Règles Existentielles

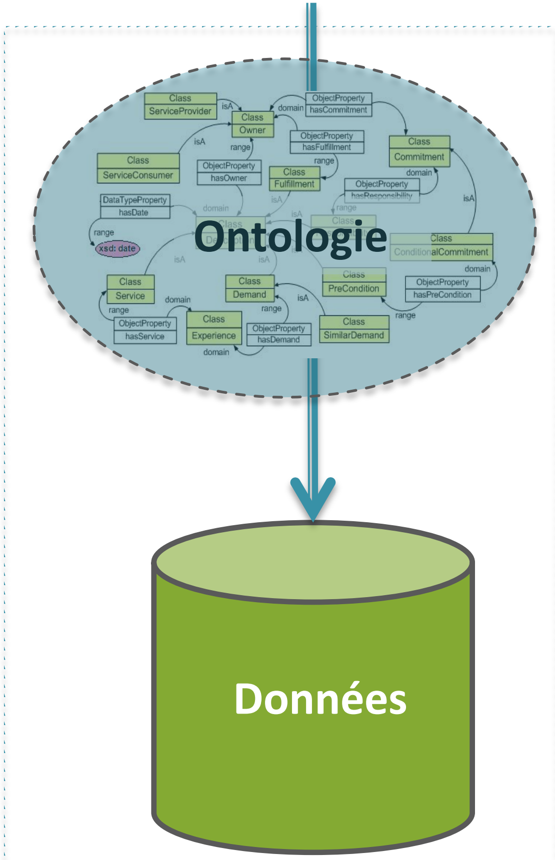
Théorie des bases de données et de connaissances

HAI933I

Cours de ML MUGNIER

RETOUR AUX BASES DE CONNAISSANCES

Requête



Base de connaissances *K*

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

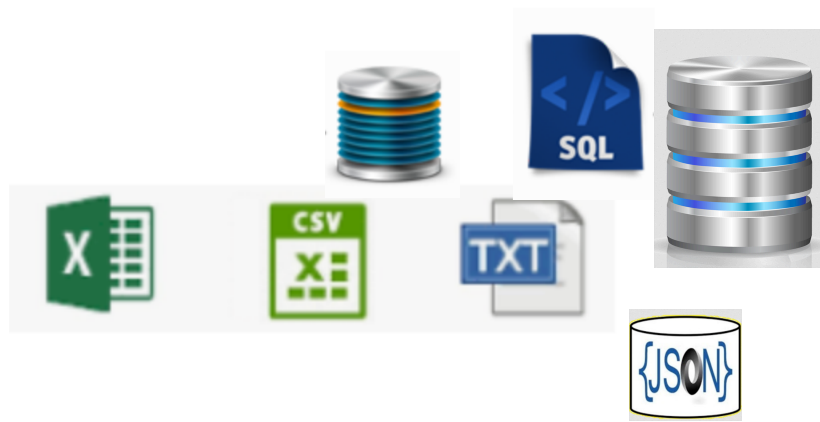
Ensemble de formules logiques, par ex :
- Règles Datalog (M1)
- Règles existentielles (extension vue ici)

Ensemble de faits :
- instanciés (M1)
- avec variables existentielles
(extension vue au cours précédent)

La réponse à une requête booléenne *q* est oui si q est **conséquence logique** de K

Example: Detecting Conflicts of Interest

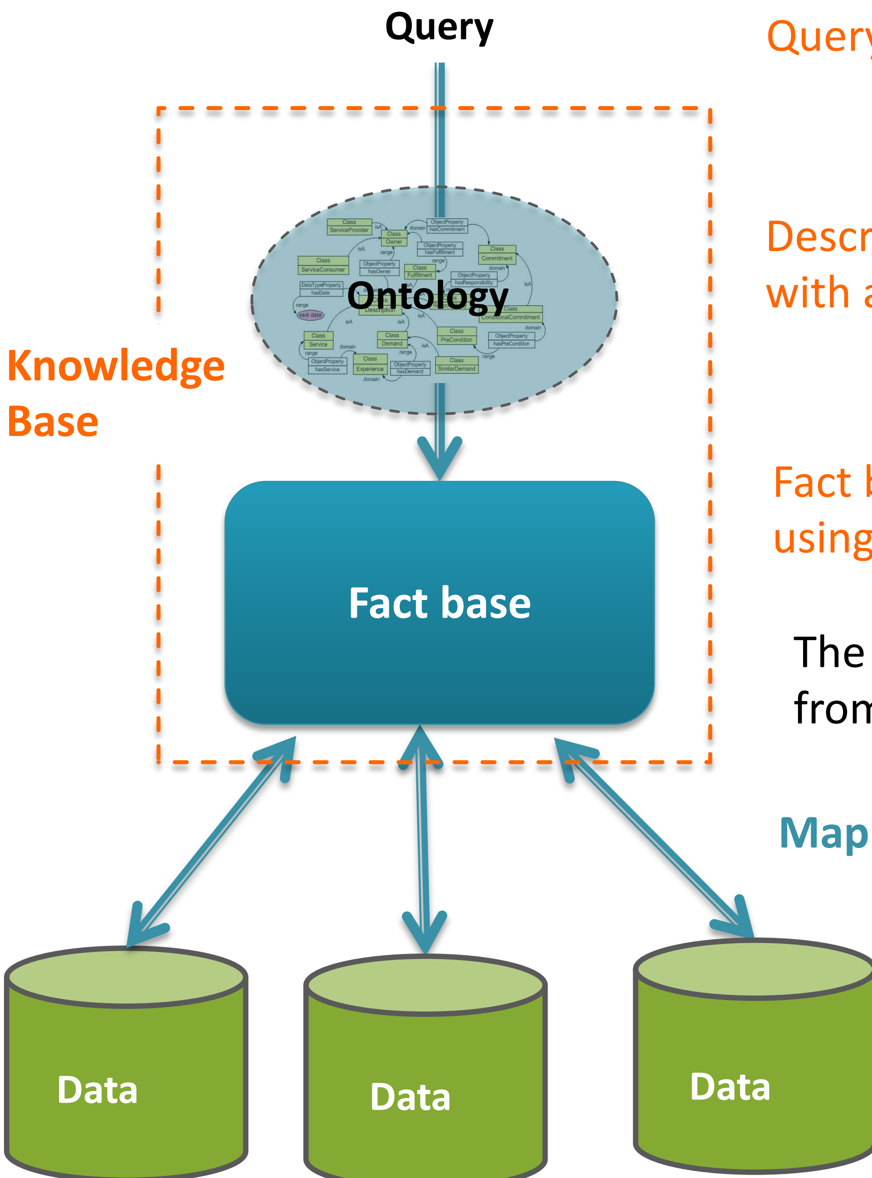
- Problem: detect links between scientific experts involved in public interest studies and industry that are likely to constitute a conflict of interest



