(Partie 2) Résolvante et principe de résolution

Dans cette partie, on ne considère que des formules qui sont déjà sous forme clausale[1]. On introduit une seconde représentation où :

— une clause est représentée par une liste de littéraux,

— une formule (conjonction de clauses) est représentée par une liste de clauses.

En conséquence, on définit les types synonymes suivants :

```
type Clause = [Formule]
type FormuleBis = [Clause]
```

1. Transformer une `Formule` en une `FormuleBis`

(Q_9) Compléter les définitions des fonctions ci-dessous qui permettent de transformer une formule `f` *(déjà sous forme clausale)* en une `FormuleBis`, i.e. en une liste de clauses.

```
-- exemples

> ouToListe (Non (Var "b"))
[Non (Var "b")]

> etToListe (Non (Var "b"))
[ [Non (Var "b")] ]

> ouToListe (Ou (Non (Var "a")) (Non (Var "b")))
[Non (Var "a"), Non (Var "b")]

> etToListe (Ou (Non (Var "a")) (Non (Var "b")))
[ [Non (Var "a"), Non (Var "b")] ]

> ouToListe (Ou (Non (Var "a")) (Ou (Var "c") (Non (Var "b"))))
[Non (Var "a"), Var "c", Non (Var "b")]

> etToListe (Ou (Non (Var "a")) (Ou (Var "c") (Non (Var "b"))))
[ [Non (Var "a"), Var "c", Non (Var "b")] ]

> etToListe (Et (Ou (Non (Var "c")) (Var "d")) (Ou (Non (Var "b")) (Var "d")))
[ [Non (Var "c"), Var "d"], [Non (Var "b"), Var "d"] ]

> etToListe (Et (Ou (Var "a") (Non (Var "b")))
               (Et (Ou (Non (Var "c")) (Var "d"))
                  (Ou (Var "b") (Ou (Non (Var "d")) (Var "c")))))
[ [Var "a", Non (Var "b")], [Non (Var "c"), Var "d"], [Var "b", Non (Var "d"), Var "c"] ]


etToListe ::
etToListe (Et g d) = (ouToListe g) :
etToListe f        =


ouToListe ::
ouToListe (Ou g d) = g :
ouToListe          =
```

1. Si l'on souhaitait traiter une formule `f` quelconque, il suffirait de lui appliquer la fonction `(formeClausale f)` vue dans la première partie.

2. Résolvante de deux clauses

Définiton 2.1 (clauses liées) Deux clauses c_1 et c_2 sont dites liées ssi il existe (au moins) un littéral (positif ou négatif) apparaissant dans c_1 et tel que sa négation apparaisse dans c_2.

Exemples :
$c_1 = a \vee \neg b \vee d$ et $c_2 = a \vee \neg d \vee \neg b$ sont liées car le littéral d appartient à c_1 et sa négation $\neg d$ appartient à c_2.
$c_1 = \neg a \vee \neg b$ et $c_2 = \neg d \vee c \vee b$ sont liées car le littéral $\neg b$ appartient à c_1 et sa négation b appartient à c_2.
$c_1 = \neg a \vee b \vee c$ et $c_2 = \neg d \vee \neg a \vee c$ ne sont pas liées ; aucun littéral figurant dans c_1 n'a sa négation dans c_2.

(Q_{10}) Définir la fonction `(neg l)` qui à un littéral `l` associe sa négation.

(Q_{11}) Compléter la définition de la fonction `(sontLiees xs ys)` qui détermine si deux clauses `xs` et `ys` sont liées.

```
-- exemples

> sontLiees [Non (Var "a"), Non (Var "b")] [Non (Var "d"), Var "b", Var "c"]   ==> True
> sontLiees [Non (Var "a"), Var "b", Var "c"] [Non (Var "d"), Var "a", Var "c"]   ==> True
> sontLiees [Var "a", Non (Var "b"), Var "d"] [Var "a", Non (Var "c"), Non (Var "d")]   ==> True
> sontLiees [Var "a", Var "b", Var "c"] [Var "a", Non (Var "c"), Non (Var "d")]   ==> True
> sontLiees [Non (Var "a"), Var "b", Var "c"] [Non (Var "d"), Non (Var "a"), Var "c"]  ==> False
> sontLiees [Var "a", Non (Var "b"), Var "d"] [Var "a", Non (Var "c"), Var "d"]   ==> False

sontLiees ::
sontLiees []
sontLiees (x:
```

