# Représentation de la Connaissance
# Répétition 2
3 octobre 2006

## Astuce du jour

Il est possible de redéfinir un prédicat prédéfini. Par exemple, pour reverse, le programme serait

```
redefine_system_predicate(reverse(_,_)).
reverse(Xs,Ys):-reverse(Xs,[],Ys).
reverse([X|Xs],Ys,Zs):-reverse(Xs,[X|Ys],Zs).
reverse([],Xs,Xs).
```

## PROLOG et les arbres de recherche

### Arbres classiques

**1.** `reverse`
Soit le prédicat prédéfini `reverse(As,Bs)`, vrai lorsque la liste `Bs` est la liste `As` renversée, et faux sinon. En le redéfinissant comme indiqué dans l'astuce du jour, que doit-on instancier au minimum ?

Réaliser l'arbre de recherche de `reverse([1,2,3],Ys)`.

**2.** `suffix`
Définir un prédicat `suffix(As,Bs)` vrai lorsque la liste `As` est un suffixe de la liste `Bs`, et faux sinon. On demande que, si `As` n'est pas complètement instancié mais que `Bs` l'est, les différentes instances de `As` telles que `suffix(As,Bs)` est vrai soient générées.

Construire l'arbre de recherche de
- `suffix([b],[a,b])` ;
- `suffix(X,[a,b])`.

### Arbres de recherche particuliers

**3.** `prefix & suffix`

Soit les prédicats prefix et suffix définis de la manière suivante :

```
suffix(Xs,Xs).
suffix(Xs,[Y|Ys]) :- suffix(Xs,Ys).
prefix([],Xs).
prefix([X|Xs],[X|Ys]) :- prefix(Xs,Ys).
```

Réaliser l'arbre de recherche de
- `prefix([a],Xs)` ;
- `suffix([a],Ys)`.

Que ce serait-il passé si on avait inversé les deux lignes de $suffix$ ?

# PROLOG et les listes

## Correction des exercices proposés

### 4. `remove_last`

Définir un prédicat `remove_last(Xs,Ys)`, où `Xs` est une liste non vide et `Ys` une liste, vrai si et seulement si `Ys` est `Xs` dont on a retiré le dernier élément. On demande que, dans le cas où `Ys` n'est pas complètement instancié mais que `Xs` l'est, l'instance de `Ys` telle que `remove_last(Xs,Ys)` est vrai soit générée.

### 5. `rep`

Définir un prédicat `rep(Xs,Ys)`, vrai si `Ys` est la liste `Xs` dont on a retiré les répétitions, en supposant que la liste `Xs` est triée. On demande que dans le cas où `Ys` n'est pas complètement instancié mais que `Xs` l'est, l'instance de `Ys` telle que `rep(Xs,Ys)` est vrai soit générée.

**Exemple(s) d'utilisation:**

```
rep([1,1,1,2,2,2,2,4,4,6,7],X).
X = [1,2,4,6,7];
No
```

### 6. `subset_bis`

Définir un prédicat `subset_bis(Ss,Ls)` vrai lorsque `Ss` est une liste d'éléments de `Ls`, dans le même ordre d'apparition. On demande que dans le cas où `Ss` n'est pas complètement instancié mais que `Ls` l'est, les différentes instances de `Ss` telles que `subset_bis(Ss,Ls)` est vrai soient générées.

**Exemple(s) d'utilisation:**

```
?- subset_bis([1,3,4],[0,1,2,3,4,5]).
Yes
```

### 7. `sublist`

Définir un prédicat `sublist(Ss,Ls)` vrai lorsque la liste `Ss` est une sous-liste de la liste `Ls`, et faux sinon. On demande que, dans le cas où `Ss` n'est pas complètement instancié mais que `Ls` l'est, les différentes instances de `Ss` telles que `sublist(Ss,Ls)` est vrai soient générées.

**Exemple(s) d'utilisation:**

```
sublist([2,3],[1,2,3,4]).
Yes
sublist([2,4],[1,2,3,4]).
No
```

### 8. `inc_sublist`

Définir un prédicat `inc_sublist(Ls,Incs)` vrai si et seulement si `Incs` est la plus longue sous-liste croissante de la liste de nombres `Ls`. Dans le cas où `Incs` n'est pas complètement instancié mais que `Ls` l'est, on demande d'écrire deux versions différentes :

- La première renverra une seule instance de `Incs` telle que `inc_sublist(Ls,Incs)` est vrai.
- La deuxième renverra une et une seule fois chacunes des instances de `Incs` telles que `inc_sublist(Ls,Incs)` est vrai.

