# Représentation de connaissances

## Interrogation de bases de connaissances

## avec règles Datalog

- Chaînage avant (Forward Chaining , « matérialisation »)
- Chaînage arrière classique (Backward Chaining)
- Variante de BC par réécriture de requête

Marie-Laure Mugnier

# Example (Forward Chaining)

$\mathcal{R}$ = { R: play(x,y), movie(y) → movieActor(x) }

*« find all movie actors »*

q(x) = movieActor(x)

F*

movie(m1)
movie(m2)
movie(m3)
movieActor(a)
movieActor(b)
play(a,m1)
play(a,m2)
play(c,m3)

F

movieActor(c)

Saturation

x = a
x = b
x = c

**Query Answering on (F, $\mathcal{R}$)**

1. **Forward chaining**: compute F* (can be done offline)
2. Evaluate queries on F* (based on homomorphisms to F*)

# EXAMPLE (BACKWARD CHAINING - CLASSICAL)

R: play(x,y), movie(y) → movieActor(x)

q(x) = movieActor(x)

movie(m1)
movie(m2)
movie(m3)
movieActor(a)
movieActor(b)
play(a,m1)
play(a,m2)
play(c,m3)

**Rewrite** queries with the **rules** and the **facts** until obtaining empty sets of atoms
Compose substitutions to « trace » answers

- an atom of q unified with a fact

unifier x → a ⇨ ø

unifier x → b ⇨ ø

x = a
x = b

- an atom of q unified with head(R):

Renaming of R's variables (to avoid confusion)
R: play(x',y'), movie(y') → movieActor(x')

unifier x → x'

⇨ q1: play(x',y'), movie(y')

- an atom of q1 unified with fact play(c,m3) x' → c, y' → m3

⇨ q2: movie(m3)

- an atom of q2 unified with fact movie(m3)

unifier (empty) ⇨ ø

x = c

x = a
x = b
x = c

and similarly with q1 and the other facts

# UNIFIERS

R1: p(x1,y1), p(y1,z1) → gp(x1,z1)
R2: mo(x2,y2) → p(x2,y2)
R3: fa(x3,y3) →p(x3,y3)

A = gp(x,a)

q(x) = gp(x,a)

A **unifier** *u* of atoms *A* and *B* is a substitution (of **variables**) such that *u(A) = u(B)*

A **most general unifier** (mgu) of *A* and *B* is a unifier *u* of *A* and *B* such that every other unifier of *A* and *B* can be written as *s* o *u* where *s* is a substitution

A = gp(x,a)

B = gp(x1,z1)

u1:
x → x1
z1 → a

u2:
x1 → x
z1 → a

mgu

Il peut y avoir plusieurs mgu de A et B,
mais ils sont « équivalents » :
on passe de l'un à l'autre par une substitution s qui ne fait que renommer bijectivement les variables

u2 = { x1 → x } o u1
u1 = { x → x1 } o u2

u3:
x → a
z1 → a
x1 → a

u3 = { x1 → a } o u1

# DIRECT REWRITING

R1: p(x1,y1), p(y1,z1) → gp(x1,z1)
R2: mo(x2,y2) → p(x2,y2)
R3: fa(x3,y3) →p(x3,y3)

Note:
Facts are seen as rules with an empty body
ex: F1 seen as → F1

Basic step: computation of a **direct rewriting** of a query *q*

1. look for a mgu *u* of an atom A in *q* and the atom in the head of a rule *R*
2. the direct rewriting of *q* with *R* according to *u* is

$$rew(q,R,u) = u(body(R)) \cup u(q \setminus A)$$

q = gp(x,a)  head(R1) = gp(x1,z1)  u: x → x1, z1 → a

*rew(q,R1,u)* = p(x1,y1),p(y1,a)

q' = gp(x,a), p(x,b)  head(R1) = gp(x1,z1)  u: x → x1, z1 → a

*rew(q',R1,u)* = p(x1,y1), p(y1,a), p(x1,b)

# BACKWARD CHAINING POUR CQ BOOLÉENNE

Rappel : les faits sont vus ici comme des règles

**Algorithme BC(K,Q)**

*// Donnée : K = (F,R) et Q une requête booléenne (liste d'atomes)*

*// Remarque : on voit un fait **p(a1,…,ak)** comme une règle → **p(a1,…,ak)***

*// Résultat : vrai si K répond positivement à Q, sinon faux*

**Début**

**Si** Q = ø, retourner vrai

**Pour** toute règle R = B → H de **F** et **R**

telle que premier(Q) s'**unifie** avec H par un unificateur u

*// penser à renommer les variables de R si besoin*
*pour que les ensembles de variables de Q et R soient disjoints*

Q' ← u(B) U u(reste(Q))

**Si** BC(K,Q') = vrai, retourner vrai

**FinPour**

Retourner faux

**Fin**

# BACKWARD CHAINING POUR CQ NON BOOLÉENNE

Rappel : les faits sont vus ici comme des règles

// Q(x1,...,xk) est une CQ vue comme un ensemble d'atomes

// S = {(x1,x1,),...,(xk,xk)} initialement

**Algorithme BC(K,Q,S)**

*// Donnée : K = (F,R) et Q une CQ, S substitution de {x1,...,xk}*

*// Résultat : Ensemble des {(x1,a1), ...(xk,ak) | (a1,...ak) est une réponse à Q dans K}*

