# Complément sur les règles existentielles :

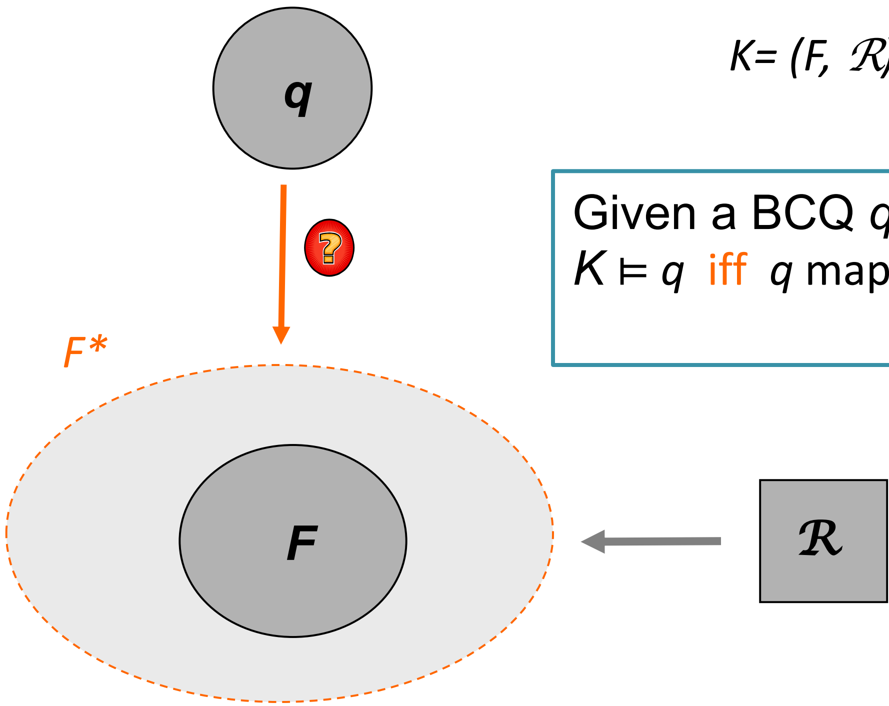
# Arrêt du Chase

HAI933I

# Approach 1 to Rules : Forward chaining / Materialisation

« chase »

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




Given a BCQ $q$,
$K \vDash q$ iff $q$ maps by homomorphism to $F^*$
(the saturation of $F$ by $\mathcal{R}$)