Data on the scientific commissions and experts
Data on companies and their products
Data on research laboratories, scientific projects and their funding
Data on lobbies declarations: members, participants to meetings
Data on scientific publications
.....

Dream query: ''Find all persons **x**, studies **y** and companies **z** such that **x** has a conflict of interest for **y** because of its relationships with **z**''

ONTOLOGY-BASED DATA ACCESS



Query using the vocabulary of the ontology

Description of the application domain
with a high abstraction level

Fact base (possibly virtual)
using the vocabulary of the ontology

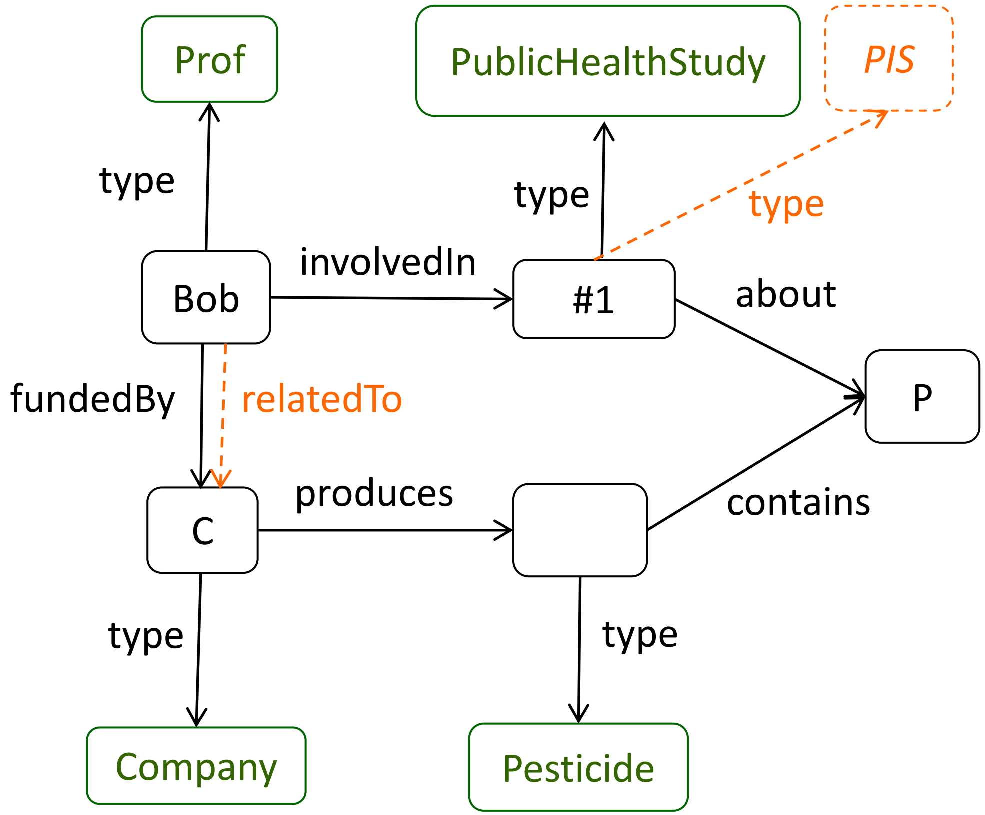
The **answers** to the query are **entailed**
from the knowledge base

Mappings from data to facts

{ Database query $\rightsquigarrow$ Facts }

Independent and heterogeneous
data sources

ASSUME WE HAVE BUILT A FACT BASE (« KNOWLEDGE GRAPH »)



Facts

∃x (
Prof(Bob) ∧
PHS(#1) ∧
Comp(C) ∧
Pest(x) ∧
involvedIn(Bob,#1) ∧
fundedBy(Bob,C) ∧
about(#1,P) ∧
produces(C,x) ∧
contains(x,P))

+ Basic ontological knowledge

PublicHealthStudy **subclass of** PublicInterestStudy
fundedBy **subproperty of** relatedTo