**Début**

Rep ← ø // sera le résultat

**Pour** toute règle R = B → H de **F** et **R** telle que premier(Q) s'**unifie** avec H par u

*// si besoin, renommer les variables de R pour que variables(Q) ∩ variables(R) = ø*

Q' ← u(B) U u(reste(Q))

S ← u o S *// mise à jour de (xi,y) par (xi,z) si (y,z) ∈ u*

**Si** Q' = ø, ajouter S à Rep

**Sinon** Rep ← Rep U BC(K,Q',S)

**FinPour**

Retourner Rep

**Fin**

# EXEMPLE BC

*K* :

R1: p(x1,y1), p(y1,z1) → gp(x1,z1)
R2: mo(x2,y2) → p(x2,y2)
R3: fa(x3,y3) →p(x3,y3)

Q(x) = gp(x,a)

*« Qui sont tous les grand-parents de a ? »*

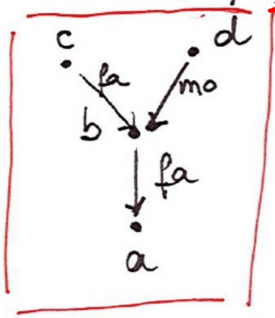
F1: fa(b,a)
F2 : fa(c,b)
F3 : mo(d,b)

