Cours 3: Récurrence

- Objectifs:
  - introduction de définitions récurrentes en Prolog
  - exemples
  - différences entre le sens déclaratif et procédural d'un programme Prolog: ordonnancement des règles et buts!
- Exercices
  - Exercices LPN chapitre 3
  - TP

Révision du cours 2:

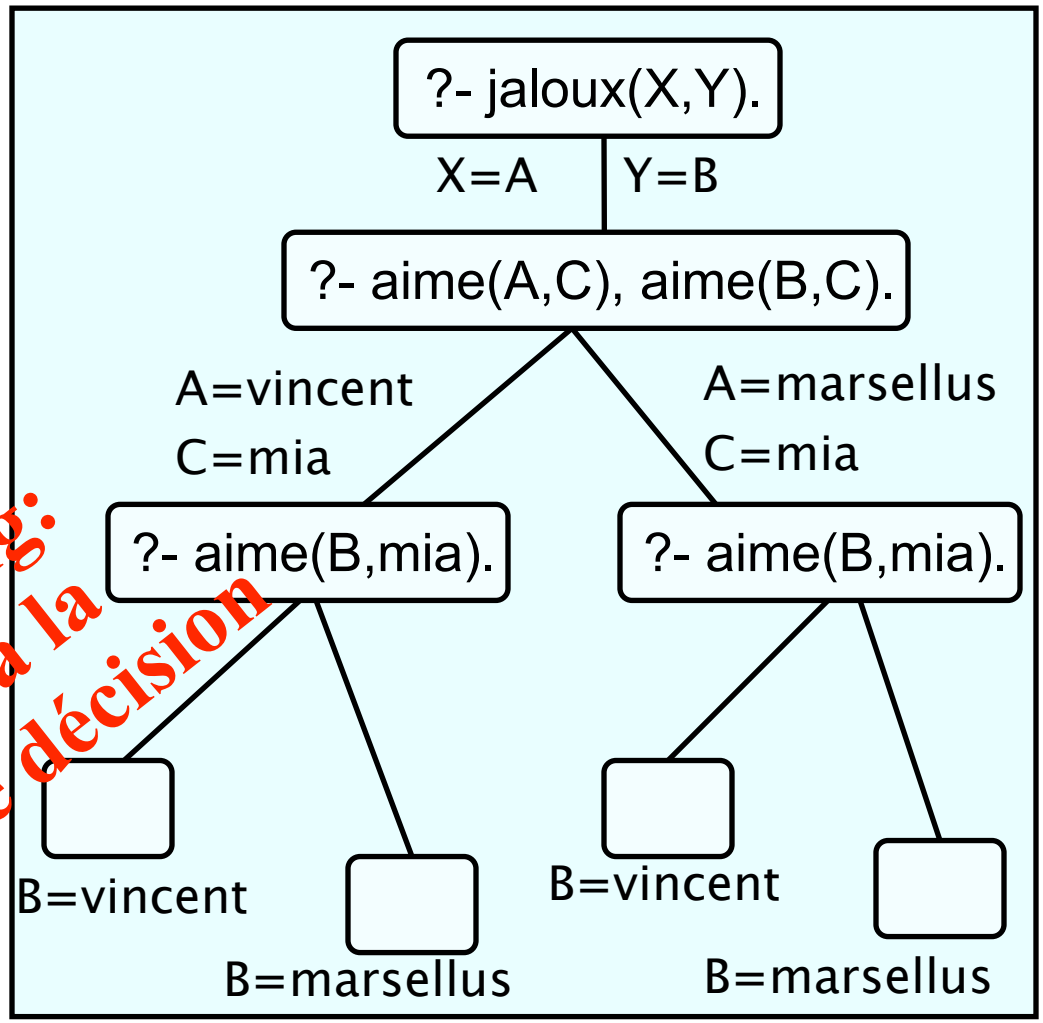
- unification de termes
- arbres de résolution
- parcours de construction de l'arbre

Exemple: énumération jalouse

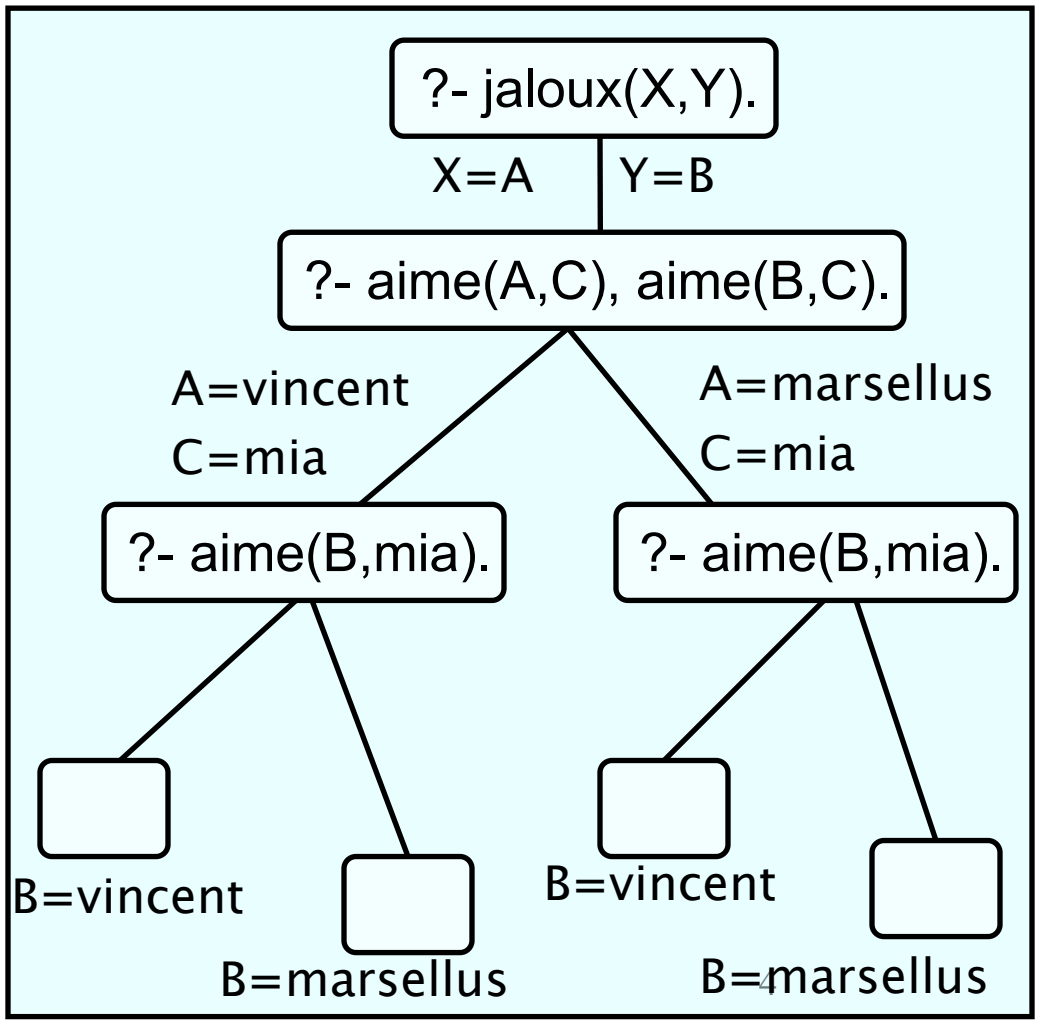
```
aime(vincent,mia).
aime(marsellus,mia).

jaloux(A,B):-
    aime(A,C),
    aime(B,C).
```

```
X=vincent
Y=vincent;
X=vincent
Y=marsellus;
X=marsellus
Y=vincent;
X=marsellus
Y=marsellus;
fail
```

