Datalog$^{\pm}$: Its Languages and Evaluation Issues

Leopoldo Bertossi[1] & Mostafa Milani

Carleton University
Ottawa, Canada
(Faculty Fellow IBM CAS)

[1]Prepared while at LogicBlox, Atlanta, 2014.

Datalog$\pm$ as an Ontological Framework

Datalog$\pm$ is a family of extensions of classic Datalog, with new kinds of rules and constraints

Its languages allow to represent ontological axioms and integrity constraints that can not be expressed in Datalog

The idea is to extend Datalog with new constructs to gain expressive power

While trying to keep the good properties of Datalog:

$\longrightarrow$ declarativity, clear logical semantics, effectiveness & efficiency

(as extensions of whatever available for Datalog)

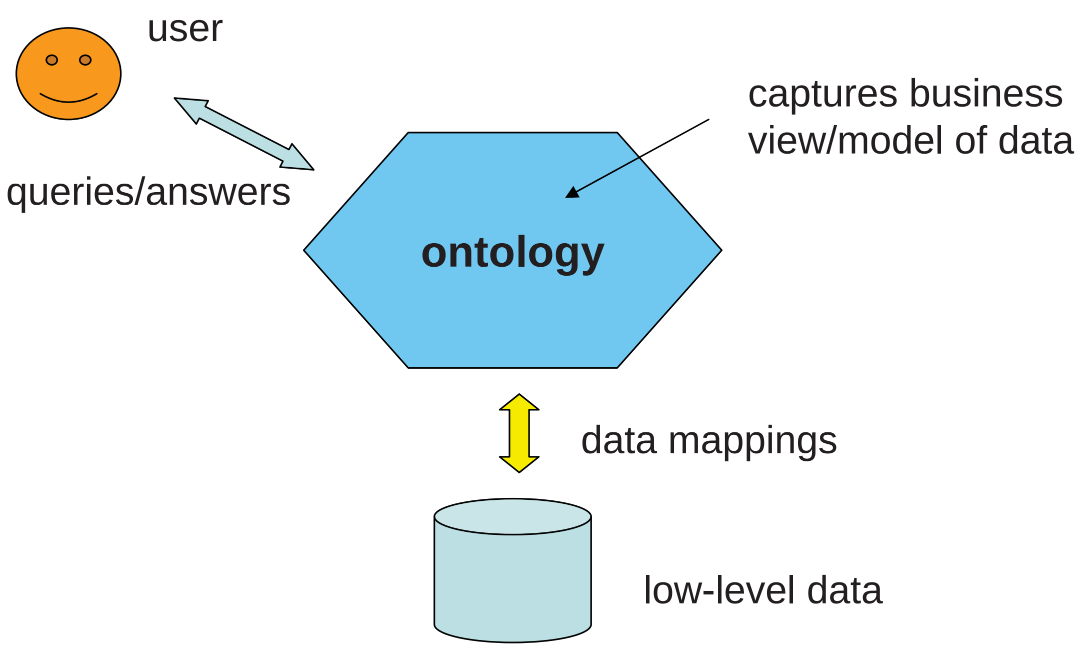
Applications

- Express/represent ontologies that interact with data sources
- Represent conceptual data models, and semantic layers on top of databases
- Ontology-Based Data Access (OBDA)
  - Query a database through the ontology
  - In the language of the ontology (better understood by- and closer to the user)
  - Automatically access the underlying data sources
  - Get answers through Datalog evaluation
- Datalog± ontologies can represent: ER, Semantic Web languages, UML with object classes, ... (classic Datalog not!)
- (Extended to Probabilistic Datalog± via MLNs)

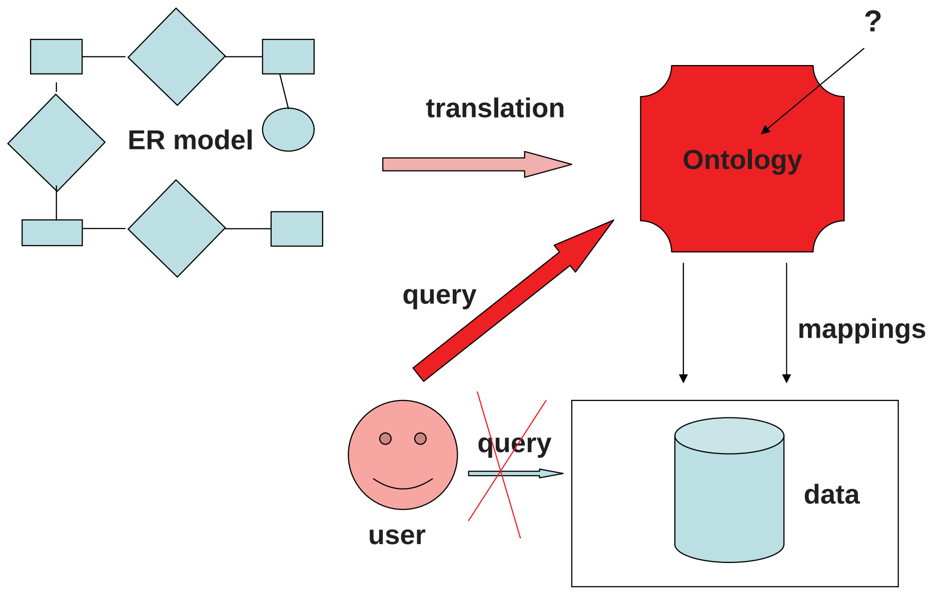
Ontology-Based Data Access:

Data stay underneath, and may not represent the way the user sees the business

Ontology is metadata or an explicit, formal ER model, etc.



Integration of data management with higher-level reasoning systems: intelligent information systems, knowledge bases, ontologies, semantic web, etc.