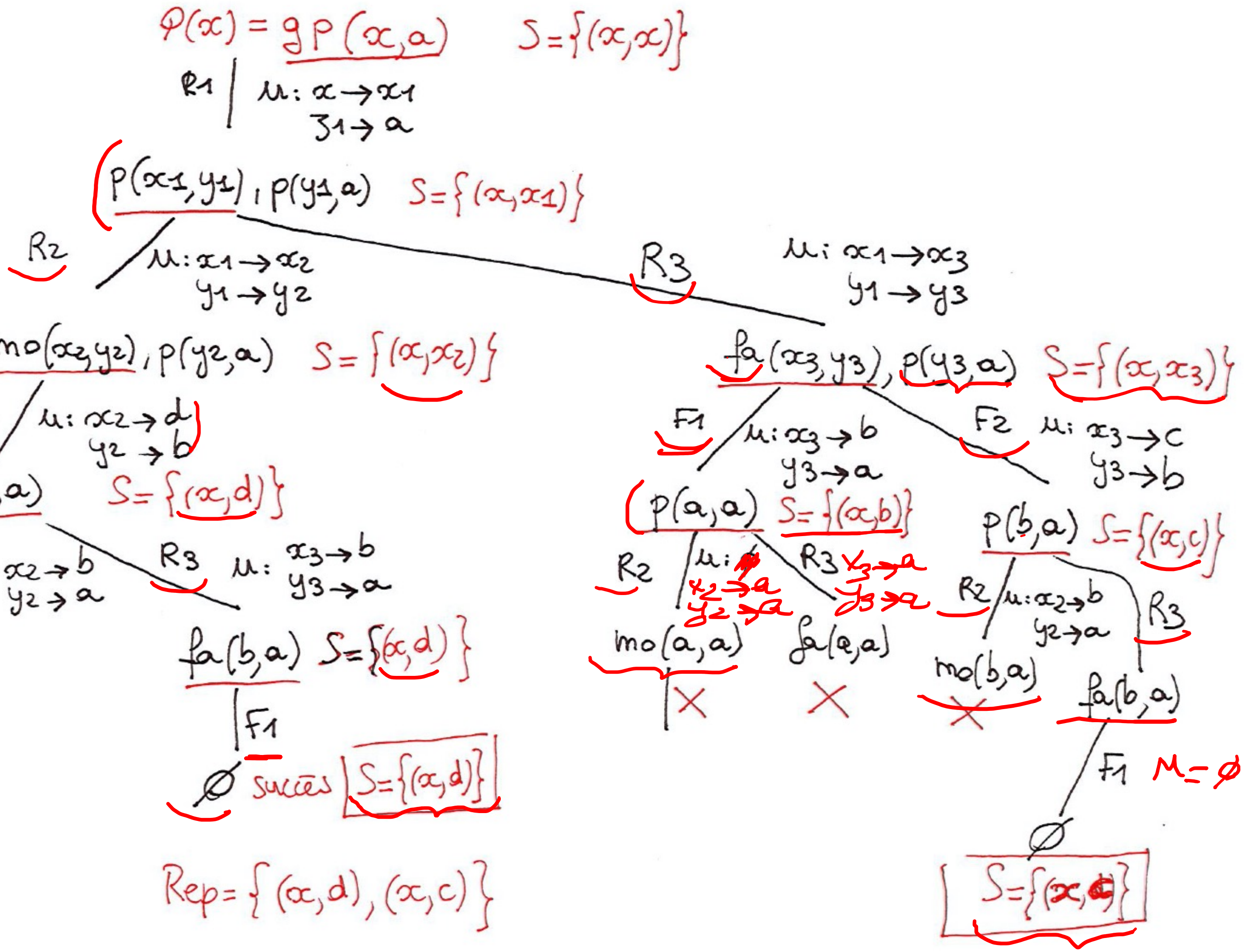
En chainage avant

**F*** = {fa(b,a), fa(c,b), mo(d,b), p(b,a), p(c,b), p(d,b), gp(c,a), gp(d,a) }

Q(K) = Q(F*) = { (c), (d) }

$F_1 = fa(b,a)$
$F_2 = fa(c,b)$
$F_3 = mo(d,b)$




Rep={(x,d), (x,c)}

# Approche Alternative : réécriture de requête

- Le **chaînage arrière classique** nécessite de faire des unifications avec la base de faits « atome par atome »
- Si la base de faits est stockée dans une base de données, cela nécessite d'accéder fréquemment à la base de données pour faire des requêtes élémentaires
  => impraticable pour de grandes bases de faits

- **Idée** : **décomposer le chaînage arrière en deux phases**

1. **Réécriture de requête** :
   réécrire la requête avec la base de règles (les « vraies » règles)
   => on obtient un ensemble de requêtes conjonctives
   que l'on voit comme une union de requêtes conjonctives (UCQ)

2. **Evaluation de requête** : évaluer cette UCQ sur la base de données

# QUERY REWRITING WITH DATALOG RULES

$\mathcal{R}$

R1: p(x1,y1), p(y1,z1) → gp(x1,z1)
R2: mo(x2,y2) → p(x2,y2)
R3: fa(x3,y3) →p(x3,y3)

q(x) = gp(x,a)

**$\mathcal{Q}$, set of rewritings of *q* with $\mathcal{R}$:**

q0(x) = gp(x,a)

R1 u : x1 → x, z1 → a

q1(x) = p(x,y1), p(y1,a)

mo(x,y1), p(y1,a) fa(x,y1), p(y1,a) p(x,y1), mo(y1,a) p(x,y1), fa(y1,a)

mo(x,y1), mo(y1,a) mo(x,y1), fa(y1,a) fa(x,y1), fa(y1,a) fa(x,y1),mo(y1,a)

**$\mathcal{Q}$ = { q0, q1, … q9}**

**UCQ($\mathcal{Q}$) = q0 V q1 V … V q9**

# Query Rewriting with Datalog Rules

$\mathcal{R}$

R1: p(x1,y1), p(y1,z1) → gp(x1,z1)
R2: mo(x2,y2) → p(x2,y2)
R3: fa(x3,y3) →p(x3,y3)

q(x) = gp(x,a)

**$\mathcal{Q}$, set of rewritings of *q* with $\mathcal{R}$**

q0(x) = gp(x,a)
q1(x) = p(x,y1), p(y1,a)
q2(x) = mo(x,y1), p(y1,a)
q3(x) = fa(x,y1), p(y1,a)
q4(x) = p(x,y1), mo(y1,a)
q5(x) = p(x,y1), fa(y1,a)
q6(x) = mo(x,y1), mo(y1,a)
q7(x) = mo(x,y1), fa(y1,a)
q8(x) = fa(x,y1), mo(y1,a)
q9(x) = fa(x,y1), fa(y1,a)

**UCQ($\mathcal{Q}$) = q0 V q1 V … V q9**

F

F1: fa(b,a)
F2 : fa(c,b)
F3 : mo(d,b)

q((F, R)) = $\mathcal{Q}$ (F)

= { (c), (d) }

Let q be a Boolean CQ and $\mathcal{Q}$ be its set of rewritings with $\mathcal{R}$

For any factbase *F*,

$F, \mathcal{R} \vDash q$

iff $F \vDash \mathcal{Q}$ ($\mathcal{Q}$ seen as a union of CQs)

iff there is $q_i$ in $\mathcal{Q}$ such that F $\vDash$ $\boldsymbol{q_i}$

Let q(x1, ..., xk) be a CQ and $\mathcal{Q}$ be its set of rewritings with $\mathcal{R}$

For any factbase *F*, a tuple of constants (a1,...,ak) is an answer to q on (F, $\mathcal{R}$)

iff (a1, ..., ak) is an answer to the UCQ $\mathcal{Q}$ on F

iff there is $q_i$ in $\mathcal{Q}$ such that (a1, ..., ak) is an answer to $\boldsymbol{q_i}$ on *F*

# PROBLÈME 1 : LA RÉÉCRITURE PEUT ÊTRE INFINIE

$R = \text{friend}(u,v) \wedge \text{friend}(v,w) \rightarrow \text{friend}(u,w)$

$q = \text{friend}(\text{Giorgos},\text{Maria})$

$q_1 = \text{friend}(\text{Giorgos}, v0) \wedge \text{friend}(v0,\text{Maria})$

$q_2 = \text{friend}(\text{Giorgos}, v1) \wedge \text{friend}(v1, v0) \wedge \text{friend}(v0,\text{Maria})$

$q_{2'} = \textit{friend}(\text{Giorgos}, v0) \wedge \textit{friend}(v0, v1) \wedge \textit{friend}(v1,\textit{Maria})$

$q_2$ and $q_{2'}$
are equivalent

$q_3 = \text{friend}(\text{Giorgos}, v2) \wedge \text{friend}(v2, v1) \wedge \text{friend}(v1, v0) \wedge \text{friend}(v1,\text{Maria})$

*Etc.*

Si on connait la taille de la base de faits, on peut borner la taille des réécritures,
mais cela donnera quand même de très grosses réécritures !

