Evolution and Change in Relational Databases

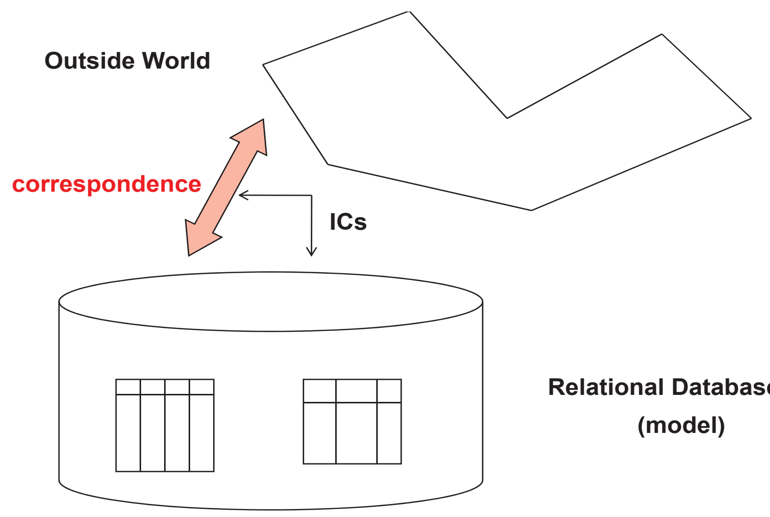
An Overview of Some Issues[1]

Leopoldo Bertossi

Carleton University
School of Computer Science
Ottawa, Canada

[1] Chapter 1 of PhD Course at U. Calabria, Arcavacata, 2013.

1. Databases and Integrity Constraints



A database instance D is a model of an outside reality

An integrity constraint on D is a condition that D is expected to satisfy in order to capture the semantics of the application domain

A set IC of integrity constraints (ICs) helps specify/maintain the correspondence between D and that reality

Several applications:

[Godfrey et al. 98]

- By being satisfied, an IC imposes a restriction on the evolution of the DB under updates

  Most typical application ...

- ICs properly represented become metadata, i.e. data about the data

  They help understand what the data are about, their meaning, their semantics

  Many uses, e.g. inter-operability

  In particular, integration of data sources into a single data repository

- When guaranteed to be satisfied, ICs can be used for semantic query optimization

  Example: Schema: $Employee(Name, Position, Salary)$

  IC: $\forall xy Employee(x, \mathtt{manager}, y) \rightarrow y \geq 100)$

  Equivalently: $\forall xy \neg (Employee(x, \mathtt{manager}, y) \wedge y < 100)$

  (as a denial constraint)

  Query: $\mathcal{Q}(x):\ \exists y Employee(x, \mathtt{manager}, y) \wedge y < 80)$

  Immediately return $\emptyset$ without scanning the table!

  More systematically?

  There is a general methodology ...

The IC can be written in clausal form (with implicitly universally quantified variables):

$$\neg Employee(u, \texttt{manager}, v) \ \vee \ v \geq 100$$

Consider the query (about x, y):

$$Employee(x, \texttt{manager}, y) \wedge y < 80)$$

Resolve the two complementary literals above, obtaining the resolvent, or *residue* (of the IC): $y \geq 100$

When the DB D satisfies the IC and the query (for some values x, y), the residue has to be satisfied by D too

So, it can appended to the original query keeping exactly the same answers

That is, the original query has the same answers as the *rewritten query*:

$$Employee(x, \texttt{manager}, y) \wedge \underbrace{y < 80 \wedge y \geq 100}_{\text{unsatisfiable!}}$$

IC enforcement:

- By the DBMS itself when ICs have been declared with the schema

  Commercial DBMSs provide limited support in this direction

- Through triggers (active, ECA rules) stored by user in DB

  Reject/notify inadmissible updates or compensate updates

  Can be derived from ICs [Widom et al. 95]

- Through application/transaction programs

2. Logical Status of IC Satisfaction

A relational DB D can be seen as a set-theoretic structure

With a domain (or universe) and some (finite) relations defined on it

Manager	Boss	Subordinate
	ken	*john*
	john	*mary*
	peter	*joe*

$D = \langle Dom, Manager^D, \ldots \rangle$, with $Manager^D \subseteq Dom \times Dom$, ...

An IC as a sentence φ in language of first-order (FO) predicate logic associated to the DB schema

$D \models \varphi$: a well defined model-theoretic notion of satisfaction in FO logic, of a sentence by a structure

Alternatively, D can be seen as a theory $Th(D)$ written in FO predicate logic

Reiter's logical reconstruction of relational DBs [Reiter 84]

Manager	Boss	Subordinate
	ken	*john*
	john	*mary*
	peter	*joe*

$\mapsto$ a theory

- Predicate extensions plus closed-world assumption (CWA):

  $\forall xy(Manager(x,y) \ \leftrightarrow \ x = ken \wedge y = john \ \vee \cdots \vee \ x = peter \wedge y = joe)$

- Possibly domain-closure: $\forall x(x = ken \vee \cdots \vee x = mary \vee x = sue)$

- Unique names assumption (UNA): $john \neq mary$, etc.

Now $D \models \varphi$ also makes sense as $Th(D) \vdash \varphi$

ICs have to be entailed by the DB (the latter as a theory) ...

Another way of looking at DBs:

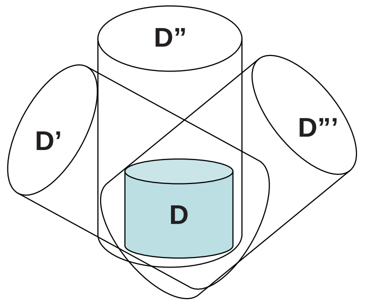
The DB above can be seen as a *set of ground atoms (or tuples)*:

$$D = \{Manager(ken, john), Manager(john, mary), Manager(peter, joe)\}$$

As such it can be seen as a structure or a set of logical formulas

Also possible to consider D to be complete (closed) or *incomplete*, the latter when D may contain more tuples than those on the RHS ($D \supseteqq \{ \ldots \}$)

As incomplete DB, D above can also be seen a *representative of a class* of closed DBs, namely those that extend D



Incomplete DBs have been actively investigated in the last few years (but at least since early 80s) [Imielinski et al. 84] [Greco et al. 12]

- For D seen as a structure (or closed set of ground atoms), maybe $D \not\models \Sigma$
- However, it is seen as an incomplete theory, it may be good enough if $D \cup \Sigma$ is consistent
- D is usually extended through Σ (chase, etc.)

Incarnation of older discussion: [Reiter 92]

- Reiter: ICs are satisfied by the DB
- Kowalski: ICs are to be consistent with DB

Example: $D = \{Manager(ken, john), Employee(ken, accounting)\}$

IC: $\forall xy(Manager(x,y) \rightarrow \exists z(Employee(x,z) \wedge Employee(y,z)))$

As a closed DB, $D \not\models$ IC

As an open DB (theory), D is consistent with IC

D can be extended to make it satisfy IC (as a closed DB): just add the tuple $Employee(john, accounting)$

This can be seen as the process of *chasing* the DB via the IC ...

But also adding $Employee(ken, \lambda_1), Employee(john, \lambda_1)$ would do, with λ_1 a *labeled null*, would do

But λ_1 would become $accounting$ in the additional presence of the FD: $Name \rightarrow Department$)

Example:

$D = \{Manager(ken, john), Manager(sue, john), Employee(ken, accounting)\}$

Is inconsistent as an open DB with the FD: $Subordinate \rightarrow Boss$

Because $D \cup \{FD\}$ is inconsistent

3. Database and View Maintenance

Database maintenance is about keeping the ICs satisfied when the DB undergoes updates

View maintenance is about keeping materialized view extensions synchronized with base tables

A view is just a (usually virtual) relation defined on top of base (usually material) relations

A view defined on table $Manager$ (cf. page 8):

$$\forall x(TopBoss(x) \ \leftrightarrow \ \exists y Manager(x,y) \wedge \neg\exists z Manager(z,x)) \qquad (*)$$

$TopBoss(D) = \{ken, peter\}$ If $Manager(sue, ken) \mapsto D$?

The two problems are related ...

- Associate a violation view V_φ to IC φ

  $$\varphi \text{ is satisfied by } D \quad \text{iff} \quad V_\varphi(D) = \emptyset$$

  Example: $FD\colon\ Subordinate \rightarrow Boss$ on previous schema

  $$V_{FD}(x,y) \leftrightarrow Manager(x,y) \wedge \exists z(Manager(z,y) \wedge x \neq z)$$

  To maintain the IC (satisfied), maintain the violation view (empty)

  BTW, a condition that is commonly used by ECA rules for IC maintenance

- A view definition, e.g. (*), can be seen as an IC expressed in an expanded language (with view predicates)

  The definition has to be kept satisfied (by the DB expanded with extensions for view predicates)

Techniques for each of database and view maintenance can be applied to the other problem

In both cases, incremental techniques are desirable

4. Incremental Maintenance

The issues:

- We do not want to check the full IC every time the DB is updated

Maybe an update on base table is irrelevant to the IC

Maybe only "a portion" of the IC has to be rechecked

- We do not want to recompute the view from scratch using the definition every time base tables are updated

Crucial for materialized views, as in DWHs [Gupta et al. 99]

Maybe the update is irrelevant to the view (definition)

Maybe it is a matter of computing a "delta"

Hopefully without using the whole DB

"Inductive" IC checking: [Nicolas 82]

1. Assume $D \models \varphi$
2. Update D into D' by a set U of updates
3. What portion of φ (if any) has to be checked on D' (or only D, U)?

Example: With $FD: \ Subordinate \to Boss$ above

Assume $D \models FD$ Instead of checking on D':

$$\forall xyz(Manager(x, y) \wedge Manager(z, y) \to x = z)$$

1. If U contains only deletions:

   Do not check anything

2. If U is $insert_{Manager}(a, b)$:

   Check on D: $\exists x(Manager(x, b) \wedge x \neq a)$?

General mechanism that relies on syntactic structure of ICs

View maintenance and relevant updates:

[Blakeley et al. 89]
[Gupta et al. 95]

- Self-maintainable views: View update without accessing base tables, but instead
  - current, material state of the view
  - view definition
  - the actual updates
  - possibly ICs on base tables

No need for the updated underlying base data ...

Example: (the gist) Schema $R(A, B)$ with $FD: \ B \rightarrow A$

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

View (projection of R on B):[2] $V(Y) \leftarrow R(X,Y)$

$V(D) = \{b, d, f\}$

With update $delete_R(a,b)$, using

- the update itself (knowing it has an effect on V)
- the pre-update extension of the view
- the FD (assumed to be satisfied so far)

we obtain right away the new extension: $V(D') = \{d, f\}$

[2]On many occasions we use Datalog notation for queries and views

- Irrelevant Updates: Determine views that are not affected by certain classes of updates on base tables

Ignore those updates for view maintenance

The "irrelevant update problem"

Example: (as above, cont.) For the view $V(X) \leftarrow R(X, Y)$

Updates of the form $change_{R[Y]}(\bar{t}; v)$ (in tuple $\bar{t}$ in R change value for Y into v) are always irrelevant

The irrelevant update problem also appears in IC maintenance: some updates never lead to inconsistency

For example, for FDs tuple deletions are always irrelevant

5. ICs on Views

Having ICs on views could be useful for the tasks above, for monitoring the DB behaviour through the views, query answering using views, metadata for interoperability in general ...

The classic problem of deriving ICs for views from view definitions and ICs on base tables [Klug 80, 82]

Example:

Manager	Boss	Subordinate	Salary
	ken	*john*	100
	john	*mary*	120
	peter	*joe*	150

$FD\colon\ Subordinate \rightarrow Boss$

View $V(x,y)\colon\ \exists z Manager(x,y,z)$

From V's definition and FD, an IC on V: FD^V: $Subordinate \rightarrow Boss$

A violation of FD^V by view extension indicates a violation of FD by underlying DB

$$\begin{tikzcd}
\text{schema } S \arrow[d] & D \arrow[d] & \models & IC \arrow[d] & \text{(ICs for } S) & \\
\text{view } V & V(D) & \models & IC' & \text{(ICs for } V) & \text{for all } D \\
 & & & ? \arrow[u, Rightarrow] & &
\end{tikzcd}$$

Problem: Compute the set IC' of non-trivial ICs on the view predicate V such that: $D \models IC \;\Rightarrow\; V(D) \models IC'$

There are syntactic techniques for deriving the ICs on views from ICs on base tables plus view definitions

- formal, deductive: For FDs [Klug 80]
- tableaux (generic, tabular representations of queries or, better, query answers) plus chase

  For FDs and join constraints [Klug 82]
- ...

Example: (as above, continued)

FD $R(A, B, C) : A \to B$ View: $\forall xy(V(x, y) \leftrightarrow \exists z R(x, y, z))$

The FD and view definition produce three clauses:[3]

$$\neg R(x, y, z) \vee \neg R(x, v, u) \vee y = v \tag{1}$$

$$\neg V(x, y) \vee R(x, y, f(x, y)) \tag{2}$$

((2) is one direction of the view definition)

Resolution of (1) and (2) produce

$$\neg V(x, y) \vee \neg R(x, u, v) \vee y = u \tag{3}$$

Now (2) and (3):

$$\neg V(x, y) \vee \neg V(x, u) \vee y = u$$

... the expected FD on V: $V(A, B) : A \to B$

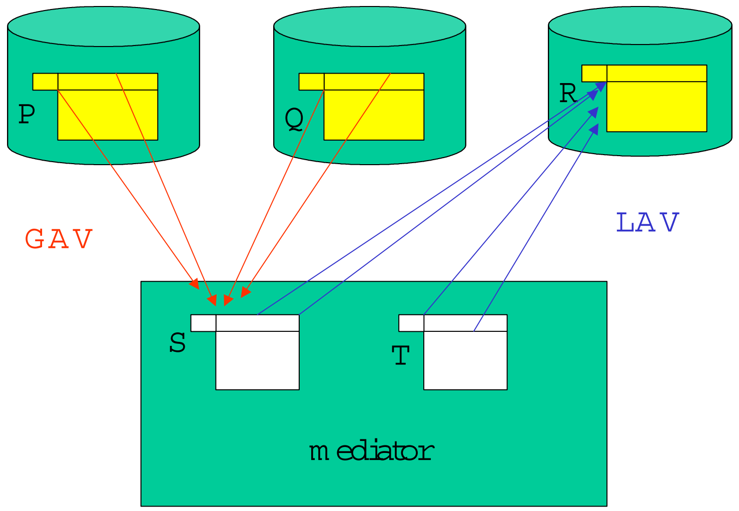
[3] (1) is clausal form of $\forall xyzvu(R(x, y, z) \wedge R(x, v, u) \to y = v)$. In (2), that comes from the view definition, $f(x, y)$ is Skolem term for the existential quantifier

Related problems:

Virtual data integration, under Global-As-View (GAV) or Local-As-View (LAV)

Global ICs imposed directly on the views (global relations)

No guarantee for their satisfaction (data stay at the sources)



- What global ICs hold if certain source and inter-source ICs hold?
- What ICs should be imposed on the sources (or combinations thereof) to enforce global ICs?

Main problem: Global ICs cannot be enforced, but are important for the semantics of the VDIS $\mapsto$ Deal with them at query time?

6. Updates through Views

Another related classic problem in relational DBs

Given:

- Base schema $\mathcal{S}$
- View definition: $\forall \bar{x}(V(\bar{x}) \ \leftrightarrow \ \varphi_{\mathcal{S}}(\bar{x}))$ (*)
- An instance D for $\mathcal{S}$, and extension $V(D)$ for the view

Apply update U on $V(D)$, propagating updates on D, keeping (*) satisfied

Example: $V(x,y) \ \leftrightarrow \ \exists z(Manager(x,y,z) \wedge z = 100)$

Manager	Boss	Subordinate	Salary
	ken	$john$	100
	$john$	$mary$	120
	$peter$	joe	150

Easy!

Less easy with base ICs, e.g. $Manager\colon Boss, Subordinate \rightarrow Salary$

However, for more complex views (defined by more complex queries) ...

Example: $D = \{R(a,b), R(c,d), S(b,c)\}$

$$V_2(x,y) \leftarrow R(x,z), S(z,y)$$

$U: \ insert_{V_2}(a,d)$

$D' = \{R(a,b), R(c,d), S(b,c), R(a, NULL), S(NULL, d)\}$?

Ordinary SQL Nulls? (in joins?)

Arbitrary values from the domain?

Conditional instances? (there could be additional conditions on X below)

$D' = \{R(a,b), R(c,d), S(b,c), R(a,X), R(X,d)\}$

What if also base IC $R: A \rightarrow B$? Should X be b?

Disjunctive views?

$$\begin{array}{lcl} V_3(x,y) & \leftarrow & R(x,y) \\ V_3(x,y) & \leftarrow & S(x,y) \end{array} \qquad U: \ \mathit{insert}_{V_3}(e,d)$$

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

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

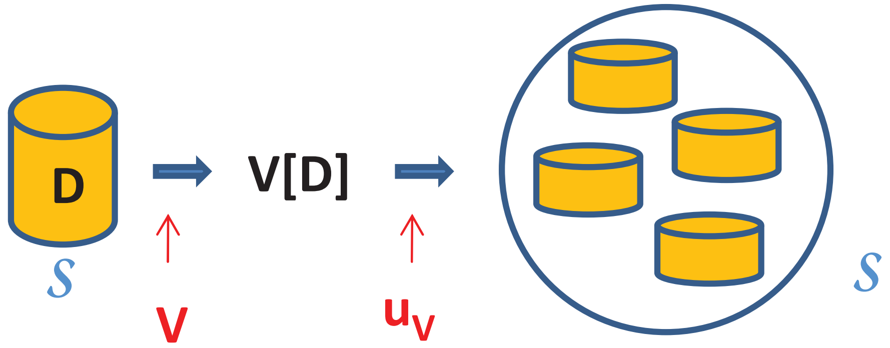
$D' = \{R(a,b), R(c,d), S(b,c), R(e,d), S(e,d)\}$?

Several choices ... Which are the right ones?

