Carleton
UNIVERSITY

The Logics of Consistent Query Answers in Databases

Leopoldo Bertossi

Carleton University
School of Computer Science
Ottawa, Canada

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

Advanced course at ESSLLI 2005, Edimburgh, August 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

- Understanding the "logics" of consistent query answering in databases

  So as (usual) query answering in databases follows a certain logics (under the assumption of consistency):

  - Model- and proof-theoretic foundations
  - Based on first-order logic semantics
    (plus some non-monotonic assumptions)
  - Relational algebra and calculus
  - Compositional
  - SQL

C.f. - Abiteboul, Hull, Vianu. "Foundations of Databases", Addison-Wesley, 1995.
- R. Reiter. "A Logical Reconstruction of Databases", 1984.

Contents

0. Preliminaries
1. Basic Notions and Overview
2. A First Approach to CQA: FO Rewriting
3. Specifying Repairs in APC
4. Specifying Repairs with Logic Programs
5. Referential ICs and Incomplete Information
6. Aggregate Queries
7. Complexity of CQA
8. CQA in Virtual Data Integration
9. Query Answering in Peer-to-Peer Data Exchange

0. Preliminaries

Relational Databases

A relational schema is a set of relation names, each with a fixed finite arity; equivalently, it is a first-order signature containing only predicates

A relational database instance D is a structure compatible with a relational schema, such that

- It has a possibly infinite domain U
- The extensions of each of the relations in it is finite

Example: Relational schema $\mathcal{S} = \{R(\cdot,\cdot),\ S(\cdot,\cdot,\cdot)\}$

A possible instance $D = \langle U, R^D, S^D \rangle$, with

- $U = \{0, 1, 2, 3, \ldots\}$
- $R^D = \{(1, 2), (4, 3), (4, 7), (5, 8)\}$,
  $S^D = \{(1, 1, 1), (2, 3, 4), (5, 5, 6), (5, 4, 4)\}$

The relational schema $\mathcal{S}$ determines a first-order language $L(\mathcal{S})$ based on the relation names

This language is usually extended with built-in predicates, that have a fixed extension given by the logic, but may be infinite in extension, e.g. equality ($=$), inequality ($\neq$), $<$, ...

In the example, the extensions for these predicates are

- $=$: $\{(0, 0), (1, 1), (2, 2), \ldots\}$
- $\neq$: $\{(0, 1), (1, 0), (0, 2), (2, 0), \ldots\}$
- $<$: $\{(0, 1), (0, 2), (0, 3), \ldots\}$

A relational database (instance) D can be identified with a finite set of ground atoms of the form $R(\bar{t})$, where R is a relation in $\mathcal{S}$, and $\bar{t}$ is a finite sequence of constants, i.e. elements of the database domain U

Notice that we are extending the language $L(\mathcal{S})$ with elements from U

This is why sometimes the database domain is considered to be a part of the schema

In the example:
$D = \{R(1,2), R(4,3), R(4,7), R(5,8), S(1,1,1), S(2,3,4),$
$S(5,5,6), S(5,4,4)\}$

A ground atom $R(\bar{t})$ is also called a database tuple

In the (extended) language $L(\mathcal{S})$, integrity constraints are sentences φ that are expected to be satisfied by a database instance D, denoted $D \models \varphi$

They capture (part of) the semantics of data, in order to keep the correspondence between an outside reality and its model (the database)

A database is consistent wrt to a given set of integrity constraints IC if the sentences in IC are all true in D, i.e. $D \models IC$

Query Languages

Queries are formulas in (the extended) $L(\mathcal{S})$, usually with free variables

An answer to a query $Q(\bar{x})$ with free variables $\bar{x}$ is a tuple of constants $\bar{t}$ that makes Q true in D when the variables $\bar{x}$ are interpreted trough $\bar{t}$, denoted $D \models Q(\bar{t})$

FO logic, used as a query language, is essentially the relational calculus

However more expressive extensions can be considered

Datalog is a query language for relational databases

The query takes the form of a Datalog program; consisting of a set of logical formulas

Datalog is a natural extension of relational calculus (loosely speaking for the moment)

The language can be used to query relational databases that are extended with view definitions

... those view definitions are given in the Datalog query (program) itself

Datalog allows defining recursive views

Based on predicate logic (predicate calculus) and its semantics, plus recursion, and some particular evaluation methodologies

Except for recursion, syntax is more restricted that the one of predicate logic

Example:

$$
\begin{aligned}
Person(x) &\leftarrow Parent(x, y) \\
Person(y) &\leftarrow Parent(x, y) \\
Grandfather(x, z) &\leftarrow Parent(x, y), Parent(y, z) \\
Ancestor(x, y) &\leftarrow Parent(x, y) \\
Ancestor(x, z) &\leftarrow Parent(x, y), Ancestor(y, z)
\end{aligned}
$$

This could be a Datalog rule

$$S(x) \leftarrow P(a, x), Q(y, a)$$

where a is a constant (an element of the domain of the database), and x, y are variables, which are assumed to be universally quantified (implicitly)

E.g. the first clause is a simplified expression for

$$\forall x \forall y \ (Person(x) \leftarrow Parent(x, y))$$

If a universally quantified variable appears only in the body (e.g. y above), it is interpreted as existentially quantified in the body, i.e. it is logically equivalent to

$$Person(x) \leftarrow \exists y \ Parent(x, y)$$

In other words, the original rule

$$Person(x) \leftarrow Parent(x, y)$$

says that $Person$ is the projection of $Parent$ on x

Someone is person as long as *there exists someone* (no matter who) who is his/her child

A rule is ground if it does not have any variables

- $P(a) \leftarrow Q(c, a)$ is ground
- $P(x) \leftarrow R(a, x)$ is not ground

The body of a clause may be empty, e.g.

- $P(a) \leftarrow$ (written $P(a)$)
- $E(x, x) \leftarrow$ (written $E(x, x)$)

Ground rules with empty bodies are called facts

$Parent(juan, pablo) \leftarrow$ (or simply $Parent(juan, pablo)$)

Facts are usually in the underlying relational database, i.e. in the extensional database

Almost every RA operation can be expressed by means of a Datalog program; just a few examples ...

- Selection

  $\sigma_{X=a}(R(X,Y)) \quad \mapsto \quad Ans(Y) \leftarrow R(a,Y)$

- Intersection (conjunction)

  $R(X,Y) \cap S(X,Y) \quad \mapsto \quad Ans(X,Y) \leftarrow R(X,Y), S(X,Y)$

- Union (disjunction)

  $R(X,Y) \cup S(X,Y) \quad \mapsto \quad Ans(X,Y) \leftarrow R(X,Y)$
  $Ans(X,Y) \leftarrow S(X,Y)$

- Projection (existential quantification)

  $\Pi_X(R(X,Y)) \quad \mapsto \quad Ans(X) \leftarrow R(X,Y)$

- Join

  $R(X,Y) \bowtie_{Y=Z} S(Z,V) \quad \mapsto \quad Ans(X,Y,V) \leftarrow R(X,Y), S(Y,V)$

- Difference

  $R(X,Y) \setminus S(X,Y) \quad \mapsto \quad ?????$

  No negation in Datalog!

- But we do have recursion in Datalog!

  And we can do things that are (provably) impossible in RA, like defining the transitive closure of a relation

If we extend Datalog with negation we will have RA and more ...

Extensions of Datalog:

- Built-in predicates: $=, \neq, <, +, \ldots$

  They can be used in conditions in the bodies, e.g.

  $$SeniorParent(x) \leftarrow Parent(x, y), Age(x, z), z > 65$$

- Disjunction, e.g. $P(x, y) \vee T(x, y) \leftarrow R(x, y), S(x, y)$

  Disjunctive Datalog used to represent uncertain information

  A disjunctive Datalog program may have several intended intentional databases!

- Aggregate functions (so as for RA): `SUM`, `MAX`, `AVG`, ...

$$Ans(X, sum(Y)) \leftarrow R(X, Y)$$

Ans returns for every value a for X the sum of all values b for Y such that $(a, b) \in R$

R	X	Y
	a	2
	a	4
	b	7
	b	2

$$Ans = \{(a, 6), (b, 9)\}$$

- Negation, e.g. $P(x, y) \leftarrow R(x, y),\ not\ S(x, y)$

  With it we recover the "set difference" of RA we were missing

  $R(X, Y) \setminus S(X, Y) \quad \mapsto$
  $Ans(X, Y) \leftarrow R(X, Y),\ not\ S(X, Y)$

  What is the intended meaning (semantics) of negation in Datalog programs?

  In a Datalog program with negation, what is the intended intentional database instance?

Semantics of negation is an extension of the Closed World Assumption in relational databases:

- If a positive fact does not appear in the DB, then it is assumed to be false

This is a weak negation, as opposed to the strong, classical negation

The combination of negation and recursion can be problematic

Example: $EDB = \{P(a,b), P(a,a), P(c,b), Q(a,b), Q(c,c), S(a,a), T(a,b)\}$

Intensional database using view definitions in Datalog^{not} :

$R(x,y) \leftarrow P(x,y),\ not\ Q(x,y)$
$R(x,y) \leftarrow T(x,y), R(x,z),\ not\ S(x,z)$

Answer query $Ans \leftarrow R(x,y)$ by bottom-up evaluation (notice both negation and recursion)

1. First propagate values into R using the first rule

   That rule is used once (only extensional tables in body)

2. Use second rule with partial contents for R in body

3. Keep iterating with the second rule until nothing new ...

Here negation and recursion do not interact; negation is applied to predicates that have been completely computed at a previous stage; the program is stratified

Example: Recursion via negation

EDB $Q = \{1, 2\}$, extended with view

$$P(x) \leftarrow \quad Q(x),\ not\ P(x)$$

Bottom-up computation of P: steps ...

1. $P = \emptyset$
2. $P = \{1, 2\}$
3. $P = \emptyset$
4. etc., etc. (infinite computation)

P is defined by recursion via negation; the program is not stratified

What is the semantics of negation here?

Talking about semantics ...

What is the intended semantics of a Datalog program?

What becomes true given a Datalog program and an underlying extensional database?

What is the intended contents of the intentional relations?

Same questions for extensions of Datalog ...

The semantics for Datalog and its extensions can be presented in a uniform manner in the context of logic programming

Semantics of Logic Programs

Definite Logic Programs (Datalog Programs)

Consist of positive Horn clauses, no negation

Example: Program P

$$\begin{aligned} path(x,z) &\leftarrow arc(x,y),\ path(y,z) \\ path(x,x) &\leftarrow \\ arc(b,c) &\leftarrow \end{aligned}$$

What is the semantics of P? What world is P describing?
Is there an intended model for P?

Since variables are universally quantified (universal formulas),
we can concentrate on the Herbrand models of P

Intuitively, P talks about the (syntactic) objects mentioned by (or can be constructed from) P (like in RDBs), then no need to consider models with other bizarre domains

A Herbrand structure for P:

- Has the Herbrand Universe H_P of P as its domain, in this case $H_P = \{b, c\}$
- In terms of interpretation of predicates, can be identified with a subset of the Herbrand Base HB_P of P; here

$$HB_P = \{arc(b,b), arc(c,c), arc(b,c), arc(c,b), path(b,b), path(c,c), path(b,c), path(c,b)\}$$

  It contains all the possible ground atomic propositions that can be potentially true; and in particular structures, only some of them will be true (and the others, false)

A Herbrand model (H-model) for P is a Herbrand structure for P that makes true all the clauses in P

- A model of the program (a subset of HB_P):

  $M_1 = \{arc(b,b), arc(c,c), arc(b,c), arc(c,b), path(b,b),$
  $path(c,c), path(b,c), path(c,b)\}$

- Another model

  $M_2 = \{arc(b,c), path(b,b), path(c,c), path(b,c)\}$

- Yet another one

  $M_3 = \{arc(b,c), path(b,b), path(c,c), path(b,c), path(c,b)\}$

- A Herbrand structure for P that is not a model of P

  $M_4 = \{arc(b,c), path(c,c), path(c,b)\}$

Since H-structures are (identified with) subsets of HB_P, they can be compared by set inclusion ($\subseteq$); a partial order in the class of subsets of HB_P

M_2 is a minimal H-model: M_2 is an H-model of P and no proper subset of M_2 is an H-model of P

Theorem: For a definite program P

- P has exactly one minimal H-model $\underline{M}(P)$
- The semantics of P can be captured by $\underline{M}(P)$, i.e. for a ground atom A:

$$P \models A \iff A \in \underline{M}(P)$$

- The ground atoms $A \in \underline{M}(P)$ are exactly those that can be proved by resolution-based refutations from P

Example: Program P

$$
\begin{aligned}
r(x) &\leftarrow p(x) \\
p(x) &\leftarrow q(x,y) \\
q(a,a) &\leftarrow \\
q(a,b) &\leftarrow
\end{aligned}
$$

Ground instantiation P_H of P (on the H-universe):

$$
\begin{aligned}
r(a) &\leftarrow p(a) \\
r(b) &\leftarrow p(b) \\
p(a) &\leftarrow q(a,a) \\
p(a) &\leftarrow q(a,b) \\
p(b) &\leftarrow q(b,b) \\
p(b) &\leftarrow q(b,a) \\
q(a,a) &\leftarrow \\
q(a,b) &\leftarrow
\end{aligned}
$$

$\underline{M}(P)$ can be obtained bottom-up, by propagating the facts through the rules, from right to left, iteratively:

First step: $q(a,a),\ q(a,b) \in \underline{M}(P)$

Second step: $p(a) \in \underline{M}(P)$

Third step: $r(a) \in \underline{M}(P)$

$$\underline{M}(P) = \{q(a,a), q(a,b), p(a), r(a)\}$$

Given an extensional database, a Datalog program defined on it defines exactly one minimal intentional database!

Disjunctive Positive Programs

We now consider more expressive FO logic programs containing (non-Horn) clauses of the form

$$A_1 \vee \cdots \vee A_n \quad \leftarrow \quad B_1, \ldots, B_m$$

where the A_i, B_j are atoms

Example:

$$\begin{aligned} P(x) \vee Q(x) &\leftarrow R(x), S(x,y). \\ R(x) &\leftarrow U(x), V(x). \\ S(x,y) &\leftarrow U(x), V(y). \\ U(a). \quad U(c). \; V(b). &\quad V(a). \end{aligned}$$

Herbrand Universe: $H_P = \{a, b, c\}$
Herbrand Base: $HB_P =$ all possible ground atoms

Now, two minimal (Herbrand) models:

$$M_1 = \{U(a), U(c), V(b), V(a), S(a,a), S(a,b), S(c,a), S(c,b), R(a), P(a)\}$$

$$M_2 = \{U(a), U(c), V(b), V(a), S(a,a), S(a,b), S(c,a), S(c,b), R(a), Q(a)\}$$

Disjunctive programs may have more than one minimal model (Minker et al.)

Cautious or skeptical semantics: What is true in all minimal models

Brave or possible semantics: What is true in some minimal model

Normal Programs

Now programs with one atom in the head, weak negation (not) in the body

Rules may be of the form:

$$A \leftarrow A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n$$

where the A, A_i are atoms

We give a declarative semantics to normal programs, i.e. given through a collection of intended models

It is the stable model semantics (or more generally, answer set semantics) (Gelfond & Lifschitz, 1988)

Can be applied to any program with negation

Let P be a normal program, and $S \subseteq HB_P$, i.e. a subset of the Herbrand Base

S is a candidate to be a (stable) model of P; a "guess" that will be accepted if properly supported by P

S can be seen as a set of assumptions

S will be a stable model of P if it can be justified on the basis P; more precisely, if assuming S, we can recover S via P

Do the following:

- Pass from P to P_H, the ground instantiation of P

- Construct a new ground program P_H^S, depending on S as follows:

  1. Delete from P_H every rule that has a subgoal $not\ A$ in the body, with $A \in S$

     Intuitively: We are assuming A to be true, then $not\ A$ is false, then the whole body is false, and nothing can be concluded with that rule, it is useless

  2. From the remaining rules, delete all the negative subgoals

     Intuitively: Those rules are left because the negative subgoals are true, and since they are true, we can eliminate them as conditions in the bodies (because they hold)

- We are left with a ground definite program P_H^S (no negation)
- Compute $\underline{M}(P_H^S)$, the minimal model of the definite program
- If $\underline{M}(P_H^S) = S$, we say that S is a stable model of P

Intuitively, we started with S (as an assumption) and we recovered it, it was stable wrt to the P-guided process described above; it is self-justified ...

Example: Program P $\quad p(a) \leftarrow \ not\ p(b)$ (already ground)

Consider $S = \{p(a)\}$

Here: $p(b) \notin S$, then $not\ p(b)$ is satisfied in S and can be eliminated from the body

We obtain P^S: $p(a) \leftarrow$, a definite program

Its minimal model is $\{p(a)\}$, that is equal to S

S is a stable model of the original program

Notice that P is a non-stratified (there is recursion via negation), but has a stable model, actually only one in this case

Example: Program P (non-stratified)

$p(x) \leftarrow q(x,y),\ not\ p(y)$
$q(a,b).$

P_H: (ground instantiation) Candidate $S = \{p(b)\}$

$p(a) \leftarrow q(a,a),\ \underline{not\ p(a)}$

$p(a) \leftarrow q(a,b),\ not\ p(b)$ $\times$

$p(b) \leftarrow q(b,a),\ \underline{not\ p(a)}$

$p(b) \leftarrow q(b,b),\ not\ p(b)$ $\times$

$q(a,b).$

P_H^S :

$$\begin{aligned} p(a) &\leftarrow q(a,a) \\ p(b) &\leftarrow q(b,a) \\ q(a,b). & \end{aligned}$$

Minimal model of P_H^S is $\{q(a,b)\} \neq S$, then S is not a stable model

The program is non-stratified, but it has the stable model $S = \{q(a,b), p(a)\}$ (check!)

Example: Program P

$$\begin{aligned} male(a) &\leftarrow not\ female(a) \\ female(a) &\leftarrow not\ male(a) \end{aligned}$$

If $S_1 = \{male(a)\}$, then P^{S_1} is $\{male(a)\}$, and then S_1 is a stable model

$S_2 = \{female(a)\}$ is also a stable model of P

There may be more than one stable model for a program!

