Carleton
UNIVERSITY

From Consistent Query Answering to Query Rewriting:

A Detour around Answer Set Programs

Leopoldo Bertossi

Carleton University
School of Computer Science
Ottawa, Canada

Consistent Query Answering

Why Consistent Query Answering?

A (relational) database instance D may be inconsistent

It does not satisfy a given set of integrity constraints IC: $D \not\models IC$

D is a FO structure, that is identified with a finite set of ground atoms of the FO language associated to the relational schema

However, we do not throw D away

Most of the data in it is still consistent, i.e. intuitively and informally, it does not participate in the violation of IC

We can still obtain meaningful and correct answers from D

Initial motivation for the research in "Consistent Query Answering" (CQA) was to:

- Characterize in precise terms the data in D that is consistent with IC
- Develop mechanisms for computing/extracting the consistent information from D

  More specifically, obtain answers to queries from D that are consistent with IC

This research program was explicitly started in this form in (Arenas, Bertossi, Chomicki; Pods 1999)

Along the way and with collaborators and by other authors several problems were attacked and many research issues raised:

- Extensions of mechanisms proposed in (Arenas et al.; Pods 1999)
- Computational complexity analysis of CQA
- Alternative (but similar in spirit) characterizations of consistent answer
- Implementations efforts
- Conceptual/theoretical applications to data integration, peer data exchange, etc.
- Logical formalization of reasoning with consistent data wrt inconsistent databases

  Understanding the "logical laws" of consistent query answering in databases

Consistent Answers and Repairs

Example: Database instance D and FD: $Name \rightarrow Salary$

$Employee$	$Name$	$Salary$
	$Page$	5K
	$Page$	8K
	$Smith$	3K
	$Stowe$	7K

D violates FD through the tuples with $Page$ in $Name$

There are two possible ways to repair the database in a minimal way if only deletions/insertions of whole tuples are allowed

Repairs D_1, resp. D_2

Employee	*Name*	*Salary*
	Page	5K
	Smith	3K
	Stowe	7K

Employee	*Name*	*Salary*
	Page	8K
	Smith	3K
	Stowe	7K

$(Stowe, 7K)$ persists in all repairs, and it does not participate in the violation of FD; it is invariant under these minimal ways of restoring consistency

$(Page, 8K)$ does not persist in all repairs, and it does participate in the violation of FD

Fixed: DB schema with (infinite) domain; and a set of first order integrity constraints IC

Definition: (Arenas et al.; Pods 1999)

A repair of instance D is an instance D'

- over the same schema and domain
- satisfies IC: $D' \models IC$
- Makes $\Delta(D, D')$ minimal wrt set inclusion

Definition: (Arenas et al.; Pods 1999)

Tuple of constants $\bar{t}$ is a consistent answer to query $\mathcal{Q}(\bar{x})$ in D iff

$\bar{t}$ is an answer to query $\mathcal{Q}(\bar{x})$ in every repair D' of D:

$$D \models_{IC} \mathcal{Q}(\bar{t}) \quad :\Longleftrightarrow \quad D' \models \mathcal{Q}(\bar{t}) \quad \text{for every repair } D' \text{ of } D$$

A model-theoretic definition ...

Example: (continued)

$$D \models_{FD} Employee(Stowe, 7K)$$

$$D \models_{FD} (Employee(Page, 5K) \ \vee Employee(Page, 8K))$$

$$D \models_{FD} \exists x Employee(Page, x)$$

Example: $D = \{P(a, b), Q(c, b)\}$, IC: $\forall x \forall y(P(x, y) \rightarrow Q(x, y))$

The repairs are:

- $D_1 = \{Q(c, b)\}$ with $\Delta(D, D_1) = \{P(a, b)\}$
- $D_2 = \{P(a, b), Q(a, b), Q(c, b)\}$ with $\Delta(D, D_2) = \{Q(a, b)\}$

But not $D_3 = \{P(a, b), Q(a, b)\}$, because
$\Delta(D, D_3) = \{Q(a, b), Q(c, b)\} \supsetneq \Delta(D, D_2)$

Computing Consistent Answers

We want to compute consistent answers, but not by computing all possible repairs and checking answers in common

Retrieving consistent answers via explicit and material computation of all database repairs may not be the right way to go

Example: An inconsistent instance wrt FD: $X \rightarrow Y$

D	X	Y
	1	0
	1	1
	2	0
	2	1
	.	.
	n	0
	n	1

It has 2^n possible repairs!

Try to avoid or minimize computation of repairs ...

FO Query Rewriting (sometimes)

First-Order queries and constraints

Approach: Transform the query and keep the database instance!