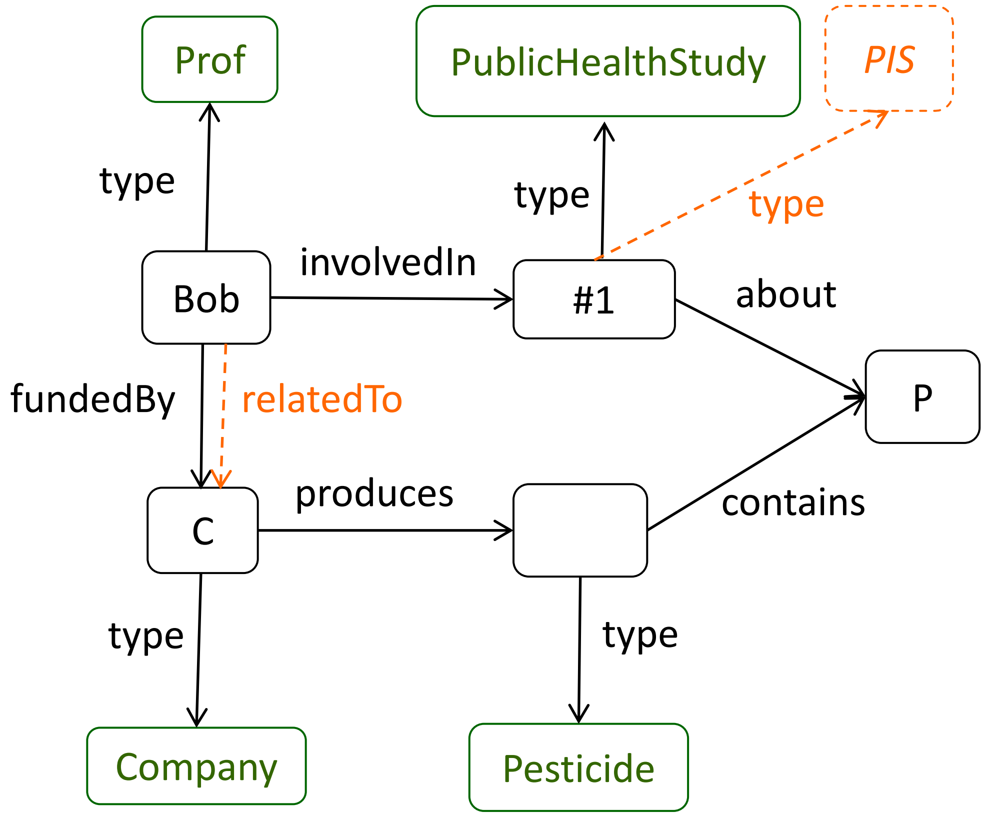
Rules

∀x (PHS(x) → PIS(x))
∀x∀y (fundedBy(x,y) → relatedTo(x,y))

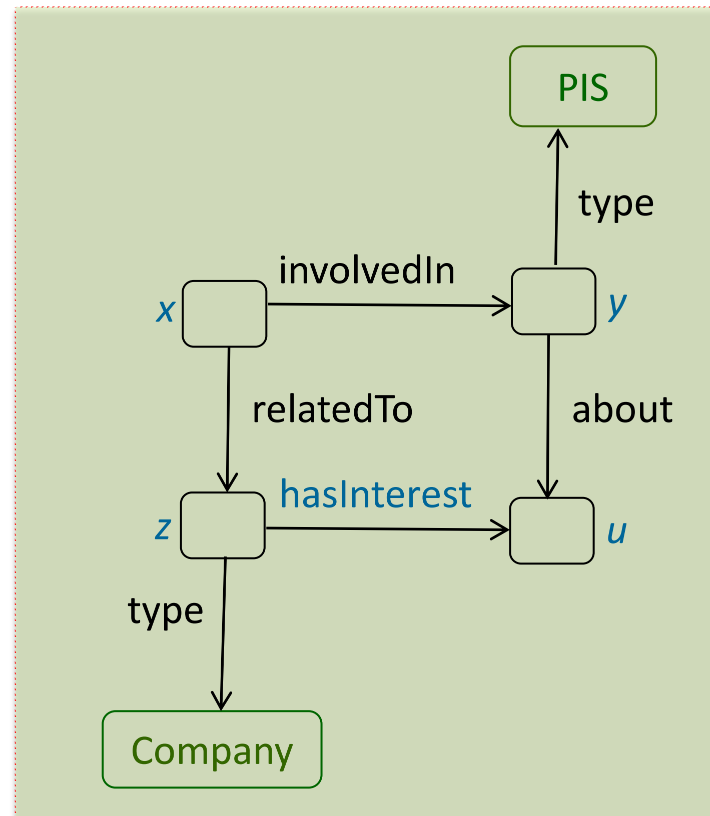
Allow to infer:

PIS(#1), relatedTo(Bob,C)

How to Infer Conflicts of Interest (C.o.I.) ?



What kind of **ontological knowledge** would allow to infer conflicts of interest?

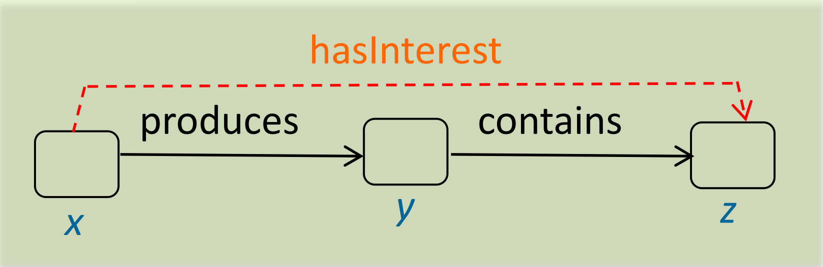


C.o.I. pattern

Query: "Find all **x, y, z** such that **x** has a conflict for study **y**
because of its relationships with company **z**"

Defining Conflicts of Interest

R_1: $\forall x \forall y \forall z$ (produces(x,y) $\wedge$ contains(y,z)
$\rightarrow$ hasInterest(x,z))

