Trois méthodes d'interrogation avec des règles Datalog
Correction

Nous reprenons ici un exercice fait en chaînage avant et ajoutons les méthodes de chaînage arrière et de réécriture de requête.

On considère la base de connaissances suivante $\mathcal{K} = (F, \mathcal{R})$:

- $\mathcal{R}$ = {R1,R2} *(intuitivement, "sg" signifie "same generation")*

  R1: flat(x1,y1) → sg(x1,y1)

  R2 : up(x2,y2) ∧ sg(y2,z2) ∧ up(t2,z2) → sg(x2,t2)

- F = {flat(a,b), flat(b,c), flat(a,c), up(d,a), up(d,b), up(e,c), up(f,d), up(g,e) }

ainsi que la requête booléenne :

q() = {sg(x,y) , up(y,z), flat(z,c)} où *x*, *y* et *z* sont des variables.

1. Chainage avant (effectué en TD)

1) **Saturez** la base de faits avec les règles, en procédant **en largeur** (cf. algorithme FC du cours). A chaque étape, on ne considère que les **nouveaux** homomorphismes. On dit qu'une application de règle est **utile** si elle produit un fait qui n'appartient pas à la base de faits courante.

Etape	Règle applicable	Homomorphisme	Fait produit	Application utile ?
1	R1	x1 --> a, y1--> b	sg(a,b)	o
	R1	x1 --> b, y1--> c	sg(b,c)	o
	R1	x1 --> a, y1--> c	sg(a,c)	o
2	R2	(x2,y2,z2,t2) --> (d,a,b,d)	sg(d,d)	o
	R2	(x2,y2,z2,t2) --> (d,a,c,e)	sg(d,e)	o
	R2	(x2,y2,z2,t2) --> (g,e,b,d)	sg(g,d)	o
	R2	(x2,y2,z2,t2) --> (d,b,c,e)	sg(d,e)	n
3	R2	(x2,y2,z2,t2) --> (f,d,d,f)	sg(f,f)	o
	R2	(x2,y2,z2,t2) --> (f,d,e,g)	sg(f,g)	o
4	*fin*			

- Autre notation pour l'homomorphisme, par exemple pour la première ligne : { (x1,a),(y1,b) }
- Notation raccourcie à partir de l'étape 2 : (x2,y2,z2,t2) → (d,a,b,d) signifie :
  x2 --> d, y2 --> a, z2 --> b, t2 --> d, ou encore : {(x2,d),(y2,a),(z2,b),(t2,d)}

Etape 1 sur F0 = F

Etape 2 sur F1 = F0 U {sg(a,b), sg(b,c), sg(a,c)}

Etape 3 sur F2 = F1 U {sg(d,d), sg(g,d), sg(d,e) }

Etape 4 sur F3 = F2 U {sg(f,f) , sg(f,g)} : pas de nouvel homomorphisme, donc F* = F3.

2)

a. Comment reconnaît-on qu'un homomorphisme est *nouveau* ?

Un homomorphisme h est nouveau à l'étape i s'il envoie le corps de la règle dans *au moins un* atome ajouté à l'étape i-1. Autrement dit, h(corps(R)) n'est pas inclus dans Fi-2.

b. On dit qu'un prédicat est *intentionnel* s'il apparaît au moins une fois en tête de règle : ici, *sg* est un prédicat intentionnel, et c'est le seul (ceux qui n'apparaissent pas en tête de règle sont dits *extensionnels*). L'ensemble de règles ci-dessus a une particularité : le corps de chaque règle contient au plus un atome avec un prédicat intentionnel. Un tel ensemble de règles est appelé *linéaire*. Comment exploiter le fait qu'un ensemble de règles soit linéaire pour ne calculer que les homomorphismes nouveaux à chaque étape de largeur ?

Pour tester si un homomorphisme est nouveau à l'étape i : un seul atome peut s'envoyer dans un atome produit à l'étape (i-1) et tous les autres s'envoient dans la base de faits de départ. On peut par exemple construire les homomorphismes partiels qui envoient l'atome intentionnel dans un atome produit à l'étape précédente, puis chercher à les étendre en considérant la base de faits initiale. Pour i = 4 par exemple, on partirait de :

y2 --> f et z2 --> f

y2 --> f et z2 --> g

3) La base de connaissances répond-elle positivement à q ? Justifiez votre réponse en vous basant sur le mécanisme de *chaînage avant*.

