From Logical Query Rewriting to Plan Selection for Ontology-Based Query Answering

Michaël Thomazo

Inria, France
michael.thomazo@inria.fr,
http://pages.saclay.inria.fr/michael.thomazo

Is Semantics Needed?

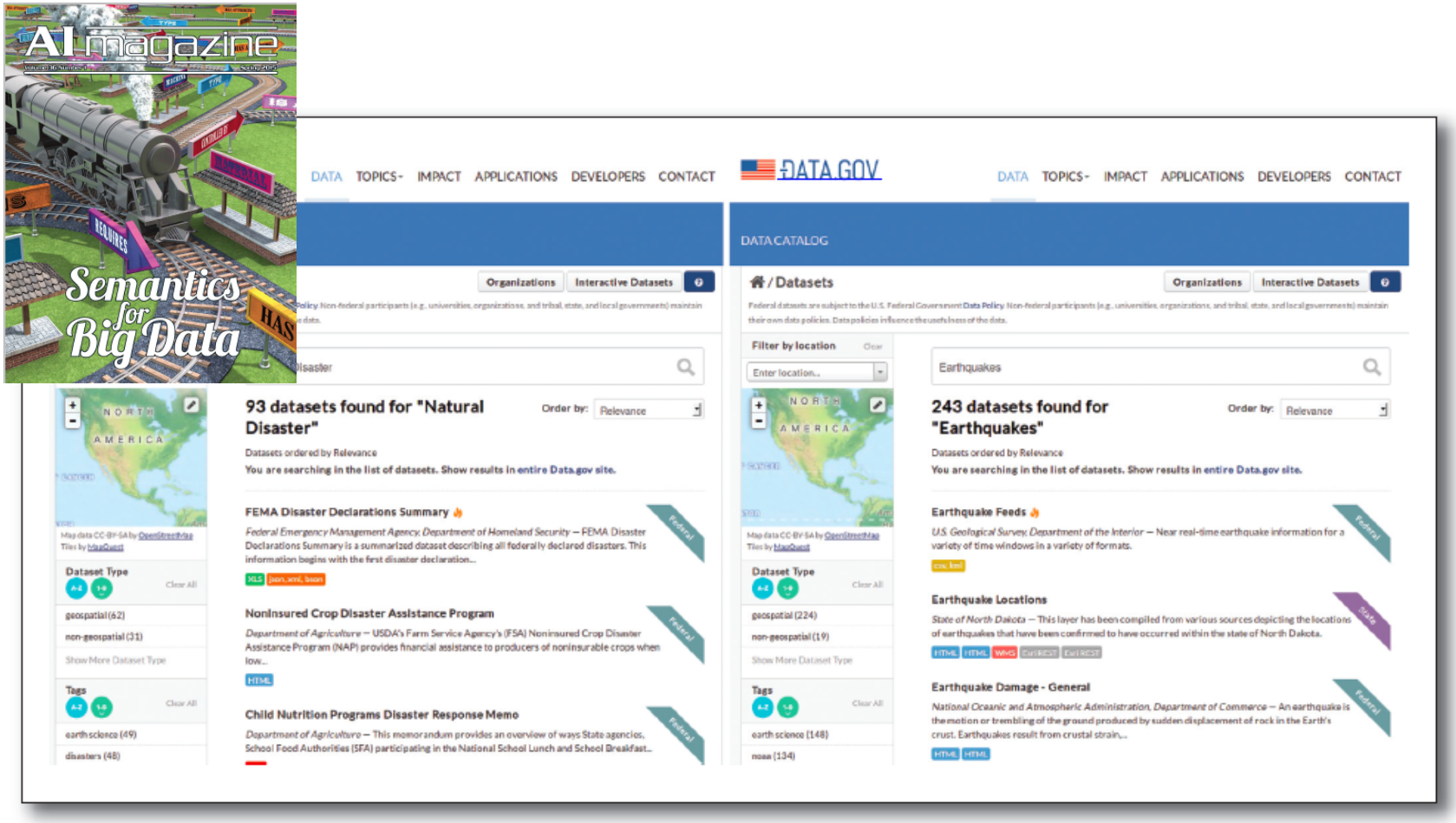


Figure 2: Searching Data.gov for Natural Disaster Data Sets.

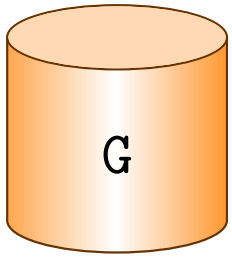
Part I

Background and outline

Ontology-Based Query Answering

- ontology – existential rules in this talk
- data – relational database in this talk
- query – conjunctive queries in this talk

