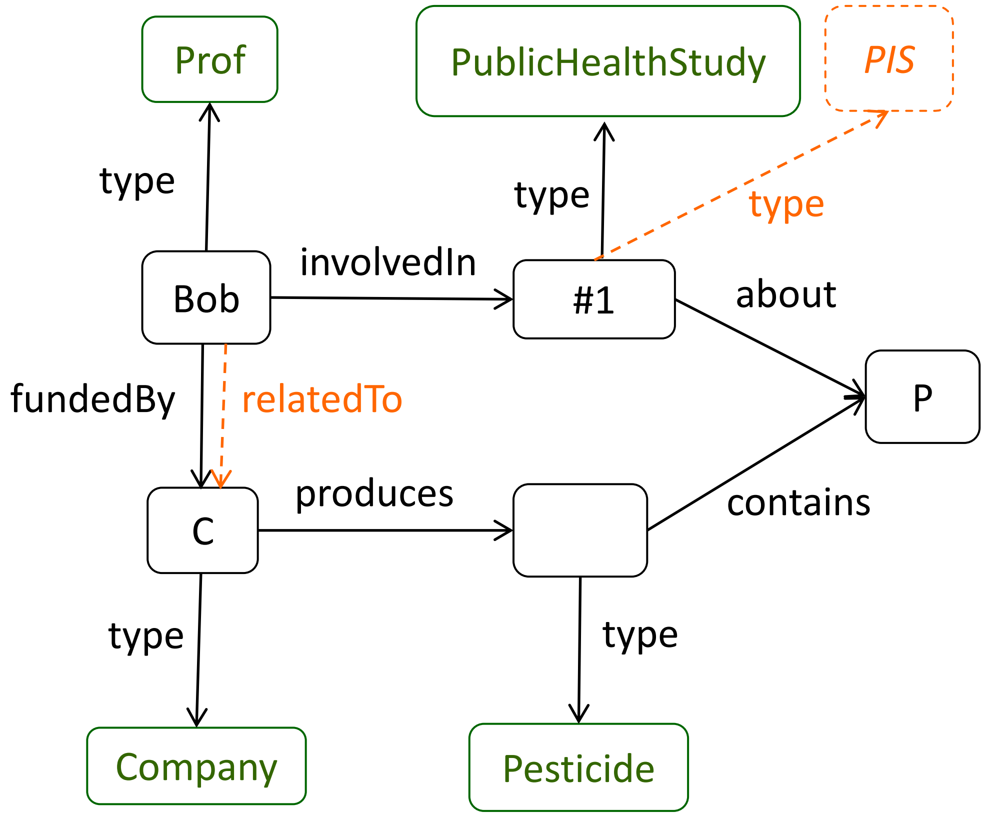
Règles Existentielles

Théorie des bases de données et de connaissances
HAI933I
Cours de ML MUGNIER

Example: Part of a ``Knowledge Graph''