**Exemple(s) d'utilisation:**

```
inc_sublist([1,2,1,4,5,6,4,3,7],X).
X = [1,4,5,6]; No
inc_sublist([1,2,3,1,3,6,1,4,8],X).
% 1ere version :     % 2eme version :
X = [1,2,3]; No      X = [1,2,3]; X = [1,3,6]; X = [1,4,8]; No
```

## Exercices proposés

### 9. `alternate`

Définir un prédicat `alternate(As,Bs)`, où `As` et `Bs` sont des listes, vrai si et seulement si `Bs` est la liste `As` dont on a retiré les éléments de positions impaires. On demande que, dans le cas où `Bs` n'est pas complètement instancié mais que `As` l'est, l'instance de `Bs` pour laquelle `alternate(As,Bs)` est vrai soit générée.

## 10. `common_head`

Définir un prédicat `common_head(As,Bs,Cs)`, où `As`, `Bs` et `Cs` sont des listes, vrai si et seulement si `Cs` est le plus long préfixe commun à `As` et `Bs`. On demande que, dans le cas où `Cs` n'est pas complètement instancié mais que `As` et `Bs` le sont, les différentes instances de `Cs` telles que `common_head(As,Bs,Cs)` est vrai soient générées.

**Exemple(s) d'utilisation:**

```
common_head([1,2,3,a,b,c],[1,2,3,4,5,6],[1,2,3]).
Yes
```

## 11. `replace_atom`

Définir un prédicat `replace_atom(As,Bs,C,D)`, où `As` et `Bs` sont des listes et `C` et `D` des atomes, vrai si et seulement si `Bs` est `As` dont on a remplacé récursivement toute occurrence de `C` par `D`. On demande que dans le cas où une des deux listes n'est pas instanciée, mais que l'autre et les deux atomes le sont, l'instance correcte de la liste non-instanciée soit générée.

## 12. `replace_atom_bis`

Définir un prédicat `replace_atom_bis(As,Bs,C,D)`, où `As` et `Bs` sont des listes et `C` et `D` des atomes, vrai si et seulement si `Bs` est `As` dont on a récursivement remplacé toute occurrence de `C` par `D` et toute occurence de `D` par `C`. On demande que dans le cas où une des deux listes n'est pas instanciée, mais que l'autre et les deux atomes le sont, l'instance correcte de la liste non-instanciée soit générée.

## 13. `replace_atom_prof`

Définir un prédicat `replace_atom_prof(As,Bs,Cs,Ds)`, où `As` et `Bs` sont des listes et `C` et `D` des atomes, vrai si et seulement si `Bs` est `As` dont on a récursivement remplacé toute occurrence de `C` par `D` en profondeur. On demande que dans le cas où une des deux listes n'est pas instanciée, mais que l'autre et les deux atomes le sont, l'instance correcte de la liste non-instanciée soit générée. Remarque : atomic(X) est vrai si et seulement si X est une liste vide ou une constante atomique.

## 14. `flatten`

Ecrire un prédicat `flatten(Ass,Bs)` vrai si et seulement si la liste `Bs` est la liste des symboles apparaissant dans la liste de liste de...`Ass` dans le même ordre. De manière intuitive : `Bs` est la liste `Ass` dont on a enlevé tous les crochets intérieurs. On demande que dans le cas où `Bs` n'est pas complètement instancié mais que `Ass` l'est, l'instance de `Bs` telle que `flatten(Ass,Bs)` est vrai soit retournée.

**Exemple(s) d'utilisation:**

```
flatten([[1,2,3,[4,5],6,[7,8,[9]]],10,11,[12]],X).
X=[1,2,3,4,5,6,7,8,9,10,11,12];
No
```

## 15. `threeparts`

Ecrire un prédicat `threeparts(Zs,As,Bs,Cs)` vrai si et seulement si la liste `Zs` est la concaténation des listes non-vides `As`, `Bs` et `Cs`. On demande que dans le cas où `As`, `Bs` et `Cs` ne sont pas complètement instanciés mais que `Zs` l'est, les différentes instances de `As`, `Bs`, `Cs` telles que `threeparts(Zs,As,Bs,Cs)` est vrai soient retournées. On demande aussi que dans le cas où `As`, `Bs` et `Cs` sont complètements instanciés mais que `Zs` ne l'est pas, l'instance de `Zs` rendant vrai le prédicat soit générée.