Unification in prologe

But de cette Séance

Le but de cette séance est de traiter de l'**unification** en Prolog et de montrer comment elle diffère de l'unification standard.

Unification

Rappel : Termes de Prolog

Les termes de Prolog peuvent être simples ou complexes, incluant des constantes, des variables, des atomes et des nombres.

> Deux termes sont unifiés s'ils sont le même terme, ou s'ils contiennent des variables qui peuvent uniformément s'instancier avec des termes de telle manière que les termes résultants sont égaux.

Ceci signifie également que :

- `woman(mia)` et `woman(jody)` ne s'unifient pas.
- `vincent` et `mia` ne s'unifient pas.
- `mia` et `mia` s'unifient.
- `42` et `42` s'unifient.
- `woman(mia)` et `woman(mia)` s'unifient.

Instanciations

Quand Prolog unifie deux termes, il effectue toutes les instanciations nécessaires afin de rendre les termes égaux. Ceci fait de l'unification un mécanisme de programmation puissant.

Définition Révisée de l'Unification

1. Si T1 et T2 sont des **constantes**, alors T1 et T2 s'unifient s'ils sont le même **atome** ou le même **nombre**.
2. Si T1 est une **variable** et T2 est n'importe quel type de terme, alors T1 et T2 s'unifient, et T1 est instancié à T2 (et vice versa).
3. Si T1 et T2 sont des **termes complexes**, ils s'unifient si :
    - a) Ils ont les mêmes **foncteur** et **arité**, et
    - b) tous leurs arguments s'unifient, et
    - c) les instanciations de variables sont compatibles.

Unification en Prolog : =/2

?- mia = mia.

yes
?- mia = vincent.
no
?- mia = X.
X = mia
yes

Exemple Complexe

?- k(s(g), Y) = k(X, t(k)).
X = s(g)
Y = t(k)
yes
?- k(s(g), t(k)) = k(X, t(Y)).
X = s(g)
Y = k
yes

Termes Infinis

Prolog n'utilise pas un algorithme d'unification standard. Considérez la requête suivante:

?- pere(X) = X.

Programmation par Unification

vertical(ligne(point(X,Y), point(X,Z))).
horizontal(ligne(point(X,Y), point(Z,Y))).

Exemples d'utilisation:

?- vertical(ligne(point(1,1), point(1,3))).
yes

?- vertical(ligne(point(1,1), point(3,2))).
no

?- horizontal(ligne(point(1,1), point(1,Y))).
Y = 1 ;
no

?- horizontal(ligne(point(2,3), Point)).
Point = point(_554, 3) ;
no

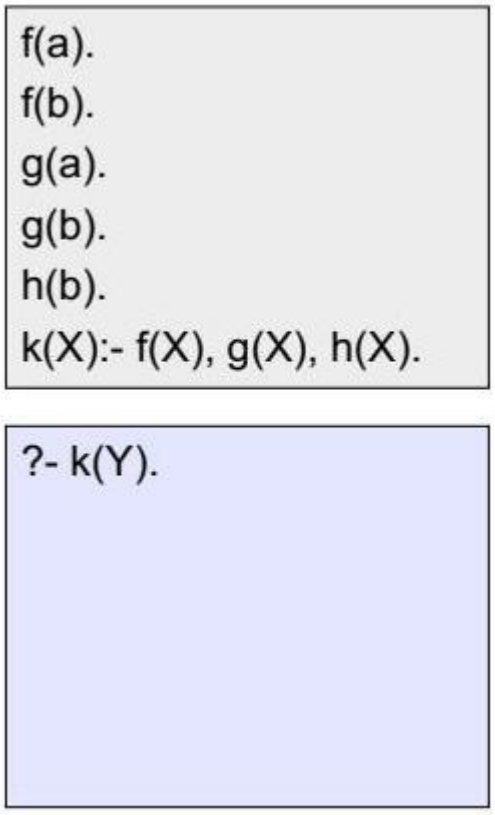
Recherche de Preuve

Maintenant que nous savons ce qu'est l'unification, nous pouvons comprendre comment Prolog recherche dans sa base de connaissances si une requête peut être satisfaite (et comment elle le peut).