It depends on the update semantics (of DBs through views)

It can be a "possible world semantics"

A class of intended (admissible, legal) instances D' that reproduce the view update U



Preference criteria can be imposed on elements of the class

An old and difficult problem in relational databases

Semantics and algorithms based on assumption of availability of a view complement [Bancil. et al. 81; Cosm. et al. 84; ...; Lecht. et al. 03]

No standard solution offered/implemented in commercial DBMS

In practice:

- Some views are considered to be updatable

  Restrictions on views make them updatable or not

- `INSTEAD-OF` triggers

  Instead of direct view update, update base tables (as indicated by trigger), causing the intended change on the view

Exercise: Read/understand this INSTEAD-OF trigger on a view

Suppose that dept and emp are tables that list departments and employees:

```
CREATE TABLE dept (
   deptno INTEGER PRIMARY KEY,
   deptname CHAR(20),
   manager_num INT
);
CREATE TABLE emp (
   empno INTEGER PRIMARY KEY,
   empname CHAR(20),
   deptno INTEGER REFERENCES dept(deptno),
   startdate DATE
);
ALTER TABLE dept ADD CONSTRAINT(FOREIGN KEY (manager_num)
      REFERENCES emp(empno));
```

The next statement defines manager_info, a view of columns in the dept and emp tables that includes all the managers of each department:

```
CREATE VIEW manager_info AS
   SELECT d.deptno, d.deptname, e.empno, e.empname
      FROM emp e, dept d WHERE e.empno = d.manager_num;
```

The following CREATE TRIGGER statement creates manager_info_insert, an INSTEAD OF trigger that is designed to insert rows into the dept and emp tables through the manager_info view:

```
CREATE TRIGGER manager_info_insert
   INSTEAD OF INSERT ON manager_info    --defines trigger event
      REFERENCING NEW AS n              --new manager data
   FOR EACH ROW                         --defines trigger action
      (EXECUTE PROCEDURE instab(n.deptno, n.empno));

CREATE PROCEDURE instab (dno INT, eno INT)
   INSERT INTO dept(deptno, manager_num) VALUES(dno, eno);
   INSERT INTO emp (empno, deptno) VALUES (eno, dno);
END PROCEDURE;
```

After the tables, view, trigger, and SPL routine have been created, the database server treats the following INSERT statement as a triggering event:

```
INSERT INTO manager_info(deptno, empno) VALUES (08, 4232);
```

This triggering INSERT statement is not executed, but this event causes the trigger action to be executed instead, invoking the instab( ) SPL routine. The INSERT statements in the SPL routine insert new values into both the emp and dept base tables of the manager_info view.

http://pic.dhe.ibm.com/infocenter/idshelp/v115/index.jsp?topic=%2Fcom.ibm.sqls.doc%2Fids_sqs_0641.htm

Vast literature on different approaches to update through views

Some use KR-based approaches:

- Abductive: Relationship between views V and base tables T given by view definitions plus base ICs

Observations are the intended view updates ($^{+}_{-}atom_V$)

Abductibles are $^{+}_{-}atom_T$, those that explain (cause) the observations

After "abducing" the $^{+}_{-}atom_T$ from the $^{+}_{-}atom_V$, execute the former to give an account of the latter

Abductive logic programming, including ICs [Kakas et al. 90, 92]

- ASPs: Possible worlds as stable models of a disjunctive ASP specifying how view and base updates are related [LB et al. 13]

Remark: In virtual data integration, the corresponding problem would be updating the sources through the mediator (containing views under GAV)

Not allowed in general, but see [De Giac. et al. 09]

8. Specifying DB Evolution

So far here, we have a logical specification of the DB, but external updates that change the DB

We can integrate everything into a single logical theory that specifies the DB and its evolution

For that we need the right language

Situation Calculus

A family of languages of many-sorted first-order logic

Used in logic-based KR to describe evolving domains subject to the execution of actions

Regained popularity in the 90's due mainly to the work of Raymond Reiter and collaborators [Reiter 01]

- A simple solution to the *frame problem* in the SC

  Given a specification of precondition and effects of actions, how to obtain a compact, economical specification of the many things that are not changed by the actions

- Basis for cognitive robotics programs: GOLOG, CONGO-LOG

The Languages

- Domain individual, situations (states), and actions at first-order level

- First–order quantifications over sorts: $\forall \bar{x}$, $\forall s$, $\forall a$

- S_0, name for initial situation

- Function name do: $do(a, s)$ is the successor state that results from executing action a at state s

- Predicate $Poss$: $Poss(a, s)$ says that action a is possible at state s

- Parameterized action terms, e.g. $promote(x, p)$

- Predicates with situation argument, e.g. $Enrolled(x, p, s)$

- Static predicates

Foundational Axioms for the SC

Unique Names Axioms for Actions: $a_i(\bar{x}) \neq a_j(\bar{y})$, for all different action names a_i, a_j, e.g. $delete(id) \neq classifyBook(isbn, id')$

Unique Names Axioms for States: $S_0 \neq do(a, s)$

$do(a_1, s_1) = do(a_2, s_2) \ \rightarrow \ a_1 = a_2 \wedge s_1 = s_2$

For some reasoning tasks the *Induction Axiom on States* (IA):

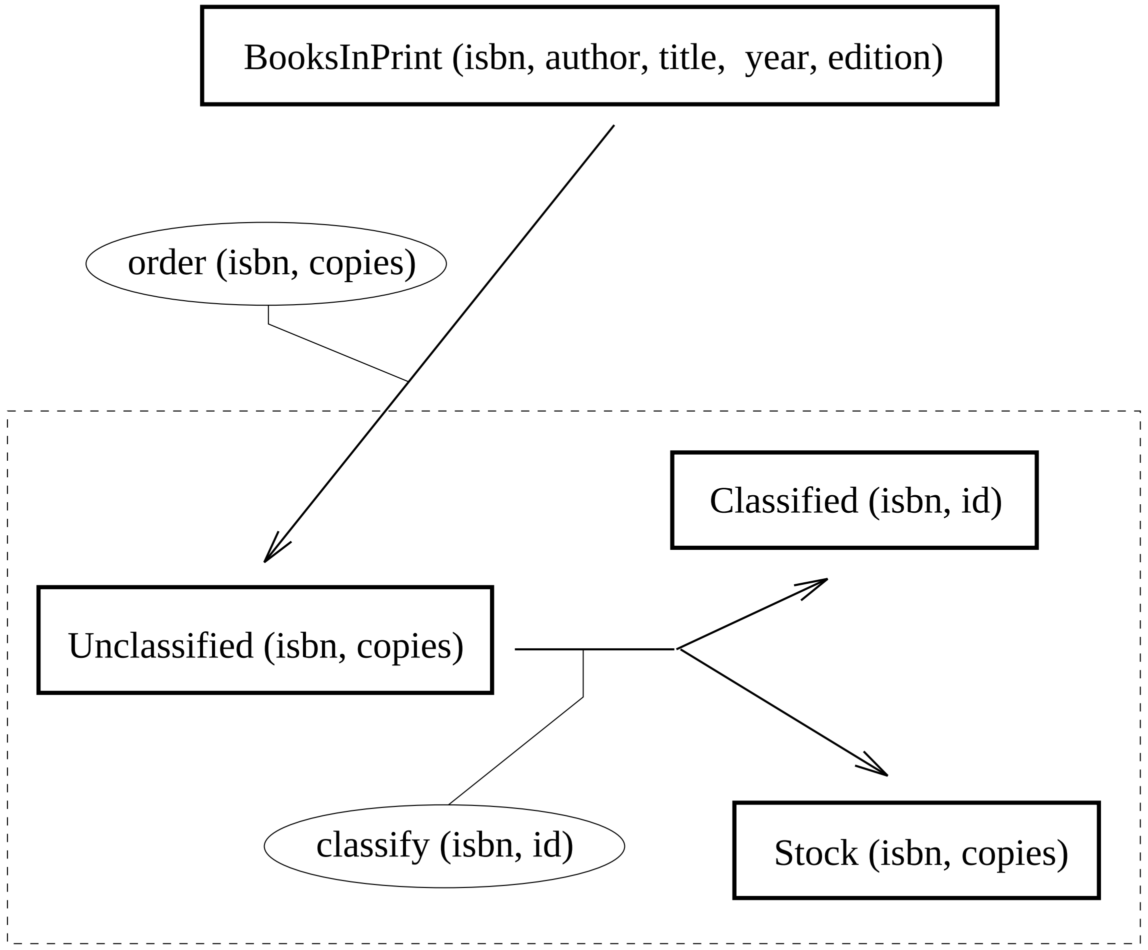
$$\forall P \ (P(S_0) \wedge \forall s \forall a \ (P(s) \rightarrow P(do(a, s))) \ \ \rightarrow \ \ \forall s \ P(s))$$

restricts the domain of situations to S_0 plus the situations obtained by executing actions

We are usually interested in reasoning about states that are *accessible* from S_0 by executing a finite sequence of legal actions:

$\neg s < S_0 \qquad\qquad s < do(a, s') \equiv Poss(a, s') \wedge s \leq s'$

Specifying DB Updates



Update actions: $order(isbn, copies)$, $classifyBook(isbn, id)$, $deleteBook(id)$

Predicates: $BooksInPrint(isbn, title, author, editor, year, edition)$, $Unclassified(isbn, copies, s)$, $Classified(isbn, id, s)$, $Stock(isbn, copies, s)$

Action Preconditions:

- $Poss(order(isbn, copies), s) \equiv$
$$(\exists\ title, author, editor, year, edition)\ BooksInPrint(isbn, title, author, editor, year, edition)$$

- $Poss(classifyBook(isbn, id), s) \equiv$
$$\neg(\exists\ isbn')\ Classified(isbn', id, s) \wedge (\exists\ copies)\ Unclassified(isbn, copies, s)$$

Successor State Axioms

Solution to the frame problem is based on the use (generation) of successor state axioms (Reiter 91)

Specify under what conditions each fluent becomes true at an arbitrary successor state $do(a, s)$

$$\forall a \forall s \; Poss(a, s) \;\rightarrow\; [R(do(a, s)) \;\equiv\; \gamma_R^+(a, s) \vee (R(s) \wedge \neg\gamma_R^-(a, s))]$$

R is true at a successor state iff it is made true or it was already true and it is not made false

This solution relies on the possibility of quantifying over deterministic actions

Actions are syntactically atomic, but semantically complex

In DB applications (Reiter 95), individual actions may result in several DB updates

A SSA

$$
\begin{array}{l}
\forall a\ Poss(a,s)\ \longrightarrow \\
\qquad (Stock(isbn,j,do(a,s))\ \equiv \\
\qquad\qquad a = delete\_book(id)\ \wedge \\
\qquad\qquad\qquad (Classified(isbn,id,s)\ \wedge \\
\qquad\qquad\qquad\qquad \exists i\ (Stock(isbn,i,s) \wedge i > 1 \wedge j = i-1))\ \vee \\
\qquad\qquad a = classify\_book(isbn,id)\ \wedge \\
\qquad\qquad\qquad (\exists i\ (Stock(isbn,i,s) \wedge j = i+1)\ \vee \\
\qquad\qquad\qquad\qquad \neg\exists i\ (Stock(isbn,i,s))\wedge \\
\qquad\qquad\qquad\qquad\qquad j = 1)\ \vee \\
\qquad\qquad Stock(isbn,j,s)\ \wedge \\
\qquad\qquad\qquad \neg(a = delete\_book(id)\ \wedge \\
\qquad\qquad\qquad\qquad\qquad Classified(isbn,id,s)\ \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad Stock(isbn,j,s)\ \vee \\
\qquad\qquad\qquad\qquad a = classify\_book(isbn,id)\ \wedge \\
\qquad\qquad\qquad\qquad\qquad Stock(isbn,j,s)))
\end{array}
$$

(can be constructed from positive and negative effect axioms)

Integrity Constraints

Static ICs, e.g. FDs, are sentences that must hold at every legal state of the database:

$$\begin{tikzcd} DB\ spec. & \models\ \forall s\ (S_0 \leq s\ \rightarrow\ \varphi(s)) \\ Th(DB(S_0)) \cup\ Spec.\ of\ Dynamics \arrow[u] & \end{tikzcd}$$

For example, for a FD it should hold

$$\begin{aligned} DB\ spec.\ \models\ \forall S_0 \leq s\ \ \rightarrow\ (&Classified(isbn1, id, s) \land \\ &Classified(isbn2, id, s)\ \ \rightarrow\ isbn1 = isbn2) \end{aligned}$$

Induction principle for proving static ICs can be derived:

$$\begin{aligned} \forall P([P(S_0)\ \land\ \forall s \forall a(P(s) \land Poss(a,s)\ \rightarrow\ P(do(a,s))]\ \ \rightarrow \\ \forall s(S_0 \leq s\ \rightarrow\ P(s))) \end{aligned}$$

Similar treatment for dynamic ICs:

A person's salary cannot decrease:

$$\forall s, s' (S_0 \leq s \leq s' \ \rightarrow \ (Salary(x,p,s) \wedge Salary(x,p',s') \ \rightarrow \ p \leq p'))$$

ICs proved by automated mathematical induction [LB et al. 96]

Specifying the Dynamics of Relational Views

Given a specification of DB dynamics in terms of SSAs

Automatically derive SSAs for (relational calculus) views

Applications to view and database maintenance [Arenas et al. 98]

Can be extended to aggregate views

Combination with hypothetical reasoning? "What if" queries?

Hypothetical Database Reasoning [Arenas et al. 02]

Queries in first-order past temporal logic about a whole evolution of the database

Application to transformation of dynamic ICs into static ICs

Application to transformation of history dependent actions into "Markovian" actions

Change and Ontologies

Complex actions can be constructed from basic actions (cf. GOLOG)

It is possible to derive SSAs from complex actions [Fritz et al. 08]

They and GOLOG used for composition of semantic web services [McIlr. et al. 02]

Actually, generic ontologies (ontology languages) have been proposed for specifying action and change

General ontologies $\mathcal{O}^g$ for high-level descriptions of action preconditions, actions effects, etc. (e.g. OWL-S, FLOWS) [Martin et al. 07, Grün. et al. 08]

Specific theories $\mathcal{O}^s$ of action and change (as in the library example above) can feed $\mathcal{O}^g$

The combination can be applied to specify the evolution of an initially static domain, e.g. a database D

A three-layered approach ...

7. Inconsistency Handling

What If the database is inconsistent?

Inconsistencies can be detected, and data can be changed to reach a physical consistent state

This kind of data cleaning may be difficult, impossible, nondeterministic, undesirable, unmaintainable, etc.

We may have to live with inconsistent data ...

The database (the model) is departing from the outside reality that is being modeled

However, the information is not all semantically incorrect

Most likely most of the data in the database are still "consistent"

Idea:

(a) Keep the database as it is

(b) Obtain semantically meaningful information at query time; dealing with inconsistencies on-the-fly

Particularly appealing in virtual data integration ...
(no direct access to the data sources)

This requires: [LB 11]

(a) Logically characterizing consistent data within an inconsistent database

Via database repairs: Consistent instances that minimally depart from the original instance

Consistent data is invariant across the class of all repairs

(b) Developing algorithms for retrieving the consistent data: Consistent query answering

Example: For the instance D that violates

FD: $Name \rightarrow Salary$

$Employee$	$Name$	$Salary$
	$page$	5K
	$page$	8K
	$smith$	3K
	$stowe$	7K

Two possible (minimal) repairs if only deletions/insertions of whole tuples are allowed: D_1, resp. D_2

$Employee$	$Name$	$Salary$
	$page$	5K
	$smith$	3K
	$stowe$	7K

$Employee$	$Name$	$Salary$
	$page$	8K
	$smith$	3K
	$stowe$	7K

$(stowe, \text{7K})$ persists in all repairs: it is consistent information

$(page, \text{8K})$ does not; actually it participates in the violation of FD

A consistent answer to a query $\mathcal{Q}$ from a database D is an answer that can be obtained as a usual answer to $\mathcal{Q}$ from every possible repair of D wrt IC (a given set of ICs)

- $\mathcal{Q}_1 : Employee(x, y)?$

  Consistent answers: $(smith, \text{3K}), (stowe, \text{7K})$

- $\mathcal{Q}_2 : \exists y Employee(x, y)?$

  Consistent answers: $(page), (smith), (stowe)$

CQA may be different from classical data cleaning!

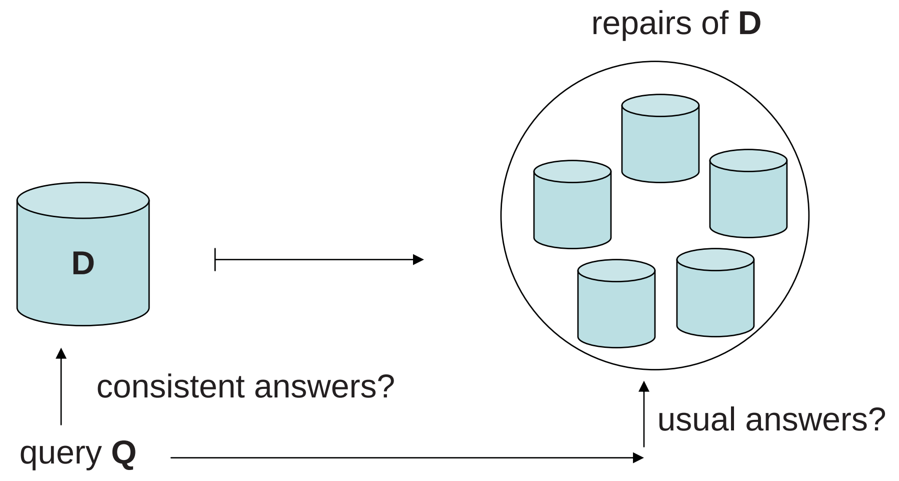
However, CQA is relevant for data quality; an increasing need in business intelligence

It also provides concepts and techniques for data cleaning

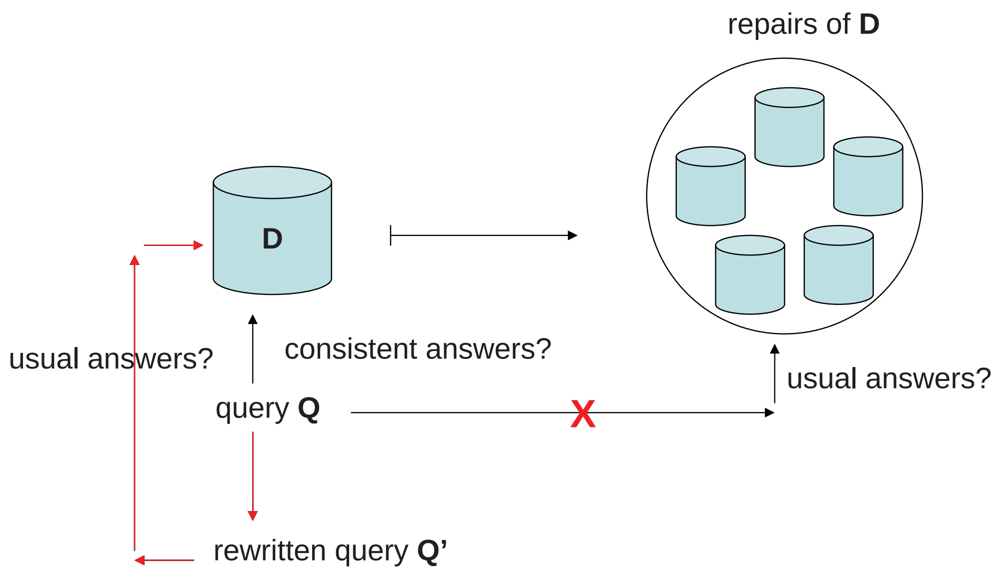
Paradigm shift: ICs are constraints on query answers, not on database states!

