Systèmes Experts 6
Prolog - Introduction

Richard.Moot@labri.fr
http://www.labri.fr/perso/moot/SE/

Projet
Votre Système Expert en Prolog

- Formez vos équipes (de 3 ou 4 personnes max),
- Trouvez votre domaine d'expertise : un sujet que vous connaissez et aimez et dont votre système expert va devoir donner de recommandations - films de Tarantino, restaurants à Bordeaux, vins, etc.
- Mais on commence avec les bases de Prolog aujourd'hui

Prolog - Naissance

- 1971 - Robert Kowalski (Edimbourg) Résolution
- 1972 - Premier implémentation d'un interpréteur de Prolog par Alain Colmerauer (Marseille)
- 1977 - Premier compilateur grâce aux travaux de David H.D. Warren (Edimbourg).
- 1987 - Constraint Logic Programming.

Prolog - Applications

- Systèmes experts
- Bases de données
- Planning, vérification des circuits électroniques
- Traitement automatique de langage naturel

Prolog - Introduction

- Prolog - Programmation en logique,
- Prolog utilise les clauses de Horn (pour les programmes) et la résolution (pour l'exécution de ces programmes)
- Prolog utilise l'unification pour trouver la solution pour des quantificateurs universels
- La syntaxe est très proche de celle de la logique des prédicats.

Prolog - Introduction

- SWI Prolog (Prolog gratuit) http://www.swi-prolog.org/
- Introduction à Prolog (en Anglais, mais le livre est disponible en Français) http://www.learnprolognow.org/

Prolog - Bons habitudes

Mais pour d'autres langages de programmation aussi

- écrivez du commentaire avec votre programme (environ un ligne de commentaire pour un ligne de code).
- choisissez de noms clairs pour des prédicats (fonctions dans des autres langages) et des variables, écrivez clairement le sens du prédicat.

Prolog Syntaxe

- atomes: séquences commençant avec un minuscule, suivi des minuscules, chiffres et "_"

jean haut_medoc bordeaux1855

Prolog Syntaxe

- atomes: séquences commençant avec un minuscule, suivi des minuscules, chiffres et "_" (aussi permis: 'Terme' (atome entre guillemets) et => (séquence de symboles parmi +-*/\<>:.?@#$&)

Attention: beaucoup de ces séquences ont un sens précis !

'Haut-Medoc' <==

jean haut_medoc bordeaux1855

Prolog Syntaxe

- atomes: séquences commençant avec un minuscule, suivi des minuscules, chiffres et "_" (aussi permis: 'Terme' (atome entre guillemets) et => (séquence de symboles parmi +-*/\<>:.?@#$&)

Attention: beaucoup de ces séquences ont un sens précis !

'Haut-Medoc' <==

jean haut_medoc bordeaux1855

Recommandation: utiliser plutôt des atomes simples

Prolog Syntaxe

- atomes: séquences commençant avec un minuscule, suivi des minuscules, chiffres et "_" (aussi permis: 'Terme' (atome entre guillemets) et => (séquence des symboles +-*/\<>:.?@#$&)
- variables: commencent avec un majuscule, suive par une combinaison de minuscules, majuscules, chiffres et "_".

X Appellation Liste0

Prolog Syntaxe

- nombres entiers: séquences des chiffres, préfixé "-" optionnel 23 -492 0
- nombres réels: 23.5 1.32E-21 10.0e100
- on appelle l'ensemble des atomes, nombres entiers et nombres réels les termes atomiques.
- les termes atomiques correspondent aux constantes dans la logique des prédicats.

Prolog Syntaxe - Termes

- des termes atomiques (atomes, nombres entiers et nombres réels) et des variables sont des termes
- si a est un atome et $t_1,\ldots,t_n$ sont des termes, alors $a(t_1,\ldots,t_n)$ est un terme. on dit que a est le symbole de fonction (Anglais **functor**), les termes t ses arguments et n son arité

livre(anna_karenina) auteur(leo_tolstoy)

contient(bibliotheque, livres, 479)

contient(bibliotheque, journeaux, 171)

Prolog Syntaxe - Termes

- des termes atomiques (atomes, nombres entiers et nombres réels) et des variables sont des termes
- si a est un atome et $t_1,\ldots,t_n$ sont des termes, alors $a(t_1,\ldots,t_n)$ est un terme.

X +(X,Y) =(X,3) =(X,+(X,1))

Prolog Syntaxe - Termes

- des termes atomiques (atomes, nombres entiers et nombres réels) et des variables sont des termes
- si a est un atome et $t_1,\ldots,t_n$ sont des termes, alors $a(t_1,\ldots,t_n)$ est un terme.

X +(X,Y) =(X,3) =(X,+(X,1))

X+Y X=3 X=X+1

Prolog permet d'écrire certains expressions sous forme infixe

Prolog Syntaxe - Termes

- des termes atomiques (atomes, nombres entiers et nombres réels) et des variables sont des termes
- si a est un atome et $t_1,\ldots,t_n$ sont des termes, alors $a(t_1,\ldots,t_n)$ est un terme.

X +(X,Y) =(X,3) =(X,+(X,1))

X+Y X=3 X=X+1

Attention: 3+2 et 5 sont des termes différents !

Prolog Syntaxe - Prédicats

- si a est un atome et $t_1,\ldots,t_n$ sont des termes, alors $a(t_1,\ldots,t_n)$ est un prédicat.
- alors, comme dans la logique des prédicats, les termes et les prédicats ont la même syntaxe

Terme = structure de données

Prédicat = formule logique atomique

= programme

Attention: 3+2 et 5 sont des termes différents !

Prolog Syntaxe - Clauses

- si p, q, r sont des prédicats, les expressions suivantes sont des clauses.
- on a deux types des clauses: des faits, du forme "p." et des règles du forme "p :- q,...,r."

on peut voir des clauses du forme "p." comme ayant la forme "p :- true." (pourquoi?)

