TP de Programmation Fonctionnelle

On définit inductivement une *formule propositionnelle* comme étant soit une variable $\mathbf{X_i}$, soit la constante **vrai**, soit la constante **faux**, soit la **conjonction** (resp. **disjonction**, **implication**, **équivalence**) de deux formules propositionnelles, soit la **négation** d'une formule propositionnelle.

Le type `Ocaml` suivant permettra de représenter les formules propositionnelles :

```
# type fp = X of int | Vrai | Faux | Et of fp * fp | Ou of fp * fp | Imp of
fp * fp | Equiv of fp * fp | Non of fp ;;
```

Une formule propositionnelle est dite *en Forme Normale Conjonctive* (FNC) si l'arbre de syntaxe abstraite de cette formule possède la propriété suivante : sur le chemin qui va de la racine à toute feuille, on ne rencontre ni implication, ni équivalence et les connecteurs logiques se rencontrent dans l'ordre suivant : d'abord un certain nombre (possiblement nul) de conjonctions, puis un certain nombre (possiblement nul) de disjonctions et enfin au plus une négation.

Voici des exemples de (valeurs `Ocaml` représentant des) formules propositionnelles qui sont en FNC :

```
# X(3) ;;
# Non(Vrai) ;;
# Ou(Vrai,Faux) ;;
# Et(Ou(Non(Vrai),X(2)),Faux) ;;
```

et des exemples qui ne sont pas en FNC :

```
# Non(Non(X(3))) ;;
# Imp(Vrai, Faux) ;;
# Equiv(Faux,Faux) ;;
# Ou(Et(Non(Vrai),X(2)),Faux) ;;
```

Le but de l'exercice est de transformer toute formule propositionnelle en une formule propositionnelle logiquement équivalente (*i.e.* ayant même table de vérité) en FNC.

1) Définir une fonction `elim_equiv` qui transforme une formule propositionnelle en une autre logiquement équivalente qui ne contient pas d'équivalence.

2) Définir une fonction `elim_imp` qui transforme une formule propositionnelle issue de **1)** en une autre logiquement équivalente qui ne contient pas d'implication.

3) Définir une fonction `repousse_neg` qui transforme une formule propositionnelle issue de **2)** en une autre logiquement équivalente dont les négations possèdent la propriété demandée par la FNC (au plus une négation avant toute feuille et nulle part ailleurs).

4) Définir une fonction `inverse_ou_et` qui transforme une formule propositionnelle issue de **3)** en une autre logiquement équivalente en FNC.