Consistent Query Answering

Jan Chomicki
Dept. CSE
University at Buffalo
State University of New York
`http://www.cse.buffalo.edu/~chomicki`

Integrity constraints

Integrity constraints describe valid database instances. Examples:

- functional dependencies: *"every employee has a single salary."*
- denial constraints: *"no employee can make more than her manager."*
- referential integrity: *"managers have to be employees."*

The constraints are formulated in first-order logic:

$$\forall n, s, m, s', m'. \neg[Emp(n, s, m) \land Emp(m, s', m') \land s > s'].$$

An inconsistent database violates the constraints.

Traditional view

Integrity constraints are always enforced.

Emp

EmpName	*Address*	*Salary*
B. Gates	Redmond, WA	20M
B. Gates	Redmond, WA	30M
A. Grove	Santa Clara, CA	10M

Functional dependency:

$EmpName \rightarrow Address\ Salary$

This instance cannot arise but ... consider data integration.

Ignoring inconsistency

```
SELECT *
FROM Emp
WHERE Salary < 25M
```

⇒

B. Gates	Redmond, WA	20M
A. Grove	Santa Clara, CA	10M

The result is not fully reliable.

Quarantining inconsistency

The facts involved in an inconsistency are not used in the derivation of query answers [Bry, IICIS'97].

```
SELECT *
FROM Emp
WHERE Salary < 25M
```

$\Rightarrow$ $\boxed{\text{A. Grove} \quad \text{Santa Clara, CA} \quad \text{10M}}$

But what about

```
SELECT EmpName
FROM Emp
WHERE Salary > 1M
```

$\Rightarrow$ $\boxed{\text{A. Grove}}$

Partial information cannot be obtained.

A middle-ground solution

Consider all repairs: possible databases that result from fixing the original database.

Return all the answers that belong to the result of query evaluation in every repair (consistent answers).

```
SELECT *
FROM Emp
WHERE Salary < 25M
```

$\Rightarrow$

A. Grove	Santa Clara, CA	10M

But

```
SELECT EmpName
FROM Emp
WHERE Salary > 1M
```

$\Rightarrow$

B. Gates
A. Grove

Inconsistent databases

There are many situations when users want/need to live with inconsistent databases:

- integration of heterogeneous databases with overlapping information
- the consistency of the database will be restored by executing further transactions
- inconsistency wrt "soft" integrity constraints (those that we hope to see satisfied but do not/cannot check) process
- not enough information to resolve inconsistencies
- preservation of all data (even erroneous).

Research goals

Formal definition of reliable ("consistent") information in an inconsistent database.

Computational mechanisms for obtaining consistent information.

Computational complexity analysis:

- tractable vs. intractable classes of queries and integrity constraints
- trade-off: complexity vs. expressiveness.

Implementation:

- preferably using DBMS technology.

Plan of the talk

1. repairs and consistent query answers
2. computing consistent query answers to relational algebra/calculus queries
3. computational complexity
4. aggregation queries
5. alternative frameworks
6. related recent and current work
7. future directions.

Consistent query answers

Arenas, Bertossi, Chomicki [PODS'99].

Repair:

- a database that satisfies the integrity constraints
- difference from the given database is minimal (the set of inserted/deleted facts is minimal under set inclusion).

A tuple $(a_1, \ldots, a_n)$ is a consistent query answer to a query $Q(x_1, \ldots, x_n)$ in a database r if it is an element of the result of Q in every repair of r.

Emp

EmpName	*Address*	*Salary*
B. Gates	Redmond, WA	20M
B. Gates	Redmond, WA	30M
A. Grove	Santa Clara, CA	10M

Functional dependency:

$EmpName \rightarrow Address\ Salary$

Repairs:

B. Gates	Redmond, WA	30M
A. Grove	Santa Clara, CA	10M

B. Gates	Redmond, WA	20M
A. Grove	Santa Clara, CA	10M

A logical aside

Belief revision:

- semantically: repairing $\equiv$ revising the database with integrity constraints
- consistent query answers $\equiv$ counterfactual inference.

Logical inconsistency:

- inconsistent database: database facts together with integrity constraints form an inconsistent set of formulas
- trivialization of reasoning does not occur because constraints are not used in relational query evaluation.

Computational issues

There are too many repairs to evaluate the query in each of them.

