Semantic Constraints for Data Quality Assessment and Cleaning

Leopoldo Bertossi
Carleton University
Ottawa, Canada
(Faculty Fellow IBM CAS)

NSERC
Business Intelligence Network

Characterizing Consistent Data wrt ICs

A database may not satisfy a given set of integrity constraints

What is the consistent data in an inconsistent database?

What are the consistent answers to a query posed to an inconsistent database?

A mathematically precise definition was needed

In (Arenas,Bertossi,Chomicki; PODS99) such a characterization was provided

Intuitively, the consistent data in an inconsistent database D is invariant under all minimal ways of restoring D's consistency

That is, consistent data persists across all the minimally repaired versions of the original instance: the repairs of D

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

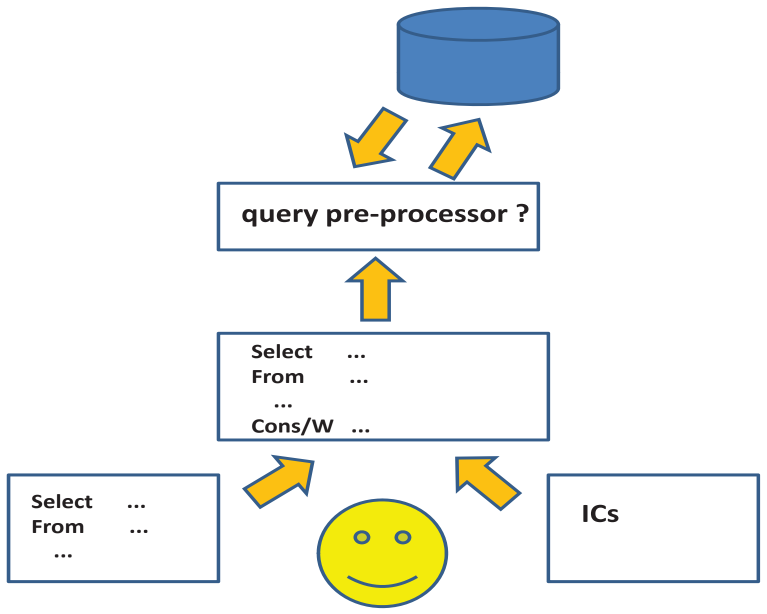
CQA may be different from classical data cleaning!

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

It also provides concepts and techniques for data cleaning

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

In particular, they should allow to be posed queries requesting for consistent data; and answer them



Why not an enhanced SQL?

SELECT	Name, Salary
FROM	Employee
CONS/W	FD: Name -> Salary;

(FD not maintained by the DBMS)

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

For some tractable cases, query rewriting algorithms have been developed

$$\mathcal{Q}(x,y):\ \ Employee(x,y) \quad \mapsto$$
$$\mathcal{Q}'(x,y):\ \ Employee(x,y) \land \neg\exists z(Employee(x,z) \land z \neq y)$$

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

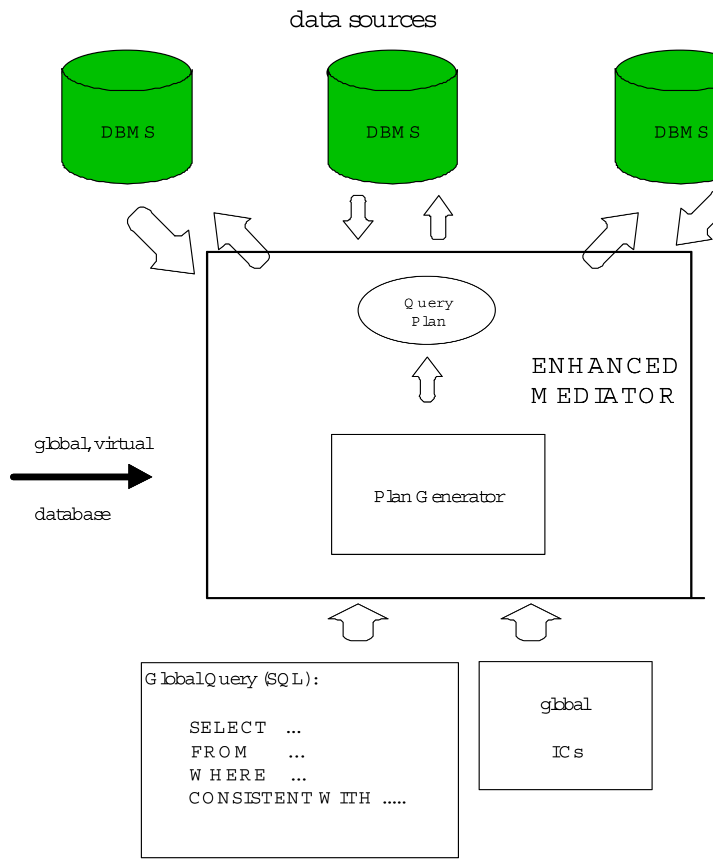
There are some implemented systems for CQA

