Licence d'informatique **Année 2003-2004**
L6M1

TP de Caml n° 2

1 Fonctionnelles

1. Écrire la fonctionnelle *ilexiste* qui, étant donnés un prédicat p et une liste l, retourne *true* si l'un au moins des éléments de l satisfait la propriété p et *false* sinon.
2. Écrire la fonctionnelle *qqsoit* qui, étant donnés un prédicat p et une liste l, retourne *true* si tous les éléments de l satisfont la propriété p et *false* sinon.
3. Au moyen de ces fonctionnelles, écrire les fonctions *estMembre* (test d'appartenance d'un élément à une liste) et *estInclus* (test d'inclusion d'un ensemble dans un autre lorsque les deux ensembles sont stockés sous forme de liste).
4. Écrire la fonctionnelle *filtrer* qui, étant donnés un prédicat p et une liste l, retourne la liste des éléments de l qui satisfont la propriété p.
5. Au moyen de *filtrer*, écrire une fonction *diffEns* qui, étant donnés deux ensembles stockés sous forme de listes $l1$ et $l2$, retourne la liste des éléments de $l1$ n'apparaissant pas dans $l2$.

2 Résolveur Prolog en Caml

On désire mettre en œuvre un résolveur Prolog sans variables (pour la logique des propositions uniquement) et sans *cut*. On arrêtera la résolution dès le premier succès.

On représente une clause par la liste de ces littéraux et un programme par la liste de ces clauses. Un but est représenté par la liste de ses littéraux.

Exemple : le programme ci dessous

```
1  p:-t,m,s.
2  p:-q,r.
3  q.
4  r:-s,t.
5  s.
6  t.
```

est représenté par « `[[p;t;m;s];[p;q;r];[q];[r;s;t];[s];[t]]` ».

Le but `-q,r,t.` est représenté par « `[q;r;t]` ».

Un programme Prolog est une base de connaissances, définie sous forme d'un ensemble de clauses qui déclarent la vérité, conditionnée ou non, de littéraux. Une clause est composée d'un littéral de tête (p dans le cas de la clause 1) et d'une queue de clause (t, m, s dans la clause 1, alors que la queue de la clause 3 est vide). Une clause exprime à quelle(s) condition(s) le littéral de tête est vrai. Par exemple, la clause 1 indique que p est vrai si t, m et s sont tous les trois vrais, alors que la clause 3 indique que q est vrai sans condition. L'existence de plusieurs clauses pour un même littéral de tête signifie que ce littéral est vrai si l'une des conditions est vraie : les clauses 1 et 2 expriment que p est vrai si $t \wedge m \wedge s$ est vraie ou $q \wedge r$ est vraie. En revanche, tout ce qui n'est pas déclaré vrai est supposé faux.

Résoudre un but, pour un programme donné, c'est montrer que tous les littéraux du but sont vrais.

Pour le programme donné en exemple, les littéraux q, s et t sont vrais puisque déclarés vrais sans conditions dans le programme (clauses 3, 5 et 6) ; le littéral r est vrai parce que s et t sont vrais (clause 4) ; le littéral p est vrai puisque q et r sont vrais (clause 4) ; le littéral m n'est pas vrai. En conséquence, le but q, r, t est démontrable, tandis que le but p, m ne l'est pas.

1. Écrire une fonction *demontrable* qui, étant donnés un but b et un programme p retourne vrai si et seulement si p fournit une démonstration de b. L'algorithme sera basé sur le fait qu'un but b est démontrable si tous les littéraux de b se trouvent en tête d'une clause de p dont tous les éléments de la queue sont démontrables.

2. Au moyen de $filtrer$, écrire une fonctions $litterauxDemontres$ qui, étant donné un programme p, retourne la liste des littéraux apparaissant dans la tête d'une clause de longueur 1 de p.

3. Au moyen de la fonctionnelle map, écrire une fonction $reduitProg$ qui, étant donnés un programme p et une liste ldd de littéraux déjà démontrés, retourne le programme p avec tous les éléments de ldd éliminés des queues de clauses.

4. Une autre méthode pour déterminer si un programme p fournit une démonstration d'un but b consiste à générer la liste ld de tous les littéraux démontrables (par réductions successives du programme au moyen de $litterauxDemontres$ et $reduitProg$). Ensuite, il suffit de tester si b est inclus dans ld. Écrire une nouvelle version de *demontrable* basée sur cette méthode. Critiquer les deux versions.