RÈGLES EXISTENTIELLES

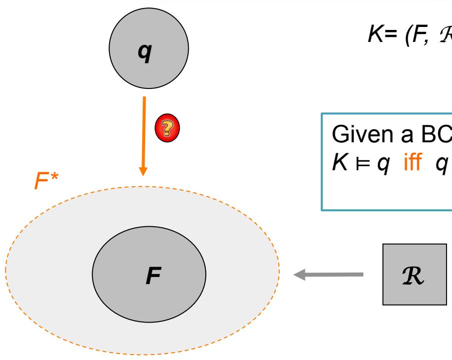
RÉÉCRITURE DE REQUÊTE

THÉORIE DES BASES DE CONNAISSANCES
HMIN312M

APPROACH 1 TO RULES : FORWARD CHAINING / MATERIALISATION

K= (F, ℛ)

« bottom-up »
« chase » (TGDs)



Given a BCQ *q*,
K ⊨ *q* iff *q* maps by homomorphism to *F**
(the saturation of *F* by ℛ)

Pros: materialisation offline, then online query answering is fast

Cons: volume of the materialisation
not adapted if data change frequently

Example (Forward Chaining)

movieActor(x) → ∃ z movie(z) ∧ play(x,z)

movie(m1)
movie(m2)
movie(x0)
movieActor(a)
movieActor(b)
play(a,m1)
play(a,m2)
play(c,x0)

movie(*z0*)
play(b,*z0*)

Saturation

q(x) = ∃ y (movie(y) ∧ play(x, y))

« find those who play in a movie »

x = a	y = m1
x = a	y = m2
x = b	y = z0
x = c	y = x0

q(F,$\mathcal{R}$) = { (a),(b),(c) }

Example (Backward Chaining by Query Rewriting)

movieActor(x) → ∃ z movie(z) ∧ play(x,z)

movie(m1)
movie(m2)
movie(x0)
movieActor(a)
movieActor(b)
play(a,m1)
play(a,m2)
play(c,x0)

q(x) = ∃ y (movie(y) ∧ play(x, y))

« find those who play in a movie »

$Rew_q(x)$ = ∃ y (movie(y) ∧ play(x, y)) ∨ movieActor(x)

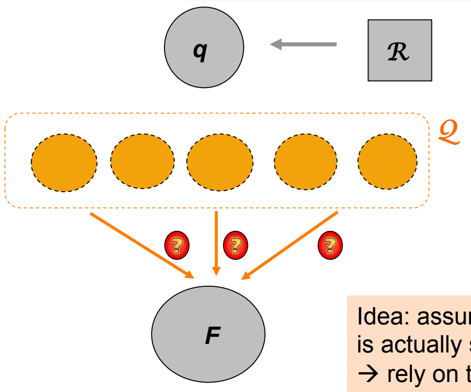
Query rewriting

x = a	y = m1
x = a	y = m2
x = c	y = x0

x = a
x = b

q(F,$\mathcal{R}$) = { (a),(b),(c) }

Approach 2 to Rules : Backward Chaining by Query Rewriting



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

« top-down »
Backward chaining
decomposed into 2 steps

Rewriting into a set of CQs, seen as a **union of conjunctive queries (UCQ)** and more generally into a « first-order » query (core SQL query)

Idea: assume the data corresponding to *F* is actually stored in a database
→ rely on the **DB manager** and its optimizations

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

Query rewriting is performed **independently** from any **factbase**

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

Standard Unification (as in Datalog Rules)

... → p(x,y,z,x)

p(u,v,w,a) *où a est une constante*

p(a,b,b,a)

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

u1:
x → a
y → v
z → w
u → a

u2:
x → a
y → b
z → b
u → a
v → b
w → b

A most general unifier (mgu) *u* of *A* and *B* is a unifier 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

u2 = s o u1 with s = {v → b w → b }

Query Rewriting with Datalog Rules (1)

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) ∧ f(x)

Basic step: computation of a **direct rewriting** of a (conjunctive) query *q*

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

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

x1 → x
z1 → a

q1(x) = p(x,y1) ∧ p(y1,a) ∧ f(x)

