Carleton
UNIVERSITY

Consistent Query Answering in Databases: Recent Developments

Leopoldo Bertossi

Carleton University

School of Computer Science

Ottawa, Canada

bertossi@scs.carleton.ca

www.scs.carleton.ca/~bertossi

Universidad de Concepción, November 2006

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: $Name \rightarrow Salary$ (a key dependency)

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: (continued) 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

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

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

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

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

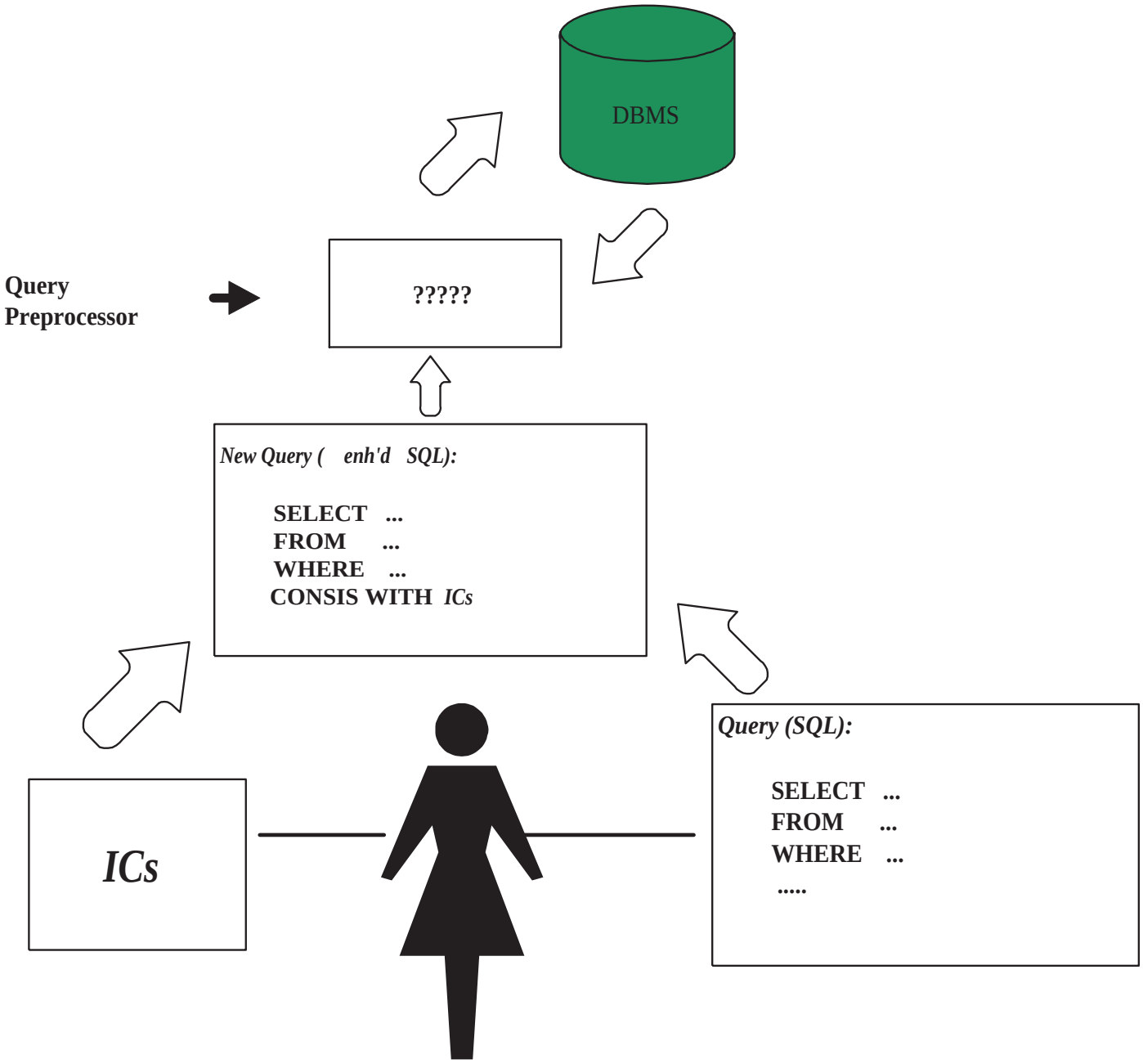
It has 2^n possible repairs!

