Consistent Query Answering in Databases

Leopoldo Bertossi

Carleton University
School of Computer Science
Ottawa, Canada

bertossi@scs.carleton.ca
www.scs.carleton.ca/~bertossi

University of Akureyri, Iceland, April 2005.

The Context

There are situations when we want/need to live with inconsistent information in a database

With information that contradicts given integrity constraints

- The DBMS does not fully support data maintenance or integrity checking/enforcing
- The consistency of the database will be restored by executing further compensating transactions or future transactions
- Integration of heterogeneous databases without a central/global maintenance mechanism

- Inconsistency wrt “soft” or “informational” integrity constraints we hope or expect to see satisfied, but are not maintained
- User constraints than cannot be checked
- Legacy data on which we want to impose (new) semantic constraints

It may be impossible/undesirable to repair the database (to restore consistency)

- No permission
- Inconsistent information can be useful
- Restoring consistency can be a complex and non deterministic process

The Problem

Not all data participate in the violation of the ICs

The inconsistent database can still give us "correct" or consistent answers to queries!

We need:

- A precise definition of consistent answer to a query in an inconsistent database
- Mechanisms for obtaining such consistent information from the inconsistent database
- Understanding of the computational complexity of the problem

Example

A database instance D

$Employee$	$Name$	$Salary$
	$J.Page$	5,000
	$J.Page$	8,000
	$V.Smith$	3,000
	$M.Stowe$	7,000

FD: $Name \rightarrow Salary$

D violates FD, by the tuples with $J.Page$ in $Name$

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

D_1

Employee	*Name*	*Salary*
	J.Page	5,000
	V.Smith	3,000
	M.Stowe	7,000

D_2

Employee	*Name*	*Salary*
	J.Page	8,000
	V.Smith	3,000
	M.Stowe	7,000

$(M.Stowe, 7,000)$ persists in all repairs, and it does not participate in the violation of FD; it is invariant under minimal forms of restoration of consisency

$(J.Page, 8,000)$ does not persist in all repairs, and it does participate in the violation of FD

Repairs and Consistent Answers

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

Definition: (Arenas, Bertossi, Chomicki; PODS 99)

A repair of a database instance D is a database instance D'

- over the same schema and domain
- satisfies IC
- differs from D by a minimal set of changes (insertions or deletions of tuples) wrt set inclusion

Given a query $Q(\bar{x})$ to D, we want as answers all and only those tuples obtained from D that are "consistent" wrt IC (even when D globally violates IC)

Definition: (Arenas, Bertossi, Chomicki; PODS 99)

A tuple $\bar{t}$ is a consistent answer to query $Q(\bar{x})$ in D iff
$\bar{t}$ is an answer to query $Q(\bar{x})$ in every repair D' of D:

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

A model theoretic definition ...

Example

Inconsistent DB instance D wrt FD: $Name \rightarrow Salary$

Employee	*Name*	*Salary*
	$J.Page$	5,000
	$J.Page$	8,000
	$V.Smith$	3,000
	$M.Stowe$	7,000

Repairs D_1, resp. D_2

Employee	*Name*	*Salary*
	$J.Page$	5,000
	$V.Smith$	3,000
	$M.Stowe$	7,000

Employee	*Name*	*Salary*
	$J.Page$	8,000
	$V.Smith$	3,000
	$M.Stowe$	7,000

$$D \models_{FD} Employee(M.Stowe, 7,000)$$

$D \models_{FD} (Employee(J.Page, 5,000) \vee Employee(J.Page, 8,000))$

$D \models_{FD} \exists X\, Employee(J.Page, X)$

We can see this is not the same as getting rid of the tuples that participates in the violation of the IC

More information is preserved than with (naive) data cleaning

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 computation of all database repairs is not possible/sensible/feasible

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!