- Consistent answers to $\mathcal{Q}(\bar{x})$ in D?
- Rewrite query: $\mathcal{Q}(\bar{x}) \longmapsto \mathcal{Q}'(\bar{x})$
  $\mathcal{Q}'(\bar{x})$ is a new FO query
- Retrieve from D the (ordinary) answers to $\mathcal{Q}'(\bar{x})$

Example: $D = \{P(a), P(b), Q(b), Q(c)\}$

IC: $\forall x(P(x) \rightarrow Q(x))$

$\mathcal{Q}(x)$: $P(x)$? (consistent answer should be (b))

If $P(x)$ holds, then $Q(x)$ must hold

An answer t to $P(x)$ is consistent if t is also answer to $Q(x)$

Rewrite $\mathcal{Q}(x)$ into $\mathcal{Q}'(x)$: $P(x) \wedge Q(x)$ and pose it to D

$Q(x)$ is a residue of $P(x)$ wrt IC

Residue obtained by resolution between query literal and IC

Posing new query to D (as usual) we get only answer (b)

Example: (continued) Same FD:

$$\forall xyz\ (\neg Employee(x, y)\ \vee\ \neg Employee(x, z)\ \vee\ y = z)$$

$\mathcal{Q}(x, y)$: $Employee(x, y)$

Consistent answers: $(Smith,\ 3K),\ (Stowe,\ 7K)$

(but not $(Page,\ 5K),\ (Page,\ 8K)$)

Can be obtained via rewritten query:

$$T(\mathcal{Q}(x, y)) := Employee(x, y) \wedge$$

$$\forall z\ (\neg Employee(x, z)\ \vee\ y = z)$$

... those tuples (x, y) in the relation for which x does not have and associated z different from y ...

In general, T has to be applied iteratively

Example:

$IC: \ \{R(x) \vee \neg P(x) \vee \neg Q(x), \ P(x) \vee \neg Q(x)\}$

$\mathcal{Q}(x)$: $Q(x)$

$T^1(\mathcal{Q}(x)) := Q(x) \wedge (R(x) \vee \neg P(x)) \wedge P(x)$

Apply T again, now to the appended residues

$T^2(\mathcal{Q}(x)) := Q(x) \wedge (T(R(x)) \vee T(\neg P(x))) \wedge T(P(x))$

$$T^2(\mathcal{Q}(x)) = Q(x) \ \wedge \ (R(x) \vee (\neg P(x) \wedge \neg Q(x))) \ \wedge P(x) \wedge (R(x) \vee \neg Q(x))$$

And again:

$$T^3(\mathcal{Q}(x)) := Q(x) \ \wedge \ (R(x) \vee (\neg P(x) \wedge T(\neg Q(x)))) \wedge \\ P(x) \wedge (T(R(x)) \vee T(\neg Q(x)))$$

Since $T(\neg Q(x)) = \neg Q(x)$ and $T(R(x)) = R(x)$, we obtain

$T^3(\mathcal{Q}(x)) \ = \ T^2(\mathcal{Q}(x))$ A finite fixed point!

Does it always exist?

In general, an infinitary query:

$$T^\omega(\mathcal{Q}(x)) \ := \ \bigcup_{n<\omega} \{T^n(\mathcal{Q}(x))\}$$

Is T^ω sound, complete, finitely terminating?

Several sufficient and necessary syntactic conditions on ICs and queries have been identified for these properties to hold

(Arenas et al.; Pods 1999)

For the sake of this presentation, we can mention that iterative application of T is correct and semantically finitely terminating when (sufficient):

- Each element of IC is simultaneously
  - Universal: universal closure of disjunctions of literals
  - Binary: at most two database literals plus built-ins
  - Uniform: every variable in a literal appears in some other literal
- Queries are projection-free conjunctions of literals

Example:

$$IC = \{\forall xy(P(x,y) \rightarrow R(x,y)),\ \forall xy(R(x,y) \rightarrow P(x,y)),$$
$$\forall xyz(P(x,y) \wedge P(x,z) \rightarrow y = z)\}$$

$$\mathcal{Q}(x,y):\ \ R(x,y) \wedge \neg P(x,y)$$

Some Limitations

CQA based on first-order query rewriting has provable intrinsic limitations (cf. later)

In particular, T^ω does not work for full FO queries and ICs

- T is defined and works for some special classes of queries and integrity constraints
- ICs are universal, which excludes referential ICs
- T does not work for disjunctive or existential queries, e.g. $\exists y\ Employee(Page, y)$?

Other approaches to FO query rewriting are in principle possible (cf. later)

What Kind of Logic for CQA?

From the logical point of view:

- We have not logically specified the database repairs
- We have a model-theoretic definition plus an incomplete computational mechanism
- From such a specification $Spec$ we might:
  - Reason from $Spec$
  - Consistently answer queries: $Spec \stackrel{?}{\models} \mathcal{Q}(\bar{x})$
  - Derive algorithms for consistent query answering

Notice ...

Example: Database D and FD: $Name \rightarrow Salary$

$Employee$	$Name$	$Salary$
	$Page$	5K
	$Smith$	3K
	$Stowe$	7K

It holds: $D \models_{FD} Employee(Page,\ 5K)$

However

$D \cup \{Employee(Page,\ 8K)\} \not\models_{FD} Employee(Page,\ 5K)$

- Consistent query answering is non-monotonic
- A non-monotonic semantics for $Spec$ and its logic is expected
- What other logical properties of CQA reasoning/entailment?