+ Basic ontological knowledge

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

Facts

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

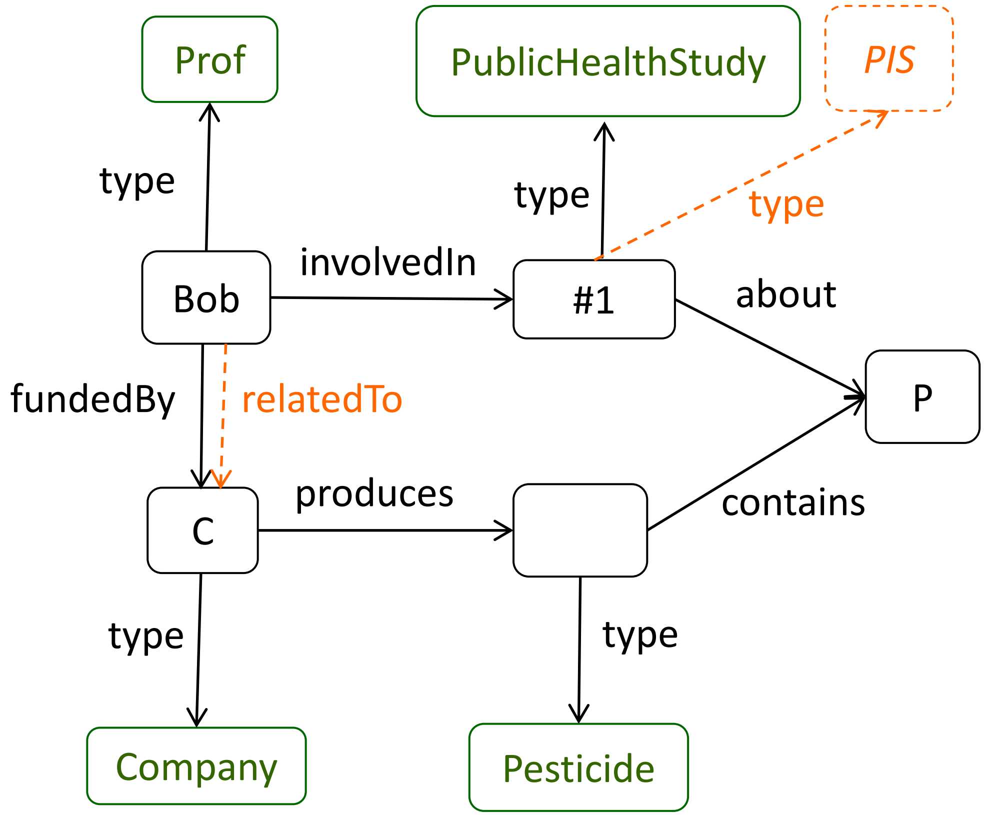
Rules

$\forall x\,(PHS(x) \rightarrow PIS(x))$
$\forall x \forall y\,(fundedBy(x,y) \rightarrow relatedTo(x,y))$

Allow to infer:

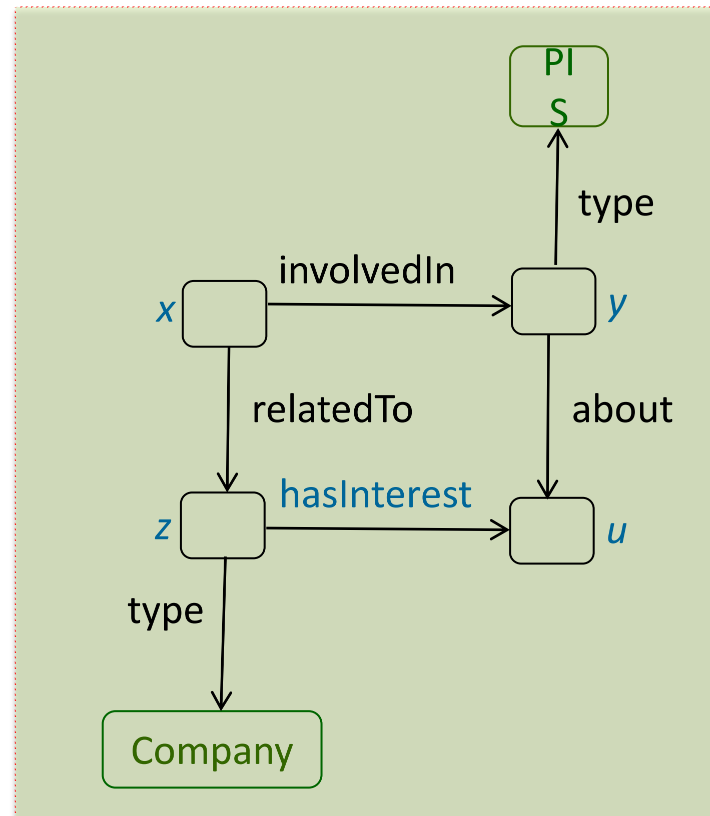
PIS(#1), relatedTo(Bob,C)

How to Infer Conflicts of Interest (CoI) ?



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

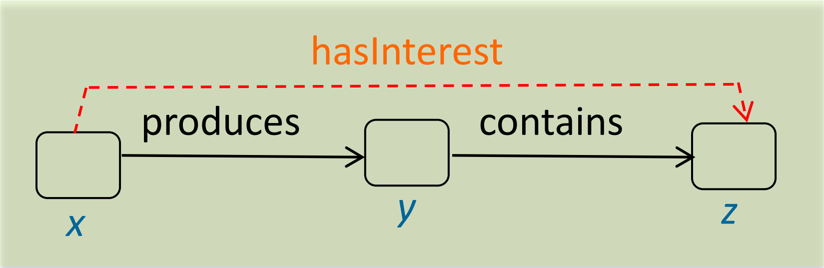
What kind of **ontological knowledge** would allow to infer conflicts of interest?