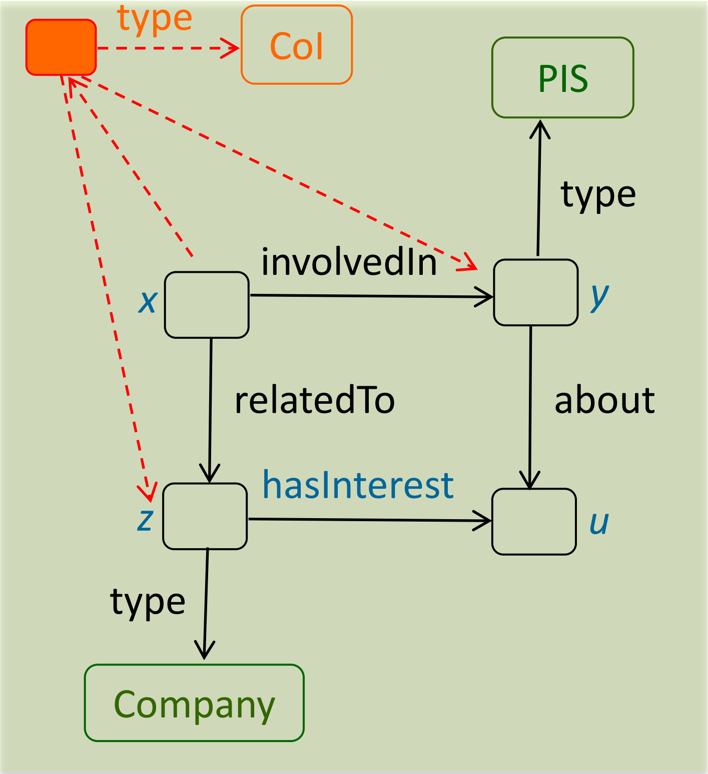


R_2: $\forall x \forall y \forall z \forall u$ (involvedIn(x,y) $\wedge$ PIS(y) $\wedge$ about(y,u) $\wedge$ relatedTo(x,z) $\wedge$ Company(z) $\wedge$ hasInterest(z,u)
$\rightarrow$ CoI(x,y,z))

What if we only have unary and binary predicates
i.e. graphs and not hypergraphs ?

Reification: new object of type CoI

R_2: $\forall x \forall y \forall z \forall u$ (*body[x,y,z,u]* $\rightarrow \exists o$
(CoI(o) $\wedge$ in(x,o) $\wedge$ on(o,y) $\wedge$ with(o,z))
)



CREATING NEW OBJECTS

R_2: $\forall x \forall y \forall z \forall u$ ($body[x,y,z,u] \rightarrow \exists o\ (CoI(o) \wedge in(x,o) \wedge on(o,y) \wedge with(o,z))$)

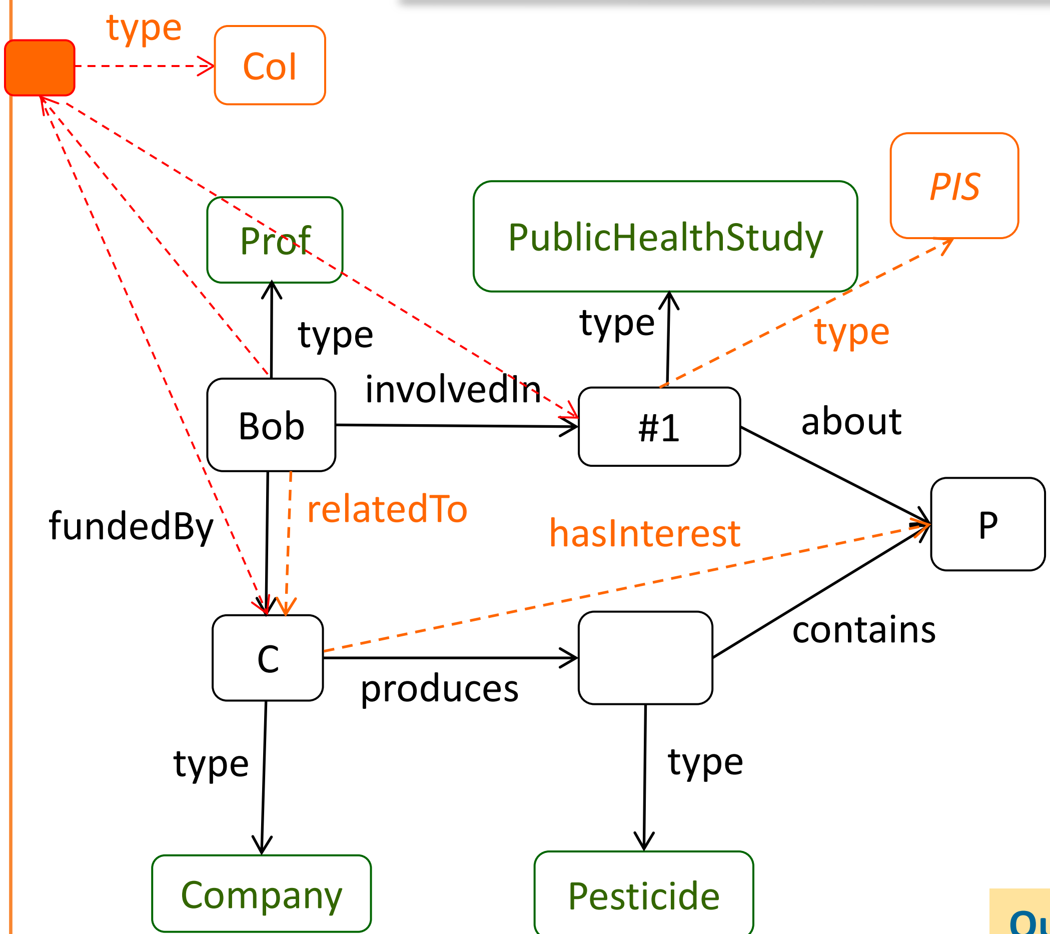
Interest of creating a new object:

- Flexible description of C.o.I. instead of a fixed arity predicate
  Not all C.o.I. need to be described by the same properties

- Ability to talk about C.o.I. because they become objects (reification)

E.g. R_3: $\forall x \forall z\ (CoI(x) \wedge with(x,z) \wedge ChemicalCompany(z) \rightarrow toBeInvestigated(x))$

Inferring Conflicts of Interest



Facts

Prof(Bob), PHS(#1), Comp(C), Pest(x)
involvedIn(Bob,#1), fundedBy(Bob,C)
about(#1,P), produces(C,x), contains(x,P)