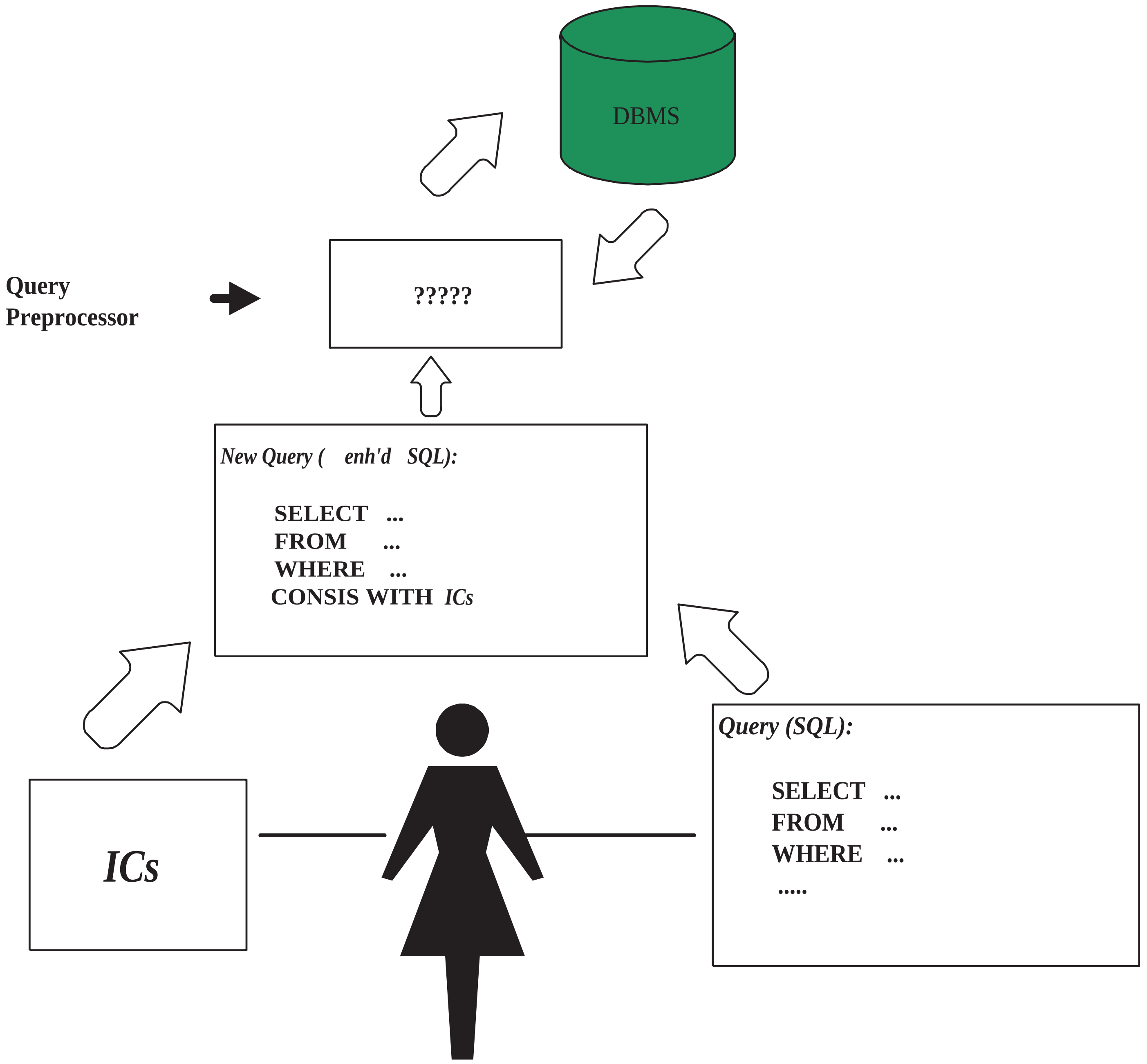
Query Transformation

First-Order queries and constraints

Idea:

- Do not compute the repairs
- Query only the available inconsistent database instance
- Transform the query and pose the new query (as usual)

(Arenas, Bertossi, Chomicki; PODS 99)
(Celle, Bertossi; DOOD 00)


DBMS
Query
Preprocessor
?????
New Query (enh'd SQL):
SELECT ...
FROM ...
WHERE ...
CONSIS WITH ICs
ICs
Query (SQL):
SELECT ...
FROM ...
WHERE ...
.....

Given a query Q to the inconsistent DB D, qualify Q with appropriate information derived from the interaction between Q and the ICs

- To locally satisfy the ICs
- To discriminate between tuples in the answer set
- Inspired by "Semantic Query Optimization" techniques

Consistent answers to $Q(\bar{x})$ in D??

Rewrite query: $Q(\bar{x}) \longmapsto Q'(\bar{x})$

$Q'(\bar{x})$ is a new first order query

Retrieve from D the (ordinary) answers to $Q'(\bar{x})$

Example

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

1. Query to D: $Q(x)?$

If $Q(x)$ holds in D, then $P(x) \rightarrow Q(x)$ holds in D

Elements in Q do not participate in a violation of IC

2. Query: $P(x)?$

If $P(x)$ holds in D, then $Q(x)$ must hold in D in order to satisfy $P(x) \rightarrow Q(x)$

An answer x to "$P(x)$?" is consistent if x is also in table Q

Transform query 2. into: $P(x) \wedge Q(x)$?

Pose this query instead

$Q(x)$ is a residue of $P(x)$ wrt $\forall x(P(x) \rightarrow Q(x))$

Residue can be obtained by resolution between the query literal and IC

Posing new query to D we get only answer $\{b\}$

For query $Q(x)?$ there is no residue, i.e. every answer to query $Q(x)?$ is also a consistent answer, i.e. we get $\{b, c\}$

3. Query $\neg Q(x)$? (not safe, just for illustration)

Residue wrt $\forall x(P(x) \rightarrow Q(x))$ is $\neg P(x)$

New query: $\neg Q(x) \wedge \neg P(x)$?

Answers to this new query (in the active domain): $\emptyset$

No consistent answers ...

Example

FD: $\forall XYZ\ (\neg Employee(X,Y)\ \vee\ \neg Employee(X,Z) \vee$
$Y = Z)$

Query: $Employee(X,Y)$?