CoI pattern

DEFINING CONFLICTS OF INTEREST

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

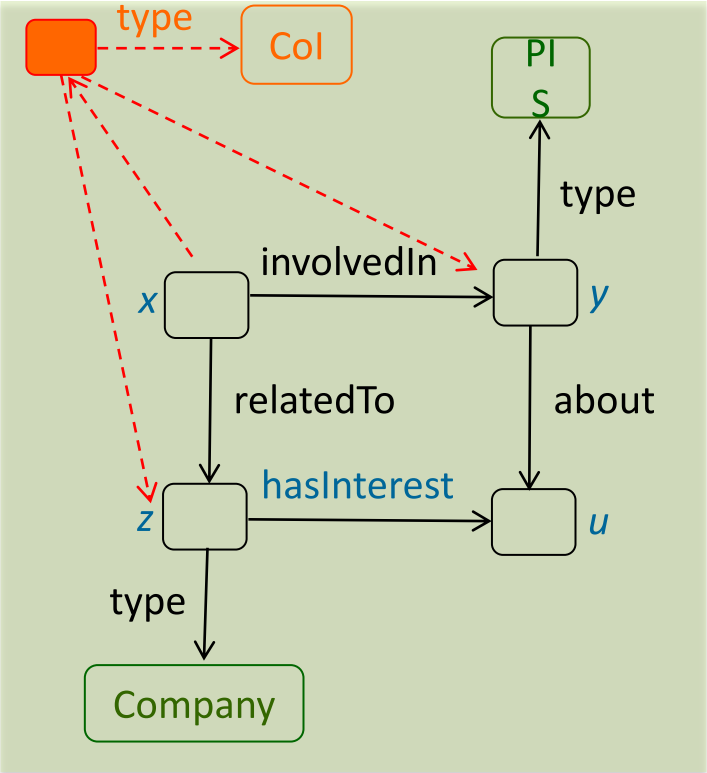


R_2: ∀x∀y∀z∀u (involvedIn(x,y) ∧ PIS(y) ∧ about(y,u) ∧ relatedTo(x,z) ∧ Company(z) ∧ hasInterest(z,u)
→ CoI(x,y,z))

What if we only have unary and binary predicates
ie graphs and not hypergraphs ?

Reification: new object of type CoI

R_2: ∀x∀y∀z∀u (*body[x,y,z,u]* → ∃o
(CoI(o) ∧ in(x,o) ∧ on(o,y) ∧ with(o,z))
)



CREATING NEW OBJECTS

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

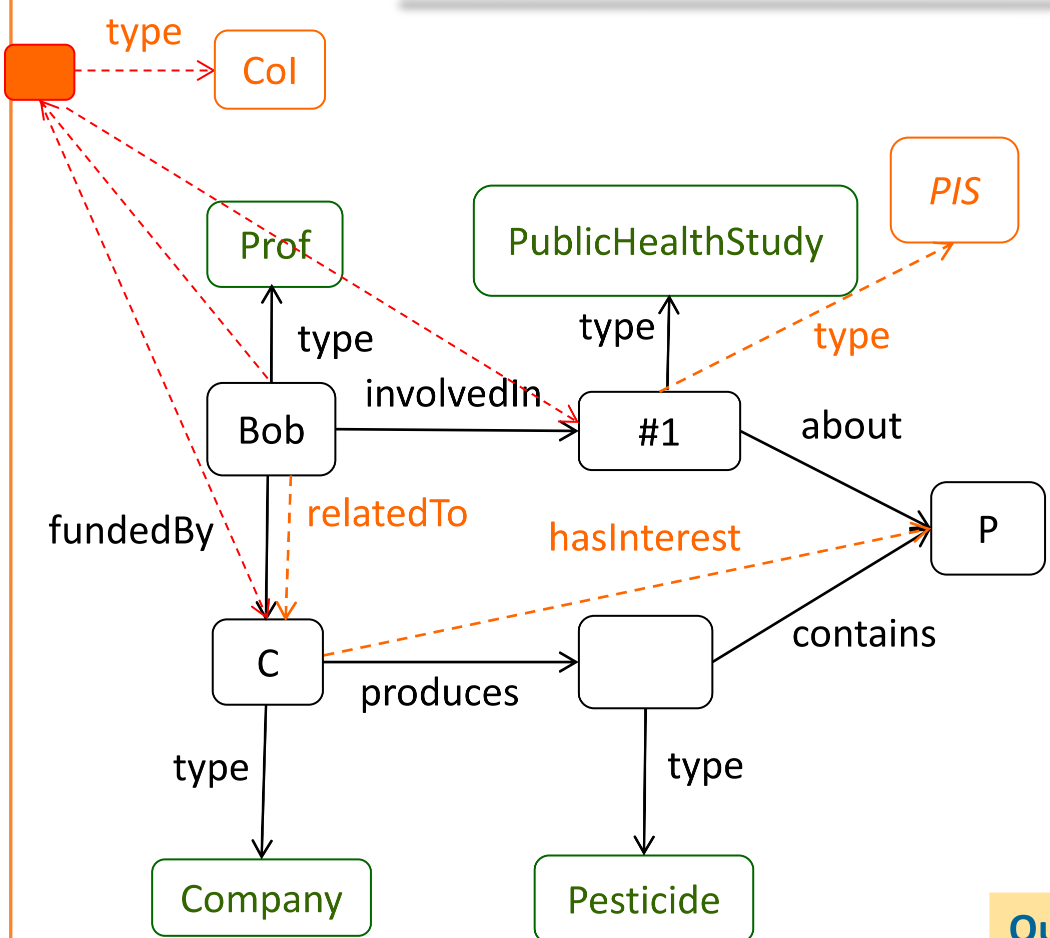
Interest of creating a new object:

- Flexible description of CoI instead of a fixed arity predicate
  Not all CoI need to be described by the same properties

- Ability to talk about CoI because they become objects (reification)

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

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 :
(possibly empty) sets *of variables*

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

$\forall x\ (\ \text{actor}(x) \rightarrow \exists z\ (\text{movie}(z) \wedge \text{play}(x,z)\ )$

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

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

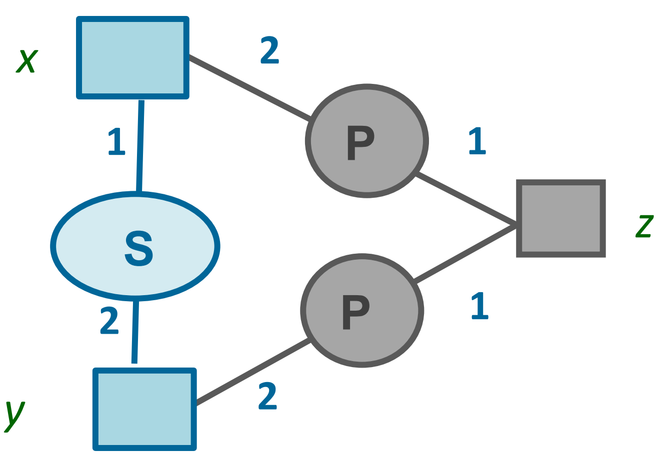
Crucial for representing ontological knowledge in « **open domains »**

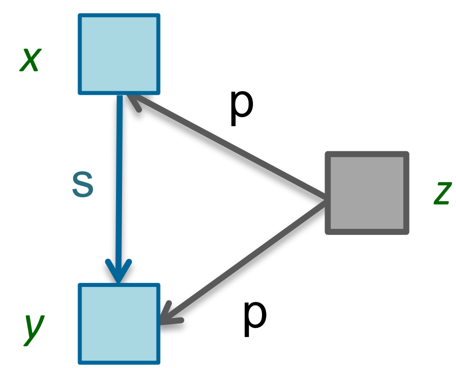
[Open domain: we do not assume that the only existing objects are those known in the factbase]

Graph View of (Existential) Rules

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

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





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

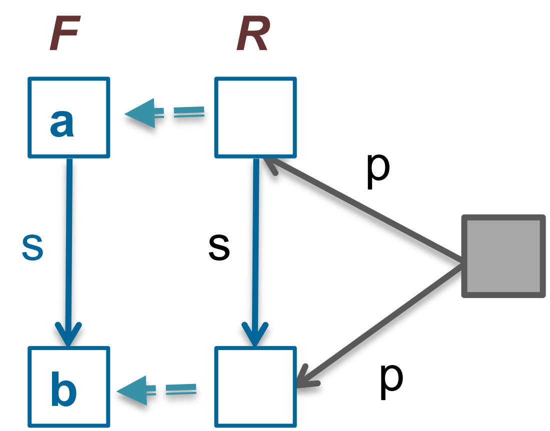
GENERATION OF FRESH (UNKNOWN) INDIVIDUALS

R = ∀x ∀y (siblingOf(x,y) → ∃ z (parentOf(z,x) ∧ parentOf(z,y)))