Again, P is non-stratified

Example : Program P

$$even(0).$$
$$even(x) \leftarrow not\ even(s(x))$$

P_H:

$$even(0).$$
$$even(0) \leftarrow not\ even(s(0))$$
$$even(s(0)) \leftarrow not\ even(s(s(0)))$$
$$\ldots \quad \ldots$$

$S = \{even(0), even(s(s(0))), even(s(s(s(s(0))))), \ldots\}$ is a stable model

P_H^S:

$$even(0).$$
$$even(0).$$
$$even(s(s(0))). \qquad \text{Etc.}$$

$\underline{M}(P_H^S) = \{even(0), even(s(s(0))), even(s(s(s(s(0))))), \ldots\}$

Some Results:

- Every stable model of P is a Herbrand model in the usual sense

  In them, not is interpreted as "not belonging to the model"

- A stable model is always a minimal model (i.e. no proper subset of it is a model of the program)

- A normal program may have several stable models; so several (stable) models determine the semantics of a program

- If there are several stable models for a program, it means that some atoms are left undetermined (those that are true in some of them, but false in others)

- Now we are giving a declarative semantics to a wider class of programs (with or without negation), even non-stratified ones
- The stable model semantics of a normal program?
  - Skeptical or cautious semantics: What is true of a program is what is true of all stable models of the program
  - Brave or possible semantics: What is true of a program is what is true of some stable model of the program
- If P is definite or normal, but stratified, then it has a unique stable model

This stable model coincides with the minimal model for definite programs, and the "natural" one for normal stratified programs

Then this semantics extends the ones we had for the "good" cases before

In particular, the unique stable model can be computed by means of an bottom-up iterative process

Example: Stratified normal program P

$$
\begin{aligned}
D(x) &\leftarrow Q(x),\ not\ R(x)\\
R(x) &\leftarrow S(x)\\
R(x) &\leftarrow H(x),\ not\ S(x)\\
S(x) &\leftarrow T(x,y),\ not\ U(y)\\
S(x) &\leftarrow U(x)\\
Q(a).\ \ Q(b).\ Q(c).\ \ H(b).\ \ T(a,b).
\end{aligned}
$$

In a stratified program there is natural hierarchy of predicates wrt their definitions; they can be placed in strata

Each predicate can be completely computed without calling via negation other predicates that are not in a lower stratum

Stratum 4 : D

$\uparrow$

Stratum 3 : R

$\uparrow$

Stratum 2 : S

$\uparrow$

Stratum 1 : Q, H, T, U

Compute their extensions upwards: (poly-time computation)

1. $M_1 = \{Q(a), Q(b), Q(c), H(b), T(a, b)\}$
2. $M_1 = \{S(a)\}$
3. $M_2 = \{R(a), R(b)\}$
4. $M_3 = \{D(c)\}$

Unique stable model

$M = M_0 \cup M_1 \cup M_2 \cup M_3$

(check that this is a stable model of the program)

- The problems of finding a stable model, determining whether one exists, checking skeptical or brave consequences from the program, etc. have all rather high complexity

- However, with not-stratified programs with stable model semantics we gain expressive power

  And we can solve problems that are intrinsically complex and require such an expressive power (and complexity of program evaluation)

- In particular, computational implementations for the stable model semantics are being used to find solutions to combinatorial optimization and decision problems

  NP-hard problems can be represented by logic programs and the stable models correspond to their solutions; thus in these applications finding one model is good enough

Disjunctive Normal Programs

Now we admit rules of the form

$$B_1 \vee \cdots \vee B_k \leftarrow A_1, \cdots, A_m, not\ A_{m+1}, \ldots, not\ A_n$$

with B_j, A_i are atoms and not is weak negation

E.g. $P(x) \vee T(x) \leftarrow R(x), Q(y), not\ S(x), not\ Q(x)$

We have a stable model semantics (Gelfond & Lifschitz, 1991)

Models are sets of ground atoms; and so are candidates S to be answer sets, i.e. of the form $S = \{p, f, k, l, d\}$

The test for S is similar:

- Fully instantiate the program
- Pruning process as for ELPs
- Now we get a ground disjunctive program without not, i.e. with clauses of the form

$$p \vee q \leftarrow s, t, u, v, w$$

- The resulting disjunctive program may have several minimal models
- S is a stable model if it coincides with one of the minimal models of the program in the previous step

A useful extension: program constraints of the form

$$\leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_k,$$

with L_i atoms, can be added to a program P

Intuitively: It is not the case that the body becomes true (for any values for the variables)

Has the effect of filtering the stable models of P where the body becomes true

Example: Program P

$$\begin{aligned} ug(x) &\leftarrow not\ grad(x), stud(x) \\ grad(x) &\leftarrow not\ ug(x), stud(x) \\ stud(mary) &\leftarrow \\ &\leftarrow ug(x) \end{aligned}$$

Without the program denial, two stable models:

$\{stud(mary), ug(mary)\}$ and $\{stud(mary), grad(mary)\}$;

with the denial, only the second one

1. Basic Notions and Overview

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

Example

$D = \{P(a), P(b), Q(a), R(a)\}$

$IC = \{\forall x(P(x) \rightarrow Q(x)), \forall x(Q(x) \rightarrow R(x))\}$

Two repairs for D:

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

They are minimal repairs, i.e. there is no consistent D_3 with:

$$D \Delta D_3 \subsetneq D \Delta D_1 \quad \text{or} \quad D \Delta D_3 \subsetneq D \Delta D_2$$

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

But not $D_3 = \{P(a, b), Q(a, b)\}$, because

$$D \Delta D_3 = \{Q(a, b), Q(c, b)\} \supsetneq D \Delta D_2$$

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 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!

Attacking the Problem (Overview)

Different alternatives for computing consistent answers

1. (Arenas, Bertossi, Chomicki; PODS 1999),
   (Celle, Bertossi; DOOD 2000)

   Transform the query into a new query

   Do not compute the repairs

   Pose the new query to the only available (inconsistent) database instance (as usual)

2. Represent in a compact way the collection of all database repairs and get information from the representation

   2.1. (Arenas, Bertossi, Kifer; DOOD 2000)

   - Repairs are some minimal models of a theory written in annotated predicate logic

   2.2. (Arenas, Bertossi, Chomicki; TPLP 2003),
   (Barcelo, Bertossi; PADL 03),
   (S.Greco et al.; TKDE 2003)

   - Repairs are stable models of a logic program
   - Repairs specified by means of a logic program
   - To obtain consistent answers, run the program

2.3. (Arenas, Bertossi, Chomicki; ICDT 2001)

- Repairs are maximal independent sets in a graph whose nodes are the DB tuples
- Arcs are drawn between two tuples participating in the violation of an FD

2.4. (Bertossi, Schwind; AMAI 2004)

- Repairs are branches in a tableau
- Tableaux are developed and their branches closed taking into account ICs

Related Work

We have used a particular notion of database repair:

- Basically no restriction on them
  (only minimality based on inclusion of sets of tuples)
- No assumption about the DB
- (Rest of this presentation refers to this notion)

However

There may be different assumptions about the DB
(in the presence of inclusion dependencies):

- DB is possibly incorrect but complete:

  Repair by deletion only
  (Chomicki, Marcinkowski; Inf. and Comp. 2005)

- DB is possibly incorrect and incomplete:

  Fix FDs by deletion, referential ICs by insertion
  (Cali, Lembo, Rosati; PODS 2003)

- Referential ICs are repaired using null values that do not propagate through other ICs
  (Barcelo, Bertossi, Bravo; LNCS 2582, 2003)

Different notions of minimal repairs:

- Minimal cardinality set of changes (c.f. page 14)
  (Arenas, Bertossi, Chomicki; TPLP 2003)

- Minimal cardinality set of updates, i.e. changes of attribute values (and not whole tuples) (c.f. page 15)
  (Wijsen; ICDT 2003)
  (Franconi, Laureti, Leone, Perri, Scarcello; LPAR 2001)
  (Bertossi, Bravo, Lopatenko, Franconi; LAAIC 2005)
  (Bohannon, Flaster,Fan, Rastogi; SIGMOD 2005)

  Idea: In example on page 61, maybe there was a mistake, and the value c in the first attribute of Q should be changed to a

2. A First Approach to CQA: FO Rewriting

Query Transformation

First-Order queries and constraints

Approach: Transform the query and keep the database instance!

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

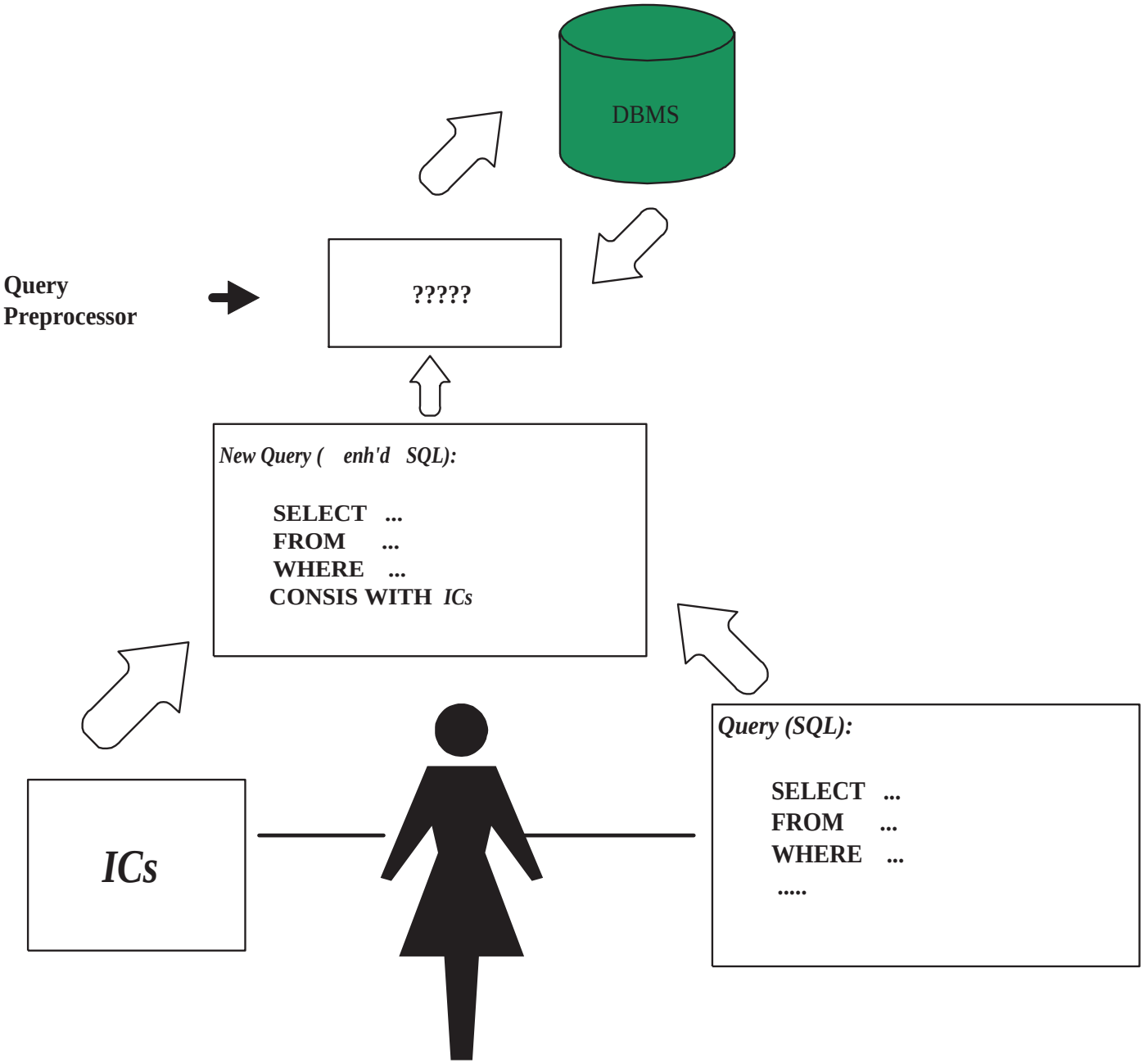
- To locally satisfy the ICs
- To discriminate between tuples in the answer set
- Inspired by "Semantic Query Optimization" techniques

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 $Q'(\bar{x})$


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

Example

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

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: $Employee(X,Y)$?

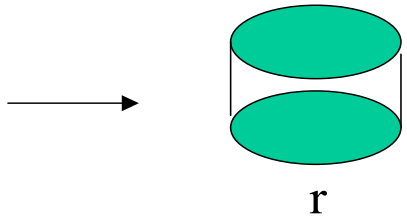
Consistent answers: $(V.Smith,\ 3{,}000),\ (M.Stowe,\ 7{,}000)$
(but not $(J.Page,\ 5{,}000),\ (J.Page,\ 8{,}000)$)

Can be obtained by means of the transformed query

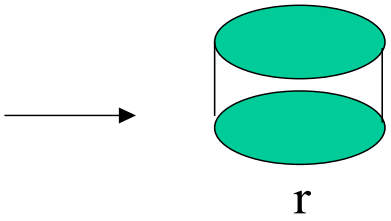
$$T(Q(X,Y)) := Employee(X,Y)\ \wedge \forall Z\ (\neg Employee(X,Z)\ \vee\ Y = Z)$$

... 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

Relations: $Supply(x,y,z)$: "x supplies item z to y"

$Class(z,w)$: "item z belongs to class w"

IC: C is the only supplier of items of class K

$$\forall x, y, z(Supply(x, y, z) \wedge Class(z, K) \rightarrow x = C)$$

An instance D that violates IC

$Supply$		
C	D_1	I_1
D	D_2	I_2

$Class$	
I_1	K
I_2	K

Query for items of class K: $Class(z, K)$?

Answer: I_1, I_2

However, IC has not been considered, and I_2 is not a consistent answer

Instead, we query with

$$T(Class(z, K)) \equiv$$
$$Class(z, K) \ \wedge \ \forall(x, y)(Supply(x, y, z) \rightarrow x = C)$$

Only consistent answer: I_1

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

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

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

And again:

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

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

Soundness

Sufficient conditions: Universal ICs

ICs and queries equivalent to formulas with universal quantifiers in prenex normal form

E.g. ICs that are functional dependencies, some inclusion dependencies, like $\forall x(P(x) \rightarrow Q(x))$, $\forall x \forall y(R(x,y) \rightarrow P(x))$

AND

- Universal queries; or
- Domain independent, but possibly non-universal queries

  E.g. $\exists x P(x,y)$

  $\exists x \forall y(R(x,y) \rightarrow S(x,y))$

Completeness

Sufficient conditions: Queries that are conjunctions of literals plus

- Sets of ICs IC for which both 1. and 2. hold:

  1. Binary integrity constraints, i.e. each of them mentions only two database relations
  2. Generic integrity constraints that do not determine the truth for particular tuples

  I.e. for every ground database tuple $P(\bar{a})$:

  Neither $IC \models P(\bar{a})$ nor $IC \models \neg P(\bar{a})$ hold

Example:

Queries: $P(u, v)$ and $R(u, v) \wedge \neg P(u, v)$

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

(FDs, full inclusion dependencies, ...)

Or

- Any set of integrity constraints, under a condition that is similar to 2., but stronger

Syntactic Finite Termination

$T^\omega(Q(\bar{x}))$ is syntactically finite if there is $n \in \mathbb{N}$ such that $T^n(Q(\bar{x}))$ and $T^{n+1}(Q(\bar{x}))$ are syntactically the same

It holds for any kind of queries iff IC is acyclic

I.e. there is $f\colon \{P_1, \ldots, P_n, \neg P_1, \ldots, \neg P_n\} \longrightarrow \mathbb{N}$,
a level mapping on database "literal" predicates, such that for every IC

$$\forall(\bigvee_{i=1}^{k} l_i(\bar{x}_i) \vee \psi(\bar{x})) \in IC$$

and every $1 \leq i, j \leq k$, if $i \neq j$ then $f(\neg l_i) > f(l_j)$

Like hierarchical logic programs, but complementary literals get values independently from each other

Intuitively, given IC $\forall(l_1(\bar{x}_1) \vee \cdots \vee l_k(\bar{x}_k) \vee \psi(\bar{x}))$, and want to find consistent answers to $\neg l_i(\bar{x}_i)$, then

$$l_1(\bar{x}_1), \ldots, l_{i-1}(\bar{x}_{i-1}), l_{i+1}(\bar{x}_{i+1}), \ldots, l_k(\bar{x}_k), \psi(\bar{x})$$

have to be evaluated; we expect them to have a lower level than $\neg l_i$

Example: Query $R(u, v) \wedge P(u, v)$

$$\begin{aligned} IC \;\; = \;\; & \{\forall x, y(P(x, y) \rightarrow R(x, y)), \\ & \forall x, y, z(P(x, y) \wedge P(x, z) \rightarrow y = z)\} \end{aligned}$$

- To compute P, from the first IC we get residue R: it should be $f(P) > f(R)$
- For P, from the second IC we get (as a part of) the residue $\neg P$: it should be $f(P) > f(\neg P)$

- To compute $\neg P$: no residue
- To compute R: no residue
- To compute $\neg R$, from the first IC we get residue $\neg P$: it should be $f(\neg R) > f(\neg P)$

The level mapping $f: \ P \mapsto 2, \ R \mapsto 1, \ \neg P \mapsto 1, \ \neg R \mapsto 2$ satisfies de constraints; and IC is acyclic

FDs are always acyclic ...

With some inclusion dependencies, acyclicity may be lost

Checking acyclicity is obviously decidable

Semantic Finite Termination

$T^\omega(Q(\bar{x}))$ is semantically finite if there is an $n \in \mathbb{N}$, such that for all $m \geq n$, $\forall \bar{x}(T^n(Q(\bar{x}) \equiv T^m(Q(\bar{x}))$ is valid

Sufficient conditions: Queries that are (conjunctions of) literals $l(\bar{x})$ plus

- Uniform constraints, i.e. same variables in all literals

  Or

- Any set of ICs such that for some $n \in \mathbb{N}$:

$$\forall \bar{x}\ T^n(l(\bar{x})) \rightarrow T^{n+1}(l(\bar{x}))$$

  is logically true (good enough to check base literals)

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

For queries $P(u,v)$ and $R(u,v)$, it holds $T^2 \rightarrow T^3$

Example: Multivalued dependencies, of the form

$$P(x,y,z) \wedge P(x,u,v) \rightarrow P(x,y,v)$$

It holds $T^3 \rightarrow T^4$

In this case, automated theorem proving (Otter) has been successfully used to check the implication

Implementation

Semantic termination is difficult to detect and implement

A new algorithm, *QUECA*, inspired by T was introduced
(Celle, Bertossi; DOOD 00)

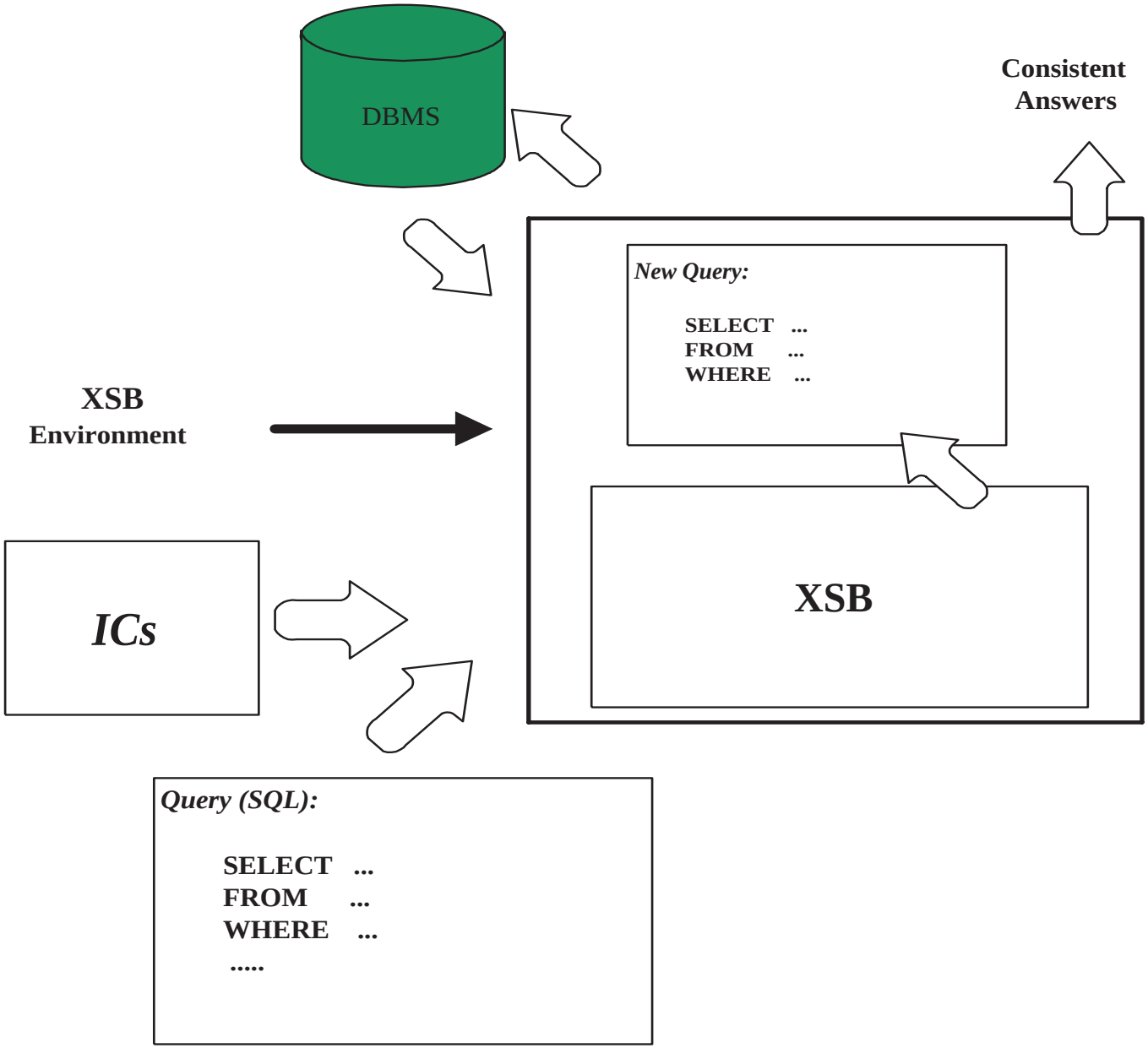
- It syntactically terminates for a wider class of ICs
- Based on a careful syntactical analysis and memorization of residues and subsumptions between them
- Implemented on XSB
  About XSB: (Sagonas, Swift, Warren; SIGMOD 94)

Implementation in XSB makes it possible:

- Trying direct unification between residues
- Using tabling to avoid redundant computation of residues
- Interaction with DBMSs; in our case, IBM DB2

Methodology works for universal binary constraints, i.e. containing at most two database literals plus built-ins, e.g.

- FDs: $P(u, x, y) \wedge P(v, x, z) \rightarrow \ y = z$
- Full inclusion dependencies: $P(\bar{x}) \rightarrow Q(\bar{x})$
- Range constraints: $P(x, y) \rightarrow \ y < 100$


DBMS
Consistent
Answers
New Query:
SELECT ...
FROM ...
WHERE ...
XSB
Environment
XSB
ICs
Query (SQL):
SELECT ...
FROM ...
WHERE ...
.....