- FO query rewriting (when possible)
- Graph-theoretic algorithmic methods

  Repairs can be implicitly represented as, e.g. maximal independent sets in a conflict graph or hypergraph
- Based on optimized (disjuntive) logic programs with stable model semantics (plus DLV)

More recently: Increasing interest in computing a single, "good" repair, or even an approximate repair

As a form of data cleaning wrt IC violation or semantic problems



Global Query (SQL):

```
SELECT    ...
FROM      ...
WHERE     ...
CONSISTENT WITH .....
```

A natural application: Virtual data integration

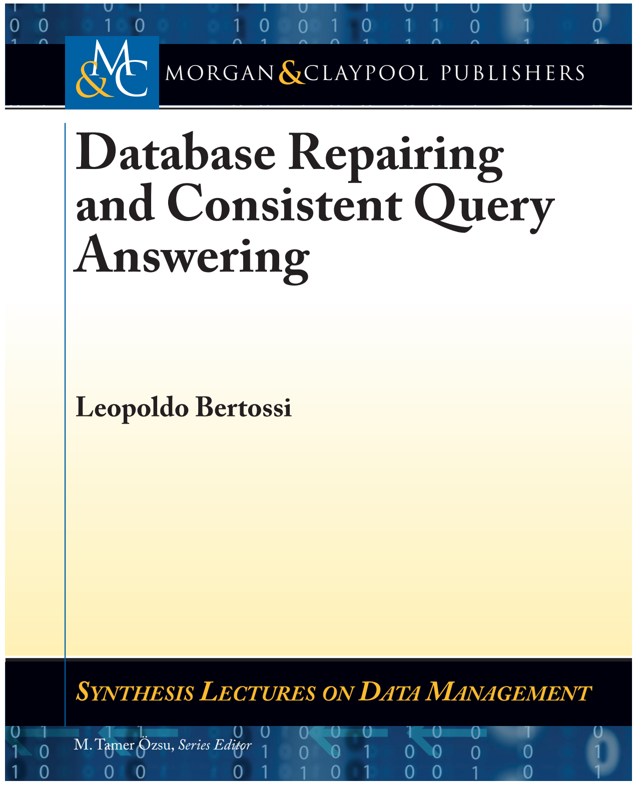
No way to enforce consistency on the sources

Inconsistencies have to be solved on-the-fly, at query time

Many problems in CQA addressed in the last few years

- Query rewriting mechanisms
- Compact representations of all DB repairs: Graph-theoretic, logic programs with stable model semantics, disjunctive databases, models of theories in non-classical logics, etc.
- Identification of tractable vs. non-tractable cases
- Applications in virtual data integration, PDMS, etc.
- Implementations

Published in 2011:



MORGAN & CLAYPOOL PUBLISHERS

Database Repairing and Consistent Query Answering

Leopoldo Bertossi

Synthesis Lectures on Data Management

M. Tamer Özsu, *Series Editor*

New Kinds of Constraints: Data Quality

Integrity constraints (ICs) have been around for a long time

They are used to capture the application semantics in the data model and database

They have been studied in general and have wide application in data management

A large body of research has been developed, in particular fundamental research

Methodologies for dealing with ICs are quite general and have broad applicability

Database repairing and CQA are newer contributions in this direction

On the other side:

Data quality assessment (DQ) and data cleaning (DC) have been mostly: Ad-hoc, rigid, vertical, and application-dependent activities

There is a lack of fundamental research in data quality assessment and data cleaning

Things are starting to change ...

