TP 11 : Logique du premier ordre, algorithme de Quine

Exercice 1 – Retour à la logique propositionnelle

▶ **Question 1** Proposez un type `formule` des formules propositionnelles, comprenant la possibilité d'écrire les formules $\top$, $\bot$, et indexant les variables propositionnelles par des entiers naturels.

On utilisera le type `type valuation = bool array` pour représenter une valuation sur les variables propositionnelles $p_0, \dots, p_{n-1}$, avec n la taille du tableau.

▶ **Question 2** Soit $\varphi = (\bot \vee p_0) \to (\neg p_1 \wedge p_2 \vee p_0) \wedge (\top \to p_2 \vee p_3)$. Écrivez-la dans une variable `exemple : formule`.

▶ **Question 3** Écrire une fonction `taille : formule -> int` qui retourne la taille de la formule en entrée, c'est-à-dire le nombre de nœuds de l'arbre syntaxique associé.

▶ **Question 4** Écrire une fonction `ind_var_max : formule -> int` qui calcule l'indice maximal des variables propositionnelles dans la formule en entrée. La fonction renverra -1 si la formule en entrée n'a pas de variable propositionnelle.

▶ **Question 5** Écrire une fonction `valeur_verite : formule -> valuation -> bool` qui, pour une formule φ et une valuation ν des variables propositionnelles de la formule, renvoie si $\nu \vDash \varphi$.

Exercice 2 – Satisfiabilité par force brute

On peut aussi voir une valuation comme un nombre binaire (positif), en interprétant `true` comme 1 et `false` comme 0. Ainsi, à une valuation ν sur $\mathcal{P} = [\![0, n-1]\!]$, on associe l'entier $x = \sum_{k=0}^{n-1} \nu(k) 2^i$.

▶ **Question 1** Écrire une fonction `incrementer_valuation : valuation -> unit` qui transforme *en place* la valuation en entrée en une valuation correspondant au nombre $x+1$.

▶ **Question 2** En déduire une fonction `satisfiable_brute : formule -> bool` qui résout le problème SAT.

▶ **Question 3** Combien de valuations existent pour n fixé ? Que peut-on en déduire ?

▶ **Question 4** De la même façon, écrire une fonction `equivalent : formule -> formule -> bool` qui vérifie que les deux formules en entrée sont équivalentes.

▶ **Question 5** Écrire la fonction `supprimer_impl : formule -> formule` qui enlève les implications de la formule grâce à l'équivalence $\varphi \to \psi \equiv \neg\varphi \vee \psi$ et la fonction `repousser_neg : formule -> formule` qui repousse les négations vers les variables propositionnelles.

★ **Question 6** Écrire une fonction `fnc : formule -> formule` qui calcule la forme normale conjonctive de la formule en entrée. La tester sur l'exemple.

Exercice 3 – Algorithme de Quine

Pour une formule de la logique propositionnelle, on peut toujours la *simplifier* pour obtenir une formule équivalente égale à $\top$, $\bot$ ou ne contenant ni l'une ni l'autre. Pour cela, il suffit d'utiliser les équivalences suivantes :

Définition 1 – Règles de simplification de Quine

$$\neg\top \equiv \bot \quad \varphi \vee \top \equiv \top \quad \varphi \vee \bot \equiv \varphi \quad \top \wedge \varphi \equiv \varphi \quad \bot \wedge \varphi \equiv \bot$$
$$\neg\top \equiv \bot \quad \top \vee \varphi \equiv \top \quad \bot \vee \varphi \equiv \varphi \quad \varphi \wedge \top \equiv \varphi \quad \varphi \wedge \bot \equiv \bot$$

▶ **Question 1** Écrire une fonction `simplifier_constantes : formule -> formule` qui renvoie une formule équivalente à l'entrée qui soit égale à $\top$, $\bot$ ou ne contient ni l'un ni l'autre. *On pourra supposer que les fonctions* `supprimer_impl` *puis* `repousser_neg` *ont déjà été appliquées à la formule en entrée.*

▶ **Question 2** Écrire une fonction `substitution : formule -> int -> formule -> formule`, telle que `substitution phi k psi` est évalué en la formule $\varphi\{p_k \mapsto \psi\}$[a].

▶ **Question 3** Appliquer la substitution à `exemple`, p_0 et la formule $p_1 \vee \neg p_2$.

L'idée de l'algorithme de Quine est la suivante :

1. on choisit une variable propositionnelle,
2. on lui affecte une valeur de vérité (en pratique, en la substituant par une constante $\top$ ou $\bot$),
3. on simplifie la formule avec les règles de la Définition 1,
4. on applique récursivement l'algorithme sur la formule obtenue, puis on revient en arrière pour choisir l'autre valeur de vérité

L'algorithme peut être vu comme la construction d'un *arbre de décision* : un arbre dans lequel chaque nœud étiqueté par k modélise le choix de la valeur de vérité de

p_k : le fils gauche correspond au cas $p_k \models$ `vrai`, et le fils droit le cas $p_k \models$ `faux`. Dans le poly du Chapitre 2, on la décrit en pseudocode. Pour l'implémenter, on va y préférer une version récursive :

Pseudo-code

Fonction QUINE(φ)
φ est supposée sans $\rightarrow$ et les $\neg$ sont descendus
  $\psi \leftarrow$ SIMP(φ)
  Si $\psi \notin \{\top, \bot\}$ **alors**
    Soit p une v.p. de ψ ▷ *qui existe, sinon ψ serait $\top$ ou $\bot$*
    retourne p (fils gauche : QUINE($\varphi\{p \mapsto \top\}$), fils droit : QUINE($\varphi\{p \mapsto \bot\}$))
  Sinon
    retourne $F(\psi)$

Maintenant, on a tout ce qu'il faut pour se lancer dans l'implémentation de l'algorithme de Quine, en utilisant les règles de la Définition 41. Pour représenter un arbre de décision, on va utiliser un type d'arbres binaires stricts, dont les feuilles sont étiquetées par des .

OCaml

```ocaml
type arbre_decision =
  | Feuille of bool
  | Noeud of arbre_decision * int * arbre_decision
```

▶ **Question 4** Écrire une fonction `construire_arbre_decision : formule -> arbre_decision`, et en déduire une fonction `satisfiable_quine : formule -> bool` qui résoud SAT.

▶ **Question 5** Proposer une heuristique[b] `choix_proposition : formule -> int` qui choisit une proposition qui sera la plus utile à substituer dans `construire_arbre_decision`.

▶ **Question 6** En déduire une fonction `construire_arbre_decision : formule -> arbre_decision` qui à une formule associe un arbre de décision.

▶ **Question 7** En choisissant toujours la proposition de plus grand index dans la formule d'entrée, proposer une suite de formules $\varphi_0, \varphi_1, \ldots$ telle que l'arbre de décision calculé est de taille exponentielle.

a. C'est-à-dire la formule dans laquelle on a remplacé toutes les occurrences de p_k par la formule ψ.

b. "une fonction qui choisit une proposition en regardant de loin la formule, cette proposition étant choisie intelligemment, intelligemment n'ayant pas de définition particulière ici"