les contraintes du sudoku, met en œuvre cet algorithme. Cette fonction modifiera le tableau t selon la valeur des cases pouvant être déduites.

III.B.5) Écrire une fonction `sudoku` qui, à partir d'un sudoku donné sous forme d'un tableau t, modifie t et renvoie la grille complétée au maximum en utilisant les techniques précédentes.

Expérimentalement, la règle de propagation unitaire permet de résoudre les sudokus les plus faciles et environ la moitié des sudokus les plus difficiles. À notre connaissance, il n'existe pas de sudoku ne pouvant être résolu intégralement à l'aide de la règle du littéral infructueux.

• • • FIN • • •