Prolog Syntaxe - Clauses

- si p, q, r sont des prédicats, les expressions suivantes sont des clauses.
- on a deux types des clauses: des faits, du forme "p." et des règles du forme "p :- q,...,r."

```
auteur(leo_tolstoy, anna_karenina).
auteur(X) :-
    auteur(X,Y).
livre(Y) :-
    auteur(X,Y).
```

$\forall x.\forall y\ \text{auteur}(x,y) \rightarrow \text{auteur}(x)$

$\forall x.\forall y\ \text{auteur}(x,y) \rightarrow \text{livre}(y)$

Prolog Syntaxe - Clauses

- si p, q, r sont des prédicats, les expressions suivantes sont des clauses.
- on a deux types des clauses: des faits, du forme "p." et des règles du forme "p :- q,...,r."

```
auteur(leo_tolstoy, anna_karenina).
auteur(X) :-
    auteur(X,_).
livre(Y) :-
    auteur(_,Y).
```

variable anonyme

$\forall x.\forall y\ \text{auteur}(x,y) \rightarrow \text{auteur}(x)$

$\forall x.\forall y\ \text{auteur}(x,y) \rightarrow \text{livre}(y)$

Prolog Syntaxe - Programme

- Un programme en Prolog est un ensemble de clauses.

```
auteur(leo_tolstoy, anna_karenina).
auteur(X) :-
      auteur(X,_).
livre(Y) :-
      auteur(_,Y).
```

Prolog Syntaxe - Programme

- Si p,q,r,... sont de prédicats, une question (Anglais: query) est du forme

```
?- p,q,...,r.
```

```
auteur(leo_tolstoy, anna_karenina).
auteur(X) :-
      auteur(X,_).
livre(Y) :-
      auteur(_,Y).
```

Prolog Syntaxe - Programme

```
?- auteur(X).
```

```
auteur(leo_tolstoy, anna_karenina).
auteur(X) :-
      auteur(X,_).
livre(Y) :-
      auteur(_,Y).
```

Prolog Syntaxe - Programme

```
?- auteur(X).
```

Remarque: c'est Prolog qui fournit le "?-"

```
auteur(leo_tolstoy, anna_karenina).
auteur(X) :-
      auteur(X,_).
livre(Y) :-
      auteur(_,Y).
```

Prolog Syntaxe - Programme

?- auteur(X).

Interprétation: pour quel X peut-on démontrer qu'il est auteur?

```
auteur(leo_tolstoy, anna_karenina).
auteur(X) :-
        auteur(X,_).
livre(Y) :-
        auteur(_,Y).
```

Sémantique

- Un programme en Prolog a un sens déclaratif, qui est sa traduction directe en logique, ":-" correspond à "→", "," (entre prédicats) correspond à "∧" et les variables de chaque clause sont (implicitement) quantifié par ∀
- Le sens procédurel est donné par la résolution; ici l'ordre des clauses et l'ordre des prédicats dans un clause est important.

Sémantique

- Le sens procédurel est donné par la résolution; ici l'ordre des clauses et l'ordre des prédicats dans un clause est important.
- Prolog commence toujours avec le premier clause qui correspond aux premier prédicat de la question.
- Prolog garde les autres clauses qui correspondent à la question : en cas d'échec, le prochain clause est essayé.

Prolog Syntaxe - Programme

?- auteur(X).

Seul possibilité !

```
auteur(leo_tolstoy, anna_karenina).
auteur(X) :-
        auteur(X,_).
livre(Y) :-
        auteur(_,Y).
```

Prolog Syntaxe - Programme

```
?- auteur(X,_).
```

Seul possibilité !

```
auteur(leo_tolstoy, anna_karenina).
auteur(X) :-
    auteur(X,_).
livre(Y) :-
    auteur(_,Y).
```

Prolog Syntaxe - Programme

```
?- auteur(X,_).
```

Seul possibilité !

```
auteur(leo_tolstoy, anna_karenina).
auteur(X) :-
    auteur(X,_).
livre(Y) :-
    auteur(_,Y).
```

Prolog Syntaxe - Programme

```
?- auteur(X,_).
```

Réponse: X = leo_tolstoy

Seul possibilité !

```
auteur(leo_tolstoy, anna_karenina).
auteur(X) :-
    auteur(X,_).
livre(Y) :-
    auteur(_,Y).
```

Prolog Syntaxe - Programme

```
?- auteur(X,Y).


auteur(leo_tolstoy, anna_karenina).
auteur(X) :-
    auteur(X,_).
livre(Y) :-
    auteur(_,Y).
```

Prolog Syntaxe - Programme

?- auteur(X,anna_karenina).

```
auteur(leo_tolstoy, anna_karenina).
auteur(X) :-
      auteur(X,_).
livre(Y) :-
      auteur(_,Y).
```

Prolog Syntaxe - Programme

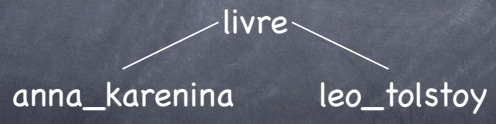
?- auteur(X),livre(Y).

```
auteur(leo_tolstoy, anna_karenina).
auteur(X) :-
      auteur(X,_).
livre(Y) :-
      auteur(_,Y).
```

Unification

- On peut voir des termes comme des arbres.

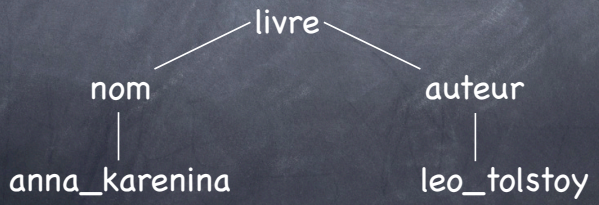
livre(anna_karenina,leo_tolstoy)



Unification

- On peut voir des termes comme des arbres.

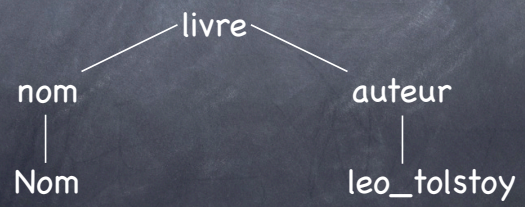
livre(nom(anna_karenina),auteur(leo_tolstoy))



Unification

- On peut voir des termes comme des arbres.

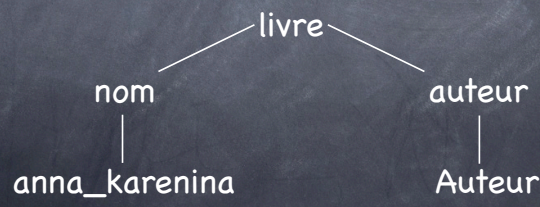
livre(nom(Nom),auteur(leo_tolstoy))



Unification

- On peut voir des termes comme des arbres.

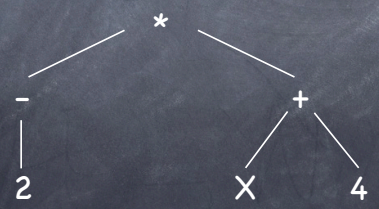
livre(nom(anna_karenina),auteur(Auteur))



Unification

- On peut voir des termes comme des arbres.

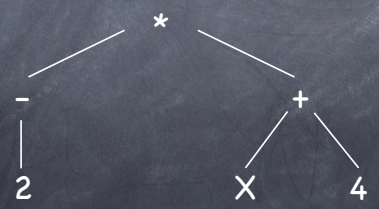
-2*(X+4) = *(-2,+(X,4)



Unification

- Les feuilles sont des termes atomiques et des variables.

-2*(X+4) = *(-2,+(X,4)



Unification

- L'unification est un algorithme simple pour déterminer si deux termes peuvent être égaux en donnant des valeurs aux variables.

