Introduction aux règles Datalog

HAI824 – Traitement sémantique des données

ML Mugnier

Rappels du cours précédent

- Toute BD relationnelle peut être vue comme une base de faits logiques, et réciproquement
- L'algèbre relationnel (base théorique de SQL) a la même expressivité que les requêtes de la logique du premier ordre
- Deux classes de requêtes fondamentales :
  - requêtes conjonctives (conjunctive queries - CQ)
  - unions de requêtes conjonctives (union of CQs - UCQ)
- Une réponse à $Q(x_1 \ldots x_k)$ sur une base de faits *F* est une liste $(c_1 \ldots c_k)$ de constantes telle que *F* répond oui à $Q(x_1/c_1, \ldots, x_k/c_k)$
  On peut voir *F* comme un modèle de $Q(x_1/c_1, \ldots, x_k/c_k)$
- Ceci correspond à l'hypothèse du monde clos. En monde ouvert, on aurait la notion de réponse certaine : $F \vDash Q(x_1/c_1, \ldots, x_k/c_k)$.
- Pour les (U)CQs, cela ne fait pas de différence !
- Les réponses à une CQ *Q* sur *F* correspondent à des homomorphismes de *Q* dans *F*.

Rappels du cours précédent

F

p(a,b)
p(b,a)
p(a,c)
q(b,b)
q(a,c)
q(c,b)

$Q_1()$ = { p(x,y), p(y,z), q(z,x) }
$Q_2(x)$ = { p(x,y), p(y,z), q(z,x) }
$Q_3(x,y,z)$ = { p(x,y), p(y,z), q(z,x) }

Homomorphismes de ces requêtes dans F

$x \mapsto b$	$x \mapsto b$
$y \mapsto a$	$y \mapsto a$
$z \mapsto c$	$z \mapsto b$

On obtient donc :

$Q_1(F) = \{()\}$
$Q_2(F) = \{ (b) \}$
$Q_3(F) = \{ (b,a,c), (b,a,b)\}$

Ne pas confondre $Q_1(F) = \{()\}$ « oui »
avec $Q_1(F) = \{\}$ « non »

Rappels du cours précédent

Soit $Q(x_1,\ldots,x_k)$ une requête **conjonctive**.

Le k-uplet de constantes $(c_1, \ldots, c_k)$ est une réponse à Q dans F

ssi

F est un modèle de $Q(x_1/c_1, \ldots, x_k/c_k)$ [par définition de la notion de réponse]

ssi

il existe un homomorphisme de Q dans F qui envoie chaque x_i sur c_i

Si k = 0 :
La réponse à Q dans F est *oui*
ssi il existe un homomorphisme de Q dans F

EXEMPLE : PISTES CYCLABLES

BF

Direct(A,B)

Direct(B,C)

Direct(C,D)

Direct(D,B)

Quelques requêtes du premier ordre (des CQs ici)

Q() = Direct(A,C) ?

Q(x,y) = Direct(x,B) $\wedge$ Direct(B,y) ?

Comment demander s'il y a un chemin de A à C ?

Impossible en requête du 1er ordre : il faudrait une UCQ infinie !

Requête Datalog

Direct(x,y) $\rightarrow$ Chemin(x,y)

Direct(x,y) $\wedge$ Chemin(y,z) $\rightarrow$ Chemin(x,z)

Chemin(A,C) $\rightarrow$ answer()

Les faits de prédicat answer donnent les réponses à la requête
(ici une seule réponse car la requête est booléenne)

RÈGLES CONJONCTIVES POSITIVES

Règle Datalog : **$\forall x_1 \ldots \forall x_n$ (H → C)** où :

- H est une conjonction d'atomes et C est un atome
- $x_1 \ldots x_n$ sont les variables de H
- **toutes** les variables de **C** apparaissent dans **H**

Hypothèse → Conclusion
Corps → Tête

Tête ← Corps

$\forall x \forall y$ $\forall z$ ((aParent(x,z) ∧ aParent(y,z)) → frèreOuSoeur(x,y) **)**

On pourrait ajouter ¬=(x,y) si on disposait de la négation

Notation simplifiée (sans les quantificateurs) :

aParent(x,z) ∧ aParent(y,z) → frèreOuSoeur(x,y)
aParent(x,z), aParent(y,z) → frèreOuSoeur(x,y)

Un fait est parfois vu comme une règle à hypothèse vide :

Un fait est donc un atome instancié puisque
« toutes les variables de C apparaissent dans H »

aParent(a,c), aParent(b,c), frèreOuSoeur(a,b), ...