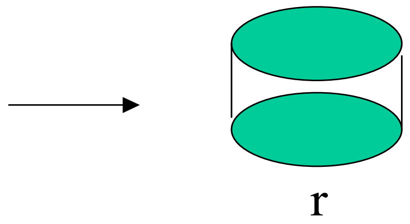
Consistent answers: $(V.Smith,$ 3,000), $(M.Stowe,$ 7,000)
(but not $(J.Page,$ 5,000), $(J.Page,$ 8,000))

Can be obtained by means of the transformed query

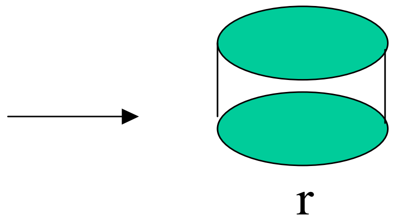
$$T(Q(X,Y)) := Employee(X,Y)\ \wedge\ \forall Z\ (\neg Employee(X,Z)\ \vee\ Y = Z)$$

... those tuples (X,Y) in the relation for which X does not have and associated Z different from Y ...

```
SELECT  Name, Salary
FROM     Employee
CONSISTENT WITH
        FD(Name;Salary)
```



```
SELECT  Name, Salary
FROM     Employee  E
WHERE   Not exists (
        SELECT E.Salary
        FROM    E
        WHERE  E.Name = Name
           AND  E.Salary <> Salary)
```



Again, the residue $\forall Z\ (\neg Employee(X, Z) \vee Y = Z)$ can be automatically obtained by applying resolution to the query and FD

In general, T is an iterative operator

Example

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

Query: $Q(x)$?

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

Apply T again, now to the appended residues

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

$$\begin{aligned} T^2(\varphi(x)) = Q(x) \ \wedge \ & (R(x) \vee (\neg P(x) \wedge \neg Q(x))) \ \wedge P(x) \wedge \\ & (R(x) \vee \neg Q(x)) \end{aligned}$$

And again:

$$\begin{aligned}T^3(Q(x)) := Q(x) \ \wedge \ & (R(x) \vee (\neg P(x) \wedge T(\neg Q(x)))) \ \wedge \\ & P(x) \wedge (T(R(x)) \vee T(\neg Q(x)))\end{aligned}$$

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

$$T^3(Q(x)) \ = \ T^2(Q(x))$$

A finite fixed point! Does it always exist?

In general, an infinitary query: $T^\omega(\varphi(x)) \ := \ \bigcup_{n<\omega} \{T_n(\varphi(x))\}$

In the example, $T^\omega(Q(x)) \ = \ \{T_1(Q(x)), T_2(Q(x))\}$

Always finite?

Some Results

There are sufficient conditions on queries and ICs for soundness and completeness of operator T (ABC; PODS 99)

- Soundness: every tuple computed via T is consistent in the semantic sense

$$D \models T_\omega(\varphi)[\bar{t}] \implies D \models_{IC} \varphi[\bar{t}]$$

- Completeness: every semantically consistent tuple can be obtained via T

$$D \models_{IC} \varphi[\bar{t}] \implies D \models T_\omega(\varphi)[\bar{t}]$$

Natural and useful syntactical classes satisfy the conditions

There are necessary and sufficient conditions for syntactic termination

- In the iteration process to determine $T^\omega(Q)$ nothing syntactically new is obtained beyond some finite step

There are sufficient conditions for semantic termination

- From some finite step on, only logically equivalent formulas are obtained

In these favorable cases, a FO SQL query can be translated into a new FO SQL query that is posed as usual to the database

In these cases CQA can be computed in polynomial time in data complexity

Some Limitations

First-order query rewriting based approaches to CQA provably have intrinsic limitations (see later)

They are incomplete for full FO queries and ICs, which applies in particular to T

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

FO query reformulation has been slightly extended using other methods; still keeping polynomial time data complexity

- Hypergraph representation of the DB (the vertices) and its semantic conflicts (the hyperedges) under denial ICs

  Graph based algorithms on original query can be translated into SQL queries
  (Chomicki, Marcinkowski, Staworko; demos at EDBT 04)

- Specific methods for conjunctive queries containing restricted projections (existential quantifiers) and FDs
  (Fuxman, Miller; ICDT05)

In the general case of FO ICs and queries, rewriting based approaches to CQA must appeal to languages that are much more expressive than FO logic (see later)

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: $\mathit{Spec} \overset{?}{\models} Q(\bar{x})$
    - Derive algorithms for consistent query answering

Consistent query answering is non-monotonic; then a non-monotonic semantics for Spec is expected

Specifying Repairs with Logic Programs

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

Use disjunctive logic programs with stable model semantics (Barcelo, Bertossi; PADL 03)

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 $(\mathbf{t_a}, \mathbf{f_a})$, use them to give feedback to the program in case additional changes become necessary $(\mathbf{t}^\star, \mathbf{f}^\star)$; and to collect the tuples in the final, repaired instances $(\mathbf{t}^{\star\star}, \mathbf{f}^{\star\star})$

Annotation	Atom	The tuple $P(\bar{a})$ is ...
$\mathbf{t_d}$	$P(\bar{a}, \mathbf{t_d})$	a fact of the database
$\mathbf{f_d}$	$P(\bar{a}, \mathbf{f_d})$	not a fact in the database
$\mathbf{t_a}$	$P(\bar{a}, \mathbf{t_a})$	advised to be made true
$\mathbf{f_a}$	$P(\bar{a}, \mathbf{f_a})$	advised to be made false
$\mathbf{t}^\star$	$P(\bar{a}, \mathbf{t}^\star)$	true or becomes true
$\mathbf{f}^\star$	$P(\bar{a}, \mathbf{f}^\star)$	false or becomes false
$\mathbf{t}^{\star\star}$	$P(\bar{a}, \mathbf{t}^{\star\star})$	true in the repair
$\mathbf{f}^{\star\star}$	$P(\bar{a}, \mathbf{f}^{\star\star})$	false in the repair