Cas fréquent: unification entre une variable et un autre terme.

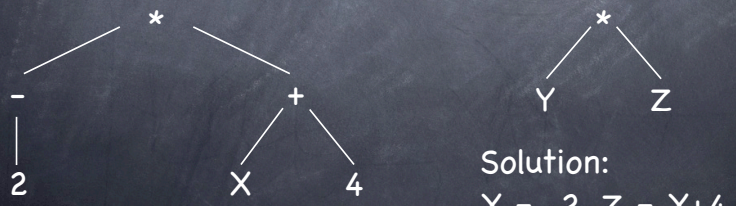
X = 3

Unification

- L'unification est un algorithme simple pour déterminer si deux termes peuvent être égaux en donnant des valeurs aux variables.

-2*(X+4) = *(-2,+(X,4)

Y*Z = *(Y,Z)



Solution:
X = -2, Z = X+4

Unification

- L'unification est un algorithme simple pour déterminer si deux termes peuvent être égaux en donnant des valeurs aux variables.

Unification

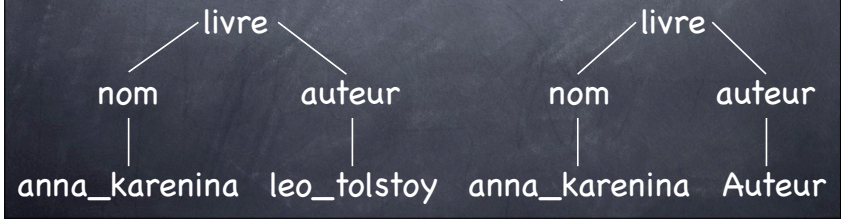
- L'unification est un algorithme simple pour déterminer si deux termes peuvent être égaux en donnant des valeurs aux variables.

Solution
Nom = anna_karenina
Auteur = leo_tolstoy

Unification

- L'unification est un algorithme simple pour déterminer si deux termes peuvent être égaux en donnant des valeurs aux variables.

Solution
Nom = anna_karenina
Auteur = leo_tolstoy



Unification

- Prolog utilise l'unification pour chaque résolution d'un littéral de la question avec un clause.
- On peut utiliser l'unification explicitement, en utilisant X = Y unifie les termes X et Y (à éviter)
- Un cas difficile est X = f(X). La réponse correct serait un échec (pourquoi?), la réponse de Prolog est f(f(f(f(....))))

Unification

Exemples

f(X) = f(a,b)
f(X,g(b,Y)) = g(b,a)
f(X,h(a)) = f(g(Z),h(Z))
f(g(b,a),c) = f(g(X,X),Y)

Unification

Exemples

~~f(X) = f(a,b)~~
~~f(X,g(b,Y)) = g(b,a)~~
f(X,h(a)) = f(g(Z),h(Z)) ← f(g(a),h(a)
~~f(g(b,a),c) = f(g(X,X),Y)~~ X = g(a)
Z = a

Unificatiom, Egalité et Inégalité

X = Y	unifie X et Y
X == Y	X et Y sont strictement égaux
X \== Y	Alors X == X et a == a sont vrai Mais X == Y et X == a sont faux

Unificatiom, Egalité et Inégalité

X = Y	unifie X et Y
X == Y	X et Y sont strictement égaux
X \== Y	X == Y est faux

Unificatiom, Egalité et Inégalité

X = Y	unifie X et Y
X == Y	X et Y sont strictement égaux
X \== Y	X == Y est faux

Alors X \== X et a \== a sont faux
Mais X \== Y et X \== a sont vrai

Sémantique Déclarative

1. étant donnée une question q, r, ... et une programme avec des clauses t :- s, u, v. dont on appelle t la tête du clause, et s, u, v le corps.
2. on prend le premier clause dont la tête t unifie avec le premier prédicat atomique q de la question, en cas d'échec on continue avec le prochain clause.
3. on remplace q pas le corps du clause et continue avec 2. jusqu'au moment où la question est vide.

Exemple avec des termes

- 0 dénote le nombre 0,
- si X dénote le nombre N, s(X) dénote le nombre N + 1,

0	0
1	s(0)
2	s(s(0))
3	s(s(s(0)))

Bien sûr que ceci n'est pas un façon très efficace pour coder les entiers !

Résolution Clauses de Horn

```
addition(0,X,X).
addition(s(X),Y,s(Z)) :-
    addition(X,Y,Z).

multiplication(0,X,0).
multiplication(s(X),Y,Z) :-
    multiplication(X,Y,V),
    addition(V,Y,Z).
```

Résolution Clauses de Horn

```
addition(0,X,X).
addition(s(X),Y,s(Z)) :-
    addition(X,Y,Z).

multiplication(0,X,0).
multiplication(s(X),Y,Z) :-
    multiplication(X,Y,V),
    addition(V,Y,Z).
```

```
addition(s(s(0),s(s(0)),Z).
```

Résolution Clauses de Horn

```
addition(0,X,X).
addition(s(X),Y,s(Z)) :-
    addition(X,Y,Z).

multiplication(0,X,0).
multiplication(s(X),Y,Z) :-
    multiplication(X,Y,V),
    addition(V,Y,Z).
```

```
addition(s(s(0),s(s(0)),Z).
```

échec d'unification
0 ≠ s(s(0))

Résolution
Clauses de Horn

```
addition(0,X,X).
addition(s(X),Y,s(V)) :-
    addition(X,Y,V).

multiplication(0,X,0).
multiplication(s(X),Y,Z) :-
    multiplication(X,Y,V),
    addition(V,Y,Z).
```

```
addition(s(s(0),s(s(0)),Z).
```

X = s(0)
Y = s(s(0))
Z = s(V)

Z = s(V)

Résolution
Clauses de Horn

```
addition(0,X,X).
addition(s(X),Y,s(V)) :-
    addition(X,Y,V).

multiplication(0,X,0).
multiplication(s(X),Y,Z) :-
    multiplication(X,Y,V),
    addition(V,Y,Z).
```

X = s(0)
Y = s(s(0))
Z = s(V)

On remplace la question par addition(X,Y,V) = addition(s(0),s(s(0)),V)

Z = s(V)

Résolution
Clauses de Horn

```
addition(0,X,X).
addition(s(X),Y,s(W)) :-
    addition(X,Y,W).

multiplication(0,X,0).
multiplication(s(X),Y,Z) :-
    multiplication(X,Y,V),
    addition(V,Y,Z).
```

```
addition(s(0),s(s(0)),V).
```

X = 0
Y = s(s(0))
V = s(W)

Z = s(V)

Résolution
Clauses de Horn

```
addition(0,X,X).
addition(s(X),Y,s(W)) :-
    addition(X,Y,W).

multiplication(0,X,0).
multiplication(s(X),Y,Z) :-
    multiplication(X,Y,V),
    addition(V,Y,Z).
```

```
addition(s(0),s(s(0)),V).
```

