T. P. Logique.
Polytech'Marseille. 3ième année
UNIVERSITÉ D'AIX-MARSEILLE
Année universitaire 2016/2017

Enseignants : Motasem Nawaf
& Odile Papini

Feuille de T. P. 3 : Les listes en Prolog

Utilisation de SWI-Prolog:

- SWI-Prolog est installé sur votre compte sous Linux. Pour une utilisation sur votre machine personnelle, URL : **http://www.swi-prolog.org** version 6.6.6.
- Documentation, URL : **http://www.swi-prolog.org/pldoc**
- Pour lancer SWI-Prolog:
  - se connecter sous Linux
  - ouvrir un terminal, pour démarrer l'interpréteur taper :

    `prolog ou swipl`
  - aide en ligne :

    `help.`
  - sortir de SWI-Prolog:

    `halt.  ou  CTRL D`
  - avec un éditeur créer un programme avec une extension .pl (par exemple : nomfic.pl)
  - pour charger le fichier dans l'interpréteur (ne pas oublier le point final) :

    `[nomfic].  ou ['nomfic.pl'].`
- saisir la liste de buts séparés par des virgules et terminée par un point.
- pour obtenir toutes les solutions taper :

  ;

Les listes en PROLOG :

- la liste a,b,c,d s'écrit en PROLOG entre crochets : $[a, b, c, d]$

- si L est une liste, $L = [X|Y]$ avec X tête de liste, Y queue de la liste. Si $L = [a, b, c, d]$ alors $X = a$ et $Y = [b, c, d]$

- si L est une liste, L = [X,Y|Z] avec X premier élément de liste, Y deuxième élément de liste, Z queue de la liste. Si L=[a,b,c,d] alors X = a, Y =b et Z =[c,d]

- La liste vide est notée [].

- Le traitement des listes se faire de façon récursive :
  - une règle (ou plusieurs) pour la condition d'arrêt de la récursivité
  - une règle (ou plusieurs) pour traiter la tête de liste et lancer l'appel récursif avec une sous-liste issue de la liste de départ

EXEMPLE :

Parcourir une liste :
Condition d'arrêt de la récursivité : liste vide
Traitement la tête de liste : afficher la tête
Appel récursif : avec la queue de la liste.

En SWI Prolog :

```
parcours([]).
parcours([X|Y]) :- write(X), parcours(Y).
```

On lance avec le but : parcours([1,2,3,4,5]).

Exercice 1 **:**

Ecrire un programme logique :

- parcours : qui permet d'afficher les éléments d'une liste L. Faire écrire l'arbre d'effacement pour le but : parcours([1,2,3]).

  ```
  parcours([]).
  parcours([X|Y]) :- write(X), parcours(Y).
  ```

- meme_longueur : qui permet de tester si deux listes ont le même nombre d'éléments.

  meme_longueur([],[]).
  meme_longueur([X|Y], [Z|T],) :- meme_longueur(Y,T).
  ou en utilisant les variables anonymes
  meme_longueur([_|Y], [_|T],) :- meme_longueur(Y,T).

- rang_pair : qui construit la liste des éléments de rang pair d'une liste contenant un nombre pair d'éléments.
  On construit la nouvelle liste en tête de règle, ce qui permet de conserver l'ordre.

  rang_pair([],[]).
  rang_pair([X,Y|L],[Y,U]) :- rang_pair(L,U).

  On lance avec rang_pair([1,2,3,4],L).

- renverser : qui construit la liste des éléments dans l'ordre inverse.
  On construit la nouvelle liste en queue de règle, ce qui permet d'inverser l'ordre.
  On utilise une liste auxiliaire, qui au début est vide, dans laquelle on construit au fur et à mesure et à la fin on transfère la liste auxiiaire dans la liste résultat.

  renverser([],L,L).
  renverser([X|Y],L,R) :- renverser(Y,[X|L],R].

  On lance avec renverser([1,2,3,4],[],L).

- tester les programmes avec SWI Prolog.