Langage de requêtes Datalog (positif)

Idée : Ajouter la **récursivité** aux requêtes du premier ordre

Inspiré par Prolog (ancêtre de nombreux langages de programmation logique)

Datalog positif = UCQ + récursivité

- Une requête (ou « programme ») Datalog a la forme $(\mathcal{R}, \text{ans})$ où :
  - $\mathcal{R}$ est un ensemble de règles positives conjonctives
  - ans est un prédicat spécial apparaissant seulement en tête de règle et n'appartenant pas au vocabulaire de la base de données

Exemple : $(\mathcal{R}, \text{answer})$

$$\text{avec } \mathcal{R} = \begin{cases} \text{Direct}(x,y) \rightarrow \text{Chemin}(x,y) \\ \text{Direct}(x,y) \wedge \text{Chemin}(y,z) \rightarrow \text{Chemin}(x,z) \\ \text{Chemin}(A,C) \rightarrow \text{answer}() \end{cases}$$

EXEMPLE

Lignes

Ligne	Station 1	Station 2
4	St Germain	Odéon
4	Odéon	St Germain
10	Odéon	Cluny
…	…	…

"Trouver les stations directement connectées à Odéon"

ans(y) ← Lignes(x, Odéon, y)   équivalent à une CQ

ans(y) ← Lignes (x, Odéon, y)
ans(y) ← Lignes (x, y,Odéon)   équivalent à une UCQ

2 solutions
selon que les lignes
soient bidirectionnelles
ou pas

"Trouver les stations atteignables à partir d'Odéon, directement ou indirectement"

connecté (y, z) ← Lignes (x, y, z)
connecté (x, z) ← connecté (x, y), connecté (y, z)
ans(x) ← connecté (x, Odéon)

IDB / EDB / Réponses à une requête Datalog

Les prédicats d'un programme Datalog sont séparés en deux sortes :

- *L'ensemble des prédicats* **intensionnels** (**IDB**): ceux qui apparaissent en tête de règle (« Intensional DataBase »)
- *L'ensemble des prédicats* **extensionnels** (**EDB**): ceux qui n'apparaissent qu'en corps de règle (« Extensional DataBase »)

En général, on suppose que les prédicats de la base de données sont EDB.

answer est un prédicat IDB

Idée : l'évaluation d'une requête Datalog sur une BD construit des tables temporaires pour les atomes de prédicat IDB. L'ensemble des réponses à la requête est donné par la table answer

CHAÎNAGE AVANT

F

Direct(A,B)
Direct(B,C)
Direct(C,D)
Direct(D,B)

$\mathcal{R}$

R1 : Direct(x,y) → Chemin(x,y)

R2 : Direct(x,y) ∧ Chemin(y,z) → Chemin(x,z)

R3 : Chemin(A,C) → answer()

Une règle R : H → C est applicable à F s'il existe un homomorphisme h de H dans F

- Cette application de R à F est utile si h(C) ∉ F
- Une règle R : H → C est satisfaite dans F si pour tout homomorphisme h de H dans F, on a h(C) ∈ F (autrement dit, aucune application de R à F n'est utile)

- Appliquer R à F consiste à ajouter h(C) à F

- F est saturée (par rapport à $\mathcal{R}$) si toute règle R ∈ $\mathcal{R}$ est satisfaite dans F

ALGORITHME DE CHAÎNAGE AVANT (EN LARGEUR)

Algorithme ForwardChaining (K) *// Données: K = (F,$\mathcal{R}$)*
Début *// Résultat : F* = F saturée par $\mathcal{R}$*

Fin ← faux
F* ← F
Tant que non fin
  nouvFaits ← ∅ *// ensemble des nouveaux faits obtenus à cette étape*
  Pour toute règle R : H → C ∈ $\mathcal{R}$
    Pour tout (nouvel) homomorphisme h de H dans F*
      Si h(C) ∉ (F* ∪ nouvFaits)
        Ajouter h(C) à nouvFaits
  Si nouvFaits = ∅
    Fin ← vrai
  Sinon F* ← F* ∪ nouvFaits
Fin