ER model can be seen as an extra, semantic layer

ER model closer to the user

Use ER to interact with DB

Higher-level layer has mapping to the DB, for data extraction and query answering

ER is replaced by (reconstructed as) a symbolic *ontology*, i.e. a logic-based knowledge base describing concepts and relationships among them

The (reconstructed) ER model can be computationally processed, e.g. for automated reasoning

More Specifically

A Datalog$\pm$ program with a new kind of rules and classical ones is combined with an extensional database (EDB)

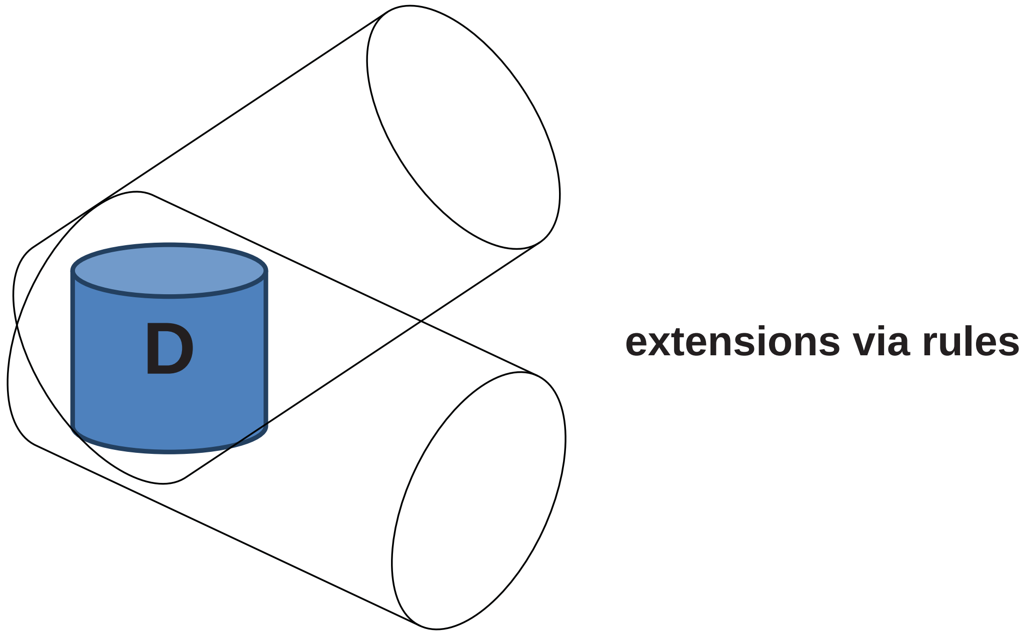
EDB is considered to be incomplete, but extended through the Datalog$\pm$ programs

Generating new tuples for EDB predicates and full extensions for intensional predicates

Depending on the kind of rules, possibly several extensions

We may want to materialize the extension(s) or keep them virtual

And query them ...



Extensions are DBs that extend the EDB and satisfy the rules as classical logical formulas

The chase (of the rules on the EDB) generates an instance that extends the EDB and "represents" the whole class of extensions

Whatever is *true in all extensions*, i.e. *certain*, is true in (the extension produced by) the chase, and viceversa

Most prominent new ingredients:

- Rules in Datalog± admit existentially quantified variables:

  $$\exists x P(x,y) \ \leftarrow \ R(y,z)$$

  Can be seen as tuple-generating dependencies (TGDs)

- Negative Constraints (NCs): (in particular, denial constraints)

  $$\perp \ \leftarrow \ P(x,y), R(y,z)$$

- Equality generating dependencies (EGDs):

  $$y = z \ \leftarrow \ P(x,y), P(x,z)$$

  In this case a key constraint (KC)

Main Issue

TGDs when applied forward (as usual in Datalog), with value invention for existentials, may create non-terminating loops

This is the main part of the "chase procedure"

So, the chase may not terminate

Query answering may become undecidable

Idea: impose syntactic restrictions of Datalog$\pm$ programs

To guarantee decidability of query answering

And hopefully efficient query answering ...

Example: An incomplete EDB D of employers and employees

• Impose on D the TGD (usually as an inclusion dependency): *"every manager is an employee"*

Expressed by a Datalog rule: $employee(x) \leftarrow manager(x)$

• Another TGD: *"every manager supervises someone"*

As a rule in Datalog±: $\exists y \; supervises(x, y) \leftarrow manager(x)$

• Impose IC: *"employees are not employers"*

As a negative constraint (NC): $\bot \leftarrow employee(x), employer(x)$

• An EGD: *"every employee is supervised by at most one manager"*

$$x = x' \leftarrow supervises(x, y), supervises(x', y)$$

Chase and Termination

We may reserve the term Datalog$\pm$ for the "good" extensions of Datalog

Each of those extensions (at least the TGD part) can be seen as a syntactic fragment of Datalog[$\exists$], the extension of Datalog with unrestricted existential rules

Query answering under Datalog[$\exists$] is undecidable

Related to (but not necessarily implied by) the fact that ...

The chase procedure for Datalog[$\exists$] may not terminate, i.e. it produces an infinite extension

Finite or infinite, we can query it ...

