Cours 4 : Listes

- Théorie
  - Les listes, une structure de données importante très souvent utilisée en programmation Prolog
  - Définition du prédicat member/2, un outil fondamental pour manipuler les listes en Prolog
  - Parcours de liste par récursivité

- Exercices
  - Exercices du chapitre 4 (LPN)
  - TP

Listes

- Une liste est une séquence finie d'éléments
- Exemples de listes en Prolog :

[mia, vincent, jules, yolanda]

[mia, robber(honeybunny), X, 2, mia]

[ ]

[mia, [vincent, jules], [butch, friend(butch)]]

[[ ], dead(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

Les listes

- Les éléments d'une liste sont entourés de crochets [A,B,…]
- La longueur d'une liste est le nombre d'éléments qu'elle contient
- Tous les types d'éléments de Prolog peuvent être des éléments d'une liste (termes simples, complexes, listes, …)
- Il existe une liste spéciale :
  la liste vide [ ]

Tête et Queue

- Une liste non vide peut être vue comme étant composée de deux parties :
  - La tête
  - La queue
- La tête est le premier élément
- La queue est le reste
  - La queue est la liste qui reste lorsque l'on enlève le premier élément
  - La queue d'une liste est une liste

Tête et Queue exemple 1

- [mia, vincent, jules, yolanda]

  Tête :
  Queue :

Tête et Queue exemple 1

- [mia, vincent, jules, yolanda]

  Tête : mia
  Queue :

Tête et Queue exemple 1

- [mia, vincent, jules, yolanda]

  Tête : mia
  Queue : [vincent, jules, yolanda]

Tête et Queue exemple 2

- [[ ], dead(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

  Tête :
  Queue :

Tête et Queue exemple 2

- [[ ], dead(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

  Tête : [ ]
  Queue :

Tête et Queue exemple 2

- [[ ], dead(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

  Tête : [ ]
  Queue : [dead(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

Tête et Queue exemple 3

- [dead(z)]

  Tête :
  Queue :

Tête et Queue exemple 3

- [dead(z)]

  Tête : dead(z)
  Queue :

Tête et Queue exemple 3

- [dead(z)]

  Tête : dead(z)
  Queue : [ ]

Tête et Queue d'une liste vide

- La liste vide n'a ni queue ni tête
- Pour Prolog, [ ] est une liste spéciale, simple, sans structure interne

- La liste vide joue un rôle important dans les prédicats récursif pour manipuler les listes en Prolog

L’opérateur prédéfini |

- Prolog dispose d’un opérateur prédifini pour décomposer les listes en sa tête et sa queue : |
- L’opérateur | est un outil indispensable pour écrire des prédicats qui manipulent des listes

L’opérateur prédéfini |

```
?- [Head|Tail] = [mia, vincent, jules, yolanda].

Head = mia
Tail = [vincent,jules,yolanda]
yes

?-
```

L’opérateur prédéfini |

```
?- [X|Y] = [mia, vincent, jules, yolanda].

X = mia
Y = [vincent,jules,yolanda]
yes

?-
```

L’opérateur prédéfini |

```
?- [X|Y] = [ ].

no

?-
```

L’opérateur prédéfini |

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

X = [ ]
Y = dead(z)
Z = _4543
Tail = [[2, [b,c]], [ ], Z, [2, [b,c]]]
yes

?-
```

Variables anonymes

- Supposons que nous nous intéressons au 2ème et au 4ème élements d'une liste and fourth element of a list

```
?- [X1,X2,X3,X4|Tail] = [mia, vincent, marsellus, jody, yolanda].
X1 = mia
X2 = vincent
X3 = marsellus
X4 = jody
Tail = [yolanda]
yes

?-
```

Variables anonymes

- Le moyen le plus simple d'obtenir l'information désirée :

```
?- [ _,X2, _,X4|_ ] = [mia, vincent, marsellus, jody, yolanda].
X2 = vincent
X4 = jody
yes