On trouve deux homomorphismes de q dans F* :

h1 : q --> F*	h2 : q --> F*
x --> d	x --> d
y --> d	y --> d
z --> a	z --> b

Il suffit d'exhiber l'un de ces 2 homomorphismes pour prouver que la base de connaissances répond positivement à q. Formellement : q(𝒦) = { () }. Ne pas confondre { () }avec {} (ou ø).

Si on cherchait les réponses à q(x,y,z), on aurait :

q(𝒦) = { (d,d,a), (d,d,b) }.

2. Chainage arrière

1) Montrer que la base de connaissances répond positivement à q() en utilisant le chaînage arrière.

Au lieu de prendre à chaque étape le premier atome de la requête courante, vous adopterez les priorités suivantes :

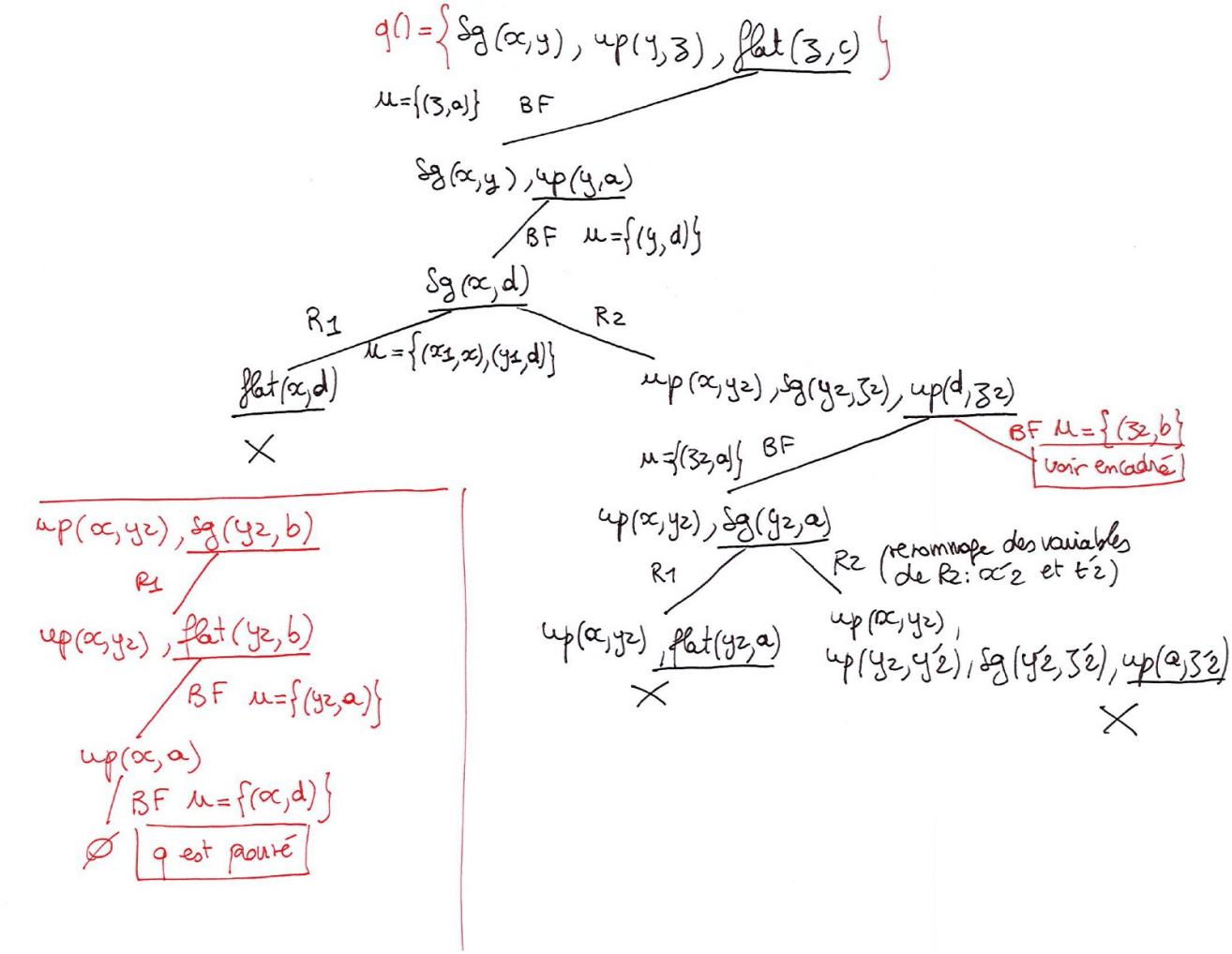
1. Les atomes ayant au moins une constante
2. Les atomes avec un prédicat extensionnel
3. Les atomes avec un prédicat qui apparaît aussi dans les faits.