ontology



query q

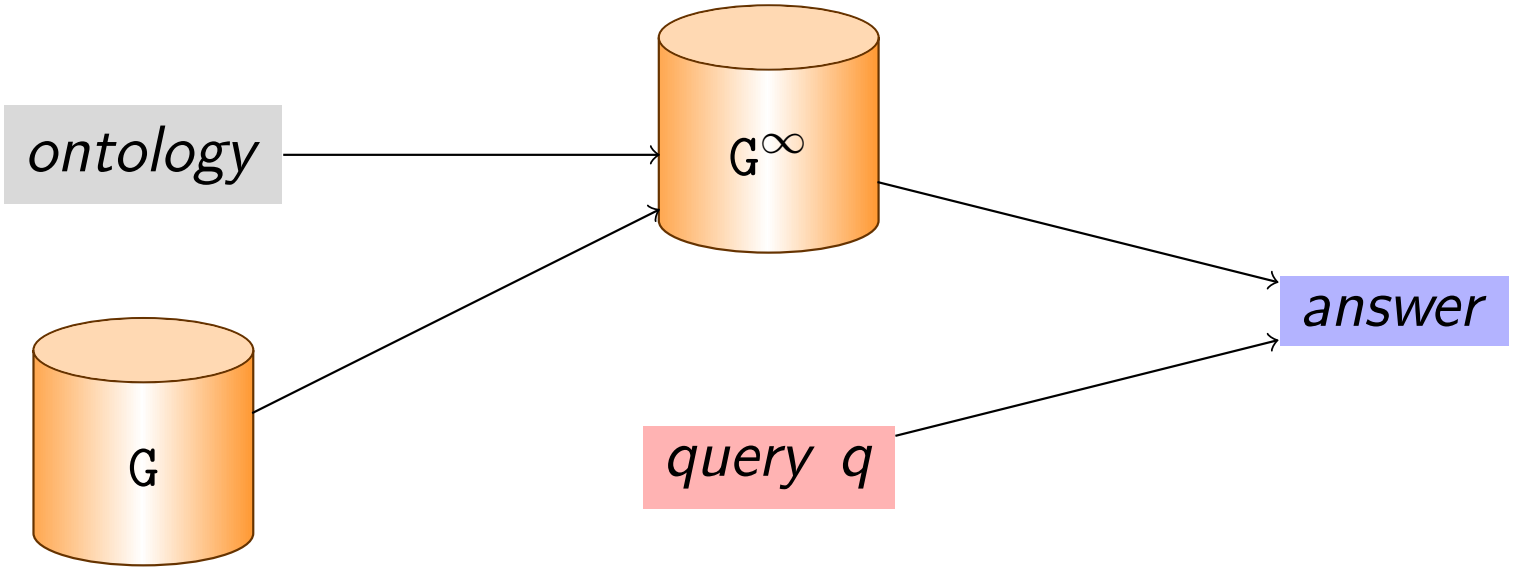
The need for reasoning

Query answering needs explicit and implicit data!

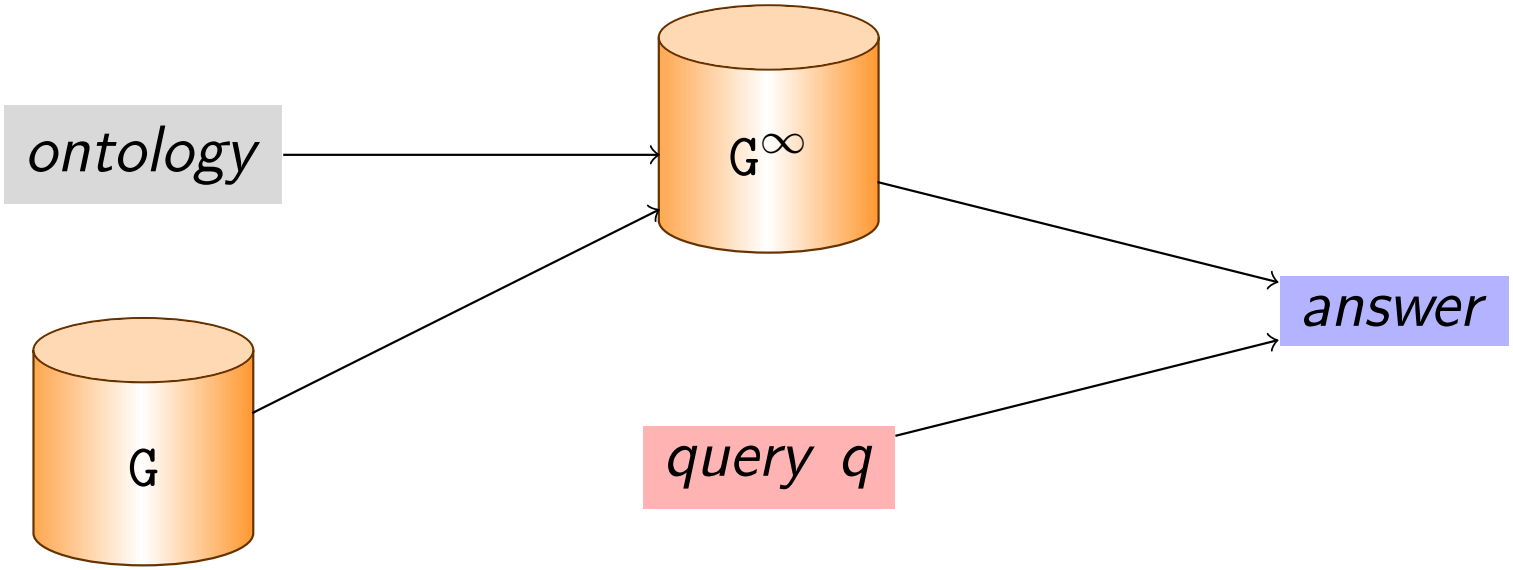
- Materialization-based query answering
- Reformulation-based query answering
- Hybrids of the above: combined approaches

Reformulation is the focus of this talk.

