

PROLOG 1

Prolog - Initiation

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 - Bonnes habitudes

(et pour d'autres langages de programmation aussi)

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

Prolog Syntaxe

④ 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 de 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

④ atomes: séquences commençant avec un minuscule, suivi des minuscules, chiffres et “_” (aussi permis: ‘Terme’ (atome entre guillemets) et => (séquence des symboles +-*\<>:.?:@#\$&)

④ variables: commencent avec un majuscule, suivie par une combinaison de minuscules, majuscules, chiffres et “_”.

X

Appellation

Liste0

Prolog Syntaxe

❷ nombres entiers: séquences des chiffres,
préfixé “-” optionnel 23 -492 0

❸ nombres réels: 23.5 1.32E-21
10.0e100

❹ on appelle l'ensemble des atomes,
nombres entiers et nombres réels les
termes atomiques.

❺ les termes atomiques correspondent aux
constantes dans la logique des prédicats.

Prolog Syntaxe - Termes

- ① des termes atomiques (atomes, nombres entiers et nombres réels) et des variables sont des termes
- ② si a est un atome et t_1, \dots, t_n sont des termes, alors $a(t_1, \dots, t_n)$ 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, \dots, t_n sont des termes, alors $a(t_1, \dots, t_n)$ 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, \dots, t_n sont des termes, alors $a(t_1, \dots, t_n)$ 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, \dots, t_n sont des termes, alors $a(t_1, \dots, t_n)$ 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 t₁,...,t_n sont des termes,
alors a(t₁,...,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

Prolog Syntaxe - Clauses

- si p, q, r sont des prédictats, 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 prédictats, les expressions suivantes sont des clauses.
- on a deux types des clauses: des faits, de la forme “p.” et des règles du forme “p :- q,...,r.”

auteur(leo_tolstoy, anna_karenina).

auteur(X) :-
 auteur(X,Y). $\forall x. \forall y \text{ auteur}(x,y) \rightarrow \text{auteur}(x)$

livre(Y) :-
 auteur(X,Y). $\forall x. \forall y \text{ auteur}(x,y) \rightarrow \text{livre}(y)$

Prolog Syntaxe - Clauses

- si p, q, r sont des prédictats, les expressions suivantes sont des clauses.
- on a deux types des clauses: des faits, de forme “p.” et des règles de la forme
- “p :- q,...,r.”

Prolog Syntaxe - Clauses

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 prédicts, une question (Anglais: query) est de la 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 que X 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 “ \rightarrow ”, “,” (entre prédicats) correspond à “ \wedge ” et les variables de chaque clause sont (implicitement) quantifiées par \forall

Sémantique

- Le sens procédural provient de la résolution; à ce niveau l'ordre des clauses et l'ordre des prédictats dans une clause est important.
- Prolog commence toujours avec la première clause qui correspond au premier prédictat de la question.
- Prolog garde les autres clauses qui correspondent à la question : en cas d'échec, on essaie la clause suivante.

Prolog Syntaxe - Programme

?- auteur(X).

Seule
possibilité !

auteur(leo_tolstoy, anna_karenina).

→ auteur(X) :-
 auteur(X,_).
livre(Y) :-
 auteur(_,Y).

Prolog Syntaxe - Programme

?- auteur(X,_).

Seule
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,_).

Seule
possibilité !

Réponse:

X = leo_tolstoy

→ 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).

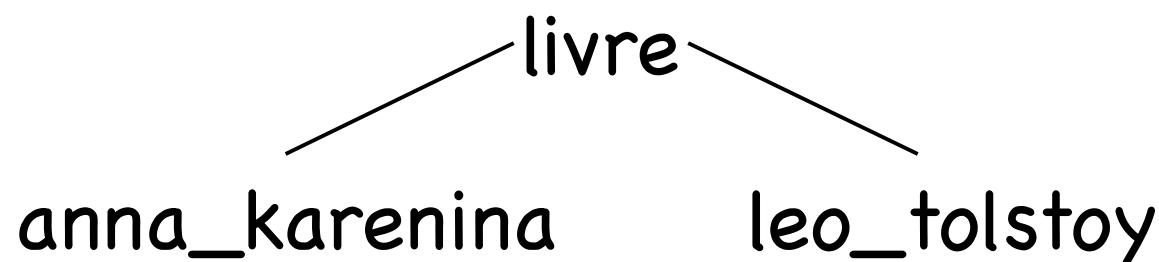
auteur(X) :-
 auteur(X,_).

livre(Y) :-
 auteur(_,Y).