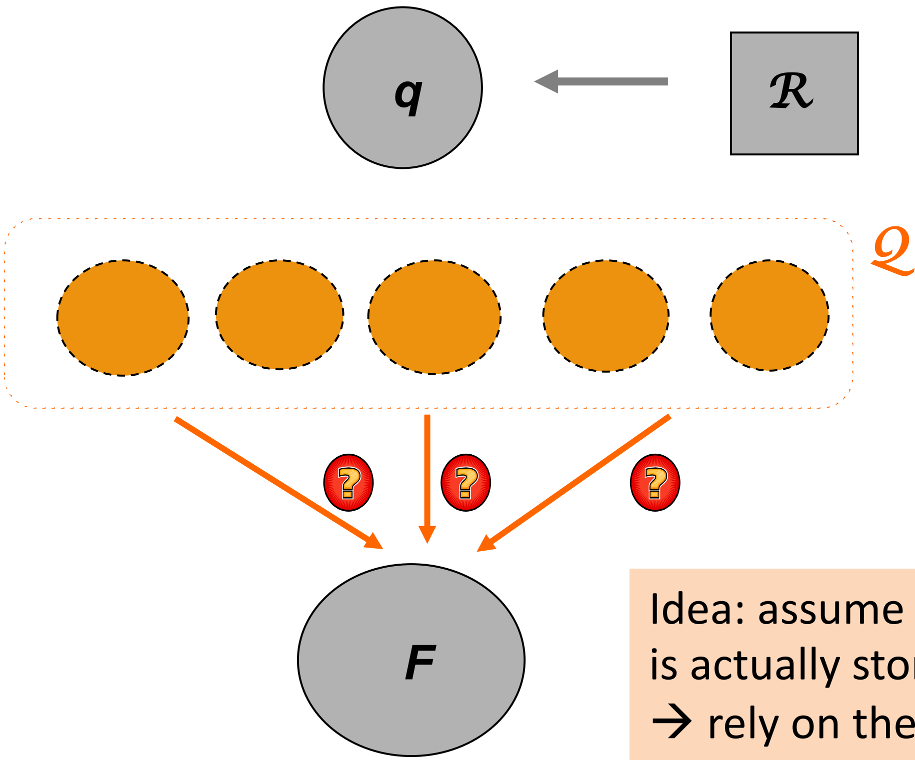
Given a CQ $q$,
$q(K) = q(F^*)$

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

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

**And of course, the chase has to halt …**

# Approach 2 to Rules : Backward Chaining by Query Rewriting




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

Rewriting into a set of CQs, seen as a **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 changes in the data
**Cons:** rewriting done at query time, easily leads to huge and unusual queries

**And of course, query rewriting has to halt …**

# Saturation may not halt

$R$ = person(x) → ∃y hasParent(x,y) ∧ person(y)

$F$ = person(a)

∧ person(y0) ∧ hasParent(a, y0)

∧ person(y1) ∧ hasParent(y0, y1)

Etc.

No redundancies are added

The KB has **no finite universal model**,

so even the core chase does not halt

# Query rewriting may not halt (even with Datalog rules)

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

$q = friend(Giorgos, Maria)$

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

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

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

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

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

*Etc.*

There is an infinite number of non-redundant rewritings

So, there is no UCQ $\mathcal{Q}$ such that for **any** *F*, $q(F,\{R\}) = \mathcal{Q}(F)$

# Undecidability of BCQ entailment

**BCQ entailment problem**

Input: $K = (F, \mathcal{R})$ knowledge base, $q$ Boolean conjunctive query

Question: is $q$ entailed by $K$ ?

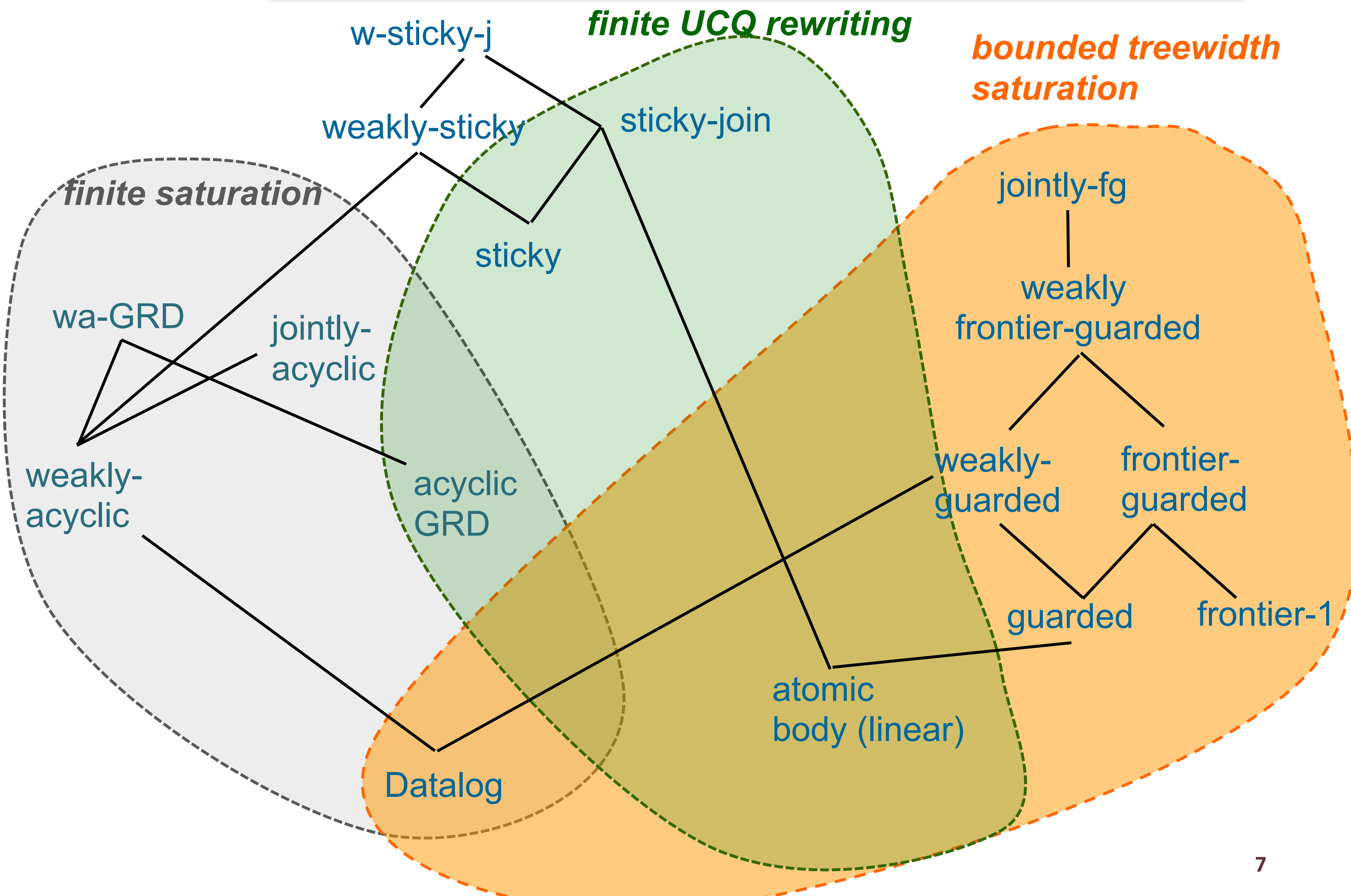
This problem is **undecidable** (only semi-decidable)

