Mise sous forme clausale d'une formule du calcul propositionnel

1. Définitions générales

— un **littéral** est une variable propositionnelle ou sa négation.
Exemples : a, b, $\neg p$, $\neg q$
Les deux premiers littéraux sont des littéraux positifs, les 2 derniers des littéraux négatifs.

— une **clause** est une **disjonction de littéraux**.
Exemples : $(a \vee b)$, $(\neg a \vee \neg b \vee c \vee \neg d \vee e)$

— une **formule** `f` est sous **forme clausale** (ou Forme Normale Conjonctive) [1] ssi `f` est une conjonction de clauses.
Exemple : $f = (a \vee b) \wedge (\neg a \vee \neg e \vee c) \wedge (\neg d \vee \neg a)$

2. Représentation en Haskell

On définit le type `Formule` ci-dessous où un littéral positif est représenté par le constructeur `Var` portant sur une chaîne de caractères (`String`).

On considère les connecteurs usuels qui sont notés comme suit : `~` (négation), `&` (conjonction), `v` (disjonction), `=>` (implication) et `<=>` (équivalence).

```
data Formule = Var String
     | Non Formule
     | Et Formule Formule
     | Ou Formule Formule
     | Imp Formule Formule
     | Equi Formule Formule
        deriving (Eq, Show)

-- Exemples
f1 = (Ou (Et (Var "c") (Var "d")) (Et (Var "a") (Var "b")))
f2 = (Imp (Var "d") (Ou (Var "c") (Non (Var "b"))))
f3 = (Equi  (Var "d") (Et (Var "c") (Var "d")))
f4 = (Imp (Non (Var "d")) (Ou (Var "c") (Var "b")))
f5 = (Non (Non (Var "a")))
f6 = (Imp (Non (Ou (Var "a") (Var "d"))) (Ou (Var "c") (Var "b")))
```

(Q_1) compléter la définition de la fonction `(visuFormule f)` qui permet de visualiser une formule `f`.

```
-- Exemples

> visuFormule f1   ==> "((c & d) v (a & b))"
> visuFormule f2   ==> "(d => (c v ~b))"
> visuFormule f3   ==> "(d <=> (c & d))"
> visuFormule f4   ==> "(~d => (c v b))"
> visuFormule f5   ==> "~~a"
> visuFormule f6   ==> "(~(a v d) => (c v b))"

visuFormule :: Formule -> String
visuFormule (Var p)   =
visuFormule (Non f)   = "~" ++
visuFormule (Et g d)  = "(" ++ (visuFormule g) ++ " & " ++
visuFormule (Ou g d)  =
visuFormule (Imp g d) =
visuFormule (Equi g d) =
```

1. https://fr.wikipedia.org/wiki/Forme_normale_conjonctive

3. Mise sous forme clausale d'une formule

La mise sous forme clausale d'une formule s'effectue en 3 étapes :

1. éliminer les opérateurs `Imp` et `Equi`
2. amener les `Non` à l'intérieur des formules (afin qu'ils ne portent que sur des littéraux positifs)
3. faire apparaître une conjonction de clauses (distributivité entre les opérateurs `Et` et `Ou`)

```
-- Exemples

> visuFormule (formeClausale f1)   ==> "((c v a) & ((c v b) & ((d v a) & (d v b))))"
> visuFormule (formeClausale f2)   ==> "(~d v (c v ~b))"
> visuFormule (formeClausale f3)   ==> "((~d v c) & ((~d v d) & ((~c v ~d) v d)))"
```

Dans la suite de cet énoncé, nous allons définir, une par une, les 3 fonctions qui correspondent à ces 3 étapes successives.

3.1 Etape # 1 : éliminer les opérateurs `Imp` et `Equi`

(Q_2) Soient `g` et `d` deux formules.

— pourquoi peut-on remplacer `(Imp g d)` par `(Ou (Non g) d)` ?

— en déduire comment remplacer `(Equi g d)` ?

(Q_3) Compléter la définition de la fonction `(elimine f)` qui fait disparaître les opérateurs `Imp` et `Equi` de la formule `f`.

```
-- Exemples

> visuFormule f2   ==> "(d => (c v ~b))"
> visuFormule (elimine f2)   ==> "(~d v (c v ~b))"
> visuFormule f3   ==> "(d <=> (c & d))"
> visuFormule (elimine f3)   ==> "((~d v (c & d)) & (~(c & d) v d))"
> visuFormule f4   ==> "(~d => (c v b))"
> visuFormule (elimine f4)   ==> "(~~d v (c v b))"


elimine :: Formule -> Formule
elimine (Var p) =
elimine (Non f) =
...
...
elimine (Imp g d) =
elimine (Equi g d) =
```

3.2 Etape # 2 : amener les négations devant les littéraux positifs

Pour réaliser cette étape, on utilise la loi de la double négation et les 2 lois de De Morgan.

