Université de Monastir

Cours: Programmation déclarative

Chapitre 3: Récursivité et SD Liste en Prolog

Réalisé par:
Dr. Taoufik Sakka Rouis
https://github.com/srtaoufik/Cours-Prog-Declarative/

1

Les types de base en Prolog

En Prolog, les types de base sont les éléments primitifs utilisés pour exprimer des faits et des règles.

Les types de base les plus courants sont :
- Entiers (int)
- Flottants (float)
- Atomes (atomes / strings)

Les atomes sont des valeurs constantes utilisées pour représenter des noms ou des étiquettes.
Les chaînes de caractères (strings) sont aussi considérées comme des atomes.

2

Les types de base en Prolog

Les types en Prolog sont dynamiquement typés, ce qui signifie que Prolog gère automatiquement le type des valeurs.

Requêtes :

```
?- X is 5 + 3.                          % rep.  X = 8
?- X is 3.5 + 1.5.                      % rep.  X = 5.0
?- X = 'Prolog'.                        % rep.  X = 'Prolog'
?- atom_concat('Hello', 'World', X).    % rep. X = 'HelloWorld'
```


Introduction à la Récursivité en Prolog

Exercice: Écrire une règle récursive permettant de calculer la factorielle d'un nombre N (notée N!).

Solution :

```
facto(0, 1).       % Cas de base
facto(N, F) :- N > 0, N1 is N - 1, facto(N1, F1), F is N * F1.
```

Requête d'exemple :

```
?- facto(3, X).          % rep.  X = 6.
?- facto(X, 24).         % rep.  X = 4.
```

Les Listes en Prolog

• Une liste est définie entre crochets, ex. [a, b, c].
• Elle est constituée d'une tête (**le** premier élément) et d'une queue (reste de la liste).

Notation :
[T|Q] : T représente la tête (un seul élément) et Q la queue.
Exemple : Dans [1, 2, 3], la tête est 1 et la queue est [2, 3].

Exemples de requête standard

```
?- member(2, [1, 2, 3]).      % rep. true
?- length([1, 2, 3, 4], X).   % rep.  X = 4
?- select(b, [a, b, c], L).   % rep. L = [a, c].
?- append([a, b], [c, d], L). L = [a, b, c, d].
```

5

Les listes en Prolog

Exercice: Écrire une règle récursive permettant de calculer la somme (pas le nombre) des éléments d'une liste :

Solution:
• Cas de base : La somme d'une liste vide est 0.
• Cas récursif : Ajouter le premier élément à la somme des éléments restants.
➔Code en Prolog :

```
somme([], 0).
somme([T|Q], S) :- somme(Q, S1), S is T + S1.
```

Requête d'exemple :

```
?- somme([2, 4, 6], X).       % rep. X = 12.
```

6

Les Listes en Prolog

Exercice: Écrire une règle récursive permettant de compter le nombre d'éléments d'une liste.

Solution :
• Cas de base : La longueur d'une liste vide est 0.
• Cas récursif : Compter le reste de la liste et ajouter 1.
➔Code en Prolog :

```
longueur([], 0).
longueur([_|Q], N) :- longueur(Q, N1), N is N1 + 1.
```

Requête d'exemple :

```
?- longueur([a, b, c], N).          % rep.   N = 3.
```


Les Listes en Prolog

Exercice: Écrire une règle récursive permettant d'inverser une liste.

Solution :
• Cas de base : L'inversion d'une liste vide est une liste vide.
• Cas récursif : Inverser le reste de la liste, puis ajouter la tête à la fin.

➔Code en Prolog :

```
inverse([], []).
inverse([T|Q], L) :- inverse(Q, L1), append(L1, [T], L).
```

Requête d'exemple :

```
?- inverse([1, 2, 3], L).          % rep.   L = [3, 2, 1].
```

Les Listes en Prolog

• **Exercice:** En utilisant la règle permettant d'inverser une liste; on vous demande d'écrire une règle permettant de vérifier si une liste est un palindrome ou non

➔**Code en Prolog :**

```
inverse([], []).
inverse([T|Q], L) :- inverse(Q, L1), append(L1, [T], L).
is_palindrome (L) :- inverse(L, L)
```

Requetes:

```
?- is_palindrome([r, a, d, a, r]).   % rep. true.
?- is_palindrome([a, b, c]).         % rep. false.
```


Les Listes en Prolog

• **Exercice:** proposez une règle permettant de calculer le nieme terme de la suite du Fibonacci.

➔**Code en Prolog :**

```
fibo(0,0) .
fibo(1,1) .
fibo(N,F) :- N>1 , N1 is N-1, fibo(N1,F1) , N2 is N-2, fibo(N2,F2) ,
F is F1+F2 .
```

Requetes:

```
?- fibo(5,Y).      % rep. Y=8
```

Les Listes en Prolog

Exercice: Ecrire un prédicat élément, de deux manières, avec et sans le cut (!), permettant de savoir si X est un élément de la liste L. Expliquer la différence et tracer l'arbre de résolution correspondant.
NB: Le cut (!) permet de couper l'exploration une fois qu'une solution est trouvée. Cela évite de rechercher des solutions supplémentaires.
➔Code Prolog

```
% Cas de base : X est la tête de la liste.
element(X, [X|_]).
 % Cas récursif : X est un élément de la queue.
element(X, [_|Q]) :- element(X, Q).

%requetes
?- element(2, [2, 2, 2]).
% rep.  true ; true ; true ; false.
```

```
% Cas de base : X est la tête de la liste.
 element(X, [X|_]) :- !.
% Cas récursif : X est un élément de la queue.
element(X, [_|Q]) :- element(X, Q).

%requetes
?- element(2, [2, 2, 2]).
% rep.   true.
```


Les Listes en Prolog

Exercice: Ecrire un prédicat compresse permettant de supprimer des doublons consécutifs dans une liste L pour obtenir une liste L1. L'ordre des éléments doit être respecté.
Exemple:
?- compresse([a,a,a,a,b,c,c,a,a,d,e,e,e,e],L1).
L1 = [a,b,c,a,d,e]
Solution:
- Si la liste est vide, la liste compressée est également vide.
- Si le premier élément est le même que le second, on l'ignore et continue avec la queue de la liste.
- Sinon, on garde le premier élément et applique la récursion à la queue de la liste.

Les Listes en Prolog

Exercice: Ecrire un prédicat compresse permettant de supprimer des doublons consécutifs dans une liste L pour obtenir une liste L1. L'ordre des éléments doit être respecté.

➔ **Code en Prolog**

```
compresse([], []).

compresse([X, X | Q], L1) :-    compresse([X | Q], L1).

compresse([X | Q], [X | L1]) :-    Q \= [],    Q = [Y | _],
  X \= Y,     compresse(Q, L1).
% requete
?- compresse([a, a, b, b, c, c, a], L1). % rep.  L1 = [a, b, c, a]
```

13