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: Vertices are the DB tuples, and their simultaneous semantic conflicts under denial ICs are the hyperedges
  (Chomicki, Marcinkowski; Inf. and Comp. 2005)

  Graph based algorithms on original query can be translated into SQL queries
  (Chomicki, Marcinkowski, Staworko; demos at EDBT 2004)

- Specific methods for conjunctive queries containing restricted projections (existential quantifiers) and FDs (Fuxman, Miller; ICDT 2005)

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

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

Example: Database D:

$Employee$	$Name$	$Salary$
	$J.Page$	5000
	$V.Smith$	3000
	$M.Stowe$	7000

and $FD: Name \rightarrow Salary$, it holds

$$D \models_{FD} Employee(J.Page,\ 5000)$$

However

$$D \cup \{Employee(J.Page,\ 8000)\} \not\models_{FD} Employee(J.Page,\ 5000)$$

3. Specifying Repairs in APC

Annotated Logic

We want to specify database repairs, by means of a consistent theory

The database instance D (seen as a set of ground atomic formulas) and the set of integrity constraints IC are mutually inconsistent

Use a different logic, that allows generating a consistent theory!

Use annotated predicate calculus (APC)
(Kifer, Lozinskii; J. Aut. Reas. 92)

Inconsistent classical theories can be translated into consistent annotated theories

Usual annotations: true ($\mathbf{t}$), false ($\mathbf{f}$), contradictory ($\top$), unknown ($\bot$)

Atoms in an APC theory are annotated with truth values, at the object level, e.g.
$Employee(V.Smith, 3000)$:$\mathbf{t}$, $Employee(V.Smith, X)$:$\mathbf{f}$,
$Employee(V.Smith, X)$:$\top$

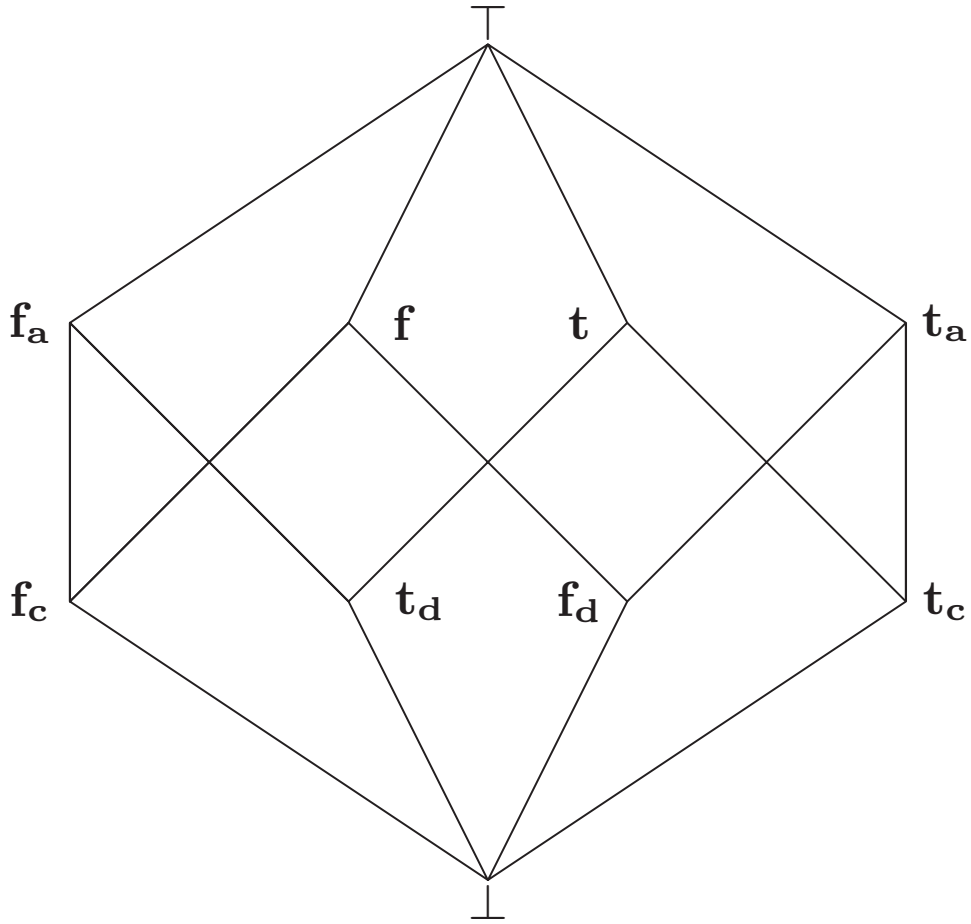
Embed both D and IC into a single consistent APC theory (Arenas, Bertossi, Kifer; DOOD 00)

- ICs are hard, not to be given up
- Data is flexible, subject to repairs
- In case of conflict between the constraint and the database the advise is to change the truth value to the value prescribed by the constraint

Choose an appropriate truth values lattice $\mathcal{L}at$:

- Database values: $\mathbf{t_d}$, $\mathbf{f_d}$
- Constraint values: $\mathbf{t_c}$, $\mathbf{f_c}$
- Advisory values: $\mathbf{t_a}$, $\mathbf{f_a}$

  They advise to solve conflicts between $\mathbf{d}$-values and $\mathbf{c}$-values in favor of $\mathbf{c}$-values



Intuitively, ground atoms A for which $A{:}\mathbf{t_a}$ or $A{:}\mathbf{f_a}$ become true are to be inserted into, resp. deleted from D

Navigation in the lattice plus an adequate definition of APC formula satisfaction help solve the conflicts between database facts and constraint facts

- For every $\mathbf{s} \in \mathcal{L}at$: $\bot \leq \mathbf{s} \leq \top$
- $lub(\mathbf{t}, \mathbf{f}) = \top$, $lub(\mathbf{t_c}, \mathbf{f_d}) = \mathbf{t_a}$, etc.
- Use Herbrand structures, i.e sets of ground annotated atoms, with the DB domain as the universe
- Formula satisfaction: I a structure, $\mathbf{s} \in \mathcal{L}at$, A a classical atomic formula

  $I \models A{:}\mathbf{s}$ iff there exists $s' \in \mathcal{L}at$ such that $A{:}\mathbf{s}' \in I$ and $\mathbf{s} \leq \mathbf{s}'$

  For other formulas, as usual in FO logic

Generate an APC theory $Spec$ embedding D and IC into APC:

- Translate the constraint:

  $\neg Employee(X, Y) \ \vee \ \neg Employee(X, Z) \ \vee \ Y = Z$

  into

  $Employee(X, Y)\text{:}\mathbf{f_c} \ \vee \ Employee(X, Z)\text{:}\mathbf{f_c} \ \vee \ Y = Z\text{:}\mathbf{t}$

- Translate database facts, e.g. $Employee(J.Page, 5,000)$ into $Employee(J.Page, 5,000)\text{:}\mathbf{t_d}$

- Plus axioms for unique names assumption, closed world assumption, ...

Due to the notion of satisfaction, we concentrate on models that have their atoms annotated with $\mathbf{t_a}$, $\mathbf{f_a}$, $\mathbf{t}$ or $\mathbf{f}$ only

(It can be proved that an atom in a model of the theory is never annotated with $\top$, that is, our theory is *epistemologically* consistent)

It can be proved that the database repairs correspond to the models of $Spec$ that make true a minimal set of atoms annotated with $\mathbf{t_a}$, $\mathbf{f_a}$

Change a minimal set of database atoms!!!

Reasoning with the minimal models of $Spec$ makes reasoning non-monotonic, as expected

From the specification $Spec$ algorithmic and complexity results for consistent query answering can be obtained

Most importantly, this approach motivated a more general and practical approach to specification of database repairs based on logic programs

Example

D:

Employee		
Name	*Position*	*Salary*
Steven Lerman	*CEO*	4,000
Irwin Pearson	*Salesman*	2,000
Irwin Pearson	*Salesman*	2,500
John Miller	*Salesman*	1,600

Integrity constraints IC: Universally quantified disjunction of DB literals plus built-ins

$$Employee(x, y, z) \wedge Employee(x, u, v) \rightarrow y = u$$
$$Employee(x, y, z) \wedge Employee(x, u, v) \rightarrow z = v$$

D has two repairs:

Employee

Name	*Position*	*Salary*
Steven Lerman	*CEO*	4,000
Irwin Pearson	*Salesman*	2,000
John Miller	*Salesman*	1,600

Employee

Name	*Position*	*Salary*
Steven Lerman	*CEO*	4,000
Irwin Pearson	*Salesman*	2,500
John Miller	*Salesman*	1,600

The APC theory $Spec$:

- $$\neg Employee(x, y, z) \vee \neg Employee(x, u, v) \vee y = u$$
  $$\neg Employee(x, y, z) \vee \neg Employee(x, u, v) \vee z = v$$

  are transformed into

  $$Employee(x, y, z) : \mathbf{f_c} \vee Employee(x, u, v) : \mathbf{f_c} \vee y = u : \mathbf{t_c}$$
  $$Employee(x, y, z) : \mathbf{f_c} \vee Employee(x, u, v) : \mathbf{f_c} \vee z = v : \mathbf{t_c}$$

- We also add for every predicate two rules:

  $$Employee(x, y, z) : \mathbf{t_c} \vee Employee(x, y, z) : \mathbf{f_c}$$
  $$\neg(Employee(x, y, z) : \mathbf{t_c}) \vee \neg(Employee(x, y, z) : \mathbf{f_c})$$

  A unique constraint truth value!

- Transforming Database Instance:

$$Employee(Steven\ Lerman, CEO, 4{,}000) : \mathbf{t_d}$$
$$Employee(Irwin\ Pearson, Salesman, 2{,}000) : \mathbf{t_d}$$
$$Employee(Irwin\ Pearson, Salesman, 2{,}500) : \mathbf{t_d}$$
$$Employee(John\ Miller, Salesman, 1{,}600) : \mathbf{t_d}$$

- Closed World Assumption:

$$Employee(x, y, z) : \mathbf{f_d}$$
$$\vee$$
$$x = Steven\ Lerman : \mathbf{t_d} \wedge y = CEO : \mathbf{t_d} \wedge z = 4{,}000 : \mathbf{t_d}$$
$$\vee$$
$$x = Irwin\ Pearson : \mathbf{t_d} \wedge y = Salesman : \mathbf{t_d} \wedge z = 2{,}000 : \mathbf{t_d}$$
$$\vee$$
$$x = Irwin\ Pearson : \mathbf{t_d} \wedge y = Salesman : \mathbf{t_d} \wedge z = 2{,}500 : \mathbf{t_d}$$
$$\vee$$
$$x = John\ Miller : \mathbf{t_d} \wedge y = Salesman : \mathbf{t_d} \wedge z = 1{,}600 : \mathbf{t_d}$$

- Equality theory plus Unique Names Assumption

  True built-in atoms:

  $Steven\ Lerman = Steven\ Lerman : \mathbf{t}$
  $CEO = CEO : \mathbf{t}$
  $2{,}000 = 2{,}000 : \mathbf{t}$, etc.

  False built-in atoms:

  $Steve\ Lerman = Irwin\ Pearson : \mathbf{f}$
  $Irwin\ Pearson = Steve\ Lerman : \mathbf{f}$
  $CEO = Salesman : \mathbf{f}$
  $Salesman = CEO : \mathbf{f}$, etc.

  Finally,the axiom:

  $\neg(x = y : \top)$ (a unique truth value)

Every model of $\mathcal{T}(D, IC)$ assigns values $\mathbf{t}$, $\mathbf{f}$, $\mathbf{t_a}$, $\mathbf{f_a}$ (only) to atoms

The minimal models of $Spec$ with respect to $\Delta = \{\mathbf{t_a}, \mathbf{f_a}\}$ correspond to the repairs of the database:

- Comparison wrt to inclusion of the sets of atoms annotated with $\mathbf{t_a}, \mathbf{f_a}$ in each model

- For a Δ-minimal model $\mathcal{M}$,

  $$D_{\mathcal{M}} = \{p(\bar{a}) \mid \mathcal{M} \models p(\bar{a}) : \mathbf{t} \vee p(\bar{a}) : \mathbf{t_a}\}$$

  is a repair of D

  And every repair can be obtained in this way

The minimal models are $\mathcal{M}_1$:

$Employee(Steven\ Lerman, CEO, 4{,}000) : \mathbf{t}$

$Employee(Irwin\ Pearson, Salesman, 2{,}000) : \mathbf{t}$

$Employee(Irwin\ Pearson, Salesman, 2{,}500) : \mathbf{f_a} \Leftarrow$

$Employee(John\ Miller, Salesman, 1{,}600) : \mathbf{t}$

and $\mathcal{M}_2$:

$Employee(Steven\ Lerman, CEO, 4{,}000) : \mathbf{t}$

$Employee(Irwin\ Pearson, Salesman, 2{,}000) : \mathbf{f_a} \Leftarrow$

$Employee(Irwin\ Pearson, Salesman, 2{,}500) : \mathbf{t}$

$Employee(John\ Miller, Salesman, 1{,}600) : \mathbf{t}$

Then, $D_{\mathcal{M}_1}$:

$Employee(Steven\ Lerman, CEO, 4{,}000)$

$Employee(Irwin\ Pearson, Salesman, 2{,}000)$

$Employee(John\ Miller, Salesman, 1{,}600)$

and $D_{\mathcal{M}_2}$:

$Employee(Steven\ Lerman, CEO, 4{,}000)$

$Employee(Irwin\ Pearson, Salesman, 2{,}500)$

$Employee(John\ Miller, Salesman, 1{,}600)$

Consistent Query Answering

We have embedded D, IC and built-in atoms into a consistent APC theory

FO queries waiting for consistent answers can be transformed into APC queries

- FO query $Q(\bar{x})$
- compute $Q^{an}(\bar{x})$ simultaneously replacing
  - negative DB literals
    $$\neg p(\bar{s}) \quad \mapsto \quad p(\bar{s}){:}\mathbf{f} \ \vee \ p(\bar{s}){:}\mathbf{f_a}$$
  - positive DB literals
    $$p(\bar{s}) \quad \mapsto \quad p(\bar{s}){:}\mathbf{t} \ \vee \ p(\bar{s}){:}\mathbf{t_a}$$
  - Built-in literals: $p(\bar{s}) \quad \mapsto \quad p(\bar{s}){:}\mathbf{t}$

(logically equivalent FO queries produce annotated queries with the same consistent answers)

Example: Want the consistent answers to the query

$Q(x)$: $\exists y \exists z \exists w \exists t(Book(x, y, z) \wedge Book(x, w, t) \wedge y \neq w)$

Generate $Q^{an}(\bar{x})$:

$$\exists y \exists z \exists w \exists t(Book(x, y, z):\mathbf{t} \vee Book(x, y, z):\mathbf{t_a}) \wedge (Book(x, w, t):\mathbf{t} \vee Book(x, w, t):\mathbf{t_a}) \wedge (y \neq w):\mathbf{t})$$

Theorem: Given D, IC, FO query $Q(\bar{x})$:

$$D \models_{IC} Q(\bar{t}) \text{ iff } \mathcal{T}(D, IC) \models_{\Delta} Q^{an}(\bar{t})$$

RHS means true wrt Δ-minimal models of the APC theory

Consistent query answering is reduced to non-monotonic entailment in APC

What about answer computation on the RHS?

4. Specifying Repairs with Logic Programs

Stable Model Semantics for Repairs

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 $IC\colon\ \forall\bar{x}(P(\bar{x}) \rightarrow Q(\bar{x}))$

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

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

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

2. Whatever was true (false) or becomes true (false), gets annotated with $\mathbf{t}^\star$ ($\mathbf{f}^\star$):

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

... 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

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

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

$$\mathcal{M}_1 = \{P(c,l,\mathbf{t_d}), ..., P(c,l,\mathbf{t}^\star), Q(c,l,\mathbf{f}^\star), Q(c,l,\mathbf{t_a}), P(c,l,\mathbf{t}^{\star\star}), Q(c,l,\mathbf{t}^\star), P(d,m,\mathbf{t}^{\star\star}), Q(d,m,\mathbf{t}^{\star\star}), \ldots, Q(c,l,\mathbf{t}^{\star\star}), ...\}$$
$$\equiv \{P(c,l), Q(c,l), P(d,m), Q(d,m), Q(e,k)\}$$

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

$$\mathcal{M}_2 = \{P(c,l,\mathbf{t_d}), ..., P(c,l,\mathbf{t}^\star), P(c,l,\mathbf{f}^\star), Q(c,l,\mathbf{f}^\star), P(c,l,\mathbf{f}^{\star\star}), Q(c,l,\mathbf{f}^{\star\star}), P(d,m,\mathbf{t}^{\star\star}), Q(d,m,\mathbf{t}^{\star\star}), \ldots, P(c,l,\mathbf{f_a}), ...\}$$
$$\equiv \{P(d,m), Q(d,m), Q(e,k)\}$$

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

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

Methodology:

1. $Q(\cdots P(\bar{u}) \cdots) \longmapsto Q' := Q(\cdots P(\bar{u}, \mathbf{t}^{\star\star}) \cdots)$
2. $Q'(\bar{x}) \longmapsto (\Pi(Q'), Ans(\bar{X}))$
   (Lloyd-Topor transformation)
   - $\Pi(Q')$ 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 $\Pi(Q')$
3. "Run" $\Pi := \Pi(Q') \cup \Pi(D, IC)$
4. Collect ground atoms
   $Ans(\bar{t}) \in \bigcap\{S \mid S \text{ is a stable model of } \Pi\}$

Example: (continued)

- Consistent answers to query $P(x, y)$?

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 tuple (d, m)

- Consistent answers to query $\exists y Q(x, y)$?

Run repair program with query $Ans(x) \leftarrow Q(x, y, \mathbf{t}^{\star\star})$

Obtain answer values d, e

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

Run repair program with either of the queries

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

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

No ground Ans atoms can be found in the intersection of the two (extended) models

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

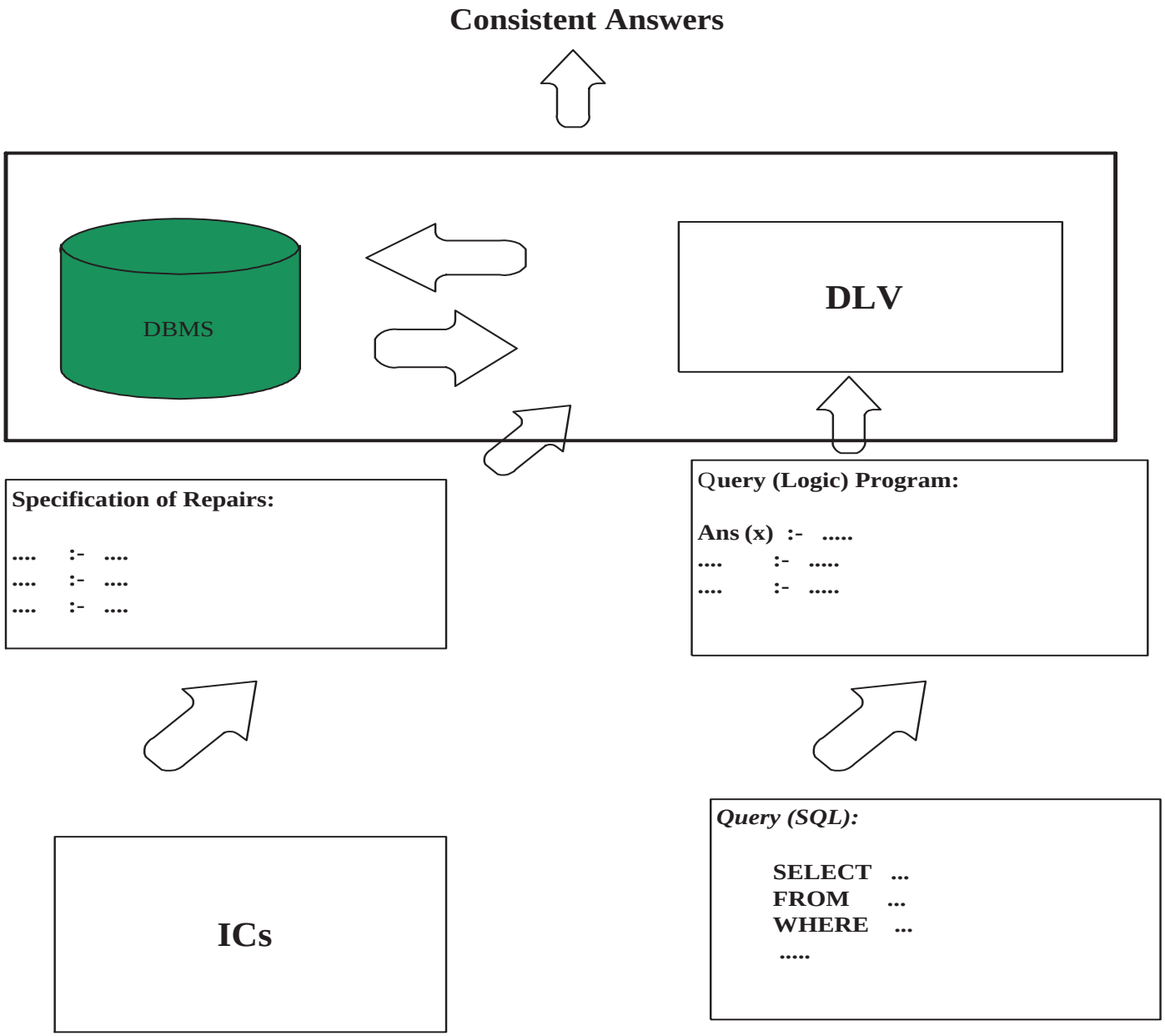
Remarks:

- 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)

- Related methodologies:
  (Arenas, Bertossi, Chomicki; TPLP 03)
  (Greco, Greco, Zumpano; IEEE TKDE 03)

- We have successfully experimented with the DLV system for computing the stable models semantics
  (N. Leone et al.; ACM Transactions on Comp. Logic)


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

- 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, are handled with introduction of null values or cascaded deletions (see later)
  (Barcelo,Bertossi,Bravo; LNCS 2582)
  (Bravo,Bertossi; CASCON 04)

- The program can be optimized in several ways; e.g. eliminating: materialization of CWA, annotations of DB facts, some annotation constants, coherence program denials (sometimes), etc.
  (Barcelo,Bertossi,Bravo; LNCS 2582),
  (Caniupan, Bertossi; 2005)

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

Facts of the program are the atoms $P(\bar{c})$ in the database, without annotation

$$
\begin{array}{rcl}
P'(X,Y,\mathbf{t_d}) & \leftarrow & P(X,Y) \\
P'(X,Y,\mathbf{t}^\star) & \leftarrow & P'(X,Y,\mathbf{t_d}) \\
P'(X,Y,\mathbf{t}^\star) & \leftarrow & P'(X,Y,\mathbf{t_a}) \\
P'(X,Y,\mathbf{f_a}) & \vee & P'(Y,X,\mathbf{t_a}) \leftarrow P'(X,Y,\mathbf{t}^\star), \\
 & & \quad not\ P'(Y,X,\mathbf{t_d}) \\
P'(X,Y,\mathbf{f_a}) & \vee & P'(Y,X,\mathbf{t_a}) \leftarrow P'(X,Y,\mathbf{t}^\star), P'(Y,X,\mathbf{f_a}) \\
P'(X,Y,\mathbf{t}^{\star\star}) & \leftarrow & P'(X,Y,\mathbf{t_a}) \\
P'(X,Y,\mathbf{t}^{\star\star}) & \leftarrow & P'(X,Y,\mathbf{t_d}),\ not\ P'(X,Y,\mathbf{f_a}) \\
 & \leftarrow & P'(X,Y,\mathbf{t_a}), P'(X,Y,\mathbf{f_a})
\end{array}
$$

Open problems and 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 stable models completely is undesirable

  Better try generation of "partial" repairs, relative to the ICs and the data that is relevant to them; and efficient and compact encoding of the collection of stable models

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

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

  Magic sets (or similar query-directed methodologies) for evaluating logic programs could be used for CQA
  (Greco et al.; TPLP 2005?),
  (Faber et al.; ICDT 2005)

- Efficient integration of relational databases and answer set programming environments

5. Referential ICs and Incomplete Information

Inconsistent Databases

DBMSs should provide more flexible, powerful, and user friendly mechanisms for dealing with semantic constraints

ICs could be another input to query answering process, and be taken into account as answers are computed

A query expressed in a possible enhanced version of SQL

```
SELECT Name, Salary
FROM Employee
WHERE Position = 'manager'
CONSIST/W FD: Name -> Salary;
```

Where the FD may not be maintained by the DBMS

Consistent answers to queries are true in every possible repair; and a repair is minimally repaired version of the original instance; in particular, a repair satisfies the ICs

So, in order to characterize and compute CQAs we have to agree on the semantics of IC satisfaction

But, what is the right semantics of IC satisfaction in the presence of NULL values?

Little agreement in the literature on the semantics of incomplete information

In particular, different DBMSs, like IBM DB2, have different semantics implemented (if they can be called "semantics")

We want a suitable NULL value semantics that:

- Generalizes the semantic used by IBM DB2
- Is uniform for a wider class of ICs

This semantics would allow as to integrate our results on CQA in a compatible way with current commercial implementations

NULL values are not only important because the DB may contain them, but also because they can be used in a natural way to repair an inconsistent database

Integrity Constraints

(1) Universal Integrity Constraints (UIC)

$$\bar{\forall}(\bigwedge_{i=1}^{m} P_i(\bar{x}_i) \rightarrow \bigvee_{j=1}^{n} Q_j(\bar{y}_j) \vee \varphi)$$

where φ contains only built-in atoms, e.g. $X = Y,\ X = a$

For example:
Full inclusion dependencies, like $P[X, Y] \subseteq R[X, Y]$:

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

Functional dependencies, like T: $X \rightarrow Y$:

$$\forall xy_1y_2(T(x, y_1) \wedge T(x, y_2) \rightarrow y_1 = y_2)$$

(2) Referential Integrity Constraints (RICs)

$$\forall \bar{x}\ (P(\bar{x}) \rightarrow \exists \bar{y}\ Q(\bar{x}', \bar{y})), \quad \text{with } \bar{x} \subseteq \bar{x}'$$

For example, the non-full inclusion dependency $S[X] \subseteq V[X]$:

$$\forall x \forall y (S(x, y) \rightarrow \exists u V(x, u))$$

We basically know how to repair universal ICs, c.f. chapter 4

What about referential ICs?

In the example above, if we want to repair by inserting tuples (x, u), what value does u should take?

Most naturally, a NULL value ...

Example

RIC: Course[StdID] $\subseteq$ Student[StdID]

DB:

Course	StdID	Code
	21	*COMP1512*
	34	*COMP1805*

Student	StdID	Name
	21	*Ann*
	45	*Paul*

Repairs?

- Delete tuple *(34, COMP1805)*, or

- Add a tuple to table Student

One repair:

Course	StdID	Code
	21	*COMP1512*

Student	StdID	Name
	21	*Ann*
	45	*Paul*

The other(s):

Course	StdID	Code
	21	*COMP1502*
	34	*COMP1805*

Student	StdID	Name
	21	*Ann*
	45	*Paul*
	34	*???*

What value for attribute Name?

We can get a specific repair by giving to Name an arbitrary value taken from the data base domain

An infinite number of repairs; not very appealing ...

The decision problem of CQA wrt universal and referential ICs is undecidable under this repair semantics
(Cali, Lembo, Rosati; PODS 03)

A better alternative seems to be inserting NULL values to repair RICs instead

However, no agreement in literature on the notion of satisfaction of ICs in DBs with NULL values

NULL values can be interpreted as Unknown, Not applicable

We can propose a (repair) semantics for (with) NULL values:

- Only one type of NULL value (no multi-NULL values)
- Staying close to the semantics of integrity satisfaction as implemented in DBMS like DB2 (no explicit formal semantics has been defined)

First let's see the issues ...

DB2 NULL Value Semantics (examples)

DB2 is compatible with the SQL:1999 standard

But not all of the SQL standard is implemented in DB2,[1]
in particular, not all the ICs in the standard are supported

DB2 enforces primary keys, unique constraints, foreign key constraints, NOT NULL, and check constraints

NULL accepted unless the column is restricted to NOT NULL

Primary keys and UNIQUEs have to be set to NOT NULL

[1] Same claim applies to the other well-known commercial DBMSs

```
CREATE TABLE EMPLOYEE (
    ID            INTEGER NOT NULL,
    NAME          VARCHAR(15) NOT NULL,
    SALARY        INTEGER CHECK(SALARY >0),
    POSITION      VARCHAR(10),
    MANAGER       INTEGER,
    PRIMARY KEY (ID),
    UNIQUE (NAME),
    FOREIGN KEY (MANAGER) REFERENCES EMPLOYEE,
    CHECK (POSITION ='MANAGER' OR SALARY < 60500),
    CHECK (POSITION ='MANAGER' OR MANAGER IS NOT NULL));
```

NULL in column Manager can be read as Not Applicable

SQL standard proposes a 3-valued logic, i.e. statements can be TRUE, FALSE or UNKNOWN

For example, all the following are UNKNOWN:

NULL = NULL	NULL = 'A'
NULL <> NULL	NULL <> 'A'

In SQL queries only tuples whose WHERE statement evaluates to TRUE are selected

A check constraint is accepted if the condition is TRUE or UNKNOWN for every row

Primary Key(StdID):

$$\forall xy(Student(x, y_1) \wedge Student(x, y_1) \rightarrow y_1 = y_2)$$

DB:

Student	StdID	Name
	21	*Ann*
	45	*NULL*

DB is accepted as a consistent state in DB2

The following SQL statements are rejected by DB2:

INSERT INTO STUDENT VALUES (21,Ann);

... because the DBMS only checks that the primary key is unique, and 21 is already in the table

INSERT INTO STUDENT VALUES (21,NULL);

Foreign key constraint:

$$\forall xyz(Course(x, y, z) \rightarrow \exists w Prof(z, w))$$

DB:

Course	Code	Term	ID
	COMP2702	W04	*21*
	COMP1805	*NULL*	*34*
	COMP5073	W05	*NULL*

Prof	ID	Name
	21	*Ann*
	34	*NULL*
	45	*Paul*

DB is accepted as a consistent state in DB2

NULL values in columns Term and Name are relevant to check satisfaction

If a NULL value is in the relevant attribute ID of table Course, the tuple is considered to be consistent without checking table Prof

The following SQL statement is rejected by DB2:

INSERT INTO COURSE VALUES (COMP4132, NULL,18);

The attribute ID is relevant to the constraint and is different from NULL, but there is no tuple in table Prof with ID=18

If the non-relevant attribute Term in Course takes a NULL, the condition in the IC will be checked, and there might be a violation, as with the last insertion

Check constraint:

$$\forall ID \forall Name \forall Salary\ (Emp(ID, Name, Salary) \rightarrow Salary > 0)$$

DB:

Emp	ID	Name	Salary
	32	*NULL*	*1000*
	41	*Paul*	*NULL*

DB is accepted as a consistent state in DB2

NULL does not compare with any value

The following SQL statement is rejected by DB2:

```
INSERT INTO Emp VALUES (64, John, -2500);
```

After these considerations, we see that there are several related challenges:

- Give a precise semantics for IC satisfaction in DBs
- One that is logically sound, and hopefully compositional
- One that extends or is compatible with practice in DBMSs
- Use NULL values to repair IC violations, considering that the DB may already contain them
- Be careful when repairing not to introduce new violations; or solve them before the repair process stabilizes
- Keep in mind that the goal is the right characterization and computation of consistent query answers

Repairs for DBs with NULL values

- Tuples with NULL values can be inserted into DBs to solve inconsistencies
- In principle, one could introduce new inconsistencies; however
- NULL values in relevant attributes are not propagated when repairing, because the ICs are satisfied (unless it is NOT NULL)
- Tuples with NULL values in non-relevant attributes can generate inconsistencies (they are treated as any usual value), but since the NULL values are in non-relevant attributes they will not propagate

Example: IC: $\forall xyz(Course(x, y, z) \rightarrow \exists w Prof(z, w))$ and DB:

Course	Code	Term	ID
	CS27	W04	*21*
	CS18	*NULL*	*34*
	CS50	W05	*NULL*

Prof	ID	Name
	21	*Ann*
	45	*Paul*

ID is the only relevant attribute since it is the only one needed to check the satisfaction of the constraint

Since tuple Course(*CS50, W05,NULL*) has a NULL in the relevant attribute ID, it cannot create an inconsistency, and so it is not propagated to table Prof in order to restore consistency

However, tuple Course(*CS18, NULL,34*) has a NULL value in a non-relevant attribute, therefore we need to check if the value of attribute ID, i.e 34, is in ID of table Prof

The value is not there, so we have an inconsistency, but the NULL does not propagate to Prof either

Now in order to repair we have two alternatives:

-Delete the tuple from $Course$, or

-Add tuple (34,NULL) to $Student$

Repair 1:

Course	Code	Term	ID
	CS27	W04	*21*
	CS50	W05	*NULL*

Prof	ID	Name
	21	*Ann*
	45	*Paul*

Repair 2:

Course	Code	Term	ID
	CS27	W04	*21*
	CS18	*NULL*	*34*
	CS50	W05	*NULL*

Prof	ID	Name
	21	*Ann*
	45	*Paul*
	34	*NULL*

NULL values do not propagate in the repair process!

As for universal ICs, and now with both universal and RICs, the repairs of database instances with NULL values and using NULL values can be specified by means of disjunctive programs with stable model semantics

The relevant new program rules are the following:

- $dom(a)$ for each constant $a \in U \setminus \{NULL\}$
- For every universal IC: (replace the old ones)

$$\bigvee_{i=1}^{n} P_i(\bar{x}_i, \mathbf{f_a}) \vee \bigvee_{j=1}^{m} Q_j(\bar{y}_j, \mathbf{t_a}) \leftarrow \bigwedge_{i=1}^{n} P_i(\bar{x}_i, \mathbf{t}^\star), \bigwedge_{Q_j \in Q'} Q_j(\bar{y}_j, \mathbf{f_a}), \bigwedge_{Q_k \in Q''} not\ Q_k(\bar{y}_k, \mathbf{t_d}), dom(\bar{x}_R).$$

for every sets Q' and Q'' such that $Q' \cup Q'' = \bigcup_{i=1}^{m} Q_i$ and $Q' \cap Q'' = \emptyset$, where $\bar{x}_R$ are the relevant attributes

- For every referential IC:

$$P(\bar{x}, \mathbf{f_a}) \vee Q(\bar{x}', null, \mathbf{t_a}) \leftarrow P(\bar{x}, \mathbf{t}^\star),\ not\ aux(\bar{x}'), dom(\bar{x}_R).$$
$$aux(\bar{x}') \leftarrow Q(\bar{x}', y, \mathbf{t_d}),\ not\ Q(\bar{x}', y, \mathbf{f_a}),\ dom(\bar{x}').$$
$$aux(\bar{x}') \leftarrow Q(\bar{x}', y, \mathbf{t_a}),\ dom(\bar{x}').$$

where $\bar{x}_R$ are the relevant attributes (notice that $\bar{x}' \subseteq \bar{x}_R$)
In the repair program the non propagation of NULL values is based on the use in te rules of predicate $dom(\bar{x})$, which does not contain NULL

Example: IC: $Course[StdID] \subseteq ID[StdID]$

Course	StdID	Code
	18	*NULL*
	NULL	*CS25*

ID	StdID

The relevant attribute is $StdID$

$Course(18, NULL)$ is involved in an inconsistency because 18 is in the relevant attribute of $Course$ but not in the relevant attribute of ID

$Course(NULL, CS25)$ is not involved in an inconsistency because there is a $NULL$ value in the relevant attribute

The principal rule in the repair program is:

$$Course(x, y, \mathbf{f_a}) \vee ID(x, \mathbf{t_a}) \leftarrow Course(x, y, \mathbf{t}^\star),\ ID(x, y, \mathbf{f}^\star),\ dom(x).$$

Predicate dom restricts the values in the relevant attribute $StdID$ to be different from NULL

The repairs are the following:

Course	StdID	Code
	NULL	*CS25*

ID	StdID

Course	StdID	Code
	18	*NULL*
	NULL	*CS25*

ID	StdID
	18

Example: IC: $\forall xyz(Course(x, y, z) \rightarrow \exists w Prof(z, w))$ and DB:

Course	Code	Term	ID
	CS27	W04	*21*
	CS18	*NULL*	*34*
	CS50	W05	*NULL*

Prof	ID	Name
	21	*Ann*
	45	*Paul*

The main rules are:

$$Course(x, y, z, \mathbf{f_a}) \vee Prof(z, null, \mathbf{t_a}) \leftarrow Course(x, y, z, \mathbf{t}^\star), \; not\ aux(z), dom(z).$$

$$aux(z) \leftarrow Prof(z, w, \mathbf{t_d}), \; not\ Prof(z, w, \mathbf{f_a}), \; dom(z).$$

$$aux(z) \leftarrow Prof(z, w, \mathbf{t_a}), \; dom(z).$$

$aux(z)$ says that there is something in Prof, and predicate dom restricts the values in the relevant attribute ID to non NULL

The repairs are:

Course	Code	Term	ID
	CS27	W04	*21*
	CS50	W05	*NULL*

Prof	ID	Name
	21	*Ann*
	45	*Paul*

Course	Code	Term	ID
	CS27	W04	*21*
	CS18	*NULL*	*34*
	CS50	W05	*NULL*

Prof	ID	Name
	21	*Ann*
	45	*Paul*
	34	*NULL*

Example: Inconsistent DB

P	X	Y
	a	*b*
	g	*NULL*

T	X
	c

R	X	Y
	a	NULL
	e	f

IC: $P[x, y] \subseteq R[x, y]$ $T[x] \subseteq P[x]$

Main repair rule:

$$P(x, y, \mathbf{f_a}) \vee R(x, y, \mathbf{t_a}) \leftarrow P(x, y, \mathbf{t}^\star),\ R(x, y, \mathbf{f}^\star),\ dom(x),\ dom(y).$$

Relevant attributes are X, Y, so their are restricted to be non NULL (for NULL values in them the IC is satisfied)

$$T(x, \mathbf{f_a}) \vee P(x, null, \mathbf{t_a}) \leftarrow T(x, \mathbf{t}^\star),\ not\ aux(x),\ dom(x).$$
$$aux(x) \leftarrow P(x, y, \mathbf{t_d}),\ not\ P(x, y, \mathbf{f_a}),\ dom(x).$$
$$aux(x) \leftarrow P(x, y, \mathbf{t_a}),\ dom(x).$$

Here the relevant attribute is X. If there is a NULL in Y, it is necessary to check if the tuple is involved in an inconsistency

The four repairs are obtained by the following changes (red means delete, blue means add):

$$\Delta(D, D_1) = \{P(c, NULL), R(a, b)\}$$
$$\Delta(D, D_2) = \{R(a, b), T(c)\}$$
$$\Delta(D, D_3) = \{P(a, b), P(c, NULL)\}$$
$$\Delta(D, D_4) = \{P(a, b), T(c)\}$$

We can see how NULL are not propagated by looking at repair D_3

P	X	Y
	g	*NULL*
	c	*NULL*

T	X
	c

R	X	Y
	a	NULL
	e	f

$(c, NULL)$ was inserted into P, but it does not propagate to table R (for the same reason that $P(g, null)$ in P, but not in R is not an inconsistency)

Final Remarks

- The programs just given are complete in the sense that every repair appears as a stable model
- When the set of RICs is cyclic (and only in this case), the program may have stable models that are not repairs; but it can be refined to obtain exactly repairs as stable models (a bit more involved)
- Since cycles are avoided and NULL is used to repair, with this semantics of satisfaction of ICs in DBs with NULL values and repairs with non propagated NULL values, CQA becomes decidable
- The programs just given have to be refined to deal with built-ins in ICs (comparisons with NULL values are problematic)

6. Aggregate Queries

So far only first order queries

What about aggregate queries?

- They are natural and usual in DBs, and part of SQL
- They are crucial in scenarios where inconsistencies are likely to occur, e.g. data integration, in particular, dataware-housing

We will see:

- Semantics may need revision
- Aggregation is challenging for CQA
- Some graph theoretic techniques can be developed

A restricted scenario:

- Functional dependencies

- Standard set of SQL-2 scalar aggregation operators: `MIN`, `MAX`, `COUNT(*)`, `COUNT(A)`, `SUM`, and `AVG`

  No `GROUP BY`

- Atomic queries applying just one of these operators

Redefining Consistent Anwers

Example: A database instance and a FD: $Name \rightarrow Amount$

$Salary$	$Name$	$Amount$
	$V.Smith$	5000
	$V.Smith$	8000
	$P.Jones$	3000
	$M.Stone$	7000

The repairs:

$Salary$	$Name$	$Amount$
	$V.Smith$	5000
	$P.Jones$	3000
	$M.Stone$	7000

$Salary$	$Name$	$Amount$
	$V.Smith$	8000
	$P.Jones$	3000
	$M.Stone$	7000

Query: `MIN(Amount)`?

We should get 3000 as a consistent answer: `MIN(Amount)` returns 3000 in every repair

Query: `MAX(Amount)`?