(Q_4) soit `f` une formule. Par quoi peut-on remplacer `(Non (Non f))` ?

(Q_5) Rappeler les 2 lois de De Morgan [2].

On définit (ci-dessous) la fonction `(ameneNon f)` qui amène les négations devant les littéraux positifs de la formule `f`.

(Q_6) Indiquer ce que doit faire la fonction `(disNon f)` et compléter sa définition.

```
-- Exemples

f2b = (Imp (Non (Var "d")) (Ou (Var "c") (Var "b")))
> visuFormule f2b   ==> "(~d => (c v b))"
> visuFormule (elimine f2b)   ==> "(~~d v (c v b))"
> visuFormule (ameneNon (elimine f2b))   ==> "(d v (c v b))"

> visuFormule f4   ==> "(~d => (c v b))"
> visuFormule (elimine f4)   ==> "(~~d v (c v b))"
> visuFormule (ameneNon (elimine f4))   ==> "(d v (c v b))"

> visuFormule (ameneNon f5)    ==> "a"


> disNon (Ou (Var "a") (Var "b"))
   Et (Non (Var "a")) (Non (Var "b"))
> disNon (Non (Ou (Var "a") (Var "b")))
   Ou (Var "a") (Var "b")
> disNon (Ou (Var "a") (Et (Var "b") (Var "c")))
   Et (Non (Var "a")) (Ou (Non (Var "b")) (Non (Var "c")))
> disNon (Ou (Var "a") (Et (Var "b") (Non (Var "c"))))
   Et (Non (Var "a")) (Ou (Non (Var "b")) (Var "c"))


ameneNon, disNon :: Formule -> Formule

ameneNon (Var p)   = (Var p)
ameneNon (Non f)   = disNon f
ameneNon (Et g d)  = (Et (ameneNon g)   ...)
ameneNon (Ou g d)  = (Ou (ameneNon g)   ...)

disNon (Var p)   =
disNon (Non f)   =
disNon (Et g d)  =
disNon (Ou g d)  =
```

2. https://fr.wikipedia.org/wiki/Lois_de_De_Morgan

3.3 Etape # 3 : faire apparaître une conjonction de clauses

Cette étape est donnée sous forme d'une **question ouverte**. Par la suite, une idée de solution est présentée, mais vous pouvez proposer **votre propre solution** qui soit différente.

Idée. Une clause étant une disjonction de littéraux (positifs ou négatifs), il faut utiliser la distributivité [3] de manière à obtenir une formule qui soit une conjonction de clauses.

Exemple. Une formule de la forme `(Ou (Et x1 x2) y)` devra être transformée en `(Et (Ou x1 y) (Ou x2 y))`

```
> normalise (Ou (Et (Var "c") (Var "d")) (Et (Var "a") (Var "b")))
   ==> (Et (Ou (Var "c") (Var "a"))
           (Et (Ou (Var "c") (Var "b"))
               (Et (Ou (Var "d") (Var "a"))
                   (Ou (Var "d") (Var "b")))))

> visuFormule (Ou (Et (Var "c") (Var "d")) (Et (Var "a") (Var "b")))
"((c & d) v (a & b))"
> visuFormule (normalise (Ou (Et (Var "c") (Var "d")) (Et (Var "a") (Var "b"))))
"((c v a) & ((c v b) & ((d v a) & (d v b))))"
```

L'idée présentée ici consiste à récursivement mettre sous forme de conjonction de clauses les différentes sous-formules, puis

- à appliquer la distributivité à l'aide de la fonction `(developper f)` lorsqu'il s'agit d'un `Ou`
- à joindre les deux à l'aide de la fonction `(concEt f)` lorsqu'il s'agit d'un `Et`

```
normalise :: Formule -> Formule
normalise (Et g d) = concEt (normalise g) (normalise d)
normalise (Ou g d) = developper (normalise g) (normalise d)
normalise f        = f

concEt :: Formule -> Formule -> Formule
concEt (Et g d) f = (Et g (concEt d f))
concEt g f        = (Et g f)
```

(Q_7) Définir la fonction `(developper g d)` (qui pourra utiliser une ou plusieurs fonctions auxiliaires).

4. Conclusion

(Q_8) En déduire la définition la fonction `(formeClausale f)` qui applique successivement chacune des 3 étapes à la formule `f`.

```
formeClausale :: Formule -> Formule
formeClausale f =
```

3. https://fr.wikiversity.org/wiki/Logique_de_base/Algèbre_de_Boole