Remarque : la valeur de F* ne dépend pas de l'ordre d'application des règles

CHAÎNAGE AVANT

F

Direct(A,B)
Direct(B,C)
Direct(C,D)
Direct(D,B)

$\mathcal{R}$

R1 : Direct(x,y) → Chemin(x,y)

R2 : Direct(x,y) ∧ Chemin(y,z) → Chemin(x,z)

R3 : Chemin(A,C) → answer()

(On note F_i = F* à l'étape i de l'algorithme)

F0 = F

1. nouvFaits = { Ch(A,B), Ch(B,C), Ch(C,D), Ch(D,B) }
   $F_1 = F_0 \cup$ nouvFaits
2. nouvFaits = {Ch(A,C),Ch(D,C), Ch(B,D), Ch(C,B)}
   $F_2 = F_1 \cup$ nouvFaits
3. nouvFaits = { Ch(C,C), Ch(A,D), Ch(D,D), Ch(B,B), answer() }
   $F_3 = F_2 \cup$ nouvFaits
4. nouvFaits = ø
   $F^* = F_3$ $|F^*| = 17$

Propriétés de la base de faits saturée

- Le chaînage avant calcule une base de faits saturée (F^*)
- F^* est unique
- F^* est le plus petit modèle de $F \cup \mathcal{R}$
  1. C'est un modèle de F car $F \subseteq F^*$
  2. C'est un modèle de $\mathcal{R}$: toute règle $R \in \mathcal{R}$ est satisfaite
     1+2 : c'est un modèle de $F \cup \mathcal{R}$
  3. C'est un plus petit modèle de $F \cup \mathcal{R}$: si on enlève un seul fait, ce n'est plus un modèle de $F \cup \mathcal{R}$
- F^* contient exactement l'ensemble des faits qui sont conséquence logique de $F \cup \mathcal{R}$:

  pour tout fait f, $f \in F^*$ ssi $F \cup \mathcal{R} \models f$

- L'ensemble de réponses à une requête Datalog ($\mathcal{R}$, $ans_{/k}$) sur une base de faits F est l'ensemble des k-uplets ($c_1 \ldots c_k$) tels que $ans(c_1 \ldots c_k) \in F^*$

Chaînage avant et réponses à une requête Datalog

F

Direct(A,B)
Direct(B,C)
Direct(C,D)
Direct(D,B)

$\mathcal{R}$

R1 : Direct(x,y) → Chemin(x,y)

R2 : Direct(x,y) ∧ Chemin(y,z) → Chemin(x,z)

R3 : Chemin(A,C) → answer()

F0 = F

1. nouvFaits = { Ch(A,B), Ch(B,C), Ch(C,D), Ch(D,B) }
   $F_1 = F_0 \cup$ nouvFaits
2. nouvFaits = {Ch(A,C),Ch(D,C), Ch(B,D), Ch(C,B)}
   $F_2 = F_1 \cup$ nouvFaits
3. nouvFaits = { Ch(C,C), Ch(A,D), Ch(D,D), Ch(B,B), answer() }
   $F_3 = F_2 \cup$ nouvFaits
4. nouvFaits = ø
   $F^* = F_3$

Réponses à ($\mathcal{R}$, answer) = { () }

CHAÎNAGE AVANT ET RÉPONSES À UNE REQUÊTE DATALOG

F

Direct(A,B)
Direct(B,C)
Direct(C,D)
Direct(D,B)

$\mathcal{R}$

R1 : Direct(x,y) → Chemin(x,y)
R2 : Direct(x,y) ∧ Chemin(y,z) → Chemin(x,z)
R3 : Chemin(B,x) → answer(x)

F0 = F

1. nouvFaits = { Ch(A,B), Ch(B,C), Ch(C,D), Ch(D,B) }
   $F_1 = F_0 \cup$ nouvFaits
2. nouvFaits = {Ch(A,C),Ch(D,C), Ch(B,D), Ch(C,B), **answer(C)**}
   $F_2 = F_1 \cup$ nouvFaits
3. nouvFaits = { Ch(C,C), Ch(A,D), Ch(D,D), Ch(B,B), **answer(D)** }
   $F_3 = F_2 \cup$ nouvFaits
4. nouvFaits = { **answer(B)** }
   $F_4 = F_3 \cup$ nouvFaits
5. $F^* = F_4$

Réponses à ($\mathcal{R}$, answer) = { (B), (C), (D) }

QUEL RAPPORT AVEC LES CQ ET UCQ ?