Depending on the ICs and the queries, tractable and intractable cases for CQA have been identified

That it may be intractable is not unthinkable: there could be many repairs and we have to query each of them (at least conceptually)



For some tractable cases, query rewriting algorithms have been developed



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

$\mathcal{Q}'(x,y): \;\; Employee(x,y) \wedge \neg\exists z(Employee(x,z) \wedge z \neq y)$

Pose the second query to the inconsistent database to obtain the consistent answers to the first query

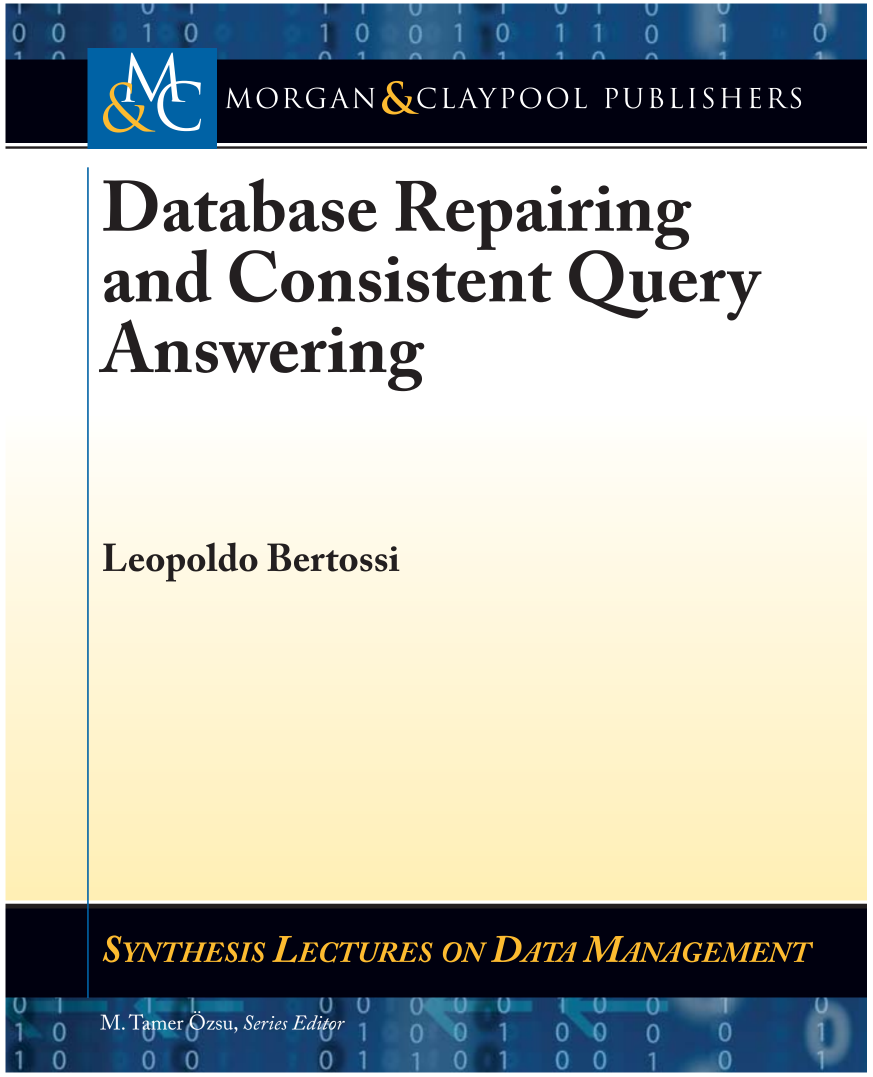
$\mathcal{Q}$ is conjunctive query, but $\mathcal{Q}'$ is not, but still evaluable in polynomial time in the size of the DB

When the rewritten query $\mathcal{Q}'$ is a relational calculus (i.e. FO) query, CQA is tractable ...

For higher-complexity cases, specifications of repairs by means of logic programs with stable model semantics can be used

CQA becomes querying (as usual) a logic program, say a Datalog program with possible complex extensions

The boundary between tractable and intractable cases of CQA has to be understood, in particular, to know the scope of applicability of FO query rewriting


MORGAN & CLAYPOOL PUBLISHERS

Database Repairing and Consistent Query Answering

Leopoldo Bertossi

SYNTHESIS LECTURES ON DATA MANAGEMENT

M. Tamer Özsu, *Series Editor*

2011

References:

[Arenas et al. 98] Arenas, M., Bertossi, L. The Dynamics of Database Views. In 'Transactions and Change in Logic Databases', Springer LNCS 1472, (1998)

[Arenas et al. 02] Arenas, M., Bertossi, L. Hypothetical Temporal Queries in Databases. Journal of Intelligent Information Systems, 2002, Volume 19, Number 2, pp. 231-259.

[Bancil. et al. 81] Francois Bancilhon, Nicolas Spyratos: Update Semantics of Relational Views. ACM Trans. Database Syst. 6(4): 557-575 (1981)

[LB et al. 96] Bertossi, L., Pinto, J., Saez, P., Kapur, D., Subramaniam, S. Automating Proofs of Integrity Constraints in the Situation Calculus. In 'Foundations of Intelligent Systems', Proc. ISMIS'96, Springer LNAI 1079 (1996)

[LB et al. 13] Leopoldo Bertossi, Lechen Li: Achieving Data Privacy through Secrecy Views and Null-Based Virtual Updates. IEEE TKDE, 2013.

[LB 11] Leopoldo Bertossi: Database Repairing and Consistent Query Answering. Morgan & Claypool Publishers (2011)

[Blakeley et al. 89] Jose Blakeley, Neil Coburn, Per-Ake Larson: Updating Derived Relations: Detecting Irrelevant and Autonomously Computable Updates. ACM Trans. Database Syst. 14(3): 369-400 (1989)

[Cosm. et al. 84] Stavros S. Cosmadakis, Christos H. Papadimitriou: Updates of Relational Views. J. ACM 31(4): 742-760 (1984)

[De Giac. et al. 09] Giuseppe De Giacomo, Maurizio Lenzerini, Antonella Poggi, Riccardo Rosati: On Instance-level Update and Erasure in Description Logic Ontologies. J. Log. Comput. 19(5): 745-770 (2009)

[Fritz et al. 08] Fritz, C.; Baier, J. A.; and McIlraith, S. A. ConGolog, Sin Trans: Compiling ConGolog into Basic Action Theories for Planning and Beyond. Proceedings KR'08 (2008)

[Godfrey et al. 98] Parke Godfrey, John Grant, Jarek Gryz, Jack Minker: Integrity Constraints: Semantics and Applications. In *Logics for Databases and Information Systems*, 1998: 265-306

[Greco et al. 12] Sergio Greco, Cristian Molinaro, Francesca Spezzano: Incomplete Data and Data Dependencies in Relational Databases. Morgan & Claypool Publishers (2012)

[Grün. et al. 08] Michael Grüninger, Richard Hull, Sheila A. McIlraith. Bull. IEEE CS on Data Engineering (2008)

[Gupta et al. 95] Ashish Gupta, Inderpal Singh Mumick: Maintenance of Materialized Views: Problems, Techniques, and Applications. IEEE Data Eng. Bull. 18(2): 3-18 (1995)

[Gupta et al. 99] Ashish Gupta, Inderpal Singh Mumick (eds.): Materialized Views. Morgan Kaufmann (1999)

[Imielinski et al. 84] Tomasz Imielinski, Witold Lipski Jr.: Incomplete Information in Relational Databases. J. ACM 31(4): 761-791 (1984)

[Kakas et al. 90] Antonis C. Kakas, Paolo Mancarella: Database Updates through Abduction. VLDB 1990: 650-661

[Kakas et al. 92] Antonis C. Kakas, Robert A. Kowalski, Francesca Toni: Abductive Logic Programming. J. Log. Comput. 2(6): 719-770 (1992)

[Klug 80] Anthony C. Klug: Calculating Constraints on Relational Expressions. ACM Trans. Database Syst. 5(3): 260-290 (1980)

[Klug 82] Anthony C. Klug, Rod Price: Determining View Dependencies Using Tableaux. ACM Trans. Database Syst. 7(3): 361-380 (1982)

[Lecht. et al. 03] Jens Lechtenboerger, Gottfried Vossen: On the computation of relational view complements. ACM Trans. Database Syst. 28(2): 175-208 (2003)

[Martin eta al. 07] David Martin, Mark Burstein, Drew McDermott, Sheila McIlraith, Massimo Paolucci, Katia Sycara, Deborah L. McGuinness, Evren Sirin, Naveen Srinivasan: Bringing Semantics to Web Services with OWL-S. World Wide Web 10:243-277 (2007)

[McIlr. et al. 02] McIlraith, S., and Son, T. Adapting Golog for Composition of Semantic Web Services. 2002. Proc. KR'02 (2002)

[Reiter 84] Raymond Reiter: Towards a Logical Reconstruction of Relational Database Theory. In "On Conceptual Modeling", M. Brodie, J. Myolopoulos, J. Schmidt (eds.), Springer (1984)

[Reiter 92] Raymond Reiter: What Should a Database Know? J. Log. Program. 14(1&2): 127-153 (1992)

[Reiter 01] Raymond Reiter: Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems, MIT Press (2001)

[Widom et al. 95] J. Widom, S. Ceri. Active Database Systems: Triggers and Rules for Advanced Database Processing. Morgan Kaufmann (1995)

Characterizing and Computing Consistent Information from Databases[1]

Leopoldo Bertossi

Carleton University
School of Computer Science
Ottawa, Canada

[1]Chapter 2 of PhD Course at U. Calabria, Arcavacata, 2013.

Why Consistent Query Answering?

A (relational) database instance D may be inconsistent

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

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

However, we do not throw D away

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

We can still obtain meaningful and correct answers from D

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

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

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

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

Consistent Answers and Repairs: The Gist

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

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

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

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

Repairs D_1, resp. D_2

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

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

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

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

A Long and Fruitful Research Trajectory

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

- Extensions of mechanisms proposed in (Arenas et al.; Pods 1999)
- Computational complexity analysis of CQA
- Alternative (but related) characterizations of consistent answer

  Other kinds of repairs, special kinds of ICs, special kinds of data and data models
- Implementations efforts
- Conceptual/theoretical applications to data integration, peer data exchange, etc.

- Logical formalization of reasoning with consistent data wrt inconsistent databases
- Understanding the “logical laws” of consistent query answering in databases
- Extension of notions and techniques to other kinds of “databases”: DL ontologies, ...

Fixed: DB schema with (infinite) domain; and a set of FO ICs IC

Definition: A repair of instance D is an instance D'

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

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

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

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

A model-theoretic definition ... (Arenas et al.; Pods 1999)

Example: (continued)

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

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

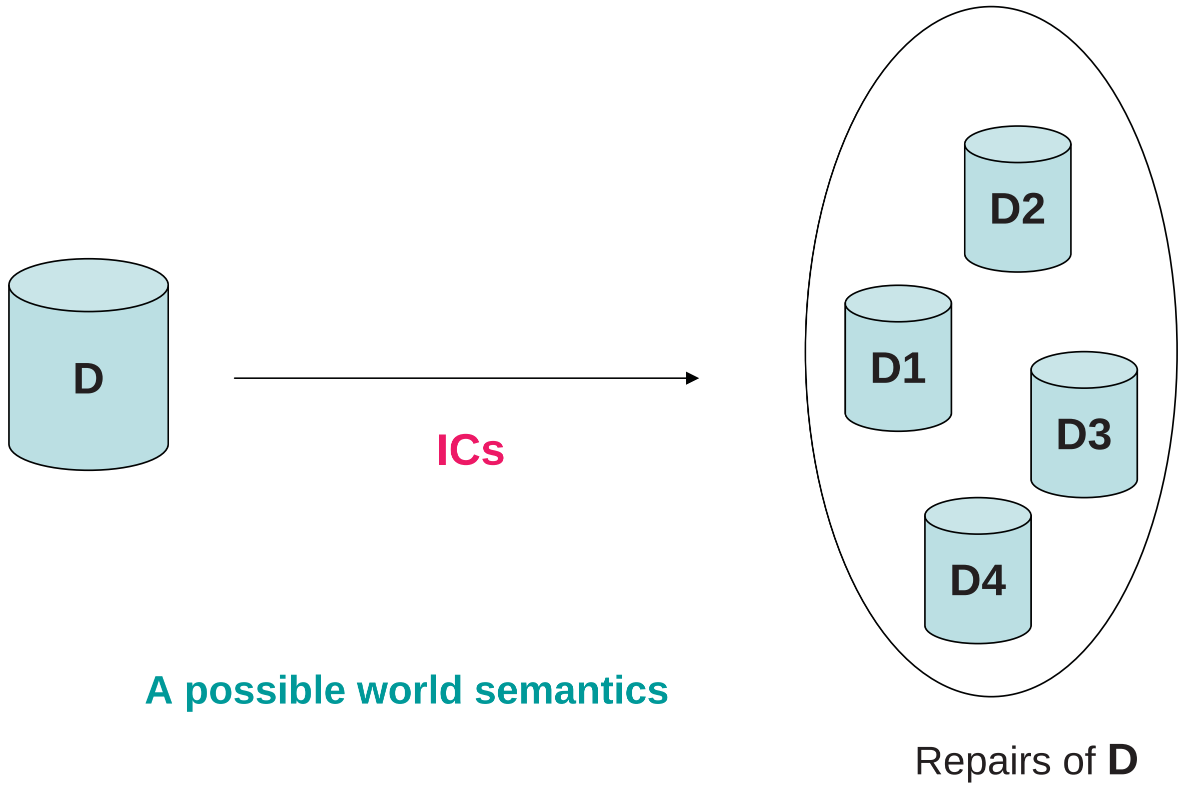
$$D \models_{FD} \exists x Employee(Page, x)$$

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

The repairs are:

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

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



What is consistently true is what is true in (the class of) all repairs, simultaneously

A model-theoretic definition, in terms of intended models (instances in this case)

Can we specify this class and reason from the specification?

Computing Consistent Answers?

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

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

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

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

It has 2^n possible repairs!

Try to avoid or minimize computation of repairs ...

FO Query Rewriting (sometimes)

First-Order queries and constraints

Approach: Transform the query and keep the database instance!

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

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

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

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

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

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

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

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

Residue obtained by resolution between query literal and IC

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

Appending the residue is like locally enforcing the IC!

Example: (continued) Same FD:

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

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

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

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

Can be obtained via rewritten (FO but not conjunctive) query:

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

Equivalent to: $Employee(x,y)\ \wedge$

$$\neg\exists z\ (Employee(x,z)\ \wedge\ y \neq z)$$

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

We have seen that already with simple FDs, we can have exponentially many repairs

However, for some queries consistent answers wrt FDs can be computed in polynomial time

Because the rewritten query is posed to the original database as a usual query written in relational calculus

In general, T has to be applied iteratively

Example:

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

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

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

Apply T again, now to the appended residues

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

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

And again:

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

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

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

Does it always exist?

In general, an infinitary query:

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

Is T^ω sound, complete, finitely terminating?

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

(Arenas et al.; Pods 1999)

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

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

Example:

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

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

Some Limitations:

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

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

T may not give us a FO rewriting, but other approaches to FO query rewriting are in principle possible (cf. later)

What are the limitations of FO query rewriting for CQA?

Applicable to T and any other possible attempt?

CQA based on first-order query rewriting has provable intrinsic limitations

This is because the complexity of CQA may be intrinsically higher than polynomial (cf. later)

What Kind of Logic for CQA?

From the logical point of view:

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

Notice that consistent query answering is non-monotonic

A non-monotonic semantics for Spec and its logic is expected

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

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

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

However

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

(What other logical properties of CQA reasoning/entailment?)

What kind of logic can be used to specify repairs?

Not classical logic, which is monotonic, i.e. for which it holds:

$$\Sigma \models \varphi \ \Rightarrow\ \Sigma \cup \{\psi\} \models \varphi$$

It is possible to specify the repairs of a database wrt FO ICs as the models of disjunctive logic programs with stable model semantics, aka. *disjunctive answer set programs* (cf. later)

(DASP $=$ disjunctive ASPs)

We can immediately obtain some complexity upper bounds from general results for query answering from DASPs (cf. later)

Complexity of CQA

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

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

- The problem of CQA is a decision problem:

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

(usually conjunctive queries)

The decision problem is parameterized by the ICs and the query

If the queries are *boolean* (i.e. yes/no, no free variables, sentences) only D matters

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

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

This provides an upper bound for data complexity of CQA

- The decision problem of repair checking is

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

Notice that the test involves checking minimality ...