backtracking: remonter à la dernière décision

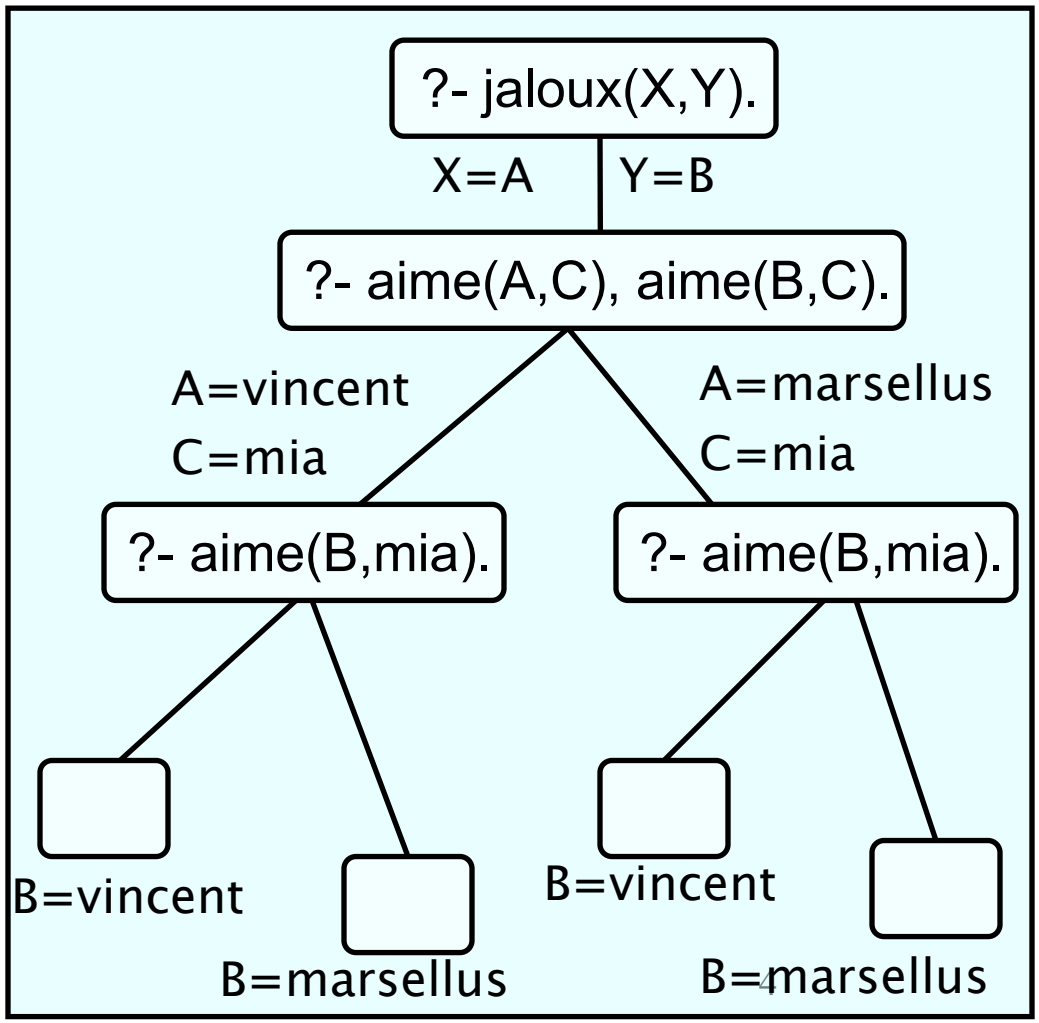


Parcours de construction de l'arbre?



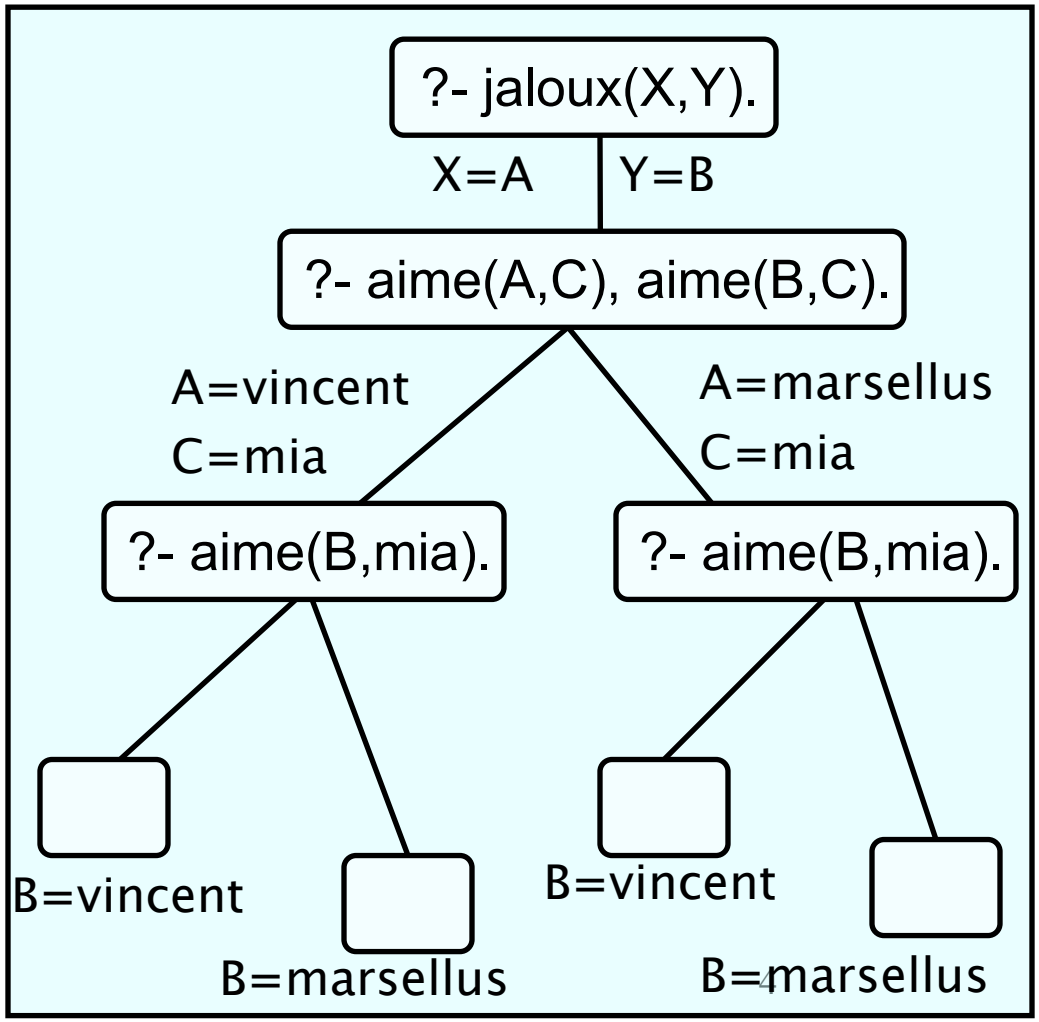
Parcours de construction de l'arbre?