Une CQ **$q(x_1 \ldots x_k) = \exists x_{k+1},\ldots,x_m\ A_1 \wedge \ldots \wedge A_p$** peut être vue comme une requête Datalog composée d'une seule règle :

$$A_1 \wedge \ldots \wedge A_p \rightarrow answer(x_1 \ldots x_k)$$

« Trouver les cinémas dans lesquels on passe un film de Tarantino »

$Q(z) = \exists x\ \exists y\ \exists t\ (Film(x,qt,y) \wedge Programme(z,x,t))$

$Q(z) = \{\ Film(x,qt,y), Programme(z,x,t)\ \}$

En requête Datalog :

$Film(x,qt,y) \wedge Programme(z,x,t) \rightarrow answer(z)$

QUEL RAPPORT AVEC LES CQ ET UCQ ?

Une UCQ **$q(x_1 \dots x_k) = Q_1(x_1 \dots x_k) \vee \dots \vee Q_n(x_1 \dots x_k)$** où chaque Q_i est une CQ peut être vue comme une requête Datalog composé de n règles :

$$Q_1 \rightarrow answer(x_1 \dots x_k)$$

$$\dots$$

$$Q_n \rightarrow answer(x_1 \dots x_k)$$

« Trouver les cinémas et titres de films au programme de ces cinémas tel que ce soient des films de Tarantino ou avec Travolta ou le film « The Chef »

$$\begin{aligned} Q(z,x) = \; & (\exists y\, \exists t\, (Film(x,qt,y) \wedge Programme(z,x,t))) \vee \\ & (\exists y\, \exists t\, (Film(x,y,jt) \wedge Programme(z,x,t))) \vee \\ & (\exists t\, (Programme(z,x,t) \wedge x = \text{« The Chef »})) \end{aligned}$$

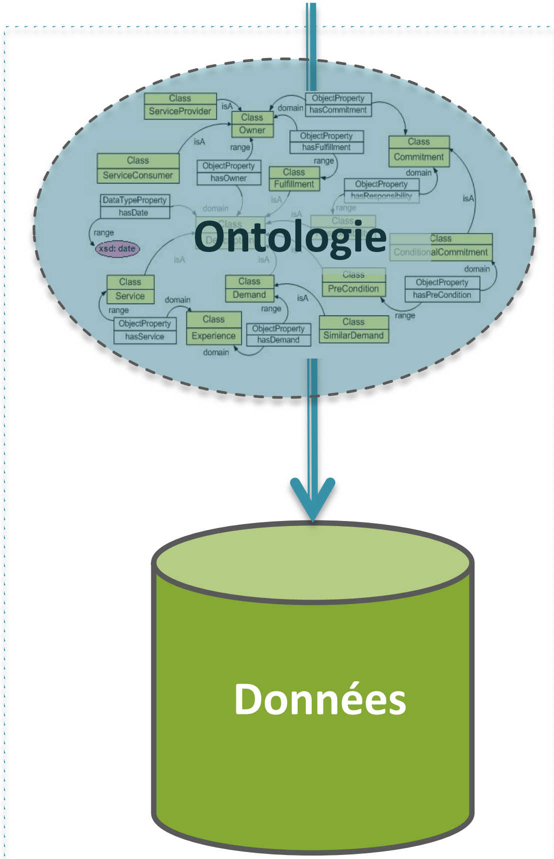
$$Film(x,qt,y) \wedge Programme(z,x,t) \rightarrow answer(z,x)$$

$$Film(x,y,jt) \wedge Programme(z,x,t) \rightarrow answer(z,x)$$

$$Programme(z,x,t) \rightarrow answer(z, \text{« The Chef »})$$

QUEL RAPPORT AVEC LES BASES DE CONNAISSANCES ?