# Problème 2 : Efficacité de l'approche « réécriture » en pratique ?

- La taille de la réécriture peut être prohibitive en pratique

A
|
$B_1$
|
$B_2$
⋮
$B_n$

$B_1(x) \rightarrow A(x)$

$B_2(x) \rightarrow B_1(x)$

…

$B_n(x) \rightarrow B_{n-1}(x)$

$q = A(x_1) \wedge \ldots \wedge A(x_k)$

UCQ produite : $(n+1)^k$ CQ

Ce n'est pas un « pire des cas » théorique : se produit souvent en pratique

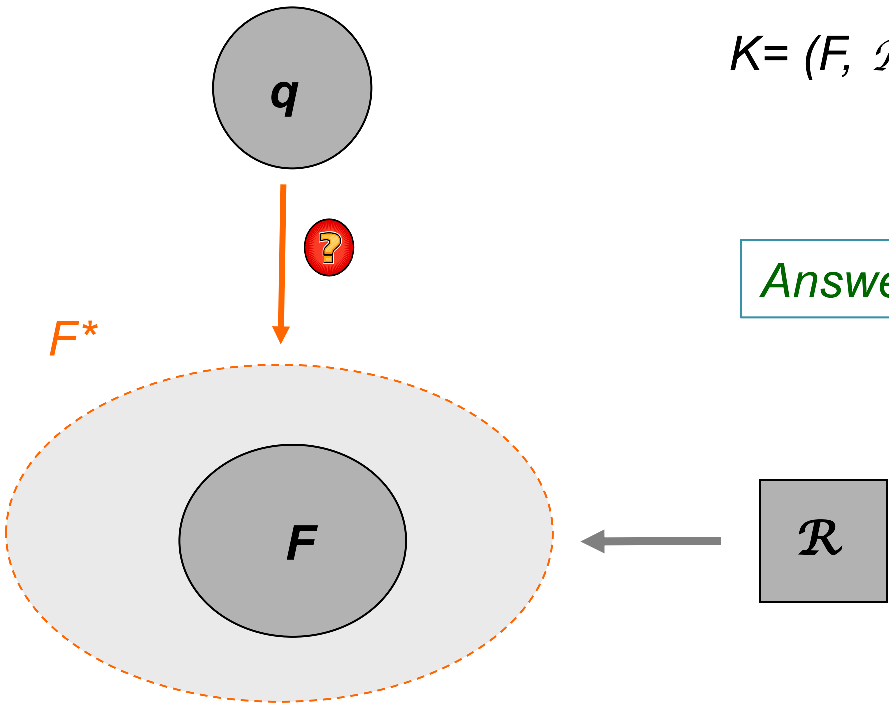
→ Réécriture en des formes de requêtes plus compactes

$$( A(x_1) \vee B_1(x_1) \ldots \vee B_n(x_1) ) \wedge \ldots \wedge ( A(x_k) \vee B_1(x_k) \ldots \vee B_n(x_k) )$$

$(n+1) \times k$ atomes au lieu de $(n+1)^k$

# Approach 1 to Rules : Forward chaining / Materialisation

$K = (F, \mathcal{R})$ « bottom-up »