A	B
a_1	b_1
a_1	b_1'
a_2	b_2
a_2	b_2'
$\ldots$	
a_n	b_n
a_n	b_n'

Under the functional dependency $A \rightarrow B$, this instance has 2^n repairs.

Computing consistent query answers

Query rewriting: given a query Q and a set of integrity constraints, construct a query Q' such that for every database instance r

> the set of answers to Q' in r = the set of consistent answers to Q in r.

Representing all repairs: given a set of integrity constraints and a database instance r:

1. construct a space-efficient representation of all repairs of r
2. use this representation to answer (many) queries.

Specifying repairs as logic programs.

Query rewriting

First-order queries transformed using semantic query optimization techniques: [Arenas, Bertossi, Chomicki, PODS'99].

Residues:

- associated with single literals $p(\bar{x})$ or $\neg p(\bar{x})$ (only one of each for every database relation p)
- for each literal $p(\bar{x})$ and each constraint containing $\neg p(\bar{x})$ in its clausal form, obtain a local residue by removing $\neg p(\bar{x})$ and the quantifiers for $\bar{x}$ from the constraint
- for each literal $\neg p(\bar{x})$ and each constraint containing $p(\bar{x})$ in its clausal form, obtain a local residue by removing $p(\bar{x})$ and the quantifiers for $\bar{x}$ from the constraint
- for each literal, global residue $=$ conjunction of local residues.

Functional dependencies:

$$(\forall x)(\forall y)(\forall z)(\forall y')(\forall z')(\neg Emp(x,y,z) \vee \neg Emp(x,y',z') \vee y = y')$$

$$(\forall x)(\forall y)(\forall z)(\forall y')(\forall z')(\neg Emp(x,y,z) \vee \neg Emp(x,y',z') \vee z = z')$$

Query:

$$Emp(x,y,z).$$

Local residues:

$$(\forall y')(\forall z')(\neg Emp(x,y',z') \vee y = y').$$

$$(\forall y')(\forall z')(\neg Emp(x,y',z') \vee z = z').$$

Constructing the transformed query

Given a first-order query Q.

Literal expansion: for every literal, construct an expanded version as the conjunction of this literal and its global residue.

Iteration: the expansion step is iterated by replacing the literals in the residue by their expanded versions, until no changes occur.

Query expansion: replace the literals in the query by their final expanded versions.

Functional dependencies:

$$(\forall x)(\forall y)(\forall z)(\forall y')(\forall z')(\neg Emp(x,y,z) \vee \neg Emp(x,y',z') \vee y = y')$$

$$(\forall x)(\forall y)(\forall z)(\forall y')(\forall z')(\neg Emp(x,y,z) \vee \neg Emp(x,y',z') \vee z = z')$$

Query:

$$Emp(x,y,z).$$

Transformed query:

$$\begin{aligned} Emp(x,y,z) \quad \wedge \quad & (\forall y')(\forall z')(\neg Emp(x,y',z') \vee y = y') \\ \wedge \quad & (\forall y')(\forall z')(\neg Emp(x,y',z') \vee z = z'). \end{aligned}$$

Integrity constraints:

$$(\forall x)(\neg p(x) \vee r(x))$$

$$\forall x)(\neg r(x) \vee s(x)))$$

Literal	Residue	First expansion	Second (final) expansion
$r(x)$	$s(x)$	$r(x) \wedge s(x)$	$r(x) \wedge s(x)$
$p(x)$	$r(x)$	$p(x) \wedge r(x)$	$p(x) \wedge r(x) \wedge s(x)$
$\neg r(x)$	$\neg p(x)$	$\neg r(x) \wedge \neg p(x)$	$\neg r(x) \wedge \neg p(x)$
$\neg s(x)$	$\neg r(x)$	$\neg s(x) \wedge \neg r(x)$	$\neg s(x) \wedge \neg r(x) \wedge \neg p(x)$

Scope of query rewriting

Query rewriting:

- queries involving conjunctions of literals (*relational algebra*: $\sigma, \bowtie, -$) and binary universal integrity constraints [Arenas, Bertossi, Chomicki, PODS'99].
- existentially-quantified conjunctions ($\pi, \sigma, \bowtie$) and single-key dependencies (under certain syntactic restrictions) [Fuxman, Miller, ICDT'05].

```
SELECT Name
FROM Emp
WHERE Salary > 1M
```

$\longmapsto$

```
SELECT Name
FROM Emp e1
WHERE Salary > 1M
  AND NOT EXISTS
      (SELECT *
       FROM EMPLOYEE e2
       WHERE e2.Name = e1.Name
         AND e2.Salary <= 1M)
```

Conflict hypergraph

Denial constraints only.