Unification

💡 Les termes vus comme des arbres.

livre(anna_karenina,leo_tolstoy)

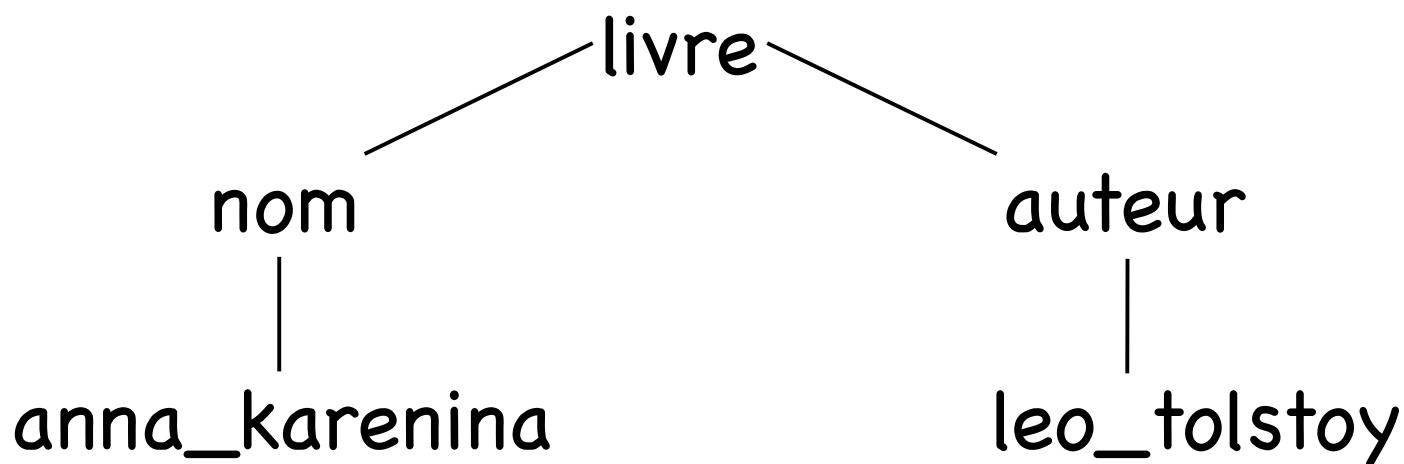


Unification



Les termes vus comme des arbres.

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

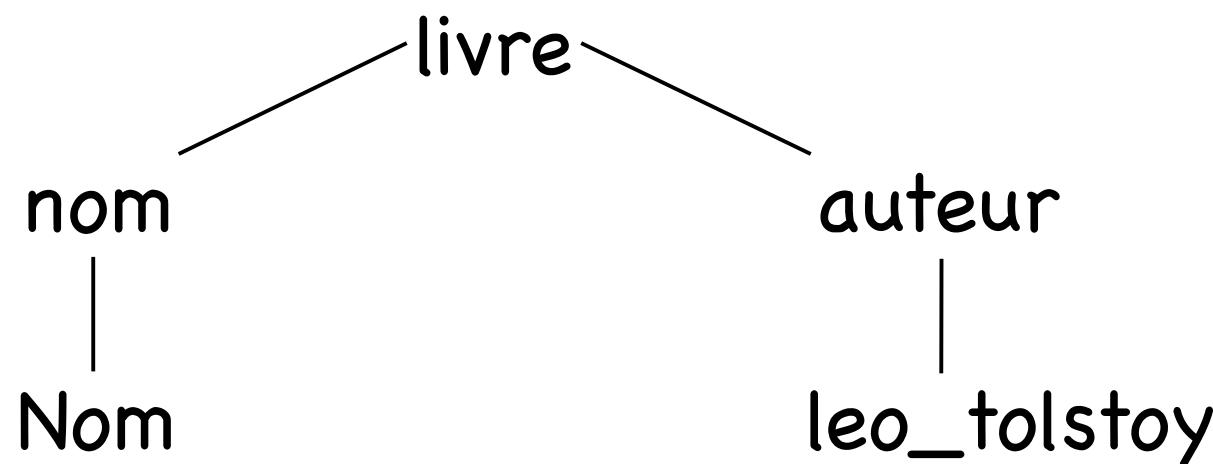


Unification



Les termes vus comme des arbres.

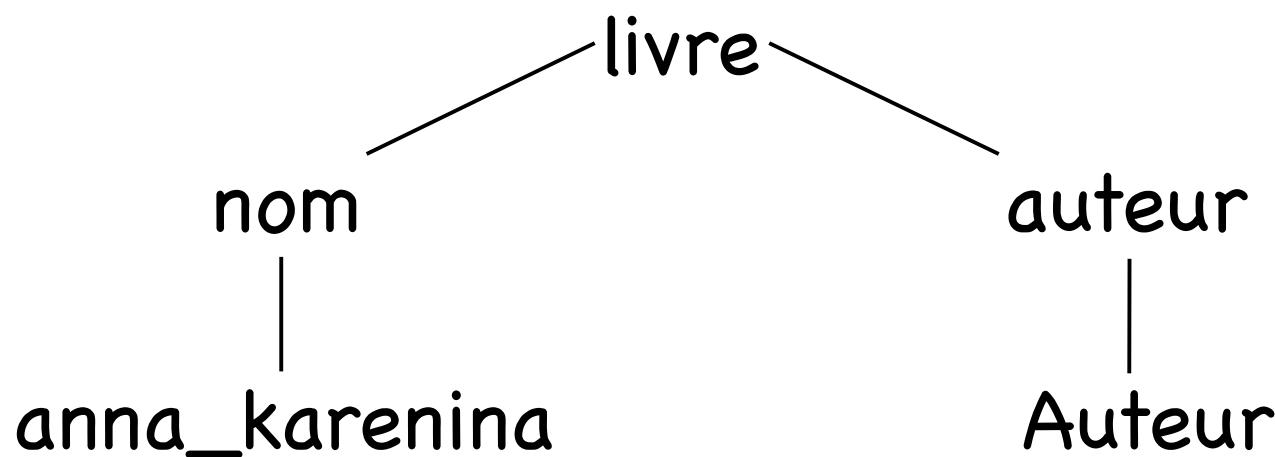
livre(nom(Nom),auteur(leo_tolstoy))



Unification

Les termes vus comme des arbres.