F = siblingOf(a,b)

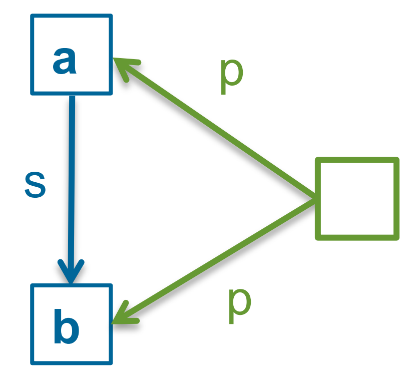
R is **applicable** to *F* if there is a **homomorphism** *h* from *body(R)* to *F*

x → a
y → 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) ∧ parentOf(z0,a) ∧ parentOf(z0,b))



Notation (when needed) : **F ∪ h^{safe}(head(R))**

where **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

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

- 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
  - Donc, étant donnée une CQ Booléenne q, pour déterminer si **K ⊨ 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 $\mathcal{V} = (\mathcal{P}, \mathcal{C})$
Base de faits *F* (sans variables) sur $\mathcal{V}$

Modèle **canonique** de *F*

$\mathcal{M}$: $D^{\mathcal{M}} = \mathcal{C}$
pour tout $p \in \mathcal{P}$ d'arité k, $p^{\mathcal{M}} = \{ (c_1, \ldots, c_k) \mid p(c_1, \ldots, c_k) \in F \}$

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

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

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

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

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**):

$\mathcal{M}$:

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

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

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

M_f: $D = \{a, b, x, y, z\}$
$p^{Mf} = \{ (x,y), (y,z) \}$
$r^{Mf} = \{ (x, z, a) \}$
$s^{Mf} = \emptyset$

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

Modèles Universels

Le modèle canonique d'une base de faits avec variables n'est plus un « plus petit modèle » ☹

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

$f = \exists x \exists y \exists z\, (p(x,y) \wedge p(y,z))$

M_f: $D = \{a, b, x, y, z\}$
$p^{Mf} = \{ (x,y), (y,z)\}$

Quels plus petits modèles de f ?

$D = \{a, b\}$
$p^M = \{ p(a,a) \}$

ou $D = \{a, b\}$
$p^M = \{ p(b,b) \}$

ou $D = \{a, b,x\}$
$p^M = \{ p(x,x) \}$

D'ailleurs, il n'y a pas d'unique plus petit modèle ☹

Mais ...

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

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 factbases

- If $\mathcal{I}_1$ maps by homomorhism to $\mathcal{I}_2$ then,
  for any *f* in FOL($\exists,\wedge$), $\mathcal{I}_1$ model of *f* $\Rightarrow$ $\mathcal{I}_2$ 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*****

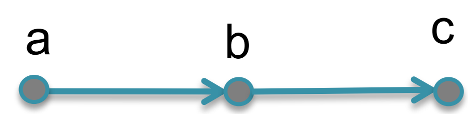
Other changes: **F* is not unique** (but all F*we will see are logically **equivalent**)

DIFFERENT VARIANTS OF THE CHASE

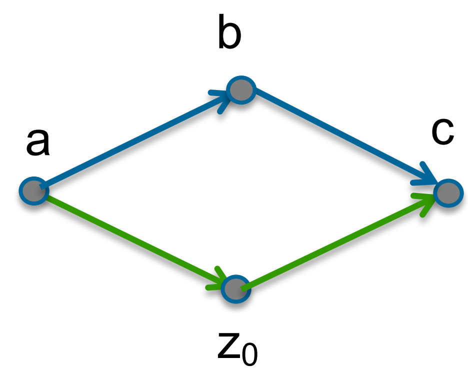
All chase variants we will see compute **universal models** of the KB

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**: $(F_0 = F)\ (R_1,h_1)\ F_1\ (R_2,h_2)\ F_2\ , \ldots$
  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}

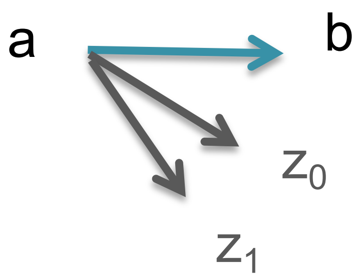
R = p(x,y) → ∃ z p(x,z)

F = p(a,b)

p(a,z_0)

p(a,z_1)

...