Recently, DQ constraints have been proposed and investigated

They provide generic languages for expressing quality concerns

Suitable for specifying adaptive and generic DQ/C mechanisms

Proposed and studied by the Edinburgh DB group around Wenfei Fan

Conditional Dependencies (CDs)

Example: Database relation with FDs:

$FD_1: \quad [CC, AC, Phone] \rightarrow [Street, City, Zip]$

$FD_2: \quad [CC, AC] \rightarrow [City]$

CC	AC	Phone	Name	Street	City	Zip
44	131	1234567	mike	mayfield	NYC	EH4 8 LE
44	131	3456789	rick	crichton	NYC	EH4 8LE
01	908	3456789	joe	mtn ave	NYC	07974

FDs are satisfied, but they are "global" ICs

They may not capture natural data quality requirements, as related to specific data values (important in data quality assessment and data cleaning)

What about a *conditional functional dependency* (CFD)?

$$CFD_1\!: \quad [CC = 44, \mathit{Zip}] \rightarrow [\mathit{Street}]$$

Conditional in that the FD of Street upon Zip applies when the country code is 44

Not satisfied anymore, and data cleaning may be necessary ...

More generally, CDs are like classical ICs with a *tableau* for forced data value associations

$CFD_2\!:$

$$[CC = 44, AC = 131, \mathit{Phone}] \rightarrow [\mathit{Street}, \mathit{City}{=}\,\text{`}\mathit{EDI}', \mathit{Zip}]$$

When $CC = 44, AC = 131$ hold, the FD of Street and Zip upon Phone applies, and the city is 'EDI'

Not satisfied either ...

CQA and database repairs have been investigated for CFDs

[Kolahi, Lakshmanan], [Beskales, Ilyas, Golab], ...

Conditional Inclusion Dependencies:

$$Order(Title, Price, Type = \text{`}book') \subseteq Book(Title, Price)$$

It can be expressed in classical FO predicate logic:

$$\forall x \forall y \forall z (Order(x, y, z) \wedge z = \text{`}book' \ \rightarrow \ Book(x, y))$$

Still a classic flavor ...

And semantics ...

Matching Dependencies (MDs)

MDs are related to Entity Resolution (ER)

ER is a classical, common and difficult problem in data cleaning

It is about discovering and matching records that represent the same entity in the application domain

Again, several ad hoc mechanisms have been proposed

ER, and DC in general, are fundamental for data analysis and decision making in BI

Particularly crucial in data integration, and even more in virtual data integration (VDI)

In VDI, DC and ER have to be made on-the-fly, at query time

MDs express and generalize ER concerns

They specify attribute values that have to be made equal under certain conditions of similarity for other attribute values

Example: Schema $R_1(X, Y), R_2(X, Y)$

$$\forall X_1 X_2 Y_1 Y_2 (R_1[X_1] \approx R_2[X_2] \quad \longrightarrow \quad R_1[Y_1] \doteq R_2[Y_2])$$

When the values for attributes X_1 in R_1 and X_2 in R_2 in two tuples are similar, then the values in those two tuples for attribute Y_1 in R_1 and Y_2 in R_2 must be made equal (matched)

(R_1 and R_2 can be same predicate)

$\approx$: Domain-dependent similarity relation

Introduced by W. Fan et al. (PODS 2008, VLDB 2009)

Although declarative, MDs have a procedural feel and a dynamic semantics

An MD is satisfied by a pair of databases (D, D'):

D satisfies the antecedent, and D', the consequent, where the matching is realized

But this is local, one-step satisfaction ...