So, no technique is ensured to halt !

But many **subclasses** of existential rules are known, which ensure that:

- Saturation by the chase halts for any fact base
- Query rewriting halts for any conjunctive query
- Saturation may not halt *but* for any fact base
  the generated facts have a tree-like structure (« bounded treewidth »),
  which allows to have a finite representation by capturing regularities

# (Partial) Map of decidable cases



# Deux classes de base assurant l'arrêt du chase

Idée 1 : s'intéresser aux **positions** des termes **dans les prédicats**

- voir dans quelles positions les nouvelles variables se créent
- suivre leur « propagation » dans les différentes positions

⇒ Notion de **weak-acyclicity** :
le graphe encodant cette circulation des positions ne doit pas avoir de circuit dangereux

Idée 2 : s'intéresser aux **interactions** entre règles

- déterminer si une règle peut en déclencher une autre (ou se déclencher elle-même)

⇒ Notion de **acyclic Graph of Rule Dependencies (aGRD)**
le graphe des interactions entre règles doit être sans circuit

Ces deux notions sont **incomparables** :
un ensemble de règles peut être admis par l'une mais pas par l'autre, et vice-versa

Mais on peut les **combiner**,
ce qui fournit un critère strictement plus général que chacune des deux prise séparément

# WEAK-ACYCLICITY

**Position dependency graph**

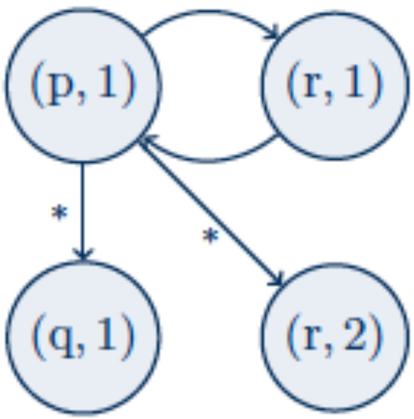
**nodes**: positions (p,i) in predicates

**edges**: for each frontier variable x in position (p,i) in a rule body

- an edge from (p,i) to each position (q,j) of x in the rule head
- a special edge from (p,i) to each position of an existential in the rule head

$\mathcal{R}$ is **weakly-acyclic** if its position graph contains no circuit with a special edge (*)

$R_1$: p(x) → ∃y r(x,y) ∧ q(y)
$R_2$: r(x,y) → p(x)




weakly acyclic

$R_1$: p(x) → ∃y ∃z r(x,y) ∧ r(y,z) ∧ r(z,x)
$R_2$: r(x,y) ∧ r(y,x) → p(x)

