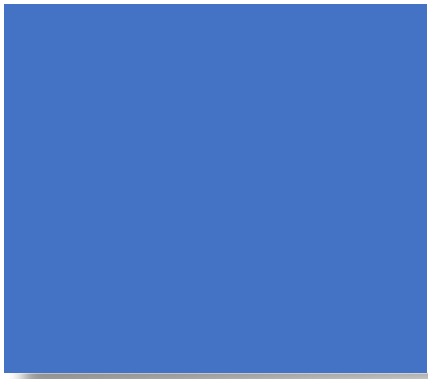
Programmation Logique

Listes



Enseignant: NGUYEN Thi Minh Tuyen

Plan

1. Définition d'une liste
2. Définition récursive avec la liste
3. Prédicats prédédinis
4. Exercices

Plan

1. Définition d'une liste
2. Définition récursive avec la liste
3. Prédicats prédédinis
4. Exercices

Introduction

- Prolog utilise les termes composés pour représenter des données.
- Limitation: chaque prédicat doit avoir un nombre fixe d'arguments.

→ Prolog fournit un type d'objet de données flexible: liste

- Un séquence des valeurs
- Par exemple: `[chien, chat, poisson, homme]`

Définition d'une liste

- Les éléments sont mis entre `[]`, séparés par des virgules.
- La longeur d'une liste = le nombre des éléments.
- La liste vide (de longueur 0): `[]`.
- Une liste contient des termes :
  - constantes, variables, termes complexes qui peuvent être mélangés.
  - Une liste peut elle-même contenir d'autres listes
- Exemple:
  - `[mia, vincent, jules, yolanda]`
  - `[mia, robber(honey_bunny), X, 2, mia]`
  - `[mia, [vincent, jules], [butch, girlfriend(butch)]]`

Tête et queue

- Une liste vide
  - n'a pas ni tête ni queue.
- Une liste non vide
  - La tête: Le premier élément de la liste
  - La queue: La liste privée de la tête.
- Exemple
  - `[mia, vincent, jules, yolanda]`: la tête est `mia` ; la queue est `[vincent, jules, yolanda]`.
  - `[dead(zed)]`: la tête est `dead(zed)` ; la queue est `[]`.

Décomposition de liste

- Le prédicat "|" décompose une liste entre sa tête et sa queue.
- Exemple:

```
?- [X|Y]=[mia,vincent,jules,yolanda].
X = mia,
Y = [vincent, jules, yolanda].
?- [X|Y]=[].
false.
?-  [X,Y|Tail] = [[], dead(z), [2, [b,c]], [], Z,
[2, [b,c]]] .
X = [],
Y = dead(z),
Tail = [[2, [b, c]], [], Z, [2, [b, c]]].
```

Exercice 1:

- Learn Prolog Now, exercice 4.1, 4.2

Plan

1. Définition d'une liste
2. Définition récursive avec la liste
3. Prédicats prédédinis
4. Exercices

Exercice 2: Définition récursive

1. `membre/2`
2. `longeur/2`
3. `concat/3`
4. `inverse/2`

Execice 2: Définition récursive [1]


```
?-membre(a,[a,b,c]).
true.
?- membre(mypred(a,b,c),[q,r,s,mypred(a,b,c),w]).
true.
?- membre(x,[]).
false.
?- membre([1,2,3],[a,b,[1,2,3],c]).
true.
?- membre(X,[a,b,c]).
X=a;
X=b;
X=c;
false.
```

Execice 2: Définition récursive [2]


```
?- longeur([a,b,c,d],X).
X=4
?- longeur([[a,b,c],[d,e,f],[g,h,i]],L).
L=3
?- longeur([],L).
L=0
?- longeur([a,b,c],3).
true.
?- longeur([a,b,c],4).
false.
?- N is 3, longeur([a,b,c],N).
N=3
```

Execice 2: Définition récursive [3]


```
concat(L1, L2, LR)
?- concat([a,b,c,d],[3,4,5],[a,b,c,d,3,4,5]).
true
?- concat([a,b,c],[3,4,5],[a,b,c,d,3,4,5]).
false
?- concat([a,b,c,d],[1,2,3,4,5], X).
X=[a,b,c,d,1,2,3,4,5]
```

Concaténation de deux listes


```
concat([ ], L, L).
concat([T|Q], L, [T|QR]):- concat(Q, L, QR).
```

Efficacité de `concat/3`

- utilisable
- mais pas efficace

Question

- Utilise `concat/3` pour concaténer deux listes.
- Exemple:
  - Liste 1: `[a, b, c, d, e, f, g, h]`, Liste 2: `[i, j , k]`
  - Comment concaténer `Liste 1` et `Liste 2` dans lesquelles l'ordre des éléments n'est pas important.
  - Laquelle est le plus efficient?

`?- concat([a,b,c,d,e,f,g,h], [i,j,k], R)` et

`?- concat([i,j,k], [a,b,c,d,e,f,g,h], R) ?`

Réponse

- Selon la définition récursive de `concat/3`:

```
concat([ ], L, L).
concat([T|Q], L, [T|QR]):- concat(Q, L, QR).
```

- Ce que nous observons:
  - Il traverse le premier paramètre, ne touche pas le deuxième.
  - C'est-à-dire: Il est préférable de l'appeler avec la liste la plus courte comme première paramètre.

