LA RÉCURSIVITÉ ET LES LISTES

La récursivité

Considérons l'exemple vu en TD sur la famille, on souhaite rajouter les règles définissant:parent(X,Y), grand_pere(X,Y) et ancêtre(X,Y).

parent(X,Y) :-père(X,Y).

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

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

La récursivité

Le prédicat ancêtre ne peut être définit que si on fait appel à la récursivité. La condition de la terminaison de la récursivité est lorsqu'il s'agit d'un parent direct.

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

Condition d'arrêt **en premier**

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

Information discriminante **en premier**

Exemple: La fonction puissance

On s’appuie sur la définition récurrente :

a. $X0 = 1$

b. $Xn = Xn-1 * X$, avec $n>0$

d’où le programme Prolog suivant :

```
puissance(_,0,1).

puissance(X,N,P):-N>0, N1 is N-1,
        puissance(X,N1,P1) ,P   is  P1*X.
```

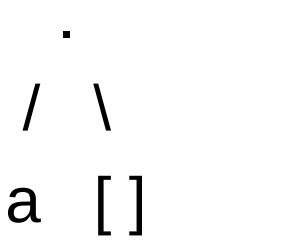
Les listes

Les listes sont des séquences ordonnées d'éléments qui peuvent avoir n'importe quelle taille. Les listes peuvent être représentées par un type d'arbre spécial. Une liste est soit vide, soit c'est une structure ayant deux éléments : la tête et la queue. La fin d'une liste est représentée par une queue comprenant seulement la liste vide [].

La tête et la queue sont les arguments d'une fonction nommée **"."**

Ainsi, la liste contenant seulement 'a' est écrite comme ceci : **.(**a,[]**)**

et son arbre s'écrit :



Les listes

Une liste contenant les éléments a, b et c peut s'écrire : .(a,.(b,.(c,[])))

Mais on écrira souvent les listes comme ceci :

Liste	tête	queue
[a,b,c]	a	[b,c]
[a]	a	[ ]
[]	(fails)	(fails)
[[the,cat],sat]	[the,cat]	[sat]
[the,[cat,sat]]	the	[[cat,sat]]
[X+Y,x+y]	X+Y	[x+y]

Nous pouvons séparer une liste en deux parties (la tête et la queue) avec le symbole "|".

Nous aurons [X | Y] où X est la tête et Y la queue.

RECHERCHE D'UN ELEMENT : SANS COUPE-CHOIX

membre(X,[X|_]) .

membre(X,[_|L]) :- membre(X,L).

A la question : ?- membre(X,[c,h,a,t]).

L'interpréteur répond par :

X=c;

X=h;

X=a;

X=t;

RECHERCHE D'UN ELEMENT : AVEC COUPE-CHOIX

Si maintenant, on souhaite avoir seulement un élément de la liste, on introduit une coupure dans l'arbre et la définition devient :

```
membre(X,[X|_]) :- !.
membre(X,[_|L]) :- membre(X,L).
```

Maintenant à la même question, l'interpréteur répond :

X=c

RECHERCHE D'UN ELEMENT : AVEC POSITION

```
ieme(X,1,[X|_]).
ieme(X,P,[_|R]) :- P1 is P-1, ieme(X, P1, R).
```

RECHERCHE D'UN ELEMENT : AVEC POSITION

```
ieme(X,1,[X|_]).
ieme(X,P,[_|R]) :- P1 is P-1, ieme(X, P1, R).
```

Vérifier si un élément X est à la position P dans la liste L,

```
?- ieme(d,2,[a,b,c,d]).
No ;
```

Renvoyer la position P d'un élément X donné dans la liste L,

```
?- ieme(d,P,[a,b,c,d]).
P = 4 ;
```

Renvoyer l'élément X qui se trouve à la position P d'une liste L.

```
?- ieme(X,4,[a,b,c,d]).
X = d ;
```

LONGUEUR D'UNE LISTE

```
long([], 0).

long([_ | L], N1) :-        long(L, N),  N1 is N + 1.
```

SUPPRESSION D'UN ÉLÉMENT DONNE D'UNE LISTE

```
efface(_, [], []).

efface(X,[X|L], L).

efface(X,[Y|L1], [Y|L2]) :- X \== Y,  efface(X,L1,L2).
```

SUPPRESSION D'UN ÉLÉMENT D'UNE LISTE SELON SA VALEUR OU/ET SA POSITION

```
supprimer(X,P,[X|L],L).

supprimer(X,P,[Y|L1], [Y|L2]) :- P > 1, P1 is P - 1,
                                 supprimer(X,P1,L1,L2).
```

INSERTION D'UN ÉLÉMENT DANS UNE LISTE

```
inserer(X,P,L,R) :- supprimer(X,P,R,L).
```