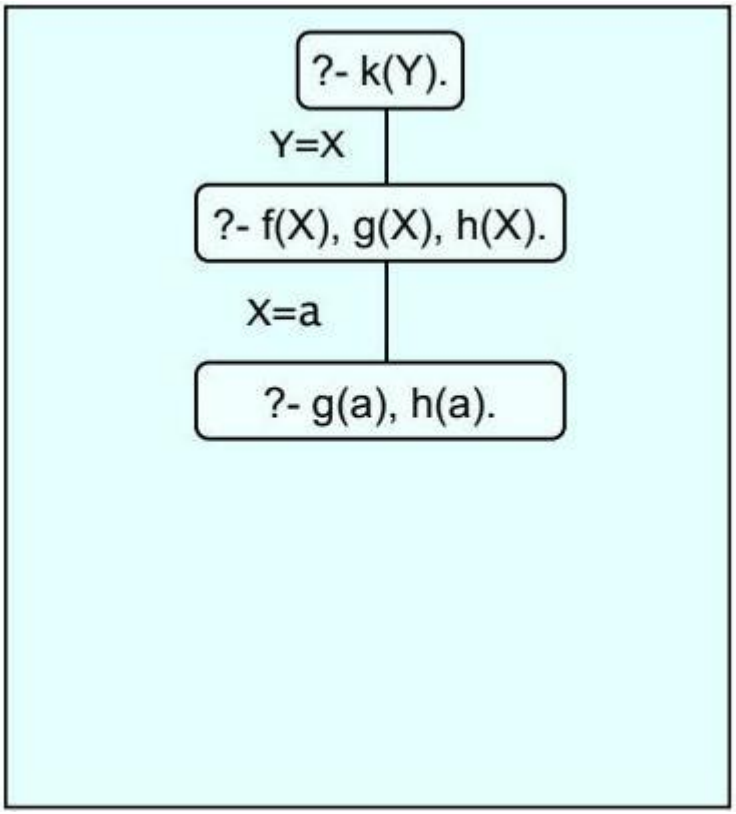
Exemple d'Arbre de Recherche

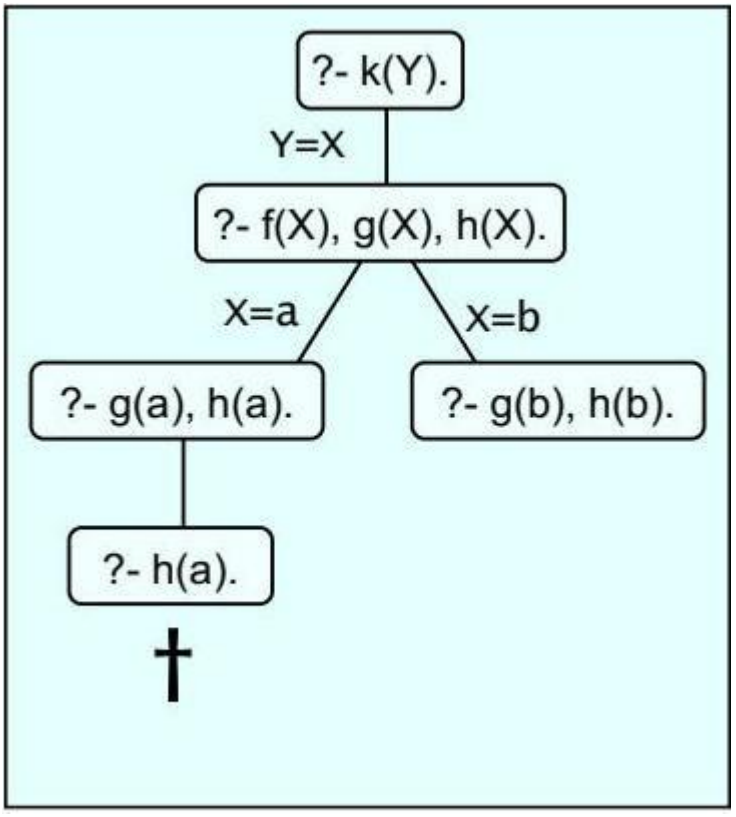
Voici un exemple pour illustrer la recherche de preuve :