X = 0
Y = s(s(0))
V = s(W)

Unification est possible que avec la deuxième clause.
s(0) ≠ 0

Z = s(s(W))

Résolution
Clauses de Horn

```
addition(0,X,X).
addition(s(X),Y,s(W)) :-
    addition(X,Y,W).

multiplication(0,X,0).
multiplication(s(X),Y,Z) :-
    multiplication(X,Y,V),
    addition(V,Y,Z).
```

```
addition(s(0),s(s(0)),V).
```

X = 0

Y = s(s(0))

V = s(W)

Remarque: la solution Z devient plus grande

Z = s(V) = s(s(W))

Z = s(s(W))

Résolution
Clauses de Horn

```
addition(0,X,X).
addition(s(X),Y,s(W)) :-
    addition(X,Y,W).

multiplication(0,X,0).
multiplication(s(X),Y,Z) :-
    multiplication(X,Y,V),
    addition(V,Y,Z).
```

X = 0

Y = s(s(0))

V = s(W)

On remplace la question par addition(X,Y,V) = addition(0,s(s(0)),W)

Z = s(s(W))

Résolution
Clauses de Horn

```
addition(0,X,X).
addition(s(X),Y,s(W)) :-
    addition(X,Y,W).

multiplication(0,X,0).
multiplication(s(X),Y,Z) :-
    multiplication(X,Y,V),
    addition(V,Y,Z).
```

```
addition(0,s(s(0)),W).
```

W = X

X = s(s(0))

Z = s(s(W))

Résolution
Clauses de Horn

```
addition(0,X,X).
addition(s(X),Y,s(W)) :-
    addition(X,Y,W).

multiplication(0,X,0).
multiplication(s(X),Y,Z) :-
    multiplication(X,Y,V),
    addition(V,Y,Z).
```

```
addition(0,s(s(0)),W).
```

W = X

X = s(s(0))

Finalement, le premier clause s'applique et on trouve une solution.

Z = s(s(s(s(0))))

Résolution
Clauses de Horn

```
addition(0,X,X).
addition(s(X),Y,s(Z)) :-
    addition(X,Y,Z).

multiplication(0,X,0).
multiplication(s(X),Y,Z) :-
    multiplication(X,Y,V),
    addition(V,Y,Z).
```

```
addition(X,s(0),s(s(0))).
```

Un avantage de la formulation des propriétés en logique comme ça est qu'il n'y a pas vraiment d'entrée et de sortie et qu'on peut poser des questions comme on veut.

Résolution
Clauses de Horn

```
addition(0,X,X).
addition(s(V),Y,s(Z)) :-
    addition(V,Y,Z).

multiplication(0,X,0).
multiplication(s(X),Y,Z) :-
    multiplication(X,Y,V),
    addition(V,Y,Z).
```

```
addition(X,s(0),s(s(0))).
```

X = s(V)
Y = s(0)
Z = s(0)

Résolution
Clauses de Horn

```
addition(0,X,X).
addition(s(V),Y,s(Z)) :-
    addition(V,Y,Z).

multiplication(0,X,0).
multiplication(s(X),Y,Z) :-
    multiplication(X,Y,V),
    addition(V,Y,Z).
```

```
addition(V,s(0),s(0)).
```

X = s(V)
Y = s(0)
Z = s(0)

X = s(V)

Résolution
Clauses de Horn

```
addition(0,W,W).
addition(s(X),Y,s(Z)) :-
    addition(X,Y,Z).

multiplication(0,X,0).
multiplication(s(X),Y,Z) :-
    multiplication(X,Y,V),
    addition(V,Y,Z).
```

```
addition(V,s(0),s(0)).
```

V = 0
W = s(0)

X = s(V)

Résolution
Clauses de Horn

```
addition(0,W,W).
addition(s(X),Y,s(Z)) :-
    addition(X,Y,Z).

multiplication(0,X,0).
multiplication(s(X),Y,Z) :-
    multiplication(X,Y,V),
    addition(V,Y,Z).
```

```
addition(V,s(0),s(0)).
```

V = 0

W = s(0)

X = s(0)

Résolution
Clauses de Horn

```
addition(0,Z,Z).
addition(s(V),Y,s(Z)) :-
    addition(V,Y,Z).

multiplication(0,X,0).
multiplication(s(X),Y,Z) :-
    multiplication(X,Y,V),
    addition(V,Y,Z).
```

```
addition(X,Y,s(s(0))).
```

X = s(V)

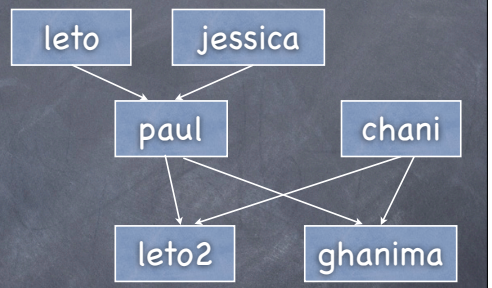
Y = Z

Z = s(s(0))

Familles

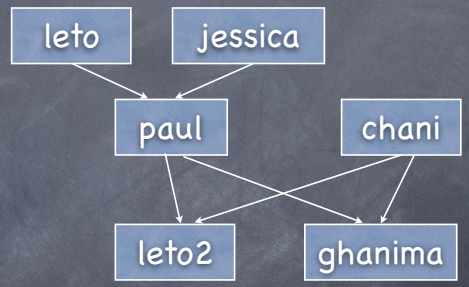
- C'est presque obligatoire de parler des familles un premier cours de Prolog.
- Alors, on en parlera un petit peu.

Familles



Familles

```
parent(leto, paul).
parent(jessica, paul).
parent(paul, leto2).
parent(paul, ghanima).
parent(chani, leto2).
parent(chani, ghanima).
```



Familles

```
parent(leto, paul).
parent(jessica, paul).
parent(paul, leto2).
parent(paul, ghanima).
parent(chani, leto2).
parent(chani, ghanima).
```

```
homme(leto).
homme(paul).
homme(leto2).

femme(jessica).
femme(chani).
femme(ghanima).
```

Familles

```
parent(leto, paul).
parent(jessica, paul).
parent(paul, leto2).
parent(paul, ghanima).
parent(chani, leto2).
parent(chani, ghanima).

pere(Parent, Enfant) :-
    parent(Parent, Enfant),
    homme(Parent).
```

```
homme(leto).
homme(paul).
homme(leto2).

femme(jessica).
femme(chani).
femme(ghanima).

mere(Parent, Enfant) :-
    parent(Parent, Enfant),
    femme(Parent).
```

Familles

```
pere(leto, paul).
pere(paul, leto2).
pere(paul, ghanima).

mere(jessica, paul).
mere(chani, leto2).
mere(chani, ghanima).

parent(Parent, Enfant) :-
    pere(Parent, Enfant).

parent(Parent, Enfant) :-
    mere(Parent, Enfant).
```

On peut choisir les prédicats de base qui nous conviennent. Alors, on pourrait prendre pere/2 et mere/2 comme prédicats de base et donner des règles pour parent/2.