- SLD resolution



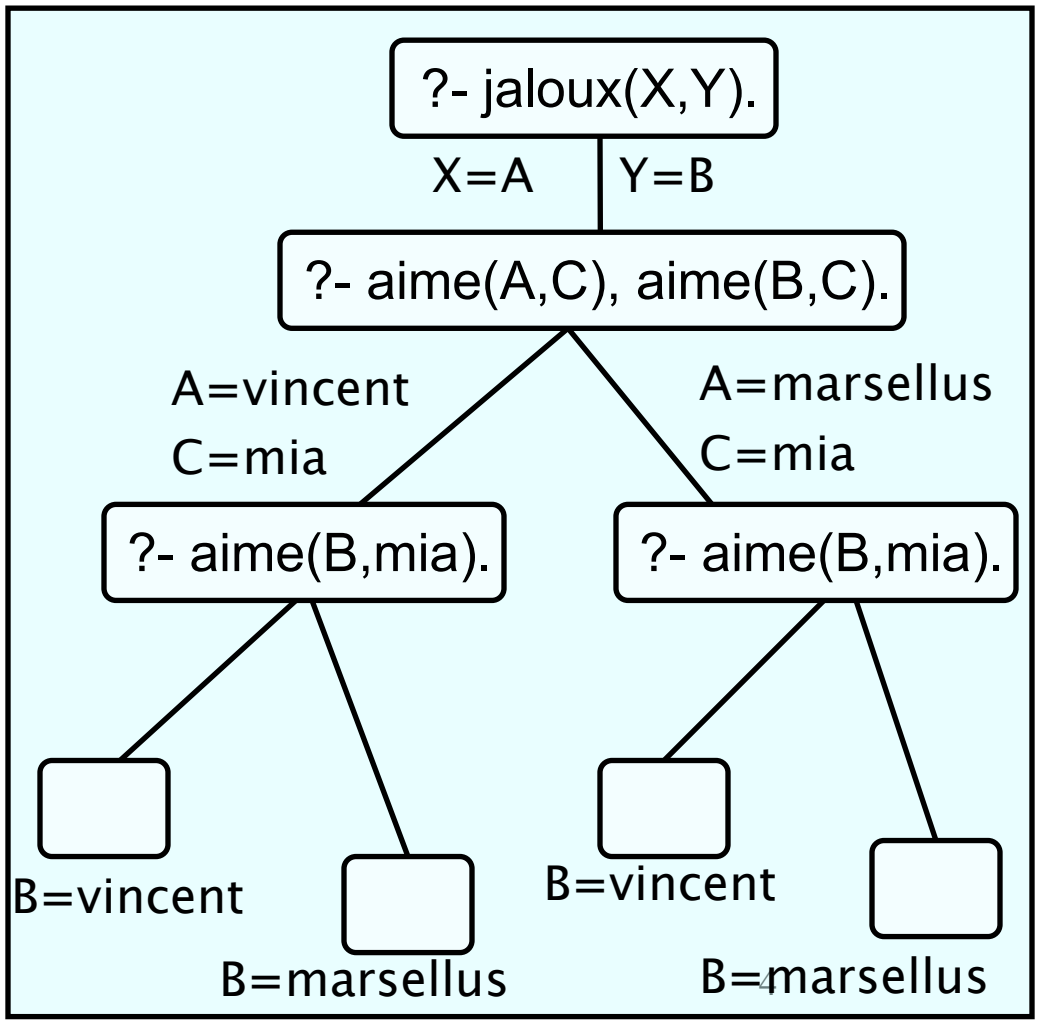
Parcours de construction de l'arbre?