Our research: [ICDT'11, KR'12, ..., LID'11, SUM'12, DATALOG 2.0'12]

- Alternative, refined semantics for MDs
- Investigation of the dynamic semantics
- Definition and computation of clean instances (there may be several of them)
- Definition of "clean query answering", and computational methods to obtain them
- Comparisons between clean instances wrt MDs and database repairs wrt FDs
- Query rewriting methodologies for clean query answering (in Datalog plus aggregation)
- ASP-based specification of clean instances

MDs as originally introduced do not say how to identify values

$$\forall X_1 X_2 Y_1 Y_2 (R_1[X_1] \approx R_2[X_2] \quad \longrightarrow \quad R_1[Y_1] \doteq R_2[Y_2])$$

We have considered the two directions:

- With matching functions (MFs) (ICDT 2011, etc.), and
- Without MFs (LID 2011, etc.)

Matching Dependencies with MFs

"similar name and phone number $\Rightarrow$ identical address"

D_0	*name*	*phone*	*address*
	John Doe	(613)123 4567	Main St., Ottawa
	J. Doe	123 4567	25 Main St.

$$\Downarrow$$

D_1	*name*	*phone*	*address*
	John Doe	(613)123 4567	25 Main St., Ottawa
	J. Doe	123 4567	25 Main St., Ottawa

A dynamic semantics!

$$m_{address}(\underline{MainSt., Ottawa}\ ,\ \underline{25MainSt.}) := \underline{25MainSt., Ottawa}$$

Addresses treated as strings or objects, i.e. sets of pairs attribute/value

(Join work with Solmaz Kolahi and Laks Lakshmanan)

Semantics of MDs: [W. Fan et al., VLDB'09]

$$\varphi: \quad R_1[X_1] \approx R_2[X_2] \;\rightarrow\; R_1[Y_1] \doteq R_2[Y_2]$$

$(D, D') \models \varphi$ if for every R_1-tuple t_1 and R_2-tuple t_2:

$$t_1[X_1] \approx t_2[X_2] \text{ in } D \quad \Longrightarrow \quad \begin{array}{l} t_1[Y_1] = t_2[Y_2] \text{ in } D' \\ t_1[X_1] \approx t_2[X_2] \text{ in } D' \end{array}$$

D' is stable if $(D', D') \models \Sigma$ (a set of MDs)

Dirty instance $D \Rightarrow D_1 \Rightarrow D_2 \Rightarrow \ldots\ldots \Rightarrow D'$

$\uparrow$

stable, clean instance!

- How are the MDs enforced?
- Can we expect that $(D, D') \models \Sigma$? (too strong)

Matching Functions: Some ingredients

- Set of MDs Σ
- For every attribute A with Dom_A
  - A similarity relation $\approx_A \subseteq Dom_A \times Dom_A$ reflexive and symmetric
  - A matching function
    $$\mathtt{m}_A : Dom_A \times Dom_A \to Dom_A$$
    idempotent, commutative, and associative

Induces a semilattice with partial order defined as

$$a \preceq_A a' \iff \mathtt{m}_A(a, a') = a'$$

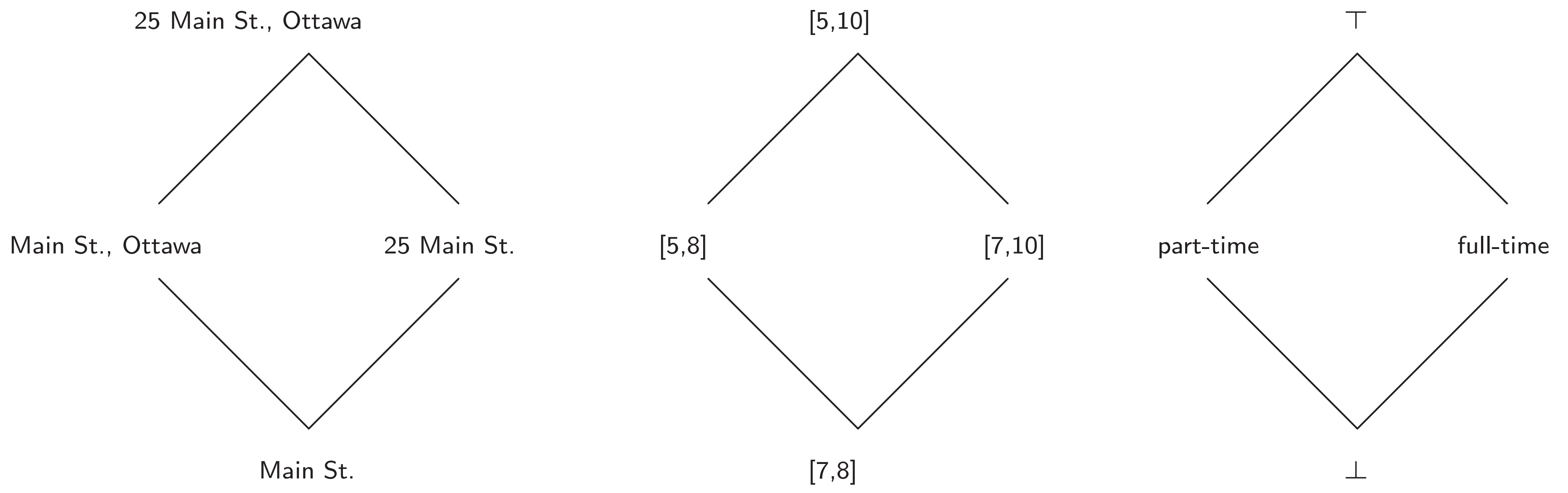
Least upper bound operator coincides with matching function