Rules (universal quantifiers omitted)

PHS(x) → PIS(x)
fundedBy(x,y) → relatedTo(x,y)

R_1: produces(x,y) ∧ contains(y,z)
→ hasInterest(x,z)

R_2: involvedIn(x,y) ∧ PIS(y) ∧ about(y,u) ∧
relatedTo(x,z) ∧ Company(z) ∧
hasInterest(z,u)
→ ∃o CoI(o) ∧ in(x,o) ∧ on(o,y) ∧ with(o,z)

Inferred facts

PIS(#1), relatedTo(Bob,C), hasInterest(C,P)
CoI(o_1), in(Bob,o_1), on(o_1,#1), with(o_1,C)

Query: find (x,y,z) such that
∃o CoI(o) ∧ in(x,o) ∧ on(o,y) ∧ with(o,z)

Answer: (Bob,#1,C)

Existential rules

$$\forall X \, \forall Y \, (\underbrace{\text{Body}\,[X,Y]} \rightarrow \exists Z \, \underbrace{\text{Head}\,[X,Z]})$$

X, Y, Z : sets *of variables* (possibly empty)

any **positive conjunction** (without functional symbols)

$\forall x$ (actor(x) → $\exists z$ (movie(z) ∧ play(x,z))

$\forall x \, \forall y$ (siblingOf(x,y) → $\exists z$ (parentOf(z,x) ∧ parentOf(z,y)))

Key point: ability to assert **the existence of unknown entities**

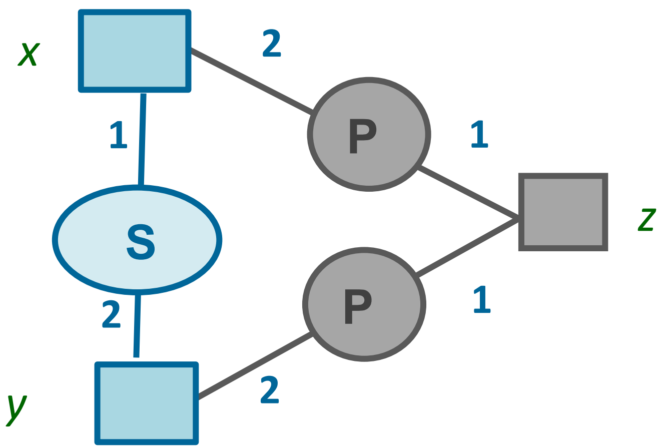
Crucial for representing ontological knowledge in « **open domains** »

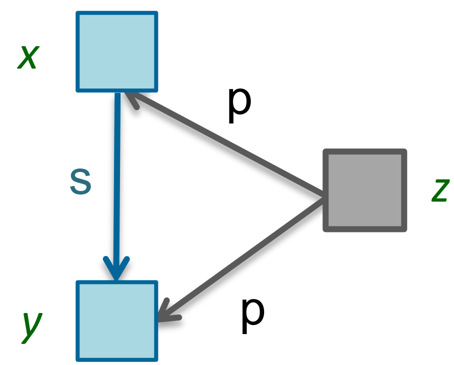
[Open domain = we do not assume that the only existing objects are those known in the factbase]

Graph View of (Existential) Rules

$$\forall X\ \forall Y\ (\ \underbrace{Body\ [X,Y]}_{\text{graph}} \rightarrow \exists\ Z\ \underbrace{Head\ [X,Z]}_{\text{graph}}\ )$$

$\forall x\ \forall y\ (\ siblingOf(x,y) \rightarrow\ \exists\ z\ (parentOf(z,x) \wedge parentOf(z,y))\ )$



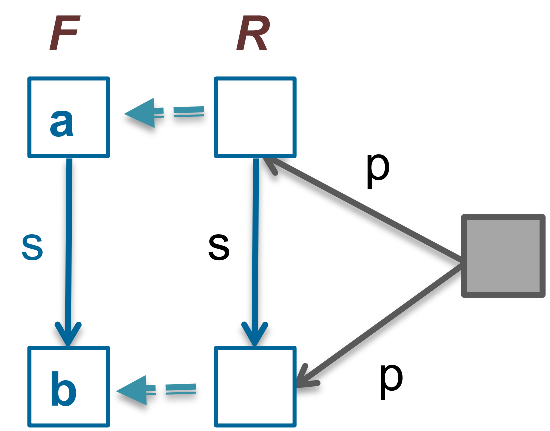


The rule head has 2 kinds of variables (or unlabelled term nodes):
- **frontier**: shared with the body **(X)** {x,y} on the example
- **existential**: **(Z)** {z} on the example

Generation of fresh (unknown) individuals

R = $\forall x \forall y$ (siblingOf(x,y) $\rightarrow$ $\exists$ z (parentOf(z,x) $\wedge$ parentOf(z,y)))

F = siblingOf(a,b)



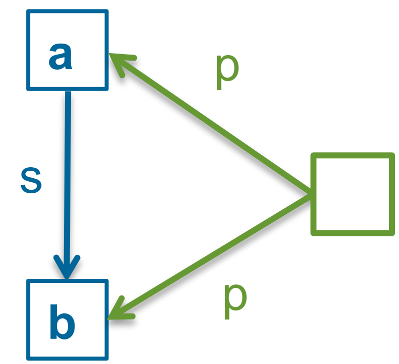
R is **applicable** to *F* if there is a **homomorphism** *h* from *body(R)* to *F*

x $\rightarrow$ a
y $\rightarrow$ b

Applying *R* to *F* w.r.t. *h* produces *F* ∪ *h(head(R))*

where a **fresh** *variable (a «null») is created for each* existential *variable in R*

F' = ∃ **z0** (siblingOf(a,b) Λ parentOf(z0,a) Λ parentOf(z0,b))