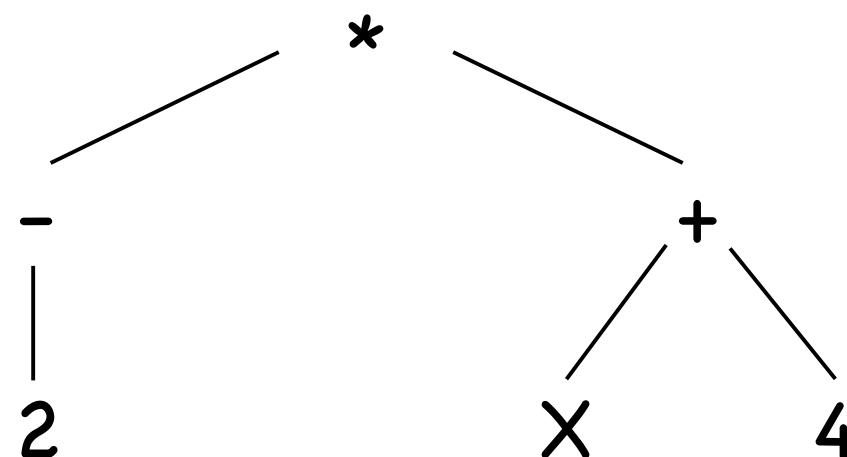
livre(nom(anna_karenina),auteur(Auteur))



Unification

Les termes vus
comme des
arbres.

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

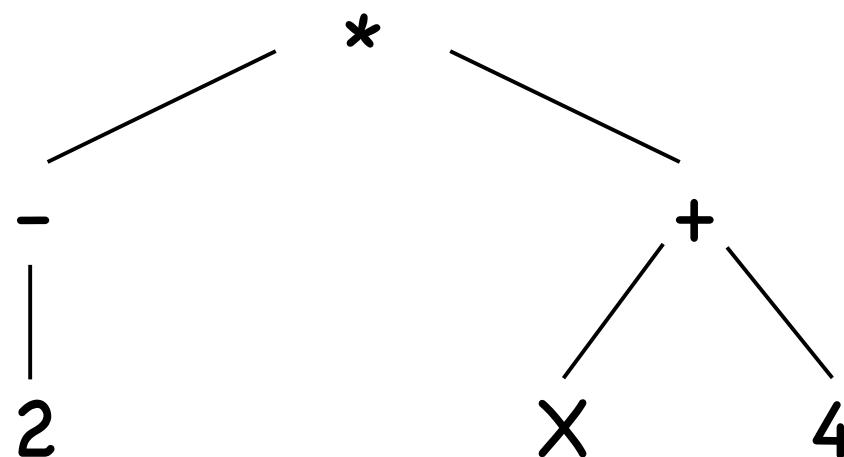


Unification



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.

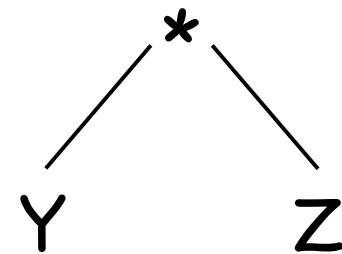
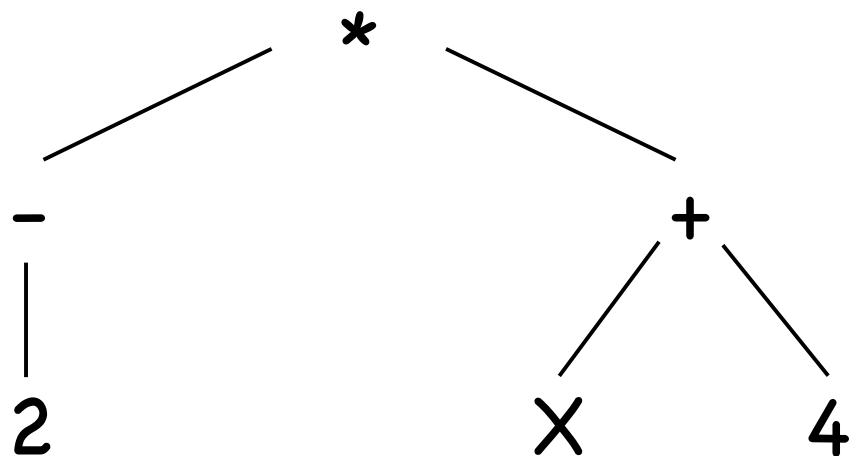
$$x = 3$$