The maximum, 8000, comes from a tuple that participates in the violation of FD

`MAX(Amount)` returns a different value in each repair: 7000 or 8000

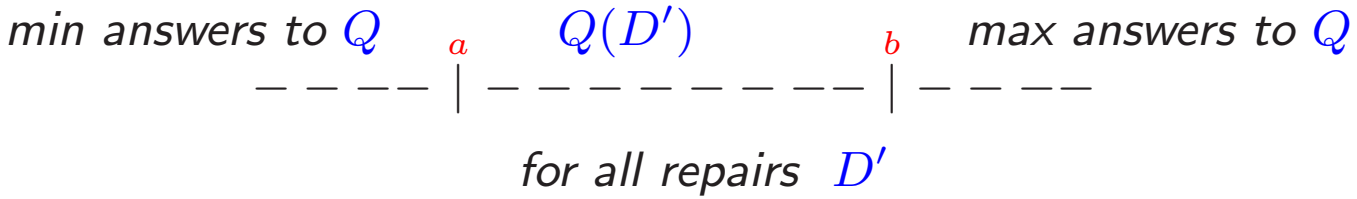
There is no consistent answer as previously defined

Modify the definition of consistent answer:

Definition: The consistent answer to an aggregate query Q in the database instance D is the shortest numerical interval that contains all the answers to Q obtained from the repairs of D

In the example $[7000, 8000]$ is the consistent answer to query `MAX(Amount)`

This is the range semantics for CQA (numerical queries) (Arenas, Bertossi, Chomicki; ICDT 01)



- a: the *max-min answer*
- b: the *min-max answer*

Problem: Develop algorithms for computing the optimal bounds: the *max-min answer* a, and the *min-max answer* b, by querying D only!

Sometimes we are interested in one of the two only

In the example, in min-max for `MIN(Amount)`, and max-min for `MAX(Amount)`

Problem: Determine the computational complexity of finding the min-max and max-min answers

We need the right tools to attack these problems ...

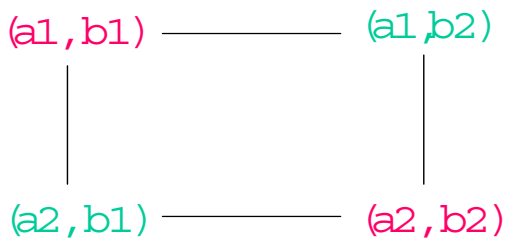
Graph Representation of Repairs

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

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

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

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



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 $CG_{FD}(D)$

Some Complexity Results

- MAX(A) can be different in every repair

  Maximum of the MAX(A)'s is MAX(A) in D

  Then computing the min-max answer to MAX(A) from D is direct

  $$\overset{b}{- - - - - - - \,|\, - -}$$

- Computing directly from D the minimum of the MAX(A)'s, i.e. the max-min answer to MAX(A), is not that direct

  $$- - \,\overset{a}{|}\, - - - - - -$$

  But still, computing the max-min answer to MAX(A) for one FD F is in PTIME (in data complexity)

Algorithm: For computing max-min answer to `MAX(A)` for the FD $R\colon\ X \to Y$

Actually a sequence of SQL queries to the inconsistent database: query rewriting

For each group of (X, Y)-values, store the maximum of `A`:

```
CREATE VIEW S(X,Y,C) AS
  SELECT X,Y,MAX(A) FROM R
  GROUP BY X,Y;
```

For each value of X, store the minimum of the maximums:

```
CREATE VIEW T(X,C) AS
  SELECT X, MIN(C) FROM S
  GROUP BY X;
```

Output the maximum of the minimums:

```
SELECT MAX(C) FROM T;
```

- For more than one FD, the problem of deciding whether the max-min answer to $\texttt{MAX(A)} \leq k$ is NP-complete

  NP-hardness: By reduction from SAT

  Propositional formula φ in CNF $C_1 \wedge \cdots \wedge C_n$ with propositional variables $p_1, \ldots p_m$

  Build database D with attributes X, Y, Z, W and tuples

  1. $(p_i, 1, C_j, 1)$ if making p_i true makes C_j true
  2. $(p_i, 0, C_j, 1)$ if making p_i false makes C_j true
  3. $(w, w, C_j, 2)$, $1 \leq j \leq n$, with w a new symbol

  Consider FD: $X \rightarrow Y$ (each propositional variable cannot have more than one truth value) and $Z \rightarrow W$

  φ is satisfiable iff for D, FD, and $k = 1$, the answer to our problem is Yes

Membership to NP: Take $D' \subseteq D$, a possible certificate

It is feasible to check whether D' is a repair of D (for functional dependencies) and $\texttt{MAX(A)} \leq k$ in D'

If max-min answer to $\texttt{MAX(A)} \leq k$, there is a (short) repair certificate D' that gives answer *Yes* to the question $\texttt{MAX(A)} \leq k$ in D'

- Even for one FD, the problem of deciding if the maximal min-answer to $\texttt{COUNT(A)} \leq k$ is NP-complete
  (reduction from HITTING SET)

In general:

	maximal min-answer		minimal max-answer	
	$\|FD\| = 1$	$\|FD\| \geq 2$	$\|FD\| = 1$	$\|FD\| \geq 2$
`MIN(A)`	PTIME	PTIME	PTIME	NP-complete
`MAX(A)`	PTIME	NP-complete	PTIME	PTIME
`COUNT(*)`	PTIME	NP-complete	PTIME	NP-complete
`COUNT(A)`	NP-complete	NP-complete	NP-complete	NP-complete
`SUM(A)`	PTIME	NP-complete	PTIME	NP-complete
`AVG(A)`	PTIME	NP-complete	PTIME	NP-complete

(Arenas,Bertossi,Chomicki,He,Raghavan,Spinrad; Theoretical Computer Science 2003)

These are results in data complexity, i.e. for fixed queries and FDs, and changing database instances (sizes)

Then, in most of the cases, query rewriting as on page 178 for CQA is bound to fail (unless $P = NP$)

We have identified normalization conditions, e.g. BCNF, (and other conditions) on the DB under which more efficient algorithms can be designed

However, improvements are not impressive

CQA for aggregate queries is an intrinsically complex problem

It seems necessary to approximate optimal consistent answers to aggregate queries, but "maximal independent set" seems to have bad approximation properties (see later ...)

Complexity analysis of aggregate queries opened the ground for more general study of complexity of CQA

7. Complexity of CQA

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

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

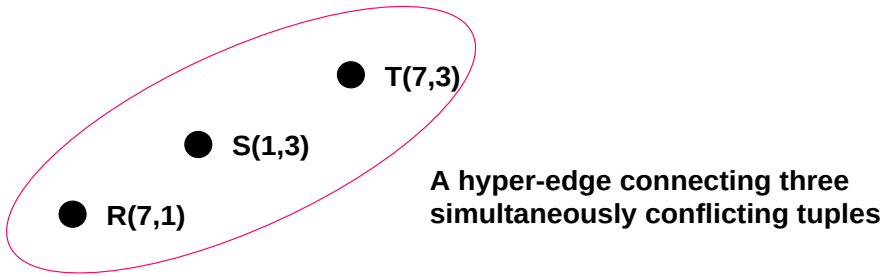
Graph theoretic techniques for CQA for aggregate queries were extended to:

- Extend the PTIME computation to other classes of FO queries, e.g. with very restricted forms of projection (existential quantifiers), but denial constraints

  E.g. of the form

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

  i.e. a forbidden conjunction of DB atoms plus built-ins



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

Now hyper-graphs ...

- Study the complexity of CQA for FO queries for wider classes of integrity constraints, e.g. including referential ICs (but only deletions for repair)

(Chomicki, Marcinkowski; Information and Computation 2005)

For FDs, conflict graphs are still good enough

Some Complexity Results

CQA is a decision problem:

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

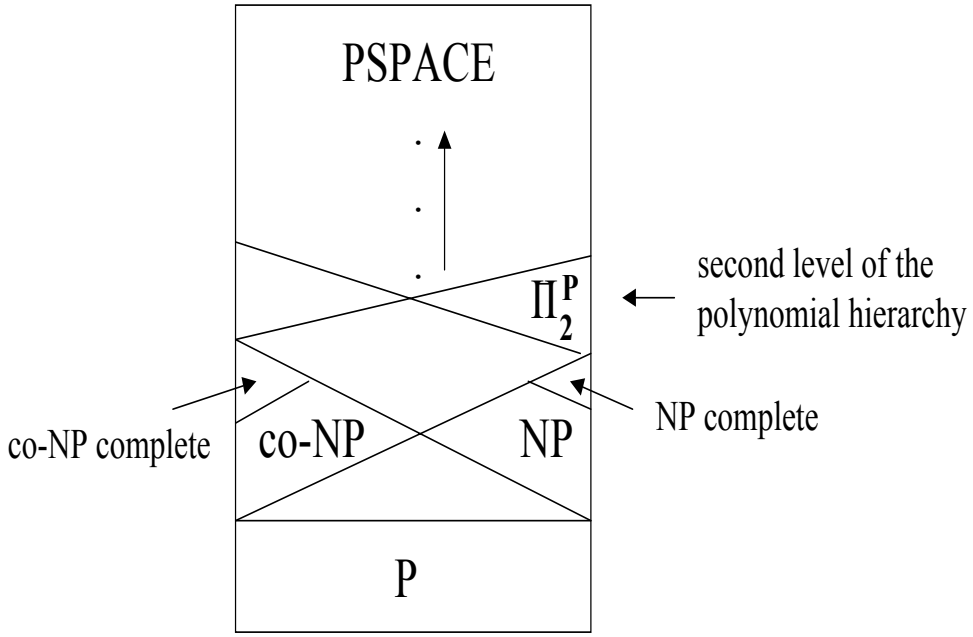
In those cases where CQA can be solved in PTIME, the problem of repair checking can be solved in PTIME

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

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

However: (deletions only)

- For arbitrary FDs and inclusion dependencies, repair checking becomes coNP-complete



- For FDs and some rather sharp syntactic classes of existentially quantified conjunctive queries, CQA becomes PTIME (Chomicki, Marcinkowski; op. cit.)
(Fuxman, Miller; ICDT 05)

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

- Then, forget about first-order query rewriting for CQA in the general case (unless the polynomial hierarchy collapses)!
- But query answering from disjunctive logic programs under skeptical stable models semantics is Π_2^P-complete!!
  (Dantsin, Eiter, Gottlob, Voronkov; ACM Computer Surveys 01)

  These logic programs with this semantics provide the right expressive and computational power

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

Among others:

- For arbitrary FDs and inclusion dependencies (in particular, referential ICs), CQA becomes undecidable

Issues?

- 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 disjunctive logic programs (DLPs) coincides with the complexity of CQA
- 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 classes of queries and ICs, CQA has a lower complexity, e.g. in P time
- It becomes relevant to identify classes of ICs and queries for which the DLP can be automatically "simplified" into, e.g. a FO query
- Upper bounds can be obtained from complexity theoretic results in the area of belief revision
  (Eiter, Gottlob; AIJ 92)

Comparison with Belief Revision/Update

There are several similarities

Here we have an empty domain theory, one model –the database instance, an a revision by a set of ICs

The revision of the database instance by IC produces new database instances, the repairs of the original database

The database repairs coincide with the revised models defined by Winslett in her "Possible Models Approach":

> They are the models that are closer to the original database instance wrt sets of changes and set inclusion

Our implicit notion of revision, satisfies then the postulates (R1) – (R5), (R7), (R8) introduced by Katsuno & Mendelzon

Winslett concentrates on the computation of the models of the revised theory, i.e. the repairs in our case

Instead

- Our motivation and starting point is different from belief revision
- We do not compute repairs (whenever possible), but keep querying the original database, posing a modified query
- We provide a methodology for representing and querying simultaneously and implicitly all the repairs of the database

- We deal with the first-order case wrt the original DB and ICs, and DB queries
- We take direct advantage of the semantic information contained in the ICs in order to answer queries, rather than revising the database
- Revising the database means repairing *all* the inconsistencies in it, instead we may be interested in *some* consistent information, the one related to particular queries
- In particular, a query referring only to the consistent portion of the database can be answered without repairing the database

There has also been some research on repairs that minimize the number of changes (still insertion/deletion of whole tuples)
(Arenas,Bertossi,Chomicki;TPLP 03)
(Lopatenko, Bertossi; submitted 2005)

In the belief revision/update they are usually called Dalal's models

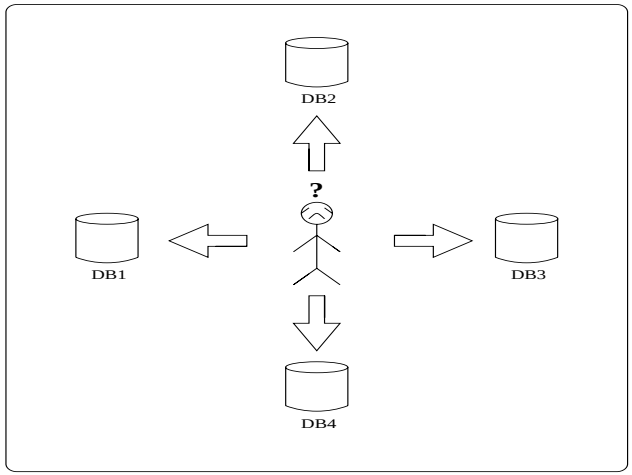
8. CQA in Virtual Data Integration

Virtual Data Integration

Number of available on-line information sources has increased dramatically

How can users confront such a large and increasing number of information sources?

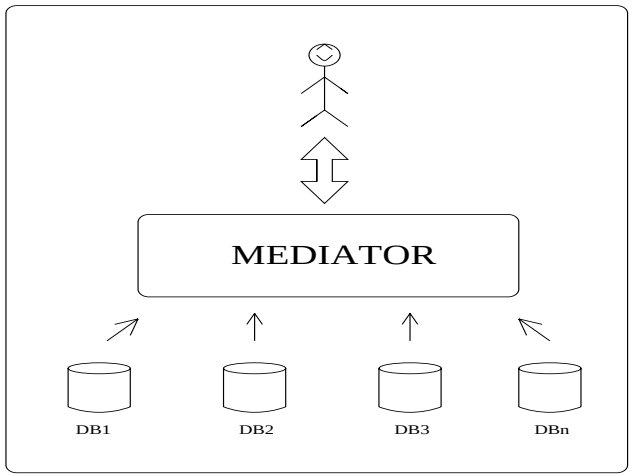
In particular, when information has to be integrated?



- Interacting with each of the sources independently?
- Considering all available sources?
- Selecting only those to be queried?
- Querying the relevant sources on an individual basis?
- Handcraft the combination of results from different sources?

A long, tedious, complex and error prone process

An approach: Virtual integration of sources via a mediator

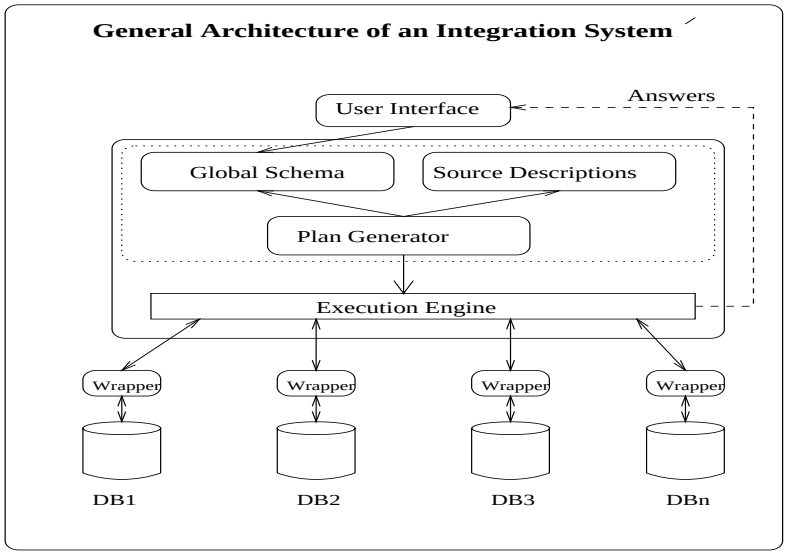


A software system that offers a common user interface to query a set of heterogeneous and independent data sources

System offers a single integrated, global schema

User feels like interacting with a single database

- Sources are mutually independent and non cooperative
- Data kept in sources, and extracted at mediator's request
- Interaction from the system to the sources via queries
- Mediator composes query results for the user
- Update operations are not supported via the mediator
- System should allow sources to get in and out



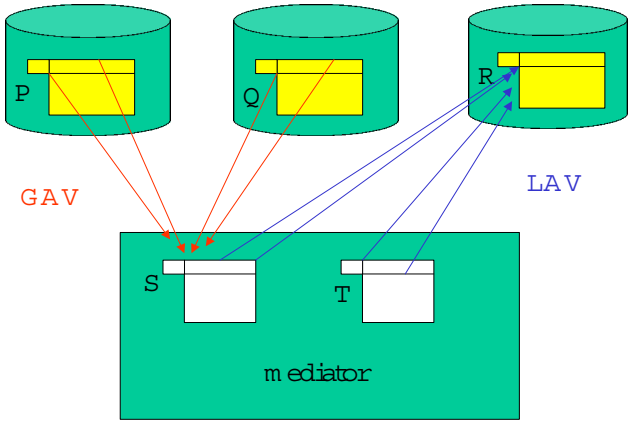
- User poses queries in terms of global schema
- Relationship between global schema and local, source schemas specified in the mediator, as source descriptions
- Mediator is responsible of solving problems of data:
  - redundancy: to avoid unnecessary computations
  - complementarity: data of the same kind may be spread through different sources
  - inconsistency: sources, independently, may be consistent, but together, possibly not

    E.g. Same ID card number may be assigned to different people in different sources

Description of the Sources:

- Mediator needs to know what data is in the sources and how it relates to the global schema
- Sources are described by means of logical formulas; like those used to express queries and define views
- Those formulas define the mappings between the global schema and the local schemas
- There are two main approaches (and combinations of them):

- Global-as-View (GAV): Relations in the global schema are described as views over the tables in the local schemas
- Local-as-View (LAV): Relations in the local schemas (at the source level) are described as views over the global schema
- Mixed approaches (GLAV): Mappings between views at the source and global levels
  c.f. survey in (Lenzerini; PODS 2002)


P
Q
R
GAV
LAV
S
T
mediator

Plan Generator:

- Gets a user query in terms of global relations
- Uses the source descriptions and rewrites the query as a query plan

  Which involves a set of queries expressed in terms of local relations
- Rewriting process depends on LAV or GAV approach
- Query plan includes a specification of how to combine the results from the local sources

LAV

LAV is more flexible wrt participation of sources; but more complex for query answering

A global data schema is designed, and contributors of data (sources) describe how their data fits into the integration system

Example: Definition of tables at the sources:

$$S_1(Title, Year, Director) \leftarrow Movie(Title, Year, Director, Genre), American(Director), Genre = comedy, Year \geq 1960.$$

S_1: comedies, after 1960, with American directors and years

$$S_2(Title,\ Review) \leftarrow$$
$$Movie(Title, Year, Director, Genre),$$
$$Review(Title, Review),\ Year \geq 1990.$$

S_2: movies after 1990 with their reviews, but no directors

Here, sources defined as conjunctive queries (views) with built-ins

Definition of each source does not depend on other sources

From the perspective of S_2, there could be other sources containing information about comedies after 1990 with their reviews, i.e. data in the sources could be "incomplete"

Query posed to $\mathcal{G}$: "Comedies with their reviews produced since 1950?"

$$Ans(Title,\ Review) \leftarrow Movie(Title, Year, Director, comedy),\ Review(Title, Review),\ Year \geq 1950.$$

Query expressed in terms of relations in global schema only

Not possible to obtain answers by a simple and direct computation of the RHS of the query: Information is in the sources, now views ...

A plan is a rewriting of the query as a set of queries to the sources and a prescription on how to combine their answers

A possible query plan for our query:

$$Ans'(Title, Review) \leftarrow S_1(Title, Year, Director), \\ S_2(Title, Review).$$

Query was rewritten in terms of the views; and now can be easily computed

Due to the limited contents of the sources, we obtain comedies by American directors with their reviews filmed after 1990

We get correct answers; and also the most we can get ...

Meaning?

Semantics of a LAV Data Integration System

We assume source relations are open or incomplete

Example: Global system $\mathcal{G}_1$ with source definitions and sources extensions

$$S_1(X,Y) \leftarrow R(X,Y) \quad \text{with} \quad s_1 = \{(a,b),(c,d)\}$$
$$S_2(X,Y) \leftarrow R(Y,X) \quad \text{with} \quad s_2 = \{(c,a),(e,d)\}$$

The global relations can be materialized in different ways, still satisfying the source descriptions, so different global instances are possible

A global (material) instance D is legal if the view definitions applied to it compute extensions $S_1(D), S_2(D)$ such that $s_1 \subseteq S_1(D)$ and $s_2 \subseteq S_2(D)$

That is, each source relation contains a subset of the data of its kind in the global system

$D = \{R(a, b), R(c, d), R(a, c), R(d, e)\}$ and its supersets are the legal instances

Global query Q: $R(X, Y)?$

What is a correct answer to the query, considering that there are many possible legal global instances?

