Cours de Programmation Déclarative et Bases de Données

Prolog - unification

Nicolas Jouandeau

n@up8.edu

2022

unification

- procédé par lequel on donne des valeurs aux variables pour rendre deux formules identiques

exemples d'unification de formules à 2 et 3 arguments

```
?- f(B,C) = f(2,3).
B = 2,
C = 3.

?- f(X,Y,L) = f(1,2,a).
X = 1,
Y = 2,
L = a.

?- f(X,X,Y) = f(Y,u,v).
false.
```

unification (suite)

- notons l'unification entre accolades (i.e. "{" et "}").

exemples

```
?- f(B,C) = f(2,3).
{ B=2 , C=3 }

?- f(X,Y,L) = f(1,2,a).
{ X=1 , Y=2 , L=a }

?- f(X,X,Y) = f(Y,u,v).
{ X=Y , X=u , Y=v }
```

unification (suite)

- une règle est composée d'une tête et d'un corps
  `p(a) :- f(a), g(X,a).`
- un fait est une règle sans tête (en réalité avec une tête toujours vraie)
  `p(a).`
- l'unification est une substitution qui rend deux termes identiques
- la résolution d'un programme consiste à réaliser des unifications jusqu'à obtenir des valeurs vraies ou fausses
- la résolution d'un prédicat (nommé but)
  - identification de la règle correspondant au but
  - unification entre le but et la tête
  - propagation de l'unification dans les prédicats du corps
  - résolution de ces prédicats (qui sont de nouveaux buts)

exemple

```
pour savoir si le but f(a) est vrai
on trouve la règle f(X) :- g(X), h(X,_)
on obtient l'unification { X=a }
on obtient les sous-buts g(a), h(a,_)
```

unification (suite)

- le résultat de l'unification n'est pas unique
  - pour l'unification de `f(a)` avec `f(X)`
  - `{ X=a }` est possible
  - `{ X=Z, Z=a }` est également possible
  - cette deuxième unification est moins générale que la première
- il existe une unification générale dont toutes les autres unifications sont des cas particuliers
- l'unification peut échouer

exemple

```
?- f(2,3) = f(X,X).
    { 2=X , 3=X }
false.
```

résolution ordonnée des clauses : l'ordre est important

- pour un but `p(a),q(a)` (composé de plusieurs clauses)
  - `p(a)` est la première clause
  - `q(a)` est la deuxième clause
  - `p(a)` est évaluée avant `q(a)`

principe du démonstrateur Prolog

```
pour chacune des clauses c à résoudre
  pour chacune des règles r correspondant à c
    unifier r et c
    si succès
      si la clause courante est un fait
        passer à la clause suivante
      sinon
        démontrer la clause courante
    si échec, alors passer à la règle suivante
    si aucune règle ne correspond à c
      revoir l'unification précédente (backtrack)
    si chaque clause est unifiée avec succès
      si on appuie sur . ou ENTREE
        on met fin à la résolution
      si on appuie sur ;
        on poursuit la résolution
```

démontrer une clause

```
si la clause est un fait
  on retourne un succès
si la clause est une règle
  on remplace la tête par le corps de la règle
  on substitue les valeurs du corps par celles de la tête
  (on renomme des variables si besoin)
  on appelle le démontrateur sur les clauses du corps
sinon
  on retourne un échec
```

sémantique des déclarations

- pour des faits ou règles de même nom, plusieurs écritures sont possibles

faits de même nom

```
p(a).
p(b).
```

réécriture non autorisée

```
p(a);p(b).
```

réécriture autorisée

```
p(X):-X=a;X=b.
```

règles de même nom

```
h(X,Y):-f(X),g(Y).
h(X,Y):-f(X),h(Y).
```

réécriture autorisée

```
h(X,Y):-f(X),g(Y);f(X),h(Y).
```

exemple de résolution d'un but

- programme

```
p(a).
q(c).
```

- résolution de `p(X),q(X)`

```
?- p(X),q(X).        % but
   |
  { X=a }            % unification
  ?- q(a).           % nouveau but
     |
    { a=c }          % unification
    faux             % résultat
```

but avec plusieurs solutions : point de choix

- quand le corps d'une règle est une disjoinction de clauses (ou encore quand une clause correspond à différents faits)
- programme

```
p(a).
p(c).
```