Familles

```
pere(leto, paul).
pere(paul, leto2).
pere(paul, ghanima).

mere(jessica, paul).
mere(chani, leto2).
mere(chani, ghanima).

parent(Parent, Enfant) :-
    pere(Parent, Enfant).

parent(Parent, Enfant) :-
    mere(Parent, Enfant).
```

Remarque: on a perdu de l'information ! homme/1 et femme/1 ne sont pas totalement définissable en termes de pere/2 et mere/2.
Pourquoi?

Familles

```
parent(leto, paul).
parent(jessica, paul).
parent(paul, leto2).
parent(paul, ghanima).
parent(chani, leto2).
parent(chani, ghanima).

homme(leto).
homme(paul).
homme(leto2).

femme(jessica).
femme(chani).
femme(ghanima).

grandpere(GrandParent, Enfant) :-
    parent(GrandParent, Parent),
    parent(Parent, Enfant),
    homme(GrandParent).
```

Familles

```
parent(leto, paul).
parent(jessica, paul).
parent(paul, leto2).
parent(paul, ghanima).
parent(chani, leto2).
parent(chani, ghanima).

homme(leto).
homme(paul).
homme(leto2).

femme(jessica).
femme(chani).
femme(ghanima).

grandpere(GrandParent, Enfant) :-
    parent(GrandParent, Parent),
    parent(Parent, Enfant),
    homme(GrandParent).

?- grandpere(leto, PetitEnfant).
```

GrandParent = leto,
PetitEnfant = Enfant

Familles

```
parent(leto, paul).
parent(jessica, paul).
parent(paul, leto2).
parent(paul, ghanima).
parent(chani, leto2).
parent(chani, ghanima).

homme(leto).
homme(paul).
homme(leto2).

femme(jessica).
femme(chani).
femme(ghanima).

grandpere(leto, PetitEnfant) :-
    parent(leto, Parent),
    parent(Parent, PetitEnfant),
    homme(leto).

?- parent(leto, Parent), parent(Parent, PetitEnfant),
homme(leto).
```

GrandParent = leto,
PetitEnfant = Enfant

Familles

```
parent(leto, paul).
parent(jessica, paul).
parent(paul, leto2).
parent(paul, ghanima).
parent(chani, leto2).
parent(chani, ghanima).

homme(leto).
homme(paul).
homme(leto2).

femme(jessica).
femme(chani).
femme(ghanima).

grandpere(leto, PetitEnfant) :-
        parent(leto, Parent),
        parent(Parent, PetitEnfant),
        homme(leto).
```

GrandParent = leto,
PetitEnfant = Enfant

```
?- parent(leto, Parent), parent(Parent, PetitEnfant),
homme(leto).
```

Familles

```
parent(leto, paul).
parent(jessica, paul).
parent(paul, leto2).
parent(paul, ghanima).
parent(chani, leto2).
parent(chani, ghanima).

homme(leto).
homme(paul).
homme(leto2).

femme(jessica).
femme(chani).
femme(ghanima).

grandpere(GrandParent, Enfant) :-
        parent(GrandParent, Parent),
        parent(Parent, Enfant),
        homme(GrandParent).
```

Parent = paul

```
?- parent(leto, paul), parent(paul, PetitEnfant), homme(leto).
```

Familles

```
parent(leto, paul).
parent(jessica, paul).
parent(paul, leto2).
parent(paul, ghanima).
parent(chani, leto2).
parent(chani, ghanima).

homme(leto).
homme(paul).
homme(leto2).

femme(jessica).
femme(chani).
femme(ghanima).

grandpere(GrandParent, Enfant) :-
        parent(GrandParent, Parent),
        parent(Parent, Enfant),
        homme(GrandParent).
```

PetitEnfant = leto2

```
?- parent(paul, PetitEnfant), homme(leto).
```

Familles

```
parent(leto, paul).
parent(jessica, paul).
parent(paul, leto2).
parent(paul, ghanima).
parent(chani, leto2).
parent(chani, ghanima).

homme(leto).
homme(paul).
homme(leto2).

femme(jessica).
femme(chani).
femme(ghanima).

grandpere(GrandParent, Enfant) :-
        parent(GrandParent, Parent),
        parent(Parent, Enfant),
        homme(GrandParent).
```

PetitEnfant = leto2

```
?- parent(paul, leto2), homme(leto).
```

Familles

```
parent(leto, paul).
parent(jessica, paul).
parent(paul, leto2).
parent(paul, ghanima).
parent(chani, leto2).
parent(chani, ghanima).

homme(leto).
homme(paul).
homme(leto2).

femme(jessica).
femme(chani).
femme(ghanima).

grandpere(GrandParent, Enfant) :-
    parent(GrandParent, Parent),
    parent(Parent, Enfant),
    homme(GrandParent).
```

Solution: PetitEnfant = leto2

Familles

```
parent(leto, paul).
parent(jessica, paul).
parent(paul, leto2).
parent(paul, ghanima).
parent(chani, leto2).
parent(chani, ghanima).

homme(leto).
homme(paul).
homme(leto2).

femme(jessica).
femme(chani).
femme(ghanima).

grandpere(GrandParent, Enfant) :-
    parent(GrandParent, Parent),
    parent(Parent, Enfant),
    homme(GrandParent).
```

";" demande à Prolog de trouve des autres solutions

Solution: PetitEnfant = leto2

Familles

```
parent(leto, paul).
parent(jessica, paul).
parent(paul, leto2).
parent(paul, ghanima).
parent(chani, leto2).
parent(chani, ghanima).

homme(leto).
homme(paul).
homme(leto2).

femme(jessica).
femme(chani).
femme(ghanima).

grandpere(GrandParent, Enfant) :-
    parent(GrandParent, Parent),
    parent(Parent, Enfant),
    homme(GrandParent).
```

PetitEnfant = ghanima

```
?- parent(paul, PetitEnfant), homme(leto).
```

Familles

```
parent(leto, paul).
parent(jessica, paul).
parent(paul, leto2).
parent(paul, ghanima).
parent(chani, leto2).
parent(chani, ghanima).

homme(leto).
homme(paul).
homme(leto2).

femme(jessica).
femme(chani).
femme(ghanima).

grandpere(GrandParent, Enfant) :-
    parent(GrandParent, Parent),
    parent(Parent, Enfant),
    homme(GrandParent).
```

PetitEnfant = ghanima

```
?- parent(paul, ghanima), homme(leto).
```

Familles

```
parent(leto, paul).
parent(jessica, paul).
parent(paul, leto2).
parent(paul, ghanima).
parent(chani, leto2).
parent(chani, ghanima).

homme(leto).
homme(paul).
homme(leto2).

femme(jessica).
femme(chani).
femme(ghanima).

grandpere(GrandParent, Enfant) :-
    parent(GrandParent, Parent),
    parent(Parent, Enfant),
    homme(GrandParent).
```

PetitEnfant = ghanima

?- homme(leto).