- SLD resolution
  - Standard



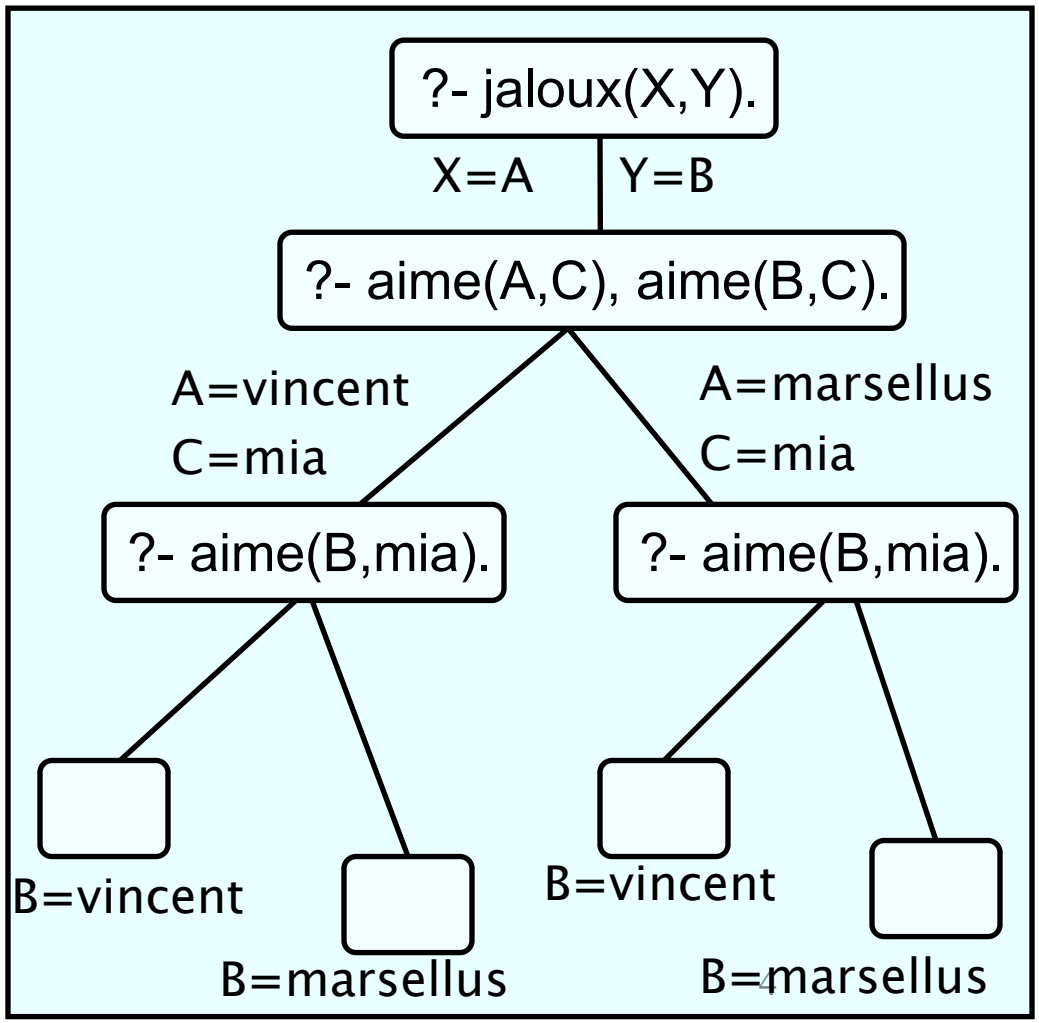
Parcours de construction de l'arbre?

- SLD resolution
  - Standard
  - LeftMost



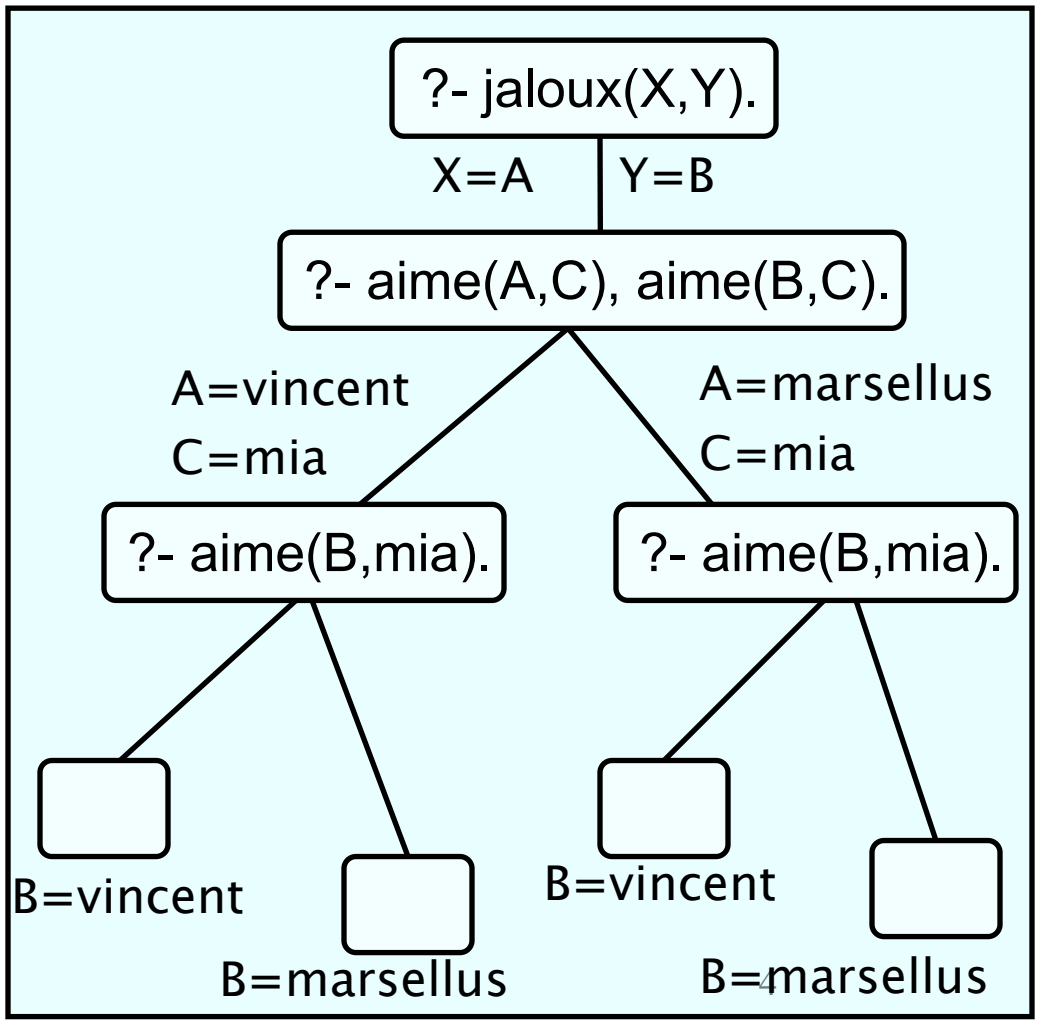
Parcours de construction de l'arbre?