- Complexity of checking if a set of database atoms (a subset of the Herbrand base) is a stable model of a DASP is *coNP*-complete in data

This provides an upper bound for repair checking

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

The head-cycle free programs (HCF) are DASPs with special syntactic properties

HCF DASPs can be transformed into a (non-disjunctive) normal program with the same stable models (same repairs in our case)

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

- For some classes of ICs, repair programs become HCF

For sets IC of denial constraints, the DASPs are HCF

This includes

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

Also FDs ... $\forall xy \neg (R(x,y) \wedge R(x,z) \wedge z \neq y)$

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

For FDs and key-constraints CQA and repair checking have upper-bounds $coNP$ and $PTIME$ in data

- Actually, it is easy to give a direct proof of tractability of repair checking for FDs

For FDs (and denials in general), repairs are maximal subsets of the original instance

To check if a subset D' of original D:

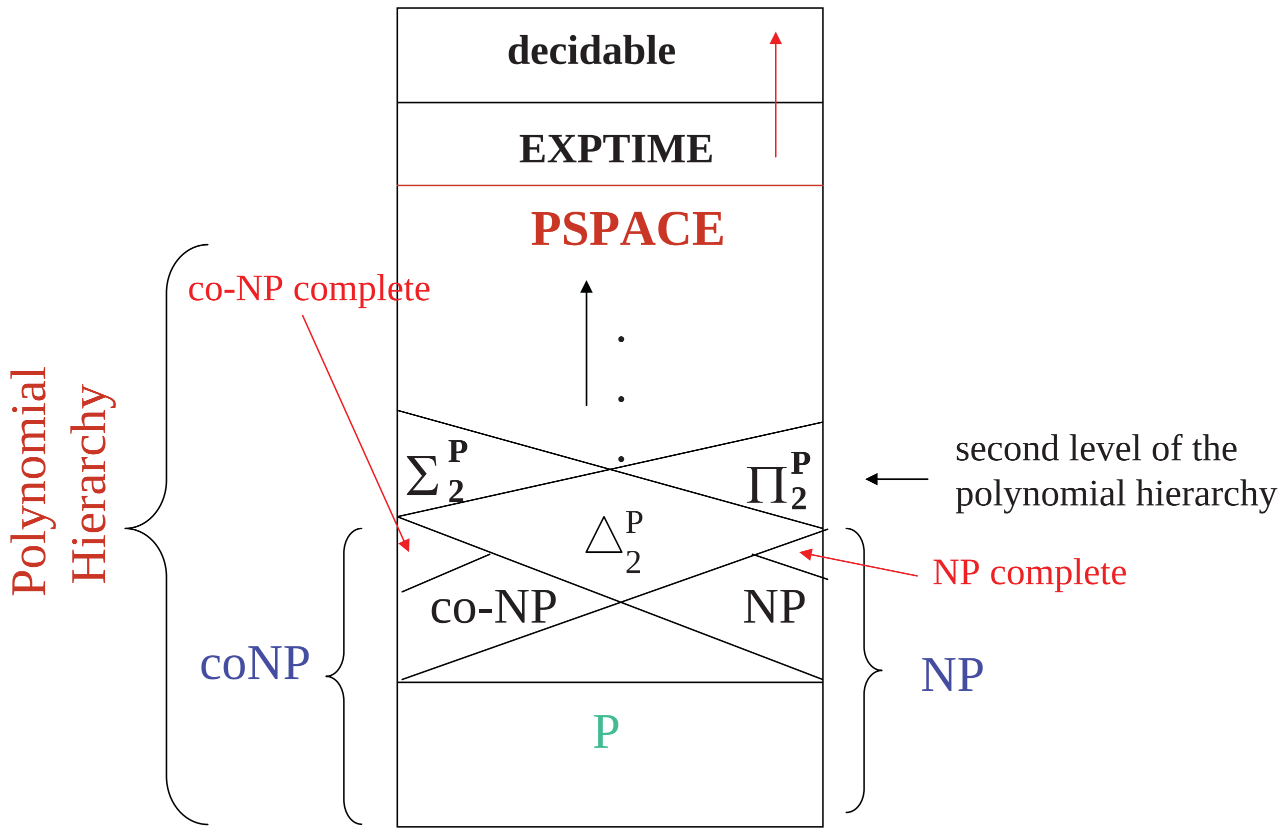
1. Check if it is consistent (easy, just a query)
2. Check if can be extended with tuples in $D \smallsetminus D'$ while still being consistent (easy)

These methodology also applies to denial constraints in general

Closing and Understanding the Gap

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

$$
\begin{aligned}
\Pi_0^P := \Sigma_0^P &:= \Delta_0^P := P \\
\Delta_{i+1}^P &:= P^{\Sigma_i^P} \\
\Sigma_{i+1}^P &:= NP^{\Sigma_i^P} \\
\Pi_{i+1}^P &:= coNP^{\Sigma_i^P} \\
\\
\Pi_2^P &= coNP^{NP}
\end{aligned}
$$



● First explicit analysis of complexity of CQA was done for atomic scalar (no group-by) aggregate queries and FDs (cf. later) (Arenas, Bertossi, Chomicki; ICDT 2001)

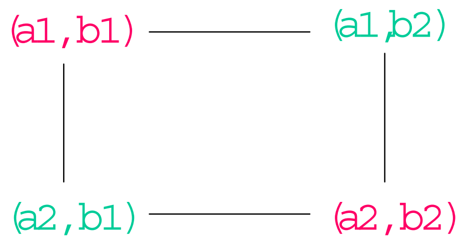
Graph-theoretic methods were applied

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

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

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

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



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

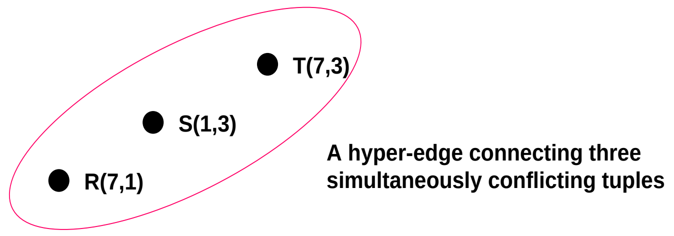
Each repair of D corresponds to a maximal independent set in $CG_{FD}(D)$

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

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

Also to denial constraints; actually using conflict hypergraphs

(Chomicki et al.; I&C 2005)


T(7,3)
S(1,3)
R(7,1)
A hyper-edge connecting three simultaneously conflicting tuples

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

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

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

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

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

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

1. FDs and projection-free conjunctive queries

   Also some conjunctive queries with limited projection

   (Chomicki et al., I&C 2005)

2. Key Constraints (KCs) and some syntactic classes of conjunctive queries with restricted projection

   (Fuxman, Miller; ICDT 2005)

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

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

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

(underlined variables show the key for the relation, as usual)

Most of the research has concentrated on boolean conjunctive queries as above, i.e. without free variables

They get *Yes/No* answers (the complexity is the same as for non-boolean queries)

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

   Same property of FO rewritability (Wijsen; DBPL 2007)

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

(For more details, and complete and up-to-date references see my monograph; also (Wijsen; PODS 2013) for more recent results)

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

However, PTIME CQA based on graph algorithms (or anything else) does not imply FO query rewriting ...

- Repair checking is *PTIME* for arbitrary FDs and acyclic inclusion dependencies (deletion-based repair semantics)

  (Chomicki et al., I&C 2005)

What about lower bounds?

• Consistent conjunctive query answering wrt FDs is $coNP$-complete

To prove this we need to prove that the complement of this problem is NP-complete

For membership, notice:

$\bar{t}$ *is not* a consistent answer to $\mathcal{Q}(\bar{x})$ iff there is a repair D' such that $D' \not\models \mathcal{Q}(\bar{t})$

Such a witness D' is "short" and can be checked in PTIME (wrt to being repair and not making the query true)

For hardness, see later ...

However, in general terms we need a reduction, e.g. from SAT:

- Fix the schema, ICs and boolean query $\mathcal{Q}$
- The only variable part is the (inconsistent) instance D (and the candidate answer when query non-boolean)
- To a propositional formula φ in CNF, efficiently associate an instance D
- φ is satisfiable iff Yes is not the consistent answer to $\mathcal{Q}$ in D

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

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

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

- For arbitrary FDs and inclusion dependencies (deletions only) (Chomicki et al., I&C 2005)
  - Repair checking becomes *coNP*-complete
  - CQA becomes Π_2^P-complete

Notice that this matches the complexity of skeptical reasoning for DASPs

- However, for arbitrary universal ICs and RICs (even cyclic), CQA becomes Π_2^P-complete if RICs are repaired with non-propagating SQL nulls (Bravo, Bertossi; IIDB 2006)
- For arbitrary FDs and inclusion dependencies (including RICs), "combined" CQA becomes undecidable (Cali, Lembo, Rosati; Pods'03)

  Here, inclusion dependencies are solved by insertions

• FO Rewriting vs. *PTIME*

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

$\mathcal{Q}: \ \exists x \exists y (R(\underline{x}, y) \land R(\underline{y}, c))$ (Wijsen; DBPL 2007)

Using Ehrenfeucht-Fraisse games

EFGs are used to prove that two structures D_1, D_2 satisfy the same FO-sentences, i.e. that they are *elementary equivalent* $(D_1 \equiv_{FO} D_2)$

In our case, two inconsistent instances D_1, D_2 for same schema and KCs as above

If one of them, D_1, *Yes* is the consistent answer to $\mathcal{Q}$, but for D_2, *No* is the consistent answer (i.e. false in some repair of D_2)

The original query cannot be FO rewritable into a new boolean FO query $\mathcal{Q}'$

Both D_1 and D_2 would have to satisfy $\mathcal{Q}'$ or not, so they would have the same consistent answer ...

In which PTIME logic can the rewriting be done (if any)?

Newer Emphasis

Most of the results mentioned above can be seen from a higher-level perspective

● As contributions to two more general problems that have started to be addressed: (Wijsen; TODS'12) (Kolaitis, Pema; IPL'12)

(A) Find an algorithm that takes as input a boolean FO query $\mathcal{Q}$ (with a given set of key constraints KC which are indicated together with the query), and decides whether $CQA(\mathcal{Q})$ is FO definable (i.e. there is a FO rewriting)

(B) Find an algorithm that takes as input a boolean FO query $\mathcal{Q}$, and decides whether $CQA(\mathcal{Q})$ is in $PTIME$ or $coNP$-complete (i.e. tractable or intractable)

Neither may be the case: If $PTIME \neq coNP$, there are problems in $coNP \smallsetminus P$ that are not $coNP$-complete

- Research on the above problems (A) and (B) has focused on conjunctive queries

Additional restrictions are often imposed:

(a) $\mathcal{Q}$ must be acyclic (in the classical sense); and/or

(b) $\mathcal{Q}$ has no self-join, meaning that every relation name occurs at most once in $\mathcal{Q}$

- Problem (A) has been solved under both restrictions

(Wijsen; TODS'12)

For this class of queries, the decision boundaries in (A) and (B) do not coincide:

For the conjunctive query $\mathcal{Q}\colon \exists x \exists y (R(\underline{x}, y) \wedge S(\underline{y}, x))$, $CQA(\mathcal{Q})$ is in $PTIME$, but not FO definable
(different from the one on page 39, that has self-joins)

- More queries with this property have later been identified

(Kolaitis, Pema; IPL'12)

The class of acyclic conjunctive queries without self-joins, notwithstanding its restrictions, remains a large class of practical interest

- Relatively little is known about $CQA(\mathcal{Q})$ for conjunctive queries $\mathcal{Q}$ that are cyclic and/or contain self-joins

As mentioned above, FO definability of $CQA(\mathcal{Q})$ is guaranteed for all conjunctive queries $\mathcal{Q}$ belonging to the semantic class of *key-rooted* conjunctive queries

Key-rooted queries can be cyclic and contain self-joins

However, no algorithm is known to test whether a conjunctive query is key-rooted

- Concerning problem (B), it is an open conjecture that for every conjunctive query $\mathcal{Q}$ without self-join, it is the case that $CQA(\mathcal{Q})$ is in $PTIME$ or $coNP$-complete

An open dichotomy conjecture ...

Not anymore????? A recently submitted paper claims to have settled the conjecture, positively ...

- For a class of conjunctive queries with two atoms, a dichotomy result for CQA, i.e. about being in $PTIME$ or being $coNP$-complete, has been obtained (Kolaitis, Pema; IPL'12)

Beyond Data Complexity

- For arbitrary FDs and inclusion dependencies (including RICs), CQA becomes undecidable (Cali, Lembo, Rosati; Pods 2003)

Issues:

- Three parameters (ICs, query, and DB) are input
- Inclusion dependencies repaired through insertions
- Infinite underlying domain that can be used for insertions
- Cycles in the set of inclusion dependencies
- Problematic interaction of FDs and RICs

- Different combinations of input parameters

(Arenas, Bertossi; AMW'10)

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

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

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

  Reduction from SAT^c for finite structures

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

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

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

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

  Encode halting problem for Turing machines

  ICs are universal

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

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

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

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

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

  is *coNEXP*-complete

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

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

Logical Specifications of Database Repairs[1]

Leopoldo Bertossi

Carleton University
School of Computer Science
Ottawa, Canada

[1]Chapter 3 of PhD Course at U. Calabria, Arcavacata, 2013.

The Reason Why ...

We have seen that a DB may have multiple repairs wrt a set of ICs

What is consistent in the DB wrt the ICs is what is true in *all* of them

That is, what is certain ...

How to handle and reason about all repairs?

Axiomatize them ... (by means of logical formulas)

As groups and vector spaces in math ...

In this case, we say that we *specify them*

And we may have to use special, adequate logics $\mathfrak{L}$

If $\mathcal{T}$ is such a theory (axiomatization), we expect the repairs D' to be *the models* of $\mathcal{T}$

$$D' \text{ is repair} \iff D' \models_{\mathfrak{L}} \mathcal{T}$$

(here $\models_{\mathfrak{L}}$ used for satisfaction)

That is, those structures that make $\mathcal{T}$ true (according to the semantics of the logic)

Then, a statement (formula) φ is true in all repairs (i.e. consistent) if it is a logical consequence of $\mathcal{T}$:

$$\varphi \text{ is true in every repair } D' \iff \mathcal{T} \models_{\mathfrak{L}} \varphi$$

(here $\models_{\mathfrak{L}}$ used for logical entailment)

So, obtaining/confirming consistent knowledge becomes logical reasoning with/from specification $\mathcal{T}$!

Another crucial issue:

Independently from the logical representation, consistent knowledge from a DB exhibits a non-monotonic behavior

Example: Consider schema $R(X,Y)$, instance $D = \{R(a,b), R(c,d)\}$ and the FD $X \rightarrow Y$

Consistent answers to query $R(x,y)?$: $(a,b), (c,d)$

If we add tuple $R(a,e)$, then we lose the consistent answer (a,b)

That is, previous conclusions may have to be retracted when new knowledge is added

Classical logic is monotonic: $\Sigma \models \varphi \implies \Sigma \cup \{\psi\} \models \varphi$

So, we need a non-monotonic logic! (then, non-classical)

A. Specifying Repairs in APC

Annotated Predicate Logic and DBs

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

The database instance D (seen as Reiter's FO theory) 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, non-classical theories

(classical $=$ with syntax/semantics of FO predicate logic)

Example: Instance 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 sentences (implicitly)

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

Or, universally quantified disjunctions of DB literals plus built-ins

E.g. $\neg Employee(x, y, z) \ \vee \ Employee(x, u, v) \ \vee \ y = u$

D has two repairs:

Employee

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

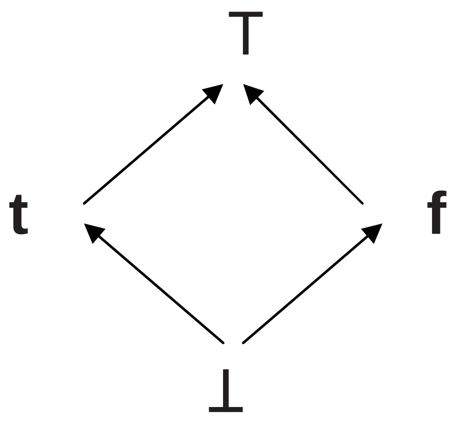
Employee

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

Want to create an APC theory $\mathcal{T}(D, IC)$ from D and IC ...

Usual annotations in APC:

true ($\mathbf{t}$), *false* ($\mathbf{f}$), *contradictory* ($\top$), *unknown* ($\perp$)



4-value lattice

Atoms in an APC theory are annotated with truth values, at the object (syntactic) level, e.g. we could have

$Employee(Steven\ \ Lerman, CEO, 4000)$:$\mathbf{t}$

$Employee(Irwin\ \ Pearson, Salesman, 2000)$:$\mathbf{f}$

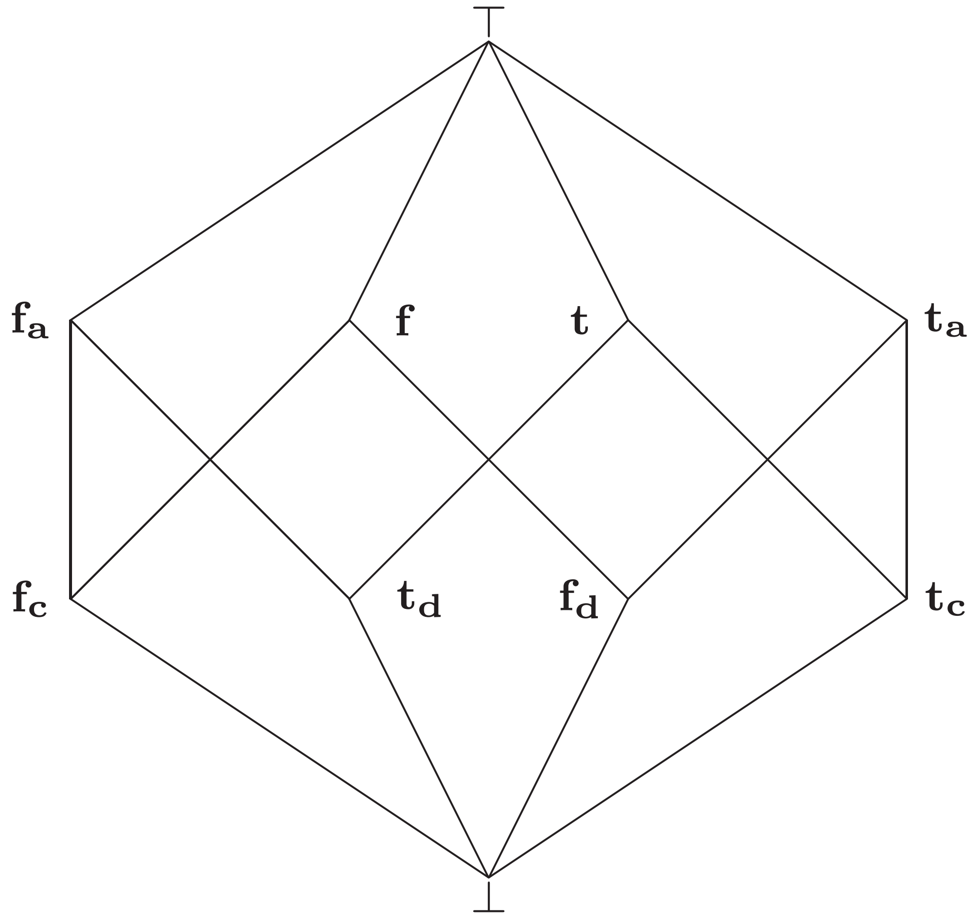
$Employee(V.Smith, CIO, 3000)$:$\top$

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 data-base the advise is to change the truth value to the value prescribed by the constraint
- We need a more refined lattice that the one above ...
- 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: $\mathbf{t_a}$, $\mathbf{f_a}$



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

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

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

- For every $\mathbf{s} \in \mathcal{L}at$: $\quad \bot \leq \mathbf{s} \leq \top$
- $lub(\mathbf{t}, \mathbf{f}) = \top$, $lub(\mathbf{t_c}, \mathbf{f_d}) = \mathbf{t_a}$, 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 $\mathcal{T}(D, IC)$ embedding D and IC into APC:

- Translate the constraint:

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

  into

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

- Translate database facts, e.g. $Employee(J.Page, 5000)$ into $Employee(J.Page, 5000){:}\mathbf{t_d}$

- 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)

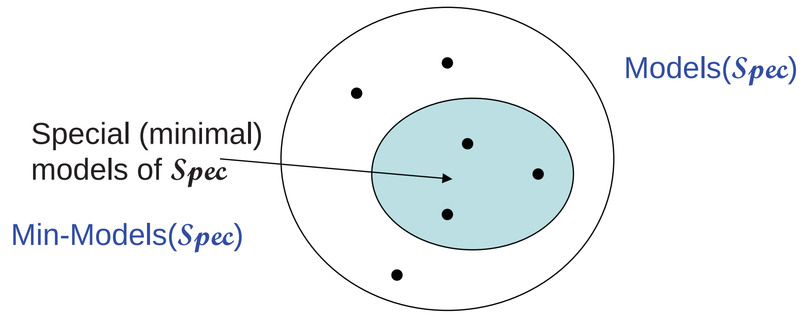
Transition to Non-Monotonicity

We haven't captured the non-monotonic behavior that CQA should exhibit Where is it?

A general fact: When we reason with a subclass of special or intended models of a theory, logical consequence wrt the theory becomes non-monotonic

Intended usually means "minimal" in some sense ...

So, we need to identify a subclass of intended models of the APC theory above ...