Unification

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

$$-2*(X+4) = *(-2,+(\text{X},4))$$

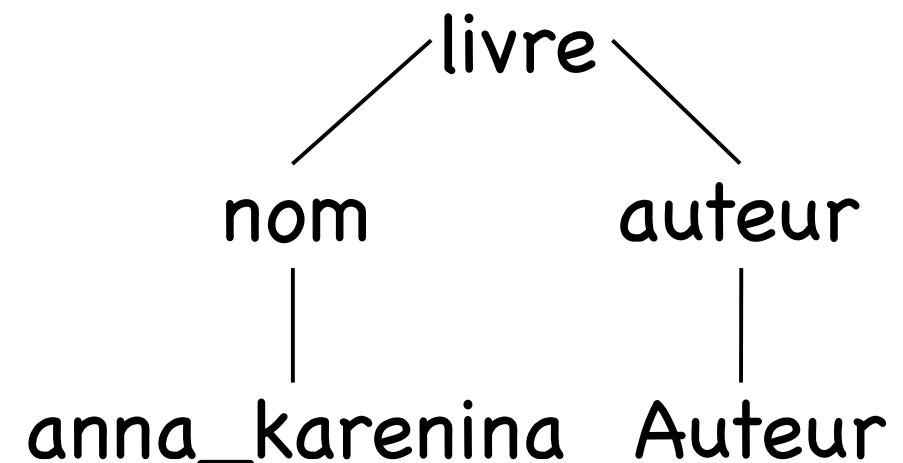
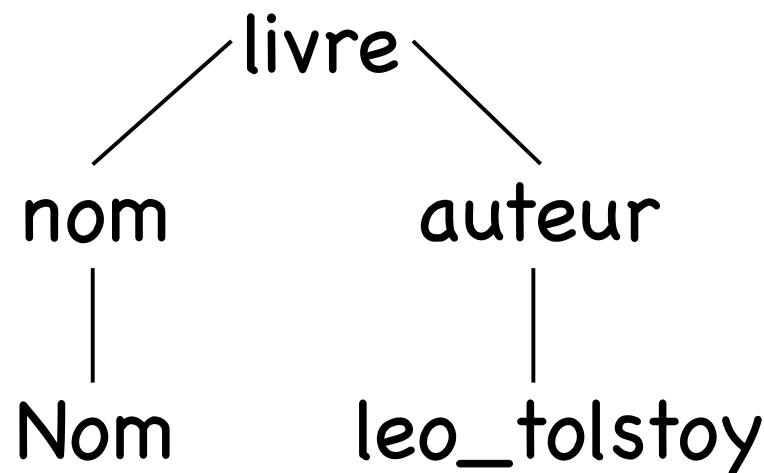
$$Y*Z = *(Y,Z)$$



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.



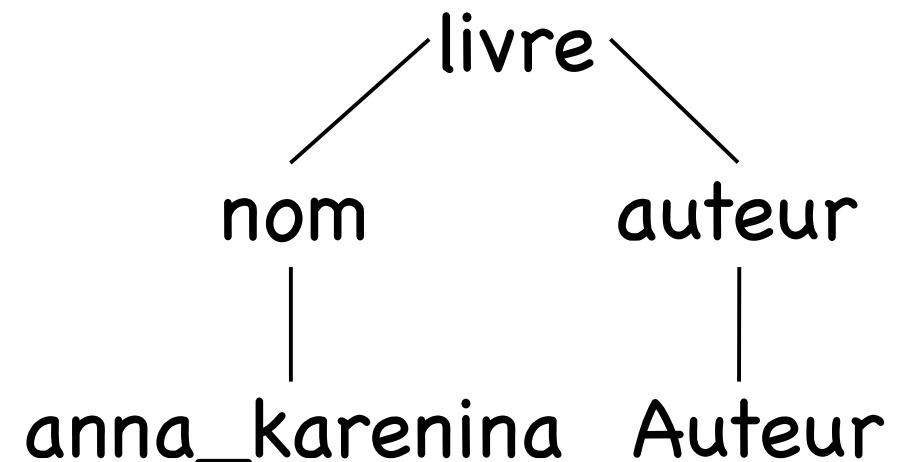
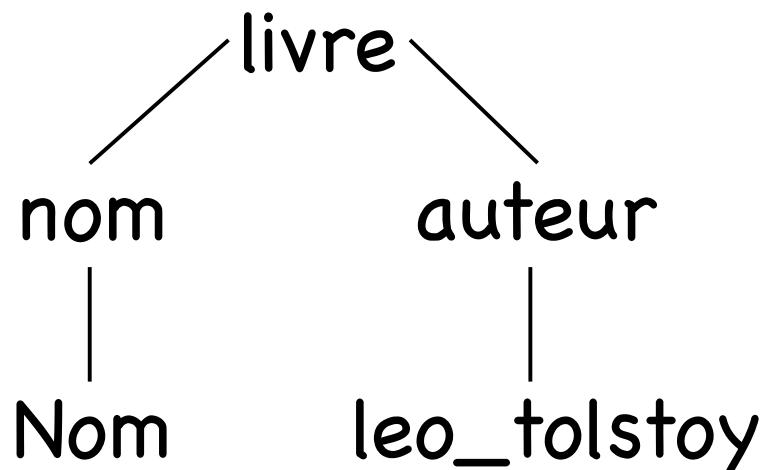
Unification

