TP n°4 - Révisions de logique

Exercice 1

On considère la formule $\psi = (a \wedge \neg b) \vee (\neg a \wedge b)$.

1. Constiture l'arbre syntaxique de ψ.
2. Établir la table de vérité de ψ.
3. Proposer une forme normale disjonctive de ψ.
4. Proposer une forme normale conjonctive de ψ.

Exercice 2

On considère la formule $\Phi = ((a \Rightarrow \neg b) \Rightarrow \neg a) \wedge c$.

1. Construre l'arbre syntaxique de Φ.
2. Établir la table de vérité de Φ.
3. Proposer une forme normale disjonctive de Φ.
4. Proposer une forme normale conjonctive de Φ.

Exercice 3

1. Proposer un type OCaml pour représenter les formules logiques. (On doit pouvoir associer un nom de type `string` à une variable)
2. Représenter les arbres syntaxiques des formules logiques suivantes, puis les coder en OCaml.
   (a) $f_1 : a \vee (b \wedge c)$,
   (b) $f_2 : (a \wedge \neg b) \vee (b \wedge \neg(c \vee a))$,
   (c) $f_3 : \neg a \vee (a \Rightarrow b)$,
   (d) $f_4 : (a \wedge (b \Leftrightarrow c))$,

Exercice 4

On va représenter les valuations par une liste de couples (nom de variable, valeur booléenne).

1. Écrire une fonction `evalue` qui, étant données une valuation et une formule, évalue la formule pour cette valuation.
2. Écrire une fonction `variables` qui, étant donné une formule, renvoie la liste des variables qui apparaissent à l'intérieur. Attention : on ne veut pas de doublons !

Pour énumérer les valuations, on définit un ordre dessus par analogie aux nombres binaires. On considère `False` comme 0 et `True` comme 1. La première valuation à 3 variables a, b, c est donc `["a", false; "b", false; "c", false]` (comme 000), la deuxième `["a", true; "b", false; "c", false]` (comme 001), la troisième `["a", false; "b", true; "c", false]` (comme 010) et la dernière `["a", true; "b", true; "c", true]` (comme 111).

3. Écrire une fonction `next` qui, étant donné une valuation, renvoie la prochaine valuation.
4. Écrire une fonction `bruteforce` (d'une formule) qui trouve une valuation qui satisfait la formule.
5. Écrire une fonction `bruteforce_nb` (d'une formule) qui compte le nombre de valuations qui satisfont la formule.
6. Écrire une fonction `tautologie` (d'une formule), qui décide si la formule est une tautologie
7. Écrire une fonction `pretty_print` (d'une valuation), qui affiche la valuation de manière jolie. Exemple d'exécution de la fonction :

```
pretty_print [``a'', false; "b", true]
a --> 0  b --> 1
-: unit=()
```

8. Améliorer la fonction `tautologie` pour qu'elle renvoie un contre-exemple, c'est à dire une valuation telle que la formule ne s'évalue pas à vrai. Indice : on peut utiliser une exception.
9. Écrire une fonction `forme_normale` qui met une formule sous forme normale conjonctive.