Vertices:

- facts in the original instance.

Edges:

- (minimal) sets of facts that violate some constraint.

Repair: a maximal independent set.

B. Gates Redmond, WA 20M

A. Grove Santa Clara, CA 10M

B. Gates Redmond, WA 30M

Ground queries

Observations:

- the query is in CNF $\Rightarrow$ each conjunct can be processed separately
- all repairs satisfy $\Phi \Leftrightarrow$ no repair satisfies $\neg\Phi$

Algorithm HProver:

1. $\neg\Phi = P_1(t_1) \wedge \cdots \wedge P_m(t_m) \wedge \neg P_{m+1}(t_{m+1}) \wedge \cdots \wedge \neg P_n(t_n)$
2. find a repair including $P_1(t_1), \ldots, P_m(t_m)$ and excluding $P_{m+1}(t_{m+1}), \ldots, P_n(t_n)$ by enumerating the appropriate edges.

Excluding a fact A:

- A is not in the original instance, or
- A belongs to an edge $\{A, B_1, \ldots, B_k\}$ in the conflict hypergraph and $B_1, \ldots, B_k$ belong to the repair.

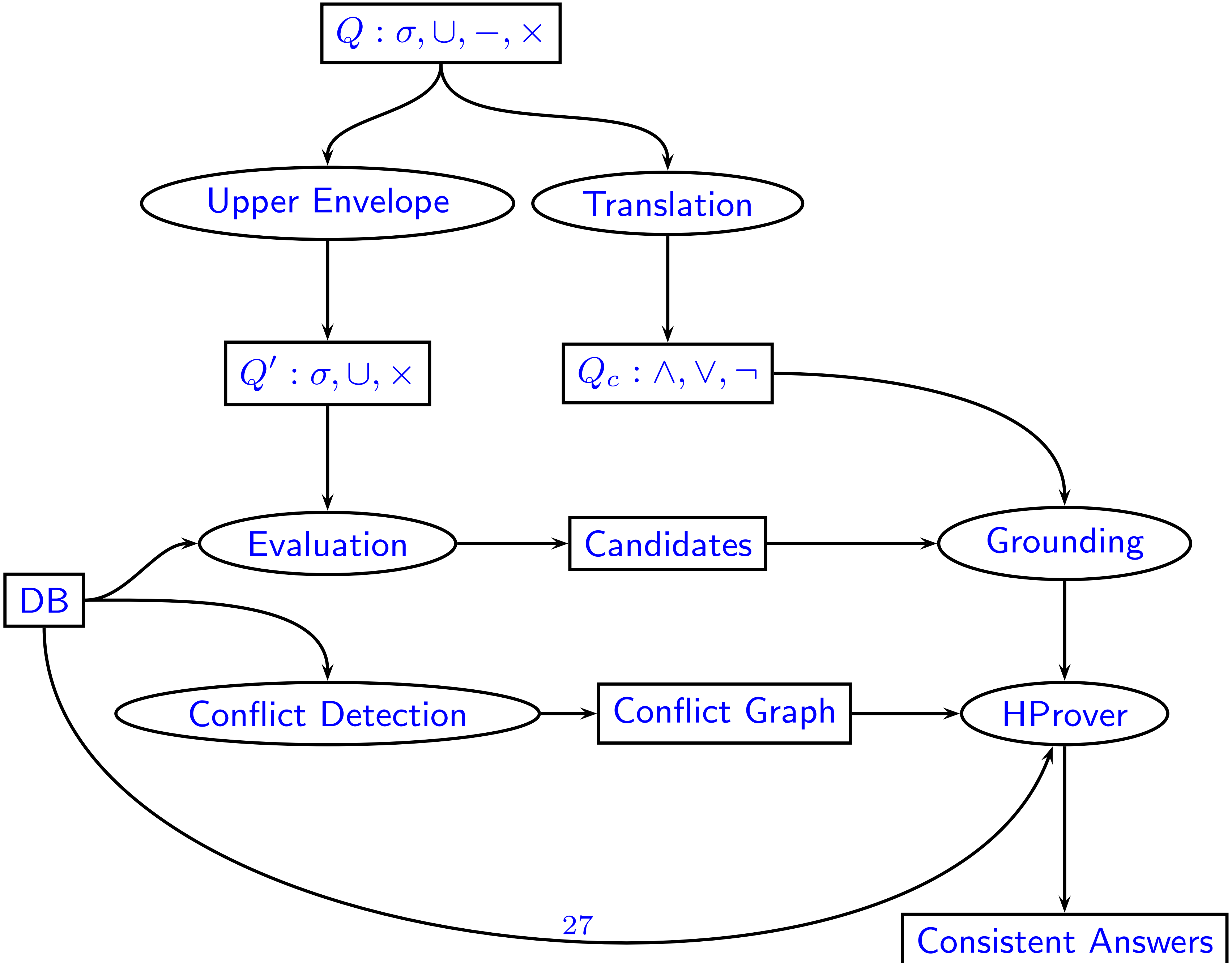
Properties of HProver

HProver works in PTIME (data complexity):

- $n - m$ choices from a set of polynomial size
- if all choices successful, a repair can be completed.

Generalizing to open, quantifier-free queries:

- possible bindings for free variables come from evaluating an upper envelope of the original query


$Q : \sigma, \cup, -, \times$
Upper Envelope
Translation
$Q' : \sigma, \cup, \times$
$Q_c : \wedge, \vee, \neg$
Evaluation
Candidates
Grounding
DB
Conflict Detection
Conflict Graph
HProver
Consistent Answers

Experimental results

[Chomicki, Marcinkowski, Staworko, CIKM'04].

The system Hippo:

- back-end: PostgreSQL
- conflict hypergraph (edges) in main memory
- optimization can eliminate many (sometimes all) database accesses in HProver
- tested for synthetic databases with up to 200K tuples, 2% conflicts