Formal notation (when needed) : **F ∪ $\mathbf{h^{safe}}$(head(R))**

where $\mathbf{h^{safe}}$ is a substitution of variables(head(R))
such that: $h^{safe}(x) = h(x)$ if x is in frontier(R)
otherwise $h^{safe}(x)$ is a fresh variable (a null)

Retour sur Datalog

- Les règles Datalog sont un cas particulier de règles existentielles

  $\forall X\ \forall Y\ (\ \text{Body}\ [X,Y] \rightarrow \exists\ Z\ \text{Head}\ [X,Z]\ )$ avec $Z = \emptyset$

- Soit une base de connaissances **K =(F,$\mathcal{R}$)** où F est une **base de faits sans variables** et $\mathcal{R}$ est un ensemble de règles **Datalog**.
  Alors:
  - K possède un **unique plus petit modèle** qui est l'**intersection** de tous ses modèles (« plus petit » au sens de l'inclusion et du cardinal)
  - Donc, étant donnée une CQ Booléenne q, pour déterminer si **K $\models$ q** il **suffit** de vérifier si le plus petit modèle de K est un modèle de q
  - Le plus petit modèle de K se calcule en **saturant** F avec $\mathcal{R}$ (« chainage avant »)

Qu'est-ce qui change quand on passe aux **règles existentielles** ?

MODÈLE CANONIQUE D'UNE BASE DE FAITS (SANS VARIABLES)

RAPPEL

Vocabulaire V = (P, C)
Base de faits *F* (sans variables) sur V

Modèle **canonique** de *F* *(ou : modèle isomorphe à F)*

M: $D_M = C$
pour tout *p* ∈ P d'arité *k*, $p^M = \{ (c_1, ..., c_k) \mid p(c_1, ...,c_k) \in F \}$

Le modèle canonique de *F* correspond à l'**intersection** de tous les modèles de *F*

$V = (\{r_{/3}, p_{/2}, q_{/1} \}, \{a, b, c, d, e\})$

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

M: $D_M = \{a,b,c,d,e\}$
$p^M = \{ (a,b), (b,c) \}$
$q^M = \{ c \}$
$r^M = \emptyset$

Qu'est-ce qui change quand la base de faits peut avoir des variables ?

Model "isomorphic" to a closed FOL(∃,∧) formula

To a closed formula *F* in FOL(∃,∧), we assign its **isomorphic model** (also called **canonical model**):

$\mathcal{M}$:

- $D^{\mathcal{M}} = \mathcal{C} \cup$ *variables(f)* *We add a domain element for each variable*
- for all *p* in $\mathcal{P}$, $p^{M} = \{(t_1 \dots t_k) \mid p(t_1 \dots t_k) \text{ in } f\}$,

$\mathcal{V} = (\{s_{/1}, p_{/2}, r_{/3}\}, \{a,b\})$

$F = \exists x \exists y \exists z\ (\ p(x,y) \wedge p(y,z) \wedge r(x,z,a)\ )$

M:
$D_M = \{a, b, x, y, z\}$
$p^M = \{\ (x,y), (y,z)\ \}$
$r^M = \{\ (x, z, a)\ \}$
$s^M = \emptyset$

Reciprocally, any **interpretation *I*** can be seen as a **closed FOL(∃,∧) formula**

Each element from $D_I \setminus \mathcal{C}$ is translated into a new variable

« Intersection des modèles » : ?

Le modèle canonique d'une base de faits F avec variables n'est plus un « plus petit modèle » au sens de l'inclusion ☹

$\mathcal{V}$ = ({$p_{/2}$}, {a,b})

F = ∃x ∃y (p(a,x) ∧ p(a,b) ∧ p(b,y))

M: D = {a, b, x, y}
p^M = { (a,x), (a,b), (b,y)}

Plus petit modèle au sens de l'inclusion ?

$D_{M'}$ = {a, b, y}
$p^{M'}$= { (a,b), (b,y) }

Ceci est dû au redondances
(F n'est pas un core)

Et ce n'est plus l'intersection des modèles de F

$\mathcal{V}$ = ({$p_{/2}$}, {a,b})

F = ∃x p(a,x)

M: D = {a, b, x}
p^M = { (a,x)}

Autres modèles de ***F*** ?

$D_{M'}$ = {a, b}
$p^{M'}$= { (a,b) }

$D_{M'}$ = {a, b}
$p^{M'}$= { (a,a) }

Etc.

MODÈLES UNIVERSELS

Le modèle canonique d'une base de faits F avec variables n'est plus un « plus petit modèle » au sens de l'inclusion, ce n'est plus non plus l'intersection des modèles de F

Mais c'est un modèle « le plus général » :

Le modèle canonique d'une formule close *F* de FOL($\exists$,$\wedge$)
est un modèle **universel** de *F :*

il s'envoie par homomorphisme dans tous les modèles de F

Conséquence :

Si un CQ *Q* est satisfaite dans un modèle universel de *F*

alors Q est satisfaite dans tous les modèles de *F*

Donc $F \models Q$

HOMOMORPHISMS AGAIN AND AGAIN

- One can define **homomorphisms** between **interpretations**

  Homomorphism *h* from $\mathcal{I}_1=(D_1, .^{\mathcal{I}1})$ to $\mathcal{I}_2 = (D_2, .^{\mathcal{I}2})$:
  mapping from D_1 to D_2 such that:

  for all *c* in $\mathcal{C}$, h(c) = c

  for all *p* in $\mathcal{P}$ and $(t_1 \dots t_k)$ in $p^{\mathcal{I}1}$, $(h(t_1) \dots h(t_k))$ in $p^{\mathcal{I}2}$

- Homomorphisms between interpretations correspond to homomorphisms between the associated fact bases

- If $\mathcal{I}_1$ maps by homomorphism to $\mathcal{I}_2$ then,
  for any *F* in FOL(∃,∧), $\mathcal{I}_1$ model of *f* ⇒ $\mathcal{I}_2$ model of *f*