Materialization-based query answering

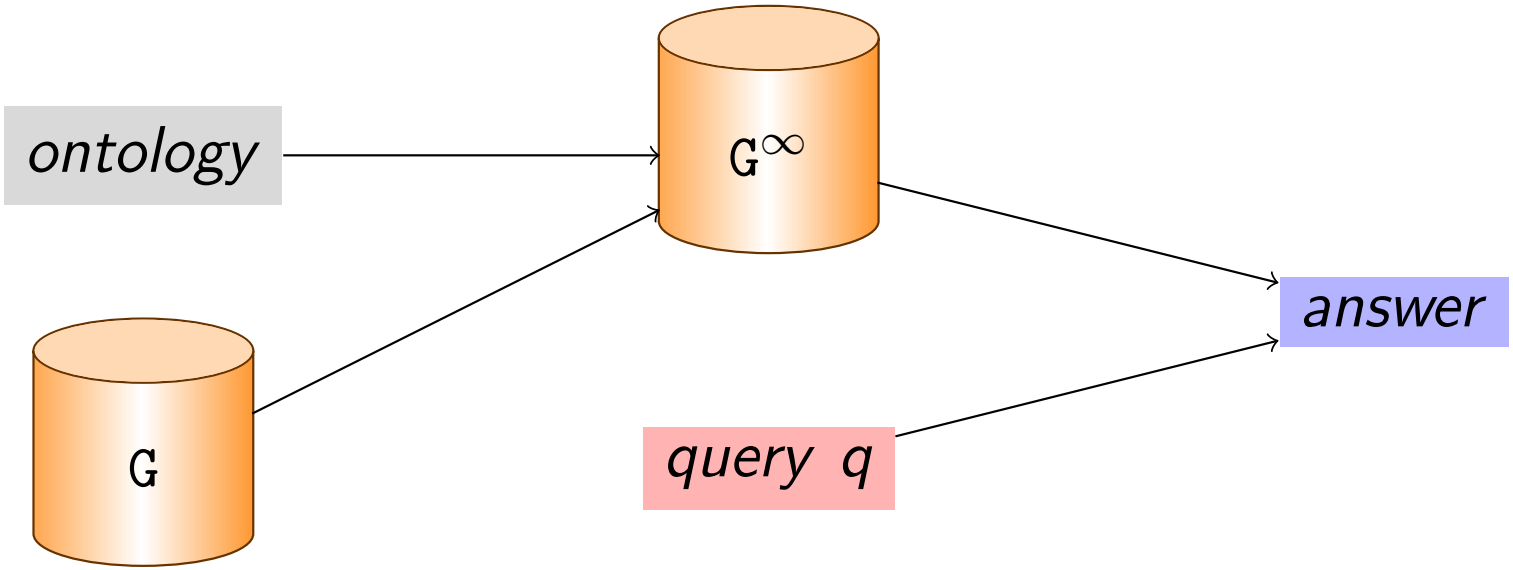


Materialization-based query answering



- $q(\mathbb{G}^\infty)$ can be computed using an RDBMS
- $\mathbb{G}^\infty$ needs time to be computed and space to be stored
- Not suitable for high update rate (data and/or schema triples)

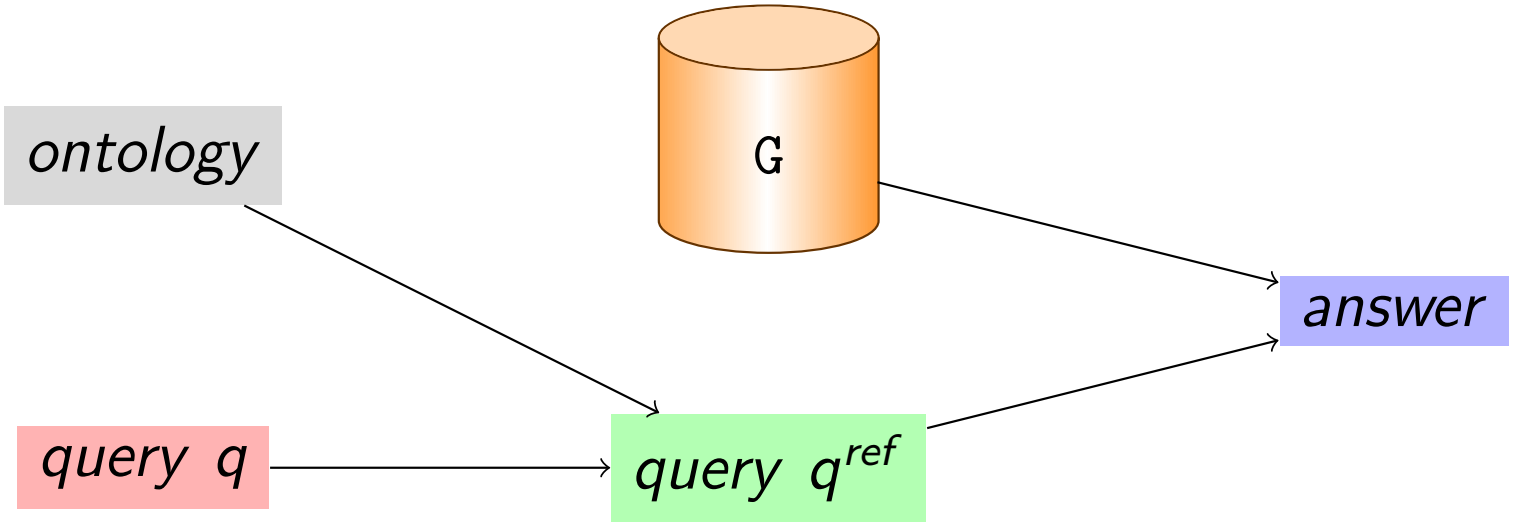
Materialization maintenance



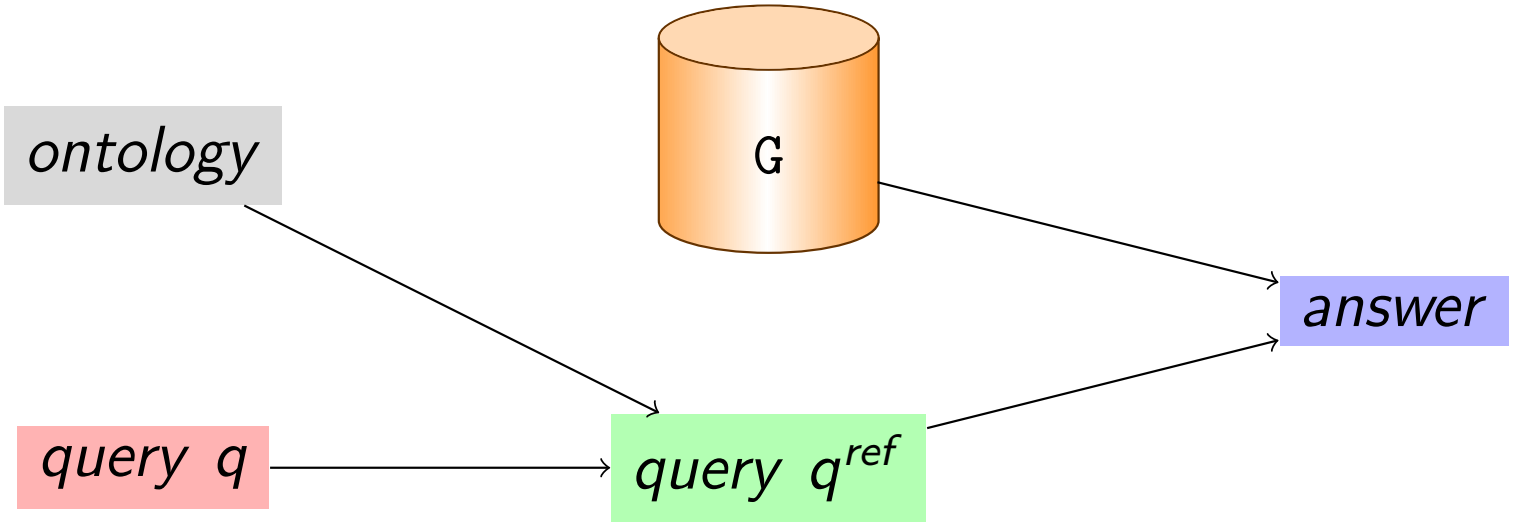
Compute Δ for an update of an RDF graph $\mathtt{G}$ s.t.