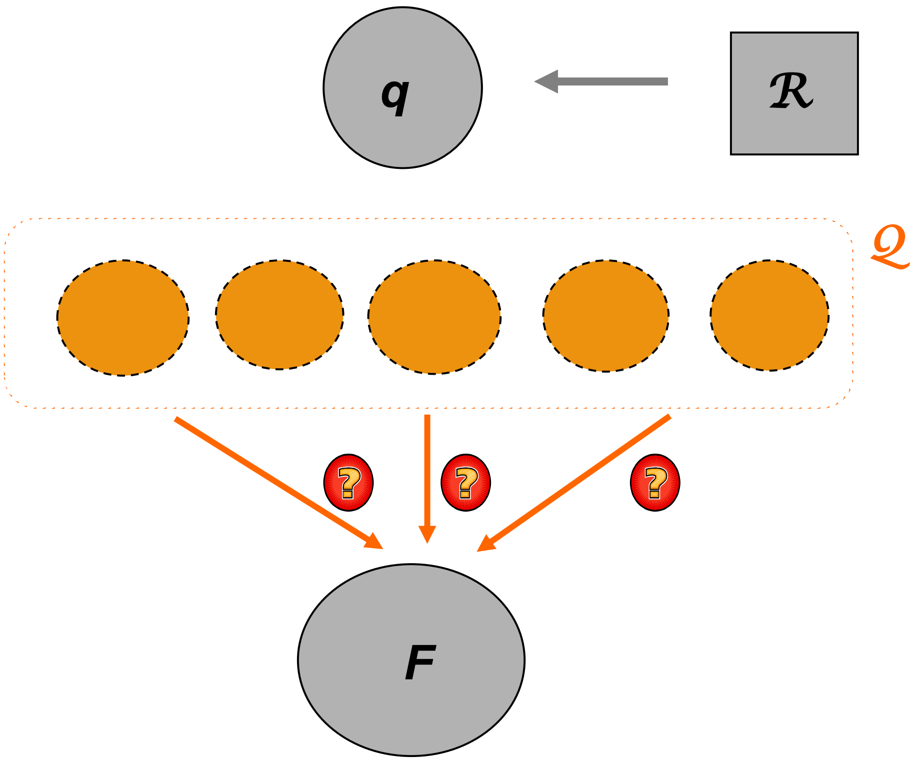



*Answers(q,K) = Answers(q, F*)*

**Pros:** materialisation offline, then online query answering is fast

**Cons:** volume of the materialisation
not feasible if data is distributed among several databases
not adapted if data change frequently

# APPROACH 2 TO RULES : QUERY REWRITING




$K = (F, \mathcal{R})$

« top-down »

Rewriting into a set of CQs, seen as a **union of conjunctive queries (UCQ)**

or to a more general first-order formula that still corresponds to an SQL query

Query rewriting is independent from any factbase

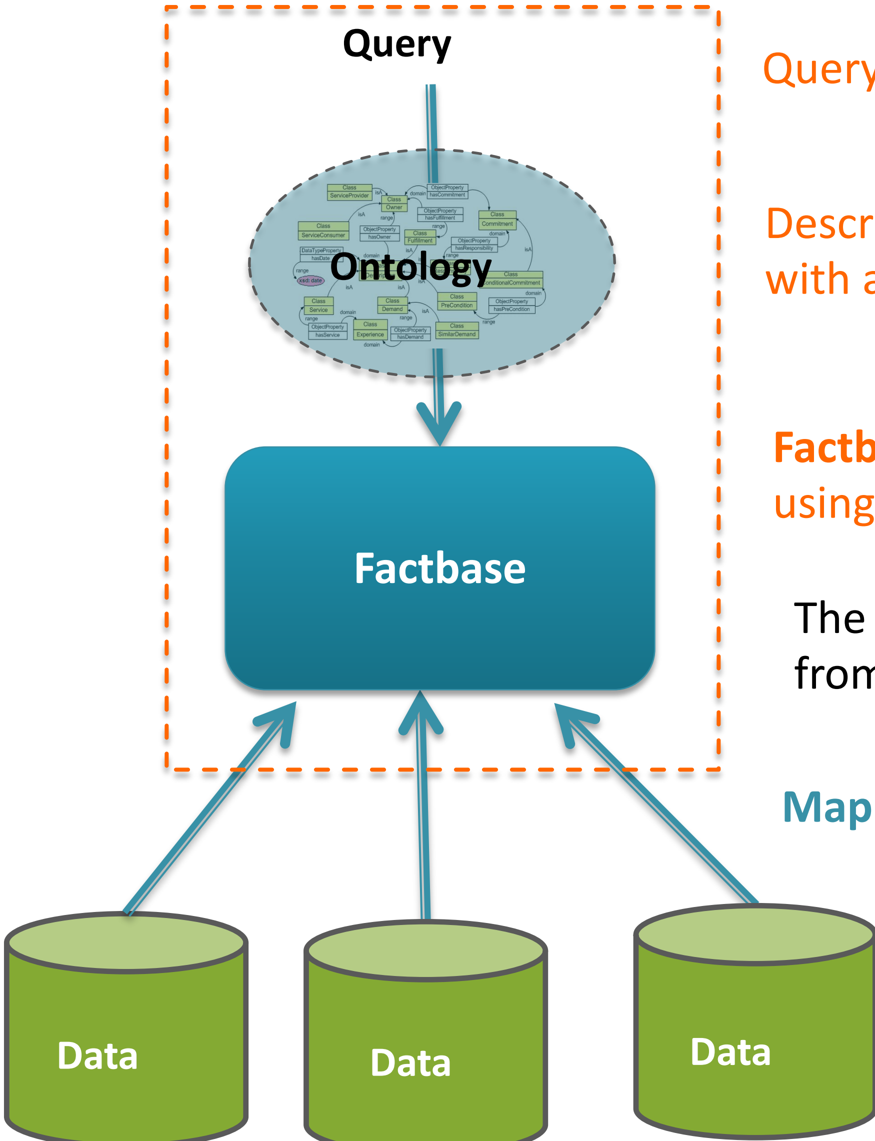
For **any** $F$, $Answers(q,(F,\mathcal{R})) = Answers(\mathcal{Q}, F)$

**Pros:** independent from the data
**Cons:** rewriting done at query time, easily leads to huge and unusual queries

# A MORE GENERAL SCHEMA

« Ontology-Based **Data Access** »
[Poggi et al., JoDS, 2008]

