PROLOG

Programmation Logique

Yannick Estève
Université du Maine

Cours 3: Récursivité

- Théorie
  - Définitions récursives en Prolog
  - Quatre exemples
  - Différences entre les aspects déclaratifs et procéduraux d'un programme Prolog
- Exercices
  - Exercices (LPN, chapitre 3)
  - Travaux pratiques

Définitions récursives

- Les prédicats Prolog peuvent être définis récursivement
- Un prédicat est défini récursivement si une ou plusieurs règles font référence à ce prédicat dans sa définition

Exemple 1: Eating

```
digere(X,Y):- vientDeManger(X,Y).
digere(X,Y):- vientDeManger (X,Z), digere(Z,Y).

vientDeManger(moustique,sang(john)).
vientDeManger(grenouille, moustique).
vientDeManger(cigogne,grenouille).
```

```
?-
```

Schéma de la situation

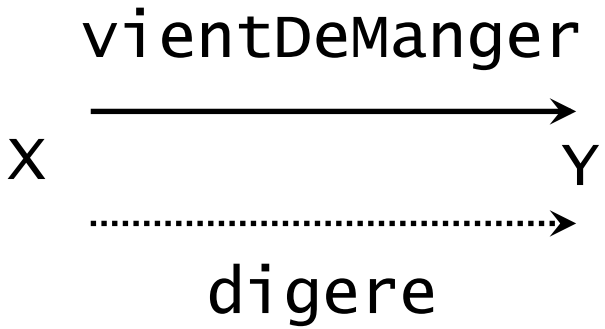
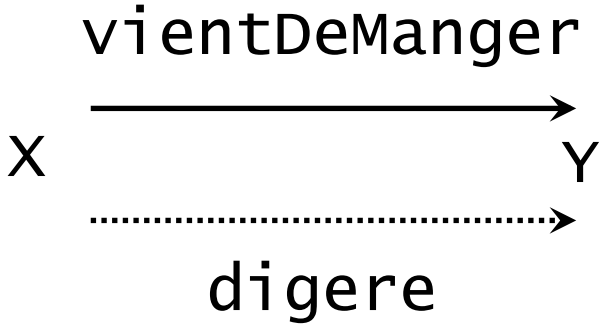
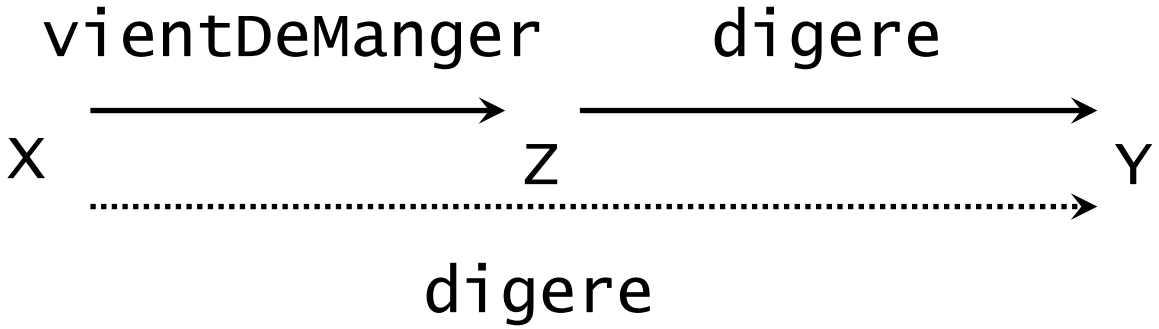


Schéma de la situation





Exemple 1: Manger

```
digere(X,Y):- vientDeManger(X,Y).
digere(X,Y):- vientDeManger (X,Z), digere(Z,Y).

vientDeManger(moustique,sang(john)).
vientDeManger(grenouille, moustique).
vientDeManger(cigogne,grenouille).
```

```
?- digere(cigogne,moustique).
```

Une autre définition récursive

```
p:- p.
```

```
?-
```

Une autre définition récursive

```
p:- p.
```

```
?- p.
```

Une autre définition récursive

```
p:- p.
```

```
?- p.
ERROR: out of memory
```

Exemple 2: Descendant

```
child(bridget,caroline).
child(caroline,donna).

descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z), child(Z,Y).
```

Exemple 2: Descendant

```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).


descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z), child(Z,Y).
```

Exemple 2: Descendant

```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).


descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z), child(Z,Y).
```

```
?- descend(anna,donna).
no
?-
```