- $(\mu(\mathtt{G}))^\infty = \mathtt{G}^\infty \cup \Delta$ when μ an insertion
- $(\mu(\mathtt{G}))^\infty = \mathtt{G}^\infty \setminus \Delta$ when μ a deletion

Reformulation-based query answering



Reformulation-based query answering



- $q^{ref}(\mathtt{G})$ can be evaluated using an RDBMS
- Robust to updates
- Reformulated queries are complex, thus costly to evaluate

Take-home message

Re-using database technology is a great idea

...

But out-of-the-box solutions do not exists yet

Outline

1. ontology languages;
2. logical reformulation: theoretical limits and practical approaches;
3. the need for novel techniques in query optimization;
4. first results and challenges in query optimization.

Part II

Logical Reformulation

Existential Rules

Definition and Example

An existential rule (TGD, Datalog+/- rule) is a first-order formula of the shape:

$$\forall \mathbf{x}\ \forall \mathbf{y}\ B[\mathbf{x}, \mathbf{y}] \rightarrow \exists \mathbf{z} H[\mathbf{y}, \mathbf{z}],$$

where $\mathbf{x}, \mathbf{y}$ and $\mathbf{z}$ are tuples of variables, B and H are conjunctions of atoms.

- $\forall x\ human(x) \rightarrow \exists y\ parent(x, y) \wedge human(y)$
- $\forall x, y\ teaches(x, y) \wedge MasterCourse(y) \rightarrow PhD(x)$

Existential Rules

Definition and Example

An existential rule (TGD, Datalog+/- rule) is a first-order formula of the shape:

$$\forall \mathbf{x}\ \forall \mathbf{y}\ B[\mathbf{x}, \mathbf{y}] \rightarrow \exists \mathbf{z} H[\mathbf{y}, \mathbf{z}],$$

where $\mathbf{x}, \mathbf{y}$ and $\mathbf{z}$ are tuples of variables, B and H are conjunctions of atoms.

- $\forall x\ \mathit{human}(x) \rightarrow \exists y\ \mathit{parent}(x, y) \wedge \mathit{human}(y)$
- $\forall x, y\ \mathit{teaches}(x, y) \wedge \mathit{MasterCourse}(y) \rightarrow \mathit{PhD}(x)$

Generalize RDFS axioms, OWL 2 profiles, Horn-Description Logics.

Existential Rules

Definition and Example

An existential rule (TGD, Datalog+/- rule) is a first-order formula of the shape:

$$\forall \mathbf{x}\ \forall \mathbf{y}\ B[\mathbf{x}, \mathbf{y}] \rightarrow \exists \mathbf{z} H[\mathbf{y}, \mathbf{z}],$$

where $\mathbf{x}, \mathbf{y}$ and $\mathbf{z}$ are tuples of variables, B and H are conjunctions of atoms.

- $\forall x\ human(x) \rightarrow \exists y\ parent(x, y) \wedge human(y)$
- $\forall x, y\ teaches(x, y) \wedge MasterCourse(y) \rightarrow PhD(x)$

Generalize RDFS axioms, OWL 2 profiles, Horn-Description Logics.

Out of the scope of this talk

Possibility to add equality generating dependencies and negative constraints.

Materialization

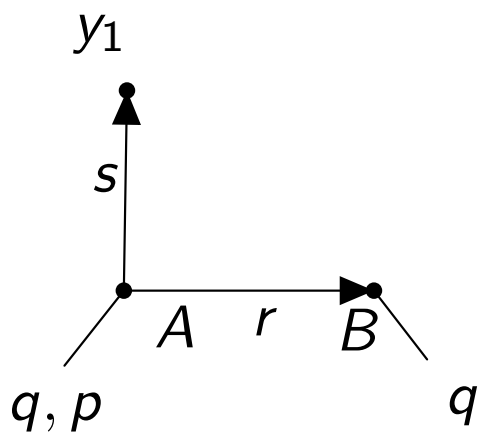
- $\forall x\ \forall y\ r(x, y) \rightarrow p(x)$
- $\forall x\ q(x) \rightarrow \exists y\ s(x, y)$

Materialization

- $\forall x\ \forall y\ r(x, y) \rightarrow p(x)$
- $\forall x\ q(x) \rightarrow \exists y\ s(x, y)$

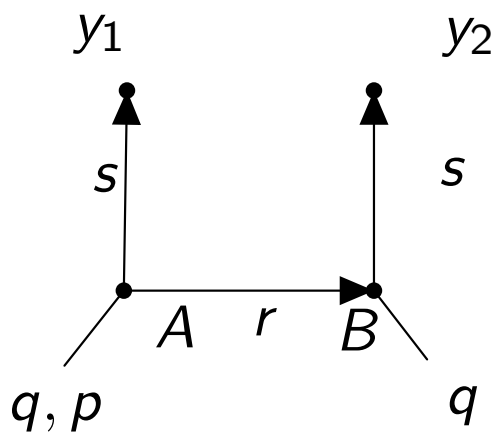
Materialization

- $\forall x\ \forall y\ r(x, y) \rightarrow p(x)$
- $\forall x\ q(x) \rightarrow \exists y\ s(x, y)$



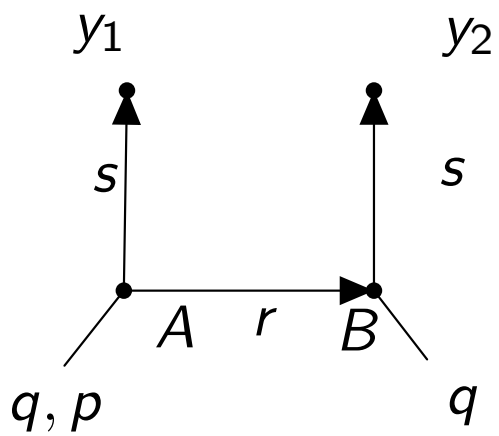
Materialization

- $\forall x\ \forall y\ r(x, y) \rightarrow p(x)$
- $\forall x\ q(x) \rightarrow \exists y\ s(x, y)$



Materialization

- $\forall x\ \forall y\ r(x, y) \rightarrow p(x)$
- $\forall x\ q(x) \rightarrow \exists y\ s(x, y)$



Computation of the so-called **canonical model**.