Query Transformation

First-Order queries (then expressible in SQL) and constraints

Idea developed in (Arenas, Bertossi, Chomicki; Pods 99):

- Do not compute the repairs
- Query only the available inconsistent database instance
- Transform the query Q into a new query Q' by qualifying Q with appropriate information derived from the interaction between Q and the ICs
- Want consistent answers to $Q(\bar{x})$ in D?
  Then retrieve from D the (ordinary) answers to Q'
  Rewrite query: $Q(\bar{x}) \longmapsto Q'(\bar{x})$


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

Example: (continued) The functional dependency

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

Query: $Q(X,Y)$: $Employee(X,Y)$?

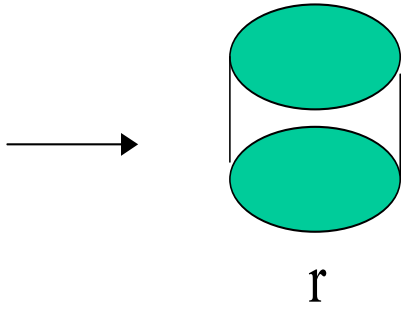
Consistent answers: $(V.Smith,\ 3{,}000)$, $(M.Stowe,\ 7{,}000)$
(but not $(J.Page,\ 5{,}000)$, $(J.Page,\ 8{,}000)$)

Can be obtained by means of the transformed query

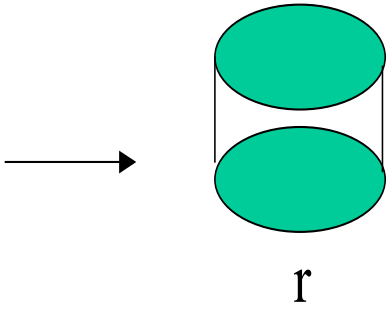
$Q'(X,Y)$: $Employee(X,Y)\ \wedge\ \forall Z\ (\neg Employee(X,Z)\ \vee\ Y = Z)$