Disjunctive Logic Programs for CQA

Specifying Database Repairs

The class of all database repairs can be represented in a compact form

This class can be specified using logic programs

Use disjunctive logic programs with stable model semantics (a.k.a. Answer Set Programs) (Gelfond, Lifschitz; NGC 1991)

Here we use the ASPs essentially introduced in (Barcelo, Bertossi; PADL 2003)

Repairs correspond to distinguished models of the program, namely to its stable models

The programs use annotation constants in an extra attribute in the database relations

- To keep track of the atomic repair actions, i.e. insertions or deletions of tuples ($\mathbf{t}, \mathbf{f}$)
- To give feedback to the program in case additional changes become necessary due to interacting ICs ($\mathbf{t}^\star$)
- To collect the tuples in the final, repaired instances ($\mathbf{t}^{\star\star}$)

Annotation	Atom	The tuple $P(\bar{a})$ is ...
$\mathbf{d}$	$P(\bar{a}, \mathbf{d})$	fact in original database
$\mathbf{t}$	$P(\bar{a}, \mathbf{t})$	made true (inserted)
$\mathbf{f}$	$P(\bar{a}, \mathbf{f})$	made false (deleted)
$\mathbf{t}^\star$	$P(\bar{a}, \mathbf{t}^\star)$	true or made true
$\mathbf{t}^{\star\star}$	$P(\bar{a}, \mathbf{t}^{\star\star})$	true in the repair

Example: $IC:\ \forall xy(P(x,y) \rightarrow Q(x,y))$

$$D = \{P(c,l), P(d,m), Q(d,m), Q(e,k)\}$$

Repair program $\Pi(D, IC)$:

1. Original data facts: $P(c,l,\mathbf{d})$, etc.

2. Whatever was true or becomes true, is annotated with $\mathbf{t}^\star$:

$$P(\bar{x}, \mathbf{t}^\star) \leftarrow P(\bar{x}, \mathbf{d})$$
$$P(\bar{x}, \mathbf{t}^\star) \leftarrow P(\bar{x}, \mathbf{t}) \qquad \text{(the same for } Q\text{)}$$

3. There may be interacting ICs (not here), and the repair process may take several steps, changes could trigger other changes

$$P(\bar{x}, \mathbf{f}) \ \vee \ Q(\bar{x}, \mathbf{t}) \ \leftarrow \ P(\bar{x}, \mathbf{t}^\star), Q(\bar{x}, \mathbf{f})$$
$$P(\bar{x}, \mathbf{f}) \ \vee \ Q(\bar{x}, \mathbf{t}) \ \leftarrow \ P(\bar{x}, \mathbf{t}^\star), \ not\ Q(\bar{x}, \mathbf{d})$$

$$P(\bar{x}, \mathbf{f}) \vee Q(\bar{x}, \mathbf{t}) \leftarrow P(\bar{x}, \mathbf{t}^\star), Q(\bar{x}, \mathbf{f})$$
$$P(\bar{x}, \mathbf{f}) \vee Q(\bar{x}, \mathbf{t}) \leftarrow P(\bar{x}, \mathbf{t}^\star), \ not\ Q(\bar{x}, \mathbf{d})$$

Two rules per IC; that says how to repair the IC (c.f. the head) in case of a violation (c.f. the body)

Passing to annotation $\mathbf{t}^\star$ allows to keep repairing the DB wrt to all the ICs until the process stabilizes

4. Repairs must be coherent: Program denial constraints prune undesirable models

$$\leftarrow P(\bar{x}, \mathbf{t}), P(\bar{x}, \mathbf{f})$$
$$\leftarrow Q(\bar{x}, \mathbf{t}), Q(\bar{x}, \mathbf{f})$$

5. Annotations constants $\mathbf{t}^{\star\star}$ are used to read off the atoms in a repair

$$P(\bar{x}, \mathbf{t}^{\star\star}) \leftarrow P(\bar{x}, \mathbf{t})$$
$$P(\bar{x}, \mathbf{t}^{\star\star}) \leftarrow P(\bar{x}, \mathbf{d}), \ not\ P(\bar{x}, \mathbf{f})$$

Etc.

The program has two stable models (and two repairs):

$$\begin{aligned}\mathcal{M}_1 &= \{P(c,l,\mathbf{d}),...,P(c,l,\mathbf{t}^\star),Q(c,l,\mathbf{t}),P(c,l,\mathbf{t}^{\star\star}),\\ &\quad Q(c,l,\mathbf{t}^\star),P(d,m,\mathbf{t}^{\star\star}),Q(d,m,\mathbf{t}^{\star\star}),\ldots,Q(c,l,\mathbf{t}^{\star\star})\}\\ &\equiv \{P(c,l),Q(c,l),P(d,m),Q(d,m),Q(e,k)\}\end{aligned}$$

... insert $Q(c,l)$!!

$$\begin{aligned}\mathcal{M}_2 &= \{P(c,l,\mathbf{d}),...,P(c,l,\mathbf{t}^\star),P(d,m,\mathbf{t}^{\star\star}),Q(d,m,\mathbf{t}^{\star\star}),\\ &\quad \ldots,P(c,l,\mathbf{f}),...\}\\ &\equiv \{P(d,m),Q(d,m),Q(e,k)\}\end{aligned}$$

... delete $P(c,l)$!!