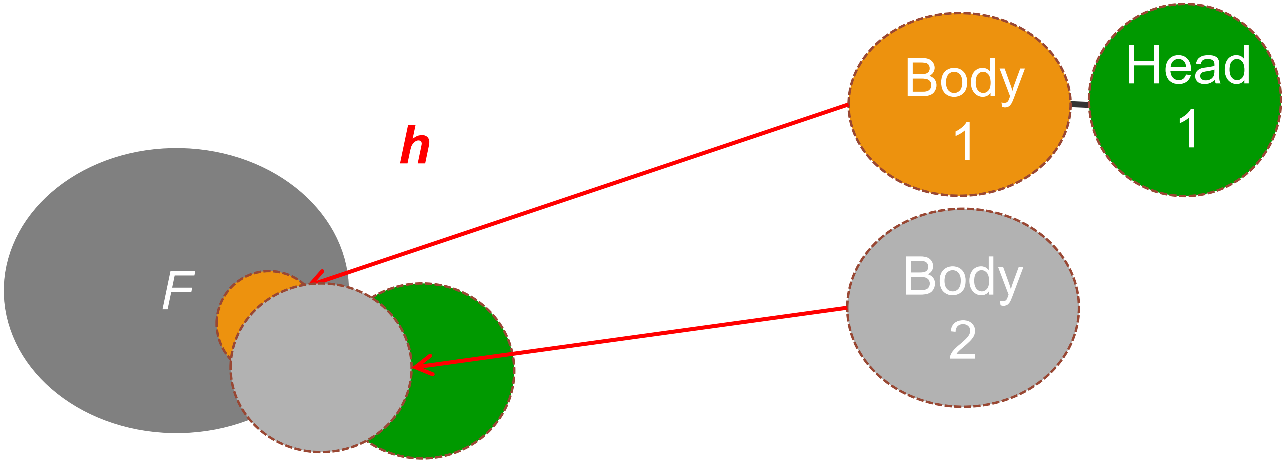
not weakly acyclic

special edge (p,1) → (r,1) due to $R_1$
edge (r,1) → (p,1) due to $R_2$

# Rule Dependency

*R2* depends on *R1* if an application of *R1* may lead to a *new* application of *R2*

i.e., there is a fact base *F* s.t. *R1* is applicable to *F* but *R2* is not and
there is an application of *R1* to *F* leading to *F'* s.t. *R2* is applicable to *F'*



# ACYCLIC GRAPH OF RULE DEPENDENCY (1)

**Graph of Rule Dependencies**

**nodes**: the rules

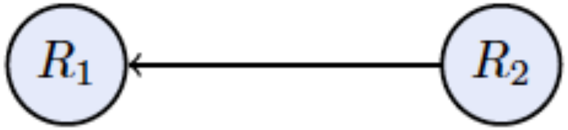
**edges**: an edge from $R_i$ to $R_j$ if an application of $R_i$ may lead to trigger a new application of $R_j$ (« $R_j$ depends on $R_i$ »)

Dependency can be effectively computed by checking if there is a **piece-unifier** of body($R_j$) and head($R_i$)

$R_1$: $p(x) \rightarrow \exists y\, r(x,y) \wedge q(y)$
$R_2$: $r(x,y) \rightarrow p(x)$

Cyclic GRD since $R_1$ and $R_2$ depend on each other (but *wa*)

$R_1$: $p(x) \rightarrow \exists y \exists z\, r(x,y) \wedge r(y,z) \wedge r(z,x)$
$R_2$: $r(x,y) \wedge r(y,x) \rightarrow p(x)$




aGRD (= GRD without circuit) (but *not wa*)

These examples show that weak-acyclicity and acyclic GRD are incomparable criteria

