

Structures de données | Opérateur de coupure !

Introduction :

En prolog, la récursivité est une manière naturelle de visualiser les structures et les programmes.

1. Règles récursives.

Soit le prédicat *ancestre(X,Y)* défini par :

« X est ancêtre de Y si X est parent de Y ou si X est parent de ...parent ..de Y etc. »

Ce qui se formule en règles prolog par :

ancestre(X, Y) :- parent(X, Y).

ancestre(X, Y) :- parent(X, Z), parent(Z, Y).

ancestre(X, Y) :- parent(X, Z), parent(Z, V), parent(V, Y).

.....

Etc ..

Ce qui nécessite une infinité de règles pour compléter les solutions. Pour éviter cet inconvénient, nous allons redéfinir le prédicat *ancestre* de manière récursive comme suit :

ancestre(X, Y) :- parent(X, Y).

ancestre(X, Y) :- parent(X, Z), ancetre(Z, Y).

Dans la définition de la seconde clause, on utilise le prédicat *ancestre* en prémisse pour définir ce même prédicat en conclusion.

2. Les structures.

Soit à représenter l'énoncé d'une date quelconque :

« 6 janvier 2009 » par l'atome *date1*

Cette représentation ne fait pas apparaître le jour, mois et année ; on doit donc écrire des clauses pour cela :

date(date1).

jour(date1, 6).

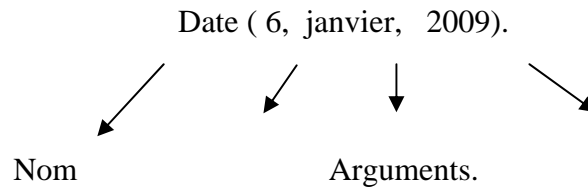
mois(date1, janvier).

annee(date1, 2009).

Inconvénient :

Pour chaque date, on doit utiliser un nom d'objet différent et donc une infinité de clauses. Dans cette représentation, on ne dispose pas d'entité unique qui contient toutes les informations de l'objet, mais celles-ci sont contenues dans des faits !

Prolog permet une telle représentation complexe par les structures, entité qui regroupe toutes les informations de l'objet (date par ex.) décomposé en jour, mois année.



L'accès aux composants de la structure se fait par les faits suivants :

Jour(date(X, Y, Z), X).

Mois(date(X, Y, Z), Y).

Annee(date(X, Y, Z),).

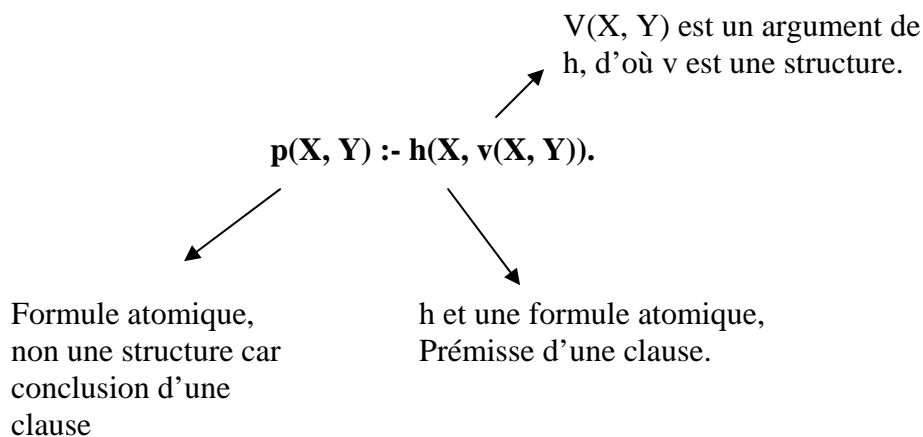
2.1. Syntaxe.

La syntaxe des structures ressemble à celle des formules atomiques, mais les concepts sont très différents :

- ✓ une formule atomique est une relation entre objets possédant une valeur de vérité contrairement à
- ✓ une structure de données qui est un objet (tuple) ; la différence entre les deux est déterminée par le contexte.