One-to-one correspondence between repairs and stable models of the program

Consistent Query Answering

To obtain consistent answers to a (FO) query:

1. Transform or provide the query as a logic program (a standard process)

1. $\mathcal{Q}(\cdots P(\bar{u}) \cdots) \longmapsto \mathcal{Q}' := \mathcal{Q}(\cdots P(\bar{u}, \mathbf{t}^{\star\star}) \cdots)$
2. $\mathcal{Q}'(\bar{x}) \longmapsto (\Pi(Q'), Ans(\bar{X}))$

$\Pi(\mathcal{Q}')$ is a query program, a third layer on top of the DB and the repair program

$Ans(\bar{X})$ is a query atom defined in $\Pi(\mathcal{Q}')$

2. Run the query program together with the specification program under the skeptical or cautious stable model semantics

It sanctions as true of a program what is true of all its stable models

"Run" $\Pi := \Pi(\mathcal{Q}') \cup \Pi(D, IC)$

3. Collect ground atoms

$$Ans(\bar{t}) \in \bigcap\{S \mid S \text{ is a stable model of } \Pi\}$$

Example: (continued)

Consistent answers to $\mathcal{Q}(x, y) : \ P(x, y)$

Run repair program $\Pi(D, IC)$ together with query program

$$Ans(\bar{x}) \leftarrow P(\bar{x}, \mathbf{t}^{\star\star})$$

The two previous stable models become extended with ground Ans atoms

$$\begin{aligned} \mathcal{M}_1' &= \mathcal{M}_1 \cup \{Ans(c, l), Ans(d, m)\} \\ \mathcal{M}_2' &= \mathcal{M}_2 \cup \{Ans(d, m)\} \end{aligned}$$

Then the only answer is (d, m)

Discussion

- ASPs can be used to provide declarative and executable specifications of database repairs

- ASP based specification of repairs and CQAs as consequences from a program provide some sort of logic for CQA

A non-classical logic though ...

- ASPs extended with query programs provide a form of query rewriting: $D, \mathcal{Q} \;\; \mapsto \;\; \Pi(D, IC) \cup \Pi(\mathcal{Q})$

Leaving aside database (program facts), this is query rewriting

In a language that is more expressive than FOL ...

- The same repair program can be used with all queries, the same applies to the computed stable models

The query at hand adds a final layer on top

- For existential ICs, like referential ICs (RICs), there are alternatives
  - The setting in (Arenas et al.; Pods 1999) allows for introduction of arbitrary constants from the database domain

    Studied in (Cali, Lembo, Rosati; PODS 2005)
  - Only tuple deletions to satisfy RICs are allowed

    Studied in (Chomicki, Marcinkowski; Inf.&Comp. 2005)
  - Use a notion of repair that allows for the introduction of null values or cascaded deletions

    Nulls do not propagate, creating new inconsistencies

    Nulls as in a "logical reconstruction" of their use and IC satisfaction in presence of nulls in SQL

    There are corresponding repair programs in this case (acyclic RICs) (Bravo, Bertossi; IIDB 2006)

- Use of DLP is a general methodology that works for universal ICs and referential ICs, general FO queries (and beyond)

- Complete computation of all the stable models is undesirable

Better try generation of less and/or "partial" repairs

- Compute repairs wrt ICs and data that are relevant to the query

And efficient and compact (DB) encoding of the collection of stable models

Optimization of the access to DB and relevant portions of it
(Eiter, Fink, G.Greco, Lembo; ICLP 03), (Caniupan, Bertossi; SUM 2007)

• Query evaluation based on skeptical stable model semantics should be guided by the query and its relevant information in the database

Magic sets for evaluating ASPs can be used for CQA
(Caniupan, Bertossi; SUM 2007)

• In some situations, doing CQA via repair programs is most natural, at least conceptually

For example, in virtual data integration systems and P2P data exchange

Mappings and legal instances can be specified with LPs that can be coupled with repair programs

(Bravo, Bertossi; IJCAI 2003), (Bertossi, Bravo; LPAR 2007)

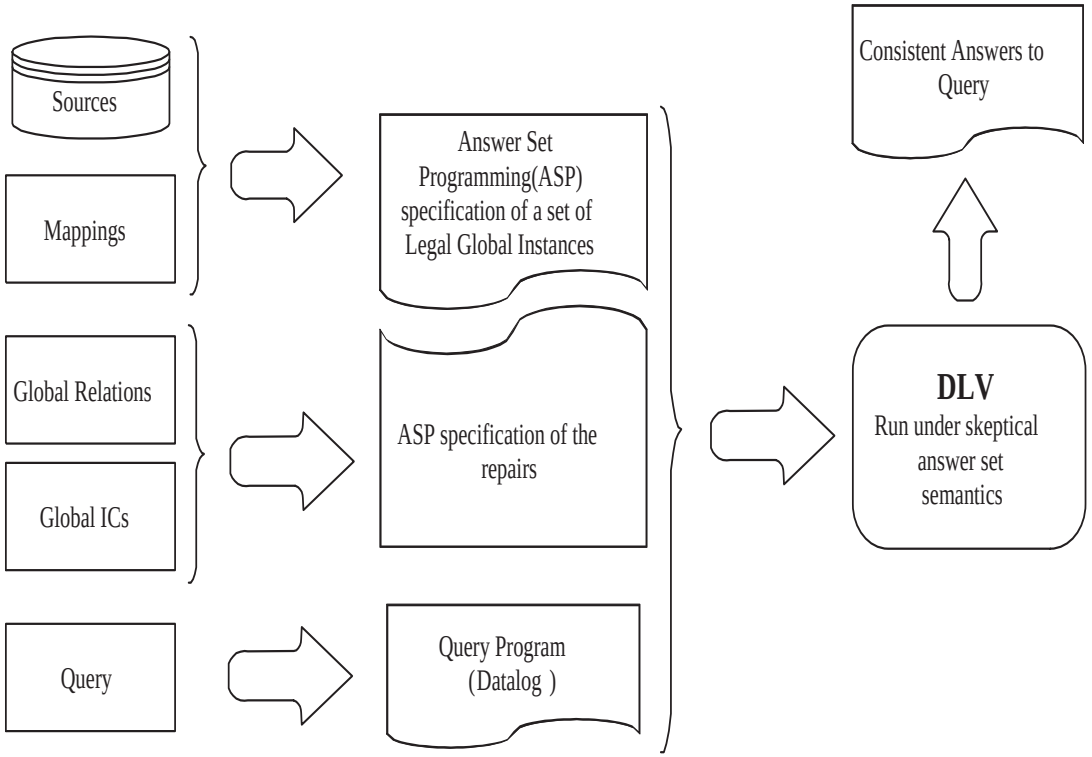
Example: Data integration system $\mathcal{G}$ under LAV and open sources

