

# Résolution d'un sudoku par calcul booléen

## 1 Présentation du jeu de sudoku

### 1.1 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  $3 \times 3$  appelées blocs. 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.

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.

### 1.2 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  $(i, j) \in \llbracket 0, 8 \rrbracket^2$ , 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  $(b, r) \in \llbracket 0, 8 \rrbracket^2$ , 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.

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

### 1.3 Préliminaires

1. Écrire une fonction **appartient** d'appartenance d'un élément à une liste.
2. Écrire une fonction **suppression** de suppression de toutes les occurrences d'un élément dans une liste.
3. Écrire une fonction **ajout** 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).
4. Écrire une fonction **indice** qui, étant donné un couple  $(b, r) \in \llbracket 0, 8 \rrbracket^2$  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  $(5, 0)$ .

La fonction aura pour signature `indice : int * int -> int * int = < fun >`.

## 2 Codage de la formule initiale

### 2.1 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  $V$  de variables propositionnelles, on considère l'ensemble  $F$  des formules logiques construites à partir de  $V$  et de l'ensemble de connecteurs  $\{\wedge, \vee, \neg\}$ .

Une valuation sur  $V$  est une application  $\sigma : V \rightarrow B$  (où  $B = \{vrai, faux\}$  désigne l'ensemble des booléens) ; cette application  $\sigma$  s'étend en une application  $Ev_\sigma : F \rightarrow B$ , attribuant une valeur de vérité à toute formule  $f \in F$  sous la valuation  $\sigma$ . Une formule logique  $f$  est satisfiable s'il existe une valuation  $\sigma$  telle que  $Ev_\sigma(f) = vrai$ . Deux formules logiques  $f$  et  $g$  sont équivalentes si  $Ev_\sigma(f) = Ev_\sigma(g)$  pour toute valuation  $\sigma$ , 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  $\perp$  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  $x_{(i,j)}^k$ , avec la sémantique suivante : pour un triplet  $(i, j, k) \in \llbracket 0, 8 \rrbracket \times \llbracket 0, 8 \rrbracket \times \llbracket 1, 9 \rrbracket$ , une valuation  $\sigma$  assigne la valeur vrai à  $x_{(i,j)}^k$  lorsque la case d'indice  $(i, j)$  de la grille de sudoku contient la valeur  $k$ . L'ensemble  $V$  utilisé dans le problème est donc constitué des  $9^3 = 729$  variables propositionnelles  $x_{(i,j)}^k$ , où  $(i, j, k)$  parcourt  $\llbracket 0, 8 \rrbracket \times \llbracket 0, 8 \rrbracket \times \llbracket 1, 9 \rrbracket$ .

À titre d'exemples :

- La clause  $x_{(0,0)}^5 \vee x_{(0,8)}^5 \vee x_{(8,0)}^5 \vee x_{(8,8)}^5$  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  $\bigwedge_{i=0}^8 \left( \bigvee_{k=1}^9 x_{i,7}^k \right)$  exprime que chacune des valeurs de la colonne 7 est constituée de chiffres compris entre 1 et 9 ;
- La clause  $\bigvee_{j=0}^8 x_{\text{indice}3j}^6$  exprime que l'une des cases du bloc numéro 3 de la grille contient le chiffre 6 (où **indice** est la fonction écrite partie 1.3).

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, la formule  $x_{1,8}^7 \wedge (x_{0,4}^3 \vee \neg x_{2,7}^5)$  sera représentée par la liste  $[(1, 8, 7); [(0, 4, 3); \overline{(2, 7, 5)}]]$  (où un triplet surligné désigne un littéral négatif). Plus précisément On déclare le type littéral comme suit :