  Indeed: *f* maps to $\mathcal{I}_1$ and $\mathcal{I}_1$ maps to $\mathcal{I}_2$, hence *f* maps to $\mathcal{I}_2$

NICE SEMANTIC PROPERTIES OF FOL(∃,∧)

- For any f in FOL(∃,∧), the canonical model of *f* is **universal:**
  for all *M'* model of *f*, M_f maps by homomorphism to *M'*

- ***g* ⊨ *f*** (i.e., every model of *g* is a model of *f*) iff
  M_g *is a model of **f*** (the canonical model of *g* is a model of *f*) iff
  f maps to g (there is a homomorphism from *f* to g)

Donc : pour déterminer si **F ⊨ Q** lorsque F a des variables,
on peut toujours se reposer sur l'homomorphisme

Ajoutons un ensemble $\mathcal{R}$ de règles existentielles :

- peut-on saturer F avec $\mathcal{R}$?

- le résultat correspond-il à un modèle universel de (F,$\mathcal{R}$) ?

Knowledge bases with existential rules

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

$\mathcal{R}$ is a set of **existential rules**

F is a set of **facts** (rules with an empty body): existential conjunctions of atoms

Forward chaining called « **chase** » (we still denote by F^* the result of the chase)

Main change with respect to Datalog rules: F^* can be **infinite**

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

F = person(a)

∧ hasParent(a, y0) ∧ person(y0)

∧ hasParent(y0, y1) ∧ person(y1) Etc.

but it remains a **universal model**

Hence, for Boolean CQs : **K ⊨ *q* iff *q* maps to *F****

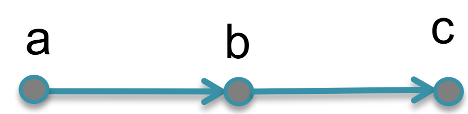
Other changes: **F* is not unique** (but all F*we will see are logically **equivalent**)

DIFFERENT VARIANTS OF THE CHASE

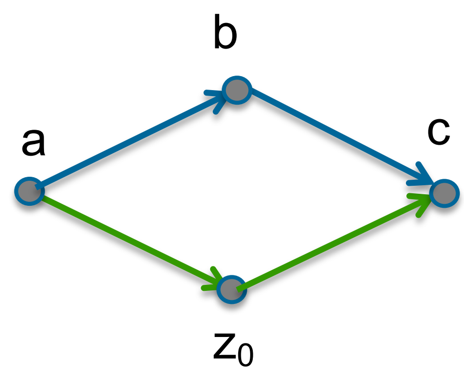
All chase variants we will see compute **universal models** of the KB

but they differ on how they handle **redundancies** possibly caused by nulls

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



p(a,b), p(b,c),
$\exists\, z_0\ p(a,z_0)\ p(z_0,c)$



$z_0 \mapsto b$

Core: set of atoms without homomorphism to one of its strict subsets

Derivation

- **Trigger** for a factbase F: **(R,h)** | h homomorphism from body(R) to F

- **Derivation**: **$(F_0 = F)\ (R_1,h_1)\ F_1\ (R_2,h_2)\ F_2$, ...**
  where for all i, (h_i, R_i) trigger for F_{i-1}
  and $F_i = F_{i-1} \cup h_i^{safe}(head(R_i))$

  When the triggers are not needed, we note **$(F_0=F), F_1, F_2, \ldots$**

- **Different chase variants** with their own rule application criteria

→ different notions of *active* trigger (R_i, h_i)

A chase variant considers only derivations with active triggers

OBLIVIOUS CHASE

Oblivious (or **naive**): « performs all rule applications according to all new triggers »

A trigger (R,h) to F_i is *active on F_i* iff this trigger has *not* already been used in the derivation from F_0 to F_{i-1}

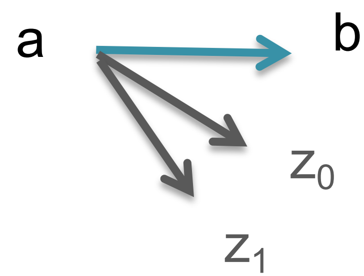
R = p(x,y) → ∃ z p(x,z)

F = p(a,b)

p(a,z_0)

p(a,z_1)

...



stupid rules to keep examples simple!

infinite derivation

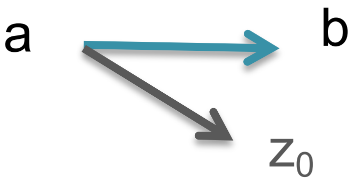
SEMI-OBLIVIOUS = SKOLEM CHASE

Semi-oblivious: consider only homomorphisms that differ on the rule frontier (x)

A trigger (R,h) to F_i is *active on F_i* iff there is *no* trigger (R,h') such that $h'(x) = h(x)$ for all x in frontier(R) in the derivation from F_0 to F_{i-1}

F = p(a,b)

$R = p(x,y) \rightarrow \exists z\ p(x,z)$



Skolem chase: similar behavior

(1) skolemize rules: in R, replace each existential variable z by a function f_R^z(frontier(R))

(2) perform the oblivious chase on skolemized rules

$R = p(x,y) \rightarrow p(x,f(x))$

p(a,b)

p(a,f(a))

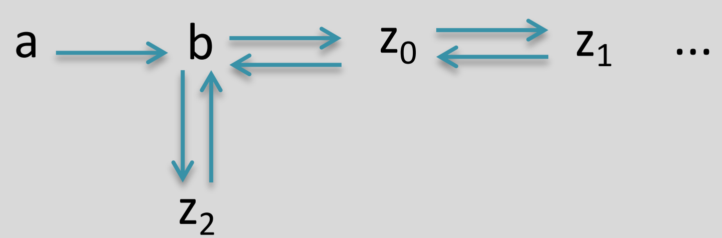
Skolemization can be seen as a way of naming existential variables and « tracking » the nulls created during the semi-oblivious chase