Example: Incomplete EDB $D = \{person(John)\}$

Set Σ of Datalog[$\exists$] rules:

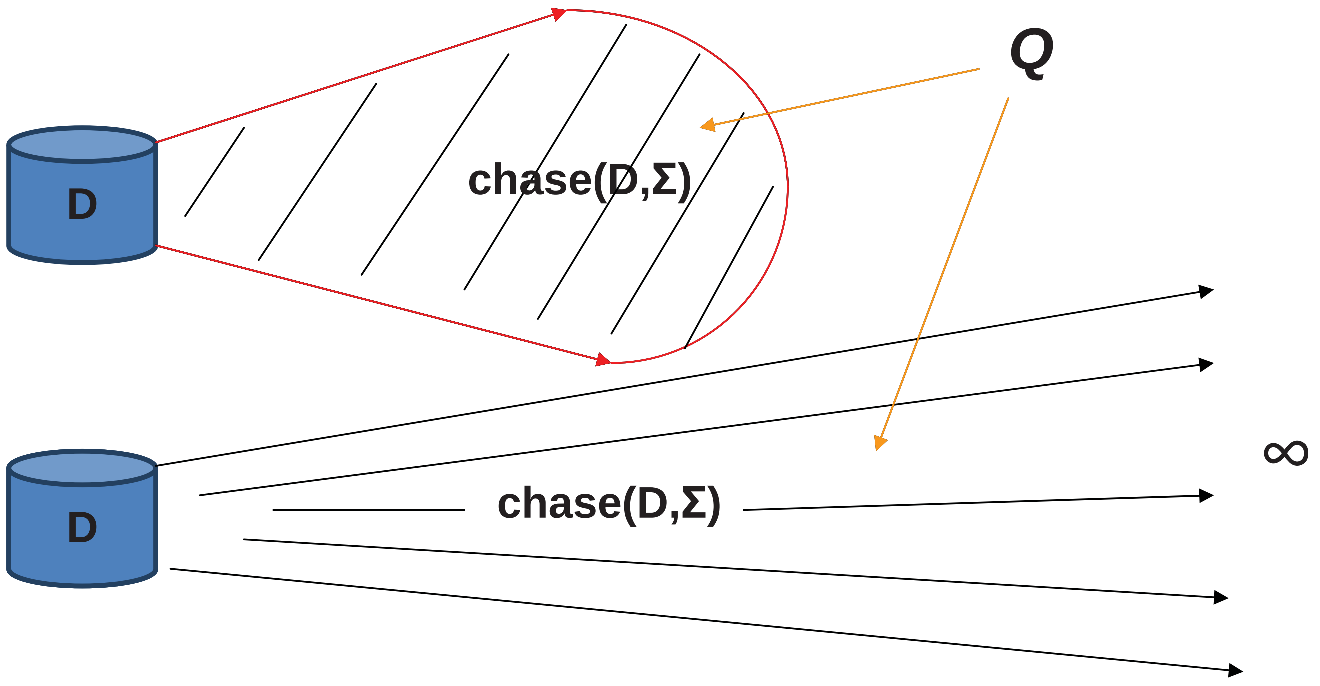
$$\begin{aligned} \exists x\ father(x,y) &\leftarrow person(y) \\ person(x) &\leftarrow father(x,y) \end{aligned}$$

The chase is a procedure that applies the TGDs in a forward manner, generating new tuples

$$\begin{aligned} chase(D,\Sigma) = \{&father(z_1, John), person(z_1), \\ &father(z_2, z_1), person(z_2), \\ &father(z_3, z_2), person(z_3), \ldots\} \end{aligned}$$

(each z_i is a labeled null value)

Even with infinite chase, things are not always hopeless ...



- In first case, QA is obviously decidable

  If the chase can be built in PTIME (in data), QA too

- In second case, QA may be (and sometimes is) undecidable

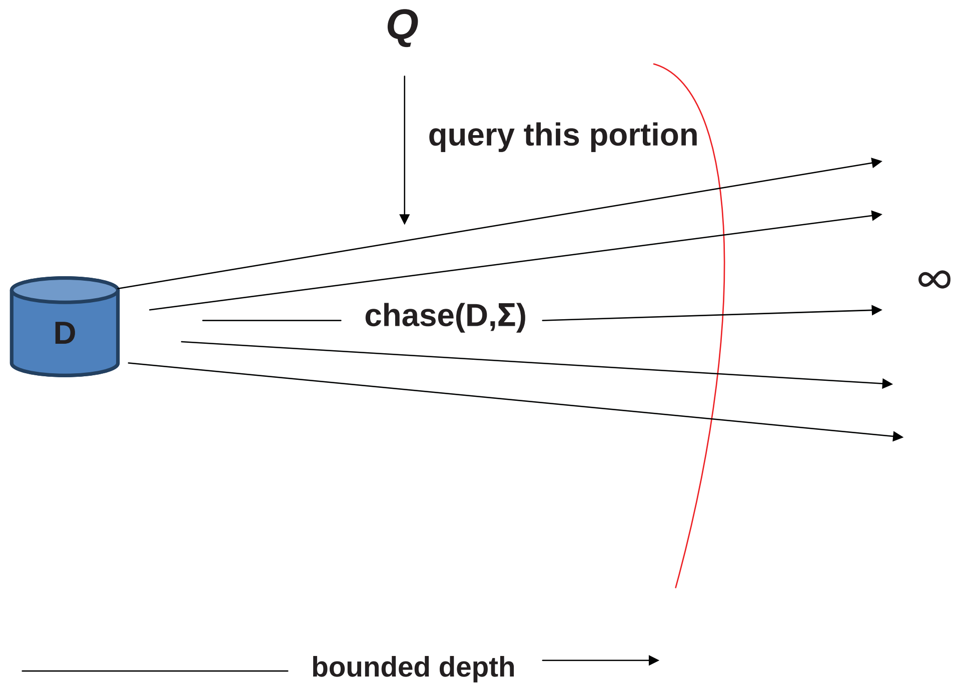
  But also possibly decidable depending on the program (and the class of queries, but we assume them conjunctive)

  Good cases of programs that ensure decidability of QA? And efficient QA?