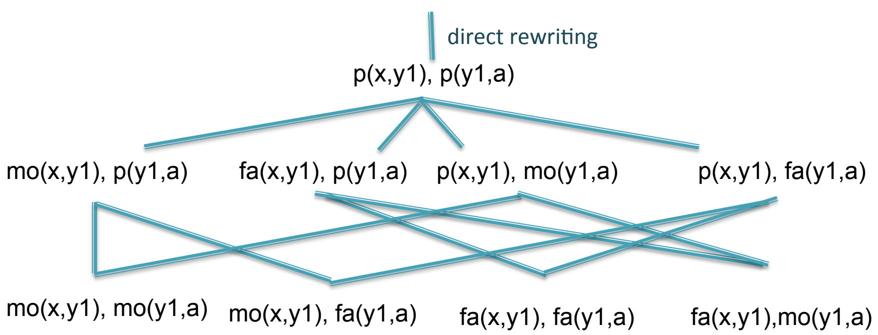
A **rewriting** of *q* with $\mathcal{R}$ is obtained by a (possibly empty) sequence of direct rewritings starting from *q* and using the rules in $\mathcal{R}$

Query Rewriting with Datalog Rules (2)

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 $\mathcal{R}$:



Let q be a Boolean CQ and $\mathcal{Q}$ be its set of rewritings.
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 ⊨ q_i

REDUNDANCE IN REWRITINGS

Let Q = $q_1 \vee q_2$, where $q_1 \sqsubseteq q_2$

Then q_1 is useless because every answer to q_1 is also an answer to q_2

We can keep only a **cover** of the rewritings

A **cover** of $\mathcal{Q}$ is a subset $\mathcal{Q}_c$ of $\mathcal{Q}$ such that:

1. for any *q* in $\mathcal{Q}$, there is *q'* in $\mathcal{Q}_c$ such that $q \sqsubseteq q'$
2. elements of $\mathcal{Q}_c$ are pairwise incomparable with respect to $\sqsubseteq$

We can also supress redundancies **inside** each conjunctive query *q* (computation of the **core** of *q*)

QUERY REWRITING CAN BE INFINITE

R = friend(u,v) ∧ friend(v,w) → friend(u,w)

q = friend(Giorgos,Maria)

q_1 = friend(Giorgos, v0) ∧ friend (v0,Maria)

q_2 = friend(Giorgos, v1) ∧ friend(v1, v0) ∧ friend (v0,Maria)

$q_{2'}$ = *friend*(Giorgos, *v0) ∧ friend(v0, v1) ∧ friend (v1,Maria)*

q_2 and $q_{2'}$ are equivalent

q_3 = friend(Giorgos, v2) ∧ friend(v2, v1) ∧ friend(v1, v0) ∧ friend (v1,Maria)

Etc.

Here, if we know the size of the data we can bound the number of atoms in a rewriting, (however, it will result in very large rewritings!)

FROM DATALOG TO EXISTENTIAL RULES

∀x (movieActor(x) → ∃ z (movie(z) ∧ play(x,z)))

q(x) = ∃ y (movie(y) ∧ play(x, y))   *« find those who play in a movie »*

If we compute rewritings as in Datalog, we obtain as direct rewritings of q:

q1(x) = play(x,y) ∧ movieActor(x1)   *incorrect (unsound)*

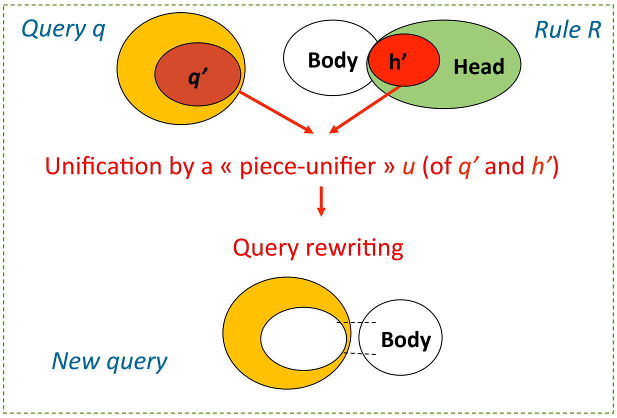
q2(x) = movie(y) ∧ movieActor(x)   *too restrictive*