Example: Full inclusion dependency $IC\colon\ \forall \bar{x}(P(\bar{x}) \rightarrow Q(\bar{x}))$

Inconsistent instance $D = \{P(\bar{c}), P(\bar{d}), Q(\bar{d}), Q(\bar{e})\}$

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

1. Original data as facts: $P(\bar{c}, \mathbf{t_d}), P(\bar{d}, \mathbf{t_d}), Q(\bar{d}, \mathbf{t_d}), Q(\bar{e}, \mathbf{t_d})$
2. Whatever was true (false) or becomes true (false), gets annotated with $\mathbf{t}^\star$ $(\mathbf{f}^\star)$:

$$P(\bar{x}, \mathbf{t}^\star) \leftarrow P(\bar{x}, \mathbf{t_d})$$
$$P(\bar{x}, \mathbf{t}^\star) \leftarrow P(\bar{x}, \mathbf{t_a})$$
$$P(\bar{x}, \mathbf{f}^\star) \leftarrow \ \mathit{not}\ P(\bar{x}, \mathbf{t_d})$$
$$P(\bar{x}, \mathbf{f}^\star) \leftarrow P(\bar{x}, \mathbf{f_a})$$

... the same for Q ...

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

   We need annotation constants for the local changes $(\mathbf{t_a}, \mathbf{f_a})$, but also annotations $(\mathbf{t}^\star, \mathbf{f}^\star)$ to provide feedback to the rules that produce local repair steps

   $$P(\bar{x}, \mathbf{f_a}) \ \vee \ Q(\bar{x}, \mathbf{t_a}) \ \leftarrow \ P(\bar{x}, \mathbf{t}^\star), Q(\bar{x}, \mathbf{f}^\star)$$

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

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

4. Repairs must be coherent: Use denial constraints at the program level to prune undesirable models

$$\leftarrow P(\bar{x}, \mathbf{t_a}), P(\bar{x}, \mathbf{f_a})$$
$$\leftarrow Q(\bar{x}, \mathbf{t_a}), Q(\bar{x}, \mathbf{f_a})$$

5. Annotations constants $\mathbf{t}^{\star\star}$ and $\mathbf{f}^{\star\star}$ are used to read off the literals that are inside (outside) a repair

$$P(\bar{x}, \mathbf{t}^{\star\star}) \leftarrow P(\bar{x}, \mathbf{t_a})$$
$$P(\bar{x}, \mathbf{t}^{\star\star}) \leftarrow P(\bar{x}, \mathbf{t_d}),\ \ not\ P(\bar{x}, \mathbf{f_a})$$
$$P(\bar{x}, \mathbf{f}^{\star\star}) \leftarrow P(\bar{x}, \mathbf{f_a})$$
$$P(\bar{x}, \mathbf{f}^{\star\star}) \leftarrow\ not\ P(\bar{x}, \mathbf{t_d}),\ \ not\ P(\bar{x}, \mathbf{t_a})\ \ \text{... etc.}$$

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

$$\{P(\bar{c},\mathbf{t_d}), ..., P(\bar{c},\mathbf{t}^\star), Q(\bar{c},\mathbf{f}^\star), Q(\bar{c},\mathbf{t_a}), P(\bar{c},\mathbf{t}^{\star\star}), Q(\bar{c},\mathbf{t}^\star),$$
$$P(d,\mathbf{t}^{\star\star}), Q(d,\mathbf{t}^{\star\star}), \ldots, Q(\bar{c},\mathbf{t}^{\star\star}), ...\} \quad \equiv$$
$$\{P(\bar{c}), Q(\bar{c}), P(\bar{d}), Q(\bar{d}), Q(\bar{e})\}$$

... insert $Q(\bar{c})$!!

$$\{P(\bar{c},\mathbf{t_d}), ..., P(\bar{c},\mathbf{t}^\star), P(\bar{c},\mathbf{f}^\star), Q(\bar{c},\mathbf{f}^\star), P(\bar{c},\mathbf{f}^{\star\star}), Q(\bar{c},\mathbf{f}^{\star\star}),$$
$$P(d,\mathbf{t}^{\star\star}), Q(d,\mathbf{t}^{\star\star}), \ldots, P(\bar{c},\mathbf{f_a}), ...\} \quad \equiv$$
$$\{P(\bar{d}), Q(\bar{d}), Q(\bar{e})\}$$

... delete $P(\bar{c})$!!

To obtain consistent answers to a FO query:

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

2. Run the query program together with the specification program

   ... under the skeptical or cautious stable model semantics that sanctions as true of a program what is true of all its stable models

Example: (continued)

Consistent answers to query $P(\bar{x}) \wedge \neg Q(\bar{x})$?

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

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

The two previous stable models become extended with ground Ans atoms

None of them in the intersection of the two models

In consequence, under the skeptical stable model semantics, $Ans = \emptyset$, i.e. no consistent answers, as expected ...