2.2. Exemple.

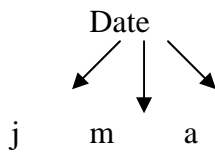
Soit la clause :



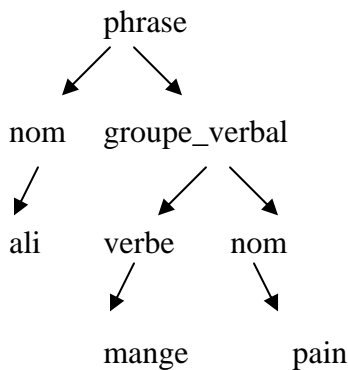
NB. Une formule atomique ne peut pas être argument d'une autre formule atomique !

2.3. Représentation par arbre.

Il est plus facile de décrire la forme d'une structure à l'aide d'un arbre.



La structure :
phrase(nom, groupe_verbal(verbe, nom)).



La structure :
phrase(nom(ali), groupe_verbal(verbe(mange), nom(pain))).

2.4. Exemple.

L'ensemble des entiers N, admet une construction récursive comme suit :
 « un entier est 0 ou le successeur d'un entier »

Soit s un nom de structure à 1 argument représentant la fonction successeur, alors 0, s(0), s(s(0)), s(s(s(0))), ..., représente la suite des entiers, qu'on peut définir par les clauses suivantes :

entier(X) :- X=0.
 entier(X) :- X=s(Y), entier(Y).

entier(0) .
 entier(s(Y)) :- entier(Y).

?- entier(s(s(0))).
 yes

?-entier(X).

X=0
 X=s(0)
 X=s(s(0))

Définir une relation (X+Y=Z) par le prédicat plus(X,Y,Z).

plus(0, Y, Y).

plus(s(X), Y, s(Z)) :- plus(X, Y, Z). C'est une définition récursive de l'addition +.

3. Les listes.

La liste est une structure de données classiques. C'est une suite ordonnée, de longueur quelconque, dont les éléments sont des termes (constante, variable, structures et listes)

3.1. Définition et notations.

La liste est une structure binaire récursive construite avec les symboles • et [] .

Une liste peut être représentée par un arbre particulier et peut être vide ou une structure à 2 composants qui sont :

- ✓ la tête
- ✓ la queue.

Liste vide []

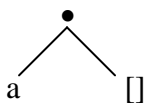
La tête et queue sont les composants du symbole fonctionnel ou opérateur •

Plusieurs notations sont adoptées pour représenter les listes, de la notation de base à la notation plus évoluée.

Exemples :

1/ Liste à 1 élément a notée •(a, []) ou a • [] ou [a]

Son arbre



2/ liste composée de a, b, c (atomes) notée

•(a, •(b, •(c, [])))

ou

a •(b •(c • [])))

ou bien

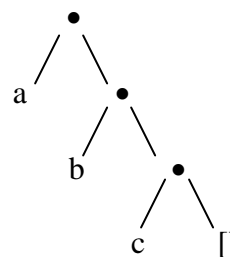
a • b • c • []

ou bien

[a | [b | [c | []]]]

ou bien

[a, b, c]



•(X, L) sera noté [X| L] où X est la tête de liste et L la queue de liste.

Remarque : contrairement à Pascal, où les éléments d'une liste sont du même type ; la liste en Prolog n'est pas homogène , mais hétérogène , les éléments de cette liste pouvant être des termes quelconques .

Soient les faits définissant des listes, suivants :

$p([1, 2, 3])$.

$p([a, b, c, [e, f, g]])$.

Et les requêtes suivantes :

?- $p([X, Y])$.

$X=1$ $Y=[2, 3]$
 $X=a$ $Y=[b, c, [e, f, g]]$

?- $p([_, _, _, [_|X]])$ le $_$ désigne une variable anonyme

$X=[f, g]$

3.5. Cas d'unifications de listes.

Liste1	Liste2	Solutions
$[X, Y, Z]$	$[ali, mange, pain]$	$X = ali$ $Y = mange$ $Z = pain$
$[ali]$	$[X Y]$	$X = ali$ $Y = []$
$[X, Y Z]$	$[ali, mange, pain]$	$X = ali$ $Y = mange$ $Z = [pain]$
$[[le, Y] Z]$	$[[X, livre], [est, pris]]$	$X = le$ $Y = livre$ $Z = [[est, pris]]$
$[X, Y Z, W]$	Incorrecte syntaxe.	

A suivre.....