💡 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



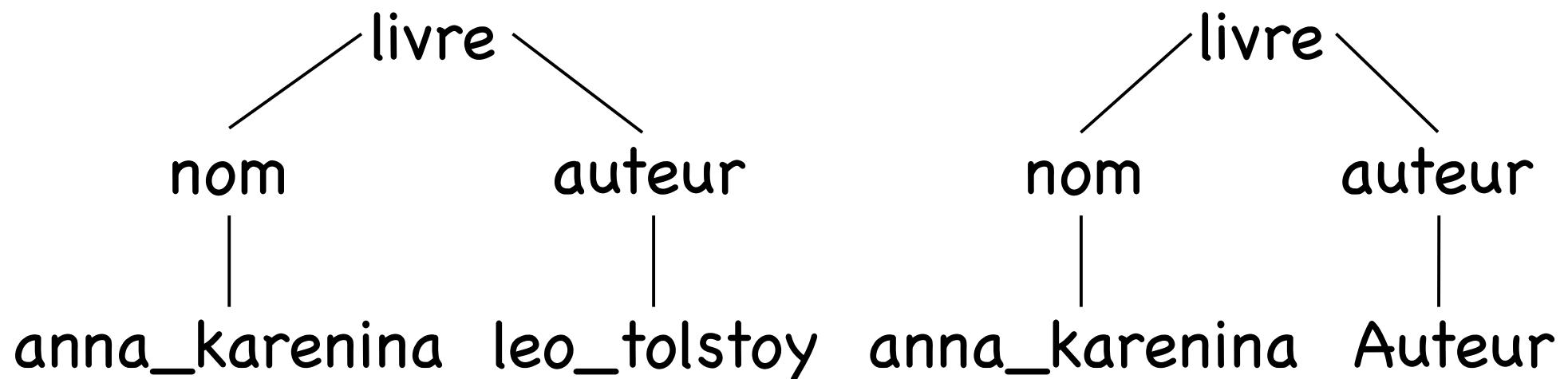
Unification

4 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 atome de la question avec une 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 $f(f(f(f(\dots))))$

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

$$\cancel{f(X) = f(a,b)}$$

$$\cancel{f(X,g(b,Y)) = g(b,a)}$$

$$f(X,h(a)) = f(g(Z),h(Z)) \quad \leftarrow f(g(a),h(a))$$

$$\cancel{f(g(b,a),c) = f(g(\cancel{X},\cancel{X}),Y)} \quad X = g(a)$$

$$Z = a$$

Unification, Egalité et Inégalité

$X = Y$ unifie X et Y

$X == Y$ X et Y sont strictement égaux

$X \backslash== Y$ Alors $X == X$ et $a == a$ sont vrais
 Mais $X == Y$ et $X == a$ sont faux

Unification, Egalité et Inégalité

$X = Y$

unifie X et Y

$X == Y$

X et Y sont strictement égaux

$X \backslash== Y$

$X == Y$ est faux

Unification, Egalité et Inégalité

$X = Y$ unifie X et Y

$X == Y$ X et Y sont strictement égaux

$X \neq Y$ $X == Y$ est faux
Alors $X \neq X$ et $a \neq a$ sont
faux

Mais $X \neq Y$ et $X \neq a$ sont vrai

Sémantique Déclarative

1. étant donnée une question q, r, ... et un programme avec des clauses t :- s, u, v. dont on appelle t la tête, et s, u, v le corps.
2. on prend la première clause dont la tête t s'unifie avec le premier prédicat atomique q de la question, en cas d'échec on continue avec la clause suivante.
3. on remplace q par le corps de la clause et on 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 une façon efficace de 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
 $0 \neq s(s(0))$

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
 $\text{addition}(X,Y,V) =$
 $\text{addition}(s(0),s(s(0)),V)$

$Z = s(V)$

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.
 $s(0) \neq 0$

$Z = s(s(W))$

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
 $\text{addition}(X,Y,V) = \text{addition}(0,s(s(0)),W)$