Query Rewriting

- $\forall x\ \forall y\ r(x, y) \rightarrow p(x)$
- $\forall x\ q(x) \rightarrow \exists y\ s(x, y)$

Rewriting of $\exists x, y\ p(x) \wedge s(x, y)$

$$(\exists x, y\ p(x) \wedge s(x, y)) \vee (\exists x\ p(x) \wedge q(x)) \vee$$
$$(\exists x, y, z\ r(x, z) \wedge s(x, y)) \vee (\exists x, z\ q(x) \wedge r(x, z))$$

Decidability issues

The canonical model may not be finite. A finite first-order rewriting may not exist. Moreover, it is undecidable to know if one exists.

First-order rewritability

- INPUT: a ruleset $\mathcal{R}$ and a query q
- OUTPUT: yes if and only if there exists a first order query q' such that for all database D, $D \models q'$ if and only if $D, \mathcal{R} \models q$.

Syntactic restrictions may ensure good properties

Three main known criteria allowing decidability:

- acyclicity, first notion weak acyclicity [Fagin et al., 05] (implies finiteness of the canonical model)
- guardedness, inspired from [Andréka et al., 98] (implies tree-likeness of the canonical model)
- "backward-shyness" (implies existence of a first order rewriting)

Syntactic restrictions may ensure good properties

Three main known criteria allowing decidability:

- acyclicity, first notion weak acyclicity [Fagin et al., 05] (implies finiteness of the canonical model)
- guardedness, inspired from [Andréka et al., 98] (implies tree-likeness of the canonical model)
- "backward-shyness" (implies existence of a first order rewriting)

Linear Rules

A rule is linear if its body contains a single atom.

Linear rules are guarded and backward shy.

Three kind of approaches for first-order rewriting

- reformulation algorithms working only for a given known good case (for instance, DL-Lite)
- reformulation algorithms providing a reformulation whenever it exists, but that do not stop when it does not (considers existential rules)
- reformulation algorithms that provide a reformulation or says it does not exists, but only for some classes (for example for $\mathcal{EL}$).

Three kind of approaches for first-order rewriting

- reformulation algorithms working only for a given known good case (for instance, DL-Lite)
- reformulation algorithms providing a reformulation whenever it exists, but that do not stop when it does not (considers existential rules)
- reformulation algorithms that provide a reformulation or says it does not exists, but only for some classes (for example for $\mathcal{EL}$).

Generic Algorithm

Perform a breadth-first exploration of the rewriting space:

- pick a query q to explore
- for each rule ρ:
  - for each possible way q' of rewriting q w.r.t. ρ, add q' to the set of queries to explore
- loop

If any of the generated query is entailed by the database, then the original query is entailed by the knowledge base.
Termination criterion: no more "new queries are added". Beware of some technicalities [König et al, 2012].

Without Existentials

$\forall x\ \forall y\ r(x, y) \rightarrow p(x)$

A possible rewriting $\exists x, y\ p(x) \wedge s(x, y)$ w.r.t to the rule

$$\exists x, y, z\ r(x, z) \wedge s(x, y)$$

- unification of the head of the rule with a subpart of the query
- possible specializations
- removal of the unified part
- addition of the (possibly specialized) head of the query

With Existentials

One rule: $\forall x\ (p(x) \rightarrow \exists y\ r(x, y))$

Let us consider:

$$p(x) \wedge r(x, z) \wedge r(y, z) \wedge q(y).$$

By the same process, we get as a possible rewriting:

$$p(x)(\wedge p(x)) \wedge r(y, z) \wedge q(y).$$

This is not a correct rewriting!

Keeping both soundness and completeness

One rule: $\forall x\ (p(x) \to \exists y\ r(x, y))$

Problem before:

- shared variable is unified with an existentially quantified variable of the rule
- forbid this $\to$ soundness is recovered... but completeness is lost
- allow to "merge atoms"