$$lub\{a, a'\} = \mathtt{m}_A(a, a')$$

$a \preceq_A a'$ can be thought of in terms of information contents

A semantic-domination lattice is created (... "domain theory")

- Domain-level lattice



- Tuple-level partial order:

$$t_1 \preceq t_2 \Longleftrightarrow t_1[A] \preceq_A t_2[A] \quad \text{(f.a. } A\text{)}$$

- Relation-level partial order

$$D_1 \sqsubseteq D_2 \Longleftrightarrow \forall t_1 \in D_1 \; \exists t_2 \in D_2 \; t_1 \preceq t_2$$

Instances can be “reduced” by eliminating tuples that are dominated by others

Theorem: The set of reduced instances with $\sqsubseteq$ forms a lattice

Relevant for comparison of sets of query answers seen as instances ...

Clean Instances:

$$\varphi : R_1[X_1] \approx R_2[X_2] \;\rightarrow\; R_1[A_1] \doteq R_2[A_2]$$

One step of chase: Enforcing φ on $D \Rightarrow D'$

- In D, $t_1[X_1] \approx t_2[X_2]$, but $t_1[A_1] = a_1 \neq t_2[A_2] = a_2$
- In D', replace them with $\mathtt{m}_A(a_1, a_2)$

Clean instance: Stable instance resulting from chase

$$D_0 \Rightarrow D_1 \Rightarrow \ldots \Rightarrow D_{clean}$$

Theorem: Matching functions idem, comm, assoc give us:

(a) Chase termination after polynomial number of steps

(b) $D_0 \sqsubseteq D_1 \sqsubseteq \ldots \sqsubseteq D_{clean}$

In general:

- There could be multiple clean instances
- It may not hold $(D_0, D_{clean}) \models \Sigma$

For two special cases:

- Similarity-preserving matching functions

$$a \approx a' \Longrightarrow a \approx \mathtt{m}_A(a', a'')$$

- Interaction-free MDs

- There is a unique clean instance D_{clean}, and

- $(D_0, D_{clean}) \models \Sigma$

Clean answers to a query $\mathcal{Q}$: (two bounds)

- Certain answers: $glb_{\sqsubseteq}\{\mathcal{Q}(D) \mid D$ clean instance$\}$
- Possible answers: $lub_{\sqsubseteq}\{\mathcal{Q}(D) \mid D$ clean instance$\}$

Assume only these two clean instances:

D'	$name$	$address$
	John Doe	25 Main St., Ottawa
	J. Doe	25 Main St., Ottawa
	Jane Doe	25 Main St., Vancouver

D''	$name$	$address$
	John Doe	Main St., Ottawa
	J. Doe	25 Main St., Vancouver
	Jane Doe	25 Main St., Vancouver

Query $\mathcal{Q}$: $\pi_{address}(\sigma_{name=\text{“J. Doe”}}(R))$

$Certain = \{\underline{\text{25 Main St.}}\}$

$Possible = \{\underline{\text{25 Main St., Ottawa}}\ ,\ \underline{\text{25 Main St., Vancouver}}\}$

Theorem: Computing certain clean answers is coNP-complete

Monotonicity?