$I \models \varphi$ replaced by $I \models_{MinMod} \varphi$

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 $\mathcal{T}(D, IC)$ makes reasoning non-monotonic, as expected

From the specification $\mathcal{T}(D, IC)$ 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: (cont.) Translation into the APC theory $\mathcal{T}(D, IC)$

1. $$\neg Employee(x,y,z) \lor \neg Employee(x,u,v) \lor y = u$$
$$\neg Employee(x,y,z) \lor \neg Employee(x,u,v) \lor z = v$$

transformed into

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

2. We also add for every predicate two rules:

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

A unique constraint truth value!

3. Transforming database instance:

$$Employee(Steven\ Lerman, CEO, 4000) : \mathbf{t_d}$$
$$Employee(Irwin\ Pearson, Salesman, 2000) : \mathbf{t_d}$$
$$Employee(Irwin\ Pearson, Salesman, 2500) : \mathbf{t_d}$$
$$Employee(John\ Miller, Salesman, 1600) : \mathbf{t_d}$$

4. Closed World Assumption:

$$\begin{gathered} Employee(x,y,z) : \mathbf{f_d} \\ \vee \\ x = Steven\ Lerman : \mathbf{t_d} \wedge y = CEO : \mathbf{t_d} \wedge z = 4000 : \mathbf{t_d} \\ \vee \\ x = Irwin\ Pearson : \mathbf{t_d} \wedge y = Salesman : \mathbf{t_d} \wedge z = 2000 : \mathbf{t_d} \\ \vee \\ x = Irwin\ Pearson : \mathbf{t_d} \wedge y = Salesman : \mathbf{t_d} \wedge z = 2500 : \mathbf{t_d} \\ \vee \\ x = John\ Miller : \mathbf{t_d} \wedge y = Salesman : \mathbf{t_d} \wedge z = 1600 : \mathbf{t_d} \end{gathered}$$

5. Equality theory plus Unique Names Assumption

True built-in atoms:

$Steven\ Lerman = Steven\ Lerman{:}\mathbf{t}$, $CEO = CEO{:}\mathbf{t}$, $2000 = 2000{:}\mathbf{t}$, 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)

Repairs as Minimal Models of *Spec*

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 $\mathcal{T}(D, IC)$ 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: $p(\bar{s}) \quad \mapsto \quad p(\bar{s})\text{:}\mathbf{t}$

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

Example: Want the consistent answers to the query

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

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

$$\exists y \exists z \exists w \exists t(Book(x,y,z)\text{:}\mathbf{t} \ \lor \ Book(x,y,z)\text{:}\mathbf{t_a}) \ \land$$
$$(Book(x,w,t)\text{:}\mathbf{t} \lor Book(x,w,t)\text{:}\mathbf{t_a}) \ \land \ (y \neq w)\text{:}\mathbf{t})$$

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?

B. Specifying Repairs with Logic Programs

C. Interlude on Answer Set Programs

Datalog

Example: Intentional DB:

$$\begin{aligned} Ancestor(x, y) &\leftarrow Parent(x, y) & (1) \\ Ancestor(x, y) &\leftarrow Parent(x, z), Ancestor(z, y) & (2) \end{aligned}$$

Extensional DB: $Parent(a, aa), Parent(a, ab), Parent(aa, aaa),$
$Parent(aa, aab), Parent(aaa, aaaa),$
$Parent(c, ca)$

Query: $Q(x) \leftarrow Ancestor(aa, x)$

Computation:

1. Initialize $Ancestor$ and Q as empty:

   $Ancestor \; = \; \emptyset \qquad Q \; = \; \emptyset$

2. View *Ancestor* needs to be computed

   First $Ancestor \; = \; \emptyset$

   Apply rule (1) once, obtaining by forward propagation:

   $$\begin{aligned} Ancestor = \{&(a, aa), (a, ab), (aa, aaa), (aa, aab), (aaa, aaaa), \\ &(c, ca)\} \end{aligned}$$

   This is a partial computation of the view

3. Apply (2) with the tuples obtained in the previous step as input for the right-hand side, and propagate to the head

   This means performing the join $Parent \bowtie \text{“}Ancestor\text{”}$, where “$Ancestor$” is only a partial version of (final) $Ancestor$

Newly generated tuples for $Ancestor$:

$$(a, aaa), (a, aab), (aa, aaaa)$$

New state:

$$\begin{aligned} Ancestor = \{&(a, aa), (a, ab), (aa, aaa), (aa, aab), (aaa, aaaa), \\ &(c, ca), (a, aaa), (a, aab), (aa, aaaa)\} \end{aligned}$$

4. Since new tuples were generated wrt 1., apply rule (2) again, with the partial extension for $Ancestor$ as input, from righ to left (forwards)

   Newly generated tuples for $Ancestor$:

   $$\underline{(a, aaa)}, (a, aab), (aa, aaaa), (a, aaaa)$$

   New state:

   $$\begin{aligned} Ancestor = \{&(a, aa), (a, ab), (aa, aaa), (aa, aab), (aaa, aaaa), \\ &(c, ca), (a, aaa), (a, aab), (aa, aaaa), (a, aaaa)\} \end{aligned}$$

   The underlined tuple was recomputed!

5. Since new tuples were generated, apply rule (2) once more

   Generated tuples for $Ancestor$:

   $$(a, aaa), (a, aab), (aa, aaaa), (a, aaaa)$$

6. No new tuples were obtained (redundant recomputation!); same state

   Block for $Ancestor$ is completely computed

7. Now compute the extension of Q applying the query rule (a selection followed by a projection)

   Generated tuples: $Q = \{aaa, aaaa, aab\}$

Remarks:

- The answer set for the query, or more generally, the contents of the views, were obtained after reaching a fixpoint of the iterative upwards, bottom-up, propagation process

- Several tuples were recomputed; we might try a more incremental evaluation mechanism

  Improvement on this: Semi-Naive Evaluation

- All the computations before last step (7.) were done without considering the parameter aa in the query

  Many tuples were taken up to the upper level and then filtered out: too much useless computation

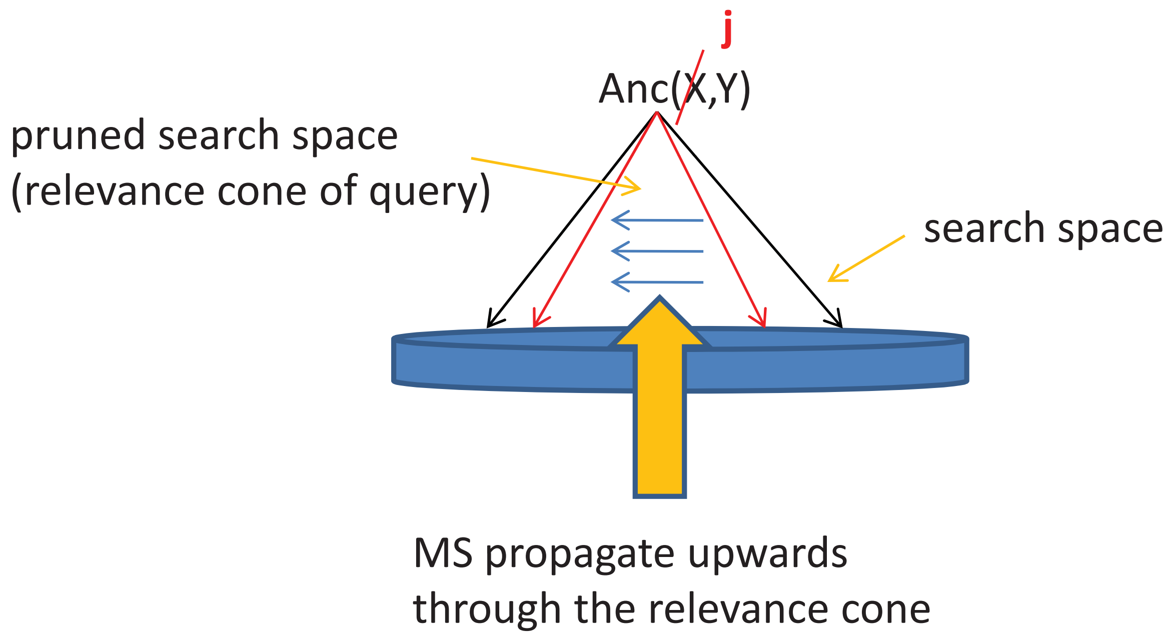
- Improvement on the last issue: Magic Sets Method makes the computation more sensitive/suited to the parameters in the query, and the relations and rules that are relevant to the query

  Only the relevant data in the extensional DB is considered

  We can say that MSM is a bottom-up query evaluation mechanism that in some aspects simulates a top-down approach

  In general, top-down mechanisms are more "focused" on the relevant data (as in Prolog), but less efficient, at least for DB applications

  In different forms and with different limitations, MSM are implemented in DBMS



This is achieved by using the query to rewrite the original program into a new, magic program, which has additional predicates to impose more and adequate conditions (related to the structure and parameters in the query)

The new program is used for bottom-up evaluation as before, but through a narrower cone (Ceri, Gottlob, Tanca; Springer 1990)

Datalog and Negation

Example: Extensional DB

$D = \{P(a,b), P(a,a), P(c,b), Q(a,b), Q(c,c), T(a,b), M(a,a,b)\}$

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

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

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

$S(x,y) \leftarrow M(x,y,z)$

P, Q, T, M are base tables, and S, R are intensional relations

Program uses both negation and recursion

A query on top: $Ans(x,y) \leftarrow R(x,y)$

We can try to answer it by bottom-up evaluation, as before

1. First propagate values into R using the first rule, obtaining: $R = \{(a, a), (c, b)\}$ (partial extension)

   This rule is used only once, and computes a (set) difference of relations via the not negation

   A non-monotonic negation in the sense that we do not have to prove that, e.g. $not\ Q(a, a)$ is true, but just fail to find/compute the atom $Q(a, a)$ (as with the CWA)

2. Next propagate values into S using the last rule, obtaining $S(a, a)$

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

   $(T \bowtie R)(x, y, z) = \{(\underline{a}, b, \underline{a})\}$, but $(a, a) \in S$

   The body gives no answer, at this step $R = \emptyset$

   We are left as before: $R = \{(a, a), (c, b)\}$, no new tuples, and a fixpoint is reached

4. We obtained $Ans = \{(a, a), (c, b)\}$

Notice: negation and recursion do not interact

Example: (recursion via negation) Extensional DB $Q = \{1, 2\}$

Extended with view definition:

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

Bottom-up computation of P: steps ...

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

P is defined by recursion via negation

What is the semantics (intended model) of a program like this?

Can we give to it a reasonable semantics?

The first program is stratified, with strata:

$$\begin{tikzcd}
3. & R \\
2. & S \arrow[u] \\
1. & P, Q, T, M \arrow[u]
\end{tikzcd}$$

Stratification: ordered sequence of layers (levels), each predicates in one of the layers

Mutually dependent predicates (defined in terms of each other) belong to the same stratum

If a predicate is defined in term of the negation of another predicate, the latter belongs to a lower stratum

Like a hierarchical sequence of blocks of predicates

Bottom-up evaluation follows the upwards hierarchy of strata

Each strata is computed completely before moving to the next stratum

There is always a unique fixpoint that produces the intended model of the program (and gives the semantics to it)

Datalog (no negation) and Datalog$^{str.not}$ programs have a unique model that can be computed bottom-up, and turns out to be a minimal model (it is a model and no proper subset is)

The second program is not stratified (no way to define strata with restrictions as above)

The stable model semantics gives semantics also to non-stratified programs (extending the semantics of Datalog and Datalog$^{str.not}$)

Answer Set Programs

First Normal Programs:

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

Rules may be of the form:

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

where the A, A_i are atoms

The program can be stratified or not ...

Example: Extensional DB: $D = \{U(a), U(c), V(b), V(a)\}$

$$\begin{array}{rcl} P(x) & \longleftarrow & R(x), S(x,y). \\ R(x) & \longleftarrow & U(x),\ not\ V(x). \\ S(x,y) & \longleftarrow & U(x), V(y) \end{array}$$

Herbrand Universe of the program: $HU = \{a, b, c\}$

Herbrand Base for the program: HB contains all possible ground atoms with constants from HU

$$HB = \{R(a), R(b), R(c), P(a), \ldots, S(a,a), \ldots, V(c)\}$$

A potential model $\mathcal{M}$ will be a subset of HB, telling us which atoms are true (and the others false by default)

E.g. $\{R(a), P(b), S(b,a), U(a), U(c), V(b)\}$ could be, in principle, a model (is it?)

We give a declarative semantics to normal programs through a collection of intended models ... A Possible-World Semantics!

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

Can be applied to stratified and non-stratified programs

Let Π be a normal program, and $S \subseteq HB$, i.e. a subset of the Herbrand Base, i.e. a set of ground atoms

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

S can be seen as a set of assumptions

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

The Test does the following:

- Pass from Π to Π_H, the ground instantiation of Π
- Construct a new ground program Π_H^S, depending on S as follows:

1. Delete from Π_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 Π_H^S (no negation)

- Compute $\underline{M}(\Pi_H^S)$, the minimal model of the definite program
- If $\underline{M}(\Pi_H^S) = S$, we say that S is a stable model of Π

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

Example: Non-stratified (ground) program Π: $p(a) \leftarrow \ not\, p(b)$

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

Example: Program Π (non-stratified)

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

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

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

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

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

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

$q(a, b).$

Π_H^S :

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

Minimal model of Π_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 Π (non-stratified)

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

If $S_1 = \{person(a), male(a)\}$, then Π^{S_1} is:

$$\begin{aligned} male(a) \leftarrow\ & person(a) \\ person(a) & \end{aligned}$$

It has $\{persona, male(a)\}$ as minimal model, then, S_1 is a stable model of Π

$S_2 = \{person(a), female(a)\}$ is also a stable model of Π

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

Exercise: Check that $\emptyset$, $\{person(a), male(a), female(a)\}$ are not stable models

Some Results:

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

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

- The semantics is non-monotonic: additional facts/rules may invalidate previous conclusions (cf. below for "conclusions")

- 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)

- We are giving a declarative semantics to a wider class of programs (with or w/o negation), even non-stratified ones
- Consequences from normal programs with stable model semantics?
  - 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 Π is definite or normal, but stratified, then it has a unique stable model

  In this case the only stable model coincides with the minimal (positive Datalog program) or standard model (stratified Datalog program)

  This SM 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

Disjunctive, Negation-Free Programs:

We now consider more expressive logic programs containing clauses of the form

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

where the A_i, B_j are atoms

Example: Extensional DB: $D = \{U(a), U(c), V(b), V(a)\}$

$$\begin{aligned} P(x) \vee Q(x) \quad &\longleftarrow \quad R(x), S(x,y) \\ R(x) \quad &\longleftarrow \quad U(x), V(x) \\ S(x,y) \quad &\longleftarrow \quad U(x), V(y) \end{aligned}$$

$HU = \{a, b, c\}$, $HB = \{Q(a), Q(b), \ldots\}$

Now, two minimal (Herbrand) models:

$$\begin{aligned} \mathcal{M}_1 = \{&U(a), U(c), V(b), V(a), S(a,a), S(a,b), S(c,a), S(c,b), \\ &R(a), P(a)\} \end{aligned}$$

$$\mathcal{M}_2 = \{U(a), U(c), V(b), V(a), S(a,a), S(a,b), S(c,a),$$
$$S(c,b), R(a), Q(a)\}$$

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

Under this minimal model semantics, disjunctions in rules are interpreted, unless otherwise implied by the other program rules, in an exclusive manner

Computation can be bottom-up, making true one chosen atom from the disjunctive head, in turns

Gives rise to a branching process ...

Disjunctive Normal Programs: (Gelfond & Lifschitz; 1990)

$A_1 \vee \cdots \vee A_n \;\longleftarrow\; B_1, \ldots, B_n$

A_i: positive $\quad$ B_j: literals, possibly with $\;not$

Example: Extensional DB: $D = \{U(a), U(c), V(b), V(a)\}$

$$\begin{array}{rcl}
P(x) \vee Q(x) & \longleftarrow & R(x), S(x,y) \\
R(x) & \longleftarrow & S(x,y),\; not\; R(y) \\
R(x) & \longleftarrow & U(x), V(x) \\
S(x,y) & \longleftarrow & U(x), V(y)
\end{array}$$

Semantics as for normal programs

Pick up a set of ground atoms S as a guess

The test is the same

After elimination of negated atom and some unsatisfiable rules (exactly as for normal programs), we are left with a disjunctive, positive Datalog program

If S is one of the minimal models of this residual program, S is a stable model

B. 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: $\mathbf{t_a}, \mathbf{f_a}$

To give feedback to the program in case additional changes become necessary: $\mathbf{t}^\star, \mathbf{f}^\star$

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

The program rules have to capture this intended semantics for annotations

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: $P(c,l,\mathbf{t_d}), P(d,m,\mathbf{t_d}), Q(d,m,\mathbf{t_d}), \ldots$
2. Whatever was true (false) or becomes true (false), gets annotated with $\mathbf{t}^\star$ $(\mathbf{f}^\star)$:

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

... the same for Q ...