- résolution de `p(X)`

```
?- p(X).
   |_________________
   |                 |
 { X=a }           { X=c }
 vrai              vrai
```

arbres et résolutions

- programme

```
p(a).                    r1(X):-p(X), q(1,X).
p(b).                    r2(X):-q(1,X), p(X).
p(c).
q(1,a).
```

- entre `r1` et `r2`, seul l'ordre change
- résolution de `r1(c)`

```
?- r1(c).
   |
  { X=c }
  ?- p(c), q(1,c).
     |_____________________________
     |              |               |
   { a=c }       { b=c }        { c=c }
   faux          faux           ?- q(1,c).
                                    |
                                  { 1=1 , a=c }
                                  faux
```

- la réponse est FAUX
- la résolution `r1(c)` demande 5 unifications

arbres et résolutions (suite)

- programme

```
p(a).                    r1(X):-p(X), q(1,X).
p(b).                    r2(X):-q(1,X), p(X).
p(c).
q(1,a).
```

- résolution de `r2(c)`

```
?- r2(c).
    |
  { X=c }
  ?- q(1,c), p(c).
      |
    { 1=1 , a=c }
    faux
```

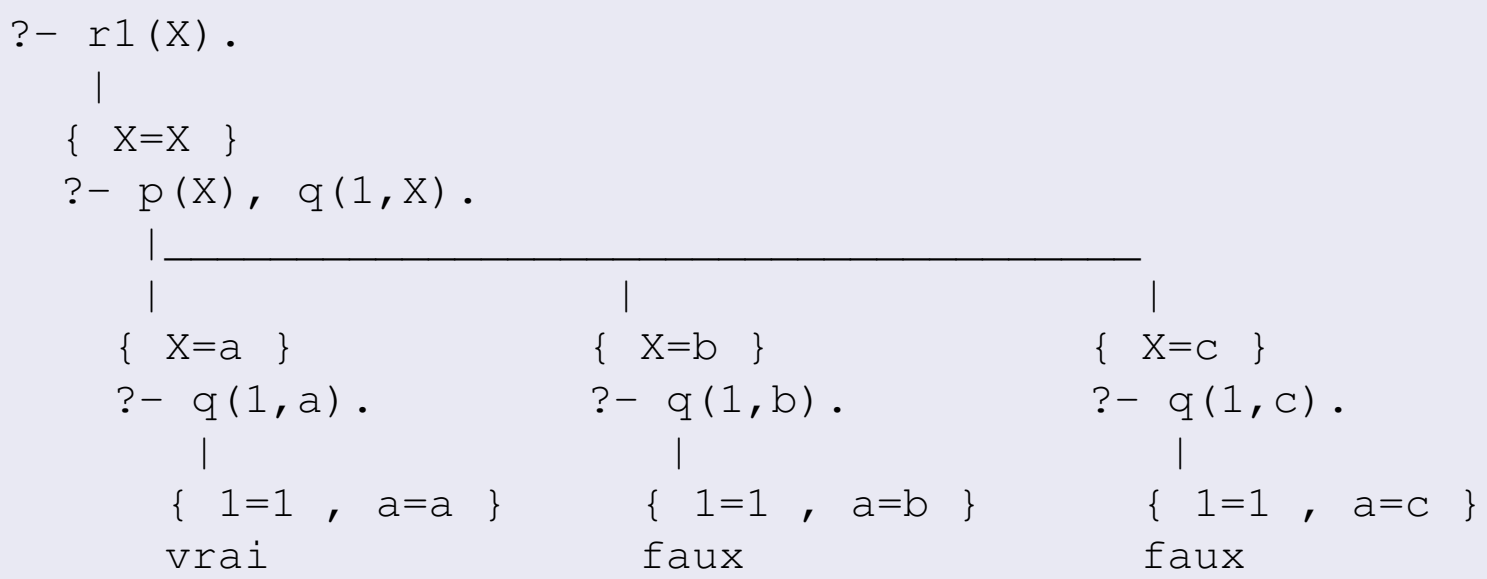
- la réponse est FAUX
- la résolution `r2(c)` demande 2 unifications
- l'ordre des clauses influe sur la longueur de la résolution

