Retour sur trace et révisions de logique propositionnelle

1 Sudoku

Une grille de *Sudoku* est une grille de taille 9×9, contenant dans chaque case un entier de 1 à 9, telle que :

- chaque ligne ne contient pas deux fois la même valeur
- chaque colonne ne contient pas deux fois la même valeur
- si l'on décompose la grille en neuf sous-grille de taille 3×3, chaque sous-grille ne contient pas deux fois la même valeur

Initialement, certaines cases sont remplies, et leur contenu ne doit pas être modifié. Le but est de remplir les cases restantes en respectant les contraintes.

On représentera, à un moment donné, la grille par un tableau à deux dimensions d'entiers (`int array array`) où 0 indiquera une case encore non remplie.

1. Proposer une fonction `possibles` de signature `int array array -> int -> int -> int list` prenant en argument une grille en cours de remplissage et des coordonnées d'une case contenant un 0, et retournant la liste des valeurs qu'il est permis de mettre dans la case.

2. En déduire une fonction `resout` de signature `int array array -> bool` indiquant si la grille fournie en paramètre peut être résolue (si c'est le cas, la grille fournie en paramètre contiendra la solution). On pourra tester la fonction sur la grille de gauche ci-dessous :

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

3. Modifier la fonction pour déterminer le nombre de solutions qui existent pour la grille fournie en paramètre.

4. Envisager des stratégies pour gagner en rapidité (on ne demande pas d'implémentation).

2 Arbres logiques

On dispose d'un arbre représentant une formule propositionnelle f, dont les feuilles sont les variables logiques et/ou les constant $\top$ et $\bot$, et les nœuds interne les opérateurs $\wedge$, $\vee$ et $\neg$.

1. Proposer une méthode permettant de construire l'arbre représantant $\neg f$.

2. Implémenter cette négation en OCaml pour une formule propositionnelle de type

```
type 'a form =
  | V | F | Var of 'a
  | Conj of 'a form * 'a form
  | Disj of 'a form * 'a form
  | Neg of 'a form
```

3 Tables de vérité

1. Construire la table de vérité de $(\mathscr{A} \vee \mathscr{B} \wedge \mathscr{C}) \to (\mathscr{B} \leftrightarrow \neg\mathscr{C})$.

2. En déduire une formule propositionnelle équivalente sous *forme normale conjonctive* et une formule propositionnelle équivalente sous *forme normale disjonctive*.

3. Quelle est la *hauteur logique* et la *taille* de chacune des formules propositionnelles construites ?

4 Tautologies, Antilogies

Dans la liste suivante, déterminer lesquelles sont des *tautologies*, lesquelles sont des *antilogies*, lesquelles sont *satisfiables*, et proposer un modèle dans le cas des formules satisfiables.

- $\mathscr{B} \to (\mathscr{A} \to \mathscr{B})$;
- $\neg((\mathscr{A} \to \mathscr{B}) \vee (\mathscr{A} \oplus))$;
- $\neg((((\mathscr{A} \to \mathscr{B}) \to \mathscr{C}) \leftrightarrow (\mathscr{A} \to (\mathscr{B} \to \mathscr{C}))))$;
- $(\mathscr{A} \to \mathscr{B}) \vee (\mathscr{B} \to \mathscr{C})$;

5 Preuve

Montrer que

$$\{\mathscr{A} \wedge \mathscr{B} \to \mathscr{C},\ \mathscr{A},\ \neg\mathscr{C}\} \vDash \neg\mathscr{B}$$

6 Systèmes complets

Un *système complet* est un ensemble d'opérateurs (et possiblement de constantes) logiques tel que pour toute formule propositionnelle f sur des variables logiques v_i on puisse trouver une formule propositionnelle f' équivalente sur ces mêmes variables logiques v_i n'utilisant que ces opérateurs et constantes.

1. Justifier que $\{\rightarrow, \bot\}$ est un système complet.

2. Justifier que $\{\oplus, \top\}$ est un système complet.

7 Algorithme de Quine

On définit le type suivant pour des formules propositionnelle (en utilisant le fait que $\{\wedge, \neg\}$ est un système complet)

```
type form = V | F | Var of int
  | Conj of form * form | Neg of form
```

1. Proposer une fonction `simplifie` de signature `form -> form` prenant en argument une formule propositionnelle et retournant une formule propositionnelle équivalente qui soit $\top$, $\bot$, ou ne contienne ni `V`, ni `F`.

2. Proposer une fonction `substitue` de signature `form -> int -> bool -> form` prenant en argument une formule propositionnelle, un entier i et un booléen, et renvoie la formule propositionnelle dans laquelle v_i a été substitué par le booléen fourni.

3. Proposer une fonction `sat` de signature `form -> int -> bool list option` prenant en argument une formule propositionnelle sur des variables propositionnelles v_i indicées de 0 à $n-1$ et un entier n, et renvoyant `None` si la formule propositionnelle n'est pas satisfiable, et un modèle sinon, sous forme d'un liste de longueur n de booléens où le booléen en position j correspond à la valeur de v_j dans le modèle.

On notera que l'on peut voir l'algorithme de Quine comme une méthode de retour sur trace où l'on effectue une simplification de la formule à chaque étape de l'exploration.

8 Preuves

Soit f, g et h trois formules propositionnelles.

1. Montrer que $f \rightarrow (\neg h \rightarrow \neg g)$ est une tautologie et et seulement si $g \rightarrow (f \rightarrow h)$ en est une.

2. Montrer que $(f \rightarrow g) \wedge (f \rightarrow h)$ est une tautologie et et seulement si $f \rightarrow (g \wedge h)$ en est une.