$D \sqsubseteq D'$ is not set-inclusion

A query $\mathcal{Q}$ is monotone if: $D \sqsubseteq D' \implies \mathcal{Q}(D) \sqsubseteq \mathcal{Q}(D')$

Why not taking advantage of lattice-theoretic domain structure when posing queries?

Proposition: A positive relational algebra query composed of $\pi, \times, \cup$, $\sigma_{a \preceq A}$, $\sigma_{A_1 \bowtie_{\preceq} A_2}$ is monotone, where

$$\begin{aligned} t \in \sigma_{a \preceq A}(D) \quad &:\Longleftrightarrow \quad a \preceq t[A] \\ t \in \sigma_{A_1 \bowtie_{\preceq} A_2}(D) \quad &:\Longleftrightarrow \quad glb\{t[A_1], t[A_2]\} \neq \bot \end{aligned}$$

We obtain monotone queries

Monotonicity and clean query answering?

The two clean instances:

D'	$name$	$address$
	John Doe	25 Main St., Ottawa
	J. Doe	25 Main St., Ottawa
	Jane Doe	25 Main St., Vancouver

D''	$name$	$address$
	John Doe	Main St., Ottawa
	J. Doe	25 Main St., Vancouver
	Jane Doe	25 Main St., Vancouver

Query $\mathcal{Q}: \pi_{name}(\sigma_{\text{“25 Main St.”} \preceq address}(R))$ (is monotone)

$\mathcal{Q}(D') = \{$John Doe, J. Doe, Jane Doe$\}$

$\mathcal{Q}(D'') = \{$J. Doe, Jane Doe$\}$

$Certain(\mathcal{Q}) = \{$Jane Doe$\}$

We have: $\mathcal{Q}(glb_{\sqsubseteq}\{D', D''\}) = Certain(\mathcal{Q})$

In general, for the class $\mathcal{D}$ of clean instances

Proposition: For a monotone query:

$$\mathcal{Q}(\overbrace{glb_{\sqsubseteq}\{D \mid D \in \mathcal{D}\}}^{D_{\downarrow}}) \quad \sqsubseteq \quad \overbrace{glb_{\sqsubseteq}\{\mathcal{Q}(D) \mid D \in \mathcal{D}\}}^{\text{certain}}$$

$$\underbrace{lub_{\sqsubseteq}\{\mathcal{Q}(D) \mid D \in \mathcal{D}\}}_{\text{possible}} \quad \sqsubseteq \quad \mathcal{Q}(\underbrace{lub_{\sqsubseteq}\{D \mid D \in \mathcal{D}\}}_{D_{\uparrow}})$$

- Under-approximate certain answers by $\mathcal{Q}(D_{\downarrow})$
- Over-approximate possible answers by $\mathcal{Q}(D_{\uparrow})$

Adding heuristics to chase to obtain approximations to $D_{\downarrow}, D_{\uparrow}$?

Ongoing Research

- Make query posing/answering sensitive to semantic-domination lattice
- Approximate query answering based on relaxation using semantic domination lattice
- Computing clean answers from data subject to MDs (without physically cleaning it)

Query rewriting, approximations, ...

