TP n°20 – Logique propositionnelle et algorithme de Quine

Dans ce TP, on représente les formules logiques par le type Ocaml suivant (noter que la représentation d'une formule logique en C serait beaucoup moins élégante) :

```
type formule =
| Top
| Bot
| V of int (* Plutôt qu'un nom, on donne un numéro à chaque variable, entre 0 et n *)
| Et of formule * formule
| Ou of formule * formule
| Imp of formule * formule
| Non of formule
```

On notera qu'il s'agit d'une représentation sous forme d'arbres et on adoptera le vocabulaire correspondant dans la suite.

On utilise des entiers pour désigner les variables $x_0, \dots, x_n$, ce qui nous permet de facilement représenter les valuations par des tableaux de booléens.

```
type valuation = bool array
```

Par exemple la valuation v qui à x_0 associe V, à x_1 associe F et à x_2 associe F est représenté par `[|true;false;false|]`

- **Q1.** Définir la formule φ suivante en Ocaml : $x_0 \wedge (\neg x_1 \vee x_2)$

1 Algorithme en force brute pour SAT

- **Q2.** On définit la taille $|\varphi|$ d'une formule φ comme le nombre total de nœuds (internes ou non) qu'elle contient. Écrire la fonction `taille : formule -> int`
- **Q3.** Écrire une fonction `var_max : formule -> int` telle que l'appel `var_max phi` renvoie le plus grand entier i tel que la formule φ contienne un nœud `V i`. On renverra -1 si φ ne contient aucune variable.
- **Q4.** Écrire une fonction `evalue` qui prend en entrée une formule φ et une valuation v et renvoyant $[|\varphi|]_v$. On pourra supposer sans le vérifier que le tableau fourni est de longueur au moins `var_max phi + 1`.
- **Q5.** À une valuation v sur n variables, on peut associer un entier $x = \sum_{i=0}^{n} v_i 2^i$ où $v_i = 1$ si $v(x_i)$ vaut `true` et $v_i = 0$ sinon.
  Écrire une fonction `suivant : valuation -> unit` qui prend en entrée une valuation correspondant à une valeur x et la modifie pour qu'elle corresponde après l'appel à l'entier $x + 1$. Si $x = 2^n - 1$, cette fonction lèvera l'exception `Derniere` (définie par `exception Derniere`), et le contenu du tableau après l'appel n'aura pas d'intérêt.
- **Q6.** Écrire une fonction `satisfiable_brute : formule -> bool` qui détermine si une formule est satisfiable, en essayant toutes les valuations possibles jusqu'à les épuiser ou en trouver une convenable.
- **Q7.** Déterminer la complexité dans le pire cas de la fonction `satisfiable_brute`, en fonction de la taille $|\varphi|$ de la formule φ et son nombre n de variables. On supposera que les variables de φ sont numérotées consécutivement à partir de zéro.

2 Algorithme de Quine

- **Q8.** En utilisant les règles de simplification de Quine, écrire une fonction `elimine_constantes : formule -> formule` qui prend en entrée une formule φ et renvoie une formule φ' telle que $\varphi' \equiv \varphi$ et :
  - soit φ' est réduite à $\top$ ou $\bot$;
  - soit φ' ne contient aucun $\top$ et aucun $\bot$.
- **Q9.** Écrire une fonction `substitue : formule -> int -> formule -> formule` telle que l'appel `substitue phi i psi` (où φ et ψ sont des formules et i un numéro de variable) renvoie la formule $\varphi[x_i \backslash \psi]$
- **Q10.** Implémenter l'algorithme de Quine. Tester.

3 Tautologie

- **Q11.** Écrire une fonction qui détermine si une formule est une tautologie. On pourra adapter pour s'entraîner la méthode exhaustive et l'algorithme de Quine.