Restricted (also known as standard) Chase

Restricted: do not perform a rule application that brings *only* redundant information

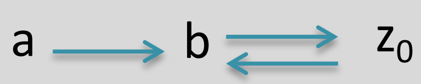
A trigger (R,h) to F_i is *active on F_i* iff
h cannot be extended to homomorphism *h'*: body ∪ head → F_i

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



(semi-) oblivious chase:

infinite



$x \mapsto b$, $y \mapsto z_0$, $z \mapsto b$

$x \mapsto z_0$, $y \mapsto b$, $z \mapsto z_0$

restricted chase:

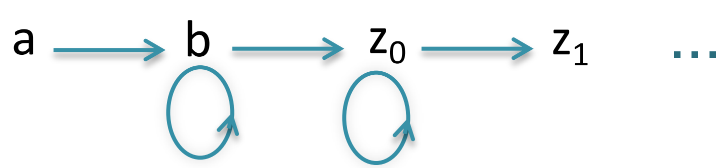
halts after one rule application

RESTRICTED CHASE: NATURAL BUT TRICKY

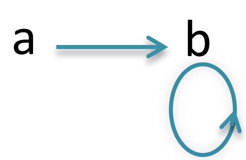
- For the same KB, some derivations may halt while others may not

F : p(a,b) R_1: $p(x,y) \rightarrow \exists z\ p(y,z)$
                        R_2: $p(x,y) \rightarrow p(y,y)$

If R_1 is always applied before R_2 for a given homomorphism of p(x,y):



If R_2 is applied first:



Core Chase

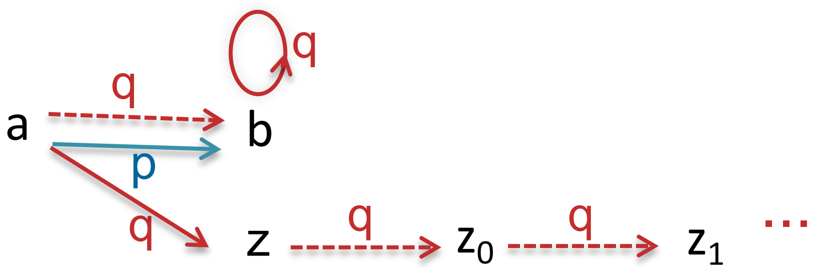
Iterate:

(1) perform a finite number of rule applications as in the restricted chase

(2) compute the core of the result

F : p(a,b), q(b,b), q(a,z) where z is a variable

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



The restricted chase only checks redundancy of **newly** added atoms ⇒ infinite here

The core chase outputs { p(a,b), q(b,b), q(a,b) }

The core chase allows to detect **global** redundancies

What does « the chase halts on this KB » mean?

- **Terminating** derivation:
  (1) finite and (2) there is no active trigger on the last factbase

- A chase derivation has to be **fair**: no active trigger is indefinitely delayed
  Formally: if (R,h) is an active trigger on F_i

  then there is F_j with $j > i$ such that F_j is obtained by applying (R,h)
  or (R,h) is not active anymore on F_j

Terminating = finite and fair

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

F= p(a,b)

unfair infinite derivation: apply only R_1 ...

(semi-) oblivious: all fair derivations are infinite

restricted: some terminating derivations,
some infinite fair derivations

core: all fair derivations are terminating

For a chase variant C, C **halts** on a KB K if all **fair** derivations on K are finite

IN SHORT

All previous chase variants compute **universal** models of a KB

They can be **strictly ordered wrt termination**:

oblivious < semi-oblivious = skolem < restricted < core

[X < Y means that: for any KB K, if X-chase halts on K then Y-chase halts on K
and there is a KB on which Y-chase halts but not X-chase]

Only the **core** chase halts if and only if the KB admits a **finite** universal model
but it is costly (involves homomorphisms from the whole factbase)

The **O**, **S-O** and **core** chases yield a **unique** result (up to the name of nulls):
all fair derivations for a given chase variant yield the same result on a given KB
but not the **R chase**: we can even have finite and infinite fair derivations

The **R chase** seems to achieve a good tradeoff
redundancy elimination / efficiency of computation (when it stops)
but its behavior is difficult to control

TRICKY RESTRICTED CHASE

Open question:
is there an ordering strategy that terminates more often than the others?

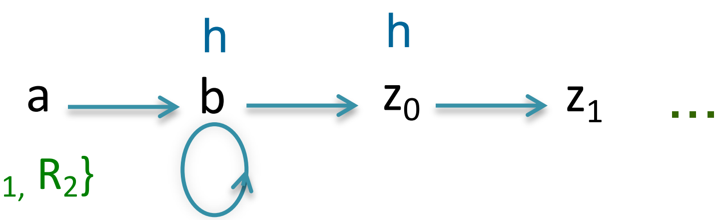
- **Breadth-first ordering** is a natural candidate (iterate:
  (1) compute all rule body homomorphisms to the current factbase,
  (2) apply all active triggers according to these homomorphisms)
- however, it is **not optimal for restricted chase** termination

R_1: $p(x,y) \rightarrow \exists z\ p(y,z)$
R_2: $p(x,y) \rightarrow h(y)$
R_3: $h(x) \rightarrow p(x,x)$

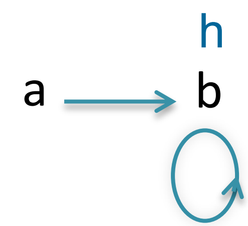
$F = p(a,b)$

$p(b,z_0), h(b)$ $\{R_1, R_2\}$

$p(z_0,z_1), h(z_0), p(b,b)$ $\{R_1, R_2, R_3\}$



Optimal order: apply R_2 then R_3 (ie delay application of R_1)



- Usual heuristic: at each step, first saturate with all datalog rules, then apply an existential rule

→ would be optimal on this example but it is not true in general