Familles

```
parent(leto, paul).
parent(jessica, paul).
parent(paul, leto2).
parent(paul, ghanima).
parent(chani, leto2).
parent(chani, ghanima).

homme(leto).
homme(paul).
homme(leto2).

femme(jessica).
femme(chani).
femme(ghanima).

grandpere(GrandParent, Enfant) :-
    parent(GrandParent, Parent),
    parent(Parent, Enfant),
    homme(GrandParent).
```

PetitEnfant = ghanima

Solution: PetitEnfant = ghanima

Familles

```
parent(leto, paul).
parent(jessica, paul).
parent(paul, leto2).
parent(paul, ghanima).
parent(chani, leto2).
parent(chani, ghanima).

homme(leto).
homme(paul).
homme(leto2).

femme(jessica).
femme(chani).
femme(ghanima).

grandpere(GrandParent, Enfant) :-
    parent(GrandParent, Parent),
    parent(Parent, Enfant),
    homme(GrandParent).
```

";" une deuxième fois ne donne pas d'autres solutions.

Solution: PetitEnfant = ghanima

Familles

```
parent(leto, paul).
parent(jessica, paul).
parent(paul, leto2).
parent(paul, ghanima).
parent(chani, leto2).
parent(chani, ghanima).

homme(leto).
homme(paul).
homme(leto2).

femme(jessica).
femme(chani).
femme(ghanima).

grandpere(GrandParent, Enfant) :-
    parent(GrandParent, Parent),
    parent(Parent, Enfant),
    homme(GrandParent).
```

";" une deuxième fois ne donne pas d'autres solutions.

?- parent(paul, PetitEnfant), homme(leto).

Familles

```
parent(leto, paul).
parent(jessica, paul).
parent(paul, leto2).
parent(paul, ghanima).
parent(chani, leto2).
parent(chani, ghanima).

homme(leto).
homme(paul).
homme(leto2).

femme(jessica).
femme(chani).
femme(ghanima).

grandpere(GrandParent, Enfant) :-
    parent(GrandParent, Parent),
    parent(Parent, Enfant),
    homme(GrandParent).
```

";" une deuxième fois ne donne pas d'autres solutions.

```
?- parent(leto, Parent), parent(Parent, PetitEnfant),
homme(leto).
```

Familles

```
parent(leto, paul).
parent(jessica, paul).
parent(paul, leto2).
parent(paul, ghanima).
parent(chani, leto2).
parent(chani, ghanima).

homme(leto).
homme(paul).
homme(leto2).

femme(jessica).
femme(chani).
femme(ghanima).

ancetre(Ancetre, Descendant) :-
    parent(Ancetre, Descendant).
ancetre(Ancetre, Descendant) :-
    parent(Parent, Descendant),
    ancetre(Ancetre, Parent).
```

Familles

```
parent(leto, paul).
parent(jessica, paul).
parent(paul, leto2).
parent(paul, ghanima).
parent(chani, leto2).
parent(chani, ghanima).

homme(leto).
homme(paul).
homme(leto2).

femme(jessica).
femme(chani).
femme(ghanima).

ancetre(Ancetre, Descendant) :-
    parent(Ancetre, Descendant).
ancetre(Ancetre, Descendant) :-
    ancetre(Ancetre1, Descendant),
    ancetre(Ancetre, Ancetre1).
```

Familles

```
?- ancetre(A, jean).
```

```
ancetre(Ancetre, Descendant) :-
    parent(Ancetre, Descendant).
ancetre(Ancetre, Descendant) :-
    ancetre(Ancetre1, Descendant),
    ancetre(Ancetre, Ancetre1).
```

Familles

?- parent(A, jean).

échec (car on ne sait rien de jean).

ancetre(Ancetre, Descendant) :-
    parent(Ancetre, Descendant).
ancetre(Ancetre, Descendant) :-
    ancetre(Ancetre1, Descendant),
    ancetre(Ancetre, Ancetre1).

Familles

?- ancetre(A1,A), ancetre(A, jean).

on revient sur ancetre et on essaye la deuxième (et dernier) clause

ancetre(Ancetre, Descendant) :-
    parent(Ancetre, Descendant).
ancetre(Ancetre, Descendant) :-
    ancetre(Ancetre1, Descendant),
    ancetre(Ancetre, Ancetre1).

Familles

?- ancetre(A2,A1), ancetre(A1,A), ancetre(A, jean).

... et on peut continuer comme ça

ancetre(Ancetre, Descendant) :-
    parent(Ancetre, Descendant).
ancetre(Ancetre, Descendant) :-
    ancetre(Ancetre1, Descendant),
    ancetre(Ancetre, Ancetre1).

Familles

?- ancetre(A3,A2), ancetre(A2,A1), ancetre(A1,A), ancetre(A, jean).

... et on peut continuer comme ça

Conclusion: une bonne définition en logique ne donne pas nécessairement un bon program

ancetre(Ancetre, Descendant) :-
    parent(Ancetre, Descendant).
ancetre(Ancetre, Descendant) :-
    ancetre(Ancetre1, Descendant),
    ancetre(Ancetre, Ancetre1).

Arithmétique

- Malgré le fait qu'on peut faire l'arithmétique avec des termes en Prolog (et de façon purement logique), il est utile de pouvoir faire du calcul arithmétique directement.
- Prolog fournit le prédicat "is" pour faire ça.

Arithmétique

- Par exemple, on peut utiliser le prédicat:
  X is X0 + 1
- Ceci donne une erreur si on ne sait pas la valeur de X0
- X est un variable.

Arithmétique

- Quel est le sens de l'expression
  X is X + 1 ?
- Pour avoir un sens, on doit déjà connaître la valeur pour tout les variables à droite de "is". Alors supposons qu'on sait la valeur de X dans notre programme et que X est 2. L'expression X+1 s'évalue alors comme 3.
- On finit par 2 is 3 ce qui correspond à faux !

Arithmétique

- X is Y + Z
- X est une variable libre, dont on sait pas encore la valeur.
- Y et Z sont des variable déjà liées, dont on sait les valeurs. Prolog donne une erreur si cette condition n'est pas respecté pendant la résolution du programme.

Arithmétique

- X is Y + Z
- rappel que ceci est un façon d'écrire is(X,+(Y,Z))
- Etant donnée que des expressions avec "is" n'ont un sens que quand on sait les valeurs des variables à droite de "is", cette opération n'est pas purement logique.