The intended answers to a global query are the certain answers, those that can be obtained from all the legal instances

$$Certain_{\mathcal{G}_1}(Q) = \{(a, b), (c, d), (a, c), (d, e)\}$$

Certain answers to a query are true in all the legal instances

Consistency in Virtual Data Integration

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

How can we be sure that those global ICs hold?

They are not maintained at the global level

Most likely they are not fully satisfied

The goal is to retrieve answers to global queries from the virtual integration system that are "consistent with the ICs"

We need a characterization of consistent answers and a mechanism to obtain them ... at query time ...

Example: (continued) Global system $\mathcal{G}_1$

What if we had a global functional dependency $R\colon X \to Y$?

(local FDs $S_1\colon X \to Y$, $S_2\colon Y \to X$ satisfied in the sources)

Global FD not satisfied by $D = \{(a,b),(c,d),(a,c),(d,e)\}$
(nor by its supersets)

From the certain answers to the query Q: $R(X,Y)?$, i.e. from

$$Certain_{\mathcal{G}_1}(Q) = \{(a,b),(c,d),(a,c),(d,e)\}$$

only $(c,d),(d,e)$ should be consistent answers

Minimal Legal Instances and Consistent Answers

There are algorithms for generating plans to obtain the certain answers (with some limitations)

Not much for obtaining consistent answers

Here we do both, in stages ...

First concentrating on the minimal legal instances of a virtual systems, i.e. those that do not properly contain any other legal instance

- Minimal legal instances do not contain unnecessary information; that could, unnecessarily, violate global ICs

In the example, $D = \{R(a, b), R(c, d), R(a, c), R(d, e)\}$ is the only minimal instance

The minimal answers to a query are those that can be obtained from every minimal legal instance:

$$Certain_{\mathcal{G}}(Q) \subseteq Minimal_{\mathcal{G}}(Q)$$

For monotone queries they coincide

By definition, consistent answers to a global query wrt IC are those obtained from all the repairs of all the minimal legal instances wrt IC

(Bertossi, Chomicki, Cortes, Gutierrez; FQAS 02)

In the example:

- The only minimal legal instance

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

  violates the FD $R\colon X \rightarrow Y$

- Its repairs wrt FD are
  $D^1 = \{R(a, b), R(c, d), R(d, e)\}$ and
  $D^2 = \{R(c, d), R(a, c), R(d, e)\}$

  A repair of an instance D wrt a set of ICs is an instance D' that satisfies the ICs and minimally differs from D (under set inclusion, considering a DB as a set of facts)

- Consistent answers to query Q: $R(X, Y)?$
  Only $\{(c, d), (d, e)\}$

Computing consistent answers? (Idea)

(Bravo, Bertossi; IJCAI 03)

- Answer set programming (ASP) (stable model semantics) gives a semantics to Datalog (logic) programs with negation (and possibly disjunction in the heads)
- ASP based specification of minimal instances of a virtual data integration system
- ASP based specification of repairs of minimal instances
- Global query in Datalog (or its extensions) to be answered consistently
- Run the three combined programs above under skeptical answer set semantics (skeptical stable model semantics)

- Methodology works for first-order queries (and Datalog extensions), and universal ICs combined with referential ICs
- Important subproduct: A methodology to compute certain answers to monotone queries

Specifying Minimal Instances

Example: Domain: $\mathcal{D} = \{a, b, c, \ldots\}$ Global system $\mathcal{G}_2$

$S_1(X, Y) \leftarrow P(X, Z), R(Z, Y)$ $\quad s_1 = \{(a, b)\}$ open
$S_2(X, Y) \leftarrow P(X, Y)$ $\quad s_2 = \{(a, c)\}$ open

$MinInst(\mathcal{G}_2) = \{\{P(a, c), P(a, z), R(z, b)\} \mid z \in \mathcal{D}\}$

Specification of minimal instances: $\Pi(\mathcal{G}_2)$

- $P(X, Z) \leftarrow S_1(X, Y), F((X, Y), Z)$
- $P(X, Y) \leftarrow S_2(X, Y)$
- $R(Z, Y) \leftarrow S_1(X, Y), F((X, Y), Z)$

The first and third rules come from the first mapping; the Z in the head is "shared" and must be bounded in value; this is captured through the "functional predicate" F that has to be specified now as (a) picking up values for Z for tuples (X, Y), and (b) satisfying the functional requirement $X, Y \rightarrow Z$

- $F((X,Y),Z) \leftarrow S_1(X,Y), dom(Z), choice((X,Y),(Z))$
- $dom(a).,\quad dom(b).,\quad dom(c).,\quad \ldots, S_1(a,b).,\quad S_2(a,c).$

Inspired by inverse rules algorithm for computing certain answers (Duschka, Genesereth, Levy; JLP 00)

Now global relations are defined in terms of the local relations

F is a functional predicate, whose functionality on the second argument is imposed by the choice operator

$choice((X,Y)),(Z))$: non-deterministically chooses a unique value for Z for each combination of values for X, Y
(Giannotti, Pedreschi, Sacca, Zaniolo; DOOD 91)

Models of $\Pi(\mathcal{G}_2)$ are the choice models, but the program can be transformed into one with stable models semantics

That is, the "operational" semantics of $choice$ can be replaced by a declarative semantics by using "standard" program rules (c.f. page 226)

$$M_b = \{dom(a), \ldots, S_1(a,b), S_2(a,c), \underline{P(a,c)}, choice((a,b),b), F(a,b,b), \underline{R(b,b)}, \underline{P(a,b)}\}$$

$$M_a = \{dom(a), \ldots, S_1(a,b), S_2(a,c), \underline{P(a,c)}, choice(a,b,a), F((a,b),a), \underline{R(a,b)}, \underline{P(a,a)}\}$$

$$M_c = \{dom(a), \ldots, S_1(a,b), S_2(a,c), \underline{P(a,c)}, choice((a,b),c), F(a,b,c), \underline{R(c,b)}\}$$

...

Here: 1-1 correspondence between stable models and minimal instances of $\mathcal{G}_2$

In general:

- The minimal instances are all among the stable models of the program
- All the models of the program are (determine) legal instances
- In consequence, the program can be used to compute all the certain answers to monotone queries
- The program can be refined to compute all and only the minimal legal instances (c.f. example on page 226; also appendix 1)
- The program can also be used to compute certain answers to monotone queries; more general than any other algorithm for LAV
- Specification programs can be produced when there are also closed or clopen sources (appendix 2)

Repairs and Consistent Answers

Intuitively, consistent answers are invariant under minimal restorations of consistency

Definition based on notion of repair
(Arenas, Bertossi, Chomicki; PODS 99)

A repair is a global instance that minimally differs from a minimal legal instance

Example: Global system $\mathcal{G}_1$ (extended)

$$S_1(X,Y) \leftarrow R(X,Y) \quad \text{with} \quad s_1 = \{(a,b),(c,d)\} \quad \text{open}$$
$$S_2(X,Y) \leftarrow R(Y,X) \quad \text{with} \quad s_2 = \{(c,a),(e,d)\} \quad \text{open}$$
$$S_3(X) \leftarrow P(X) \quad \text{with} \quad s_3 = \{(a),(d)\} \quad \text{open}$$

$MiniInst(\mathcal{G}_1) = \{\{R(a,b), R(c,d), R(a,c), R(d,e), P(a), P(d)\}\}$

$\mathcal{G}_1$ is inconsistent wrt $FD: \quad X \rightarrow Y$

$Repairs^{FD}(\mathcal{G}_1)$:

- $D^1 = \{R(a,b), R(c,d), R(d,e), P(a), P(d)\}$
- $D^2 = \{R(c,d), R(a,c), R(d,e), P(a), P(d)\}$

(we relax legality for repairs)

Queries:

- $Q(X,Y): \quad R(X,Y)?$

  $(c,d), (d,e)$ are the consistent answers
- $Q_1(X): \quad \exists Y \ R(X,Y)?$

  a is a consistent answer, together with c, d

Specification of Repairs

So far: specification of minimal instances of an integration system; they can be inconsistent

Now: Specify their repairs

Idea: Combine the program that specifies the minimal instances with the "repair program" that specifies the repairs of each minimal instance

Repairs of single databases are specified using disjunctive logic programs with stable model semantics and are used to compute consistent answers to queries (Barcelo, Bertossi; PADL03)

Example: $\mathcal{G}_3$ $\quad S_1(X) \leftarrow P(X,Y) \quad \{s_1(a)\}$ open

$S_2(X,Y) \leftarrow P(X,Y) \quad \{s_2(a,c)\}$ open

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

$MinInst(\mathcal{G}_3) = \{\{P(a,c)\}\}$ ... inconsistent system

In the program, the $P(\cdot,\cdot,\mathbf{t_d})$ are the output of the first layer, that specifies the minimal instances

They are taken by the second layer specifying the repairs, whose output are the $P(\cdot,\cdot,\mathbf{t}^{\star\star})$

A third layer can be the query program, that uses the $P(\cdot,\cdot,\mathbf{t}^{\star\star})$

Repair Program:

First Layer is refined program for minimal instances (with standard version of $choice$, as used by DLV)

$$
\begin{array}{rcl}
dom(a).\ \ dom(c). & & S_1(a).\ \ S_2(a,c). \\
P(X,Y,\mathbf{t_d}) & \leftarrow & P(X,Y,s_1) \\
P(X,Y,\mathbf{t_d}) & \leftarrow & P(X,Y,t_o) \\
P(X,Y,ns_1) & \leftarrow & P(X,Y,t_o) \\
addS_1(X) & \leftarrow & S_1(X),\ \ not\ \ auxS_1(X) \\
auxS_1(X) & \leftarrow & P(X,Z,ns_1) \\
fz(X,Z) & \leftarrow & addS_1(X), dom(Z), chosens1z(X,Z) \\
chosens1z(X,Z) & \leftarrow & addS_1(X), dom(Z),\ \ not\ \ diffchoices1z(X,Z) \\
diffchoices1z(X,Z) & \leftarrow & chosens1z(X,U), dom(Z), U \neq Z \\
P(X,Z,s_1) & \leftarrow & addS_1(X), fz(X,Z) \\
P(X,Y,t_o) & \leftarrow & S_2(X,Y)
\end{array}
$$

Second Layer computes the repairs

$$
\begin{array}{rcll}
P(X,Y,t^\star) & \leftarrow & P(X,Y,\mathbf{t_d}) & \\
P(X,Y,t^\star) & \leftarrow & P(X,Y,t_a) & \\
P(X,Y,f_a) & \vee & P(Y,X,t_a) \leftarrow & P(X,Y,t^\star), \\
 & & & not\ P(Y,X,\mathbf{t_d}) \\
P(X,Y,f_a) & \vee & P(Y,X,t_a) \leftarrow & P(X,Y,t^\star), P(Y,X,f_a) \\
P(X,Y,\mathbf{t}^{\star\star}) & \leftarrow & P(X,Y,t_a) & \\
P(X,Y,\mathbf{t}^{\star\star}) & \leftarrow & P(X,Y,\mathbf{t_d}),\ not\ P(X,Y,f_a) & \\
 & \leftarrow & P(X,Y,t_a), P(X,Y,f_a). &
\end{array}
$$

Disjunctive rules are crucial; they repair: If a violation of IC occurs (c.f. body), then either delete or insert tuples (c.f. head)

Stable models obtained with DLV: (parts of them)

$\mathcal{M}_1^r =$ {dom(a), dom(c), S1(a), S2(a,c), P(a,c,ns1), P(a,c,s2), P(a,c,td), P(a,c,t*), auxS1(a), P(c,a,ta), P(a,c,t**), P(c,a,t*), P(c,a,t**)}
$\equiv \{P(a,c), P(c,a)\}$

$\mathcal{M}_2^r =$ {dom(a),dom(c), S1(a), S2(a,c), P(a,c,ns1), P(a,c,s2), P(a,c,td), P(a,c,t*), auxS1(a), P(a,c,fa)} $\equiv \emptyset$

Repair programs specify exactly the repairs of an integration system for universal and simple (non cyclic) referential ICs

Computing Consistent Answers

Computing consistent answers $\bar{t}$ to a query $Q(\bar{x})$ posed to a VDIS? (Bravo, Bertossi; IJCAI 03)

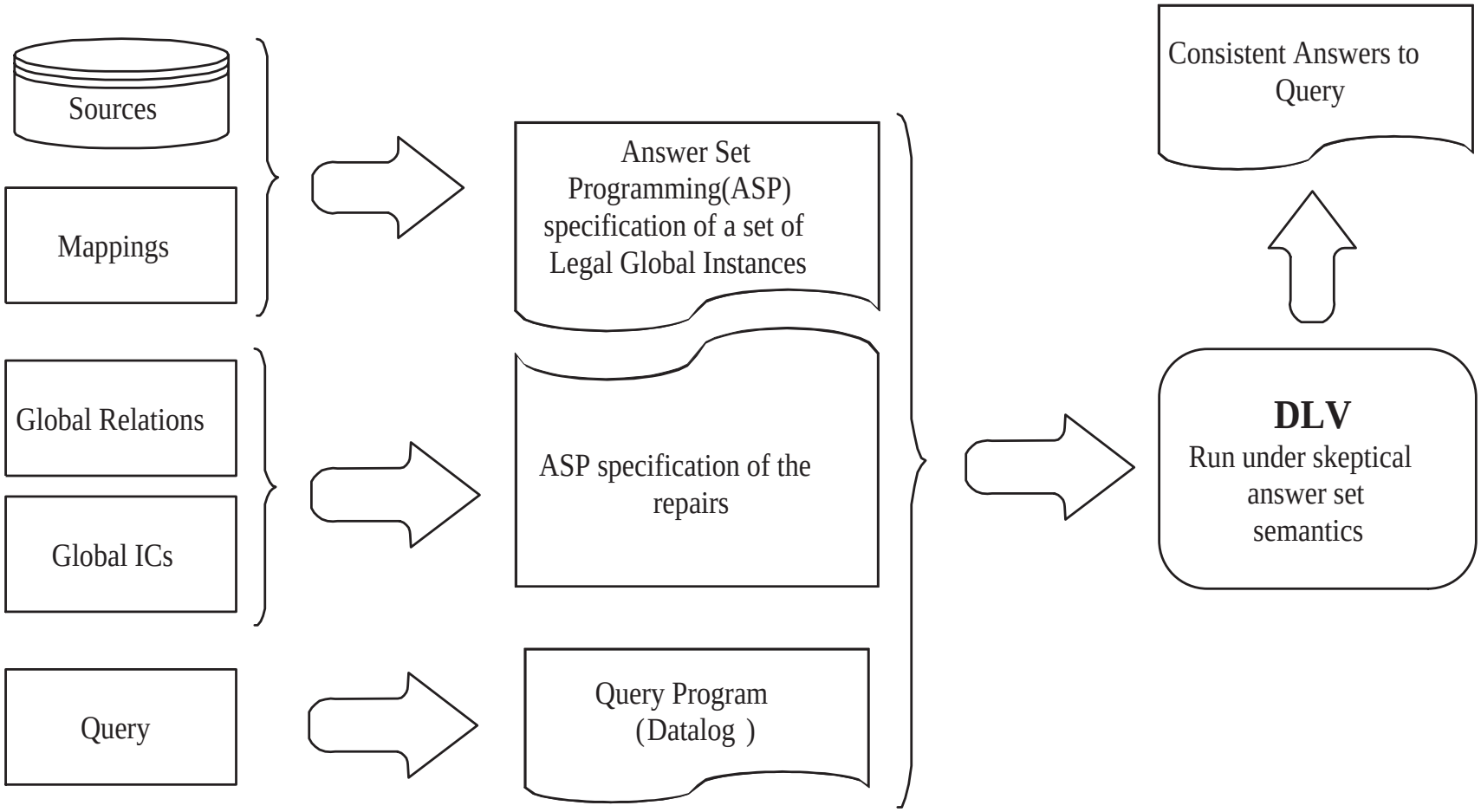
$$Repairs^{IC}(\mathcal{G}) := \text{set of repairs of } \mathcal{G} \text{ wrt } IC$$

A tuple $\bar{t}$ is a **consistent answer** to query Q wrt IC if for every $D \in Repairs^{IC}(\mathcal{G})$: $D \models Q(\bar{t})$

Methodology: (we already know how to do first two items)

- Logic programming specification of minimal instances of a virtual data integration system $\Pi(\mathcal{G})$
- Logic programming specification of repairs of minimal instances (we saw how to do this, e.g. programs with annotation constants) $\Pi(\mathcal{G}, IC)$

- Global query in Datalog (or its extensions) to be answered consistently $\Pi(Q)$
- Run combined program $\Pi(\mathcal{G}, IC) \cup \Pi(Q)$ above under skeptical stable model semantics
- Methodology works for first-order queries (and Datalog extensions), and universal ICs combined with referential ICs
- Important subproduct: A methodology to compute certain answers to monotone queries


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

Example: $\mathcal{G}_3$ Query Q: $P(x, y)$

1. Q': $P(x, y, \mathbf{t}^{\star\star})$

2. $\Pi(Q')$: $Ans(X, Y) \leftarrow P(X, Y, \mathbf{t}^{\star\star})$

3. $\Pi(\mathcal{G}_3, IC)$ as before; form $\Pi = \Pi(\mathcal{G}_3, IC) \cup \Pi(Q')$

4. Repairs corresponding to the stable models of the program Π become extended with query atoms

$$\overline{\mathcal{M}}_1^r = \mathcal{M}_1^r \cup \{Ans(a, c),\ Ans(c, a)\};$$
$$\overline{\mathcal{M}}_2^r = \mathcal{M}_2^r$$

5. No Ans atoms in common, then query has no consistent answers (as expected)

Example: Repair program for $\mathcal{G}_1$ with the FD

```
domd(a).    domd(b).    domd(c).                    %begin subprogram for minimal instances
domd(d).    domd(e).    v1(a,b).
v1(c,d).    v2(c,a).    v2(e,d).

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

R(X,Y,ts) :-  R(X,Y,ta), domd(X), domd(Y).          %begin repair subprogram
R(X,Y,ts) :-  R(X,Y,td), domd(X), domd(Y).
R(X,Y,fs) :- domd(X), domd(Y), not R(X,Y,td).
R(X,Y,fs) :-  R(X,Y,fa), domd(X), domd(Y).

R(X,Y,fa) v R(X,Z,fa) :-  R(X,Y,ts), R(X,Z,ts), Y!=Z, domd(X),domd(Y),domd(Z).

R(X,Y,tss) :- R(X,Y,ta), domd(X), domd(Y).
R(X,Y,tss) :- R(X,Y,td), domd(X), domd(Y), not R(X,Y,fa).
:- R(X,Y,fa), R(X,Y,ta).

Ans(X,Y) :-  R(X,Y,tss).                            %query subprogram
```

The consistent answers obtained for the query Q: $R(X, Y)$, correspond to the expected ones, i.e., $\{(c, d), (d, e)\}$

Appendix 1: The Refined Program

Example: $\mathcal{G}_3$

$$
\begin{array}{lcll}
S_1(X) & \leftarrow & P(X,Y) & \{s_1(a)\} \quad \text{open} \\
S_2(X,Y) & \leftarrow & P(X,Y) & \{s_2(a,c)\} \quad \text{open}
\end{array}
$$

$MinInst(\mathcal{G}_3) = \{\{P(a,c)\}\}$

However, the legal global instances corresponding to stable models of $\Pi(\mathcal{G}_3)$ are of the form $\{\{P(a,c), P(a,z)\} \mid z \in \mathcal{D}\}$

More legal instances (or stable models) than minimal instances

As S_2 is open, it forces $P(a,c)$ to be in all legal instances, which makes the same condition on S_1 automatically satisfied (no other values for Y needed)

Choice operator, as used above, may still choose other values $z \in \mathcal{D}$

We want $\Pi(\mathcal{G})$ to capture only the minimal instances

A refined version of $\Pi(\mathcal{G})$ detects in which cases it is necessary to use the function predicates

$$F(X,Y) \leftarrow add\_S_1(X), dom(X), choice((X),Y)$$

where $add\_S_1(X)$ is true only when the openness of S_1 is not satisfied through other views (which has to be specified)

$$\text{stable models of } \Pi(\mathcal{G}) \equiv MinInst(\mathcal{G})$$

This program not only specifies the minimal instances, but can be also used to compute certain answers to monotone queries

More general than any other algorithm for LAV ...

Specification programs can be produced for case where sources may be closed or clopen

Appendix 2: Minimal Instances for Mixed Sources

Mixed means that some sources may be open (sound, incomplete), others closed (complete), and others clopen (closed and open, exact)

Definitions are similar ...