arbres et résolutions (suite)

- programme

```
p(a).                    r1(X):-p(X), q(1,X).
p(b).                    r2(X):-q(1,X), p(X).
p(c).
q(1,a).
```

- résolution de `r1(X)`



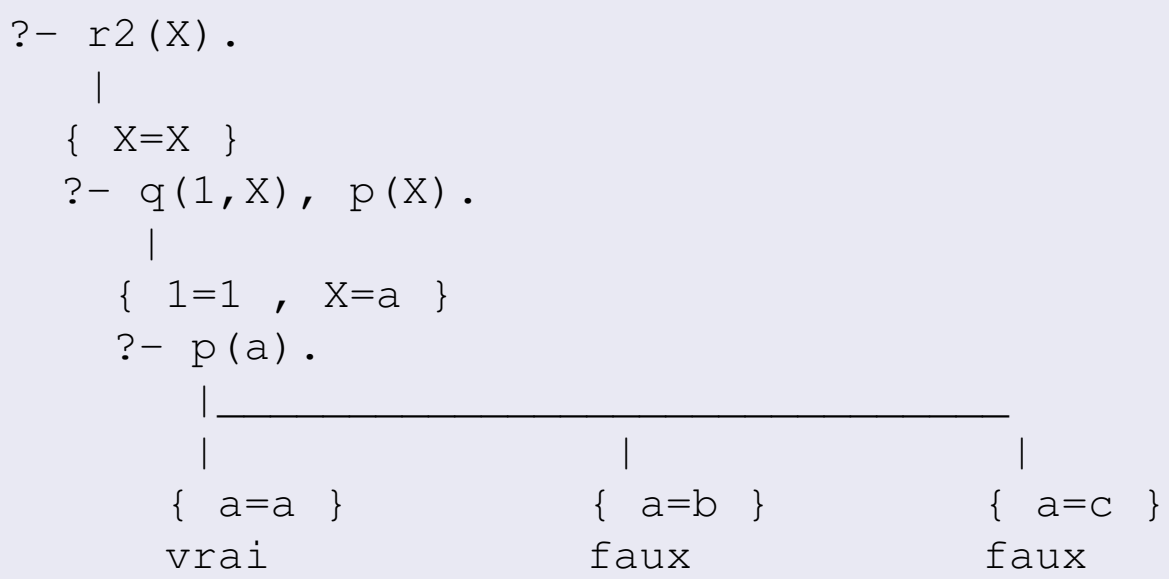
- la réponse est `vrai { X=a}`
- la première réponse demande 3 unifications
- la deuxième réponse demande 7 unifications

arbres et résolutions (fin)

- programme

```
p(a).                    r1(X):-p(X), q(1,X).
p(b).                    r2(X):-q(1,X), p(X).
p(c).
q(1,a).
```