... 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)
```



Some Limitations

First-order query rewriting based approaches to CQA provably have intrinsic limitations

Notice that FO query rewritability implies that CQA can be solved in polynomial time in data

The rewriting methodology presented before works for some special classes of queries and ICs

Sometimes the resulting query is also first-order (or SQL)

Methodology is applicable to ICs and queries without existential quantifiers (projections)

More precisely, for binary universal ICs (at most two database atoms plus built-ins) and queries that are quantifier-free conjunctions of literals

System for CQA: $Queca$ (Celle, Bertossi;2000)

Which excludes referential ICs, e.g.

$$\forall XY(R(X,Y) \rightarrow \exists Z\, P(Y,Z))$$

And also disjunctive or existential queries, e.g.

$$\exists Y\, Employee(X,Y)$$

However, *some* existentially quantified conjunctive queries can be rewritten as FO queries to obtain consistent answers wrt key dependencies

$$Q(X): \exists Y\, Employee(X, Y) \;\mapsto\; Q'(X): \exists Y\, Employee(X, Y)$$

FO query reformulation has been recently extended using other methods

- 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)

  System for CQA: *Hippo*

- Specific methods for conjunctive queries containing restricted projections, i.e. existential quantifiers, and key dependencies (Fuxman, Miller)

  System for CQA: *ConQuer*

FO rewriting provides polynomial time data complexity for CQA

If CQA has a higher intrinsic complexity, FO rewriting is bound to be incomplete ...

