

☞ **Rappels cours précédent**

(VIDEO-PPT 4.1)

- Prolog a comme caractéristique innovante la réversibilité rendant possible plusieurs interrogations sur un même prédicat.
- Il existe deux formes de récursivité :
 - la récursivité enveloppée s'appuyant sur l'hypothèse de récurrence
 - la récursivité terminale calculant un résultant intermédiaire dans un accumulateur comme dans une boucle
- (hors programme, seulement ici pour la « connaissance » de l'existence)
Lorsque les prédicats manipulent des listes, il existe une technique syntaxique permettant de passer d'une récursivité enveloppée (plus facile à mettre en place) à une récursivité terminale (plus efficace) : les d-listes.
- Le coupe-choix (anciennement coupure) permet d'éviter des résolutions inutiles mais doit être utilisé en dernier recours car il empêche la réversibilité.

8. Extensions au langage Prolog originel

8.1. Arithmétique

Gprolog définit des expressions arithmétiques faites de nombre et d'opérations : +, -, *, / (division réelle), // (division entière), rem (reste), ** (puissance).

Par ailleurs les prédicats suivants permettent de manipuler des expressions arithmétiques :

```
E1 == E2 (teste l'égalité de l'évaluation de E1 et E2)
E1 \= E2 (teste la différence de l'évaluation de E1 et E2)
E1 < E2 (teste < sur l'évaluation de E1 et E2)
E1 <= E2 (teste <= sur l'évaluation de E1 et E2)
E1 > E2 (teste > sur l'évaluation de E1 et E2)
E1 >= E2 (teste >= sur l'évaluation de E1 et E2)
```

Attention, si E1 ou E2 comportent des variables, elles doivent être instanciées.

Le prédicat is (X is E) est utilisé pour instancier une variable X la valeur d'une expression E.

Les prédicats = et \= permettent de tester si deux termes sont unifiables.

➤ Le prédicat factoriel

```
1. factoriel(0, 1).
2. factoriel(N, R) :- N >= 1, N1 is N-1, factoriel(N1, R1), R is
    N*R1.
```

8.2. Egalité et différence syntaxique

Il s'agit là non pas de tester les valeurs mais les variables atteintes avec == et \==. La comparaison directe de variable libre n'a guère d'intérêt mais devient intéressante quand les variables sont liées car c'est la liaison qui est testée.

➤ Tester si deux paramètres sont indépendants (ne sont pas liés à une même variable, qui peut être encore libre) :

```
predicat(X, Y) :- X\==Y, ...
```

8.3. Pseudo-prédicats d'entrée-sortie

Gprolog en tant que langage de programmation propose une panoplie de prédicats prédéfinis permettant de réaliser des interactions avec l'utilisateur, notamment les prédicats read, read-atom, read-integer, write, ln... (se référer au manuel utilisateur).

8.4. Pseudo-prédicats d'introspection

Gprolog propose de nombreux prédicats prédéfinis parmi lesquels certains permettent de vérifier la nature des termes manipulés :

```

var(X) (teste si x est libre)
nonvar(X) (teste x est liée)
integer(X) (teste x est liée à un entier)
is_list(X) (teste si x est une liste)
arg(N, S, SN) teste si SN est le nième argument de S

```

8.5. Prédicats méta-logiques

Les prédicats méta-logiques sont des prédicats pour mettre à jour la base de connaissances. Ils permettent d'ajouter ou supprimer des clauses en tête ou en queue dans un paquet de clauses.

En Gprolog, les paquets de clauses modifiables de la base de connaissances doivent être préalablement déclarés `dynamic` dans la base de connaissances.

```
:- dynamic(pere/2).
```

Les prédicats de mises à jour de la base de connaissances sont notamment : `asserta` (insertion en tête de paquet de clauses), `assertz` (insertion en queue de paquet de clauses) et `retract` (suppression).


➤ *Exemple* (déjà vu) dans « `familleKennedy3.pl` » la méta-règle :

```

complete_pere_par_mere :-      epoux(X,Y), mere(Y,Z),
                                non(pere(X,Z)),
                                assertz(pere(X,Z)).

```

Ces prédicats dynamiques sont intéressants pour la programmation de bases de connaissance pouvant agréger de nouveaux faits lors de l'exécution (systèmes experts).

 **Attention** : il faut manipuler ces prédicats avec précaution. En effet ils permettent à un programme de s'auto-modifier. Ainsi, deux exécutions successives suite à une même question peuvent éventuellement générer des résultats différents.

MiniTP : De l'uchronie....

En repartant de « *familleKennedy.pl* », après une première exécution de `grand_mere(rose,Y)` listant la descendance de `rose`, écrivez le prédicat `si_X_nEtaitPasNe(X)`, puis posez une seconde question :

```
si_X_nEtaitPasNe(john), grand_mere(rose,Y).
```