3. There may be interacting ICs (not here)

   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 (the head) in case of a violation (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 program constraints to prune undesirable models, those where the body of the rule becomes true

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

   Similarly to denial constraints for DBs, they say that the conjunction in the body is not allowed

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):

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

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

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

... 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. $Q(\cdots P(\bar{u}) \cdots) \quad \longmapsto \quad Q' := \; Q(\cdots P(\bar{u}, \mathbf{t}^{\star\star}) \cdots)$

2. $Q'(\bar{x}) \quad \longmapsto \quad (\Pi(Q'), Ans(\bar{X}))$

   (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" $\quad \Pi := \; \Pi(Q') \; \cup \; \Pi(D, IC)$

4. Collect ground atoms

   $Ans(\bar{t}) \; \in \; \bigcap\{S \mid S$ 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 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]

- 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, e.g. $\forall x(P(x) \rightarrow \exists y R(x,y))$, are handled via introduction of null values or deletions
  [Barcelo,Bertossi,Bravo; LNCS 2582] [Bravo,Bertossi; IIDB'06]

- The repairs program can be seen as query rewritings into a highly expressive query language
- The expressive power comes with higher computational complexity
- Skeptical reasoning under disjunctive ASPs is Π^P_2-complete in data [Dantsin et al., ACM Comp. Surveys 2001]
- However, CQA is a Π^P_2-complete decision problem in data
- The repairs program provide the right expressive power (for the worst cases)
- For "easier" cases, other methods like FO query rewriting are more appealing ...
- Optimization of the access to the DB, to the relevant portions of it [Eiter, Fink, G.Greco, Lembo; TODS 04]

- 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 ASPs could be used for CQA [Faber et al.; ICDT 2005]
- Efficient integration of relational databases and answer set programming environments
- We have successfully experimented with the DLV system for computing the stable models semantics [Leone et al.; ACM TCL'05]
- The program can be optimized in several ways [Barcelo,Bertossi,Bravo; LNCS 2582] [Caniupan, Bertossi; DKE 2010]
- System implemented *ConsEx* [Caniupan, Bertossi; DKE 2010]

  In particular, we adapted and implemented the magic sets techniques for our purposes

Magic Sets for Repair Programs

- Consistent answers are obtained from the intersection of the stable models of the repair program plus the query program

The repair programs -and also its stable models- contain more information than necessary to answer a query

  - They consider all database predicates and database facts
  - Query predicates are related to a subset of the database predicates
  - Only a portion of the data is relevant to answer the query

- Most commonly, we are not interested in obtaining the stable models (or repairs), but in obtaining the consistent answers to particular queries

Important to consider only relevant predicates and facts in query evaluation

- Classically, magic set methods (MS) evaluate queries bottom-up, but simulating a top-down approach in terms of access to relevant data

- MS methods had been around for a while for Datalog

- Given a query and a repair program, MS selects the relevant rules from the program to compute the answers

It pushes down the query constants (if any) to restrict the tuples involved in the computation of the answer

It generates a rewritten program that has a subset of the original rules, plus a set of "magic" rules

- MS methods have been extended only recently to disjunctive ASPs [Alviano, Faber, Greco, Leone; AIJ 2012]

Our Application of MS:

[Caniupan, Bertossi; DKE 2010]

MS method for disjunctive repair programs with program constraints (PCs):

- Program $\Pi(DB, IC, \mathcal{Q}) := \Pi(DB, IC) \cup \Pi(\mathcal{Q})$.
- MS is applied over
  $\Pi^{-}(DB, IC, \mathcal{Q}) := \Pi(DB, IC, \mathcal{Q}) \smallsetminus PC$
  PC is the set of program denials of $\Pi(DB, IC, \mathcal{Q})$

At the end PC is put back into the resulting program, for the rewritten program to have only coherent models

Example: For $DB = \{S(a), T(a)\}$ and IC:

$\forall x(S(x) \rightarrow Q(x))$, $\forall x(Q(x) \rightarrow R(x))$, $\forall x(T(x) \rightarrow W(x))$

$\Pi(DB, IC, \mathcal{Q})$ is:

$S(a) \quad T(a)$

$\underline{S}(x, \mathbf{f_a}) \vee \underline{Q}(x, \mathbf{t_a}) \leftarrow \underline{S}(x, \mathbf{t}^\star), \underline{Q}(x, \mathbf{f_a})$

$\underline{S}(x, \mathbf{f_a}) \vee \underline{Q}(x, \mathbf{t_a}) \leftarrow \underline{S}(x, \mathbf{t}^\star),\ not\ Q(x)$

$\underline{Q}(x, \mathbf{f_a}) \vee \underline{R}(x, \mathbf{t_a}) \leftarrow \underline{Q}(x, \mathbf{t}^\star), \underline{R}(x, \mathbf{f_a})$

$\underline{Q}(x, \mathbf{f_a}) \vee \underline{R}(x, \mathbf{t_a}) \leftarrow \underline{Q}(x, \mathbf{t}^\star),\ not\ R(x)$

$\underline{T}(x, \mathbf{f_a}) \vee \underline{W}(x, \mathbf{t_a}) \leftarrow \underline{T}(x, \mathbf{t}^\star), \underline{W}(x, \mathbf{f_a})$

$\underline{T}(x, \mathbf{f_a}) \vee \underline{W}(x, \mathbf{t_a}) \leftarrow \underline{T}(x, \mathbf{t}^\star),\ not\ W(x)$

$\underline{S}(x, \mathbf{t}^\star) \leftarrow \underline{S}(x, \mathbf{t_a})$ (same for Q,R,T,W)

$\underline{S}(x, \mathbf{t}^\star) \leftarrow S(x)$ (same for Q,R,T,W)

$\underline{S}(x, \mathbf{t}^{\star\star}) \leftarrow \underline{S}(x, \mathbf{t}^\star), not\ \underline{S}(x, \mathbf{f_a})$ (same for Q,R,T,W)

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

$\mathcal{Q} : Ans(x) \leftarrow \underline{S}(x, \mathbf{t}^{\star\star})$

MS follows three steps: [Cumbo,Faber,Greco,Leone; ICLP 04]

1. Adornment: The materialization of binding information by adornments: $\boldsymbol{b}$ means bound, $\boldsymbol{f}$ stands for free

Starting from the query: $Ans(x) \leftarrow S(x, \mathbf{t}^{\star\star})$

The adorned rule is: $Ans^{f}(x) \leftarrow S^{fb}(x, \mathbf{t}^{\star\star})$

Adorned predicate S^{fb} propagate bindings to rules defining S:

- $S(x, \mathbf{f_a}) \vee Q(x, \mathbf{t_a}) \leftarrow S(x, \mathbf{t}^{\star}), Q(x, \mathbf{f_a})$

  $S^{fb}(x, \mathbf{f_a}) \vee Q^{fb}(x, \mathbf{t_a}) \leftarrow S^{fb}(x, \mathbf{t}^{\star}), Q^{fb}(x, \mathbf{f_a})$

  The adorned predicate Q^{fb} has to be processed similarly

Output: An adorned program

2. Generation of Magic Rules: A magic rule is generated for each adorned atom in the body of adorned rules

For the adorned rule: $S^{fb}(x, \mathbf{t}^{\star}) \leftarrow S^{fb}(x, \mathbf{t_a})$

The magic rule is: $magic\_S^{fb}(\mathbf{t_a}) \leftarrow magic\_S^{fb}(\mathbf{t}^{\star})$.

Output: Set of magic rules

3. Modification: Each adorned rule is modified by inserting the magic version of its head into the body

Rule: $Ans(x) \leftarrow S(x, \mathbf{t}^{\star\star})$ becomes: $Ans(x) \leftarrow magic\_Ans^{f}, S(x, \mathbf{t}^{\star\star})$.

Output: Set of modified rules

In order to evaluate them, the magic rules defined before have to be invoked

The rewritten program $\mathcal{MS}^{\leftarrow}(\Pi(DB, IC, \mathcal{Q}))$ consists of:

- The *magic rules*, and the *modified rules*
- The set of program denial constraints PC
- The query including the atom $magic\_Ans^{f}$ (the magic version of the Ans predicate)

The stable models of the rewritten program are (displayed without the magic atoms):

$$\begin{aligned}\mathcal{M}_1 &= \{S(a), T(a), \underset{\sim}{S}(a, \mathbf{t}^{\star}), \underset{\sim}{Q}(a, \mathbf{t_a}), \underset{\sim}{S}(a, \mathbf{t}^{\star\star}), \underset{\sim}{Q}(a, \mathbf{t}^{\star}), \underset{\sim}{R}(a, \mathbf{t_a}), \underline{Ans(a)}\}\\ \mathcal{M}_2 &= \{S(a), T(a), \underset{\sim}{S}(a, \mathbf{t}^{\star}), \underset{\sim}{S}(a, \mathbf{f_a})\}\end{aligned}$$

The original program had four stable models!

Only the relevant models to answer the query are computed!

And only partially computed!

The original query

$$\mathcal{Q}: \quad Ans(\S) \leftarrow \mathcal{S}(\S, \mathbf{t}^{\star\star})$$

becomes

$$Ans(x) \leftarrow magic\_Ans^{f}, \mathcal{S}(x, \mathbf{t}^{\star\star})$$

And is "run" in $\mathcal{MS}^{\leftarrow}(\Pi(DB, IC, \mathcal{Q}))$

No cautious answers in this case

That is, no consistent answers then

Which is correct in this case ...

Example: $D = \{S(a,c), S(b,c), R(b,c), T(a, null), W(null, b, c)\}$

IC: $\{\forall xy(S(x,y) \rightarrow R(x,y)), \forall xy(T(x,y) \rightarrow \exists z W(x,y,z)),$
$\forall xyz(W(x,y,z) \wedge IsNull(x) \rightarrow \mathbf{false})\}$

$\Pi(D, IC, \mathcal{Q})$ is: $D = \{\ S(a,c), S(b,c), R(b,c), T(a, null), W(null, b, c)\}$
plus rules for IC handling[2]

$$\underline{S}(x,y,\mathbf{f_a}) \vee \underline{R}(x,y,\mathbf{t_a}) \leftarrow \underline{S}(x,y,\mathbf{t}^\star), \underline{R}(x,y,\mathbf{f_a}), x \neq null, y \neq null$$
$$\underline{S}(x,y,\mathbf{f_a}) \vee \underline{R}(x,y,\mathbf{t_a}) \leftarrow \underline{S}(x,y,\mathbf{t}^\star),\ not\ R(x,y), x \neq null, y \neq null$$

$$\underline{T}(x,y,\mathbf{f_a}) \vee \underline{W}(x,y,null,\mathbf{t_a}) \leftarrow \underline{T}(x,y,\mathbf{t}^\star),\ not\ aux(x,y), x \neq null, y \neq null$$
$$aux(x,y) \leftarrow \underline{W}(x,y,z,\mathbf{t}^\star),\ not\ \underline{W}(x,y,z,\mathbf{f_a}), x \neq null, y \neq null, z \neq null$$

$$\underline{W}(x,y,z,\mathbf{f_a}) \leftarrow \underline{W}(x,y,z,\mathbf{t}^\star), x = null$$

[2] Conditions $x \neq null$, etc., are needed to handle nulls, in original data and in inserted tuples for existential ICs

Plus annotation rules:

$$\left.\begin{array}{l} S\_(x,y,\mathbf{t}^\star) \leftarrow S\_(x,y,\mathbf{t_a}) \\ S\_(x,y,\mathbf{t}^\star) \leftarrow S(x,y) \\ S\_(x,y,\mathbf{t}^{\star\star}) \leftarrow S\_(x,y,\mathbf{t}^\star),\ \ not\ S\_(x,y,\mathbf{f_a}) \end{array}\right\} \quad \text{(similarly for } R, T \text{ and } W\text{)}$$

$$\leftarrow W\_(x,y,z,\mathbf{t_a}), W\_(x,y,z,\mathbf{f_a})$$

$$Ans(x) \leftarrow S\_(b,x,\mathbf{t}^{\star\star})$$

The stable models of $\Pi(D, IC, \mathcal{Q})$ are:

$$\begin{array}{rcl} \mathcal{M}_1 & = & \{S\_(a,c,\mathbf{t}^\star), S\_(b,c,\mathbf{t}^\star), R\_(b,c,\mathbf{t}^\star), T\_(a,null,\mathbf{t}^\star), W\_(null,b,c,\mathbf{t}^\star), \\ & & W\_(null,b,c,\mathbf{f_a}), R\_(a,c,\mathbf{t_a}), S\_(a,c,\mathbf{t}^{\star\star}), S\_(b,c,\mathbf{t}^{\star\star}), R\_(b,c,\mathbf{t}^{\star\star}), \\ & & R\_(a,c,\mathbf{t}^\star), R\_(a,c,\mathbf{t}^{\star\star}), T\_(a,null,\mathbf{t}^{\star\star}), Ans(c)\} \end{array}$$

$$\begin{array}{rcl} \mathcal{M}_2 & = & \{S\_(a,c,\mathbf{t}^\star), S\_(b,c,\mathbf{t}^\star), R\_(b,c,\mathbf{t}^\star), T\_(a,null,\mathbf{t}^\star), W\_(null,b,c,\mathbf{t}^\star), \\ & & W\_(null,b,c,\mathbf{f_a}), S\_(a,c,\mathbf{f_a}), S\_(b,c,\mathbf{t}^{\star\star}), R\_(b,c,\mathbf{t}^{\star\star}), T\_(a,null,\mathbf{t}^{\star\star}), \\ & & Ans(c)\} \end{array}$$

The consistent answer to $\mathcal{Q}\colon\ Ans(x) \leftarrow S(b,x)$ is (c)

MS method for disjunctive repair programs with program constraints is applied over $\Pi^{-}(D, IC, \mathcal{Q}) := \Pi(D, IC, \mathcal{Q}) \smallsetminus PC$

$\Pi(D, IC, \mathcal{Q}) := \Pi(D, IC) \cup \Pi(\mathcal{Q})$ and PC the set of program constraints of $\Pi(D, IC, \mathcal{Q})$

After following three steps MS produces a rewritten program

The rewritten program $\mathcal{MS}^{\leftarrow}(\Pi(D, IC, \mathcal{Q}))$ consists of:

- The set of *magic rules*,
- The set of *modified rules*,
- Reinserting the rules in PC

Query $Ans(x) \leftarrow S\_(b, x, \mathbf{t}^{\star\star})$ becomes $Ans^{f}(x) \leftarrow S\_^{bfb}(b, x, \mathbf{t}^{\star\star})$

The adorned atom $S\_^{bfb}(b, x, \mathbf{t}^{\star\star})$ produces adornment over rules defining predicates $S\_$:[3]

(1) $S\_^{bfb}(x, y, \mathbf{t}^{\star}) \leftarrow S(x, y)$
(2) $S\_^{bfb}(x, y, \mathbf{t}^{\star}) \leftarrow S\_^{bfb}(x, y, \mathbf{t_a})$
(3) $S\_^{bfb}(x, y, \mathbf{f_a}) \vee R\_^{bfb}(x, y, \mathbf{t_a}) \leftarrow S\_^{bfb}(x, y, \mathbf{t}^{\star}), R\_^{bfb}(x, y, \mathbf{f_a})$
...
New magic rules: for adorned atom $S\_^{bfb}(x, y, \mathbf{t_a})$ in the body of the adorned rule 2:

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

Modified rule:

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

[3] For readability, we omit conditions with *null*

Program $\mathcal{MS}(\Pi)$

$magic\_Ans^f.$
$magic\_S\_^{bfb}(b, \mathbf{t}^{\star\star}) \leftarrow magic\_Ans^f.$
$magic\_S\_^{bfb}(x, \mathbf{t_a}) \leftarrow magic\_S\_^{bfb}(x, \mathbf{t}^{\star}).$
$magic\_S\_^{bfb}(x, \mathbf{t}^{\star}) \leftarrow magic\_S\_^{bfb}(x, \mathbf{t}^{\star\star}).$
$magic\_S\_^{bfb}(x, \mathbf{f_a}) \leftarrow magic\_S\_^{bfb}(x, \mathbf{t}^{\star\star}).$
$magic\_R\_^{bfb}(x, \mathbf{t_a}) \leftarrow magic\_S\_^{bfb}(x, \mathbf{f_a}).$
$magic\_S\_^{bfb}(x, \mathbf{t}^{\star}) \leftarrow magic\_S\_^{bfb}(x, \mathbf{f_a}).$
$magic\_R\_^{bfb}(x, \mathbf{f_a}) \leftarrow magic\_S\_^{bfb}(x, \mathbf{f_a}).$
$magic\_S\_^{bfb}(x, \mathbf{f_a}) \leftarrow magic\_R\_^{bfb}(x, \mathbf{t_a}).$
$magic\_S\_^{bfb}(x, \mathbf{t}^{\star}) \leftarrow magic\_R\_^{bfb}(x, \mathbf{t_a}).$
$magic\_R\_^{bfb}(x, \mathbf{f_a}) \leftarrow magic\_R\_^{bfb}(x, \mathbf{t_a}).$
$magic\_R\_^{bfb}(x, \mathbf{t_a}) \leftarrow magic\_R\_^{bfb}(x, \mathbf{t}^{\star}).$
$magic\_R\_^{bfb}(x, \mathbf{t}^{\star}) \leftarrow magic\_R\_^{bfb}(x, \mathbf{t}^{\star\star}).$
$magic\_R\_^{bfb}(x, \mathbf{f_a}) \leftarrow magic\_R\_^{bfb}(x, \mathbf{t}^{\star\star}).$
$Ans(x) \leftarrow magic\_Ans^f, S\_(b, x, \mathbf{t}^{\star\star}).$

$S\_(x, y, \mathbf{f_a}) \vee R\_(x, y, \mathbf{t_a}) \leftarrow magic\_S\_^{bfb}(x, \mathbf{f_a}), magic\_R\_^{bfb}(x, \mathbf{t_a}), S\_(x, y, \mathbf{t}^{\star}), R\_(x, y, \mathbf{f_a}).$

$S\_(x, y, \mathbf{f_a}) \vee R\_(x, y, \mathbf{t_a}) \leftarrow magic\_S\_^{bfb}(x, \mathbf{f_a}), magic\_R\_^{bfb}(x, \mathbf{t_a}), S\_(x, y, \mathbf{t}^{\star}),\ not\ R(x, y).$

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

$S\_(x, y, \mathbf{t}^{\star}) \leftarrow magic\_S\_^{bfb}(x, \mathbf{t}^{\star}), S(x, y).$

$R\_(x, y, \mathbf{t}^\star) \leftarrow magic\_R\_^{bfb}(x, \mathbf{t}^\star), R\_(x, y, \mathbf{t_a})$.

$R\_(x, y, \mathbf{t}^\star) \leftarrow magic\_R\_^{bfb}(x, \mathbf{t}^\star), R(x, y)$.

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

$R\_(x, y, \mathbf{t}^{\star\star}) \leftarrow magic\_R\_^{bfb}(x, \mathbf{t}^{\star\star}), R\_(x, y, \mathbf{t}^\star),\ not\ R\_(x, y, \mathbf{f_a})$.

$\leftarrow W\_(x, y, z, \mathbf{t_a}), W\_(x, y, z, \mathbf{f_a})$.

Program $\mathcal{MS}^{\leftarrow}(\Pi(D, IC, \mathcal{Q}))$ has one stable model (displayed here without magic atoms):

$$\mathcal{M}_1 = \{S\_(b, c, \mathbf{t}^\star), S\_(b, c, \mathbf{t}^{\star\star}), Ans(c)\}$$

The original program has two stable models!

Only the relevant models to answer the query are computed!

They are only partially computed!

The consistent answer to query: $Ans(x) \leftarrow S\_(b, x, \mathbf{t}^{\star\star})$ is (c)

Expressed in $\mathcal{MS}^{\leftarrow}(\Pi(D, IC, \mathcal{Q}))$ as:

$Ans(x) \leftarrow magic\_Ans^f, S\_(b, x, \mathbf{t}^{\star\star})$

Some results:

- For general disjunctive programs with constraints, MS does not always produce an equivalent rewritten program

  [Greco, G. et al.; TPLP'05]

- Two programs are cautiously equivalent wrt a query Q, if for any set of facts, they produce the same cautious answers to Q

  Theorem: Programs $\mathcal{MS}^{\leftarrow}(\Pi(D, IC, \mathcal{Q}))$ and $\Pi(D, IC, \mathcal{Q})$ are query equivalent under the cautious semantics

- MS can be used to evaluate disjunctive repair programs with program constraints

The implemented MS technique considerably improves response time in comparison with the use of the original repair program!

Consistent Answers in Virtual Data Integration?