We will see that in the general case of FO ICs and queries, rewriting based approaches to CQA must appeal to languages that are more expressive than FO logic

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: $Spec \stackrel{?}{\models} Q(\bar{x})$
  - 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

Annotation	Atom	The tuple $P(\bar{a})$ is...
$\mathbf{t_a}$	$P(\bar{a}, \mathbf{t_a})$	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
$\mathbf{t}^{\star\star}$	$P(\bar{a}, \mathbf{t}^{\star\star})$	it is true in the repair

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

$IC: \quad \forall x\ (S(x) \rightarrow Q(x))$

The program $\Pi(D, IC)$ contains the following rules and facts:

- Facts:

  $S(a).\ \ S(b).\ \ Q(b).$

- Annotation clauses (similarly for Q):

  $S\_(x, \mathbf{t}^\star) \leftarrow S(x).$
  $S\_(x, \mathbf{t}^\star) \leftarrow S\_(x, \mathbf{t_a}).$

- Rules for repairing ICs:

$$S\_(x, \mathbf{f_a}) \vee Q\_(x, \mathbf{t_a}) \leftarrow S\_(x, \mathbf{t}^\star), \ \ not\ Q(x), x \neq null.$$
$$S\_(x, \mathbf{f_a}) \vee Q\_(x, \mathbf{t_a}) \leftarrow S\_(x, \mathbf{t}^\star), Q\_(x, \mathbf{f_a}), x \neq null.$$

- Repair interpretation rules (similarly for Q):

$$S\_(x, \mathbf{t}^{\star\star}) \leftarrow S\_(x, \mathbf{t}^\star), \ \ not\ S\_(x, \mathbf{f_a}).$$

- Program constraints (coherence of models):

$$\leftarrow S\_(x, \mathbf{t_a}), S\_(x, \mathbf{f_a}).$$
$$\leftarrow Q\_(x, \mathbf{t_a}), Q\_(x, \mathbf{f_a}).$$

One has to be careful when

- Dealing with databases containing null values, and
- Using null values to repair referential ICs