$$p(x) \wedge r(x, z) \wedge r(y, z) \wedge q(y).$$

rewritten in

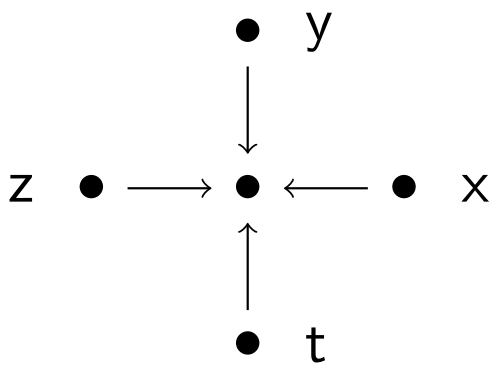
$$p(x) \wedge r(x, z) \wedge q(x),$$

then rewritten in

$$p(x) \wedge q(x).$$

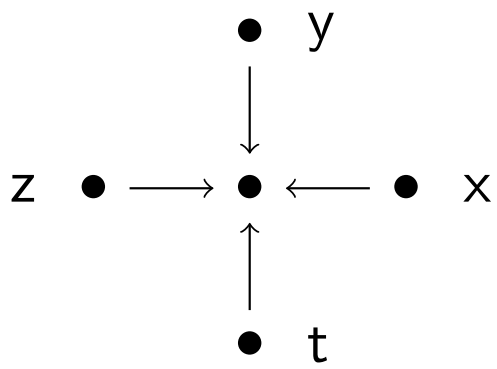
Piece-based rewriting

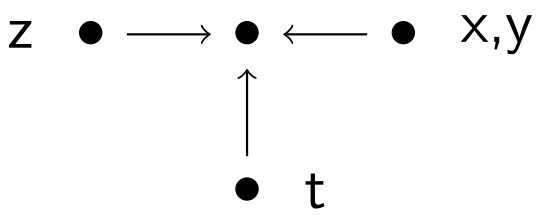
One rule: $p(x) \rightarrow r(x, y)$



Piece-based rewriting

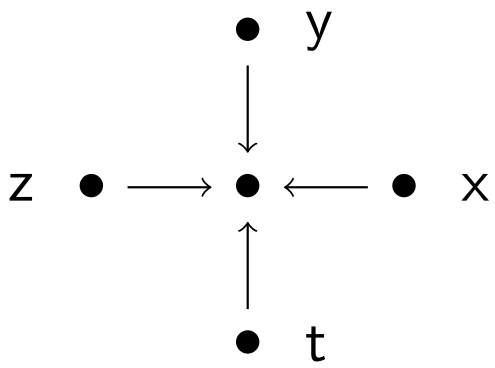
One rule: $p(x) \rightarrow r(x, y)$

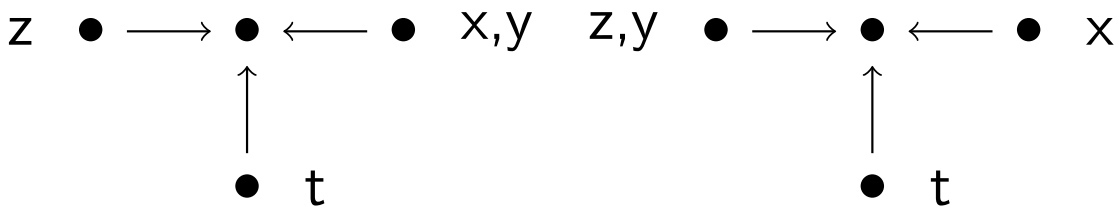




Piece-based rewriting

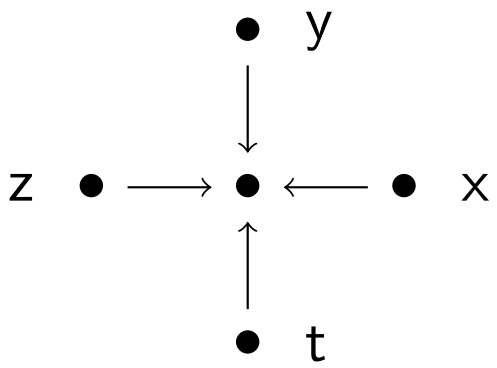
One rule: $p(x) \rightarrow r(x, y)$

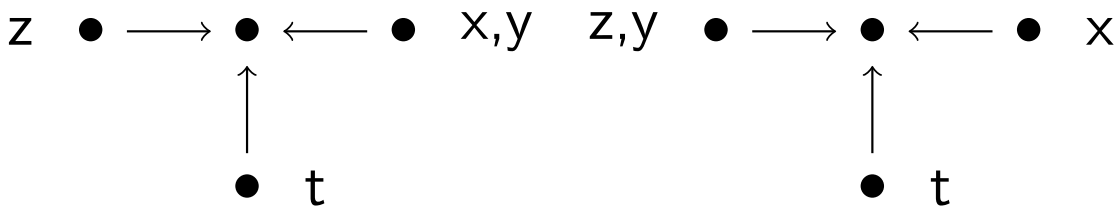




Piece-based rewriting

One rule: $p(x) \rightarrow r(x, y)$





Piece-based rewriting: directly generating $p(x)$ as the unique rewriting.

Size issues

Union of Conjunctive Queries are not Compact

Let consider that a_i (resp. b_i, c_i) are subclasses of a (resp b, c), for i from 1 to n.
The query:

$$\exists x \; a(x) \wedge b(x) \wedge c(x)$$

is reformulated in a query of size n^3.

- easy to reach reformulation of a million CQs
- these are **not even accepted** by RDBMs, let alone efficiently evaluated.

Going to higher expressivity languages

Changing the expressivity of the target languages may allow to reduce the size of the rewriting:

A polynomial result (Kikot et al. 11)

Conjunctive queries under unary inclusion dependencies admit polynomial first order rewriting.

Unary inclusion dependencies: strict subset of liner rules.
But things remain bad:

Polynomial first-order/non-recursive Datalog is not enough (Kikot et al. 12)

For "pure" first-order and non-recursive datalog rewritings, an expontential blow-up is unavoidable for query rewriting (under some complexity theoretic assumption)

In practice?

Some attempts to explain/exploit better behavior in practice:

- "tree witnesses": providing guarantuees on the size of the rewriting depending on characteristics of the query and/or of the ontology (Kikot et al 12);
- dealing with hierarchies: they represent an overwhelming share of the real-world axioms.

Compact

T. 13

- based on piece-based rewriting
- applied on more general queries than CQs: the so-called SCQs (semi-conjunctive queries)
- SCQs are ‘‘completed’’ by default – a new one is generated only if necessary

In the previous case, the rewriting would be:

$$\exists x\ (a_1(x) \vee \ldots \vee a_n(x) \vee a(x)) \wedge (b_1(x) \vee \ldots \vee b_n(x) \vee b(x)) \wedge (c_1(x) \vee \ldots \vee c_n(x) \vee c(x))$$

Summarizing

1. we have "compiled" the ontology into the query
2. we have a first-order query to evaluate
3. great! it can be expressed in SQL
4. RDBMS are mature systems... let us just use them!

Part III

Evaluating Reformulations

RDBMs

- RDBMs have a lot of great features;
- they are mature and commercially used;
- a lot of query optimization has been done...
- but only for some kind of queries!
- in particular...

Unions are typically not optimized!