Requête



Base de connaissances

Requête du premier ordre
(par exemple une CQ ou une UCQ)

Ensemble de formules logiques
(par exemple des règles conjonctives positives,
c'est-à-dire des règles Datalog)

Ensemble de faits (atomes instanciés)

Idée : la réponse a une requête booléenne est oui si cette requête est **conséquence logique de la base de connaissances**

Deux visions des règles Datalog

- **Datalog comme langage de requêtes** :

  Une requête $q = (\mathcal{R}, ans/k)$ est posée sur une BD (ou: base de faits) F

  L'ensemble des réponses à q est l'ensemble des (c1...ck) tels que ans(c1...ck) est conséquence logique de $F \cup \mathcal{R}$

- **Datalog comme langage ontologique :**

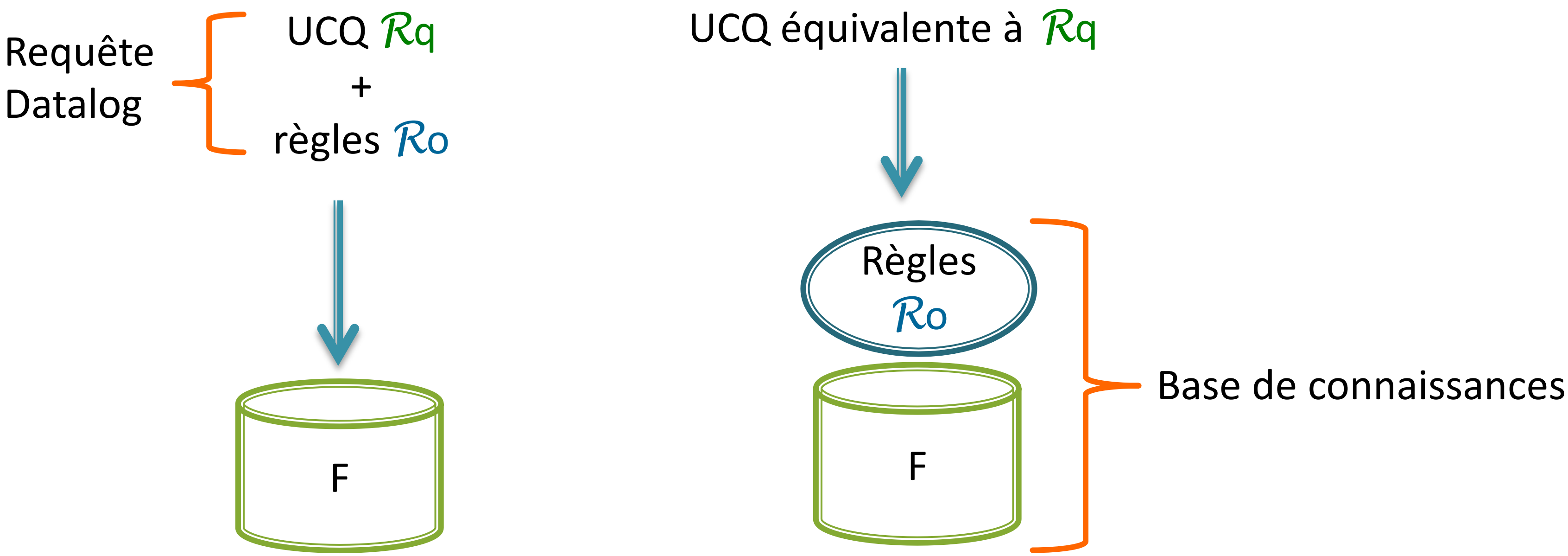
  $\mathcal{R}$ définit une ontologie
  Une requête CQ ou UCQ $q(x_1 \ldots x_k)$ est posée sur une base de connaissances composée d'une base de faits F et de $\mathcal{R}$

  L'ensemble des réponses à q est l'ensemble des (c1...ck) tels que Q(x1/c1...xk/ck) est conséquence logique de $F \cup \mathcal{R}$

REQUÊTE DATALOG VUE COMME UCQ + ONTOLOGIE

Toute requête Datalog ($\mathcal{R}$, ans) peut être décomposée en :