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

?- k(Y).
```



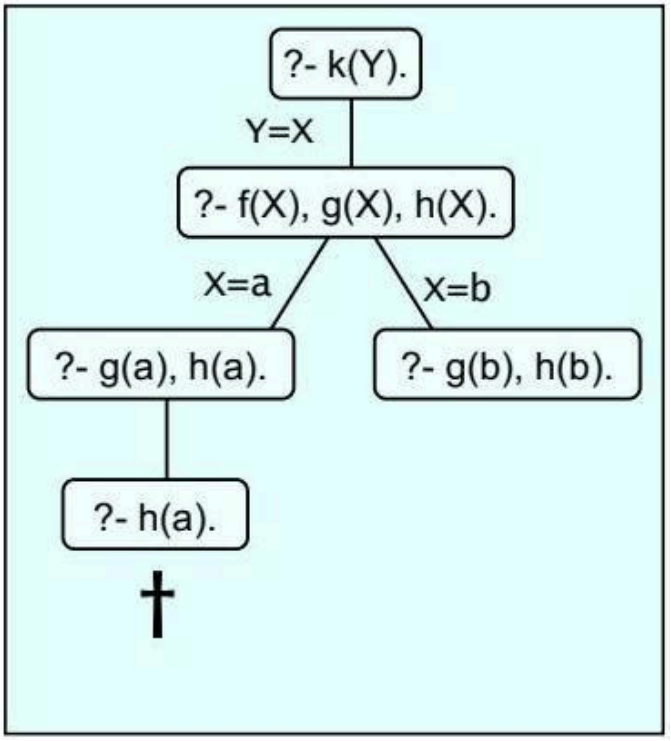
Prolog va essayer de satisfaire chaque but dans l'ordre, en utilisant l'unification pour trouver des valeurs pour les variables. Ci-dessous se trouve une illustration de la façon dont Prolog procède dans cet exemple :


?- k(Y).
Y=X
?- f(X), g(X), h(X).
X=a
?- g(a), h(a).


?- k(Y).
Y=X
?- f(X), g(X), h(X).
X=a
X=b
?- g(a), h(a).
?- g(b), h(b).
?- h(a).
†

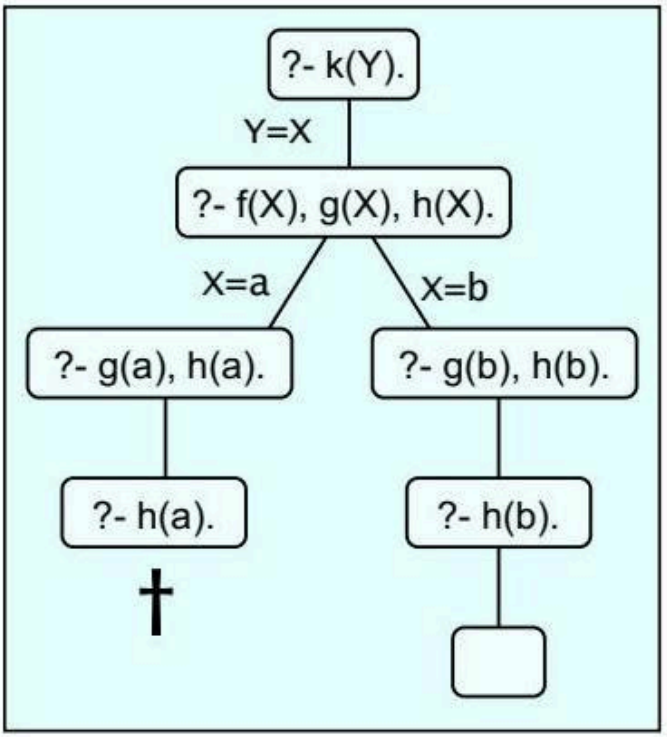
```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