Example: Global system $\mathcal{G}$ with source definitions and sources extensions with labels

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

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

The global relations can be materialized in different ways, still satisfying the source descriptions, so different global instances are possible

A global (material) instance D is legal if the view definitions applied to it compute extensions $S_1(D), S_2(D)$ such that $s_1 \subseteq S_1(D)$ and $s_2 \supseteq S_2(D)$

(For exact, it would be required $s_3 = S_3$)

$D = \{R(a, b), R(e, d)\}$ is the only legal instance

The repair programs can be extended with program denial constrains to specify the minimal instances of the integration system with mixed sources

Example: $\mathcal{D} = \{a, b, c, \dots\}$ $\mathcal{G}_4$

$$S_1(X, Z) \leftarrow P(X, Y), R(Y, Z) \qquad \{s_1(a, b)\} \quad \text{open}$$
$$S_2(X, Y) \leftarrow P(X, Y) \qquad \{s_2(a, c)\} \quad \text{clopen}$$

Like $\mathcal{G}_2$, that had the same sources but open; before:

$$MinInst(\mathcal{G}_2) = \{\{P(a, c), P(a, z), R(z, b)\} \mid z \in \{a, b, c, ...\}\}$$

Now second source restricts P to (a, c), so in this case:

$$MinInst(\mathcal{G}_4) = \{\{P(a, c), R(c, b)\}\}$$

Closure condition restricts the tuples in the legal instances, but does not add new tuples

In general, $\Pi(\mathcal{G})^{mix}$ built as follows:

- The same clauses as $\Pi(\mathcal{G})$ considering the open and clopen sources as only open
- For every source (view) predicate S of a clopen or closed source with description $S(\bar{X}) \leftarrow P_1(\bar{X}_1), \ldots, P_n(\bar{X}_n)$, add the program denial constraint:

  $$\leftarrow P_1(\bar{X}_1), \ldots, P_n(\bar{X}_n),\ not\ S(\bar{X}).$$

  It says it is not possible for a model to satisfy the conjunction at the RHS of the arrow; then it filters out models of the program

  It captures closure condition on the source that for legal D: $S(D) = P_1^D(\bar{X}_1), \ldots, P_n^D(\bar{X}_n) \subseteq s =$ set of facts for S in $\Pi(\mathcal{G})$

If the simple version of $\Pi(\mathcal{G})$ is considered, it holds:

$$MinInst(\mathcal{G}) \subseteq \text{ stable models of } \Pi(\mathcal{G})^{mix} \subseteq LegInst(\mathcal{G})$$

If the refined version of $\Pi(\mathcal{G})$ is used we have:

$$\text{stable models of } \Pi(\mathcal{G})^{mix} \equiv MinInst(\mathcal{G})$$

Example: (continued) $\mathcal{D} = \{a, b, c, \dots\}$ $\mathcal{G}_4$

$$S_1(X,Z) \leftarrow P(X,Y), R(Y,Z) \qquad \{s_1(a,b)\} \quad \text{open}$$
$$S_2(X,Y) \leftarrow P(X,Y) \qquad \{s_2(a,c)\} \quad \text{clopen}$$

$\Pi(\mathcal{G}_4)^{mix}$: (simple version, refined not needed here)

$$dom(a)., \; dom(b)., \; dom(c)., \; \dots, S_1(a,b)., \; S_2(a,c).$$
$$P(X,Z) \leftarrow S_1(X,Y), F((X,Y),Z)$$
$$R(Z,Y) \leftarrow S_1(X,Y), F((X,Y),Z)$$
$$P(X,Y) \leftarrow S_2(X,Y)$$
$$F((X,Y),Z) \leftarrow S_1(X,Y), dom(Z), choice((X,Y),(Z))$$
$$\leftarrow P(X,Y), \; not \; S_2(X,Y)$$

(S_2 stores the source contents, and P is used to compute the view extension, so we are requiring that $P \subseteq S_2$)

The only stable model of $\Pi(\mathcal{G}_4)^{mix}$ is:

$\{domd(a), \ldots, s1(a,b), s2(a,c), \underline{P(a,c)}, diffchoice(a,b,a),$
$diffchoice(a,b,b), chosen(a,b,c), f(a,b,c), \underline{R(c,b)}\}$

Corresponding, as expected, to the fact that

$MinInst(\mathcal{G}_4) = \{\{P(a,c), R(c,b)\}\}$

Relation between answers obtained from different types of queries and programs (the same applies to the mixed case):

$\Pi(\mathcal{G})$	Query	$Certain_{\mathcal{G}}(Q)$	$Minimal_{\mathcal{G}}(Q)$
Refined	Monotone	$=$	$=$
	General	$\neq$	$=$
Simple	Monotone	$=$	$=$
	General	$\neq$	$\neq$

Conclusions

- Alternative semantics for query answering in GAV virtual data integration systems under ICs have been introduced and studied (Cali, Calvanese, De Giacomo, Lenzerini; CAISE02) (Lembo, Lenzerini, Rosati; KRDB02), (Cali, Lembo,Rosati; IJCAI 03)

- Recent experiments with consistent query answering in virtual data integration systems are encouraging
  - INFOMIX Project

- Much more experimentation with and implementation of query answering in virtual data integration under ICs is necessary

- Many extensions are still necessary on semantics and methodology

- There are clear connections between query answering in VDISs and query answering in peer-to-peer data exchange systems
  - Peers exchange data at query answering time according to certain data exchange constraints or data exchange mappings
  - No central data repository; no centralized management; data resides at peers' sites ...
    - (Halevy, Ives, Suciu, Tatarinov; ICDE 03)
    - (Bertossi, Bravo; P2P&DB 04)
    - (Calvanese, De Giacomo, Lenzerini, Rosati; PODS 04)

9. Query Answering in Peer-to-Peer Data Exchange

The Context

Consider a system consisting of peers who exchange data when they answer local queries

Each peer has a local and autonomous database

Different peer's databases may have a different (relational) schemas

Data at two different peers' sites may be related by data exchange constraints (DECs)

Each peer has a set of DECs expressed as first order formulas that relate its schema with those of some other peers

Each peer does not update its instance according to its DECs and other peers' instances

However, if a peer P is answering a (local) query $Q_{\mathtt{P}}$, it may, at query time

- Import data from other peers to complement its data
- Ignore part of its own data

All this depending upon its own DECs and the peers' instances

But also upon the trust relationships that P has with other peers

An additional element to consider is P's local semantic constraints

Example 1: Peers, their schemas, instances, DECs, trust relationships:

- $\mathtt{P1}, \quad \mathcal{R}_1 = \{R^1\}, \quad r(\mathtt{P1}) = \{R^1(a, b), R^1(s, t)\}$

  $\Sigma(\mathtt{P1}, \mathtt{P2})\colon \ \forall x \forall y (R^2(x, y) \rightarrow R^1(x, y))$

  $\Sigma(\mathtt{P1}, \mathtt{P3})\colon \ \forall x \forall y \forall z (R^1(x, y) \wedge R^3(x, z) \quad \rightarrow \quad y = z)$

  P1 trusts P2 more than itself

  P1 trusts P3 the same as itself

  (also some local ICs $IC(\mathtt{P1})$)

Those answers returned by P to Q_{P} that give an account of all these extra elements are P's peer consistent answers (PCAs) to Q_{P}

In this presentation:

- We make these intuitions precise
- We give a semantics to PCAs
- Specification can be used as the basis for computing them
- We establish connections to virtual data integration

The Solutions for a Peer

Assume the peers's schemas are disjoint, with the possible exception of a shared domain

The union of all peers' instances $r(\mathtt{P})$ can be seen as a single *global* instance $\overline{r}$

A solution for a peer P is a global instance that respects P's DECs and trust relationships with its *immediate neighbors* and stays "as close as possible" to $\overline{r}$

P's peer consistent answers (PCA) are those answers that can be retrieved from P's portion of data in *every* solution for P

The notion of solution can be captured by means of the notion of repair used to characterize the notion of *consistent answer* to a query in a database r that fails to satisfy given ICs (Arenas, Bertossi, Chomicki; PODS'99)

A repair satisfies the originally violated ICs and minimizes the sets of tuples by which it departs from r

A solution may virtually change P's data

Solutions -as repairs in consistent query answering (CQA)- are virtual and used as an auxiliary tool to semantically characterize the notion of PCA

Then correct, intended answer to queries posed to a peer are captured by appealing to alternative models

Emphasis is not on computing repairs, but on defining and computing PCAs

Ideally, P should be able to obtain its PCAs just by querying its and its neighbors' instances

We are first dealing with the *direct case*, that considers the immediate neighbors; the *transitive case* is examined later

Example 1: (cont.)

- P1, $\mathcal{R}_1 = \{R^1\}$, $r(\mathtt{P1}) = \{R^1(a,b), R^1(s,t)\}$

  $\Sigma(\mathtt{P1},\mathtt{P2})$: $\forall x \forall y (R^2(x,y) \rightarrow R^1(x,y))$

  $\Sigma(\mathtt{P1},\mathtt{P3})$: $\forall x \forall y \forall z (R^1(x,y) \wedge R^3(x,z) \rightarrow y = z)$

  P1 trusts P2 more than itself

  P1 trusts P3 the same as itself
- P2, $\mathcal{R}_2 = \{R^2\}$, $r(\mathtt{P2}) = \{R^2(c,d), R^2(a,e)\}$
- P3, $\mathcal{R}_3 = \{R^3\}$, $r(\mathtt{P3}) = \{R^3(a,f), R^3(s,u)\}$

Global instance does not satisfy the DECs

$\bar{r} = \{R^1(a,b), R^1(s,t), R^2(c,d), R^2(a,e), R^3(a,f), R^3(s,u)\}$

P does not change its or other peers' data

Rather P solves its conflicts at query time, when it queries its own and other peers' databases

Obtained answers should be sanctioned as correct wrt to the "solution based semantics"

The solutions for P1 are obtained by:

1. First repairing $\bar{r}$ wrt $\Sigma(\mathtt{P1}, \mathtt{P2})$
   Changing P1's data only (less trustable that P2)
   Only one repair is obtained:

$$\bar{r}_1 = \{R^1(a,b), R^1(s,t), R^1(c,d), R^1(a,e), R^2(c,d), R^2(a,e), \\ R^3(a,f), R^3(s,u)\}$$

2. This repair has to be repaired on its own wrt $\Sigma(\mathtt{P1}, \mathtt{P3})$
   Keeping $\Sigma(\mathtt{P1}, \mathtt{P2})$ satisfied
   Now, data in P1 or P3 can change (equally trustable)

   Two repairs are obtained; and then solutions:

$$\bar{r}' = \{R^1(a,b), R^1(s,t), R^1(c,d), R^1(a,e), R^2(c,d), R^2(a,e)\}$$

$$\bar{r}'' = \{\ R^1(a,b), R^1(c,d), R^1(a,e), R^2(c,d), R^2(a,e), R^3(s,u)\}$$

There is a precise model theoretic definition of solution that corresponds to this process

It involves a minimization with fixed predicates as found in non-monotonic reasoning

Actually these two layered process can be merged into a single one

Definition: Given a FO query $Q(\bar{x}) \in \mathcal{L}(\mathtt{P})$, posed to peer P, a ground tuple $\bar{t}$ is a peer consistent answer for P iff $\bar{r}'|\mathtt{P} \models Q(\bar{t})$ for every solution $\bar{r}'$ for P

Example 1: (cont.) The query $Q : R^1(x, y)$ posed to P1 (in the language of P1) has the PCAs: $(a, b), (c, d), (a, e)$

This answer has values that did not exists in P1's instance

Data originally in P1 is now missing in the set of PCAs

Computation of PCAs

In example 1, the PCAs can be obtained by a FO rewriting of Q using first $\Sigma(\mathtt{P1}, \mathtt{P2})$ and then $\Sigma(\mathtt{P1}, \mathtt{P3})$

$$Q'': \quad [R^1(x,y) \wedge \forall z_1((R^3(x,z_1) \wedge \neg\exists z_2 R^2(x,z_2)) \rightarrow z_1 = y)] \quad \vee \quad R^2(x,y)$$

I.e. P1 first issues a query to P2 to retrieve the tuples in R^2

Next, a query is sent to P3 to discard tuples from R^1 with the same first but not the same second argument in R^3
(as long as there does not exist a tuple in R^2 that "protects" the tuple in R^1)

Rewritten query gives exactly the PCAs to Q

This FO query rewriting approach cannot be extended much

It inherits the limitations of FO query rewriting for CQA

Better look for alternative methodologies

A general approach: answer set programming based specification of a peer's solutions ...

Mixed Referential DECs

In most applications we may expect the DECs $\Sigma(\mathtt{P}, \mathtt{Q})$ for peer $\mathtt{P}$ to consist of formulas of the form

$$\forall \bar{x} \exists \bar{y} (R^{\mathtt{Q}}(\bar{x}) \wedge \varphi \quad \rightarrow R^{\mathtt{P}}(\bar{z}, \bar{y}) \wedge \psi)$$

with $R^{\mathtt{Q}}, R^{\mathtt{P}}$ relations for peers $\mathtt{Q}$ and $\mathtt{P}$, resp., φ, ψ formulas in terms of built-ins, $\bar{z} \subseteq \bar{x}$

Peer $\mathtt{P}$ wants to import data from the more trusted peer $\mathtt{Q}$

The same kind of formula could belong to $\mathtt{Q}$, if $\mathtt{Q}$ wants to validate its own data against $\mathtt{P}$'s data

We may have even more involved cases of referential DECs

Mixing tables from the two peers on each side of the implication

Example 2: Peers: P with schema $\{R_1(\cdot,\cdot), R_2(\cdot,\cdot)\}$
Q with schema $\{S_1(\cdot,\cdot),\ S_2(\cdot,\cdot)\}$

P's DEC:

$$\forall x \forall y \forall z \exists w (R_1(x,y) \wedge S_1(z,y) \ \rightarrow \ R_2(x,w) \wedge S_2(z,w))$$

Assume P considers Q's data more reliable than its own

(the case where P and Q are equally trustable according to P can be handled similarly)

If P's DEC is not satisfied by the combination of the data in P and Q, alternative solutions for P have to be found, keeping Q's data fixed in the process

This is the case, when it holds: $R_1(d, m), S_1(a, m)$, but for no t both $R_2(d, t)$ and $S_2(a, t)$

Obtaining PCAs for P amounts to virtually restoring the satisfaction of P's DEC by virtually modifying P's data

In order to specify P's (virtually) modified relations, introduce virtual versions R_1', R_2' of R_1, R_2

P's queries will be expressed in terms of relations R_1', R_2' only (plus built-ins)

Contents for R_1', R_2' are obtained from the material sources R_1, R_2, S_1, S_2

Since S_1, S_2 are fixed, the satisfaction of P's DEC requires R_1' to be a subset of R_1, and R_2', a superset of R_2

Specification of R_1', R_2' is done by means of a disjunctive extended logic program Π with answer set semantics

First rules:

$$R_1'(x,y) \leftarrow R_1(x,y),\ not\ \neg R_1'(x,y) \quad (1)$$
$$R_2'(x,y) \leftarrow R_2(x,y),\ not\ \neg R_2'(x,y) \quad (2)$$

i.e. by default, the tuples in the sources are copied into the virtual versions, with some exceptions ...

Some of the exceptions for R_1':

$$\neg R_1'(x,y) \leftarrow R_1(x,y), S_1(z,y),\ not\ aux_1(x,z),\ not\ aux_2(z) \quad (3)$$

$$aux_1(x,z) \leftarrow R_2(x,w), S_2(z,w) \quad (4)$$

$$aux_2(z) \leftarrow S_2(z,w) \quad (5)$$

I.e. $R_1(x,y)$ is deleted if simultaneously:

- It participates in a violation of DEC
  (captured by the first three literals in (3) plus rule (4))
- There is no way to restore consistency by inserting a tuple into R_2, because there is no possible matching tuple in S_2 for the possibly new tuple in R_2
  (captured by last literal in (3) plus rule (5))

In case there is such a tuple in S_2, we can either delete a tuple from R_1 or insert a tuple into R_2:

$$\neg R_1'(x,y) \vee R_2'(x,w) \leftarrow R_1(x,y), S_1(z,y), \ not\ aux_1(x,z), S_2(z,w), choice((x,z),w) \quad (6)$$

I.e. in case of a violation of DEC, when there is tuple of the form (a,t) in S_2 for the combination of values (d,a), then the *choice operator* non-deterministically chooses a unique value for t, so that the tuple (d,t) is inserted into R_2

(to minimize differences between material and virtual versions)

(as an alternative to deleting (d,m) from R_1)

choice predicate can be replaced by a standard predicate plus extra rules

Modified program has a usual answer set semantics

No exceptions are specified for R_2', which makes sense since R_2' is a superset of R_2

Then, the negative literal in the body of (2) can be eliminated

However, new tuples can be inserted into R_2' (captured by rule (6)

Finally, the program contains as facts the tuples in the material relations R_1, R_2, S_1, S_2

If P equally trusts itself and Q, both P and Qs' relations are flexible when searching for a solution

Since S_1, S_2 may also change, virtual versions for them must be introduced and specified, and the program becomes more involved

Program Π represents in a compact form all the solutions for a peer

PCAs for a peer can be obtained by running a query program expressed in terms of the virtually repaired tables, in combination with program Π

The combined program is run under the *skeptical answer set semantics*

E.g. the query $Q(x,z) : \exists y(R_1(x,y) \wedge R_2(z,y))$ to P is peer consistently answered by running program Π together with

$$Ans_Q(x,z) \leftarrow R'_1(x,y), R'_2(x,y)$$

Only the virtual versions of P's relations appear in the query, but the program will make P import Q's data

Other Considerations

(A) With referential DECs, the *choice operator* may have to choose values from the infinite domain

Several alternatives considered in the literature

The notion of solution in this regard and the class of referential DECs to deal with will have an impact on decidability, complexity, ...

1. Open infinite domain, repairing picking up elements from it

   PCA becomes undecidable with cyclic referential DECs (Cali, Lembo, Rosati; PODS'03)

2. Repair assigning null values which do not propagate through DECs
   (Barcelo, Bertossi, Bravo; 2003)

   It becomes decidable even with cyclic referential DECs

3. Consider an appropriate finite and closed proper superset of the active domains
   (Bravo, Bertossi; IJCAI'03)

4. Introduce fresh constants whenever needed from a separate domain
   (Calvanese, Damaggio,DeGiacomo,Lenzerini,Rosati; DBISP2P'03)

(B) A peer may have local ICs, e.g. a FD

$$\forall x \forall y \forall z (R_1(x, y) \wedge R_1(x, z) \rightarrow y = z)$$

The peer's program that specifies its solutions should take care of them, *at query time*

They can be integrated in our framework by treating them as DECs $\Sigma(\mathtt{P}, \mathtt{P})$ with $(\mathtt{P}, \mathtt{P}, same)$ in the trust relationship

Interaction of Peers' DECs

Peers may be indirectly related by "composition" of DECs, by transitivity ...

Peer `A` gets a query, then gets data from peer `B`, who requests data from peer `C`, ...

`A` may not even know about `C`'s existence ...

There won't be any explicit DECs from `A` to `C`; and we do not want to derive them

We propose that the semantics for such a global exchange system should be given by the "stabilized interaction" of the pair-based solutions

More precisely, by those *global instances that correspond to the stable models of the program that combines the specification programs we had for one peer and its direct neighbors*

In particular, the absence of solutions is reflected in the absence of stable models for the program

Example 2: (cont.)

- P: schema $\{R_1(\cdot,\cdot), R_2(\cdot,\cdot)\}$
  instance $r(\mathtt{P})$ with $r_1 = \{(a,b)\},\ r_2 = \{\}$

  $\Sigma(\mathtt{P},\mathtt{Q})$:

  $$\forall x \forall y \forall z \exists w (R_1(x,y) \wedge S_1(z,y) \ \rightarrow \ R_2(x,w) \wedge S_2(z,w))$$

  P trusts Q more than itself

- Q: schema $\{S_1(\cdot,\cdot), S_2(\cdot,\cdot)\}$
  instance $r(\mathtt{Q})$ with $s_1 = \{\},\ s_2 = \{(c,e),(c,f)\}$

  $\Sigma(\mathtt{Q},\mathtt{C})$: $\forall x \forall y (U(x,y) \rightarrow S_1(x,y))$

  Q trusts C more than itself

- C: schema $\{U(\cdot,\cdot)\}$
  instance $r(\mathtt{C})$ with $u = \{(c,b)\}$

Now Q's relations may also change, actually in this case only S_1, so we also need a virtual version S_1'

Rules (3), (6) are replaced by (7) and (8), resp.

$$\begin{aligned} \neg R_1'(x,y) &\leftarrow R_1(x,y), S_1'(z,y),\ not\ aux_1(x,z), \\ &\quad not\ aux_2(z) \qquad (7) \\ \neg R_1'(x,y) \vee R_2'(x,w) &\leftarrow R_1(x,y), S_1'(z,y),\ not\ aux_1(x,z), \\ &\quad S_2(z,w), choice((x,z),w) \qquad (8) \end{aligned}$$

Combination program consists of (1), (2),(4), (5), (7), (8) plus

$$\begin{aligned} S_1'(x,y) &\leftarrow S_1(x,y),\ not\ \neg S_1'(x,y) \qquad (9) \\ S_1'(x,y) &\leftarrow U(x,y),\ not\ S_1(x,y). \qquad (10) \end{aligned}$$

(9) is a persistence rule for S_1, and (10) enforces $\Sigma(\mathtt{Q},\mathtt{C})$

The solutions obtained from the stable models of the combined program (plus the material sources) are the expected ones:

$$\bar{r}' = \{S_2(c,e), S_2(c,f), U(c,b), S_1'(c,b), R_2'(a,f), R_1'(a,b)\},$$

$$\bar{r}'' = \{S_2(c,e), S_2(c,f), U(c,b), S_1'(c,b)\}$$

$$\bar{r}''' = \{S_2(c,e), S_2(c,f), U(c,b), S_1'(c,b),\ R_2'(a,e), R_1'(a,b)\}$$

P2P Data Exchange and Data Integration

There are clear connections between PCAs and querying virtual data integration systems

The logic programming-based apprach can be seen as *global-as-view* (GAV) approach: relations in the solutions are specified as views over the peers' original schemas

We explore the connection to the *local-as-view* (LAV) approach, where relations in the (local) data sources are expressed as views of virtual global relations

In example 2, we introduce virtual, global versions S_1', S_2' of S_1, S_2

We propose the following specification:

View definitions	label	source
$R_1(x,y) \leftarrow R_1'(x,y)$	$closed$	r_1
$R_2(x,y) \leftarrow R_2'(x,y)$	$open$	r_2
$S_1(x,y) \leftarrow S_1'(x,y)$	$clopen$	s_1
$S_2(x,y) \leftarrow S_2'(x,y)$	$clopen$	s_2

Labels (for the sources) in the second column depend on the kind of DECs & the trust relationships

They indicate that S_1, S_2 do not change; R_1, R_2 do change, by deletion or insertion of tuples, resp.