The direct rewriting we want is: q'(x) = movieActor(x)

We already see that we may have to unify several atoms from the query at the same time

BACKWARD CHAINING SCHEME

Basic step:

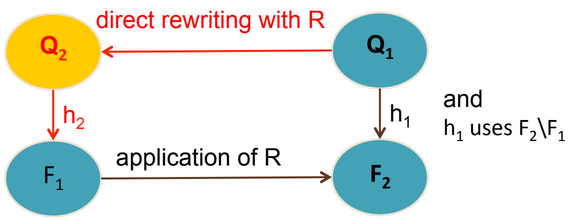


Direct rewriting of *q* with *R* and *u* =   *u(q \ q') ∪ u(body(R))*

BASIC PROPERTIES (1)

Let F_2 be obtained from F_1 by the application of Rule R
Let a (Boolean) CQ Q_1 that maps to F_2
by a homomorphism that uses at least one atom brought by R

Then there is Q_2, a direct rewriting of Q_1 with R, such that Q_2 maps to F_1

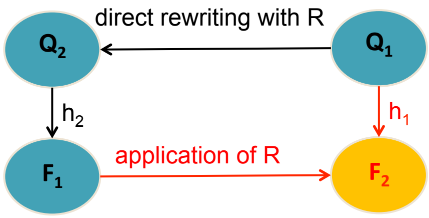


We also have the converse direction

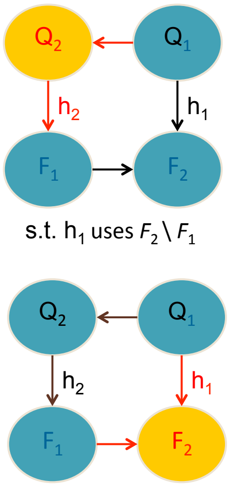
BASIC PROPERTIES (2)

Let Q_2 be a direct rewriting of Boolean CQ Q_1 with Rule R
Let F_1 be a factbase such that Q_2 maps to F_1

Then there is an application of R to F_1 that produces F_2 such that Q_1 maps to F_2
(furthermore, h_1 uses $F_2 \setminus F_1$)



EQUIVALENCE DERIVATION / REWRITING SEQUENCES



For any Boolean CQ q, for any factbase F, for any set of rules:

there is a homomorphism from q to F', where F' is obtained from F by a rule application sequence of length $\leq n$

iff

there is a homomorphism from q' to F, where q' is obtained from q by a rewriting sequence of length $\leq n$

TAKING INTO ACCOUNT EXISTENTIAL VARIABLES IN RULE HEADS (1)

- We want a **complete** set $\mathcal{Q}$ of **sound** rewritings (set of CQs):

  - completeness: for any F, if $F, \mathcal{R} \vDash q$ then $F \vDash \mathcal{Q}$ [there is $qi \in \mathcal{Q}$ such that $F \vDash q_i$]
  - soundness: for any F, if $F \vDash \mathcal{Q}$ then $F, \mathcal{R} \vDash q$

R = person(x) → ∃ y hasParent(x,y)
q = hasParent(v,w), dentist(w)
u = { x ↦ v, y ↦ w }
rew(q,R,u) = q_i = person(v), dentist(w)

q_i is **unsound**:
F = person(Maria), dentist(Giorgos)
$F \vDash q_i$ however $(F,\{R\})$ does not entail q

(1) If w in q is unified with an existential variable of R, then all atoms in which w occur must be part of the unification

Taking into account existential variables in rule heads (2)

$R = p(x) \rightarrow \exists z1 \exists z2\ r(x,z1), r(x,z2), s(z1,z2)$
$q = r(v,w), s(w,w)$
$u = \{x \mapsto v, z1 \mapsto w, z2 \mapsto w\}$
$rew(q,R,u) = q_i = p(v)$

q_i is **unsound**:

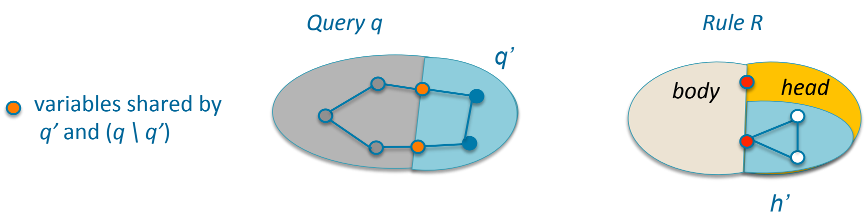
$F = p(a)$
$F \vDash q_i$ however $(F,\{R\})$ does not entail q

(2) An existential variable of *R* cannot be unified with another term in head(*R*)

Piece-unifier (for Boolean CQs)

A **piece-unifier** ***u*** of $q' \subseteq q$ and $h' \subseteq$ head(R)
is a substitution of *var(q' + h')* by *terms(q'+ h')* [if x is unchanged, we write u(x) = x]
such that :

- *u(q') = u(h')*
- existential variables of *h'* are unified only with variables of *q'* that do not occur in (q \ q')
  (i.e., if *z* is existential and *u(z) = u(t)*, then *t* is a variable of *q'* and not of (q \ q'))



To extend the notion to general CQs:
existential variables cannot be unified with answer variables

EXAMPLE

R = twin(x,y) → ∃ z motherOf(z,x) ∧ motherOf(z,y)

q = motherOf(v,w) ∧ motherOf(v,t) ∧ Female(w) ∧ Male(t) ?

R = twin(x,y) → ∃ z motherOf(z,x) ∧ motherOf(z,y)

q = motherOf(v,w) ∧ motherOf(v,t) ∧ Female(w) ∧ Male(t) ?

piece-unifier u_1 = {z ↦ v, x ↦ w, y ↦ t}

rewrite(*q*,*R*,u_1) = twin(w,t) ∧ Female(w) ∧ Male(t)

R = twin(x,y) → ∃ z motherOf(z,x) ∧ motherOf(z,y)

q = motherOf(v,w) ∧ motherOf(v,t) ∧ Female(w) ∧ Male(t) ?

piece-unifier u_2 = {z ↦ v, x ↦ w, t ↦ w}

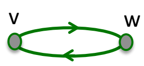
rewrite(*q*,*R*,u_2) = twin(w,y) ∧ Female(w) ∧ Male(w)

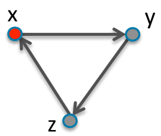
EXAMPLE

Are there piece-unifiers?

Case 1:

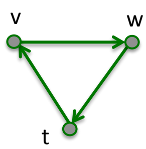
R = b(x) → ∃y∃z r(x,y), r(y,z), r(z,x)
q = r(v,w), r(w,v)

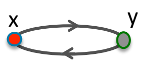




Case 2:

R= b(x) → ∃ y r(x,y), r(y,x)
q = r(v,w), r(w,t), r(t,v)





Case 1: no piece-unifier
Case 2: u = {{v,x,t},{y,w}}, which yields the rewriting b(t), r(t,t)

EXTENSION TO NON-BOOLEAN QUERIES WITHOUT EXTENDING PIECE-UNIFIERS

R = person(x) → ∃z hasParent(x,**z**)

(quantifiers omitted)

q1 = ∃v ∃w hasParent(v,w)

q'1 = ∃v person(v)

q2(w) = ∃v hasParent(v,w)

?? w should not « disapear »
(i.e. should not be unified with z)

- We transform q_2 into a Boolean query and add a special atom *ans(w)*
  [more generally, if the answer variables are x1 ...xk,
  we create an atom ans(x1..xk)]
- Since *ans* does not appear elsewhere, *w* can never be unified with an existential variable
- After rewriting, we remove all *ans* atoms

EFFICACITÉ DE L'APPROCHE « RÉÉCRITURE » EN PRATIQUE ?

- Intérêt de l'approche : indépendance vis à vis des données
- Mais la taille de la réécriture (si finie) 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
$(A(x_1) \vee B_1(x_1) \ldots \vee B_n(x_1)) \wedge \ldots \wedge (A(x_k) \vee B_1(x_k) \ldots \vee B_n(x_k))$

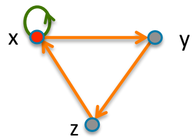
→ Utilisation combinée de la saturation et de la réécriture de requête

WHY « PIECES »?