$$S_1(X,Y) \leftarrow R(X,Y) \quad \text{with} \quad s_1 = \{(a,b),(c,d)\}$$

$$S_2(X,Y) \leftarrow R(Y,X) \quad \text{with} \quad s_2 = \{(c,a),(e,d)\}$$

Consistent query answering wrt global schema?

Want to impose on $\mathcal{G}$, at query time, the FD: $R: \quad X \rightarrow Y$



```
% subprogram minimal legal instances:

    domd(a).    domd(b).    domd(c).
    domd(d).    domd(e).    v1(a,b).
    v1(c,d).    v2(c,a).    v2(e,d).

    R(X,Y,d) :- v1(X,Y).
    R(Y,X,d) :- v2(X,Y).

% repair subprogram:

    R(X,Y,f) v R(X,Z,f) :-  R(X,Y,d), R(X,Z,d), Y!=Z,
                            domd(X),domd(Y),domd(Z).

    R(X,Y,t**) :- R(X,Y,d), domd(X), domd(Y), not R(X,Y,f).

% query program:

    Ans(X,Y) :-  R(X,Y,t**).
```

Complexity of CQA

Some Immediate Results

- When a FO query rewriting approach works (e.g. correct and finitely terminating in case of T^ω), consistent answers to FO queries can be computed in *PTIME* in data

That is, for fixed queries and ICs, but varying database instances (and their sizes)

- The problem of CQA is a decision problem:

$$CQA(\mathcal{Q}(\bar{x}), IC) := \{(D, \bar{t}) \mid D \models_{IC} \mathcal{Q}(\bar{t})\}$$

Query answering from disjunctive logic programs under skeptical stable models semantics is Π_2^P-complete in data

(Dantsin, Eiter, Gottlob, Voronkov; ACM CSs 2001)

This provides an upper bound for data complexity of CQA

- The problem of repair checking is

$$RCh(IC) := \{(D, D') \mid D' \text{ is a repair of } D \text{ wrt } IC\}$$

Complexity of checking if a subset of HB is a stable model of a program is *coNP*-complete in data

This provides an upper bound for repair checking

- There are classes of disjunctive programs for which these decision problems have lower complexity

The head-cycle free programs (HCF): defined in terms of a directed graph $G(\Pi)$

  - Each (ground) literal is a node
  - Arch from A to A' iff there is a rule in which A appears positive in the body and A' in the head

- Π is HCF iff $G(\Pi)$ has no cycles with two literals belonging to the head of the same rule
- Π is HCF if its ground version is HCF
- In this case, Π can be transformed into a (non-disjunctive) normal program with the same stable models

For HCF programs, query answering under skeptical semantics becomes *coNP*-complete; and stable model checking in *PTIME*

For some classes of ICs, repair programs become HCF

For sets IC of denial constraints, $\Pi(D, IC)$ is HCF

$$\forall xyz\neg(R(x,y) \wedge S(y,z) \wedge T(x,z) \wedge x > 5 \wedge z \neq y)$$

And we have better upper bounds for the two decision problems above

Closing and Understanding the Gap

The complexity bounds above leave plenty of room for a potentially lower data complexity

First explicit analysis of complexity of CQA was done for atomic scalar aggregate queries and FDs (not given here)
(Arenas, Bertossi, Chomicki; ICDT 2001)

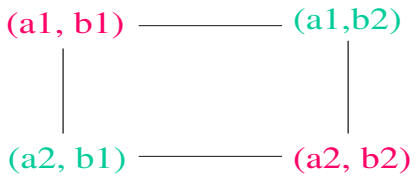
Graph-theoretic methods were applied

Given a set of FDs FD and an instance D, the conflict graph $CG_{FD}(D)$ is an undirected graph:

- Vertices are the tuples $R(\bar{t})$ in D
- Edges are of the form $\{R(\bar{t}_1), R(\bar{t}_2)\}$ for which there is a dependency in FD that is simultaneously violated by $R(\bar{t}_1), R(\bar{t}_2)$