- résolution de `r2(X)`



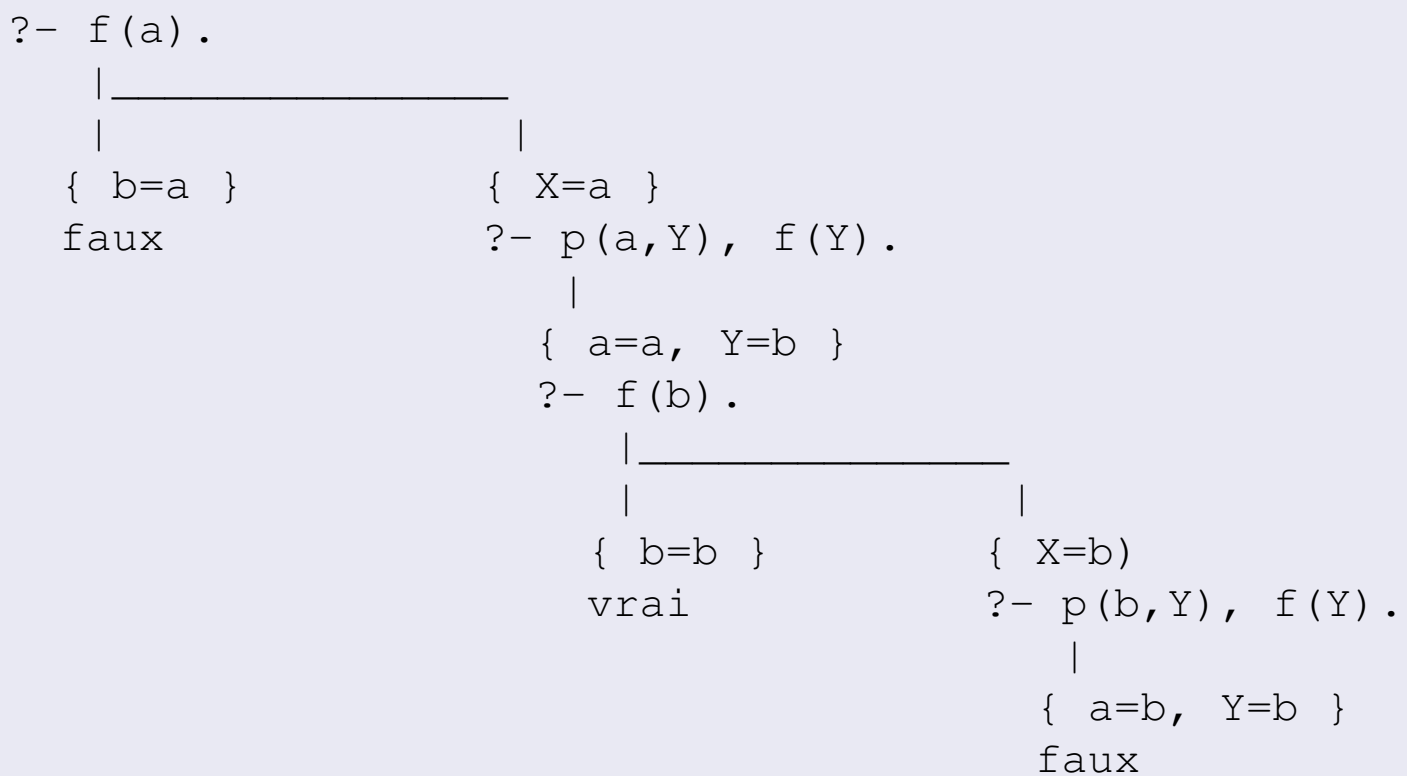
- la réponse est `vrai { X=a }`
- la première réponse demande 3 unifications
- la deuxième réponse demande 5 unifications

quand l'ordre des clauses crée des boucles infinies

- programme

```
p(a,b).                    g(b).
f(b).                      g(X):-g(Y),p(X,Y).
f(X):-p(X,Y), f(Y).
```