- $\mathcal{R}$q l'ensemble des règles dont la tête est un atome sur ans
- $\mathcal{R}$o : l'ensemble des autres règles



Les réponses à ($\mathcal{R}$q ∪ $\mathcal{R}$o, ans) sur la base de faits F sont exactement les réponses à l'UCQ équivalente à $\mathcal{R}$q sur la base de connaissances (F, $\mathcal{R}$o)

DATALOG COMME LANGAGE ONTOLOGIQUE

Avec un ensemble de règles Datalog, on peut décrire entre autres :

- une hiérarchie de classes (ou : concepts)
- une hiérarchie de relations (pas seulement binaires)
- les signatures de ces relations, c'est-à-dire la classe associée à chaque argument de la relation (« domain » et « range » en RDFS)
- les propriétés de ces relations :
  - réflexivité, symétrie, transitivité, ...
- le fait qu'une relation binaire est l'inverse d'une autre

etc.

On verra notamment comment on peut traduire
un ensemble de triplets **RDFS** en une base de connaissances Datalog

Où en sommes-nous ?

- **Base de connaissances K = (F,$\mathcal{R}$)** composée :
  - d'une **base de faits F**

    (qu'on peut voir comme une base de données relationnelle)
  - d'un ensemble **de règles** Datalog $\mathcal{R}$
- **Requêtes : (union de) requêtes conjonctives**

  (correspondant à des requêtes de base en SQL / SPARQL)
- **Problème fondamental : interrogation de la base de connaissances**

  Deux approches : matérialisation (chaînage avant, saturation)
  virtualisation (réécriture de requêtes)

Extensions de ce cadre

- **Contraintes négatives**
- (On évoquera des règles qui généralisent celles de Datalog)
- **Mappings** pour sélectionner une partie d'une base de données et la traduire en une base de faits

CONTRAINTES NÉGATIVES

- Une **contrainte négative** est de la forme

$$\forall X\ (\ \mathrm{Condition}[X] \rightarrow \bot\ )$$

où *Condition* est une conjonction d'atomes et $\bot$ le symbole absurde

$$\forall x\ (\mathrm{Film}(x) \wedge \mathrm{Personne}\ (x) \rightarrow \bot)$$

- Une base de faits *F* satisfait une **contrainte négative** *C* s'il n'y a pas d'homomorphisme de la condition de *C* dans *F*
(autrement dit, *C* vue comme une règle n'est pas applicable)

Remarque : *F* est un modèle de *C* **ssi** *F* satisfait *C*

- Une base de connaissances $K = (F, \mathcal{R}, \mathcal{C})$
où $\mathcal{C}$ est un ensemble de contraintes négatives est **consistante (satisfiable)** ssi *F** (la saturation de *F* par $\mathcal{R}$) satisfait toutes les contraintes de $\mathcal{C}$

Exercice (application directe du cours)

Soit la base de connaissances $\mathcal{K} = (F, \mathcal{R}, \mathcal{C})$

F = { r(a,b), r(b,c), r(c,a) }

$\mathcal{R}$ = { r(x,y) → s(x,y) ,

s(x,y) ∧ s(y,z) → s(x,z) }

$\mathcal{C}$ = {s(x,y) ∧ s(y,x) → ⊥}

- Quel est le plus petit modèle de $(F,\mathcal{R})$?
- F satisfait-elle $\mathcal{C}$? $\mathcal{K}$ satisfait-elle $\mathcal{C}$?
- $\mathcal{K}$ est-elle consistante (satisfiable) ?
- Soit q() = ∃x lapin(x). $\mathcal{K}$ répond-t-elle oui à q?

Interrogation de KBs avec contraintes négatives

Soit une base de connaissances $K = (F, \mathcal{R}, \mathcal{C})$

1. *K* est-elle satisfiable ?

   On calcule F^* la saturation de F par $\mathcal{R}$
   Puis on teste si F^* satisfait $\mathcal{C}$

2. Interrogation de *K*

   Si *K* n'est pas satisfiable, le problème d'interrogation « trivialise »
   (la réponse à une requête booléenne est toujours oui)

   Sinon, les réponses à une CQ *q* sont données par
   les homomorphismes de *q* dans F^*.