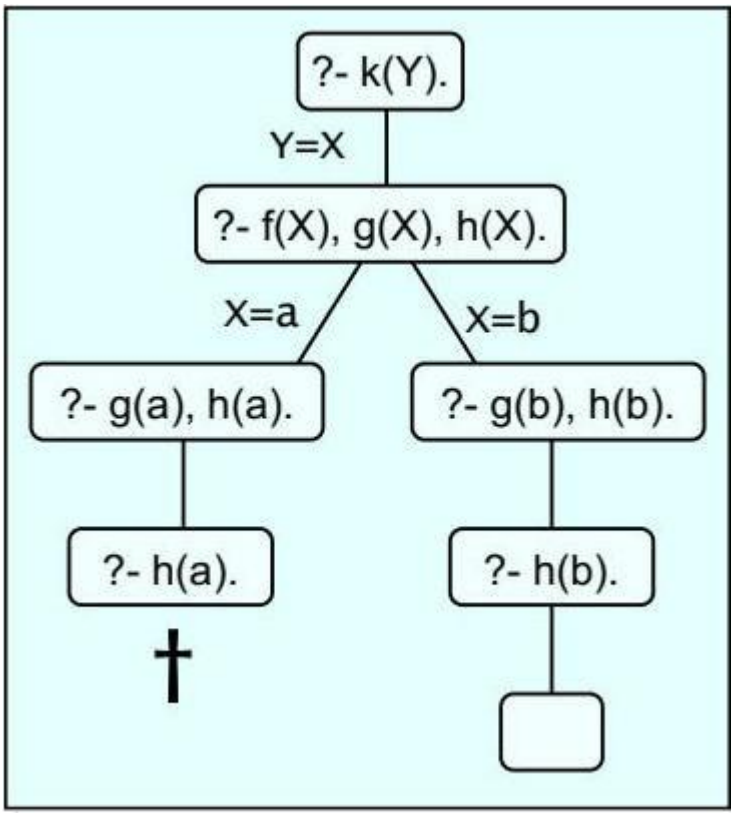
```
?- k(Y).
```



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

```
?- k(Y).
Y=b;
no
?-
```





Autre Exemple

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

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

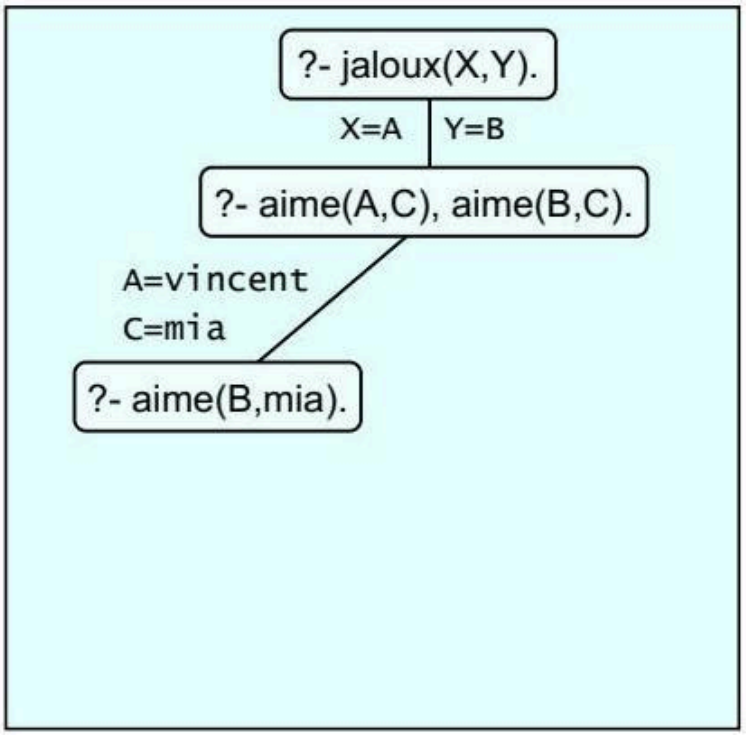
?- jaloux(X, Y).