$Z = s(s(W))$

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, la première 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 est qu'il n'y a pas vraiment d'entrée et de sortie: 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).
```

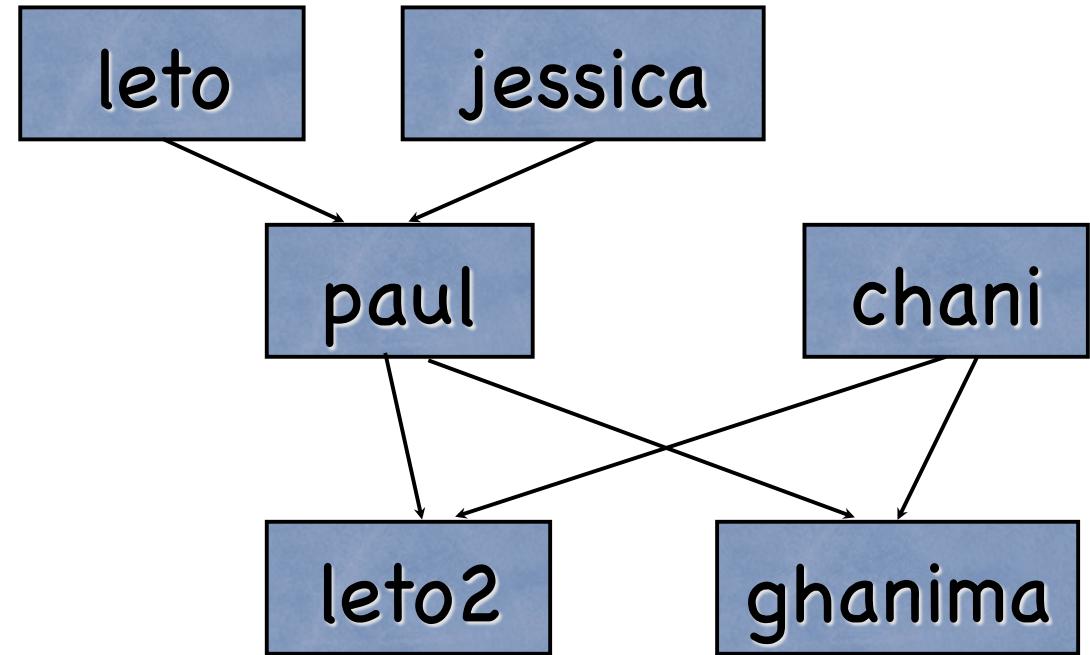
addition(X,Y,s(s(0))).

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

Familles

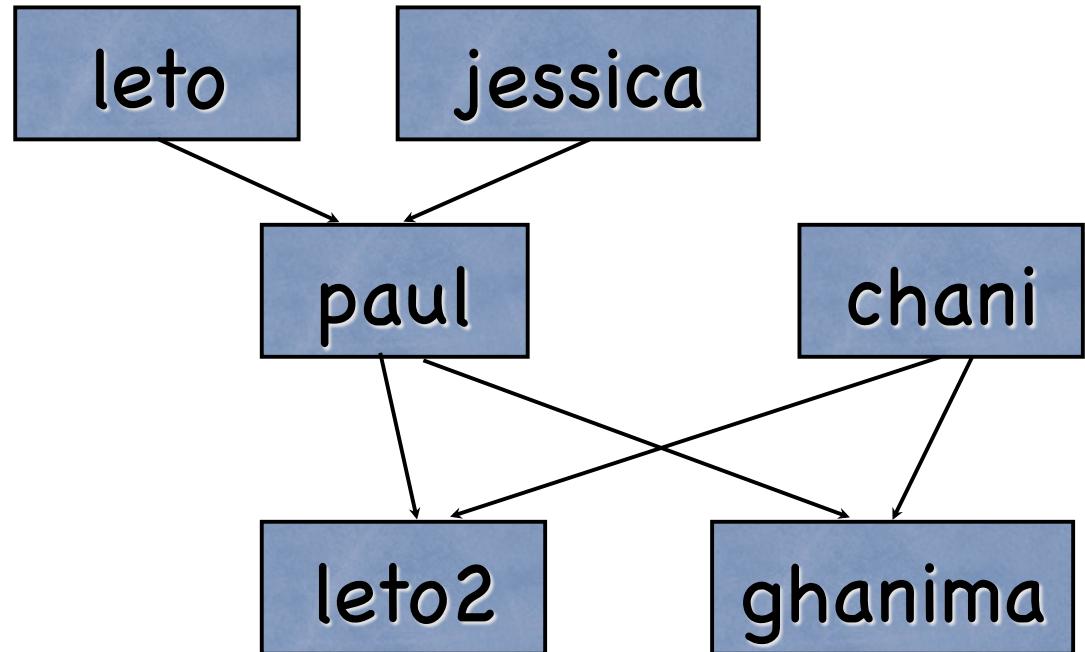
- ➊ C'est presque obligatoire de parler des familles dans un premier cours de Prolog.
- ➋ Donc, on en parle un peu.

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).

homme(leto).

homme(paul).

homme(leto2).

femme(jessica).

femme(chani).

femme(ghanima).

pere(Parent, Enfant) :-

parent(Parent, Enfant),

homme(Parent).

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).
```

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

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

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).

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?

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

GrandParent = leto,
PetitEnfant = Enfant

?- grandpere(leto, PetitEnfant).

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(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 = paul

parent(Parent, Enfant),

homme(GrandParent).

?- 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 d’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 ancêtre et on essaie la deuxième (et dernière) 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 programme

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

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

Arithmétique

- ➊ 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.
- ➋ Prolog fournit le prédictat “is” pour faire ça.

Arithmétique

- ➊ Par exemple, on peut utiliser le prédicat:
 $X \text{ is } X_0 + 1$
- ➋ Ceci donne une erreur si on ne sait pas la valeur de X_0
- ➌ X est une variable.

Arithmétique

- ➊ Quel est le sens de l'expression $X \text{ is } X + 1$?
- ➋ 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.
- ➌ On finit par $2 \text{ is } 3$ ce qui correspond à faux !