Good alternatives considered for the second (infinite) case

Decidability of QA guaranteed by different syntactic conditions on the set of rules

The idea is that, depending on the programs, QA can be correctly done by querying only a bounded, initial portion of the chase



Hopefully a "short portion"

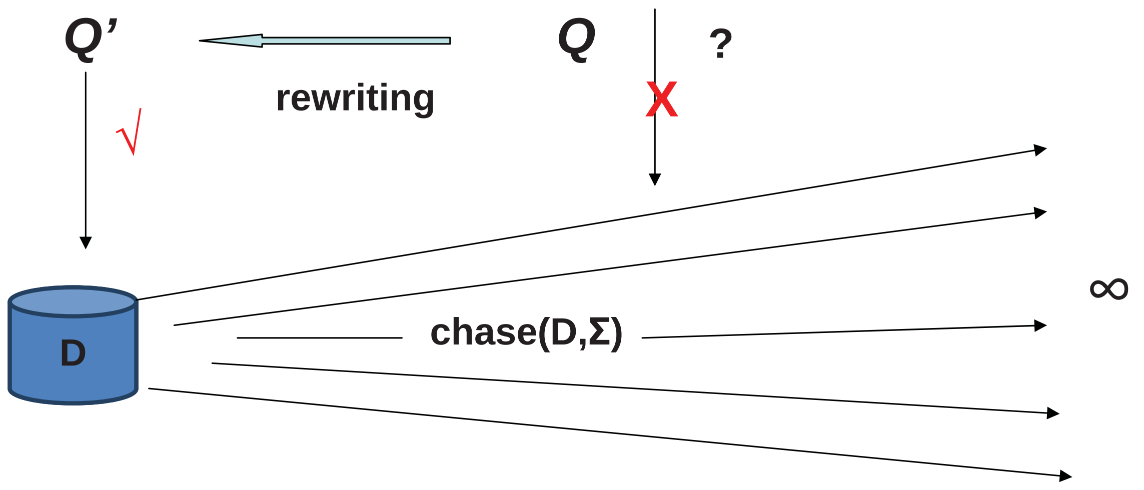
Two good cases:

(A) Bound independent from D (but dependent on $\mathcal{Q}, \Sigma$):
BDDP(roperty) (bounded derivation depth)

In this case, FO query rewriting is possible (more on this below)

Instead of posing the query to the (infinite) chase, rewrite the query $\mathcal{Q}$ into a new FO query $\mathcal{Q}'$ (independently from D)

Query D with $\mathcal{Q}'$ as usual



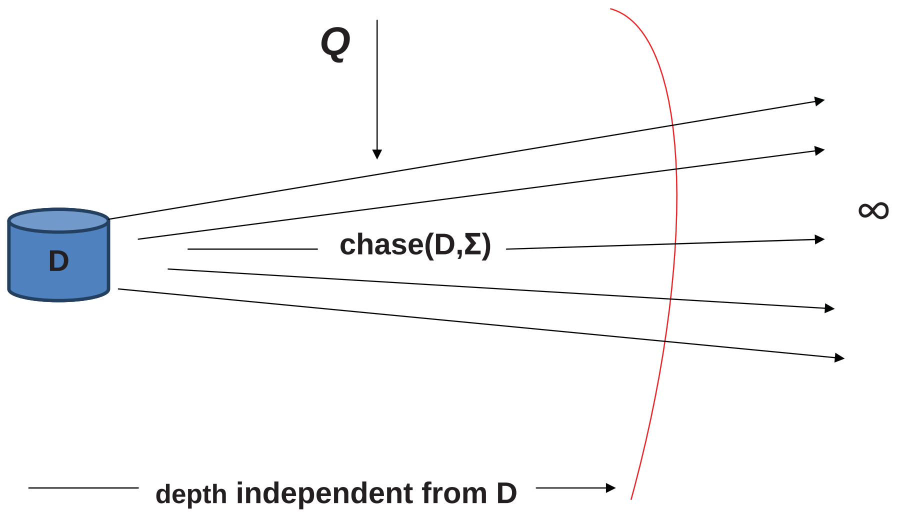
Definitely in PTIME in data

(B) Bound depends polynomially on (size of) D

- (A) is a particular case of (B)
- To achieve (B) (or (A)), different syntactic restrictions on Datalog[$\exists$] programs
- Identified various classes of Datalog$\pm$ programs: linear, guarded, sticky, weakly-sticky, ...
- For some of them, even (A) is possible

BDDP and FO-Rewritability

When a Datalog± program has BDDP, queries can be answered on a portion of the chase with depth independent from D



For programs with BDP, conjunctive queries can be rewritten and answered directly on the EDB

Rewriting via rules in the program

We'll see classes of programs that enjoy this property ...

Linear and Sticky Datalog±

- Linear Datalog±: Only one atom in rule bodies

Example: $D = \{person(John)\}$, and set of rules Σ:

$$\begin{aligned} \exists x\, father(x,y) &\leftarrow person(y) \\ person(x) &\leftarrow father(x,y) \end{aligned}$$

Σ is a linear Datalog± program

Linear Datalog± enjoys BDDP

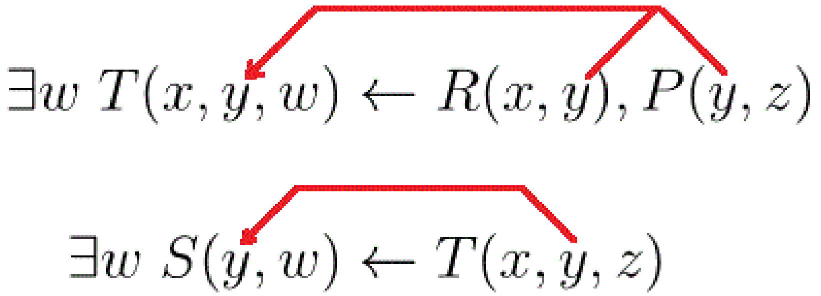
And then also FO-rewritability

- Sticky Datalog±:

Sticky programs enjoy BDDP (and then also FO-rewriting)

The chase has stickiness property (essentially, backward resolution terminates), which syntactically depends of whole program

Example: A non-linear program with the stickiness property

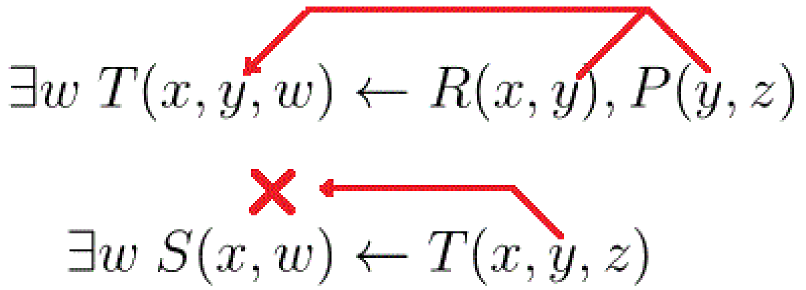


$$\exists w\ T(x, y, w) \leftarrow R(x, y), P(y, z)$$

$$\exists w\ S(y, w) \leftarrow T(x, y, z)$$

(repeated variables in the body stick in the chase)

A non- sticky program:



$$\exists w\ T(x, y, w) \leftarrow R(x, y), P(y, z)$$

$$\exists w\ S(x, w) \leftarrow T(x, y, z)$$

Stickiness can be checked syntactically

A two-step marking procedure on a set of TGDs Σ

Example:

$$\begin{array}{rcl} \exists x,y,z\ emp(w,x,y,z) & \leftarrow & dept(v,w) \\ \exists z\ dept(w,z), runs(w,y), in\_area(y,x) & \leftarrow & emp(v,w,x,y) \\ \exists z\ external(z,y,x) & \leftarrow & runs(w,x), \\ & & in\_area(x,y) \end{array}$$

1. Preliminary step : For each $\sigma \in \Sigma$ and variable $x \in body(\sigma)$, if there is an atom $a \in head(\sigma)$ such that x does not appear in a, mark each occurrence of x in $body(\sigma)$

$$\begin{array}{rcl} \exists x,y,z\ emp(w,x,y,z) & \leftarrow & dept(v,w) \\ \exists z\ dept(w,z), runs(w,y), in\_area(y,x) & \leftarrow & emp(v,w,x,y) \\ \exists z\ external(z,y,x) & \leftarrow & runs(w,x), \\ & & in\_area(x,y) \end{array}$$

$$\begin{array}{rcl}
\sigma_1\text{:}\ \ \exists x,y,z\ emp(w,x,y,z) & \leftarrow & dept(v,w) \\
\sigma_2\text{:}\ \ \exists z\ dept(w,z), runs(w,y), in\_area(y,x) & \leftarrow & emp(v,w,x,y) \\
\sigma_3\text{:}\ \ \exists z\ external(z,y,x) & \leftarrow & runs(w,x), \\
 & & in\_area(x,y)
\end{array}$$

2. Propagation step (until fixed point reached):
   for each $\sigma \in \Sigma$, if a marked variable in $body(\sigma)$ appears at position p, then for every $\sigma' \in \Sigma$ (including σ), mark each occurrence of the variables in $body(\sigma')$ that appear in $head(\sigma')$ in same position p

$$\begin{array}{lllll}
\sigma_2: & & \leftarrow & emp[1] & : v \\
\sigma_1: & emp[1] & \leftarrow & & : w \\
\sigma_1: & & \leftarrow & dept[2] & : w
\end{array}$$

$$\begin{array}{rl}
\exists x,y,z\ emp(w,x,y,z) \leftarrow & dept(v,\underline{w}) \\
\exists z\ dept(w,z), runs(w,y), in\_area(y,x) \leftarrow & emp(v,w,x,y) \\
\exists z\ external(z,y,x) \leftarrow & runs(w,x), in\_area(x,y)
\end{array}$$

Σ is sticky if no marked variable appears more than once in a $body(\sigma)$

This one is!

FO Rewriting-Based QA in Datalog±

Given: EDB D, a set of TGDs Σ (with BDDP), and conjunctive query $\mathcal{Q}$

(NCs and EGDs not considered here; see below)

- Construct FO rewriting $\mathcal{Q}_R$ of $\mathcal{Q}$ via Σ

  It holds: $\mathcal{Q}_R(D) = ans(\mathcal{Q}, D, \Sigma)$ (the certain answers)

- Evaluate $\mathcal{Q}_R$ over D

All this as long as:

- NCs hold, which can be checked separately by running associated conjunctive queries
- EGDs do not interact with TGDs (during the chase), a separability property, which can be syntactically checked

A rewriting algorithm is proposed for Datalog$\pm$ programs based on the iteration of two steps: (Gottlob, Orsi, Pieris; ICDE'11)

- Basic rewriting using the rules (resolution)
- Minimization of the query obtained from rewriting step

$\mathcal{Q}_R$ is the union of resulting conjunctive queries from iterations of the above

Example: (ex. in page 20 cont.) CQ $\mathcal{Q}$

$$q(p) \leftarrow in\_area(p,a), external(e,a,p)$$

Applying the TGD:

$$\exists z\ external(z,y,x) \leftarrow runs(w,x), in\_area(x,y)$$

basic rewriting step returns new CQ $\mathcal{Q}_1$: ($*$: don't care symbol)

$$q(p) \leftarrow in\_area(p,a), runs(*,p), in\_area(p,a)$$

$$q(p) \leftarrow \underline{in\_area(p,a)}, runs(*,p), \underline{in\_area(p,a)}$$

Minimization leads to new CQ $\mathcal{Q}_2$:

$$q(p) \leftarrow runs(*,p), in\_area(p,a)$$

The final result of the rewriting procedure is $\mathcal{Q}_R = \mathcal{Q} \vee \mathcal{Q}_2$, i.e.

$$\begin{aligned} q(p) &\leftarrow in\_area(p,a), external(e,a,p) \\ q(p) &\leftarrow runs(*,p), in\_area(p,a) \end{aligned}$$

Notice that we need to keep the original query since $external$ may have initial partial data

The resolution looks for potential additional data for it

If a predicate in $\mathcal{Q}_R$ is not EDB, its extension in D is empty

Why Stickiness?