These issues had not been considered in the original repair semantics of 1999

With null values, new semantics have to be developed for IC satisfaction, database repairs, and query answering

(Bravo, Bertossi; IIDB 06)

In this example they are not an issue

Repairs are captured as the stable models of the program $\Pi(D, IC)$

In the example, the stable models are:

$$\begin{aligned}\mathcal{M}_1 \quad = \quad & \{S(a), S(b), Q(b), S\_(a, \mathbf{t}^\star), S\_(b, \mathbf{t}^\star), Q\_(b, \mathbf{t}^\star), Q\_(a, \mathbf{t_a}), S\_(a, \mathbf{t}^{\star\star}), \\ & S\_(b, \mathbf{t}^{\star\star}), Q\_(b, \mathbf{t}^{\star\star}), Q\_(a, \mathbf{t}^\star), Q\_(a, \mathbf{t}^{\star\star})\}\end{aligned}$$

$$\begin{aligned}\mathcal{M}_2 \quad = \quad & \{S(a), S(b), Q(b), S\_(a, \mathbf{t}^\star), S\_(b, \mathbf{t}^\star), Q\_(b, \mathbf{t}^\star), S\_(a, \mathbf{f_a}), S\_(b, \mathbf{t}^{\star\star}), \\ & Q\_(b, \mathbf{t}^{\star\star})\}\end{aligned}$$

The corresponding database repairs are, as expected:

$$D_1 \quad = \quad \{S(a), S(b), Q(b), Q(a)\}$$

$$D_2 \quad = \quad \{S(b), Q(b)\}$$

To obtain consistent answers to a query:

1. Transform the query Q into (or provide it as) a logic program $\Pi(Q)$ (a standard process)

2. Run the query program together with the repair 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) Query $S(x)$

It becomes the query program $\Pi(Q)$: $\quad Ans(x) \leftarrow S(x, \mathbf{t}^{\star\star})$

The two previous stable models become extended with ground Ans atoms

The stable models of $\Pi(D, IC) \cup \Pi(Q)$ are:

$$\begin{aligned}\mathcal{M}_1 \quad = \quad & \{S(a), S(b), Q(b), S\_(a, \mathbf{t}^{\star}), S\_(b, \mathbf{t}^{\star}), Q\_(b, \mathbf{t}^{\star}), Q\_(a, \mathbf{t_a}), S\_(a, \mathbf{t}^{\star\star}), \\ & S\_(b, \mathbf{t}^{\star\star}), Q\_(b, \mathbf{t}^{\star\star}), Q\_(a, \mathbf{t}^{\star}), Q\_(a, \mathbf{t}^{\star\star}), Ans(a), Ans(b)\}\end{aligned}$$

$$\begin{aligned}\mathcal{M}_2 \quad = \quad & \{S(a), S(b), Q(b), S\_(a, \mathbf{t}^{\star}), S\_(b, \mathbf{t}^{\star}), Q\_(b, \mathbf{t}^{\star}), S\_(a, \mathbf{f_a}), S\_(b, \mathbf{t}^{\star\star}), \\ & Q\_(b, \mathbf{t}^{\star\star}), Ans(b)\}\end{aligned}$$

The consistent answer to Q is the tuple (b)

Another example: Consistent answers to the query
$S(\bar{x}) \wedge \neg Q(\bar{x})$?

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

$$Ans(\bar{x}) \leftarrow S(\bar{x}, \mathbf{t}^{\star\star}),\ not\ Q(\bar{x}, \mathbf{t}^{\star\star})$$

No Ans atom in the intersection of the two stable models; then no consistent answers

NB: The last two queries could have been handled with the FO rewriting technique presented at the vey beginning: the IC is binary universal, and the queries are projection free conjunctions of literals

Disjunctive logic programs with stable model (or *answer set*) semantics:

- A "new" logic programming paradigm (*answer set programming*)
- A powerful knowledge representation formalism

Use of DLP is a general methodology that works for general FO queries, universal ICs and (acyclic) 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; IIDB 06)

The same repair program (and its stable models) can be used for all queries