Example: Schema $R(A, B)$ FDs: $A \rightarrow B$ and $B \rightarrow A$

Instance $D = \{R(a_1, b_1), R(a_1, b_2), R(a_2, b_2), R(a_2, b_1)\}$



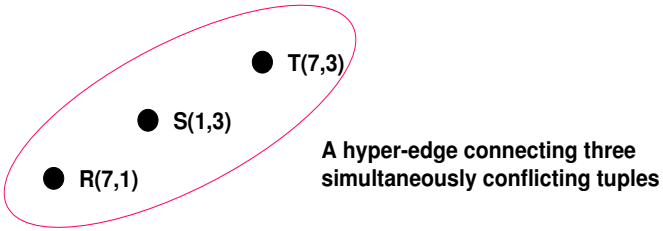
Repairs: $D_1 = \{(a_1, b_1), (a_2, b_2)\}$ and $D_2 = \{(a_1, b_2), (a_2, b_1)\}$

Each repair of D corresponds to a maximal independent set in $CG_{FD}(D)$

Each repair of D corresponds to a maximal clique in the complement of $CG_{FD}(D)$

This graph-theoretic analysis was applied to the complexity of FO conjunctive queries and FDs

And also to denial constraints; actually using conflict hypergraphs (Chomicki et al.; I&C 2005)



$$\forall xyz\neg(R(x,y) \wedge S(y,z) \wedge T(x,z) \wedge x > 5 \wedge z \neq y)$$

Vertices are the DB tuples, and their simultaneous semantic conflicts under denial ICs are the hyperedges

As before, repairs correspond to set-theoretically maximal independent sets

In a series of papers and using graph-theoretic methods, *PTIME* algorithms for CQA were provided

Graph-theoretic methods applied to repairs (conflict graphs or hypergraphs) or to syntactic structure of queries

In all those cases, the query was also FO rewritable for CQA

In those cases where CQA can be solved in *PTIME*, repair checking can be solved in *PTIME* too

1. FDs and projection-free conjunctive queries

   Also some conjunctive queries with limited projection

   Also: Repair checking is *PTIME* for arbitrary FDs and acyclic inclusion dependencies
   (deletion-based repair semantics) (Chomicki et al., I&C 2005)

2. Key Constraints (KCs) and some syntactic classes of conjunctive queries with restricted projection
   (Fuxman, Miller; ICDT 2005)

   Classes defined by the graph-theoretic syntactic structure of the query and its interaction with the KCs

   Actually, for every query $\mathcal{Q}$ in their class, $\mathcal{C}_{forest}$, there is a FO rewriting $\mathcal{Q}'$ for CQA

   $$\mathcal{Q}: \ \exists x \exists y \exists z (R(\underline{x}, z) \wedge S(\underline{z}, y)) \ \mapsto$$
   $$\mathcal{Q}': \ \exists x \exists z' (R(\underline{x}, z') \wedge \forall z (R(\underline{x}, z) \rightarrow \exists y S(\underline{z}, y)))$$

3. As in 2., but extending the class of queries (*rooted queries*)

   Same property of FO rewritability (Wijsen; DBPL 2007)

Classes of queries above are rather sharp, i.e. not satisfying some of their syntactic conditions increases complexity

What about lower bounds?

- For arbitrary FDs and inclusion dependencies (deletions only)

(Chomicki et al., I&C 2005)

  - Repair checking becomes *coNP*-complete
  - CQA becomes Π_2^P-complete

- For KCs and conjunctive queries (with some forms of projection) CQA becomes *coNP*-complete

$$\mathcal{Q}: \ \exists z \exists y \exists z (R(\underline{x}, z) \wedge S(\underline{y}, z))$$

(Chomicki et al.; I&C 2005), (Fuxman et al.; ICDT 2005), (Wijsen; DBPL 2007)

• For arbitrary FDs and inclusion dependencies (including RICs), CQA becomes undecidable (Cali et al.; Pods 2005)

Issues:

- Inclusion dependencies repaired through insertions
- Infinite underlying domain that can be used for insertions
- Cycles in the set of inclusion dependencies

• However, for arbitrary universal ICs and RICs (even cyclic), CQA becomes Π_2^P-complete if RICs are repaired with non propagating SQL nulls (Bravo, Bertossi; IIDB 2006)

Other Complexity Results

- FO Rewriting vs. *PTIME*

There are sets of KCs $\mathcal{K}$ and conjunctive queries $\mathcal{Q}$ for which CQA is in *PTIME*, but there is no FO rewriting of $\mathcal{Q}$ for CQA

  - $\mathcal{Q}: \ \exists x \exists y \exists z (R(\underline{x}, z) \wedge R(\underline{y}, z) \wedge x \neq y)$

    Reduction techniques (Fuxman, Miller; IIWeb 2003)

  - $\mathcal{Q}: \ \exists x \exists y (R(\underline{x}, y) \wedge R(\underline{y}, c))$

    Using Ehrenfeucht-Fraisse games (Wijsen; DBPL 2007)

• Not Only Data Complexity (Arenas, Bertossi; unpublished)

Relational schema $\mathcal{S}$, IC finite set of ICs, D database instance, $\mathcal{Q}$ a boolean query:

$$\begin{aligned} d\text{-}CQA(IC, \mathcal{Q}) &= \{D \mid D \models_{IC} \mathcal{Q}\} \\ ic\text{-}CQA(D, \mathcal{Q}) &= \{IC \mid D \models_{IC} \mathcal{Q}\} \\ q\text{-}CQA(D, IC) &= \{\mathcal{Q} \mid D \models_{IC} \mathcal{Q}\} \end{aligned}$$