- computing consistent query answers using the conflict hypergraph faster than evaluating transformed queries
- relatively little overhead compared to evaluating the original query using the backend

Specifying repairs as logic programs

[Arenas, Bertossi, Chomicki, FQAS'00, TPLP'03], [Greco, Greco and Zumpano, ICLP'01, TKDE'03], [Barcelo, Bertossi, NMR'02, PADL'03]:

- using logic programs with negation and disjunction
- repairs $\equiv$ answer sets
- implemented using main-memory LP systems (`dlv`, `smodels`)
- Π_2^p-complete problems

Scope:

- arbitrary universal constraints, some inclusion dependencies
- arbitrary first-order queries
- queries can be "modalized" and nested

Facts:

$$Emp('\mathrm{B.Gates}', '\mathrm{Redmond\ WA}', 20\mathrm{K}).$$

$$Emp('\mathrm{B.Gates}', '\mathrm{Redmond\ WA}', 30\mathrm{K}).$$

$$Emp('\mathrm{A.Grove}', '\mathrm{Santa\ Clara\ CA}', 10\mathrm{K}).$$

Rules:

$$\neg Emp'(x, y, z) \vee \neg Emp'(x, y', z') \leftarrow Emp(x, y, z), Emp(x, y', z'), y \neq y'.$$

$$\neg Emp'(x, y, z) \vee \neg Emp'(x, y', z') \leftarrow Emp(x, y, z), Emp(x, y', z'), z \neq z'.$$

$$Emp'(x, y, z) \leftarrow Emp(x, y, z), not\ \neg Emp'(x, y, z).$$

$$\neg Emp'(x, y, z) \leftarrow not\ Emp(x, y, z), not\ Emp'(x, y, z).$$

Summary

	Query rewriting	Conflict hypergraph	Logic programs
Integrity constraints	Binary universal/single-key FDs	Denial	Universal+INDs
Queries	$\sigma, \times, -/\pi, \sigma, \times$	$\sigma, \times, -, \cup$	$\sigma, \pi, \times, -, \cup$
Data complexity	PTIME	PTIME	Π_2^p

Tractable/intractable queries

Tractable (PTIME):

- under any denial constraints:

```
SELECT * FROM P
UNION (SELECT * FROM Q
       EXCEPT SELECT * FROM R)
```