Arithmétique

④ X is Y + Z

④ X est une variable libre, dont on sait pas encore la valeur.

④ Y et Z sont des variables déjà liées, dont on sait les valeurs. Prolog donne une erreur si cette condition n'est pas respectée pendant la résolution du programme.

Arithmétique

④ $X \text{ is } Y + Z$

④ rappel que ceci est un façon d'écrire
 $\text{is}(X, +(Y, Z))$

④ Etant donnée que des expressions avec “is” n’ont un sens que quand on connaît les valeurs des variables à droite de “is”, cette opération n’est pas purement logique.

Arithmétique

- ④ Opérations de comparaison entre expressions arithmétique. Ces opérations ont un sens que quand on connaît X et Y.
 - ④ $X < Y$, $X = < Y$,
 - ④ $X > Y$, $X >= Y$,
 - ④ $X =:= Y$, $X =\backslash= Y$

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

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

Listes

➊ Liste

➋ []

➌ [1 | []]

➍ [1 | [2 | []]]

➎ [1 | [2 | [3 | []]]]

➏ [1 | [2 | [3 | [4 | []]]]]

➐ Version simple

➑ []

➒ [1]

➓ [1,2]

➔ [1,2,3]

➕ [1,2,3,4]

Listes

```
% = est_liste(Terme) [2,3]
% vrai si Terme est un liste [2|[]]
est_liste([]). [1|2]
est_liste([_|L]) :- [1,2|3]
    est_liste(L). [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 éléments de Liste1
% suivi par les éléments de Liste2, c'est-à-dire
% Liste3 est la concatenation de Liste1 et
% Liste2

append([], Ys, Ys).

append([X|Xs], Ys, [X|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).

H = 1

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

T = [2,3]

 maximum(T, M0),

M = Max

 max(M0, H, 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).

H = 1

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

T = [2,3]

 maximum(T, M0),