- There are $\mathcal{S}, D, \mathcal{Q}$ with $ic\text{-}CQA(D, Q)$ undecidable

  Reduction from SAT^c for finite structures

  Negative literal query; ICs in correspondence with FO sentences checked for non-satisfaction

- For any schema $\mathcal{S}$ with domain $\mathbb{N}$, there are schema $\mathcal{S}' \supset \mathcal{S}$ with same domain and $<$, D over $\mathcal{S}'$, and IC in $L(\mathcal{S}')$, with $q\text{-}CQA(D, IC)$ undecidable

  Reduction from SAT to $q\text{-}CQA(D, IC)^c$

- There are schema $\mathcal{S}$, with domain containing $\mathbb{N}$ and $<$, IC, and query $\mathcal{Q}$ with d-$CQA(IC, Q)$ undecidable

  Encode halting problem for Turing machines

  ICs are universal

- For finite, universal and domain independent IC and domain independent FO queries $\mathcal{Q}(\bar{x})$: easily ...

  $CQA := \{(IC, D, Q(\bar{x}), \bar{t}) \mid D \models_{IC} Q(\bar{t})\}$ is decidable

  An extreme "combined" case of CQA; naive algorithm is exponential

- There are $\mathcal{S}, \mathcal{Q}$ such that, for domain independent universal ICs φ, the combined problem

$$(d, ic)\text{-}CQA(\mathcal{Q}) := \{(D, \{\varphi\}) \mid D \models_{\{\varphi\}} \mathcal{Q}\}$$

  is *coNEXP*-complete

  Reduction from SAT for Bernays-Schoenfinkel's class of FO sentences to $CQA(\mathcal{Q})^c$

  (Actually, a subclass with same lower bound that allows for specification of bounded tiling problems)

Getting More from ASPs

- Complexity of query evaluation from disjunctive logic programs (DLPs) coincides with the complexity of CQA

- However, for some classes of queries and ICs, CQA has a lower complexity, e.g. in *PTIME*

- The landscape between FO rewritable cases and Π_2^P-completeness for CQA still not quite clear

- Results obtained in the middle ground are scattered, isolated, and rather ad hoc

- The "logics" of CQA is not fully understood yet

Some natural questions arise ...

- Can we identify classes of ICs and queries for which repair programs can be automatically "simplified" into queries of lower complexity?
  - Can we reobtain previous classes?
  - Can we identify new ones?
  - Can we obtain new complexity results?
- Can we better understand the logic of CQA through the analysis of repair programs?
- Can we take advantage of results about updates of LPs to deal with the problem of CQA under updates? (almost untouched problem)

Some progress in this research program ...

Repair Programs and Circumscription

Example: $P(X,Y):\ X \to Y$

$D = \{P(a,b), P(a,c), P(d,e)\}$

Repair program:

$$P(x,y,\mathbf{f}) \vee P(x,z,\mathbf{f}) \leftarrow P(x,y,\mathbf{d}), P(x,z,\mathbf{d}), y \neq z$$

$$P(x,y,\mathbf{t}^{**}) \leftarrow P(x,y,\mathbf{d}), not\ P(x,y,\mathbf{f})$$

$$P(a,b,\mathbf{d}).\ \ P(a,c,\mathbf{d}).\ \ P(d,e,\mathbf{d}).$$

This program can be seen as a FO specification (forget about the stable model semantics), i.e. a FO conjunction Ψ_ρ of

$$P(x,y) \wedge P(x,z) \wedge y \neq z \rightarrow P_f(x,y) \vee P_f(x,z)$$

$$P(x,y) \wedge \neg P_f(x,y) \rightarrow P_{\star\star}(x,y)$$

$$P(a,b) \wedge P(a,c) \wedge P(d,e)$$

Quite recently a stable model semantics has been introduced for any FO sentence (Ferraris, Lee, Lifschitz; IJCAI 2007)

$\Psi \quad \mapsto \quad \Psi'$ and Ψ' is a SO sentence (same signature)

The stable models of Ψ are the Herbrand models of Ψ' (the stable sentence)

For DLPs, this "stable semantics" coincides with their original stable model semantics

In our case:

- The stable sentence for a repair program (as FO sentence Ψ_ρ) is always a circumscription

Parallel circumscription of all the predicates in the program

- Given the structure of the repair program (also including the query program), the circumscription becomes a prioritized circumscription

In the example, minimize predicates in this order: database predicates, predicates annotated with $\mathbf{f}$, predicates annotated with $\mathbf{t}^{\star\star}$, the Ans predicate

- Most complex is minimization of predicates defined by disjunctive rules (those associated to the ICs)

For all the others we can apply predicate completion

In the example, Ψ'_ρ becomes

$$\forall xy(P(x,y) \equiv (x = a \wedge y = b) \vee (x = a \wedge y = c) \vee (x = d \wedge y = e))$$
$$\wedge \ \ \forall xy((P(x,y) \wedge \neg P_f(x,y)) \equiv P_{\star\star}(x,y)) \ \wedge$$
$$\forall xyz(P(x,y) \wedge P(x,z) \wedge y \neq z \rightarrow (P_f(x,y) \vee P_f(x,z)) \ \ \wedge$$
$$\neg\exists U((U < P_f) \wedge \forall xyz(P(x,y) \wedge P(x,z) \wedge y \neq z \rightarrow$$
$$(U(x,y) \vee U(x,z))) \qquad (*)$$