Définiton 2.2 (résolvante de deux clauses)
On considère deux clauses c_1 et c_2 *qui sont supposées être liées.*

1. soit l un littéral positif apparaissant dans c_1 (i.e. $c_1 = l \vee c'_1$) et tel que sa négation $\neg l$ apparaisse dans c_2 (i.e. $c_2 = \neg l \vee c'_2$)
2. soit l un littéral positif apparaissant dans c_2 (i.e. $c_2 = l \vee c'_2$) et tel que sa négation $\neg l$ apparaisse dans c_1 (i.e. $c_1 = \neg l \vee c'_1$)

Dans les deux cas, la résolvante des clauses c_1 et c_2 est la clause $c'_1 \vee c'_2$.

Exemples :

La résolvante des deux clauses ($c_1 = a \vee \neg b \vee d$) et ($c_2 = a \vee \neg d \vee \neg b$) est la clause ($a \vee \neg b$) car le littéral d appartient à c_1 et sa négation $\neg d$ appartient à c_2.

La résolvante de ($c_1 = \neg a \vee \neg b$) et ($c_2 = \neg d \vee c \vee b$) est la clause ($\neg a \vee \neg d \vee c$), car le littéral $\neg b$ appartient à c_1 et sa négation b appartient à c_2.

Enfin, la résolvante des clauses ($c_1 = \neg a \vee \neg b$) et ($c_2 = \neg d \vee c \vee \neg a$) n'est pas définie car les clauses c_1 et c_2 ne sont pas liées (cf Définition 2.1).

Attention. La clause ($c \vee d$) n'est pas la résolvante de ($c_1 = a \vee b \vee c$) et ($c_2 = \neg a \vee \neg b \vee d$). Dans ce cas particulier, il existe 2 résolvantes possibles pour c_1 et c_2 :

- $b \vee c \vee \neg b \vee d$ en choisissant le littéral a
- $a \vee c \vee \neg a \vee d$ en choisissant le littéral b

On voit que l'on pourrait construire des exemples de clauses avec $3, 4, ..., n$ résolvantes possibles.

Mais, pour la suite du DM, le choix d'une résolvante parmi plusieurs n'aura aucune incidence. En conséquence, **toute fonction qui calcule une résolvante (peu importe laquelle) sera OK**.

(Q_{12}) Compléter la définition de la fonction `(resolvante xs ys)` qui calcule une résolvante des deux clauses `xs` et `ys` *qui sont supposées être liées*

```
-- exemples

> resolvante [Non (Var "a"), Non (Var "b")] [Var "b", Var "c"]
[Non (Var "a"), Var "c"]


> resolvante [ Var "a", Non (Var "b"), Var "d"] [Var "a", Non (Var "d"), Non (Var "b")]
[Var "a", Non (Var "b")]


> resolvante [ Var "a", Non (Var "b"), Var "d"] [Non (Var "a"), Non (Var "d"), Non (Var "b")]
[Non (Var "b"), Var "d", Non (Var "d"), Non (Var "b")]


-- deux resolvantes possibles - on retourne ici la premiere rencontree
-- une autre reponse possible aurait ete [Var "a", Var "c", Non (Var "a"), Var "d"]

> resolvante [ Var "a", Var "b", Var "c"] [Non (Var "a"), Non (Var "b"), Var "d"]
[Var "b", Var "c", Non (Var "b"), Var "d"]
```

```
resolvante ::

resolvante []
resolvante (x:
      |
      ...
      | otherwise
```

Pour Q_{11} et Q_{12}, on pourra utiliser toute fonction auxiliaire jugée nécessaire.

A VENIR les JEUX de DONNEES pour EFFECTUER les TESTS