ICs on the global schema have to be imposed at query answering time

Data and the sources may be "inconsistent" with global ICs

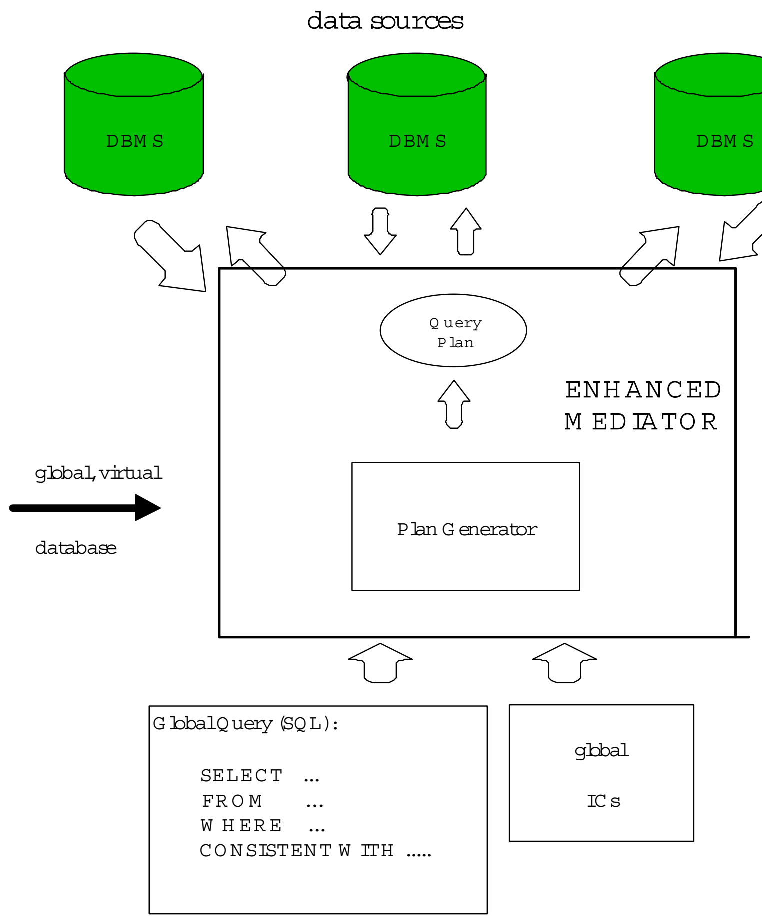
CQA can be applied in this case

This applies to GAV and LAV

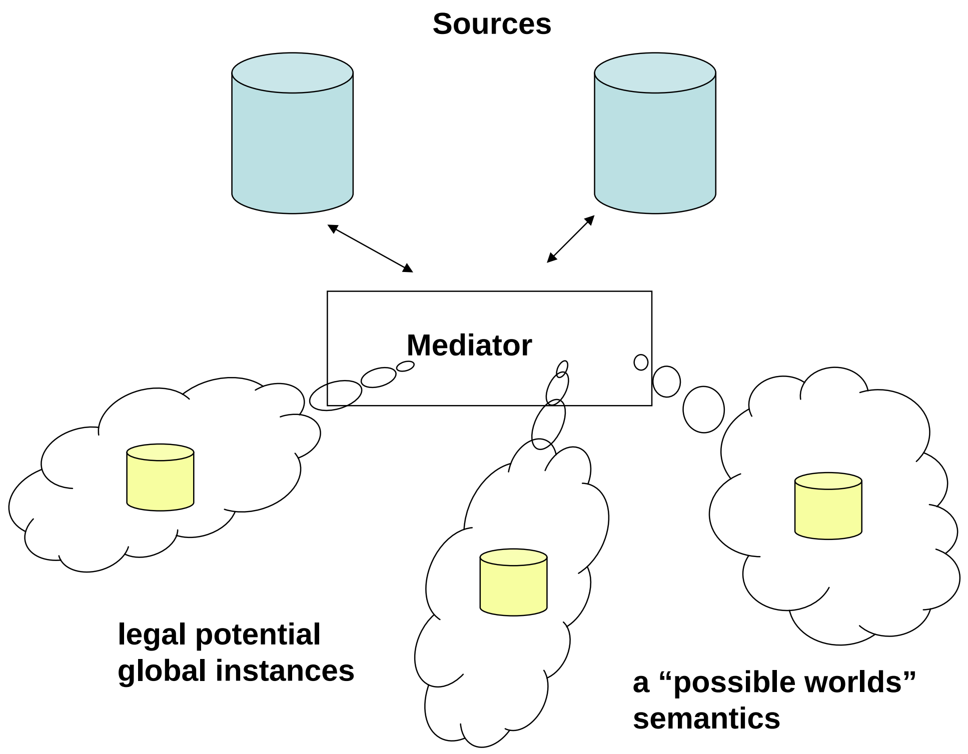
We assume open sources, etc. [Lenzerini; Pods'02]

For GAV, a single legal instance that can be specified in Datalog

For LAV, possible multiple legal instances


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

The semantics of the VDIS is given in terms of a collection of inten [illegible]


Sources
Mediator
legal potential global instances
a “possible worlds” semantics

Each of them would give extensions to the global DB predicates if they were materialized

Some of the intended, legal global instances are minimal: they do not properly contain any other legal instance

Class of minimal legal instances can be specified with DASPs

Virtual repairs are repairs of also virtual minimal legal instances of VDIS

Our Approach:

- Answer set programming (ASP) based specification of minimal legal instances of a VDIS

  For LAV, not shown here (Bravo, Bertossi; IJCAI 03)

  (Bertossi, Bravo; Springer LNCS 3300, 2004)

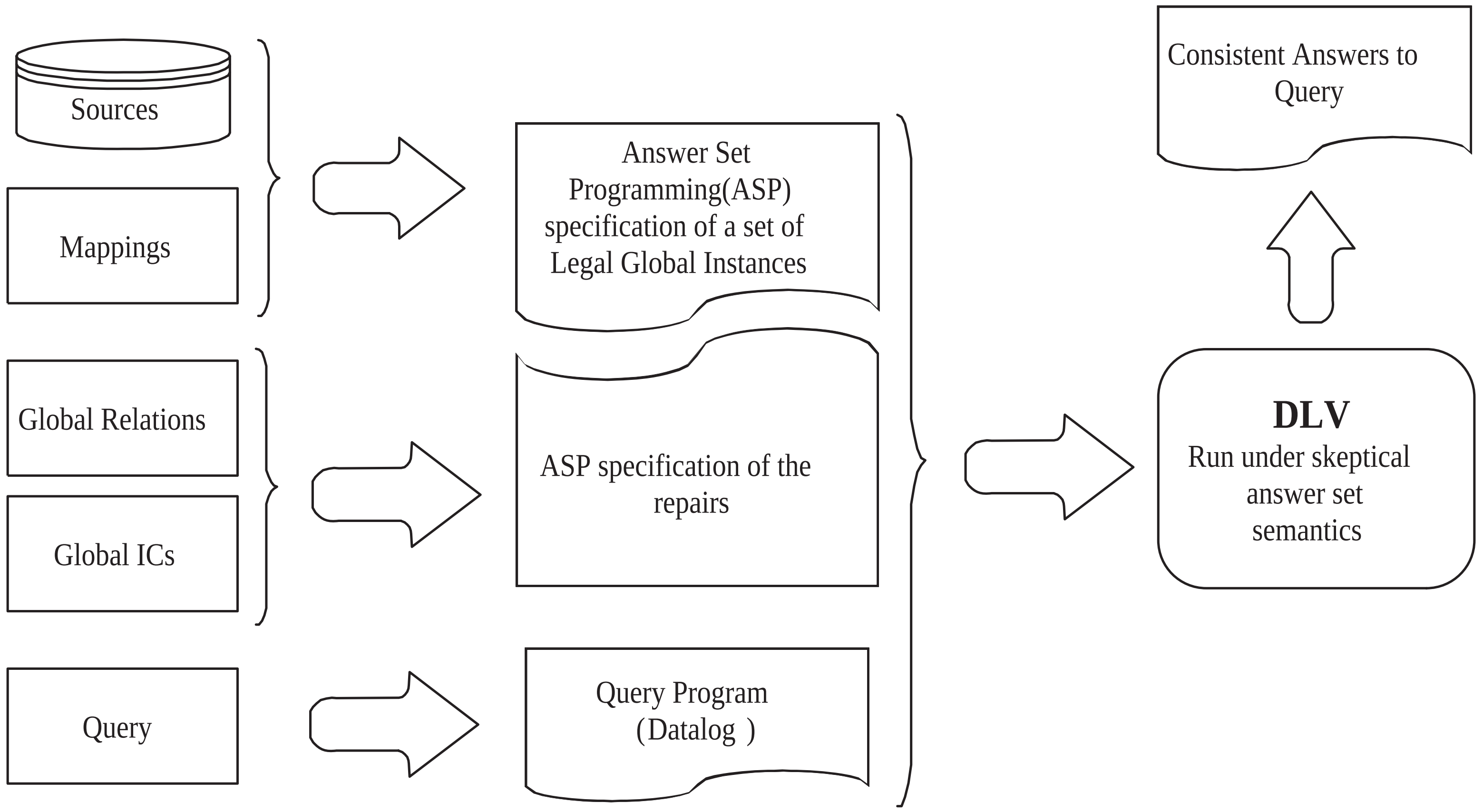
- ASP based specification of repairs of minimal instances

  Repair programs as before ...

- Global query in Datalog (or its extensions) to be answered consistently

  Non-recursive (and stratified) if original query is FO

- Run combined programs above under skeptical stable model semantics


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

Some Alternative Developments and Extensions of the CQA Semantics[1]

Leopoldo Bertossi

Carleton University
School of Computer Science
Ottawa, Canada

[1]Chapter 4 of PhD Course at U. Rome, “La Sapienza”, 2013.

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, datawarehousing

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

This seems to be quite a general situation ...

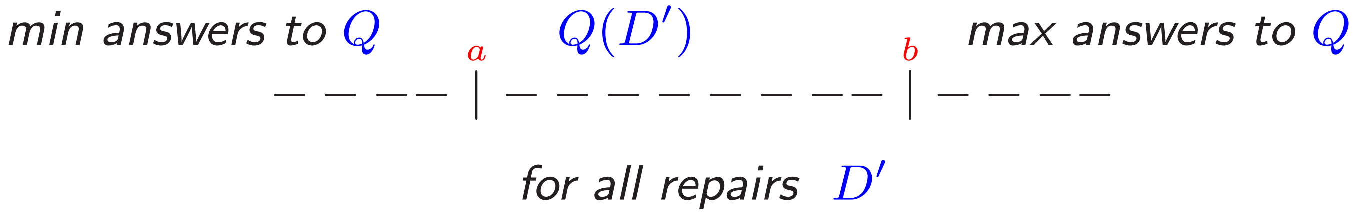
So, we modify the definition of consistent answer

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:

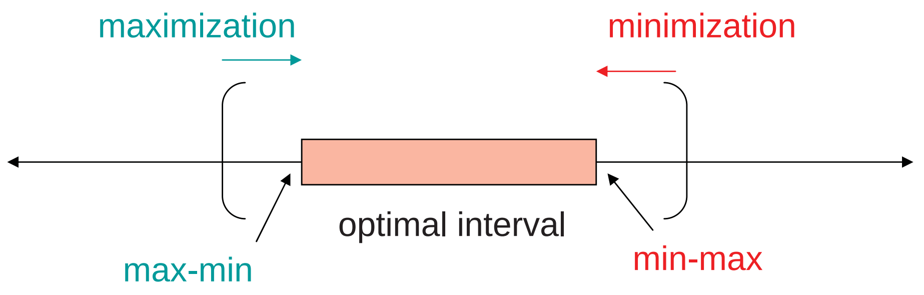
The *max-min answer* a, and the *min-max answer* b

By querying D only!

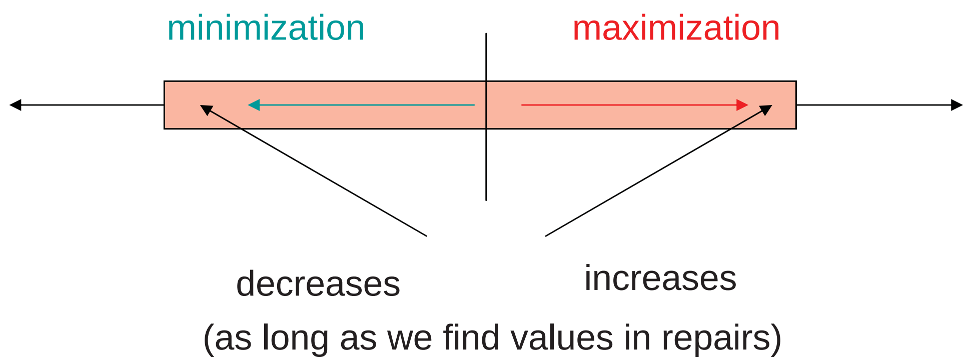
Sometimes we are interested in one of the two only, e.g.

- In max-min for `MIN(Amount)`
  The max-min is always the min in original instance
  In the example, for `MIN(Amount)`: $[3000]$
- In min-max for `MAX(Amount)`
  The min-max is always the max in the original instance
  In the example, for `MAX(Amount)`: $[7000, 8000]$

Finding the extremes of the interval becomes two optimization problems over the class of repairs

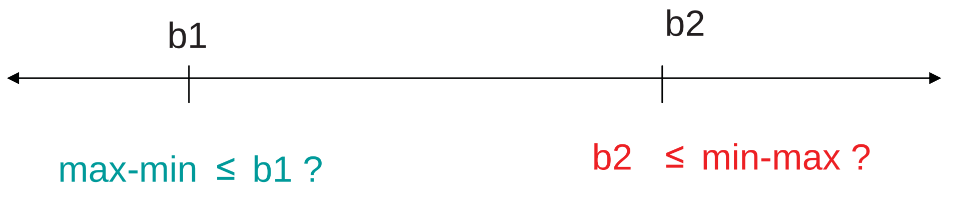


or the other way around



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

associated decision problems



Which give rise, as usual, to two corresponding decision problems, in terms of bounds or budgets $(b1, b2$ above)

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

Particularly useful is a graph-based representation of repairs

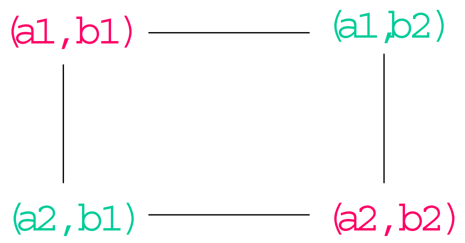
The undirected conflict graph $CG_{FD}(D)$ associated to set FD and instance D:

- 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 $\bar{t}_1, \bar{t}_2$

Graph algorithms ca be applied ...

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 (graph) of $CG_{FD}(D)$

Some Complexity Results

- `MAX(A)` can be different in every repair

However, 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(Y)` for the FD $R\colon\ X \rightarrow Y$

Create and pose a sequence of SQL queries to the inconsistent database: query rewriting using aggregate views