Schema:

```
CREATE TABLE P(A PRIMARY KEY, B);
CREATE TABLE Q(C PRIMARY KEY, D)
```

Tractable (PTIME):

```
SELECT Q.D
FROM  P, Q
WHERE P.B = Q.C
```

Intractable (co-NP-complete):

```
SELECT Q.D
FROM  P, Q
WHERE P.B = Q.D
```

Aggregation queries

```
SELECT SUM(Salary)
FROM Emp
```

$\Rightarrow$ [30,40]

A consistent answer to an aggregation query is no longer a single value:

- a set of values, or
- a range of values (polynomial size)

```
SELECT SUM(Salary)
FROM Emp
```

$\longmapsto$

```
SELECT SUM(P.MinS), SUM(P.MaxS)
FROM
  (SELECT MIN(Salary) AS MinS,
          MAX(Salary) AS MaxS
   FROM Emp
   GROUP BY Name) P
```

But that works only for a single functional dependency and some aggregation operators!

Consistent answers to aggregation queries

[Arenas, Bertossi, Chomicki, ICDT'01, TCS'03]:

	greatest lower bound		least upper bound	
	$\vert F\vert = 1$	$\vert F\vert \geq 2$	$\vert F\vert = 1$	$\vert F\vert \geq 2$
`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), AVG(A)`	PTIME	NP-complete	PTIME	NP-complete

Restriction to key dependencies improves tractability!

Alternative frameworks

Different assumptions about database completeness and correctness (in the presence of inclusion dependencies):

- possibly incorrect but complete: repairs by deletion only [Chomicki, Marcinkowski, I&C, 2005]
- possibly incorrect and incomplete: fix FDs by deletion, INDs by insertion [Cali, Lembo, Rosati, PODS'03].

Different notions of minimal repairs:

- minimal set of changes vs. minimal cardinality changes
- repairing attribute values [Wijsen, ICDT'03; Bohannon et al., SIGMOD'05].

Related work

Belief revision:

- revising database with integrity constraints
- revised theory changes with each database update
- emphasis on semantics (AGM postulates), not computation
- complexity results [Eiter, Gottlob, AI'92] do not quite transfer

Disjunctive information:

- repair $\equiv$ possible world (sometimes)
- using disjunctions to represent resolved conflicts
- query languages: representation-specific, relational algebra or calculus
- complexity results [Imielinski et al., JCSS'95] do not transfer

Current and future work

Systems:

- INFOMIX [Leone et al., SIGMOD 2005 demo]:
  - LP-based (`dlv`)
  - large databases (large number of repairs?)
- ConQuer [Fuxman, Fazli, Miller, SIGMOD'05]:
  - query rewriting
  - most TPC-H benchmark queries
  - large databases
- Hippo [Chomicki, Marcinkowski, Staworko, CIKM 2004]:
  - conflict hypergraph
  - no projection
  - large databases

Broadening scope:

- preferences and priorities [Staworko, Chomicki, Marcinkowski, IIDB 2006]:
  - source rankings, timestamps
  - probabilities [Andritsos, Fuxman, Miller, ICDE 2006]
- data integration and exchange
- data cleaning
- XML [Flesca et al, WISE 2005; Staworko, Chomicki, dataX 2006]
- spatial/spatiotemporal databases.

Selected papers:

1. M. Arenas, L. Bertossi, J. Chomicki, *"Consistent Query Answers in Inconsistent Databases,"* PODS'99.

2. M. Arenas, L. Bertossi, J. Chomicki, *"Specifying and Querying Database Repairs using Logic Programs with Exceptions,"* FQAS'00. Full version: *Theory and Practice of Logic Programming*, 2003.

3. M. Arenas, L. Bertossi, J. Chomicki, *"Scalar Aggregation in FD-Inconsistent Databases,"* ICDT'01. Full version: *Theoretical Computer Science*, 2003.

4. J. Chomicki, J. Marcinkowski, *"Minimal-Change Integrity Maintenance Using Tuple Deletions," Information and Computation*, 2005.

5. J. Chomicki, J. Marcinkowski, S. Staworko, *Computing Consistent Query Answers Using Conflict Hypergraphs,"* CIKM'04. Short version in IIWeb'04.

6. L. Bertossi, J. Chomicki, *"Query Answering in Inconsistent Databases,"* in *Logics for Emerging Applications of Databases,* J. Chomicki, R. van der Meyden, G. Saake [eds.], Springer-Verlag, 2003.