Arithmétique

- Opérations de comparaison entre expressions arithmétique. Ces opérations ont un sens que quand on connaît X et Y.
- X < Y, X =< Y,
- X > Y, X >= Y,
- X =:= Y, X =\= Y

Arithmétique

```
% = max(X, Y, Z)
% vrai si Z est le maximum de X et Y.

max(X,Y,X) :-
      X >= Y.
max(X,Y,Y) :-
      Y > X.
```

Arithmétique

```
% = factoriel(X, F)
% vrai si F est le factoriel de X.

factoriel(0, 1).
factoriel(N0, F) :-
      N0 > 0,
      N is N0 - 1,
      factoriel(N, F0),
      F is F0 * N0.
```

Listes

- Une structure de données utile, avec des abréviations syntaxiques spéciales, est le liste.
- Le liste vide: []
- Chaque liste non-vide est du forme [H|T], avec H le premier élément du liste (Head) et T un liste contenant les autres éléments (Tail).

Listes

- Liste
- []
- [1|[]]
- [1|[2|[]]]
- [1|[2|[3|[]]]]
- [1|[2|[3|[4|[]]]]]

- Version simple
- []
- [1]
- [1,2]
- [1,2,3]
- [1,2,3,4]

Listes

```
% = est_liste(Terme)
% vrai si Terme est un liste

est_liste([]).
est_liste([_|L]) :-
      est_liste(L).
```

[2,3]
[2|[]]
[1|2]
[1,2|3]
[1,2|[3]]

Listes

```
% = est_liste(Terme)
% vrai si Terme est un liste

est_liste([]).
est_liste([_|L]) :-
      est_liste(L).
```

[2,3] = [2|[3|[]]]
[2|[]] = [2]
~~[1|2]~~
~~[1,2|3]~~
[1,2|[3]] =
[1,2,3] =
[1|[2|[3|[]]]]

Listes

```
% = membre(Element, Liste)
%
% vrai si Liste contient Element.

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

Listes

```
% = append(Liste1, Liste2, Liste3)
%
% vrai si Liste3 contient les élements de Liste1
% suivi par les élements de Liste2, c'est-à-dire
% Liste3 est la concatenation de Liste1 et
% Liste2

append([], Ys, Ys).
append([X|Xs], Ys, [Z|Zs]) :-
        append(Xs, Ys, Zs).
```

Listes

```
% = maximum(Liste, Max)
% vrai si Max est la valeur du plus grand
% element de Liste.

maximum([Max], Max).
maximum([H|T], Max) :-
        maximum(T, M0),
        max(M0, H, Max).
```

Listes

```
% = maximum(Liste, Max)
% vrai si Max est la valeur du plus grand
% element de Liste.

maximum([Max], Max).
maximum([H|T], Max) :-
        maximum(T, M0),
        max(M0, H, Max).

?- maximum([1,2,3],M).
```

Listes

```
% = maximum(Liste, Max)
% vrai si Max est la valeur du plus grand
% element de Liste.

maximum([Max], Max).
maximum([H|T], Max) :-
      maximum(T, M0),
      max(M0, H, Max).

?- maximum(T, M0), max(M0, H, Max).
```

H = 1
T = [2,3]
M = Max

Listes

```
% = maximum(Liste, Max)
% vrai si Max est la valeur du plus grand
% element de Liste.

maximum([Max], Max).
maximum([H|T], Max) :-
      maximum(T, M0),
      max(M0, H, Max).

?- maximum([2,3], M0), max(M0, 1, M).
```

H = 1
T = [2,3]
M = Max

Listes

```
% = maximum(Liste, Max)
% vrai si Max est la valeur du plus grand
% element de Liste.

maximum([Max], Max).
maximum([H|T], Max) :-
      maximum(T, M0),
      max(M0, H, Max).

?- maximum(T, M0), max(M0, H, Max),
max(M0, 1, M).
```

H = 2
T = [3]
M0 = Max

Listes

```
% = maximum(Liste, Max)
% vrai si Max est la valeur du plus grand
% element de Liste.

maximum([Max], Max).
maximum([H|T], Max) :-
      maximum(T, M0),
      max(M0, H, Max).

?- maximum([3], M1), max(M1, 2, M0),
max(M0, 1, M).
```

H = 2
T = [3]
M0 = Max

Listes

```
% = maximum(Liste, Max)
% vrai si Max est la valeur du plus grand
% element de Liste.

maximum([Max], Max).
maximum([H|T], Max) :-
        maximum(T, M0),
        max(M0, H, Max).

?- maximum([3], 3), max(3, 2, M0),
max(M0, 1, M).
```

Listes

```
% = maximum(Liste, Max)
% vrai si Max est la valeur du plus grand
% element de Liste.

maximum([Max], Max).
maximum([H|T], Max) :-
        maximum(T, M0),
        max(M0, H, Max).

?- max(3, 2, M0),  max(M0, 1, M).
```

Listes

```
% = maximum(Liste, Max)
% vrai si Max est la valeur du plus grand
% element de Liste.

maximum([Max], Max).
maximum([H|T], Max) :-
        maximum(T, M0),          M = 3
        max(M0, H, Max).

?- max(3, 1, M).
```

Listes

```
% = maximum(Liste, Max)
% vrai si Max est la valeur du plus grand
% element de Liste.

maximum([Max], Max).
maximum([H|T], Max) :-
        maximum(T, M0),          M = 3
        max(M0, H, Max).
```

Listes

```
% = maximum, version deux

maximum([H|T], Max) :-
        maximum(T, H, Max).

maximum([], Max, Max).
maximum([H|T], Max0, Max) :-
        max(H, Max0, Max1),
        maximum(T, Max1, Max).
```

Listes

```
maximum([H|T], Max) :-
        maximum(T, H, Max).

maximum([], Max, Max).
maximum([H|T], Max0, Max) :-
        max(H, Max0, Max1),
        maximum(T, Max1, Max).

?- maximum(T, H, Max).
```

H = 1
T = [2,3]
M = Max

Listes

```
maximum([H|T], Max) :-
        maximum(T, H, Max).

maximum([], Max, Max).
maximum([H|T], Max0, Max) :-
        max(H, Max0, Max1),
        maximum(T, Max1, Max).

?- maximum([2,3], 1, M).
```

H = 1
T = [2,3]
M = Max

Listes

```
maximum([H|T], Max) :-
        maximum(T, H, Max).

maximum([], Max, Max).
maximum([H|T], Max0, Max) :-
        max(H, Max0, Max1),
        maximum(T, Max1, Max).