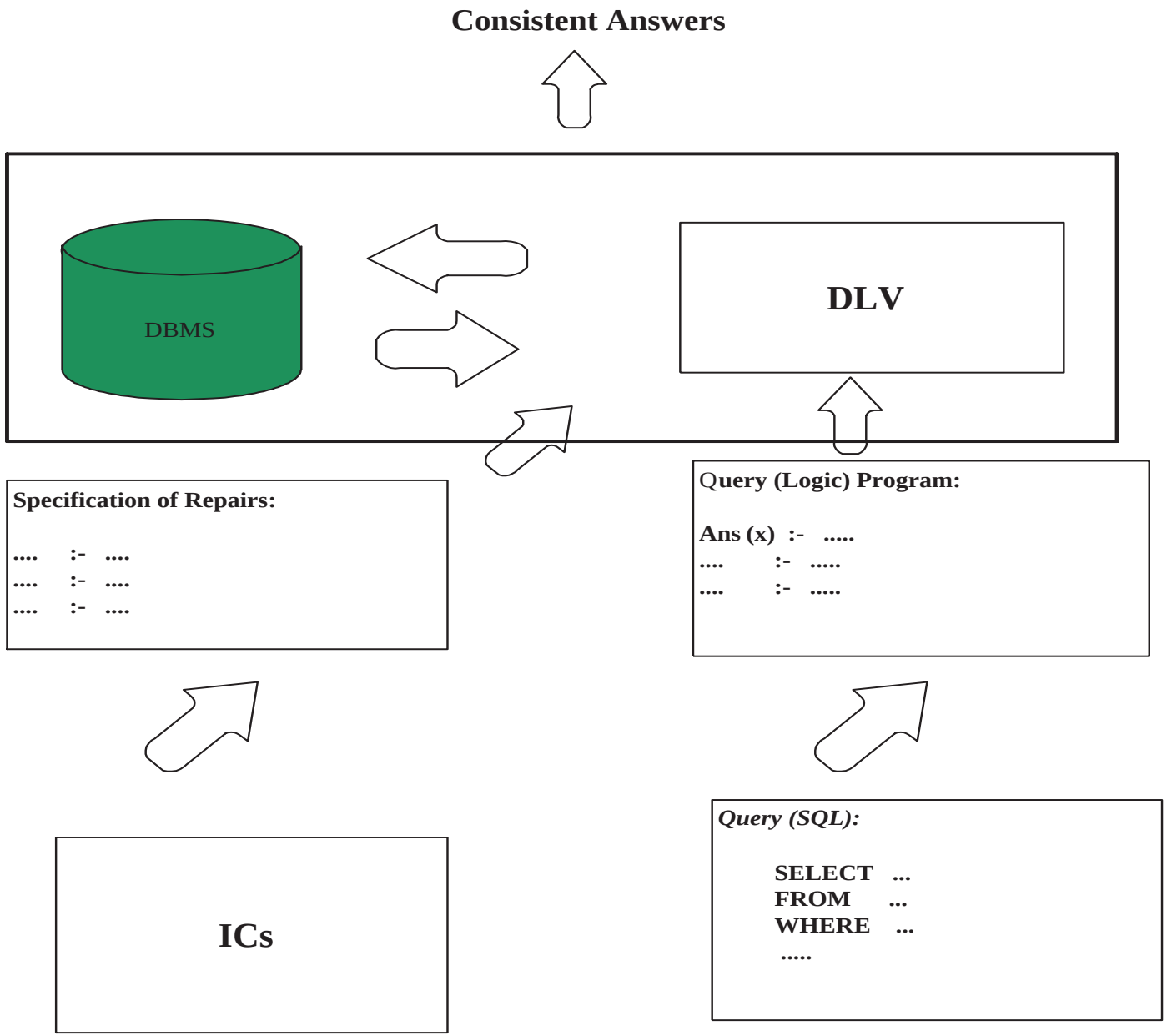
We have successfully experimented with the DLV system for computing the stable models semantics

(N. Leone et al.; ACM Transactions on Comp. Logic 06)

System: *ConsEx* (for *Cons*istency *Ex*tractor) (Caniupan,Bertossi;2006)

It uses "magic sets" techniques recently developed for logic programs with stable model semantics

(Faber,Greco,Leone;05), (Greco;03), (Cumbo,Faber,Greco,Leone;04)


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

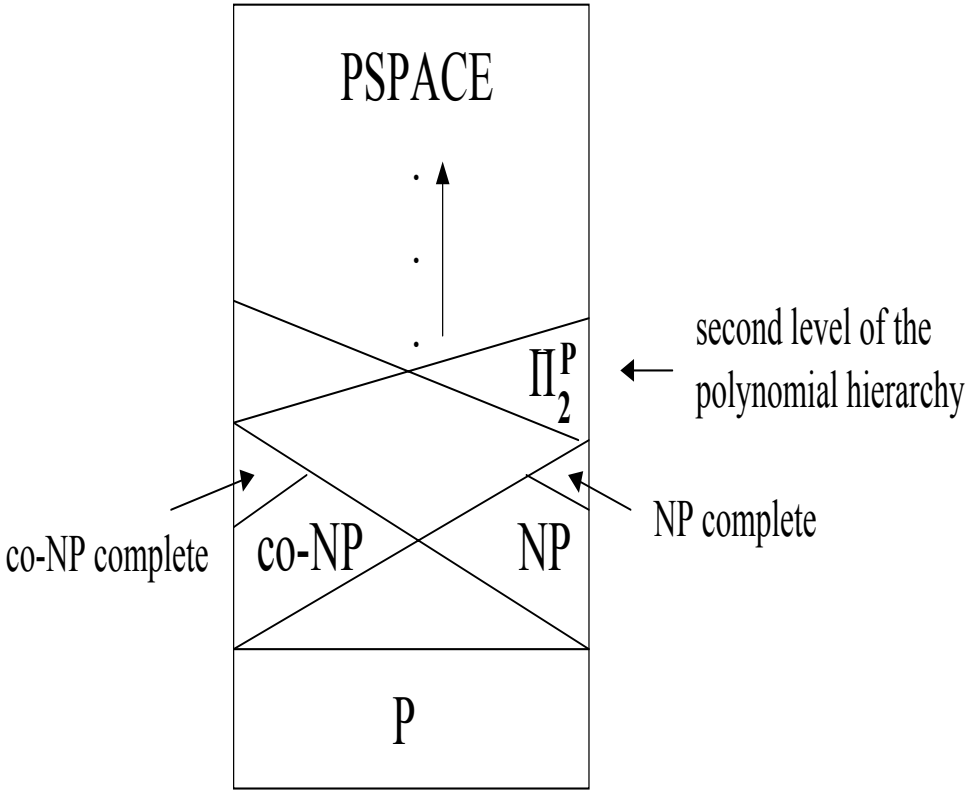
Data Complexity of CQA

When first order query rewriting works, consistent answers to FO queries can be obtained in PTIME

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

However:

- For arbitrary FDs and inclusion dependencies, repair checking becomes $coNP$-complete (deletions only)
- There are conjunctive queries (with projection) such that CQA i.e. deciding if a tuple is a CA, wrt key dependencies is $coNP$-complete
- For arbitrary FDs and inclusion dependencies, CQA is Π_2^P-complete (deletions only)


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

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

$$\forall xy(R(x,y) \rightarrow \exists z P(y,z))$$

A good reason to repair referential ICs using null values

With this new repair semantics, CQA becomes decidable

(Bravo, Bertossi; IIDB 06)

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. polynomial time or $coNP$ in data
- It becomes relevant to identify classes of ICs and queries for which the DLP can be "simplified" into a FO query or a lower complexity program

Promising line of attack:

- "A New Perspective on Stable Models" by P. Ferraris, J. Lee and V. Lifschitz, 2006.
- A disjunctive logic program Π with stable model semantics is transformed into a classical second-order sentence φ
- The models of φ are the stable models of Π
- There are techniques for eliminating second order quantifiers

  Under certain conditions φ can be transformed into an equivalent first-order sentence ψ

Application Area: Virtual Data Integration

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

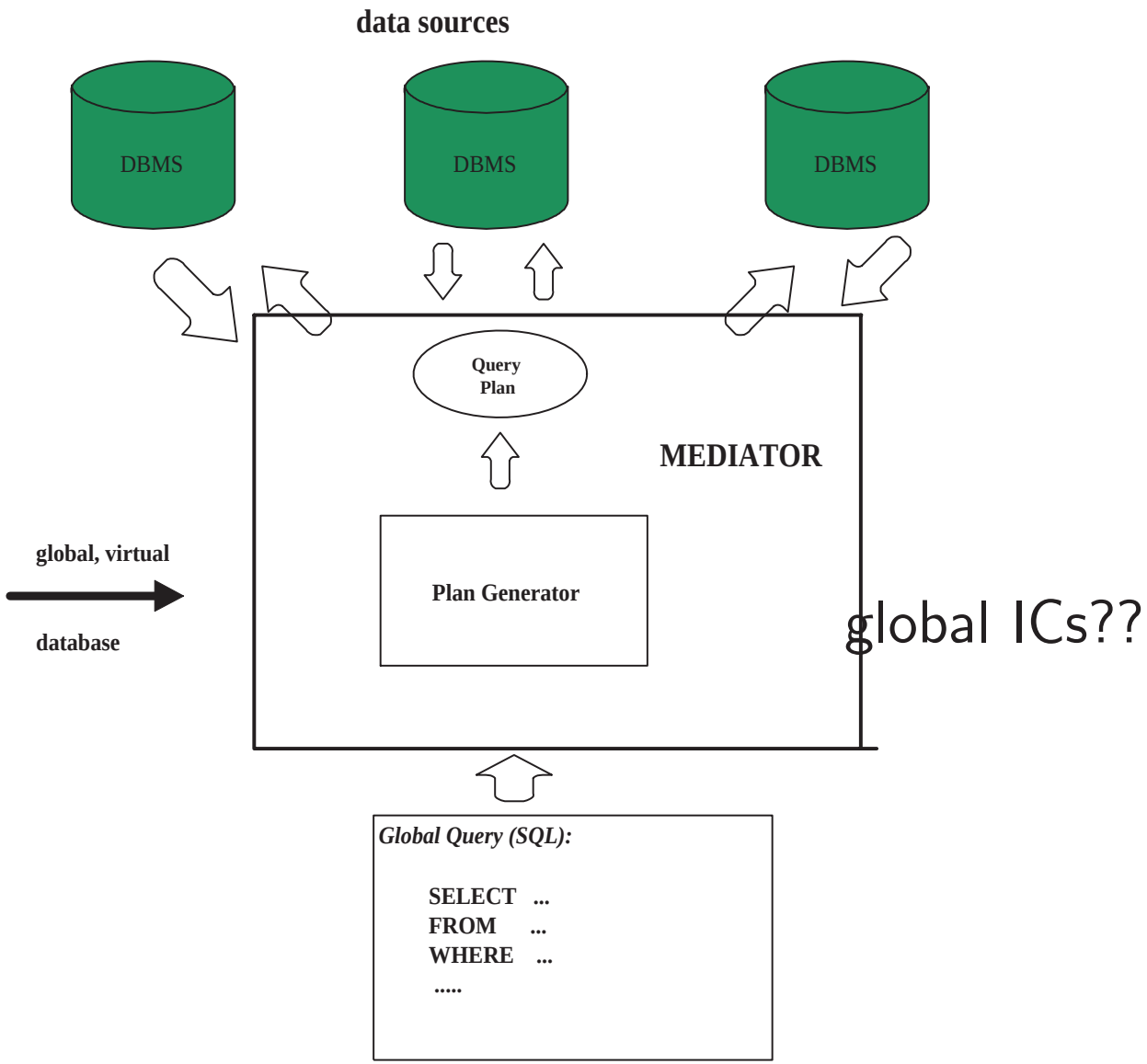
Each data source has a local schema and is assumed to be consistent wrt local ICs

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

Queries are posed to $\mathcal{G}$ in a language for the global schema

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
Plan Generator
global, virtual
database
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 the global ICs

Global ICs are used on-the-fly, when queries are answered

Here there is not possibility to clean data or restore consistency in material manner

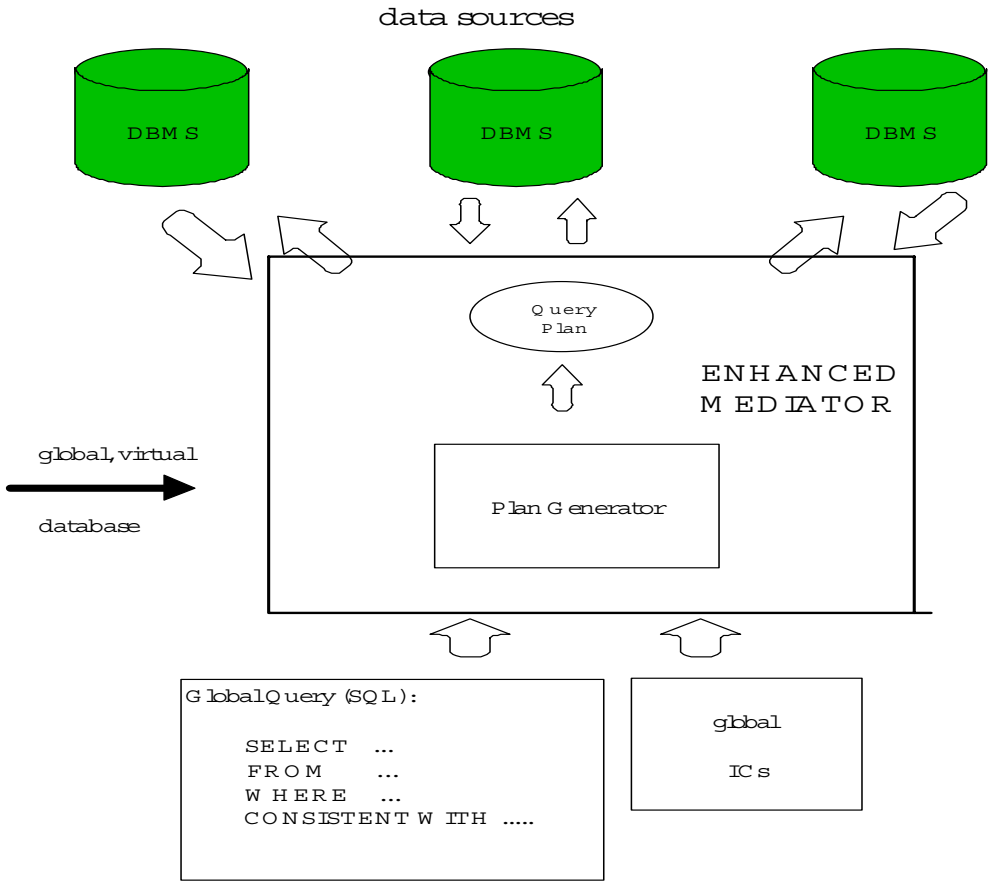
Things have to be done virtually, as in the spirit of CQA

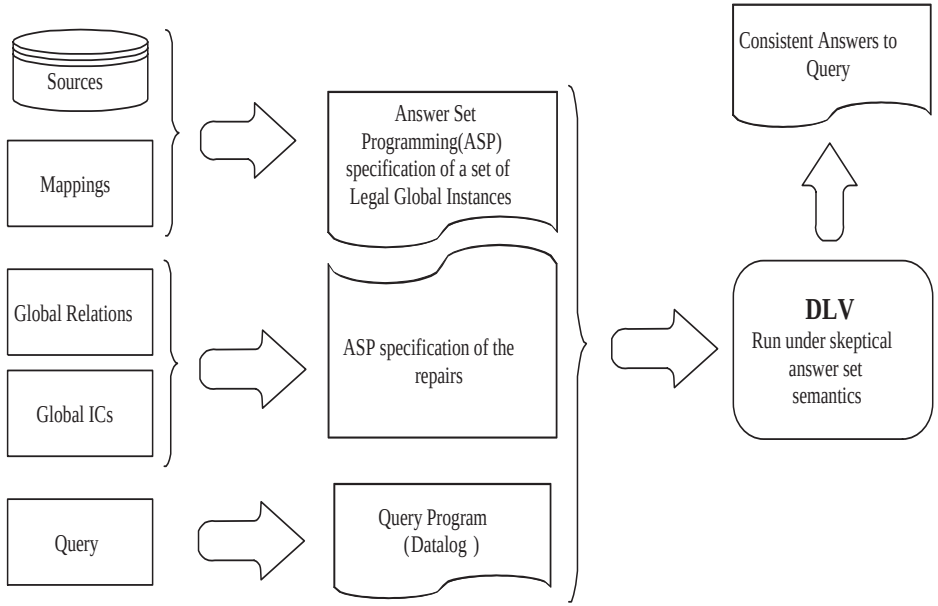
New issues appear:

- 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 data integration system $\mathcal{G}$? At query time ...

There has been work done recently in this direction

C.f. the survey (Bertossi, Bravo; LNCS 3300)


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


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

QUESTIONS?

www.scs.carleton.ca/~bertossi

www.scs.carleton.ca/~diis/