- SLD resolution
  - Standard
  - LeftMost
  - DepthFirst



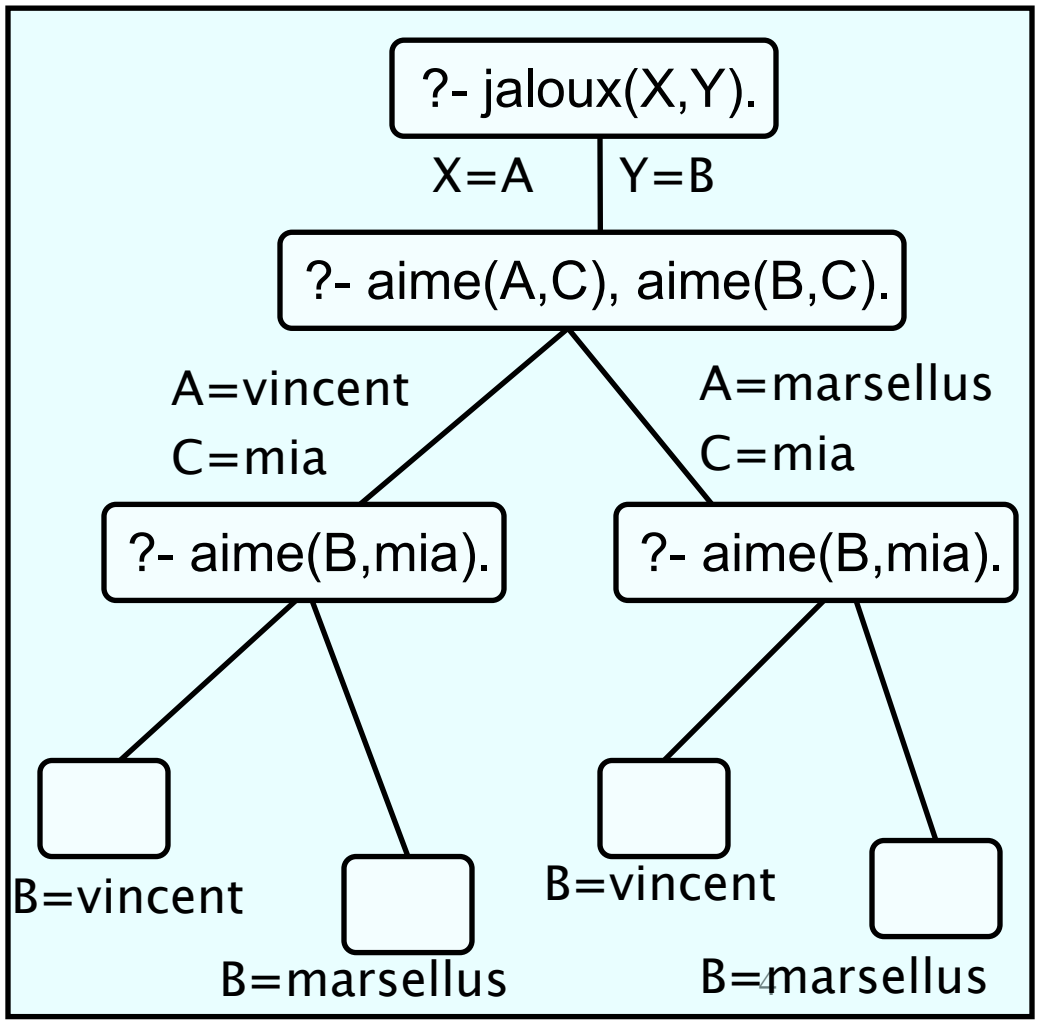
Parcours de construction de l'arbre?

- SLD resolution
  - Standard
  - LeftMost
  - DepthFirst



Parcours de construction de l'arbre?

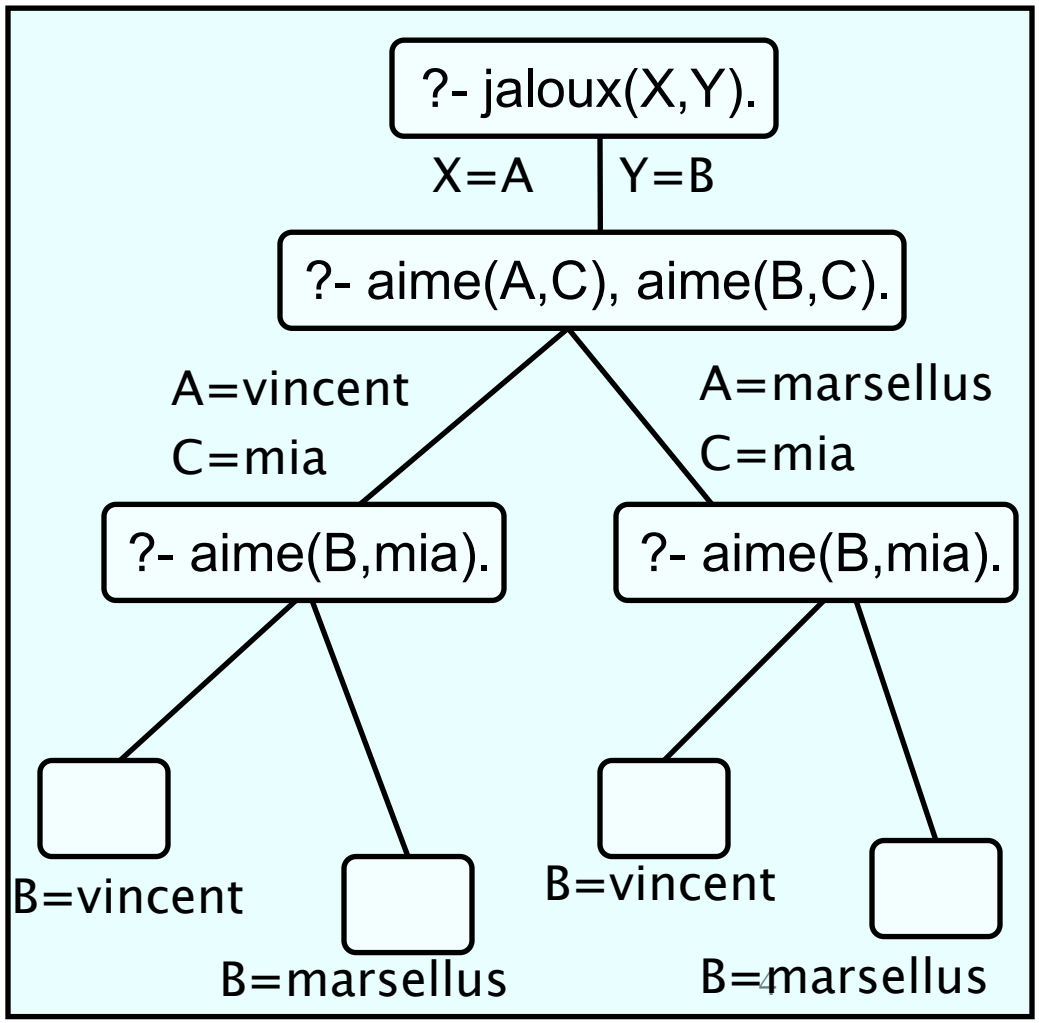
- SLD resolution
  - Standard
  - LeftMost
  - DepthFirst
- le plus à gauche



Parcours de construction de l'arbre?