A query posed to P has to be first reformulated in terms of R_1', R_2'

Its PCAs can be obtained by querying the integration system subject to the global IC:

$$\forall xyz\exists w(R_1'(x,y) \wedge S_1'(z,y) \rightarrow R_2'(x,w) \wedge S_2'(z,w))$$

There are methodologies for obtaining consistent answers to queries posed to virtual data integration systems with open sources

Also for the LAV approach with mixed sources
(Bertossi & Bravo, to appear)

Again this is a specification based on answer set programming of repairs of the virtual system

Some small adjustments required in the P2P scenario

Final Remarks

(A) What we have presented provides *semantics* and *specifications* for peer consistent aswers

In principle it is possible to compute answers from those specifications (and the data available)

As ongoing work, most urgent future work on peer consistent query answering (PCQA):

"Translate" the specifications into concrete algorithms to query the peers' databases and integrate their answers

(B) At the answer set programming level:

- It becomes necessary to derive *specialized specifications*, that are easier to handle and compute for particular classes of DECs and queries

  And from them also *specialized algorithms* for PCQA (c.f. (A))

- The *specifications* themselves have to be optimized (as logic programs)

- *Computations* of/under the answer set semantics have to be optimized

  Avoid extra complexity in cases where complexity of PC-QA is lower that general data complexity of disjunctive answer set programming

  (The latter is not higher that the *general* data complexity of peer consistent query answering)

- The *interaction* between the logic programming system and the data sources has to be optimized
  (Eiter, Fink, G.Greco, Lembo; ICLP'03)

Appendix I: Definition of solution

Definition: (direct case) Given a peer P in a P2P data exchange system and an instance $\bar{r}$ on $\mathcal{R}$, an instance $\bar{r}'$ on $\mathcal{R}$ is a solution for P if $\bar{r}'$ is a repair of $\bar{r}$ wrt to $\Sigma(\mathtt{P}) \cup IC(\mathtt{P})$ that does not change the more trusted relations

More precisely:
(a) $\bar{r}' \models \bigcup\{\Sigma(\mathtt{P}, \mathtt{Q}) \mid (\mathtt{P}, less, \mathtt{Q}) \text{ or } (\mathtt{P}, same, \mathtt{Q}) \in trust\} \cup IC(\mathtt{P})$
(b) $\bar{r}'|P = \bar{r}|P$ for every predicate $P \in \mathcal{R}(\mathtt{Q})$, where Q is a peer with $(\mathtt{P}, less, \mathtt{Q}) \in trust$
(c) $\bar{r}'$ minimally differs from $\bar{r}$ in the sense that $(\bar{r}' \setminus \bar{r}) \cup (\bar{r} \setminus \bar{r}')$ is minimal under set inclusion among those instances that satisfy (a) and (b)

Intuitively, a solution for P repairs the global instance wrt the DECs with peers that P trusts more than or the same as itself, but leaving unchanged the tables that belong to more trusted peers

As a consequence of the definition, tables belonging to peers that are not related to P or are less trustable are not changed

That is, P tries to change its own tables according to what the dependencies to more or equally trusted peers prescribe

Appendix II: Program with three peers, direct case

Example 1: (cont.) R_1, R_3 have to be flexible in the repair process, and we get interacting rules for R_1

The logic program should have the effect of repairing the database

The repair process may need to execute several steps until it stabilizes

We use program rules with annotations as introduced for CQA in the presence of interacting ICs

Annotations are constants that are used in an extra argument introduced in each database relation

$\mathbf{t_d}$: used to annotate the atoms that are in the original database instance

Single repair steps are obtained by deriving the annotations $\mathbf{t_a}$ or $\mathbf{f_a}$ (atoms getting them are advised to be made true, resp. false)

This when each IC is considered in isolation, but there may be interacting ICs, which requires an iterative process; for this we use annotations $\mathbf{t}^\star$, $\mathbf{f}^\star$

E.g. $\mathbf{t_d}$ groups together the annotations $\mathbf{t_d}$ and $\mathbf{t_a}$ for the same atom

Derived annotations are used to propagate changes through several ICs

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

Generic rules; to be found in any repair program with annotations

$$
\begin{aligned}
R_1(X, Y, \mathbf{t}^{\star}) &\leftarrow R_1(X, Y, \mathbf{t_d}). \\
R_1(X, Y, \mathbf{t}^{\star}) &\leftarrow R_1(X, Y, \mathbf{t_a}). \\
R_1(X, Y, \mathbf{f}^{\star}) &\leftarrow R_1(X, Y, \mathbf{f_a}). \\
R_1(X, Y, \mathbf{f}^{\star}) &\leftarrow dom(X), dom(Y), not\ R_1(X, Y, \mathbf{t_d}). \\
R_1(X, Y, \mathbf{t}^{\star\star}) &\leftarrow R_1(X, Y, \mathbf{t_d}), not\ R_1(X, Y, \mathbf{f_a}).
\end{aligned}
$$

$$
\begin{array}{rcl}
R_1(X, Y, \mathbf{t}^{\star\star}) & \leftarrow & R_1(X, Y, \mathbf{t_a}). \\
 & \leftarrow & R_1(X, Y, \mathbf{t_a}), R_1(X, Y, \mathbf{f_a}). \\
R_2(X, Y, \mathbf{t}^{\star}) & \leftarrow & R_2(X, Y, \mathbf{t_d}). \\
R_2(X, Y, \mathbf{t}^{\star}) & \leftarrow & R_2(X, Y, \mathbf{t_a}). \\
R_2(X, Y, \mathbf{f}^{\star}) & \leftarrow & R_2(X, Y, \mathbf{f_a}). \\
R_2(X, Y, \mathbf{f}^{\star}) & \leftarrow & dom(X), dom(Y), not\ R_2(X, Y, \mathbf{t_d}). \\
R_2(X, Y, \mathbf{t}^{\star\star}) & \leftarrow & R_2(X, Y, \mathbf{t_d}), not\ R_2(X, Y, \mathbf{f_a}). \\
R_2(X, Y, \mathbf{t}^{\star\star}) & \leftarrow & R_2(X, Y, \mathbf{t_a}). \\
 & \leftarrow & R_2(X, Y, \mathbf{t_a}), R_2(X, Y, \mathbf{f_a}). \\
R_3(X, Y, \mathbf{t}^{\star}) & \leftarrow & R_3(X, Y, \mathbf{t_d}). \\
R_3(X, Y, \mathbf{t}^{\star}) & \leftarrow & R_3(X, Y, \mathbf{t_a}). \\
R_3(X, Y, \mathbf{f}^{\star}) & \leftarrow & R_3(X, Y, \mathbf{f_a}).
\end{array}
$$

$$
\begin{aligned}
R_3(X, Y, \mathbf{f}^\star) &\leftarrow dom(X), dom(Y), not\ R_3(X, Y, \mathbf{t_d}).\\
R_3(X, Y, \mathbf{t}^{\star\star}) &\leftarrow R_3(X, Y, \mathbf{t_d}), not\ R_3(X, Y, \mathbf{f_a}).\\
R_3(X, Y, \mathbf{t}^{\star\star}) &\leftarrow R_3(X, Y, \mathbf{t_a}).\\
&\leftarrow R_3(X, Y, \mathbf{t_a}), R_3(X, Y, \mathbf{f_a}).
\end{aligned}
$$

Now we have only two specific rules, they express how to repair the databases when a violation of the DECs occurs:

$$
\begin{aligned}
R_1(X, Y, \mathbf{t_a}) &\leftarrow R_2(X, Y, \mathbf{t}^\star), R_1(X, Y, \mathbf{f}^\star).\\
R_1(X, Y, \mathbf{f_a}) \vee R_3(X, Z, \mathbf{f_a}) &\leftarrow R_1(X, Y, \mathbf{t}^\star), R_3(X, Z, \mathbf{t}^\star), Y \neq Z.
\end{aligned}
$$

The first one corresponds to a violation of $\Sigma(\mathtt{P1}, \mathtt{P2})$; the second one, to a violation of $\Sigma(\mathtt{P1}, \mathtt{P3})$

The facts of the program:

$R_1(a, b, \mathbf{t_d})$. $R_1(s, t, \mathbf{t_d})$. $R_2(c, d, \mathbf{t_d})$. $R_2(a, e, \mathbf{t_d}).R_2(t, h, \mathbf{t_d})$.
$R_3(a, f, \mathbf{t_d})$. $R_3(s, u, \mathbf{t_d})$. $R_3(t, u, \mathbf{t_d})$. $dom(a)$. $dom(b)$.
$dom(s)$. $dom(t)$. $dom(c)$. $dom(d)$. $dom(e)$. $dom(f)$. $dom(u)$.
$dom(h)$.

The non domain atoms say that originally
$R_1 = \{(a, b), (s, t)\}$
$R_2 = \{(c, d), (a, e), (t, h)\}$
$R_3 = \{(a, f), (s, u), (t, u)\}$

Here we do not need virtual versions R_1', R_3' for R_1, R_2, because their final contents will be read off from atoms annotated with $\mathbf{t}^{\star\star}$

Appendix III: Comparison with data integration

Methodology for CQA under LAV and mixed sources is based on a three-layered specification of the repairs:

- A first layer specifies the contents of the global relations in the minimal legal instances (to this layer only open and clopen sources contribute)
- A second layer consisting of program denial constraints that prunes the models that violate the closure condition for the closed sources
- A third layer specifying the minimal repairs of the legal instances left by the other layers wrt the global ICs (repairs may violate the source labels)

In the P2P scenario we consider only legal instances that:

- Satisfy the mapping in the table

- In the case of closed sources, include the maximum amount of tuples from them (virtual relations must be kept as close as possible to their original, material versions)

This can be achieved using the same specifications as for the mixed case, *but* considering the closed sources as clopen

They contribute with rules that import their contents into the system (maximizing tuples in the global relation) and denial program constraints

Trust relation also makes a difference: virtual relations must satisfy the original labels (which capture the trust relationships)

Then repairs of legal instances are based only on tuple deletions (insertions) for global relations corresponding to closed

(resp. open) sources

For clopen sources the rules can neither add nor delete tuples

This preference criterion on repairs is similar to the *loosely-sound semantic* for integration of open sources under GAV (Lembo, Lenzerini, Rosati; KRDB'02)

Methodology handles universal and acyclic referential DECs

This is when arbitrary elements from the infinite underlying domain can be picked up to satisfy the DECs

When repairs are done using null values that do not propagate through ICs, then cycles are allowed

The DEC in example 2 is not a typical referential IC, but the repair layer can be adjusted in order to generate the solutions for P

Assume the peers have the following instances:

$r_1 = \{(a, b)\}, s_1 = \{(c, b)\}, r_2 = \{\}$ and $s_2 = \{(c, e), (c, f)\}$

The layer that specifies the preferred legal instances:

$$
\begin{array}{rcl}
R_1'(X, Y, \mathbf{t_d}) & \leftarrow & R_1(X, Y). \\
S_1'(X, Y, \mathbf{t_d}) & \leftarrow & S_1(X, Y). \\
R_2'(X, Y, \mathbf{t_d}) & \leftarrow & R_2(X, Y). \\
S_2'(X, Y, \mathbf{t_d}) & \leftarrow & S_2(X, Y). \\
 & \leftarrow & R_1'(X, Y, \mathbf{t_d}), R_1(X, Y). \\
 & \leftarrow & S_1'(X, Y, \mathbf{t_d}), S_1(X, Y). \\
 & \leftarrow & S_2'(X, Y, \mathbf{t_d}), S_2(X, Y).
\end{array}
$$

The layer that specifies the repairs of the legal instances:

(The annotation constants are used as auxiliary elements in the repairs process)

$$
\begin{array}{lcl}
R_1'(X, Y, \mathbf{t}^{\star\star}) & \leftarrow & R_1'(X, Y, \mathbf{t_d}),\ not\ R_1'(X, Y, \mathbf{f_a}). \\
R_1'(X, Y, \mathbf{t}^{\star\star}) & \leftarrow & R_1'(X, Y, \mathbf{t_a}). \\
 & \leftarrow & R_1'(X, Y, \mathbf{t_a}), R_1'(X, Y, \mathbf{f_a}). \\
S_1'(X, Y, \mathbf{t}^{\star\star}) & \leftarrow & S_1'(X, Y, \mathbf{t_d}),\ not\ S_1'(X, Y, \mathbf{f_a}). \\
S_1'(X, Y, \mathbf{t}^{\star\star}) & \leftarrow & S_1'(X, Y, \mathbf{t_a}). \\
 & \leftarrow & S_1'(X, Y, \mathbf{t_a}), S_1'(X, Y, \mathbf{f_a}). \\
R_2'(X, Y, \mathbf{t}^{\star\star}) & \leftarrow & R_2'(X, Y, \mathbf{t_d}),\ not\ R_2'(X, Y, \mathbf{f_a}).
\end{array}
$$

$$
\begin{array}{rcl}
R'_2(X,Y,\mathbf{t}^{\star\star}) & \leftarrow & R'_2(X,Y,\mathbf{t_a}). \\
 & \leftarrow & R'_2(X,Y,\mathbf{t_a}), R'_2(X,Y,\mathbf{f_a}). \\
S'_2(X,Y,\mathbf{t}^{\star\star}) & \leftarrow & S'_2(X,Y,\mathbf{t_d}),\ not\ S'_2(X,Y,\mathbf{f_a}). \\
S'_2(X,Y,\mathbf{t}^{\star\star}) & \leftarrow & S'_2(X,Y,\mathbf{t_a}). \\
 & \leftarrow & S'_2(X,Y,\mathbf{t_a}), S'_2(X,Y,\mathbf{f_a}). \\
R'_1(X,X,\mathbf{f_a}) & \leftarrow & R'_1(X,Y,\mathbf{t_d}), S'_1(Z,Y,\mathbf{t_d}), \\
 & & not\ aux_1(X,Z),\ not\ aux_2(Z). \\
aux_1(X,Z) & \leftarrow & R'_2(X,U,\mathbf{t_d}), S'_2(Z,U,\mathbf{t_d}). \\
aux_2(Z) & \leftarrow & S'_2(Z,W,\mathbf{t_d}). \\
R'_1(X,Y,\mathbf{f_a}) \vee R'_2(X,W,\mathbf{t_a}) & \leftarrow & R'_1(X,Y,\mathbf{t_d}), S'_1(Z,Y,\mathbf{t_d}), \\
 & & not\ aux_1(X,Z), S'_2(Z,W,\mathbf{t_d}), \\
 & & choice((X,Z),W).
\end{array}
$$

Running this program with DLV, we obtain the following solutions (they can be obtained by selecting only the tuples with annotation $\mathbf{t}^{\star\star}$ from a stable model):

$$\bar{r}^1 = \{S_1'(c,b),\ S_2'(c,e),\ S_2'(c,f),\ R_1'(a,b),\ R_2'(a,f)\}$$

$$\bar{r}^2 = \{S_1'(c,b),\ S_2'(c,e),\ S_2'(c,f)\}$$

$$\bar{r}^3 = \{S_1'(c,b),\ S_2'(c,e),\ S_2'(c,f),\ R_1'(a,b),\ R_2'(a,e)\}$$

$$\bar{r}^4 = \{S_1'(c,b),\ S_2'(c,e),\ S_2'(c,f)\}$$

THE END

Acknowledgments: This presentation is based on original research produced with several coauthors, as reflected in the list of references that follows

References (to own work)

1. Arenas, M., Bertossi, L. and Chomicki, J. "Consistent Query Answers in Inconsistent Databases". In *Proc. 18th ACM Symposium on Principles of Database Systems (PODS 99)*, 1999, pp. 68–79.
2. Arenas, M., Bertossi, L. and Kifer, M. "Applications of Annotated Predicate Calculus to Querying Inconsistent Databases". In *Computational Logic - CL 2000*, Springer LNAI 1861, Springer, 2000, pp. 926 – 941.
3. Celle, A. and Bertossi, L. "Querying Inconsistent Databases: Algorithms and Implementation". In *Computational Logic - CL 2000*, Springer LNCS 1861, 2000, pp. 942 – 956.
4. Barcelo, P. and Bertossi, L. "Repairing Databases with Annotated Predicate Logic". *Proc. NMR'02*, 2002.
5. Bertossi, L., Chomicki, J., Cortes, A. and Gutierrez, C. "Consistent Answers from Integrated Data Sources". In *Flexible Query Answering Systems*, Springer LNAI 2522, 2002, pp. 71–85.
6. Barcelo, P. and Bertossi, L. "Logic Programs for Querying Inconsistent Databases". *Proc. Practical Aspects of Declarative Languages (PADL 2003)*, Springer LNCS 2562, 2003, pp. 208-222.

7. Arenas, M., Bertossi, L., Chomicki,J., He, X., Raghavan, V. and Spinrad, J. "Scalar Aggregation in Inconsistent Databases". *Theoretical Computer Science*, Volume 296, Issue 3, 2003, pp. 405-434.

8. Bertossi, L. and Chomicki, J. "Query Answering in Inconsistent Databases". Chapter in book *Logics for Emerging Applications of Databases*, J. Chomicki, G. Saake and R. van der Meyden (eds.), Springer, 2003.

9. Arenas, M., Bertossi, L. and Chomicki, L. "Answer Sets for Consistent Query Answering in Inconsistent Databases". *Theory and Practice of Logic Programming*, 2003, 3(4-5): 393-424.

10. Barcelo, P., Bertossi, L. and Bravo, L. "Characterizing and Computing Semantically Correct Answers from Databases with Annotated Logic and Answer Sets. In *Semantics of Databases*, Springer LNCS 2582, 2003, pp. 1–27.

11. Bravo, L., Bertossi, L. "Consistent Query Answering under Inclusion Dependencies". *Proc. 14th CASCON Conference*, IBM, Toronto Lab., 2004.

12. Bertossi, L. and Bravo, L. "Consistent Query Answers in Virtual Data Integration Systems". In *Inconsistency Tolerance*, Springer LNCS 3300, 2004, pp. 42-83.

13. Bravo, L. and Bertossi, L. "Disjunctive Deductive Databases for Computing Certain and Consistent Answers to Queries from Mediated Data Integration Systems". *Journal of Applied Logic*, 2005, 3(2):329-367.

14. Bravo, L. and Bertossi, L. "Logic Programs for Consistently Querying Data Integration Systems". In *Proc. International Joint Conference on Artificial Intelligence (IJCAI 03)*, Morgan Kaufmann, 2003, pp. 10–15.

15. Bertossi, L. and Bravo, L. "Query Answering in Peer-to-Peer Data Exchange Systems". In *Current Trends in Database Technology*. Springer LNCS 3268, 2004, pp. 478-485.

16. Bertossi, L., Bravo, L., Franconi, E. and Lopatenko, A. "Complexity and Approximation of Fixing Numerical Attributes in Data-bases Under Integrity Constraints". In *Proc. of the Databases Programming Languages conference (DBPL 2005)*, Springer LNCS, 2005.

17. Caniupan, M. and Bertossi, L. "Optimizing Repair Programs for Consistent Query Answering". To appear in *Proc. International Conference of the Chilean Computer Science Society (SCCC 05)*. IEEE Press.