Exercice 2 **:** Une liste L qui contient un nombre pair d'éléments est donnée.

Ecrire un programme logique qui construit la liste m des éléments de rang pair de L.

1) dans l'ordre d'apparition dans L.

2) en inversant cet ordre.

3) reprendre cet exercice dans le cas où L contient un nombre impair d'éléments.

```
rang_pair1([],[]).
rang_pair1([X,Y|Z],[Y,U] ) :- rang_pair1(Z,U ).

rang_pair2([],L,L).
rang_pair2([X,Y|Z],U,L ) :- rang_pair2(Z,[Y|U],L ).

rang_pair3([X],[]).
rang_pair3([X,Y|Z],[Y,U] ) :- rang_pair3(L,U ).

rang_pair4([X],L,L).
rang_pair4([X,Y|Z],U,L ) :- rang_pair4(Z,[Y|U],L ).
```

Ecrire un programme logique qui construit la liste des éléments de rang impair de L.

ordre inverse
rang impair : nbre pair d'élements

```
rang_impair([],L,L).
rang_impair([X,Z|Y],U,L) :- rang_impair(Y,[X|U],L).
```

rang impair : nbre impair d'élements

```
rang_impair([X],L,[X|L]).
rang_impair([X,Y|Z],U,L ) :- rang_impair(Z,[X|U],L ).
```

ordre d'apparition liste initiale
rang impair : nbre pair d'élements

```
rang_impair2([],[]).
rang_impair2([X,Z|Y],[X|U]) :- rang_impair2(Y,U).
```

ordre d'apparition liste initiale
rang impair : nbre pair d'élements

```
rang_impair2([X],[X]).
rang_impair2([X,Z|Y],[X|U]) :- rang_impair2(Y,U).
```

Exercice 3 **:**

- Ecrire les règles qui définissent element_de(X,L) exprimant qu'un élément X appartient à la liste L.

  element_de(X,[X|L]).
  element_de(X,[Y|L]) :- X \= Y, element_de(X,L).

- Ecrire les règles qui définissent hors_de(X,L) exprimant qu'un élément X n'appartient pas à la liste L.

  hors_de(X,[]).
  hors_de(X,[Y|L]) :- X \= Y, hors_de(X,L).

Exercice 4 **:** Ecrire un programme logique longueur qui permet de calculer le nombre déléments d'une liste.
longueur([],0).
longueur([X|L],M) :- longueur(L,N), M is N+1.
Exercice 5 **:** Ecrire un programme logique facto qui calcule factorielle n, $n \geq 0$.

facto(0,1).
facto(N,X) :- N > 0, M is N-1, facto(M,Y), X is N*Y.
(facultatif) Exercice 6 **:**
Deux listes u et v ont le même nombre d'éléments. Ecrire un programme fusionner(U,V,L) qui prend successivement un élément de U et de V pour construire L. En déduire un calcul direct des listes des éléments de rang pair et de rang impair d'une liste L donnée.

(facultatif) Exercice 7 **:**
Ecrire les règles qui définissent differents(L) exprimant que la liste L ne contient pas de répétition.

(facultatif) Exercice 8 **:**
Ecrire les règles qui définissent debut(M,L) exprimant qu'une liste M débute une autre liste L. Que se passe-t-il si M est une variable lors du lancement ?

(facultatif) Exercice 9 **:**
Ecrire les règles qui définissent contenue(M,L) exprimant que tous les éléments de m sont dans L (dans un ordre quelconque).

(facultatif) Exercice 10 **:**
Ecrire les règles qui permettent de retirer un élément X d'une liste L.

(facultatif) Exercice 11 **:**

Dans cet exercice les listes représentent des ensembles. Il n'y a donc pas d'élement répété.

- Ecrire les règles qui définissent intersection(U,V,L) où L est l'intersection de U et de V.
- Ecrire les règles qui définissent reunion(U,V,L) où L est la reunion de U et de V.
- écrire les règles intersection et réunion en utilisant la coupure.
- écrire les règles intersection et réunion sans utiliser la coupure.