- SLD resolution
  - Standard
  - LeftMost
  - DepthFirst

- le plus à gauche
- en profondeur d'abord



Définitions récurrentes

- Les prédicats de Prolog peuvent être définis de manière récurrente ou récursive
- Un prédicat est défini de manière récurrent si une ou plusieurs règles de sa définition font référence à ce même prédicat.

Exemple 1: manger & digérer

```
en_pleine_digestion(X,Y) :- vient_de_manger(X,Y).
en_pleine_digestion(X,Y) :-
        vient_de_manger(X,Z),    en_pleine_digestion(Z,Y).
vient_de_manger(moustique,sang(john)).
vient_de_manger(grenouille,moustique).
vient_de_manger(cigogne,grenouille).
```

```
?-
```

Illustration: manger & digérer

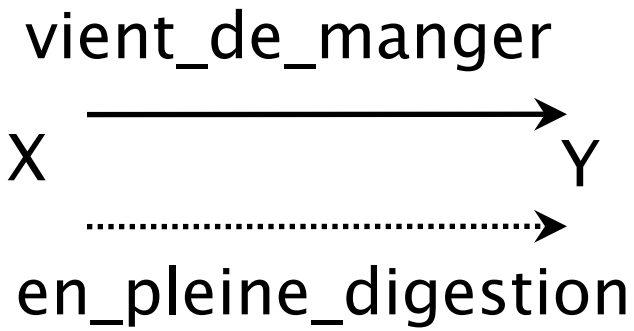
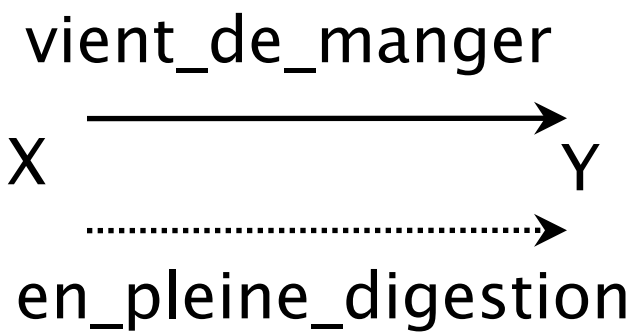
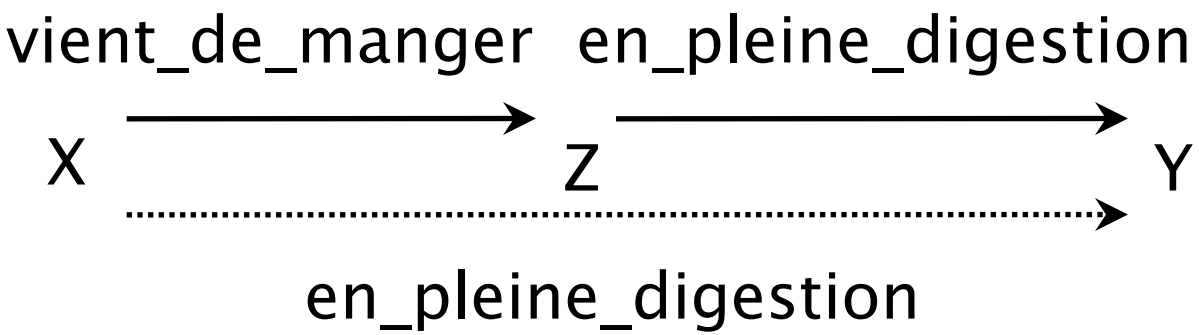


Illustration: manger & digérer





Exemple 1: manger & digérer

```
en_pleine_digestion(X,Y) :- vient_de_manger(X,Y).
en_pleine_digestion(X,Y) :-
      vient_de_manger(X,Z),     en_pleine_digestion(Z,Y).
vient_de_manger(moustique,sang(john)).
vient_de_manger(grenouille,moustique).
vient_de_manger(cigogne,grenouille).
```

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

Une autre définition récurrente

```
p :- p.
```

```
?-
```

Une autre définition récurrente

```
p:- p.
```

```
?- p.
```

Une autre définition récurrente

```
p:- p.
```

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

Exemple 2: Généalogie

```
enfant_de(bridget,caroline).  % Caroline est l'enfant de Bridget
enfant_de(caroline,donna).    % Donna est l'enfant de Caroline


descendant_de(X,Y):- enfant_de(X,Y).
descendant_de(X,Y):- enfant_de(X,Z), enfant_de(Z,Y).
```

Exemple 2: Généalogie

```
enfant_de(anne,bridget).
enfant_de(bridget,caroline).
enfant_de(caroline,donna).
enfant_de(donna,emily).



descendant_de(X,Y):- enfant_de(X,Y).
descendant_de(X,Y):- enfant_de(X,Z), enfant_de(Z,Y).
```

Exemple 2: Généalogie

```
enfant_de(anne,bridget).
enfant_de(bridget,caroline).
enfant_de(caroline,donna).
enfant_de(donna,emily).

descendant_de(X,Y):- enfant_de(X,Y).
descendant_de(X,Y):- enfant_de(X,Z), enfant_de(Z,Y).
```

```
?- descendant_de(anne,donna).
fail
?-
```

Exemple 2: Généalogie

```
enfant_de(anne,bridget).
enfant_de(bridget,caroline).
enfant_de(caroline,donna).
enfant_de(donna,emily).


descendant_de(X,Y):- enfant_de(X,Y).
descendant_de(X,Y):- enfant_de(X,Z), enfant_de(Z,Y).
descendant_de(X,Y):- enfant_de(X,Z), enfant_de(Z,U),
    enfant_de(U,Y).
```

```
?-
```

Exemple 2: Généalogie

```
enfant_de(anne,bridget).
enfant_de(bridget,caroline).
enfant_de(caroline,donna).
enfant_de(donna,emily).


descendant_de(X,Y):- enfant_de(X,Y).
descendant_de(X,Y):- enfant_de(X,Z), descendant_de(Z,Y).
```

```
?-
```

Exemple 2: Généalogie

```
enfant_de(anne,bridget).
enfant_de(bridget,caroline).
enfant_de(caroline,donna).
enfant_de(donna,emily).


descendant_de(X,Y):- enfant_de(X,Y).
descendant_de(X,Y):- enfant_de(X,Z), descendant_de(Z,Y).
```

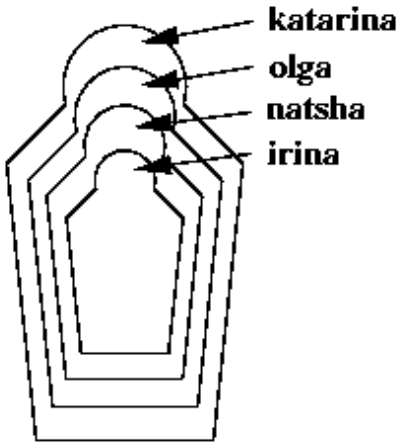
```
?- descendant_de(anne,donna).
```

Arbre de résolution

- Développez l´arbre de résolution pour

  ?- descendant_de(anne,donna).