A **piece** is a unit of knowledge brought by a rule:

- **Frontier variables** (and constants) act as **cutpoints** to decompose rule heads into pieces (« minimal non-empty subsets glued by existential variables »)

  R = b(x)→ ∃y ∃z p(x,y) ∧ p(y,z) ∧ p(z,x) ∧ q(x,x)



- A rule with *k* pieces can be decomposed into *k* rules, one for each piece, while keeping the same body

  b(x)→ ∃y ∃z p(x,y) ∧ p(y,z) ∧ p(z,x)

  b(x)→q(x,x)

- It cannot be further decomposed (except by introducing new predicates)

DECOMPOSITION OF RULES INTO ATOMIC HEAD RULES (1)

R: b(x)→ ∃y ∃z p(x,y) ∧ p(y,z) ∧ p(z,x) — rule with single-piece head

Decomposition into rules with atomic head by introducing a fresh predicate

R_0: b(x) → ∃y ∃z p_R (x,y,z)

R_1: p_R(x,y,z) → p(x,y)

R_2: p_R(x,y,z) → p(y,z)

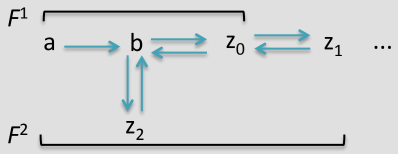
R_3: p_R(x,y,z) → p(z,x)

We lose the structure of the head

- much less efficient query rewriting
- may even lead to lose the property of having a finite universal model (if the set of rules has this property)

Decomposition of rules into atomic head rules (2)

F : p(a,b) R : p(x,y) → ∃z p(y,z), p(z,y)



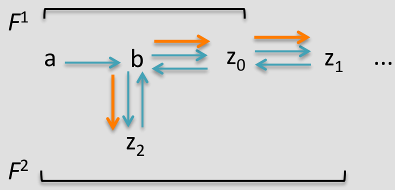
$F^2 \equiv F^1$ (F^2 maps to F^1)

hence $F^* \equiv F^1$

Finite universal model

After decomposition into atomic head rules:

R_0 : p(x,y) → ∃z p_R(y,z)
R_1 : p_R(y,z) → p(y,z)
R_2 : p_R(y,z) → p(z,y)

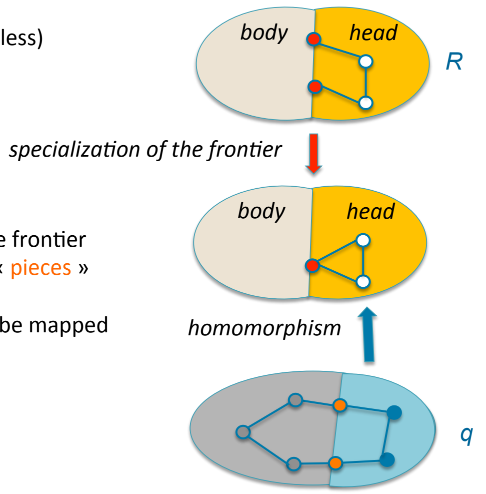


$F^2 \not\equiv F^1$

No finite universal model

Why « pieces » ? (cont'd) – Piece-unifiers

- Unification must « map » parts of q according to « pieces » that can be provided by a rule application
  (otherwise it is unsound or useless)



The terms of q unified with the frontier (or with constants) cut q into « pieces »

⇒ entire « pieces » of q must be mapped to the pieces of the rule

PIECE-UNIFIER: ALTERNATIVE DEFINITION

Let $\boldsymbol{u_1}$: frontier(R) → frontier (R) ∪ constants
(u_1 is a specialization of the frontier of R)

Let $\boldsymbol{u_2}$ be a homomorphism from $q' \subseteq q$ to $u_1(head(R))$

Cutpoints: terms of q' mapped to $u_1(frontier(R))$
or to constants

The cutpoints cut q into « **pieces** »

u_1+u_2 is a piece-unifier if q' is composed of *pieces*