Stickiness for a set of TGDs guarantees that backward resolution-based query rewriting terminates

Applying resolution with a TGD without repeated marked variable, no new variable is introduced (only new don't care symbols might appear)

Example: Sticky set of TGDs:

$$\exists x, y, z\ emp(w, x, y, z) \leftarrow dept(v, w)$$
$$\exists z\ dept(w, z), runs(w, y), in\_area(y, x) \leftarrow emp(v, w, x, y)$$
$$\exists z\ external(z, y, x) \leftarrow runs(w, x), in\_area(x, y)$$

Query: $q(p) \leftarrow in\_area(p, a), external(e, a, p)$

Applying last TGD: $q(p) \leftarrow in\_area(p, a), runs(*, p), in\_area(p, a)$

Variables are only inherited from the first query

Example: Modification of previous one, for non-sticky case

$$\begin{aligned}\exists x, y, z\ emp(w,x,y,z) &\leftarrow dept(v,w)\\ \exists z\ dept(w,z), runs(w,y), in\_area(y,x) &\leftarrow emp(v,w,x,y)\\ \exists z, t\ external(z,y,t) &\leftarrow runs(w,x), in\_area(x,y)\end{aligned}$$

(x marked and occurs twice in body of last TGD)

Same query: $q(p) \leftarrow in\_area(p,a), external(e,a,p)$

Applying last TGD: $q(p) \leftarrow in\_area(p,a), runs(*,r), in\_area(r,a)$

Generates new, "relevant" variable r (in a join), not from the original query

("Do not care variables" can get values in isolation)

Weakly-Sticky Datalog$\pm$

Weakly-sticky (WS) Datalog$\pm$ generalizes

- Sticky Datalog$\pm$ (SD)
- Weakly-Acyclic (WA) sets of of TGDs as introduced in data exchange

Accordingly, definition based on the marking procedure for SD, and dependency graphs used to define WA

Intuitively, a set of TGDs Σ is WS if every marked variable appearing more than once in a rule body also appears in a position that can take a finite number of values during the chase

The latter positions can be syntactically determined through the

Dependency graph: Directed dependency graph $G(\Sigma)$:

- Nodes are positions of predicates in Σ (e.g. $dept[2]$)

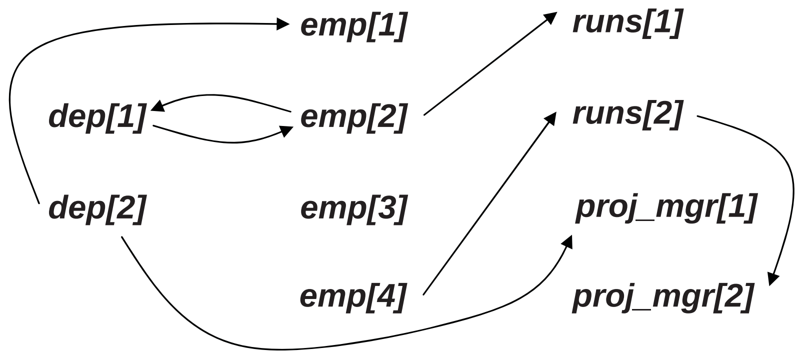
Example: $\exists x, y\ emp(w, v, x, y) \leftarrow dept(v, w)$

$$\exists z\ dept(w, z), runs(w, y) \leftarrow emp(v, w, x, y)$$

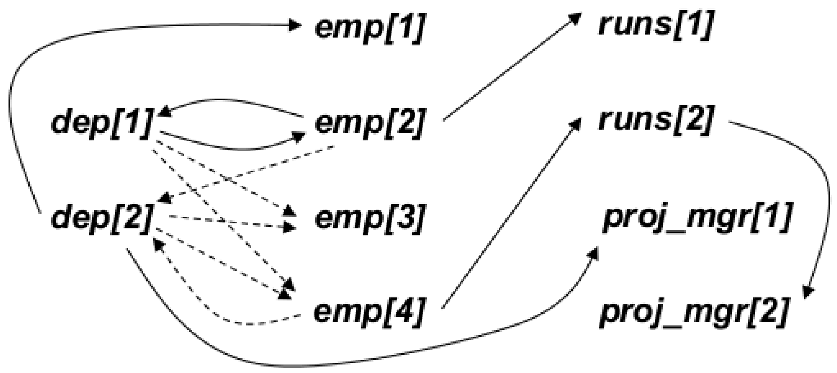
$$project\_mgr(y, x) \leftarrow runs(w, x), dept(w, y)$$

- For every $\sigma \in \Sigma$ and (universal) variable x in $head(\sigma)$ and in position p in $body(\sigma)$:

1. For each occurrence of x in position p' in $head(\sigma)$, create edge from p to p'



2. For each existential variable z in position p'' in $head(\sigma)$, create a special edge from p to p''



For a set of TGDs Σ:

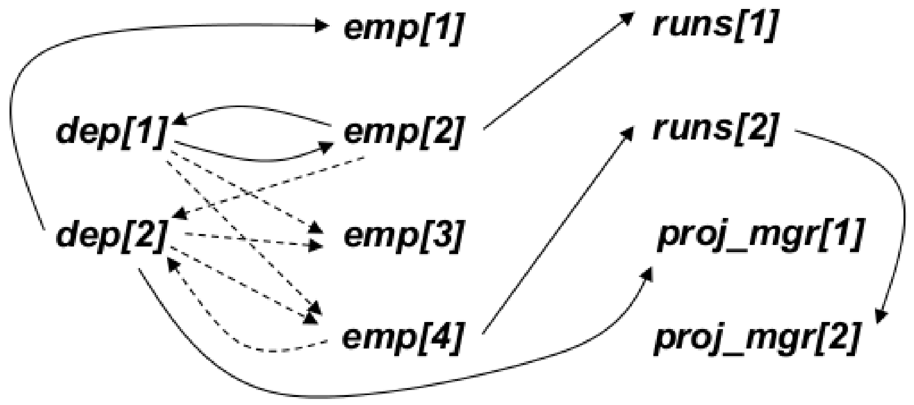
- The rank of a position is the maximum number of special edges over all (finite or infinite) paths ending at that position
- $\Pi_F(\Sigma)$: set of positions with finite rank
  $\Pi_\infty(\Sigma)$: set of positions with infinite rank

$\Pi_F(\Sigma)$ captures positions where finitely many values may appear during the chase

For those in $\Pi_\infty(\Sigma)$, infinitely many fresh null values may occur during the chase

$G(\Sigma)$ can be introduced for any Datalog[$\exists$] program, in a Datalog$\pm$ family or not

Example: (above)



$\Pi_F(\Sigma) = \{dept[1], emp[2], runs[1], \ldots\}$ $\Pi_\infty(\Sigma) = \emptyset$

A set of TGDs Σ is weakly-acyclic iff $\Pi_\infty(\Sigma) = \emptyset$

Weakly-Acyclic Datalog generalizes plain Datalog with limited use of existentially quantified variables

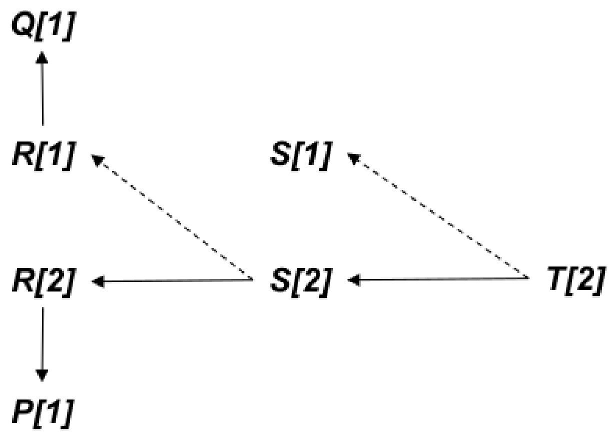
For WA Datalog the chase terminates (unlike some Datalog$\pm$ programs)

Definition: A set of TGDs Σ is weakly-sticky (WS) if every marked variable appearing more than once in a rule body, appears in that body at least once in a position in $\Pi_F(\Sigma)$

(Cali, Gottlob, Pieris; AIJ'12)

Example: Σ_1

$$\begin{array}{rcll} P(y) & \leftarrow & Q(\hat{x}), R(\hat{x}, y) & (1) \\ Q(x) & \leftarrow & R(\hat{x}, \hat{y}) & (2) \\ \exists z\, R(z, y) & \leftarrow & S(\hat{x}, y) & (3) \\ & \leftarrow & T(\hat{x}, y) & (4) \end{array}$$



$\Pi_\infty(\Sigma_1) = \emptyset$ so Σ_1 is weakly-acyclic, and then also weakly-sticky

Not sticky, due to (1)

Marking procedure as introduced for SD (hatted variables)

All repeated occurrences in (1) appear in positions in Π_F

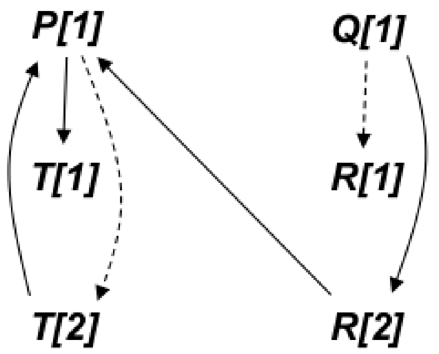
(It could be, e.g. $Q[1] \in \Pi_F$, but $R[1] \notin \Pi_F$, and still WS)

Due to the marking process, a repeated body variable is never both marked and non-marked

All repetitions are marked, and one of them in a "finite" position is good enough ....

Example: Modified set of TGDs Σ_2

$$\begin{array}{rcll} P(y) & \leftarrow & Q(\hat{x}), R(\hat{x}, y) & (5) \\ \exists x\ T(y, x) & \leftarrow & P(\hat{y}) & (6) \\ P(x) & \leftarrow & T(\hat{y}, x) & (7) \\ \exists x\ R(x, y) & \leftarrow & Q(y) & (8) \end{array}$$



$\Pi_\infty(\Sigma_2) = \{P[1], T[1], T[2]\}$ $\Pi_F(\Sigma_2) = \{Q[1], R[1], R[2]\}$

Σ_2 is not sticky: In (5) marked x occurs twice in the body

Σ_2 is not weakly-acyclic: $\Pi_\infty(\Sigma_2) \neq \emptyset$

Σ_2 is WS: Variable x appears in positions in $\Pi_F(\Sigma_2)$

Properties of Weakly-Sticky Datalog±

• For a WS program Σ it may be $\Pi_\infty(\Sigma) \neq \emptyset$

• A WA set of TGDs is WS by definition

• A SD set of TGDs is WS (there is no repeated marked variables in any rule)