Picking the right SQL query matters

Given the query

$$
\begin{aligned}
q_1(x, y) \text{ :- } & x \text{ rdf:type } y, && (\mathbf{t_1})\\
& x \textit{ ub:degreeFrom } \text{“}\textit{http://www.University532.edu}\text{”}, && (\mathbf{t_2})\\
& x \textit{ ub:memberOf } \text{“}\textit{http://www.Department1.University7.edu}\text{”} && (\mathbf{t_3})
\end{aligned}
$$

and the LUBM 100M benchmark:

SQL-query	exec. time (ms)
$(t_1, t_2, t_3)^{ref}$	**6,387**
$(t_1)^{ref} \bowtie (t_2)^{ref} \bowtie (t_3)^{ref}$	**1,074,026**
$(t_1, t_2)^{ref} \bowtie (t_3)^{ref}$	1,968
$(t_1)^{ref} \bowtie (t_2, t_3)^{ref}$	846,710
$(t_1, t_3)^{ref} \bowtie (t_2)^{ref}$	**554**
$(t_1, t_2)^{ref} \bowtie (t_1, t_3)^{ref}$	2,734
$(t_1, t_2)^{ref} \bowtie (t_2, t_3)^{ref}$	2,289
$(t_1, t_3)^{ref} \bowtie (t_2, t_3)^{ref}$	588

The UCQ rewriting contains 2256 conjunctive queries.

Need for Query Optimization

- all the above queries are semantically equivalent
- only the syntax differ...
- which has an influence on which execution plan is used

Logical Plans

Let us assume that we want to evaluate the following query:

$$(a_1(x, y) \vee a_2(x, y)) \wedge (b_1(y, z) \vee b_2(y, z)) \wedge c(z)$$

Cost Model

Among all the previous options (and others), which solution is the best **depends on the data and of the system.** Cost models are used, that typically depends on:

- system-dependent parameters
- knowledge (or guesses) about how the system is working
- cardinality estimation of tables/subexpressions of the query

Important Feature 1 – Join Ordering

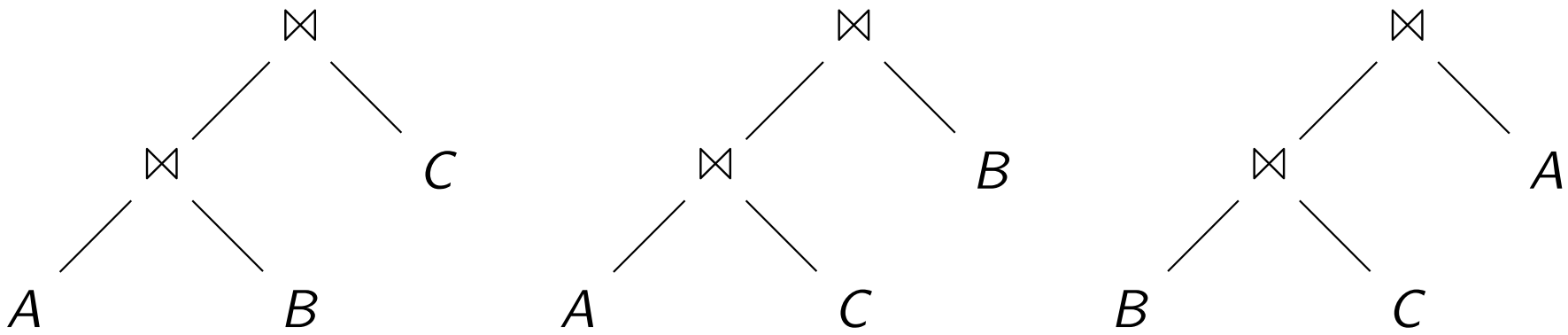
We want to evaluate the following query:

$$a(x, y) \wedge b(y, z) \wedge c(z, t)$$

Important Feature 1 – Join Ordering

We want to evaluate the following query:

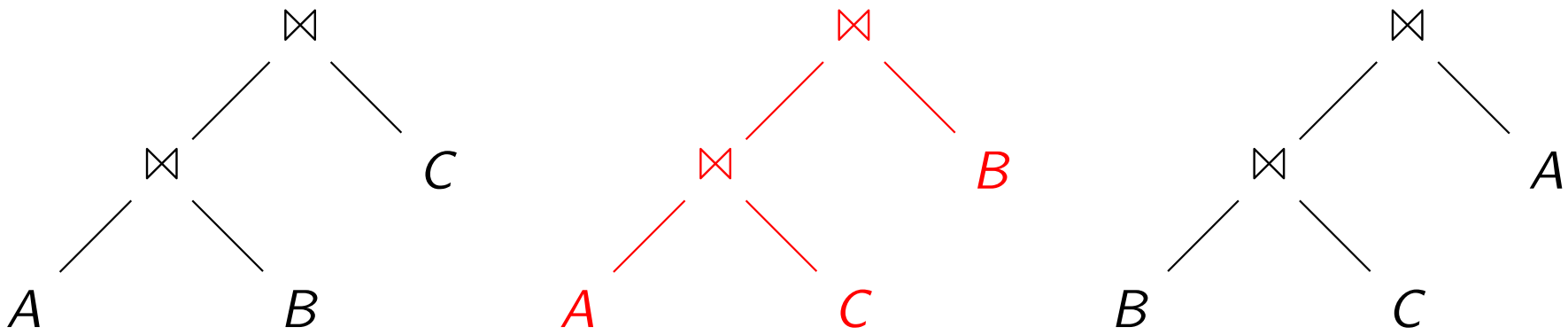
$$a(x, y) \wedge b(y, z) \wedge c(z, t)$$



Important Feature 1 – Join Ordering

We want to evaluate the following query:

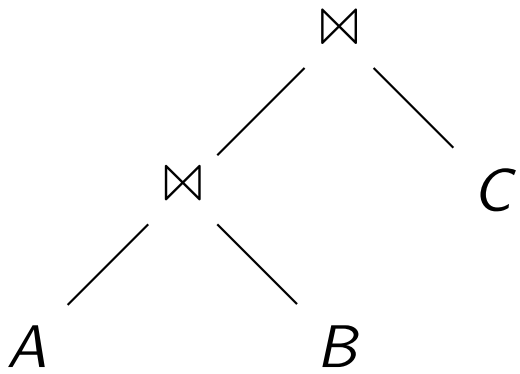
$$a(x, y) \wedge b(y, z) \wedge c(z, t)$$