M = Max

 max(M0, H, Max).

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

Listes

% = maximum(Liste, Max)

% vrai si Max est la valeur du plus grand
% élément de Liste.

maximum([Max], Max).

H = 2

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

T = [3]

 maximum(T, M0),

M0 = Max

 max(M0, H, Max).

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

Listes

% = maximum(Liste, Max)

% vrai si Max est la valeur du plus grand
% element de Liste.

maximum([Max], Max).

H = 2

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

T = [3]

 maximum(T, M0),

M0 = Max

 max(M0, H, Max).

?- maximum([3], M1), max(M1, 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).

?- 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) :-

M = 3

 maximum(T, M0),
 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 , M_0),

$$M = 3$$

`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).

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

?- maximum(T, H, 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 = 1

T = [2,3]

M = Max

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

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

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

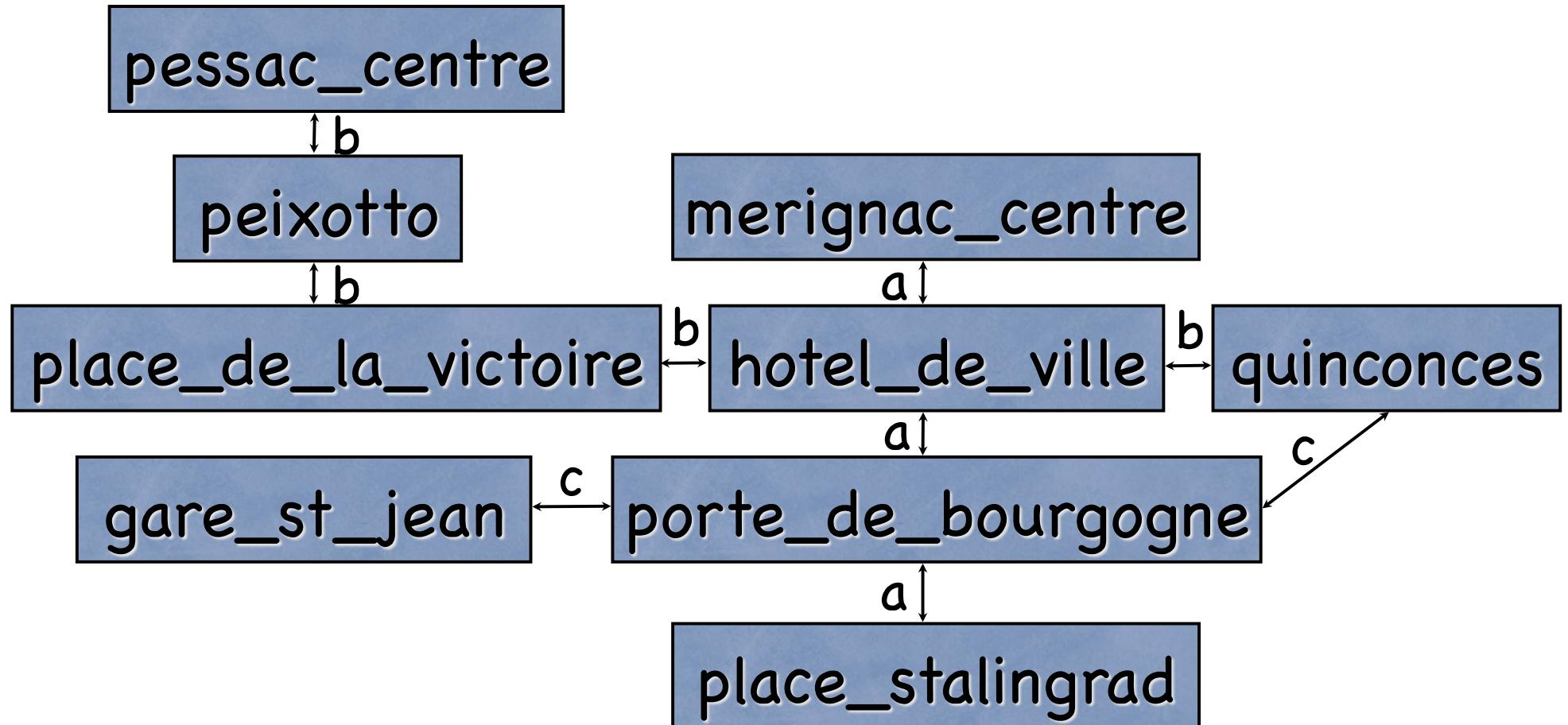
Listes

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

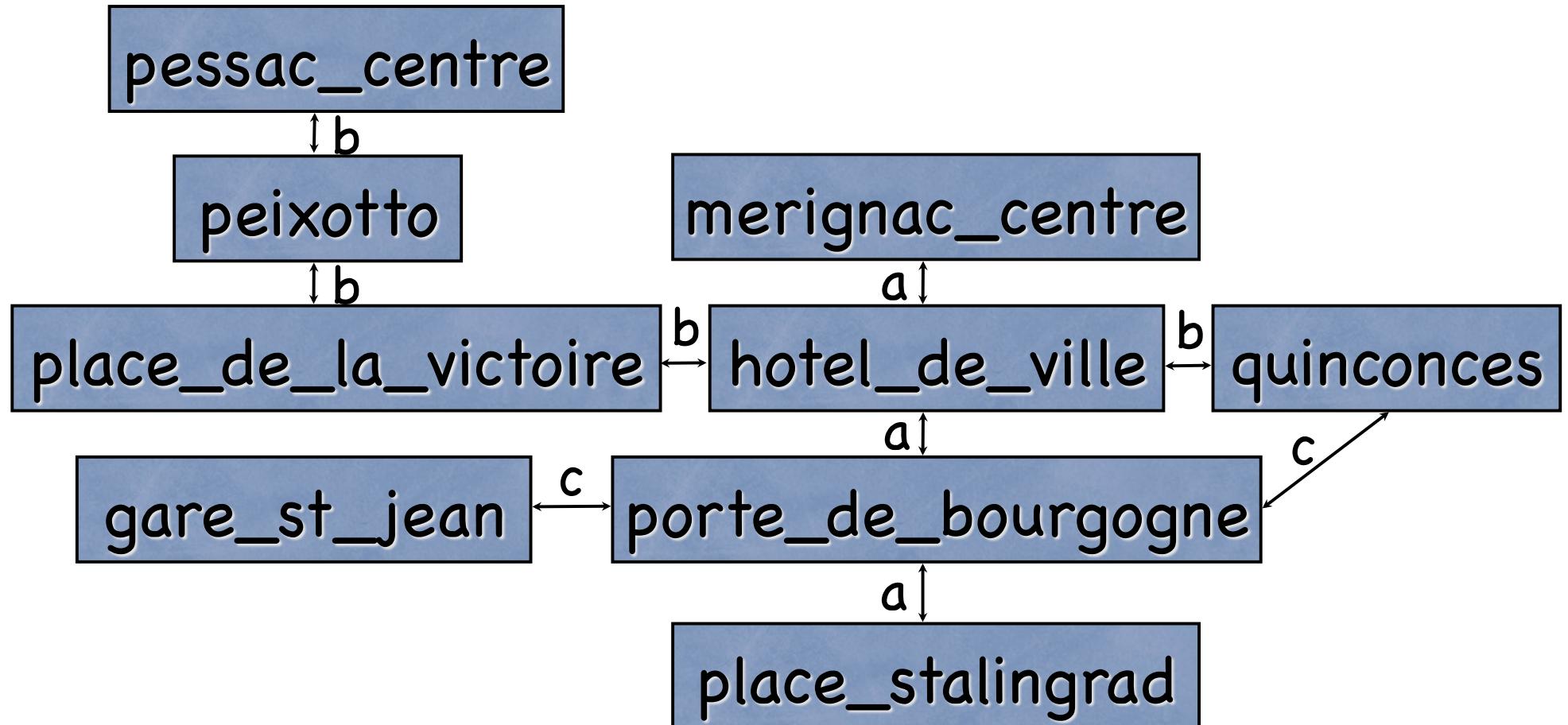
maximum([], Max, Max). Max = M = 3

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

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).
```

Quel est le problème
avec ce programme ?

```
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, 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).  
...
```