- Logic programs (ASPs) for clean QA in presence of MDs

LPs specify clean instances

LP-based declarative formulations of known ER algorithms, e.g. Swoosh

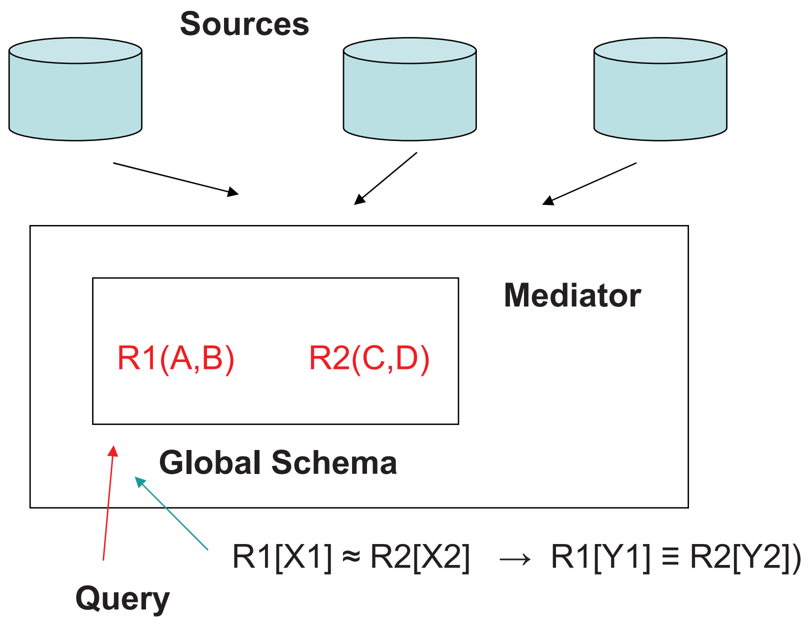
Look Ahead ...

Declarative specifications for ER could be compiled into query answering!

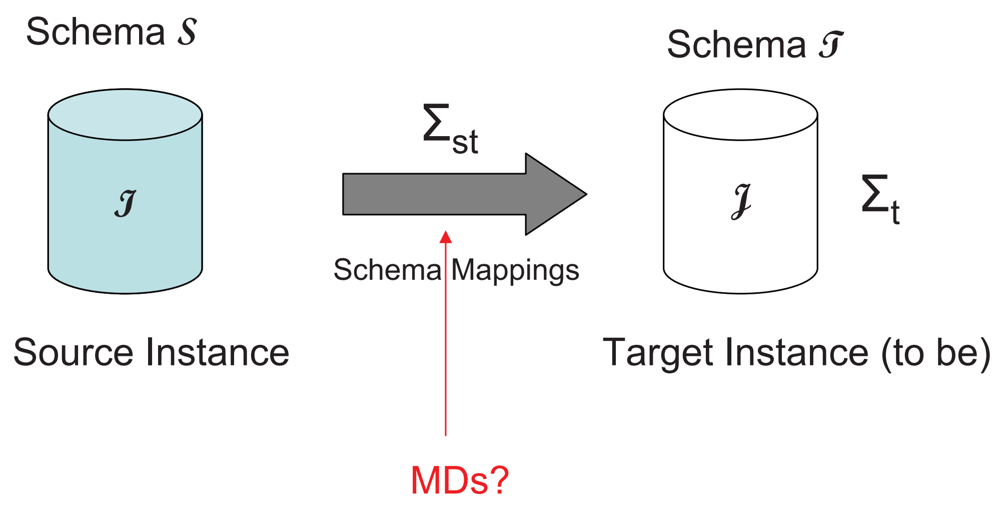
For different applications

Virtual data integration is a natural application scenario

On-the-fly ER!



Also data exchange under schema mappings:



Traditionally: Materialize a (good) target instance $\mathcal{J}$ with:

$$(\mathcal{I}, \mathcal{J}) \models \Sigma_{st} \quad \text{and} \quad \mathcal{J} \models \Sigma_t$$

Now: also apply MDs when shipping data from $\mathcal{I}$ to $\mathcal{J}$

ER at data exchange time ...

Extra Slides

On MDs:

- Under-approximate certain answers by $\mathcal{Q}(D_{\downarrow})$
- Over-approximate possible answers by $\mathcal{Q}(D_{\uparrow})$

Adding heuristics to chase to obtain $D_{\downarrow}, D_{\uparrow}$?

Under cleaning: not enforcing interacting MDs

Over cleaning: assuming matching functions are similarity preserving

Computing or approximating those two instances using D alone?

Associativity of MF is a natural assumption, not only because without it we can't have a lattice and termination of chase, etc., but also because it makes sense in any entity resolution process such as ours

That is, when during the process we identify three or more data values that are representing the same entity, the result of collapsing them into one value should not depend on the order in which we visit those values

If the aggregate function to be used is not associative, e.g. the average, we can always use union, and apply the aggregate function at the very end (average for instance)