L2/LD — TP 5 — Représentation des clauses et des conjonctions de clauses par des listes

L'objectif de ce TP est d'implanter en OCaml des fonctions qui seront utiles pour implanter la résolution propositionnelle dans un prochain TP. En particulier, on y découvre comment représenter des formules en CNF par des listes de listes de littéraux.

Rappel : Une formule propositionnelle est en forme normale conjonctive (CNF) si elle est de la forme $C_1 \wedge \ldots \wedge C_n$ où chaque C_i est une clause. Une clause est une disjonction $L_1 \vee \ldots \vee l_k$ de littéraux. Un *littéral* est soit une proposition atomique, soit la négation d'une proposition atomique.

A partir de ce TP, on ne représente plus les formules en CNF par des termes de type `pf`, mais par des listes de listes de littéraux :

— Un *littéral* est soit une proposition atomique, soit la négation d'une proposition atomique. Pour représenter un littéral, on définit le type OCaml suivant :

```
type literal = Pos of string | Neg of string;;
```

— Une clause est une disjonction $L_1 \vee \cdots \vee L_k$ de littéraux. On représente cette disjonction en OCaml par une liste de littéraux :

```
type clause = Disj of literal list;;
```

La disjonction de littéraux vide ($k = 0$) peut aussi être notée *faux*, grâce à la déclaration suivante :

```
let faux = Disj([]);;
```

— Une formule propositionnelle est en forme normale conjonctive si elle est de la forme $C_1 \wedge \ldots \wedge C_n$ où chaque C_i est une clause. On représente cette conjonction en OCaml par une liste de clauses :

```
type cnf = Conj of clause list;;
```

La conjonction de clauses vide ($n = 0$) peut aussi être notée *vrai*, grâce à la déclaration suivante :

```
let vrai = Conj([]);;
```

On rappelle que la programmation sur le type `'a list` est récursive. En général, il suffit de considérer le cas où la liste est vide, puis le cas où la liste, non vide, est de la forme `c::r`, où `c` est de type `'a` et `r` est une liste de type `'a list`. Ici, `'a` est soit `clause`, soit `literal`.

1 Affichage des formules en CNF

Dans les questions suivantes, on définit une fonction d'affichage pour chaque type présenté plus haut. Ceci permet de découvrir la représentation par des listes des formules en CNF.

1. Déclarer une formule en CNF `f1` de type `cnf`, contenant au moins trois clauses. Chaque clause doit contenir au moins deux littéraux distincts. L'une des clauses doit contenir une négation.
2. Définir une fonction `printLit : literal -> unit` qui affiche un littéral.

3. Définir une fonction `printLitList : literal list -> unit` qui affiche une liste de littéraux, en utilisant la fonction précédente.
4. Définir une fonction `printClause : clause -> unit` qui affiche une clause, en utilisant la fonction précédente.
5. Définir une fonction `printClauseList` qui affiche une liste de clauses, en utilisant la fonction précédente.
6. Ecrire une fonction récursive `printCNF : cnf -> unit` qui affiche une formule de type `cnf` de manière infixée, sans afficher les parenthèses les plus extérieures.

2 Fonctions sur les clauses

1. Déclarer une fonction `elimDouble`, de type `literal list -> literal list` qui élimine le même littéral présent plusieurs fois dans une clause. Indication : Utiliser la fonction `mem` d'OCaml, documentée ici : `http://caml.inria.fr/pub/docs/manual-ocaml/libref/List.html`. Pour utiliser cette fonction on peut soit écrire `List.mem` soit importer le module `List` avec la commande `open List;;` et écrire `mem`.
2. Déclarer une fonction `removeLiteral`, de type `literal * literal list -> literal list`, telle que `removeLiteral(l,c)` retourne la clause `c` privée du littéral `l`, si cette clause contient ce littéral, et retourne la clause `c` inchangée sinon.
3. Tester ces deux fonctions avec plusieurs exemples.