---

```
1 type littéral = 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 `littéral list` et une formule logique (en forme normale conjonctive) de type `littéral list list`.

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).

## 2.2 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.

1. (a) Pour  $(i, j) \in \llbracket 0, 8 \rrbracket^2$ , écrire une clause qui traduit la phrase mathématique :  $\exists k \in \llbracket 1, 9 \rrbracket, x_{(i,j)}^k$ .  
 (b) Traduire la condition (K1) en une phrase mathématique.  
 (c) En déduire une formule  $K1$  en forme normale conjonctive qui exprime la condition (K1).  
 (d) Déterminer le nombre de clauses qui contient  $K1$ .  
 (e) Écrire une fonction **case1** qui ne prend pas d'argument et renvoie la liste qui représente la formule logique  $K1$ .
2. Traduire la condition (L1) par une phrase mathématique et en déduire une formule  $L1$  en forme normale conjonctive qui exprime (L1).
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  $B1$ .
4. *Condition (L2)*  
 (a) Soit  $(i, k) \in \llbracket 0, 8 \rrbracket \times \llbracket 1, 9 \rrbracket$ . Exprimer mathématiquement le fait que la ligne  $i$  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 **ligne2** qui renvoie la liste qui représente  $L2$ .
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.

## 2.3 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  $k \in \llbracket 1, 9 \rrbracket$ , on considère la clause réduite au littéral positif  $x_{(i,j)}^k$  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 3, 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  $k \in \llbracket 1, 9 \rrbracket$ , on peut considérer d'emblée, en plus de la clause unitaire positive  $x_{(i,j)}^k$  (déjà prise en compte par la formule  $F$  ci-dessus), les 8 clauses unitaires négatives  $\neg x_{(i,j)}^l$  avec  $l \in \llbracket 1, 9 \rrbracket \setminus \{k\}$  ; 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  $k \in \llbracket 1, 9 \rrbracket$  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  $\neg x_{(i,j)}^k$ ; 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  $\neg x_{(1,3)}^2$ ,  $\neg x_{(1,3)}^3$ ,  $\neg x_{(1,3)}^4$ ,  $\neg x_{(1,3)}^5$ ,  $\neg x_{(1,3)}^7$  et  $\neg x_{(1,3)}^9$ ; 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  $F_{\text{grille}} = F_1 \wedge F_2$ .

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.
2. (a) Étant donnée une case d'indice  $(i, j) \in \llbracket 0, 8 \rrbracket^2$ , exprimer en fonction de  $i$  et  $j$  le numéro de bloc  $b \in \llbracket 0, 8 \rrbracket$  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  $F_2$ .
3. Montrer que le nombre de clauses de la formule  $F_{\text{grille}}$  est majoré par 729.

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 suppose dans la suite avoir écrit une fonction **formule\_initiale** qui, à partir d'une grille initiale, renvoie la formule  $F_{\text{initiale}}$ .

## 3 Résolution

### 3.1 Propagation unitaire

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

1. Combien existe-t-il de valuations satisfaisant  $F_{\text{initiale}}$  ?
2. Déterminer le nombre de lignes de la table de vérité de  $F_{\text{initiale}}$ .

*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  $\perp$ ).

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

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

*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.*

4. Appliquer l'algorithme de propagation unitaire à la formule  $F$  ci-dessous :

$$x_{0,0}^1 \wedge (x_{2,2}^4 \vee x_{3,6}^6 \vee x_{7,7}^7) \wedge (\neg x_{0,0}^1 \vee \neg x_{3,6}^6).$$

5. (a) Sur l'exemple de la figure 1, déterminer la valeur de la case (0, 7) en raisonnant à la manière d'un joueur de Sudoku.  
(b) Retrouver ce résultat 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 initiale, déduire le littéral isolé  $x_{0,7}^7$ .  
On ne sélectionnera dans la formule  $F_{initiale}$  que des clauses utiles à l'obtention du littéral isolé  $x_{0,7}^7$ .
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.
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  $F$ .
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.
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 ?
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.

## 3.2 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  $F$ .

1. Justifier formellement que si l'on peut déduire la clause vide à partir de  $F \wedge \neg x$ , alors  $F \equiv F \wedge x$ .
2. Écrire une fonction **variables** qui, à partir d'une formule, renvoie la liste de ses variables sans doublons. *On rappelle que la fonction **List.flatten** prend une liste de listes et renvoie la concaténation de celles-ci.*
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 **copie\_matrice** qui à partir d'un tableau renvoie un autre tableau distinct contenant les mêmes valeurs.*

*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.

4. 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.

*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.*

*Remarque :* La méthode proposée ici consiste à faire une hypothèse et à voir si on aboutit à une contradiction. En général, cela suffit à terminer le sudoku. Cependant, il pourrait y avoir des cas de figure où il faudrait faire une deuxième hypothèse au sein de la première pour aboutir à une contradiction. Voir une hypothèse dans l'hypothèse dans l'hypothèse...

Ainsi, pour obtenir un algorithme qui conclut à coup sûr on devrait utiliser une fonction récursive, qui émet une hypothèse puis se ré-appelle elle-même pour voir si elle aboutit à une contradiction. Dans la méthode ci-dessus, la fonction ne se ré-appelle pas elle-même mais appelle la fonction **propagation** plus simple de la partie précédente.

La méthode générale récursive s'appelle une méthode « backtracking ».