Consistent Query Answers in Inconsistent Databases

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

Joint work with Marcelo Arenas, Leo Bertossi, and Jerzy Marcinkowski, with contributions by Roger He, Vijay Raghavan and Jeremy Spinrad.

Integrity constraints

Integrity constraints describe **valid** database instances.

Examples:

- functional dependencies: *"every student has a single address."*
- denial constraints: *"no employee can make more than her manager."*
- referential integrity: *"students can enroll only in the offered courses."*
- spatial constraints: *"every ship has to be in a body of water."*

The constraints are formulated in **first-order logic**:

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

Inconsistent databases

There are situations when we want/need to live with **inconsistent** data in a database (data that **violates given integrity constraints**):

- the consistency of the database will be restored by executing further transactions
- integration of heterogeneous databases with duplicate information
- inconsistency wrt "soft" integrity constraints (those that we hope to see satisfied but do not/cannot check)
- denormalized relations in a data warehouse
- legacy data on which we want to impose semantic constraints
- it is impossible/undesirable to repair the database to restore consistency.

How to distinguish between **reliable** and **unreliable** information in an inconsistent database?

Plan of the talk

1. repairs and consistent query answers
2. first-order queries
3. aggregation queries
4. spatial constraints
5. computing consistent query answers
6. related and further work

Consistent query answers [PODS'99]

Repair:

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

Typically, more than one repair of a given inconsistent database.

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.

Functional dependency:

Name County $\rightarrow$ *Tally*

Name	*County*	*Date*	*Tally*
Brown	A	11/07	541
Brown	A	11/11	560
Brown	B	11/07	302
Green	A	11/07	653
Green	A	11/11	730
Green	B	11/07	101

Repairs:

Brown	A	11/07	541
Brown	B	11/07	302
Green	A	11/07	653
Green	B	11/07	101

Brown	A	11/11	560
Brown	B	11/07	302
Green	A	11/07	653
Green	B	11/07	101

Brown	A	11/07	541
Brown	B	11/07	302
Green	A	11/11	730
Green	B	11/07	101

Brown	A	11/11	560
Brown	B	11/07	302
Green	A	11/11	730
Green	B	11/07	101

Query languages

Ultimately: **SQL2**.

Now:

- **first-order** queries (equivalently: relational algebra)
- **scalar aggregation** queries.

The definition of consistent query answer may have to be generalized.

Consistent query answers

```
SELECT *
FROM Election
WHERE Name = 'Brown'
```

$\Rightarrow$

Brown	B	11/07	302

```
SELECT County
FROM Election
WHERE Name = 'Brown'
  AND Tally > 400
```

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

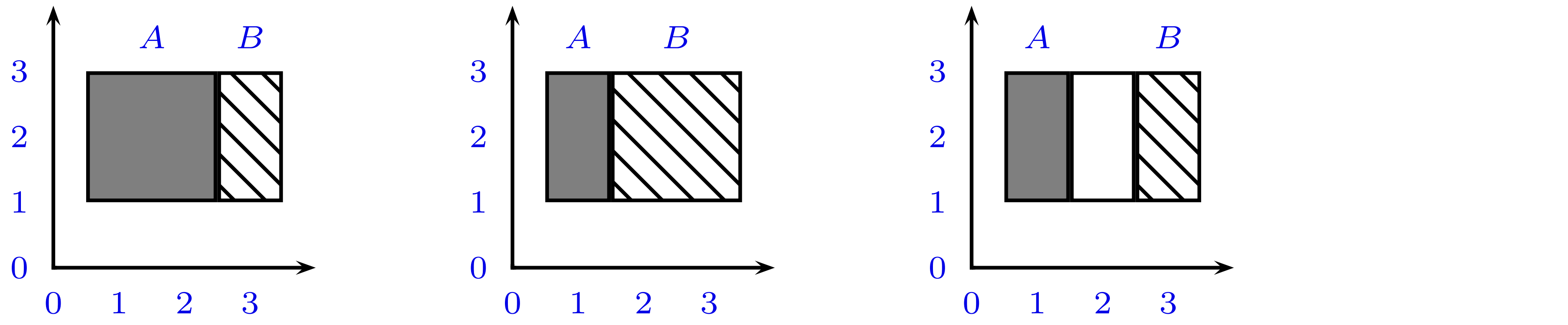
```
SELECT SUM(Tally)
FROM ELECTION
WHERE Name = 'Brown'
```

$\Rightarrow$ [843,862]

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

Spatial constraints

A and B are fighting a civil war in a country C and making conflicting claims about the territory occupied by each side.