Exercice: matriochkas



Connaissez-vous ces poupées russes (les matriochkas), dont les plus grosses contiennent des poupées plus petites?

D'abord, commencez la base de connaissances en utilisant le prédicat **directementDans/2.** Il traduit le fait que chaque poupée est directement contenue dans une autre poupée: directementDans(olga,katarina), directementDans(natsha,olga),...

Ensuite, définissez un prédicat récursif **dans/2** qui nous dit quelles poupées sont (directement ou non) dans les autres poupées. La requête *dans(natasha,katarina)* doit être considérée comme vraie, alors que la requête *dans(katarina,olga)* doit échouer.

Exemple 3: Successeur

- Nous utilisons la notation suivante pour les entiers:
  1. **0** est un entier.
  2. Si **X** est un entier, **succ(X)** en est aussi un.

Exemple 3: Successeur

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

Exemple 3: Successeur

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

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

Exemple 3: Successeur

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

```
?- entier(X).
```

Exemple 3: Successeur

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

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

Arbre de résolution

- Dessinez l´arbre de résolution pour l'énumération des entiers

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

Exo: superieur/2

- Définissez un prédicat superieur/2 qui prend en arguments deux nombres dans la notation vue précédemment (0,succ(0),succ(succ(0))) et qui détermine si le premier est plus grand que le second. Ainsi:

```
?-superieur(succ(succ(succ(0))),succ(0)).
yes
?-superieur(succ(succ(0)),succ(succ(succ(0)))).
fail
```

Supérieur

- pour tous les entiers Y, le successeur d'Y est supérieur à zéro
- si A>B alors A+1 > B+1

Supérieur

- pour tous les entiers Y, le successeur d'Y est supérieur à zéro
- si A>B alors A+1 > B+1

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

Supérieur

- pour tous les entiers Y, le successeur d'Y est supérieur à zéro
- si A>B alors A+1 > B+1

Supérieur

- pour tous les entiers Y, le successeur d'Y est supérieur à zéro
- si A>B alors A+1 > B+1

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


superieur(succ(Y),0) :- entier(Y).
superieur(succ(A),succ(B)) :- superieur(A,B).
```

Prolog et la logique

- Prolog était la première tentative raisonnable de créer un langage de programmation logique
  - Le programmateur spécifie le problème d'une manière déclarative, basée sur le langage de la logique
  - Le programmateur ne doit pas se soucier d'expliquer à l'ordinateur quoi faire
  - Pour obtenir de l'information, le programmateur pose simplement des questions

Prolog et la logique

- Prolog n'est pas *strictement* un langage de programmation logique!
- En programmant, il faut tenir compte de comment Prolog répond aux requêtes:
  - traverse la base de connaissances de haut en bas
  - traite les clauses de gauche à droite
  - fait marche arrière pour réparer des mauvais choix (*backtracking*)

genealogie1.pl

```
enfant_de(anne,bridget).
enfant_de(bridget,caroline).
enfant_de(caroline,donna).
enfant_de(donna,emily).

descendant_de(X,Y):- enfant_de(X,Y).
descendant_de(X,Y):- enfant_de(X,Z), descendant_de(Z,Y).
```

```
?- descendant_de(A,B).
A=anne
B=bridget
```

genealogie2.pl

```
enfant_de(anne,bridget).
enfant_de(bridget,caroline).
enfant_de(caroline,donna).
enfant_de(donna,emily).

descendant_de(X,Y):- enfant_de(X,Z), descendant_de(Z,Y).
descendant_de(X,Y):- enfant_de(X,Y).
```

changement: **ordre des règles.**

règle de base après règle récursive

genealogie2.pl

```
enfant_de(anne,bridget).
enfant_de(bridget,caroline).
enfant_de(caroline,donna).
enfant_de(donna,emily).

descendant_de(X,Y):- enfant_de(X,Z), descendant_de(Z,Y).
descendant_de(X,Y):- enfant_de(X,Y).
```

```
?- descendant_de(A,B).
A=anne
B=emily
```

la première solution de genealogie1.pl était:
X=anne, Y=bridget

Questions

- résultats corrects?
- ordre d'énumeration de tout les couples de descendants
- forme de l'arbre de résolution
- éfficacité par rapport à la version initiale

genealogie3.pl

```
enfant_de(anne,bridget).
enfant_de(bridget,caroline).
enfant_de(caroline,donna).
enfant_de(donna,emily).

descendant_de(X,Y):- descendant_de(Z,Y), enfant_de(X,Z).
descendant_de(X,Y):- enfant_de(X,Y).
```

racine de tous les maux quand il est question de non-terminaison: **récursivité à gauche**

genealogie3.pl

```
enfant_de(anne,bridget).
enfant_de(bridget,caroline).
enfant_de(caroline,donna).
enfant_de(donna,emily).

descendant_de(X,Y):- descendant_de(Z,Y), enfant_de(X,Z).
descendant_de(X,Y):- enfant_de(X,Y).
```

```
?- descendant_de(anne,bridget).
ERROR!!! (out of stack)
```

genealogie4.pl

```
enfant_de(anne,bridget).
enfant_de(bridget,caroline).
enfant_de(caroline,donna).
enfant_de(donna,emily).