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;
```

In the end, a new query in FO plus aggregation ...

- For more than one FD, the problem of deciding whether "max-min answer to `MAX(A)` $\leq k$" is NP-complete (corresponding to associated minimization problem)

NP-hardness: By reduction from SAT

Being data complexity, fix the schema and FDs: $T(X, Y, Z, W)$ with FDs: $X \rightarrow Y$ and $Z \rightarrow W$ (the query is already fixed)

Now, we are given a propositional formula φ in CNF $C_1 \wedge \cdots \wedge C_n$ with propositional variables $p_1, \ldots p_m$

We create an instance D for the schema above, with 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. $(x, x, C_j, 2)$, $1 \leq j \leq n$, with x a new symbol

Due to first FD, each propositional variable cannot have more than one truth value

The instance can be efficiently built from φ

Furthermore, it holds:

φ is satisfiable iff for D and $k = 1$, the answer to "max-min `MAX(W)` ≤ 1" is Yes

That is, if there is a repair that never takes the "scape value" 2, i.e. one that represents a satisfying truth assignment

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

It is feasible to check whether D' is a repair of D (satisfaction of FDs plus minimality) and `MAX(W)` $\leq k$ in D'

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

(Essentially the same proof can be used to obtain that consistent conjunctive query answering (without aggregation) wrt FDs is $coNP$-complete)

- Even for one FD, the problem of deciding if the maximal min-answer to `COUNT(A)` $\leq k$ is NP-complete

(reduction from HITTING SET)

	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; TCS 2003)

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

- We have identified normalization conditions, e.g. BCNF, and other conditions on the DB, e.g. producing a conflict graph with a certain structure, under which more efficient algorithms can be designed

However, CQA for aggregate queries stays as an intrinsically complex problem

- It seems necessary to efficiently and optimally approximate consistent answers to aggregate queries

However, "maximal independent set" seems to have bad approximation properties

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

Further developments:

- Aggregate conjunctive queries with group-by, also with range semantics (Fuxman, Miller; Sigmod'05)
- The range semantics is used in answering aggregate queries in data exchange (Afrati; Kolaitis; Pods'08)
- CQA on numerical databases with aggregation constraints
  Non-aggregate queries, no range semantics
  Repairs minimize number of changes of attribute values
  (Flesca, Furfaro, Parisi; DBPL'05)
- Numerical databases, denial constraints, changes of numerical attribute values, numerical distance minimized, aggregate queries (Bertossi, Bravo, Fanconi, Lopatenko; IS 2008)

Numerical databases open ground for interesting research on repairs and CQA ...

Fixing Numerical Attributes in DBs under ICs

(Bertossi, Bravo, Franconi, Lopatenko; IS'08)

Motivation I:

- Most of the research on CQA has concentrated on repairs of databases that minimize under set inclusion the set of insertions/deletions of whole database tuples (no matter what type of data they contain)

In some scenarios or applications this repair semantics may not be the most natural

- No much interest in the repairs *per se* or their computation (unless necessary), but in CQA

- Interest in census-like data, where other ontological assumptions, fixes, and computational needs appear

(Franconi,Laureti Palma, Leone, Perri, Scarcello; LPAR 01)

- Many attributes take numerical values
- Most natural form of fix is to change some of the values of attributes in tuples (Wijsen; ICDT 03)
- Tuples identified by a key (e.g. household identifier) that is fixed and not subject to changes

  Identifiers are kept in any repaired version of the database (census form)
- Constraints are expressed as prohibitions of combinations of positive data, i.e. denial constraints
- Few relations in the database, actually not uncommon to have a single relation
- Computation of fixes (repairs) becomes crucial and a common task in census like applications
- Aggregate queries are most important, rather than queries about individual properties

Check out:

United Nations Economic Commission for Europe, Work Session on Statistical Data Editing (Ottawa, 16-18 May 2005)

`http://www.unece.org/stats/documents/2005.05.sde.htm`

For a glossary of terms and computational tasks related to data editing

`http://www.unece.org/stats/publications/editingglossary.pdf`

The Problem:

We propose a notion of fix (repair) of a database containing numerical, integer values that:

- Allows, as basic fixing actions, changes of values in (changeable or flexible) attributes
- For the first time takes into account the occurrence of numerical values and their nature when distances between databases are measured in numerical terms
- Considers the presence of non-contradictable key constraints

In this scenario, completely new issues and problems appear:

- Computing database fixes
- Determining existence of fixes, e.g. not beyond a certain distance from the original database
- CQA to aggregate queries wrt to denial constraints

Example 1: A denial constraint DC: *Pay at most* 6000 *to employees with less than 5 years of experience*

Database D, with Name as the key, is inconsistent wrt DC

Employee	Name	Experience	Salary
	Sarah	6	12000
	Robert	4	7000
	Daniel	5	8000

Under tuple and set oriented semantics for repairs, the only minimal repair corresponds to deleting tuple Employee(Robert, 4, 7000)

Other options may make more sense that deleting employee Robert (a value for the key):

- Change the violating tuple to Employee(Robert,5, 7000)
- Change it to Employee(Robert, 4, 6000)

They satisfy the implicit requirements that:

- Key values are kept
- Numerical values associated to them do not change too much

Minimum Numerical Fixes:

We concentrate on linear denial constraints (LDCs) of the form

$$\forall \bar{x} \neg (A_1 \wedge \ldots \wedge A_m)$$

- A_i are database atoms, or built-in atoms of the form $X \theta c$, with $\theta \in \{=, \neq, <, >, \leq, \geq\}$, or $X = Y$
  (the latter can be replaced by different occurrences of the same variable)
- If we allow atoms of the form $X \neq Y$, we call them extended linear denial constraints (ELDs)

Examples: *Pay at most* 6000 *to employees with less than 5 years of experience*

$$\forall Name, Experience, Salary \neg (Employee(Name, Experience, Salary),$$
$$Experience < 5, Salary > 6000)$$

Distance between Instances:

- When numerical values are updated to restore consistency, it is desirable to make the smallest overall variation of the original values
- A database and a fix share the same key values
- Key values can be used to compute associated numerical variations

To compare instances, we need a common relational schema $\mathcal{R}$, a set of key constraints $\mathcal{K}$, assumed to be satisfied by all the instances, the set of attributes $\mathcal{A}$, a subset of fixable, numerical attributes $\mathcal{F}$ (not containing any attributes in the key)

For a tuple $\bar{k}$ of key values in relation R in instance D, $\bar{t}(\bar{k}, R, D)$ denotes the unique tuple $\bar{t}$ in relation R in instance D whose key value is $\bar{k}$

Example 1: (cont.) $\mathcal{R} = \{Employee\}$, $\mathcal{A} = \{Name, Experience, Salary\}$, $\mathcal{F} = \{Experience, Salary\}$ $\quad Key(Employee) = \{Name\}$

Employee	<u>Name</u>	Experience	Salary
	Sarah	6	12000
	Robert	4	7000
	Daniel	5	8000

The (kept) key values are: $Sarah,\ Robert,\ Daniel$

$\bar{t}(Sarah, Employee, D) = (Sarah, 6, 12000)$, etc.

Example 2: D has tables $Client(\underline{ID}, A, M)$, A is age, M is money; and $Buy(\underline{ID, I}, P)$, I is items, P is price

Denials: *People younger than 18 cannot spend more than 25 on one item nor spend more than 50 in the store*

$IC_1: \forall ID, P, A, M \neg(Buy(ID, I, P), Client(ID, A, M), A < 18, P > 25)$

$IC_2: \forall ID, A, M \neg(Client(ID, A, M), A < 18, M > 50)$

D:

Client	**$\underline{\text{ID}}$**	**A**	**M**	
	1	15	52	t_1
	2	16	51	t_2
	3	60	900	t_3
Buy	**$\underline{\text{ID}}$**	**$\underline{\text{I}}$**	**P**	
	1	CD	27	t_4
	1	DVD	26	t_5
	3	DVD	40	t_6

IC_1 is violated by $\{t_1, t_4\}$ and $\{t_1, t_5\}$; IC_2 by $\{t_1\}$ and $\{t_2\}$

$\bar{t}((1, CD), Buy, D) = Buy(1, CD, 27)$, etc. $\mathcal{F} = \{A, M, P\}$

For instances D, D' over the same schema and same key values for each relation $R \in \mathcal{R}$, their square distance is

$$\Delta(D, D') = \sum \alpha_A[\pi_A(\bar{t}(\bar{k}, R, D)) - \pi_A(\bar{t}(\bar{k}, R, D'))]^2$$

- Sum is over all relations R, fixable numerical attributes $A \in \mathcal{F}$ and tuples of key values $\bar{k}$
- π_A is the projection on attribute A
- α_A is the -application dependent- weight of attribute A

Other numerical distance functions can be considered, e.g. "city distance", obtaining results similar to those that follow

Given an instance D, with $D \models \mathcal{K}$, and a set of possibly violated ELDCs IC, a least-squares fix (LS-fix) for D wrt IC is an instance D' with:

1. Same schema and domain as D
2. Same values as D in the non-fixable attributes $\mathcal{A} \smallsetminus \mathcal{F}$ (in particular in key attributes)
3. $D' \models \mathcal{K} \cup IC$;
4. $\Delta(D, D')$ is minimum over all the instances that satisfy 1. - 3.

$LS\text{–}Fix(D, IC) := \{D' \mid D' \text{ is an LS-fix of } D \text{ wrt } IC\}$

$Fix(D, IC)$ defined as $LS\text{–}Fix(D, IC)$, but without minimality condition

Example 1: (cont.) Candidates to fixes were

$$D_1 = \{(Sarah, 6,\ 12000), (Robert, 5,\ 7000),\\ (Daniel, 5,\ 8000)\}$$

$$D_2 = \{(Sarah, 6, 12000), (Robert, 4,\ 6000),\\ (Daniel, 5,\ 8000)\}$$

With $\alpha_{Salary} = 10^{-6}$, $\alpha_{Experience} = 1$:

Square distances to D: $\Delta(D, D_1) = 1$, $\Delta(D, D_2) = 4$

Only D_1 is an LS-fix

Example 2: (cont.)

$IC_1: \forall ID, P, A, M \neg(Buy(ID, I, P), Client(ID, A, M), A < 18, P > 25)$

$IC_2: \forall ID, A, M \neg(Client(ID, A, M), A < 18, M > 50)$

D:

Client	**ID**	**A**	**M**	
	1	15	52	t_1
	2	16	51	t_2
	3	60	900	t_3
Buy	**ID**	**I**	**P**	
	1	CD	27	t_4
	1	DVD	26	t_5
	3	DVD	40	t_6

Assume $\alpha_A = \alpha_M = \alpha_P = 1$

A (minimal) fix D' with $cost = 1^2 + 2^2 + 1^2 + 2^2 = 10$

Client'	ID	A	M	
	1	15	$\cancel{5}2$ 50	t_1'
	2	16	$\cancel{5}1$ 50	$t_2{}'$
	3	60	900	t_3
Buy'	**ID**	**I**	**P**	
	1	CD	$\cancel{2}7$ 25	$t_4{}'$
	1	DVD	$\cancel{2}6$ 25	$t_5{}'$
	3	DVD	40	t_6

A fix D'' with $cost = 1^2 + 3^2 = 10$

Client"	ID	A	M	
	1	$\cancel{1}5$ 18	52	$t_1{}''$
	2	16	$\cancel{5}1$ 50	$t_2{}''$
	3	60	900	t_3
Buy"	**ID**	**I**	**P**	
	1	CD	27	t_4
	1	DVD	26	t_5
	3	DVD	40	t_6

In this example, it was possible to obtain LS-fixes by performing direct, local changes in the original conflictive tuples alone

No new, intermediate inconsistencies introduced in the repair process

This may not be always the case ...

Decidability and Complexity of LS-fixes:

It is relevant to ask about the existence of fixes

In contrast to the “classical” case of CQA, there may be no fixes

We concentrate on data complexity and denial constraints

The number of fixes can be exponential, actually

For (E)LDCs:

$NE(IC) := \{D \mid Fix(D, IC) \neq \emptyset\}$ is NP-complete

The Database Fix Problem:

$DFP(IC)$:=$\{(D,k)|$ there is $D' \in Fix(D, IC)$ with $\Delta(D,D') \leq k\}$, the *database fix problem*

It is important for transformation of inconsistent database into the closest consistent state

- What is the distance to the closest consistent state?
- Make computations more efficient by cutting off incorrect (too expensive) branches during computation or materialization of a consistent state

Complexity of DFP provides lower bounds for CQA under some assumptions

Theorem: $DFP(IC)$, the problem of deciding whether there exists a fix wrt IC at a distance $\leq k$, is NP-complete

Hardness: By encoding "Vertex Cover"

One LDC with three database atoms plus two built-ins is good enough

Membership:

- Possible values in repaired tuples are determined by the constants in the DCs
- $PTIME$ computation of the distance function

Approximate Solutions to DFP:

Given that $DFP(IC)$ is NP-complete, can we get a good and efficient approximate solution?

$DFOP$ denotes the optimization problem of finding the minimum distance to a fix

Theorem: For a fixed set of LDCs, $DFOP$ is $MAXSNP$-hard

Proof by reduction from “Minimum Vertex Cover Problem for Graphs of Bounded Degree” which is $MAXSNP$-complete

$MAXSNP$-hardness implies (unless $P = NP$) that there exists constant δ such that $DFOP$ for LDCs cannot be approximated within an approximation factor less than δ

Can we provide an approximation within a constant factor, possibly with restriction to a still useful but hard class of LDCs?

Definition: A set of LDCs IC is local if:

- Equalities between attributes and joins involve only non-fixable attributes
- There is a built-in with a fixable attribute in each IC
- No attribute A appears in IC both in comparisons of the form $A < c_1$ and $A > c_2$

Example 2: (cont.) The LDCs there are local

$$IC_1 : \forall ID, P, A, M \neg(Buy(ID, I, P), Client(ID, A, M), A < 18, P > 25)$$
$$IC_2 : \forall ID, A, M \neg(Client(ID, A, M), A < 18, M > 50)$$

Why “local” and why interesting?

- Basically, inconsistencies can be fixed tuple by tuple, without introducing new violations
- LS-fixes always exist
- Local LDCs are the most common in census-like applications (Franconi,Laureti Palma, Leone, Perri, Scarcello; LPAR 01)
- DFP still NP-complete for local LDCs
- $DFOP$ still $MAXSNP$-hard for local LDCs

  (the non-local LDCs can be eliminated from the proof of the general case)

We will reduce $DFOP$ to the “Weighted Set Cover Problem”

A Weighted Set Cover Problem:

The WSCP is defined as follows

Instance: $(U, \mathcal{S}, w)$

- $\mathcal{S}$ is a collection of subsets of set U
- $\bigcup \mathcal{S} = U$ (a cover)
- w assigns numerical weights to elements of $\mathcal{S}$

Question: Find a sub-collection of $\mathcal{S}$ with minimum weight that covers U

$WSCP$ is $MAXSNP$-hard

We create a “good” instance of $WSCP$ ...

1. A set I of database atoms (tuples) from D is a violation set for $ic \in IC$ if $I \not\models ic$, and for every $I' \subsetneqq I$, $I' \models ic$, i.e. a minimal set of tuples that participate in the violation of an IC

   $$U := \text{set of violation sets for } D, IC$$

2. $\mathcal{I}(D, ic, t)$ denotes the set of violation sets for ic in D that contain tuple t

   $$\begin{aligned} S(t,t') := \{I \mid\ & \text{exists } ic \in IC,\ I \in \mathcal{I}(D, ic, t) \text{ and} \\ & ((I \smallsetminus \{t\}) \cup \{t'\}) \models ic\}, \end{aligned}$$

   i.e. the violations sets containing tuple t that are solved by changing t to t'

$\mathcal{S}$:= collection of $S(t,t')$'s such that t' is a *local fix* of t

I.e.

(a) t, t' share same values in non-fixable attributes

(b) $S(t,t') \neq \emptyset$ (some inconsistency is solved)

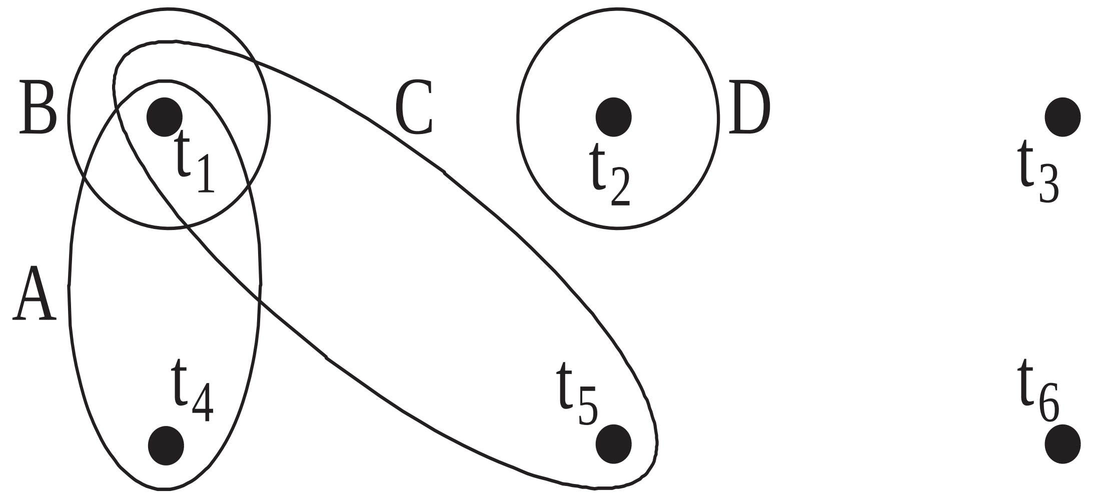
(c) $\Delta(\{t\},\{t'\})$ is minimum (relative to (a), (b))

3. Finally, $w(S(t,t')) := \Delta(\{t\},\{t'\})$

Properties of the Reduction:

- Local fixes can be obtained in polynomial time
- One-to-one correspondence between solutions to $DFOP$ and solutions to $WSCP$ that keeps the optimum values
- Each set cover corresponds to a consistent instance
- Transformation is in polynomial time
- Correspondence may be lost if applied to non-local LDCs (LDCs may not be satisfied by resulting instances)

Example 2: (cont.)



Client	<u>ID</u>	A	M	
	1	15	52	t_1
	2	16	51	t_2
	3	60	900	t_3
Buy	<u>**ID**</u>	<u>**I**</u>	**P**	
	1	CD	27	t_4
	1	DVD	26	t_5
	3	DVD	40	t_6

Violation sets: $\{t_1, t_4\}$, $\{t_1, t_5\}$ for IC_1; $\{t_1\}$, $\{t_2\}$ for IC_2

Tuple t_1: One local fix wrt IC_1, one wrt IC_2

Tuple t_2: One local fix Tuples t_4, t_5: One local fix each

Conflict (Hyper)Graphs: (Arenas, Bertossi, Chomicki; ICDT 01), (Chomicki, Marcinkowski; Information and Computation 05)

Tuples, their local fixes, and weights ...

Set cover els.	$S(t_1, t_1')$	$S(t_1, t_1'')$	$S(t_2, t_2')$	$S(t_4, t_4')$	$S(t_5, t_5')$
Local Fix	t_1'	t_1''	t_2'	t_4'	t_5'
Weight	4	9	1	4	1
Violation set A	0	1	0	1	0
Violation set B	1	1	0	0	0
Violation set C	0	1	0	0	1
Violation set D	0	0	1	0	0

1 and 0 at the bottom indicate if the violation set belongs or not to $S(t, t')$

Approximating $DFOP$:

We approximate a solution to $DFOP$ by approximating $WSCP$

In general, $WSCP$ can be approximated within a factor $O(log(n))$ by greedy algorithms (Chvatal)

In our case, the frequency (number of elements of $\mathcal{S}$ covering an element of U) is bounded by the maximum number of atoms in the ICs (small in general)

In this case $WSCP$ can be efficiently approximated within a constant factor given by the maximum frequency
(Hochbaum; 1997)

The approximation algorithm always returns a cover, from which we can compute an instance, that turns out to satisfy the LDCs, but may not be optimal (in Example 2 we get D')

Consistent Query Answering:

We have several results on data complexity for CQA, a decision or an optimization problem depending on the semantics

Some involving 1-atom denial constraints (1AD), a common case in census-like applications (one DB atom plus built-ins)

Usual semantics for CQA:

- Certain: Answer true in every LS-fix (default semantics)
- Possible: True in some LS-fix
- Range: Shortest numerical interval where all answers from LS-fixes can be found (for numerical, mainly aggregate queries) (Arenas, Bertossi, Chomicki; ICDT 2001)

  Two optimization problems: find extremes of the interval

Some results for CQA:

- Non-aggregate queries, certain semantics:
  - 1ADs; atomic ground query (and a wide class of boolean conjunctive queries): $PTIME$
  - Arbitrary extended LDCs; atomic query: P^{NP}-hard and in Π_2^P
- Aggregate queries (with one of *sum, count distinct, average*) and possible semantics (boolean query true in some fix: comparison of the aggregation to a constant) or range semantics
  - 1ADs; acyclic conjunctive query: $coNP$-hard

E.g. 1AD: $\forall u, c_1, c_2 \neg(R(u, c_1, c_2) \wedge c_1 < 1 \wedge c_2 < 1)$
Query: $q(count(distinct\ z)) \leftarrow R(u, x, y), S(u, z, x)$

The Gist: COUNT DISTINCT Aggregation

CQA under range semantics for COUNT DISTINCT acyclic conjunctive queries and one 1AD is $coNP$-complete

By reduction from MAX-SAT with instance $P = \langle U, C, K\rangle$; U a set of variables, C collection of clauses over U, K a positive integer

- Introduce relation $Var(\underline{u}, C_1, C_2)$ with tuple $(u, 0, 0)$ for every variable in $u \in U$; C_1, C_2 fixable, taking values 0 or 1
- Introduce non-fixable relation $Clause(u, c, s)$, with tuple (u, c, s) for every occurrence of $u \in U$ in clause $c \in C$, s is assignment (0 or 1) for u satisfying clause c
- 1AD: $\forall u, c_1, c_2 \neg(Var(u, c_1, c_2) \wedge c_1 < 1 \wedge c_2 < 1)$

Query: $q(count(distinct\ c)) \leftarrow Var(u, c_1, c_2), Clause(u, c, s), c_1 = s$

asks for how many clauses are satisfied in a given fix

Max value in a fix is max number of clauses that can be satisfied for P

Approximating CQA for Aggregate Queries:

Even for simple constraints, and simple aggregate queries defined on top of also simple conjunctive queries, CQA becomes hard

Finding the minimum/maximum values for an aggregate query among the LS-fixes become optimization problems

They are $MAXSNP$-hard for *sum*

For 1ADs and conjunctive aggregate query with *sum*, the maximum value for CQA (i.e. range semantics) can be efficiently approximated within a constant factor

Open Research Issues:

- Implementation issues and experimentation
- Applications in census-like scenarios
- Cases of polynomial complexity for LDCs with more that one database atoms
- Applications to data integration with preferences
- Approximation algorithm for DFP in more general cases
- Approximation algorithms for CQA to aggregate queries
- Fixes that preserve statistical validity; distribution preserving DFP and CQA
- Other numerical domains? Real numbers?