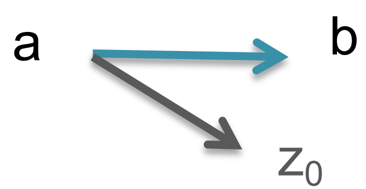
stupid rules to keep examples simple!

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) → ∃ 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) → p(x,f(x))

p(a,b)

p(a,f(a))

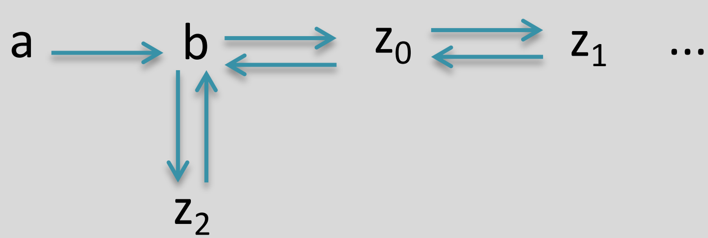
Skolemization can be seen as a way of naming existential variables and « tracking » the nulls created during the semi-oblivious chase

RESTRICTED (ALSO KNOWN AS STANDARD) CHASE

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

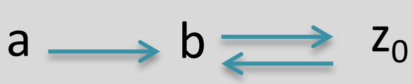
A trigger (R,h) to F_i is *active on F_i* iff
h *cannot* be extended to homomorphism *h'*: body ∪ head → F_i

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



(semi-) oblivious chase:

infinite



$$\boxed{\begin{array}{l} x \mapsto b \\ y \mapsto z_0 \end{array}} \quad z \mapsto b$$

$$\boxed{\begin{array}{l} x \mapsto z_0 \\ y \mapsto b \end{array}} \quad z \mapsto z_0$$

restricted chase:

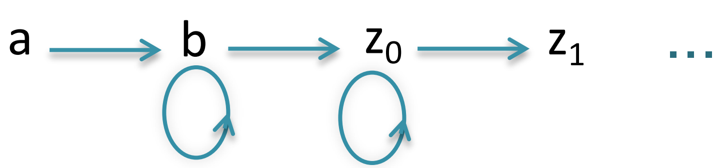
halts after one rule application

RESTRICTED CHASE: NATURAL BUT TRICKY

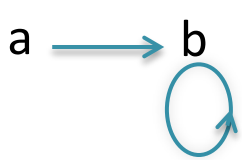
- For the same KB, some derivations may halt while others may not

F : p(a,b) R_1: p(x,y) → ∃z p(y,z)
R_2: p(x,y) → 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

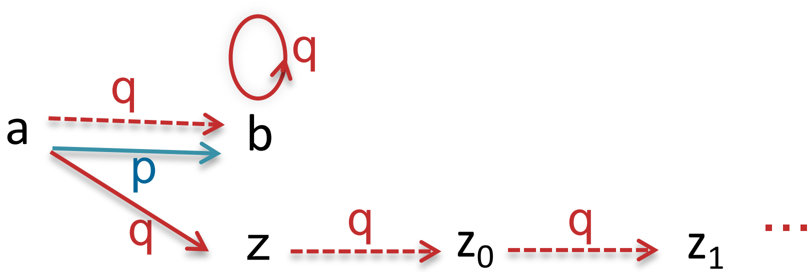
Iterate:

(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

WHEN DOES A CHASE HALT?

- **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?

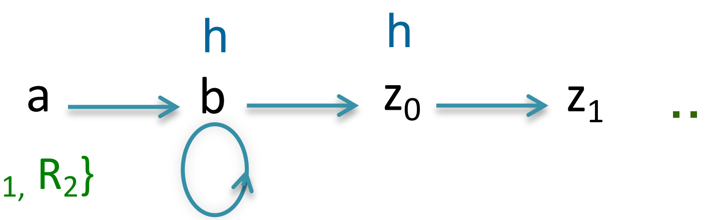
- **Breadth-first ordering** is a natural candidate (iterate:
  (1) compute all rule body homomorphisms to the current factbase,
  (2) apply all active triggers according to these homomorphisms)
- however, it is **not optimal for restricted chase** termination

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

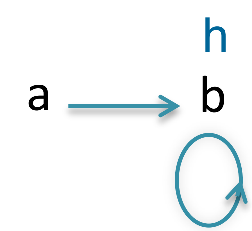
$F = p(a,b)$

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

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



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, is it always the case?