• QA in PTIME (in data) complexity

• QA can be done on an initial portion of the chase whose depth polynomially depends on the EDB size

That is, a fixed polynomial that depends on the program, but is the same for every query

• That portion may not be of constant depth

- WS Datalog$\pm$ does not the have the BDDP

- QA is PTIME-complete (in data)

- WS Datalog$\pm$ does not have FO-rewritability property

- Query rewriting algorithm above may not terminate

- There is a non-deterministic Boolean CQ answering algorithm in PTIME in size of EDB

Some Technical Issues about WS Datalog$\pm$

- They allow to obtain the results mentioned before

- They are also important for other purposes (coming)

- We want to have a more clear pictures of the values that are created and considered during the chase

We saw that the syntactic restrictions on WS programs impose restrictions on the those values

- Given a set Σ of TGDs, its functional form, Σ^f:

In each $\sigma \in \Sigma$ replace an existential variable at a position in Π_F by a Skolem term $f_\sigma(x_1, ..., x_n)$ ($x_1, ..., x_n$ are variables in $body(\sigma)$)

Example: Σ_2 as above

$$\begin{aligned} P(y) &\leftarrow Q(\hat{x}), R(\hat{x}, y) \\ \exists x\, T(y, x) &\leftarrow P(\hat{y}) \\ P(x) &\leftarrow T(\hat{y}, x) \\ \exists x\, R(x, y) &\leftarrow Q(y) \end{aligned}$$

$\Pi_\infty(\Sigma_2) = \{P[1], T[1], T[2]\}$ $\Pi_F(\Sigma_2) = \{Q[1], R[1], R[2]\}$

Only existential in last rule is replaced

Σ_2^f:

$$\begin{aligned} P(y) &\leftarrow Q(\hat{x}), R(\hat{x}, y) \\ \exists x\, T(y, x) &\leftarrow P(\hat{y}) \\ P(x) &\leftarrow T(\hat{y}, x) \\ R(f(y), y) &\leftarrow Q(y) \end{aligned}$$

- Given: Set of TGDs Σ, EDB D

Π^f-terms of D and Σ:

1. Constants of $adom(D)$ are Π^f-terms of D
2. If g is an n-ary Skolem function symbol in Σ^f, and $t_1, ..., t_n$ are Π^f-terms, then $g(t_1, ..., t_n)$ is a Π^f-term

Size of a term t in Π^f-terms of D and Σ:

1. If $t \in adom(D)$, $size(t) = 0$
2. If $t = g(t_1, ..., t_n)$, then $size(t) = 1 + \Sigma_{i\in[1..n]} size(t_i)$

Example: Previous example, $D = \{Q(a)\}$

$f(a)$, $f(f(a))$, $f(f(f(a)))$ are Π^f-terms, with sizes 1, 2, and 3, resp.

Lemma: For EDB D and set of TGDs Σ

For every Π^f-term t that occurs in $chase(D, \Sigma^f)$:

$$size(t) \leq \begin{cases} 1 + w_{\Sigma} \times (k_{\Sigma} - 1) & \text{when } b_{\Sigma} = 1 \\ 1 + w_{\Sigma} \times b_{\Sigma} \times (\frac{(b_{\Sigma})^{k_{\Sigma}} - b_{\Sigma}}{b_{\Sigma} - 1}) & \text{when } b_{\Sigma} > 1 \end{cases}$$

with

- $b_{\Sigma} = max\ \{|body(\sigma)| \mid \sigma \in \Sigma\}$
- $k_{\Sigma} = rank(G_{\Sigma}) :=$ maximum rank of the positions in Π_F
- w_{Σ} is the maximum arity of predicate symbols in Σ

Notice that linear programs fall in the first case for $size(t)$

$chase(D, \Sigma^f)$ could generate null values, because not all existentials are replaced by Skolem functions

Those labeled nulls never appear in the positions of Π_F, so avoiding undecidability of QA

Example: WS Σ_2 and EDB $D = \{Q(a)\}$

$$\begin{array}{rrcl}
r_1: & P(y) & \leftarrow & Q(\hat{x}), R(\hat{x}, y) \\
r_2: & \exists x\, T(y, x) & \leftarrow & P(\hat{y}) \\
r_3: & P(x) & \leftarrow & T(\hat{y}, x) \\
r_4: & R(f(y), y) & \leftarrow & Q(y)
\end{array}$$

$$\begin{tikzcd}
Q(a) \arrow[d, "r4"'] \arrow[dr, "r1"] & & & & & & \\
R(f(a),a) \arrow[r, "r1"'] & P(a) \arrow[r, "r2"'] & T(a,Z1) \arrow[r, "r3"'] & P(Z1) \arrow[r, "r2"'] & T(Z1,Z2) \arrow[r, "r3"'] & P(Z2) \arrow[r, dashed] & {}
\end{tikzcd}$$

Only two Π^f-terms (a and $f(a)$) appear in the chase, and infinitely many labeled nulls

Corollary: For Σ WS set of TGDs, there is a function f from EDBs D to $\mathbb{N}$ that is PTIME computable in $|adom(D)|$ such that at most $f(D)$ terms may appear at positions in Π_F in $chase(D, \Sigma)$

The result stated in page 41 suggests that, given D and Σ, there is a systematic way to generate from D all (or a superset of) Π_F-terms of (D, Σ) that may appear in the chase

Particularly important for WS programs: only Π^f-terms can replace repeated marked variables during the chase

Idea we are investigating:

1. Start with a WS program
2. Replace repeated marked variables violating stickiness with their possible Π_F-terms; this is a grounding process
3. Obtain a sticky program
4. Apply techniques for sticky programs, in particular, for QA

Motivated by our ongoing research on applications of Datalog$\pm$ to the specification of multidimensional contexts for data quality assessment