Remarks:

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

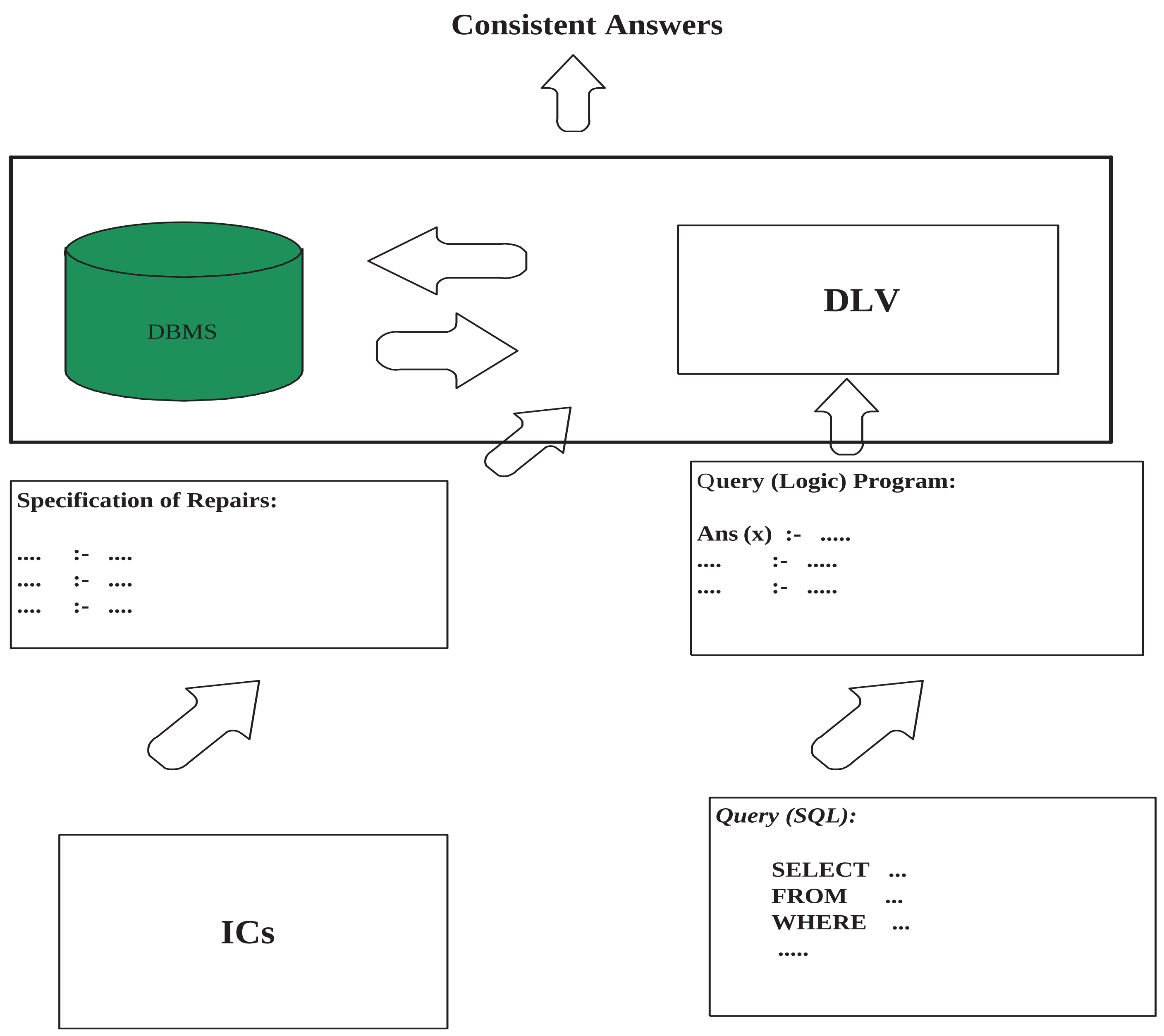
  One to one correspondence between repairs and stable models of the program

- Existential ICs, like referential ICs, can be handled, with different repair policies, e.g. introduction of null values, cascaded deletions, ...
  (Barcelo,Bertossi,Bravo; LNCS 2582)
  (Bravo,Bertossi; CASCON 04)

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

  The query at hand adds a final layer on top (obtaining a split program)

- The program can be optimized in several ways; e.g. avoiding materialization of CWA (Barcelo,Bertossi,Bravo; LNCS 2582), annotations of DB facts, etc.
- We have successfully experimented with the DLV system for computing the stable models semantics
  (N. Leone et al.; ACM Transactions on Comp. Logic)
- Related methodologies:
  (Arenas, Bertossi, Chomicki; TPLP 03)
  (Greco, Greco, Zumpano; IEEE TKDE 03)


Consistent Answers
DBMS
DLV
Specification of Repairs:
.... :- ....
.... :- ....
.... :- ....
Query (Logic) Program:
Ans (x) :- .....
.... :- .....
.... :- .....
ICs
Query (SQL):
SELECT ...
FROM ...
WHERE ...
.....