**Conceptual level**




Query using the vocabulary of the ontology

Description of the application domain with a high abstraction level

**Factbase (possibly virtual)**
using the vocabulary of the ontology

The **answers** to the query are **inferred** from the knowledge base

**Mappings from data to facts**

{ Database query ⇝ Facts }

Independent and heterogeneous data sources

# Mappings

Patient_T [ID_PATIENT, NAME,SSN]

Diagnosis_T[ID_PATIENT, DISORDER]

Patient /1
Diagnosis / 2
Influenza /1

**Mapping = database query(X) $\rightsquigarrow$ conjunction with free variables X**

$q(x): \exists n \exists s\ Patient\_T(x,n,s) \rightsquigarrow Patient(x)$ Datalog

$q'(x): \exists n \exists s\ Patient\_T(x,n,s) \land Diagnostic\_T(x,y) \land y =$ « influenza »
$\rightsquigarrow \exists z\ (diagnosis(x,z) \land Influenza(z))$ pas Datalog

Patient_T

| id | name | ssn |
|---|---|---|
| P | .. | .. |
| .. | .. | .. |
| .. | .. | .. |

Diagnosis_T

| id | dis |
|---|---|
| P | influenza |
| .. | .. |
| .. | .. |

$\rightsquigarrow$

Patient(P)
Diagnosis(P,M)
Influenza(M)

# Mappings can be seen as Rules

Patient_T [ID_PATIENT, NAME,SSN]

Diagnosis_T[ID_PATIENT, DISORDER]

q(x): ∃n∃s Patient_T (x,n,s) ⤳ Patient(x)

GAV (Global-As-View)

= Datalog

q'(x): ∃n∃s Patient_T (x,n,s) ∧ Diagnosis_T(x,y) ∧ y = « influenza »

⤳ ∃z diagnosis(x,z) ∧ Influenza(z)

Patient_T (x,n,s) → Patient(x) Datalog Rule

Patient_T (x,n,s), Diagnosis_T(x, « influenza ») →∃z diagnosis(x,z), Influenza(z)

Existential Rule

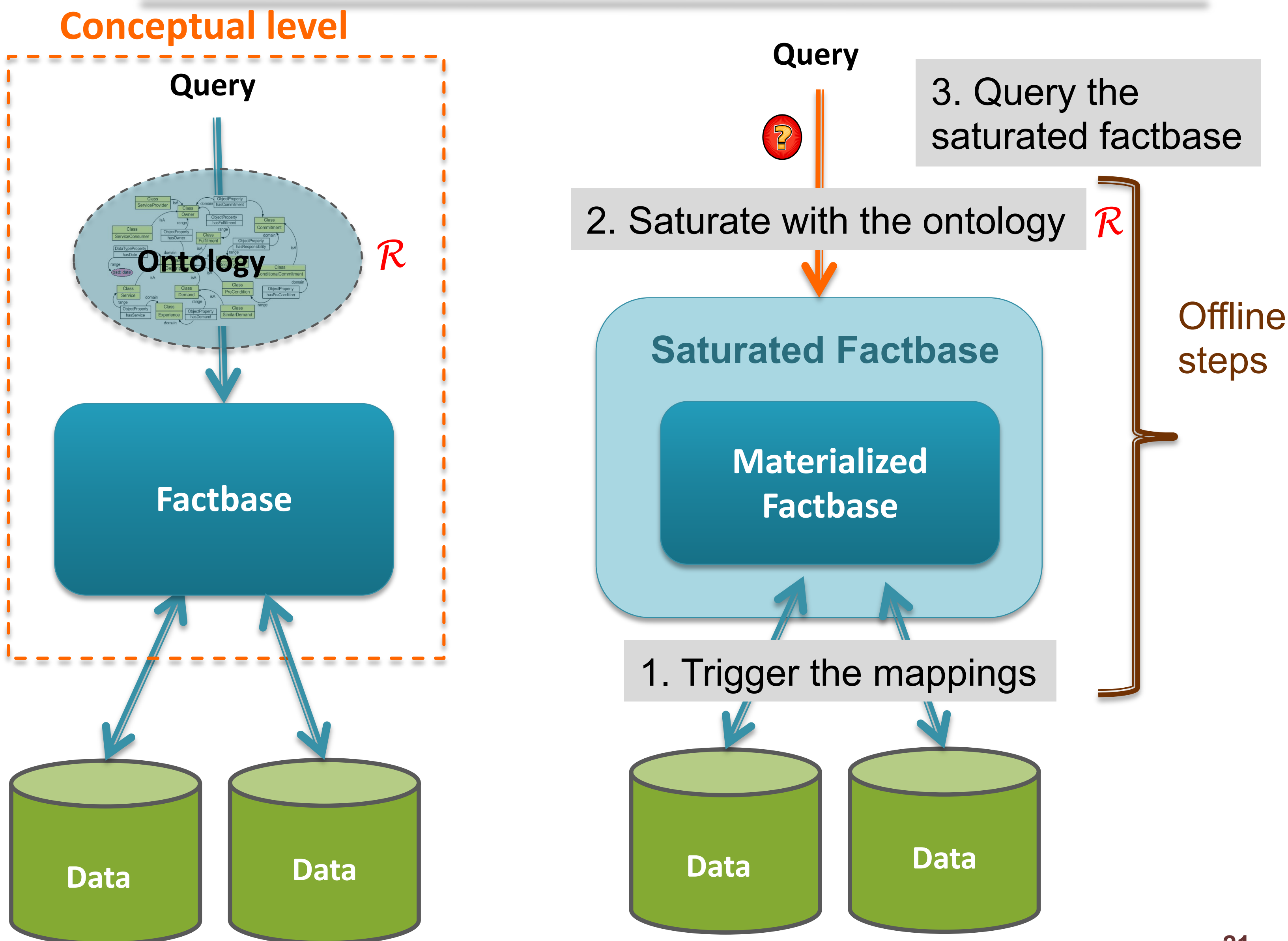
More generally: **$q_1$(X) ⤳ $q_2$(X)** where $q_1$ is expressed in a native query language