# Acyclic Graph of Rule Dependency (2)

- When the GRD is acyclic, the chase halts after at most k+1 breadth-first steps

  where k is the maximal length of a path in the GRD

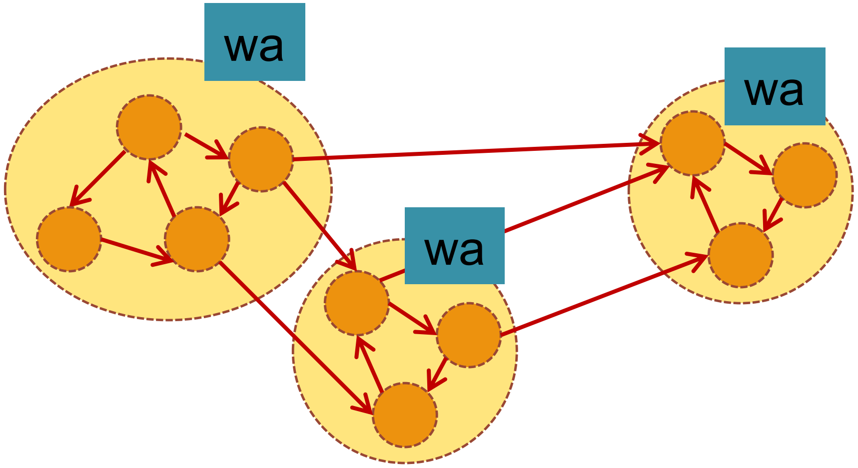
Main point: at step i > 0,
the only rules that may be applicable are those depending from a rule applied at step i-1

- With respect to chase termination,
  we can allow circuits that pass only through Datalog rules

- In other words, if each strongly connected component is Datalog,
  then the chase terminates

We can generalize this by taking weakly-acyclic instead of Datalog

# COMBINING WA AND AGRD




If all the **strongly connected components** of the GRD are **weakly-acyclic** then the chase halts on any fact base

# How to compute rule dependency in practice?

With **Datalog** rules:

R2 depends on R1 if there is a unifier of head(R1) with an atom in body(R2)

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

R1: *body*[x,y,z] → p(x,y,z,x)

R2: p(u,v,w,a) ∧ q(u,v,s) → *head* [u,s]

*where a is a constant*

Does R2 depend on R1?

Yes

$u_1$:
x → a
y → v
z → w
u → a

With existential rules, it is a bit more complex …

# TAKING INTO ACCOUNT EXISTENTIAL VARIABLES IN RULE HEADS (1)

*R1* = person(x) → ∃y hasParent(x,y)

*R2* = hasParent(v,w), dentist(w) → hasGoodTeeth(v)

Does R2 depend on R1?

Actually, no!
R1 creates a *new* parent $y_i$,
so the fact base cannot contain a fact dentist($y_i$)

(1) If *w* in body(*R2)* is unified with an existential variable of *R1*,
then all atoms in which *w* occur must be part of the unification

# Taking into account existential variables in rule heads (2)

$R1 = p(x) \rightarrow \exists z1 \exists z2\ r(x,z1), r(x,z2), r(z1,z2)$

$R2 = r(v,w), r(w,v) \rightarrow \ldots$

Does R2 depend on R1?

$u = \{x \mapsto v, z1 \mapsto w, z2 \mapsto w\}$ is not good

Actually, no!
The two variables created by R1 are distinct

(2) An existential variable of *R1* cannot be unified with another variable or constant in head(*R1)*

*In the example,*
*z1 and z2 are both unified with w, we say that they are unified together*

# PIECE-UNIFIER

Given rules R1 and R2, a **piece-unifier** ***u*** of ***B'*** $\subseteq$ *body(R2)* and **H'** $\subseteq$ head(R1)

is a substitution of *variables(B' + H')* by *terms(B'+ H')* [if x is unchanged, we write u(x) = x]

such that :

- *u(B') = u(H')*
- **existential** variables of *H'* are unified only with variables of ***B'*** that do **not** occur in (*body(R2)* \ *B'*)

  (i.e., if *z* is an existential variable from H'

  and *u(z) = u(t)*,

  then *t* is a variable of *B'* and not of (*body(R2)* \ *B'*)

This satisfies the two conditions seen before:

(1) If *w* in body(*R2)* is unified with an existential variable of *R1*, then all atoms in which *w* occur must be part of the unification

(2) An existential variable of *R1* cannot be unified with another variable or constant in head(*R1)*