?-
```

- Le souligné est la variable anonyme

Variables anonymes

- Utilisée lorsqu'une variable est nécessaire mais que son instantiation ne nous intéresse pas
- Chaque occurrence de la variable anonyme est indépendante : elles peuvent être liées à des éléments différents

Exercices

- Exercice 4.1 of LPN
- Exercice 4.2 of LPN

Membre

- Comment savoir si quelque chose est un élément d'une liste ou pas
- Ecrivons un prédicat qui indique si un élément X appartient à une liste L
- Ce prédicat est généralement appelé member/2

member/2

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

```
?-
```

member/2

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

```
?- member(yolanda,[yolanda,trudy,vincent,jules]).
```

member/2

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

```
?- member(yolanda,[yolanda,trudy,vincent,jules]).
yes
?-
```

member/2

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

```
?- member(vincent,[yolanda,trudy,vincent,jules]).
```

member/2

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

```
?- member(vincent,[yolanda,trudy,vincent,jules]).
yes
?-
```

member/2

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

```
?- member(zed,[yolanda,trudy,vincent,jules]).
```

member/2

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

```
?- member(zed,[yolanda,trudy,vincent,jules]).
no
?-
```

member/2

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

```
?- member(X,[yolanda,trudy,vincent,jules]).
```

member/2

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

```
?- member(X,[yolanda,trudy,vincent,jules]).
X = yolanda;
X = trudy;
X = vincent;
X = jules;
no
```

Réécriture de member/2

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

Parcourir une liste

- Le prédicat member/2 predicate fonctionne par récursivité pour parcourir une liste
  - Traiter la tête, puis
  - Faire récursivement la même chose à la queue
- Technique très commune en Prolog et qui doit être maîtrisée

Exemple: a2b/2

- Le prédicate a2b/2 prend deux listes comme arguments et obtient un succès
  - si le premier argument est une liste de a, et
  - le deuxième argument une liste de b de la même longueur

```
?- a2b([a,a,a,a],[b,b,b,b]).
yes
?- a2b([a,a,a,a],[b,b,b]).
no
?- a2b([a,c,a,a],[b,b,b,t]).
no
```

Définir a2b/2: étape 1

```
a2b([],[]).
```

- Souvent la meilleur façon de résoudre un problème est de l'aborder par le cas le plus facile
- Ici : la liste vide

Définir a2b/2: étape 2

```
a2b([],[]).
a2b([a|L1],[b|L2]):- a2b(L1,L2).
```

- Puis pensons à la récursivité

Testons a2b/2

```
a2b([],[]).
a2b([a|L1],[b|L2]):- a2b(L1,L2).
```

```
?- a2b([a,a,a],[b,b,b]).
yes
?-
```

Testons a2b/2

```
a2b([],[]).
a2b([a|L1],[b|L2]):- a2b(L1,L2).
```

```
?- a2b([a,a,a,a],[b,b,b]).
no
?-
```

Testons a2b/2

```
a2b([],[]).
a2b([a|L1],[b|L2]):- a2b(L1,L2).
```

```
?- a2b([a,t,a,a],[b,b,b,c]).
no
?-
```

Plus loin avec a2b/2

```
a2b([],[]).
a2b([a|L1],[b|L2]):- a2b(L1,L2).
```

```
?- a2b([a,a,a,a,a], X).
X = [b,b,b,b,b]
yes
?-
```

Plus loin avec a2b/2

```
a2b([],[]).
a2b([a|L1],[b|L2]):- a2b(L1,L2).
```

```
?- a2b(X,[b,b,b,b,b,b,b]).
X = [a,a,a,a,a,a,a]
yes
?-
```

Résumé de la séance

- Listes et prédicats récursifs pour manipuler des listes
- Type de programmation essentiel en Prolog
- La plupart des prédicats Prolog sont des variantes de ces prédicats