Decomposition of a mapping into 2 mappings

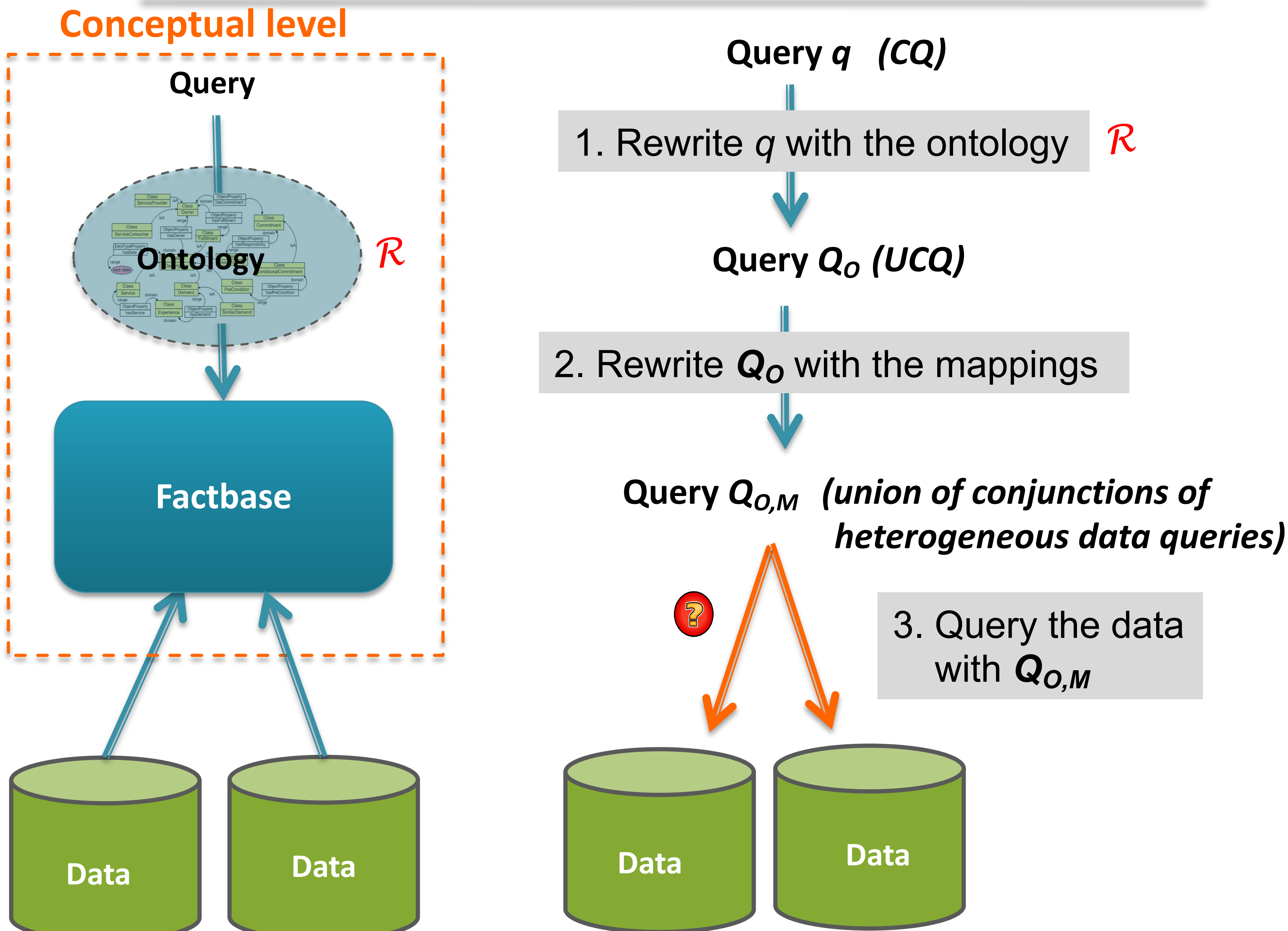
**low level** : $q_1$(X) ⤳ view(X) Result of the query stored in a view