Execice 2: Définition récursive [3] Inverse naïf

1. Si nous inversons une liste vide, nous obtenons une liste vide.
2. Si nous inversons une liste `[T|Q]`, nous obtenons une liste en inversant `Q` et en la concaténant avec `[T]`.
3. C'est-à-dire, considérons la liste `[a,b,c,d]`
    1. Si nous inversons la queue de la liste, nous obtenons `[d, c, b]`
    2. Nous concaténons la queue inversée avec `[a]` ➔ `[d,c,b,a]`

En Prolog ...


```
inverseNaif([],[]).
inverseNaif([T|Q],R):- inverseNaif(Q,QR), concat(QR,[T],R).
```

- En résumé:
  - La définition est correcte, mais pas efficace...
  - Il prend pas mal de temps pour exécuter le prédicat `concat`.

Inverser une liste avec l'accumulateur


```
accInverse([ ],L,L).
accInverse([T|Q],Acc,Inv):-accInverse(Q,[T|Acc],Inv).
inverse(L1,L2):- accInverse(L1,[ ],L2).
```

Plan

1. Définition d'une liste
2. Définition récursive avec la liste
3. Prédicats prédédinis
4. Exercices

Appartenance d'un élément à une liste

- Définition d'un prédicat `member(X,Y)`: vrai si X appartient à la liste Y.

```
member(X,[X|T]).
member(X,[H|T]) :- member(X,T).
```

- C'est un prédicat prédéfini: `member/2`
  - premier argument: un terme qui n'est pas une variable
  - deuxième argument: une list
  - réussit si le premier argument est un membre de la liste désigné par le second argument
  - Si premier argument est une variable → Prolog utilise backtracking pour trouver tous les éléments de la liste.

Exemple


```
?-member(a,[a,b,c]).
true.
?-member(mypred(a,b,c),[q,r,s,mypred(a,b,c),w]).
true.
?-member(x,[]).
false.
?- member([1,2,3],[a,b,[1,2,3],c]).
true.
?-member(X,[a,b,c]).
X=a;
X=b;
X=c;
false.
```

Prédicat `length/2`

- C'est un prédicat prédéfini

```
?- length([a,b,c,d],X).
X=4
?- length([[a,b,c],[d,e,f],[g,h,i]],L).
L=3
?- length([],L).
L=0
?- length([a,b,c],3).
true.
?- length([a,b,c],4).
false.
?- N is 3,length([a,b,c],N).
N=3
```

Prédicat `reverse/2`

- Permet d'inverser les éléments d'une liste
- Exemple:

```
?- reverse([1,2,3,4],L).
L = [4,3,2,1]
?- reverse(L,[1,2,3,4]).
L = [4,3,2,1]
?-reverse([[dog,cat],[1,2],[bird,mouse],[3,4,5,6]],L).
L = [[3,4,5,6],[bird,mouse],[1,2],[dog,cat]]
```

Prédicat append/3

- Concaténer deux listes pour créer une nouvelle liste.
- Exemple:

```
?- append([1,2,3,4],[5,6,7,8,9],L).
L = [1,2,3,4,5,6,7,8,9]
?- append([],[1,2,3],L).
L = [1,2,3]
?- append([[a,b,c],d,e,f],[g,h,[i,j,k]],L).
L = [[a,b,c],d,e,f,g,h,[i,j,k]]
```

Exemple [1]


```
?- append(L1,L2,[1,2,3,4,5]).
L1 = [] ,L2 = [1,2,3,4,5] ;
L1 = [1] , L2 = [2,3,4,5] ;
L1  = [1,2] , L2  = [3,4,5] ;
L1  = [1,2,3] , L2  = [4,5] ;
L1  = [1,2,3,4] , L2  = [5] ;
L1  = [1,2,3,4,5] , L2  = [] ;
false.
```

Exemple [2]


```
?- append(X,[Y|Z],[1,2,3,4,5,6]).
X = [] , Y=1,Z = [2,3,4,5,6] ;
X = [1] , Y=2, Z = [3,4,5,6] ;
X = [1,2] , Y=3, Z = [4,5,6] ;
X = [1,2,3] , Y=4, Z = [5,6] ;
X = [1,2,3,4] , Y=5, Z = [6] ;
X = [1,2,3,4,5] , Y=6,Z = [] ;
false.
```

Application du prédicat append/3

- Décomposer une liste en deux sous listes

```
?- append(X,Y, [a,b,c,d]).
X=[ ]                 Y=[a,b,c,d];
X=[a]                 Y=[b,c,d];
X=[a,b]           Y=[c,d];
X=[a,b,c]             Y=[d];
X=[a,b,c,d]     Y=[ ];
false
```

Exercices

1. Définir le prédicat **`prefixe(Prefixe ,Liste)`**:
   - vrai si `Prefixe` est un préfixe d'une liste `Liste`.
   - Exemple: `[a,b,c]` est un préfixe de la liste `[a,b,c,d,e]`.
2. Définir le prédicat **`suffixe(Suffixe,Liste)`**:
   - vrai si `Suffixe` est un suffixe d'une liste `Liste`.
   - Exemple: `[d,e]` est un suffixe de la liste `[a,b,c,d,e]`.
3. Définir le prédicat **`sousListe(SousL,Liste)`**:
   - vrai si `SousL` est un sous-liste d'une liste `Liste`.
   - Exemple: `[c,d]` et `[c,d,e]` sont des sous-listes de la liste `[a,b,c,d,e]`.

Préfixe


```
prefixe(P,L):- append(P,_,L).

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

Suffixe


```
suffixe(S,L):- append(_,S,L).

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

Sous liste


```
sousliste(SousL,Liste):-
                    suffixe(Suffixe,Liste),
                    prefixe(SousL,Suffixe).
```