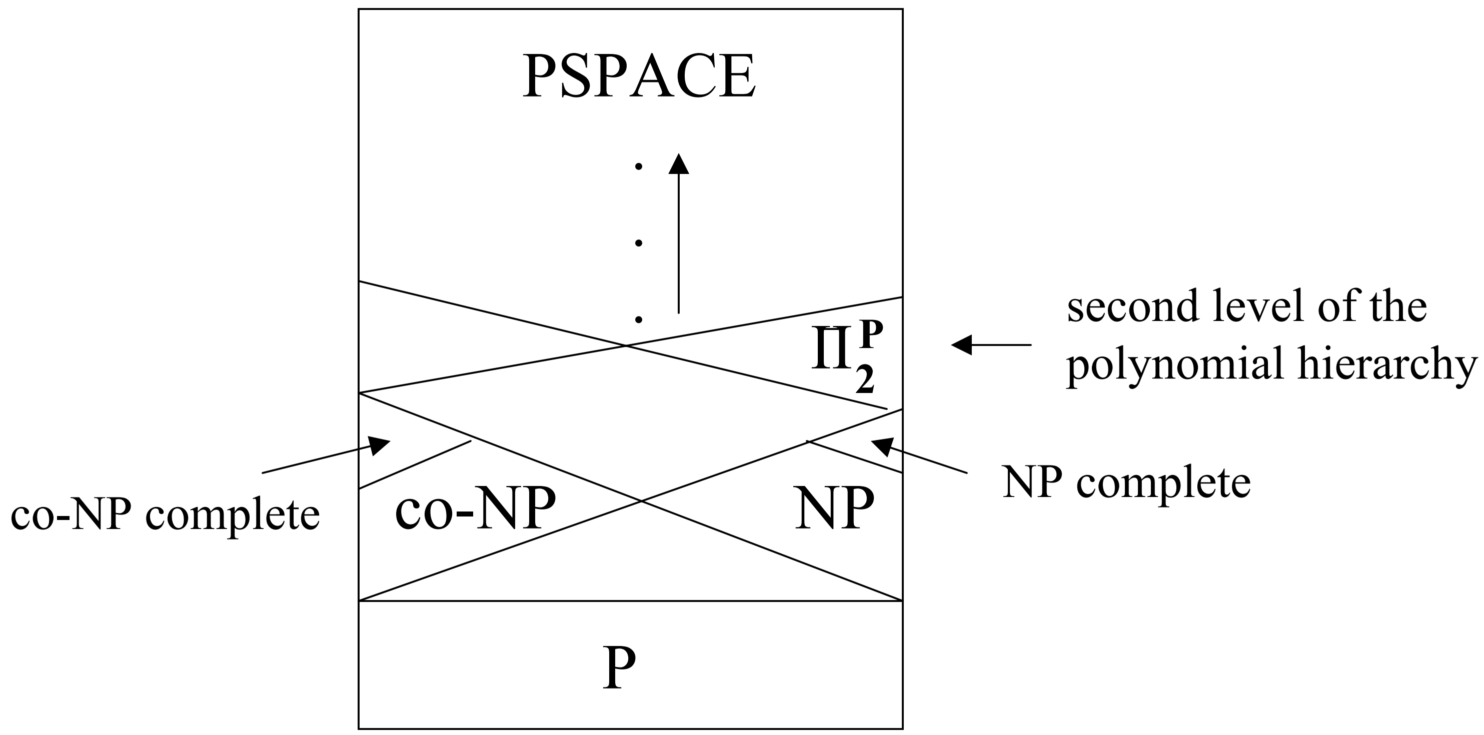
Complexity of CQA

When the first order query rewriting approach works (sound, complete and finitely terminating), consistent answers to FO queries can be obtained in PTIME (data complexity)

This applies to the T operator, its finite extensions, some graph theoretic techniques for CQA, and other specific techniques

(Chomicki, Marcinkowski; Inf. Comp. 2005): In those cases where CQA can be done in PTIME, the problem of repair checking can be solved in PTIME

Repair checking is also PTIME for arbitrary FDs and acyclic inclusion dependencies (deletions only)


PSPACE
.
.
.
Π_2^P
second level of the polynomial hierarchy
co-NP complete
co-NP
NP
NP complete
P

However: (deletions only)

- For arbitrary FDs and inclusion dependencies, repair checking becomes coNP-complete
- For arbitrary FDs and inclusion dependencies, CQA, i.e. deciding if a tuple is CA, becomes Π_2^P-complete

More complexity results: (Cali, Lembo, Rosati; PODS 03)

- For arbitrary FDs and inclusion dependencies (in particular, referential ICs), CQA becomes undecidable
  - Inclusion dependencies repaired through insertions
  - Cycles in the set of inclusion dependencies
  - Infinite underlying domain that can be used for insertions

Remarks:

- Complexity of query evaluation from DLPs under skeptical stable model semantics coincides with the complexity of CQA (Π_2^P-complete in data complexity)
- From this point of view the problem of CQA is not being overkilled by the use of the DLP approach
- However, it is known that for wide but restricted classes of queries and ICs, CQA has lower complexity, e.g. in polynomial time in data complexity
  (Chomicki, Marcinkowski; Inf. Comp., to appear)
  (Fuxman, Miller; ICDT 05)
- It becomes relevant to identify classes of ICs and queries for which the DLP can be "simplified" into a FO query, a lower complexity program, or can be evaluated with the well-founded semantics