- résolution de `f(a)`

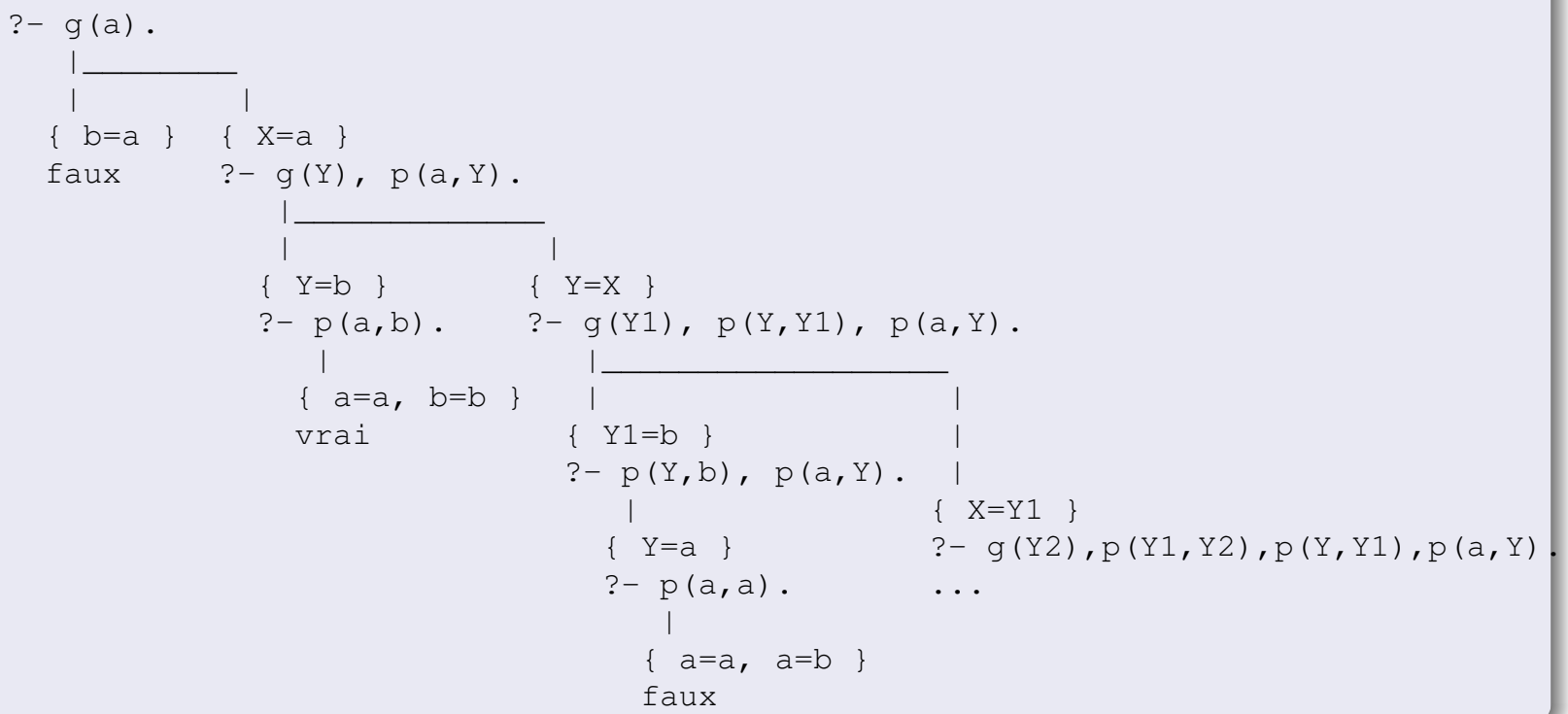


quand l'ordre des clauses crée des boucles infinies (suite)

- programme

```
p(a,b).                    g(b).
f(b).                      g(X):-g(Y),p(X,Y).
f(X):-p(X,Y), f(Y).
```

- résolution de `g(a)`



conclusion

- la programmation en Prolog n'est pas purement logique
- la résolution d'un problème correspond à un parcours en profondeur
- l'ordre des règles dans le programme et l'ordre des clauses dans le but en cours d'évaluation influent sur la longueur de la résolution et sur la terminaison de la résolution