?- max(H, Max0, Max1), maximum(T, Max1, Max).
```

H = 2
T = [3]
Max0 = 1
M = Max

Listes

```
maximum([H|T], Max) :-
        maximum(T, H, Max).

maximum([], Max, Max).
maximum([H|T], Max0, Max) :-
        max(H, Max0, Max1),
        maximum(T, Max1, Max).
```

H = 2
T = [3]
Max0 = 1
M = Max

?- max(2, 1, Max1), maximum([3], Max1, M).

Listes

```
maximum([H|T], Max) :-
        maximum(T, H, Max).

maximum([], Max, Max).
maximum([H|T], Max0, Max) :-
        max(H, Max0, Max1),
        maximum(T, Max1, Max).
```

H = 2
T = [3]
Max0 = 1
M = Max

?- max(2, 1, 2), maximum([3], 2, M).

Listes

```
maximum([H|T], Max) :-
        maximum(T, H, Max).

maximum([], Max, Max).
maximum([H|T], Max0, Max) :-
        max(H, Max0, Max1),
        maximum(T, Max1, Max).
```

?- maximum([3], 2, M).

Listes

```
maximum([H|T], Max) :-
        maximum(T, H, Max).

maximum([], Max, Max).
maximum([H|T], Max0, Max) :-
        max(H, Max0, Max1),
        maximum(T, Max1, Max).
```

H = 3
T = []
Max0 = 2
Max = M

?- max(H, Max0, Max1), maximum(T, Max1, Max).

Listes

```
maximum([H|T], Max) :-
        maximum(T, H, Max).

maximum([], Max, Max).
maximum([H|T], Max0, Max) :-
        max(H, Max0, Max1),
        maximum(T, Max1, Max).
```

H = 3
T = []
Max0 = 2
Max = M

?- max(3, 2, Max1), maximum([], Max1, M).

Listes

```
maximum([H|T], Max) :-
        maximum(T, H, Max).

maximum([], Max, Max).
maximum([H|T], Max0, Max) :-
        max(H, Max0, Max1),
        maximum(T, Max1, Max).
```

H = 3
T = []
Max0 = 2
Max = M

?- max(3, 2, 3), maximum([], 3, M).

Listes

```
maximum([H|T], Max) :-
        maximum(T, H, Max).

maximum([], Max, Max).
maximum([H|T], Max0, Max) :-
        max(H, Max0, Max1),
        maximum(T, Max1, Max).
```

Max = M = 3

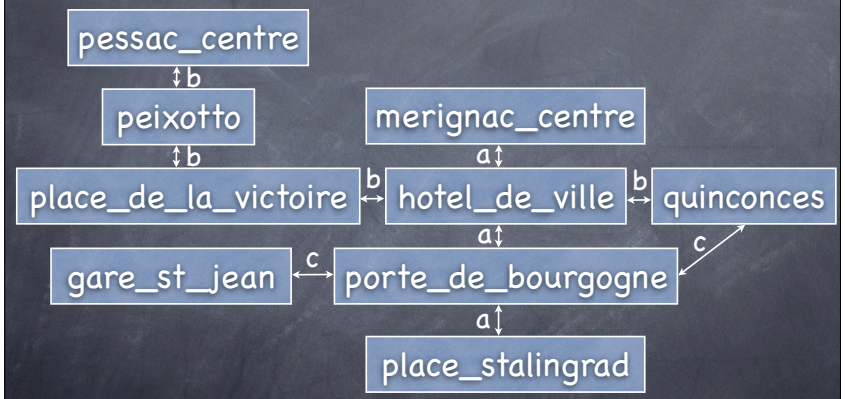
?- maximum([], 3, 3).

Listes

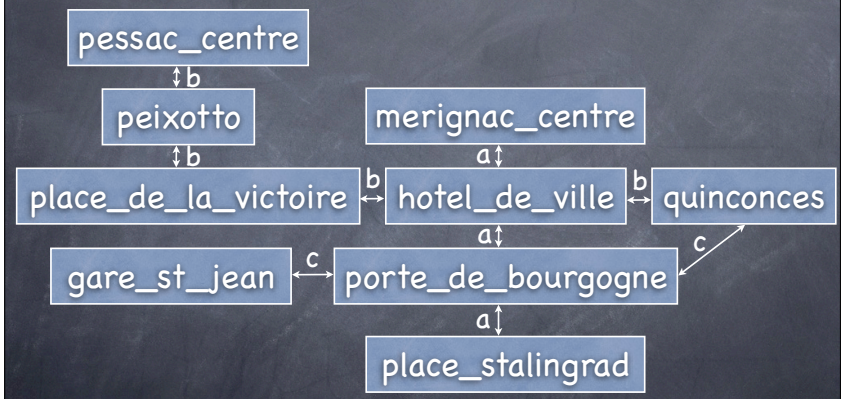
```
maximum([H|T], Max) :-
        maximum(T, H, Max).

maximum([], Max, Max).
maximum([H|T], Max0, Max) :-
        max(H, Max0, Max1),
        maximum(T, Max1, Max).
```

Max = M = 3

Tramway



Tramway




```
connexion(place_stalingrad, porte_de_bourgogne, a).
```

Tramway

On peut prendre chaque connexion dans les deux sens

```
connexion(Source, Destination, Ligne) :-
        connexion(Destination, Source, Ligne).

connexion(place_stalingrad, porte_de_bourgogne, a).
connexion(porte_de_bourgogne, gare_st_jean, c).
...
```

Tramway

```
chemin(Source, Destination) :-
        connexion(Source, Destination, _).
chemin(Source, Destination) :-
        connexion(Source, Arret, _),
        chemin(Arret, Destination).


connexion(Source, Destination, Ligne) :-
        connexion(Destination, Source, Ligne).

connexion(place_stalingrad, porte_de_bourgogne, a).
connexion(porte_de_bourgogne, gare_st_jean, c).
...
```

Tramway

```
chemin(Source, Destination) :-
        connexion(Source, Destination, _).
chemin(Source, Destination) :-
        connexion(Source, Arret, _),
        chemin(Arret, Destination).

connexion(Source, Destination, Ligne) :-
        connexion(Destination, Source, Ligne).

connexion(place_stalingrad, porte_de_bourgogne, a).
connexion(porte_de_bourgogne, gare_st_jean, c).
...
```

Quel est le problème avec ce programme ?

Tramway

```
chemin(Source, Destination, Chemin) :-
        chemin(Source, Destination, [], Chemin).
chemin(Source, Destination, Chemin0, Chemin) :-
        reverse(Chemin0, Chemin1),
        simplifier(Chemin1, Chemin).
chemin(Source, Destination, Chemin0, Chemin) :-
        connexion(Source, Arret, Ligne),
        \+ member(c(_,Arret,_), Chemin0),
        chemin(Arret, Destination,
                [c(Source,Arret,Ligne)|Chemin0], Chemin).

connexion(place_stalingrad, porte_de_bourgogne, a).
connexion(porte_de_bourgogne, gare_st_jean, c).
...
```