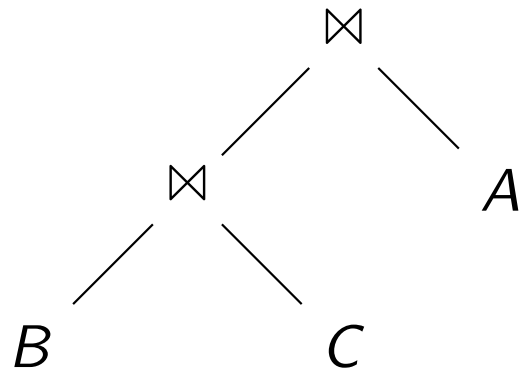


Important Feature 1 – Join Ordering

We want to evaluate the following query:

$$a(x, y) \wedge b(y, z) \wedge c(z, t)$$

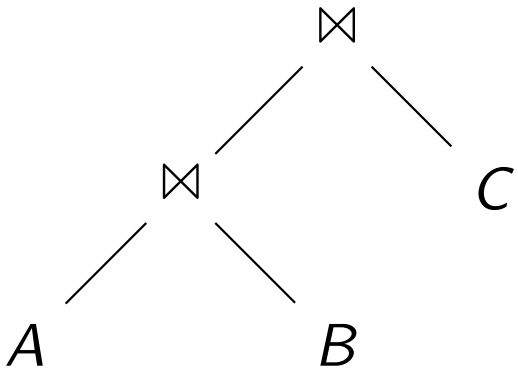


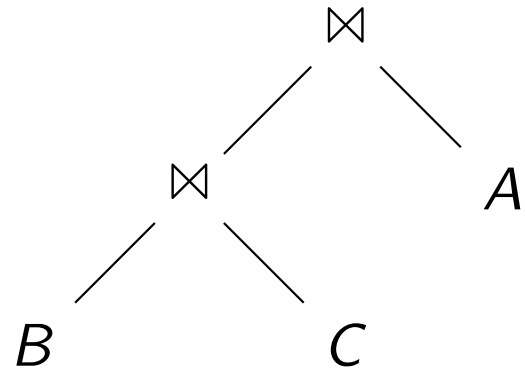


Important Feature 2 – Use of Indexes

$$a(x, y) \wedge b(y, z) \wedge c(z, t)$$

- Indexes allow fast retrieval of values;
- Let us assume that we have on index on the first attribute of b;
- ... Which plan allows to exploit that index?





And with Unions?

- search space increases even more
- in the conjunctive case, one can (heuristically) consider only left-deep plans $\rightarrow$ not true anymore
- HUGE search space
- unions are by definition not selective

JUCQ approach [Bursztyn et al., 2013]

- splitting the original query into subparts
- each subpart is rewritten and evaluated separately
- the results are then joined

$$
\begin{aligned}
q_1(x, y) \text{ :- } & x \text{ rdf:type } y, && (\mathbf{t_1})\\
& x \textit{ ub:degreeFrom } \text{“}\textit{http://www.University532.edu}\text{”}, && (\mathbf{t_2})\\
& x \textit{ ub:memberOf } \text{“}\textit{http://www.Department1.University7.edu}\text{”} && (\mathbf{t_3})
\end{aligned}
$$

$$(t_1, t_3)^{ref} \bowtie (t_2)^{ref}$$

$$(t_1, t_2)^{ref} \bowtie (t_1, t_3)^{ref}$$

JUCQ approach [Bursztyn et al., 2013]

- splitting the original query into subparts
- each subpart is rewritten and evaluated separately
- the results are then joined

$$
\begin{aligned}
q_1(x, y) \text{ :- } & x \text{ rdf:type } y, && (\mathbf{t_1})\\
& x \textit{ ub:degreeFrom } \text{“}\textit{http://www.University532.edu}\text{”}, && (\mathbf{t_2})\\
& x \textit{ ub:memberOf } \text{“}\textit{http://www.Department1.University7.edu}\text{”} && (\mathbf{t_3})
\end{aligned}
$$

$$(t_1, t_3)^{ref} \bowtie (t_2)^{ref}$$

$$(t_1, t_2)^{ref} \bowtie (t_1, t_3)^{ref}$$

Repeated Atoms?

Useful to take advantage of the selectivity of certain atoms; i.e., when there are few t_2 (resp. t_3) that have corresponding t_1.

Extending the search space

Current work:

- more complex search space (not only JUCQs, but arbitrary alternations between joins and unions)
- exploring the materialization of partial results to avoid repeating work (ie, plans that are not tree-shaped anymore)

Part IV

Conclusion and perspectives

Where we stand

Semantics is crucial for applications, so the push is continuous for chosing "the right ontology language"

Where we stand

Semantics is crucial for applications, so the push is continuous for chosing "the right ontology language"
We considered semantic query answering through **FOL reduction (i.e., SQL)**.

Where we stand

Semantics is crucial for applications, so the push is continuous for chosing "the right ontology language"
We considered semantic query answering through **FOL reduction (i.e., SQL)**.
Efficiently obtaining a first-order rewriting for real-world ontologies is manageable when applicable

Where we stand

Semantics is crucial for applications, so the push is continuous for chosing "the right ontology language"
We considered semantic query answering through **FOL reduction (i.e., SQL)**.
Efficiently obtaining a first-order rewriting for real-world ontologies is manageable when applicable
But not any SQL query resulting from reformulation is handled well by current RDBMSs!

Where we stand

Semantics is crucial for applications, so the push is continuous for chosing "the right ontology language"

We considered semantic query answering through **FOL reduction (i.e., SQL)**.

Efficiently obtaining a first-order rewriting for real-world ontologies is manageable when applicable

But not any SQL query resulting from reformulation is handled well by current RDBMSs!

Vast performance differences between different reformulations of the same query; cost-based approach

Where to go?

Theoretical bounds are important to understand which features may play well together

Where to go?

Theoretical bounds are important to understand which features may play well together

The field needs usable tools to have a strong impact beyong the community

Where to go?

Theoretical bounds are important to understand which features may play well together

The field needs usable tools to have a strong impact beyong the community

The database community is a natural partner here – and there is no nice solution yet!