q() = {sg(x,y) , up(y,z), flat(z,c)}

On unifie d'abord flat(z,c) avec un atome de F. Il y a deux possibilités :

- avec le fait flat(a,c) et u = { (z,a) }, on réécrit q en { sg(x,y) , up(y,a) }.
- avec le fait flat(b,c) et u = { (z,b) }, on réécrit q en { sg(x,y) , up(y,b) }

L'arbre qui suit correspond au premier choix. Puisqu'on parvient à "effacer" complètement la requête, et que la requête est booléenne, il n'est pas nécessaire de développer toutes les branches possibles.



2) Calculer l'ensemble des réponses à q(x,y,z) sur la base de connaissances, c'est-à-dire en considérant x,y et z comme des variables réponses. Ceci nécessite donc de mémoriser les substitutions des variables réponses au cours du calcul.

Ici, il faudrait développer l'arbre complet. En mettant la priorité sur les prédicats extensionnels, on retrouve bien les deux réponses en un nombre d'étapes fini, mais il reste des branches à développer avec R2, ce qui conduit à un arbre potentiellement infini.

3. Réécriture de requête

On considère maintenant le mécanisme d'interrogation par récriture de requête, qui décompose le processus en deux phases :

a. Récriture de la requête q (une CQ) en une union de requête conjonctive Q, en utilisant les règles, sans considérer les faits.
b. Interrogation de la base de faits avec Q (si la base de faits est stockée dans une base de données relationnelle, Q est transformée en une requête SQL).

Cette méthode est-elle utilisable avec la requête q() de l'énoncé (ou sa variante q(x,y,z)) ?

- Rappel de l'algorithme de réécriture de requête construit en TD :

```
Réécrire(q,R) // retourne l'ensemble des réécritures de q avec R
Début
Résultat = Ø // va contenir l'ensemble des réécritures de q (y compris q)
AExplorer = {q} // contient l'ensemble des réécritures pas encore explorées
TQ AExplorer n'est pas vide
        Retirer qi de AExplorer
        Ajouter qi à Résultat
        Pour toute règle Rj de R
                Pour tout unificateur u entre tête(Rj) et un atome de qi
                        Calculer qk la réécriture directe de qi avec Rj et u
                        Ajouter qk à AExplorer s'il "n'est pas déjà" dans AExplorer ou dans Résultat
                        ("à un renommage bijectif des variables près" = "à un isomorphisme près")
                FinPour
        FinPour
FinTQ
Retourner Résultat
Fin
```

q() = {sg(x,y) , up(y,z), flat(z,c)}

Puisque sg est le seul prédicat intentionnel, il n'est pas possible d'unifier les atomes up(y,z) et flat(z,c) avec des têtes de règle.

On peut unifier l'atome sg(x,y) de q() avec la tête de R1. Prenons u = {x1 --> x, y1 --> y }.
On obtient q1 = { flat(x,y) , up(y,z), flat(z,c) }, qui ne pourra pas être réécrite.

On peut aussi unifier l'atome sg(x,y) de q() avec la tête de R2. Prenons u = { x2 --> x, t2 --> y }.
On obtient q2 = { up(x,y2), sg(y2,z2), up(y,z2), up(y,z), flat (z,c) }.

On voit que q2 peut être à son tour réécrite avec R1 et avec R2. Chaque réécriture avec R2 replace un atome de prédicat sg par un atome de prédicat sg et deux atomes de prédicat up. Par récritures successives avec R2, on obtient des requêtes comportant deux "chemins d'atomes de prédicat up" de plus en plus longs. Chaque nouvelle requête obtenue n'est pas isomorphe à une requête précédente. L'ensemble de réécritures est donc potentiellement infini (autrement dit, l'algorithme de réécriture ne s'arrête pas).

On ne peut donc pas appliquer ici la méthode par réécriture de requête.