descendant_de(X,Y):- enfant_de(X,Y).
descendant_de(X,Y):- descendant_de(Z,Y), enfant_de(X,Z).
```

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

ne bugue pas pour l'exemple précédent, mais...

Règle empirique

- Dans vos définitions d'un prédicat récursif,
  - donnez le cas de base. Puis,
  - dans la règle récurrente, placez les buts qui déclenchent les appels récursifs aussi loin que possible vers la droite du corps de la règle!

Résumé de ce cours

- Nous avons introduit les prédicats récurrents
- Nous avons observé les différences entre le sens déclaratif et procédural de programmes Prolog.
- Nous avons identifié certaines limitations de Prolog comme langage de programmation logique

Remarque technique

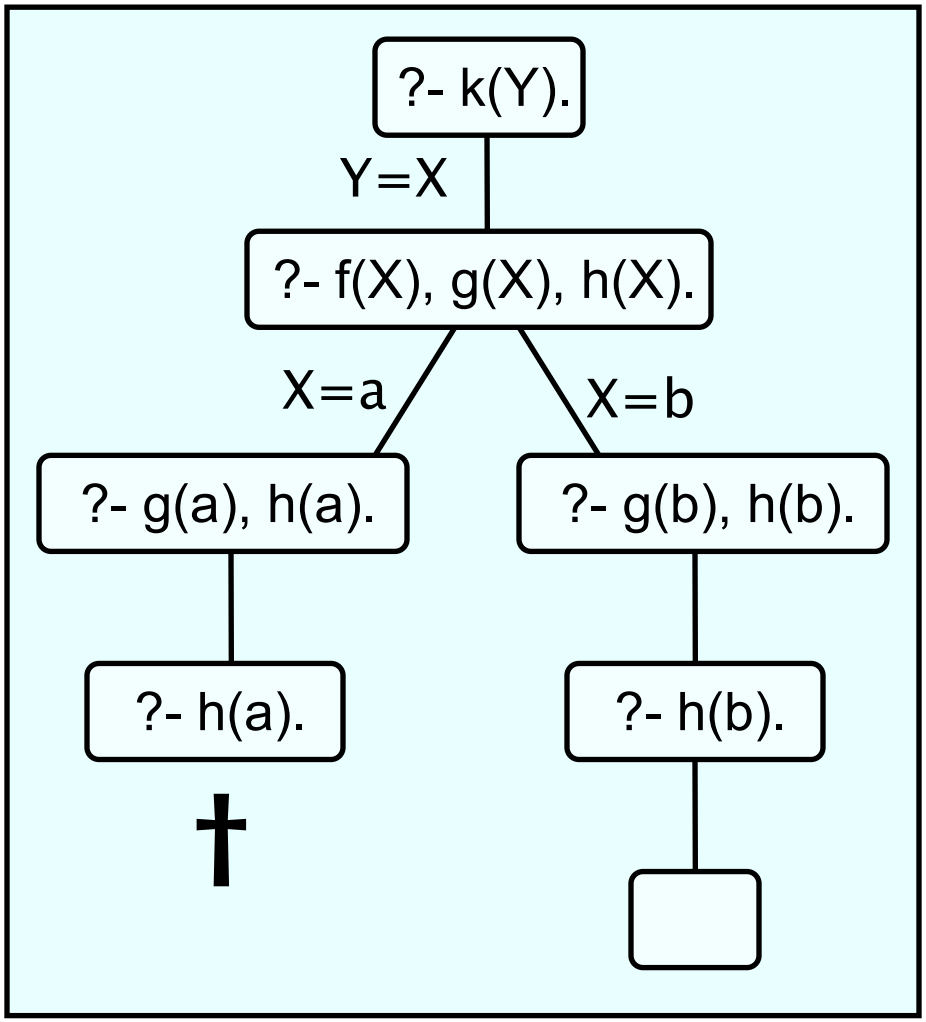
- Lorsque Prolog unifie la variable d'une requête avec une d'un fait ou d'une règle, il génère une variable interne pour représenter que ses variables partagent la même valeur.
- La requête initiale k(X) est devenue: k(_G348)

41

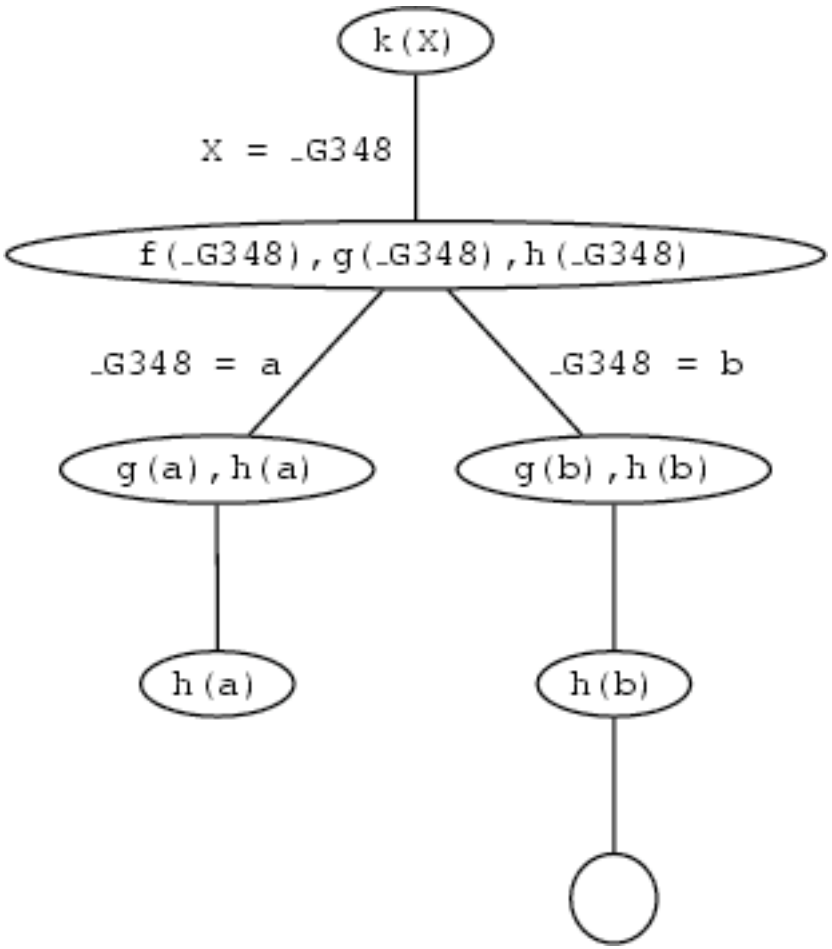
Exemple: arbre de résolution

```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

```
?- k(Y).
Y=b;
fail
?-
```



Arbre avec variables internes



Pourquoi s'en préoccuper?

- Variables internes importantes pour
  - requêtes avec variables
  - requêtes avec prédicats récursifs, quand une règle est appliquée plus d'une fois le long d'une branche de l'arbre de résolution.
- Pour simplification, nous pouvons négliger les variables internes en dessinant des arbres, mais il faut s'en souvenir en utilisant l'outil *trace* de Prolog.

44

Les Chtis

```
ville(dunkerque).
ville(bergues).
habite(michel,dunkerque).
habite(antoine,bergues).
habite(annebelle,bergues).
chti(X) :- ville(Y), habite(X,Y).
```

```
?- chti(antoine).
```

Exo: construisez les arbres de résolution!

```
?- chti(Qui).
```

Prochain cours

- Introduire le traitement de **listes** en Prolog
  - structure de données importante en programmation Prolog
  - Définition du prédicat member/2, un outil fondamental pour travailler avec les listes en Prolog
  - Actions récursives sur des listes.