Important Application: Virtual Data Integration

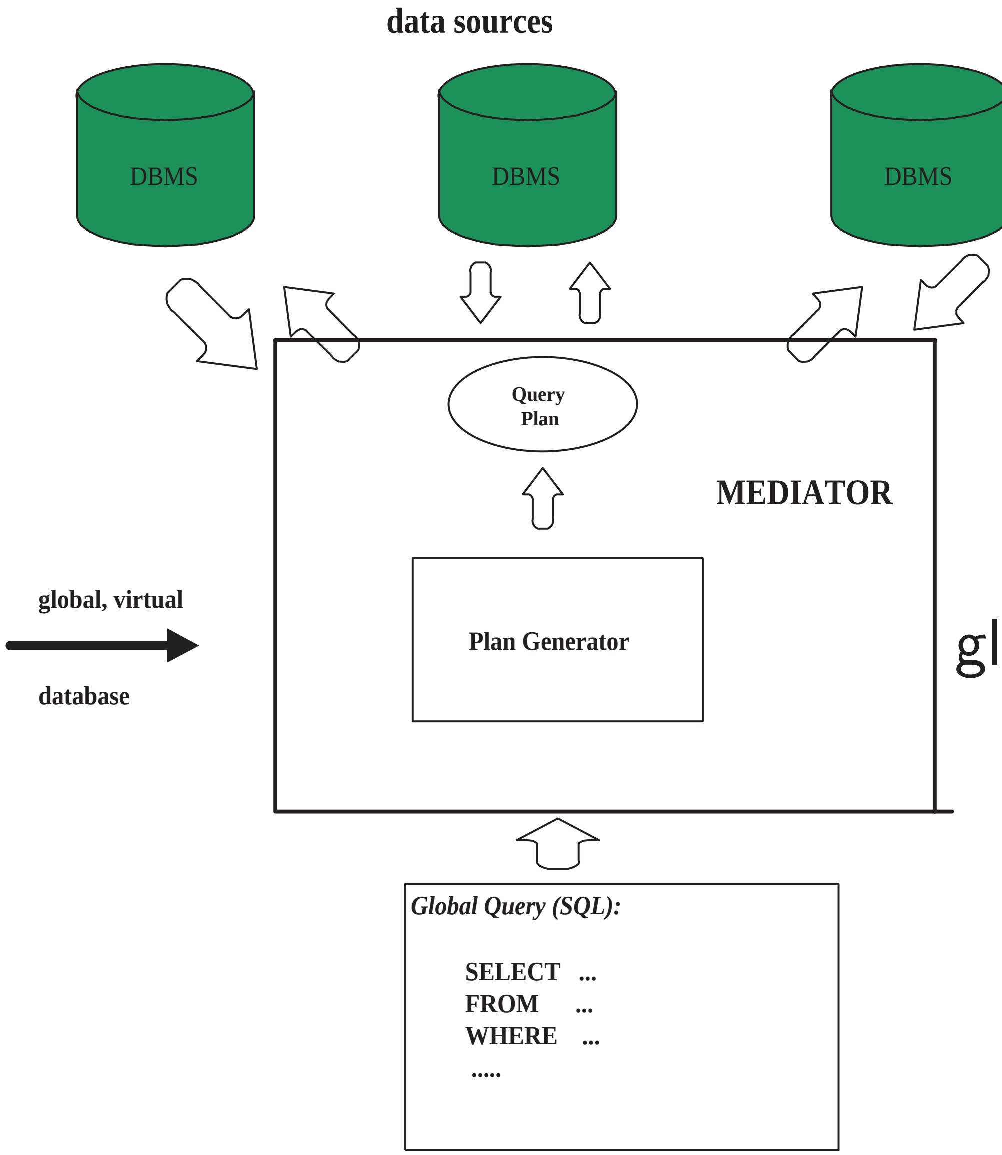
Mediator-based virtual data integration system $\mathcal{G}$, integrating a collection of material data sources $S_1, \ldots, S_n$

Each data source has a local schema and is assumed to be consistent wrt local ICs

$\mathcal{G}$ offers a database-like interaction schema, but data remains in the sources

Queries can be posed to $\mathcal{G}$: Given a (global) query Q to $\mathcal{G}$, a "query plan" is generated that extracts and combines information from the sources

Usually one assumes that certain ICs hold at the global level, and they are used in the generation of the query plan


data sources
DBMS
DBMS
DBMS
Query Plan
MEDIATOR
global, virtual
database
Plan Generator
Global Query (SQL):
SELECT ...
FROM ...
WHERE ...
.....

global ICs??

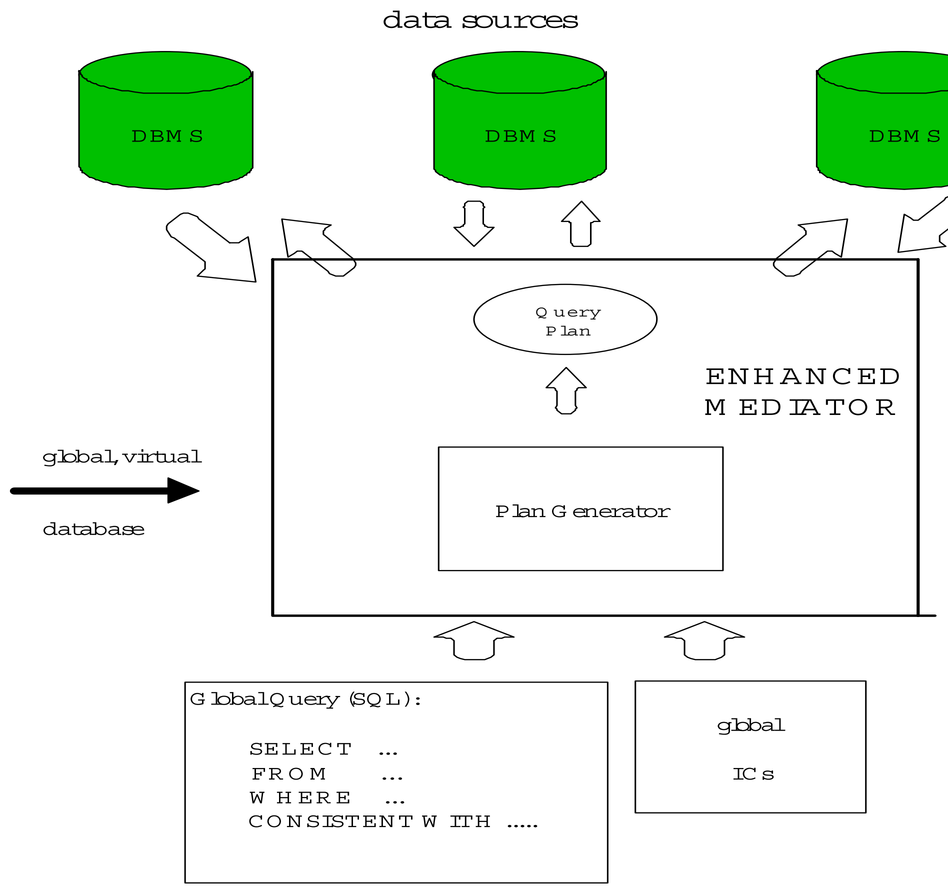
BUT, how can we be sure that such global ICs hold?