- Predicate P_f minimized via the last conjunct (*) of Ψ'_ρ
- $U < P_f$ stands for
  $\forall xy(U(x,y) \rightarrow P_f(x,y)) \wedge \exists xy(P_f(x,y) \wedge \neg U(x,y))$

Consistent query answering? $\qquad \mathcal{Q}(x,y) : \ P(x,y)$

$$\Psi'_\rho \ \wedge \ \forall x \forall y(Ans(x,y) \equiv P_{\star\star}(x,y)) \ \ \stackrel{?}{\models} \ \ Ans(x,y)$$

Classical logical entailment! $\qquad$ Anything else?

Eliminate SO quantifiers from Ψ'_ρ ...

(Doherty, Lukaszewicz, Szalas; JAR 1997)

Let $\kappa(x,y,z)$ stand for $P(x,y) \wedge P(x,z) \wedge y \neq z$

The negation of (*) is logically equivalent to

$$\exists st \exists f \exists U \forall x \forall r (\forall x_1 y_1 z_1 (\neg\kappa(x_1,y_1,z_1) \vee f(x_1,y_1,z_1) = \vee(y_1,z_1))$$
$$\wedge\ \forall yz(\neg\kappa(x,y,z) \vee r \neq f(x,y,z) \vee U(x,r)))$$
$$\wedge\ \forall uv(\neg U(u,v) \vee P_f(u,v)) \wedge (P_f(s,t) \wedge \neg U(s,t)))$$

($t = \vee(t_1,t_2)$ stands for $t = t_1 \vee t = t_2$)

Now we are ready to apply Ackermann's lemma

The formula is of the form

$$\exists st \exists f \exists U \forall x \forall r ((A(x,r) \vee U(x,r)) \wedge B(\neg U \mapsto U)) \qquad (**)$$

$B(\neg U \mapsto U)$ is formula B with predicate U replaced by $\neg U$

$$A(x,r): \quad \forall yz(\forall yz(\neg\kappa(x,y,z) \vee r \neq f(x,y,z))$$
$$B(U): \quad \forall x_1y_1z_1(\neg\kappa(x_1,y_1,z_1) \vee f(x_1,y_1,z_1) = \vee(y_1,z_1)) \wedge$$
$$\forall uv(U(u,v) \vee P_f(u,v)) \wedge (P_f(s,t) \wedge U(s,t)))$$

B is positive in U

The subformula in (**) starting with $\exists U$ can be equivalently replaced by $B(A(x,r) \mapsto U)$, eliminating U:

$$\exists st \exists f(\forall x_1y_1z_1(\neg\kappa(x_1,y_1,z_1) \vee f(x_1,y_1,z_1) = \vee(y_1,z_1)) \wedge$$
$$\forall uv(\forall yz(\neg\kappa(u,y,z) \vee v \neq f(u,y,z) \vee P_f(u,v)) \wedge$$
$$(P_f(s,t) \wedge \forall y_1z_1(\neg\kappa(s,y_1,z_1) \vee t \neq f(s,y_1,z_1)))$$

Unskolemizing:

$$\exists st \forall xyz \exists w((\neg\kappa(x,y,z) \vee w = \vee(y,z)) \wedge (\neg\kappa(x,y,z) \vee P_f(u,w))$$
$$\wedge (P_f(s,t) \wedge (x \neq s \vee \neg\kappa(x,y,z) \vee t \neq w)))$$

Its negation is equivalent (via other conjuncts in page 57) to

$$\forall st(P_f(s,t) \rightarrow \exists xyz \forall w(\kappa(x,y,z) \wedge (P_f(x,w) \rightarrow (x = s \wedge t = w))))),$$

which can be replaced for (*) in page 57, obtaining an equivalent FO specification of predicate P_f ... and a FO theory Ψ''_ρ to do CQA with

$$\mathcal{Q}(x,y): \ P(x,y)$$

$$\Psi''_\rho \ \wedge \ \forall xy(Ans(x,y) \equiv P_{\star\star}(x,y)) \quad \stackrel{?}{\models} \quad Ans(x,y)$$

Classical FO entailment!

In this case, by simple logical transformation, equivalent to

$$D \models P(x,y) \wedge \neg \exists z(P(x,z) \wedge z \neq y)$$

Reobtaining the original FO rewriting!

Final Remarks and Ongoing Research

- For FDs (and KCs), this methodology provably works

Has to be exploited now ...

- FO rewritings in (Arenas et al.; 1999) mentioned before can be reobtained

- The query is posed on top of a FO specification of repairs

The database (its completion) lies at the bottom

Like doing query answering in DBs with complex, expressive FO views

- Interesting to investigate the kind of FO theories obtained

- The FO specification of repairs can be used to reobtain FO and other new rewritings for CQA

- Use FO theory to analyze complexity of CQA

- Ackermann's Lemma can be extended and SO quantifier elimination produces a Fixpoint formula (Nonnengart, Szalas; 1998)

Relevant cases in CQA?

Relevant for *PTIME* vs. FO rewriting?

- A lot of fun to come ...!

THE END