**high level** : **view(X) → $q_2$(X)** Logical rule

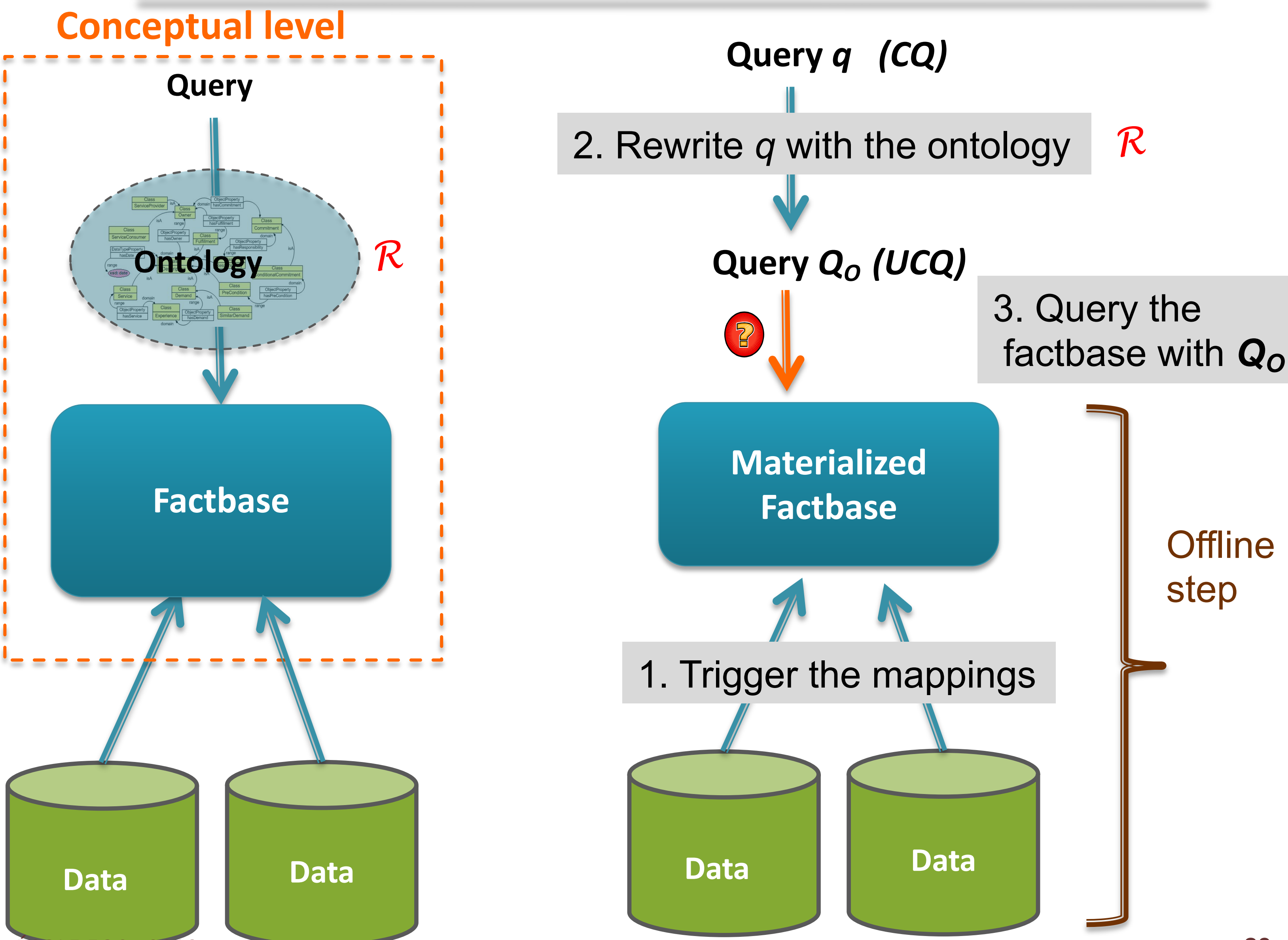
# OBDA : TOTAL MATERIALIZATION (FORWARD CHAINING)



# OBDA : TOTAL REWRITING



# OBDA : Example of Mixed Approach



# SYNTHÈSE

- **Bases de connaissances** avec **faits** et **règles datalog**
- Une **base de données** relationnelle peut être vue comme une base de faits et réciproquement
- Une **requête conjonctive (CQ)** correspond à une requête SQL de base
- Les réponses à une requête conjonctive sur une base de faits se calculent par **homomorphisme**
- Si on ajoute des **contraintes négatives**, la base de connaissances peut devenir insatisfiable
- Une base de connaissances (satisfiable) K a un **unique plus petit modèle**, qui suffit à calculer les réponses à une CQ sur K
- **Méthodes d'interrogation** de K :
  - par chaînage avant : la base de faits saturée correspond à un plus petit modèle de K
  - par chaînage arrière classique (« programmation logique », cf. Prolog)
  - par récriture de requête puis évaluation de la requête réécrite
- Cadre plus général avec des **mappings**