They are not maintained at the global level

A natural scenario for applying CQA: retrieve only information from the global system that is consistent with global ICs

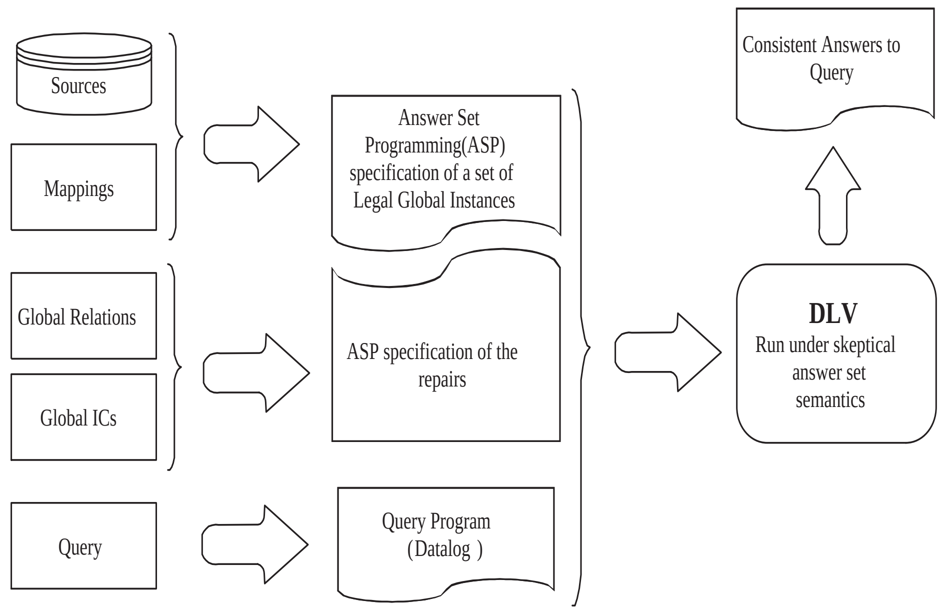
New issues appear:

- What is a repair of the global but virtual database?
- What is a consistent answer to a global query?
- How to retrieve consistent information from the global, virtual DB $\mathcal{G}$? At query time ...


data sources
DBMS
DBMS
DBMS
Query Plan
ENHANCED MEDIATOR
global, virtual
database
Plan Generator
Global Query (SQL):
SELECT ...
FROM ...
WHERE ...
CONSISTENT WITH .....
global
ICs

Work in this direction:

- CQA for monotone queries under “local-as-view” and open sources (Bravo, Bertossi; IJCAI 03), (Bravo, Bertossi; J. Appl. Logic, to appear)
- Extension to open, closed and clopen sources (Bertossi, Bravo; LNCS 3300)


Sources
Mappings
Global Relations
Global ICs
Query
Answer Set Programming(ASP) specification of a set of Legal Global Instances
ASP specification of the repairs
Query Program (Datalog)
DLV
Run under skeptical answer set semantics
Consistent Answers to Query

Other approaches:

- Repairs and CQA under different semantics under "global-as-view" and open sources
  (Lembo, Lenzerini, Rosati; KRDB 02)
  (Cali, Lembo,Rosati; IJCAI 03)

Discussion

The area of CQA in databases is an active area of research now

Many advances have been achieved since 1999

Many open problems are still open or are subject of ongoing research

- Several implementation issues, in particular in the case of most common SQL queries and constraints

  Specially for ICs that are not maintained by commercial DBMSs

- Research on many issues related to the evaluation of logic programs for consistent query answering (CQA) in the context of databases

- Existing implementations of stable models semantics are based on grounding the rules

  In database applications, this may lead to huge ground programs

- Implementations are geared to computing (some) stable model(s) and answering ground queries

  For database applications, posing and answering open queries is more natural

- Computing all the the stable models completely is undesirable

  Better try generation of “partial” repairs, relative to the ICs that are “relevant” to the query at hand

- Query evaluation based on skeptical stable model semantics should be guided by the query and its relevant information in the database
  (Eiter, Fink, G.Greco, Lembo; ICLP 03)
- Magic sets (or similar query-directed methodologies) for evaluating logic programs for CQA
  (S.Greco et. al)
- Optimization of the access to the DB, to the relevant portions of it
- Efficient integration of relational databases and answer set programming environments