Exemple 2: Descendant

```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).


descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z), child(Z,Y).
descend(X,Y):- child(X,Z), child(Z,U), child(U,Y).
```

```
?-
```

Exemple 2: Descendant

```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).


descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z), descend(Z,Y).
```

```
?-
```

Exemple 2: Descendant

```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).


descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z), descend(Z,Y).
```

```
?- descend(anna,donna).
```

Arbre de recherche

- Dessiner l'arbre de recherche pour :
  ?- descend(anna,donna).

```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).


descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z), descend(Z,Y).
```

Exemple 3 : Successeur

- Supposons que nous utilisons la manière suivante pour représenter des nombres :
  1. **0** est un nombre.
  2. Si **X** is est un nombre, alors **succ(X)** est un nombre.

Exemple 3 : Successeur

```
numeral(0).
numeral(succ(X)):- numeral(X).
```

Exemple 3 : Successeur

```
numeral(0).
numeral(succ(X)):- numeral(X).
```

```
?- numeral(succ(succ(succ(0)))).
yes
?-
```

Exemple 3 : Successeur

```
numeral(0).
numeral(succ(X)):- numeral(X).
```

```
?- numeral(X).
```

Exemple 3 : Successeur

```
numeral(0).
numeral(succ(X)):- numeral(X).
```

```
?- numeral(X).
X=0;
X=succ(0);
X=succ(succ(0));
X=succ(succ(succ(0)));
X=succ(succ(succ(succ(0))))
```

Exemple 4: Addition

?- add(succ(succ(0)),succ(succ(succ(0))), Result).
Result=succ(succ(succ(succ(succ(0)))))
yes

Exemple 4: Addition

```
add(0,X,X).                          %%% base clause
```

```
?- add(succ(succ(0)),succ(succ(succ(0))), Result).
Result=succ(succ(succ(succ(succ(0)))))
yes
```

Exemple 4: Addition

```
add(0,X,X).                          %%% base clause

add(succ(X),Y,succ(Z)):-            %%% recursive clause
      add(X,Y,Z).
```

```
?- add(succ(succ(0)),succ(succ(succ(0))), Result).
Result=succ(succ(succ(succ(succ(0)))))
yes
```

Arbre de recherche

- Dessiner l'arbre de recherche

```
add(0,X,X).                          %%% base clause

add(succ(X),Y,succ(Z)):-             %%% recursive clause
    add(X,Y,Z).
```

```
?- add(succ(succ(0)),succ(succ(succ(0))), Result).
Result=succ(succ(succ(succ(succ(0)))))
yes
```

Exercices

Prolog et Logique

- Prolog est le premier langage de programmation logique “raisonnable”
  - Le programmeur spécifie de manière déclarative un problème en utilisant la langage de la logique
  - Le programme ne doit pas dire à l’ordinateur ce qu’il doit faire
  - Pour obtenir une information, le programmeur construit simplement une requête

Prolog et Logique

- Cependant, Prolog n'est pas un langage de programmation logique complet !
- Prolog utilise une approche spécifique pour répondre aux reqûetes :
  - Recherche dans la base de connaissance de haut en bas
  - Traitement des clauses de gauche à droite
  - *Backtracking* pour revenir en arrière en cas de mauvais choix

descend1.pl

```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).

descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z), descend(Z,Y).
```

```
?- descend(A,B).
A=anna
B=bridget
```

descend2.pl

```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).

descend(X,Y):- child(X,Z), descend(Z,Y).
descend(X,Y):- child(X,Y).
```

```
?- descend(A,B).
A=anna
B=emily
```

descend3.pl

```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).

descend(X,Y):- descend(Z,Y), child(X,Z).
descend(X,Y):- child(X,Y).
```

```
?- descend(A,B).
ERROR: OUT OF LOCAL STACK
```

descend4.pl

```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).

descend(X,Y):- child(X,Y).
descend(X,Y):- descend(Z,Y), child(X,Z).
```

```
?- descend(A,B).
```

Résumé de cette séance

- Prédicats récursifs
- Differences entre les aspects déclaratifs et procéduraux en Prolog
- Mise en exergue de quelques limitations de Prolog en tant que langage de programmation logique

Prochaine séance

- Les **listes** en Prolog
  - Structure de données récursive, importante pour programmer en Prolog
  - Définition du prédicat member/2, outil fondamental pour traîter les listes en Prolog
  - Parcours récursif de listes