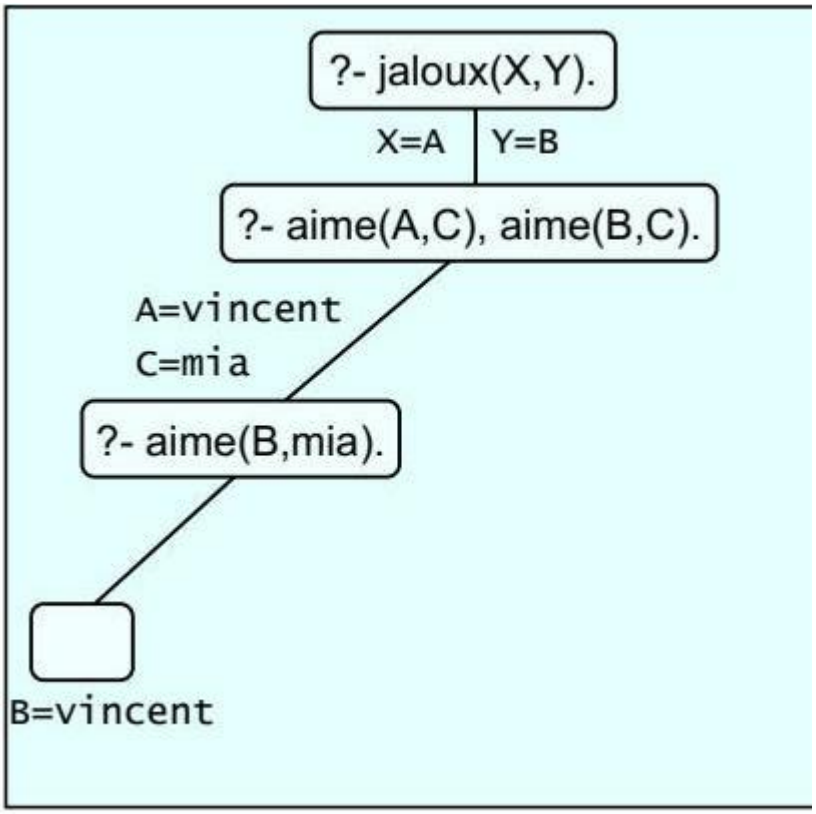
```
aime(vincent,mia).
aime(marsellus,mia).

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

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

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

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

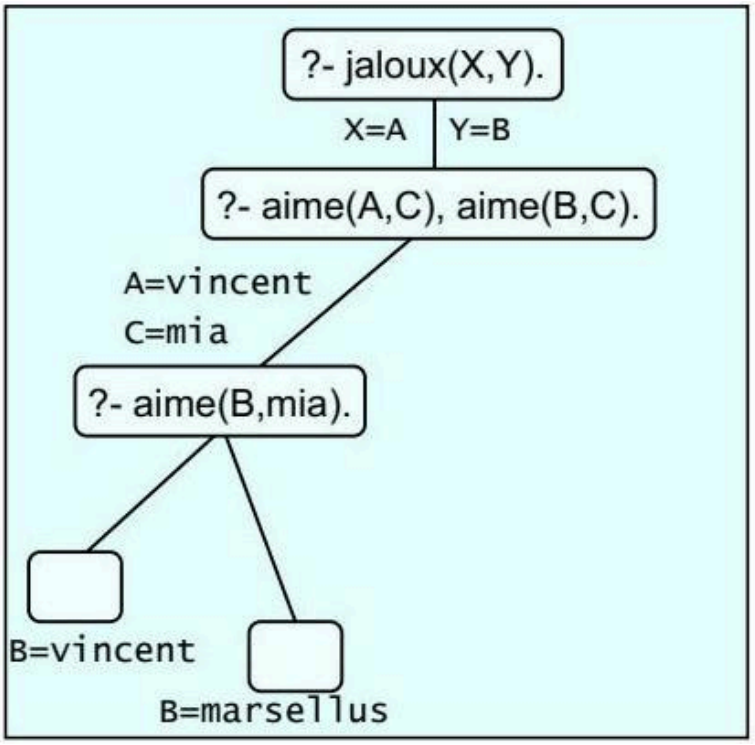
```
?- jaloux(X,Y).
```




?- jaloux(X,Y).
X=A
Y=B
?- aime(A,C), aime(B,C).
A=vincent
C=mia
?- aime(B,mia).
B=vincent

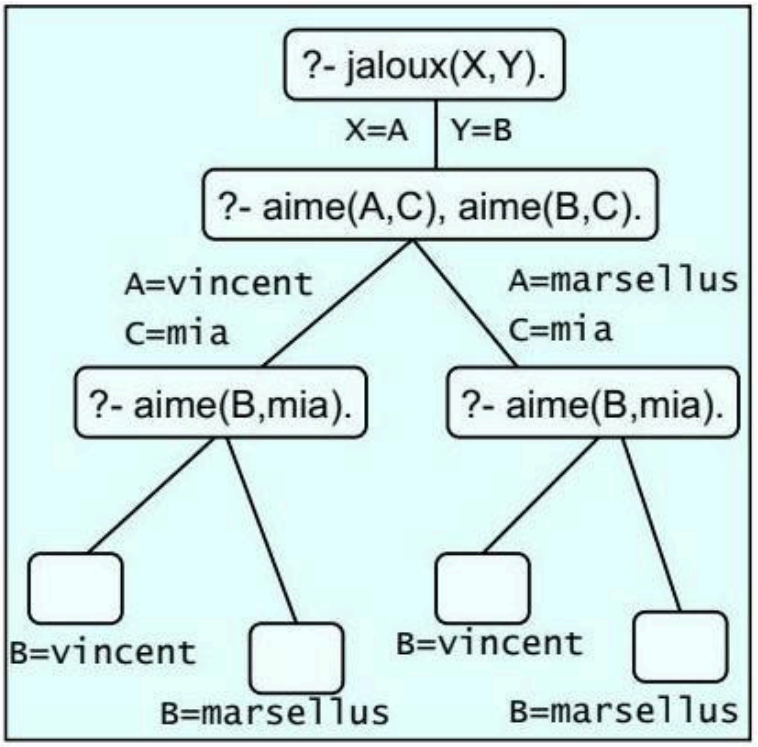
```
aime(vincent,mia).
aime(marsellus,mia).

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

```
?- jaloux(X,Y).
X=vincent
Y=vincent;
X=vincent
Y=marsellus
```




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

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

```
....
X=marsellus
Y=vincent;
X=marsellus
Y=marsellus;
no
```



Exercices

(Note: The transcript mentions exercises but doesn't provide specific details.)

Résumé de la Séance

- Défini ce qu'est l'**unification**.

- Vu la différence entre l'unification standard et celle de Prolog.
- Présenté les arbres de recherche.

Prochaine Séance

- La **récursivité** en Prolog.
- Les définitions récursives in Prolog.
- Les différences entre l'approche déclarative d'un programme en Prolog et une approche procédurale.

Références :

SWI-Prolog. (s. d.). https://www.swi-prolog.org/

Learn ProLog Now ! (s. d.). https://lpn.swi-prolog.org/lpnpage.php?pageid=top

PROLOG – Programmation Logique

Traduction : Yannick Estève
Cours dispensé à **CERI – Avignon Université**