Integrity constraint: *"the areas occupied by A and B form a disjoint partition of the territory of C"*.

Infinitely many repairs.

Consistent answer to the query about A's territory: set difference of A's and B's claims.

Computing consistent query answers

Query transformation: 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 queries.

Specifying repairs as logic programs.

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.

Query transformation [PODS'99]

First-order queries transformed using semantic query optimization techniques.

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 (possibly after variable renaming), obtain a local residue by removing $\neg p(\bar{x})$ and the quantifiers for $\bar{x}$ from the (renamed) constraint
- for each literal $\neg p(\bar{x})$ and each constraint containing $p(\bar{x})$ in its clausal form (possibly after variable renaming), obtain a local residue by removing $p(\bar{x})$ and the quantifiers for $\bar{x}$ from the (renamed) constraint
- for each literal, compute the global residue as the conjunction of local residues (possibly after normalizing variables)

The functional dependency

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

produces for $Student(x,y)$ the following local and global residue

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

The integrity constraints

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

produce the following global residues

Literal	Residue
$p(x)$	$r(x)$
$q(x)$	$r(x)$
$\neg r(X)$	$\neg p(x) \wedge \neg q(x)$

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.

The functional dependency

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

transforms the query $Student(x, y)$ into

$$Student(x,y) \wedge (\forall z)(\neg Student(x,z) \vee y = z)$$

For the 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
$r(x)$	$s(x)$	$r(x) \wedge s(x)$	$r(x) \wedge s(x)$
$p(x)$	$r(x)$	$p(x) \wedge r(x)$	$p(x) \wedge r(x) \wedge s(x)$
$\neg r(x)$	$\neg p(x)$	$\neg r(x) \wedge \neg p(x)$	$\neg r(x) \wedge \neg p(x)$
$\neg s(x)$	$\neg r(x)$	$\neg s(x) \wedge \neg r(x)$	$\neg s(x) \wedge \neg r(x) \wedge \neg p(x)$

```
SELECT *
FROM Election
WHERE Name = 'Brown'
```

$\Rightarrow$

```
SELECT *
FROM Election B1
WHERE B1.Name = 'Brown'
  AND NOT EXISTS
  SELECT *
  FROM Election B2
  WHERE B1.County = B2.County
    AND B1.Name = B2.Name
    AND B1.Tally <> B2.Tally.
```

Query transformation possible for queries involving conjunctions of literals (*relational algebra*: selection, join and difference) and binary integrity constraints.

Data complexity of consistent query answers [submitted]

Queries	Functional dependencies		Denial constraints
	$\|F\| = 1$	$\|F\| \geq 2$	
$\wedge, \vee, \neg$	PTIME	PTIME	PTIME
$\exists$	PTIME	co-NP-complete	co-NP-complete
$\exists, \wedge$	co-NP-complete	co-NP-complete	co-NP-complete
(2 literals)			

Aggregation queries [ICDT'01]

```
SELECT SUM(Tally)
FROM Election
WHERE Name = 'Brown'
```

$\Rightarrow$

```
WITH Partial(County,MinS,MaxS) AS
  (SELECT County,MIN(Tally),MAX(Tally)
   FROM Election
   WHERE Name = 'Brown'
   GROUP BY County)

SELECT SUM(MinS),SUM(MaxS)
FROM Partial;
```

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

Consistent answers to aggregation queries [ICDT'01]

	greatest lower bound		least upper bound	
	$\lvert F\rvert = 1$	$\lvert F\rvert \geq 2$	$\lvert F\rvert = 1$	$\lvert F\rvert \geq 2$
`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

How to reduce the computational cost?

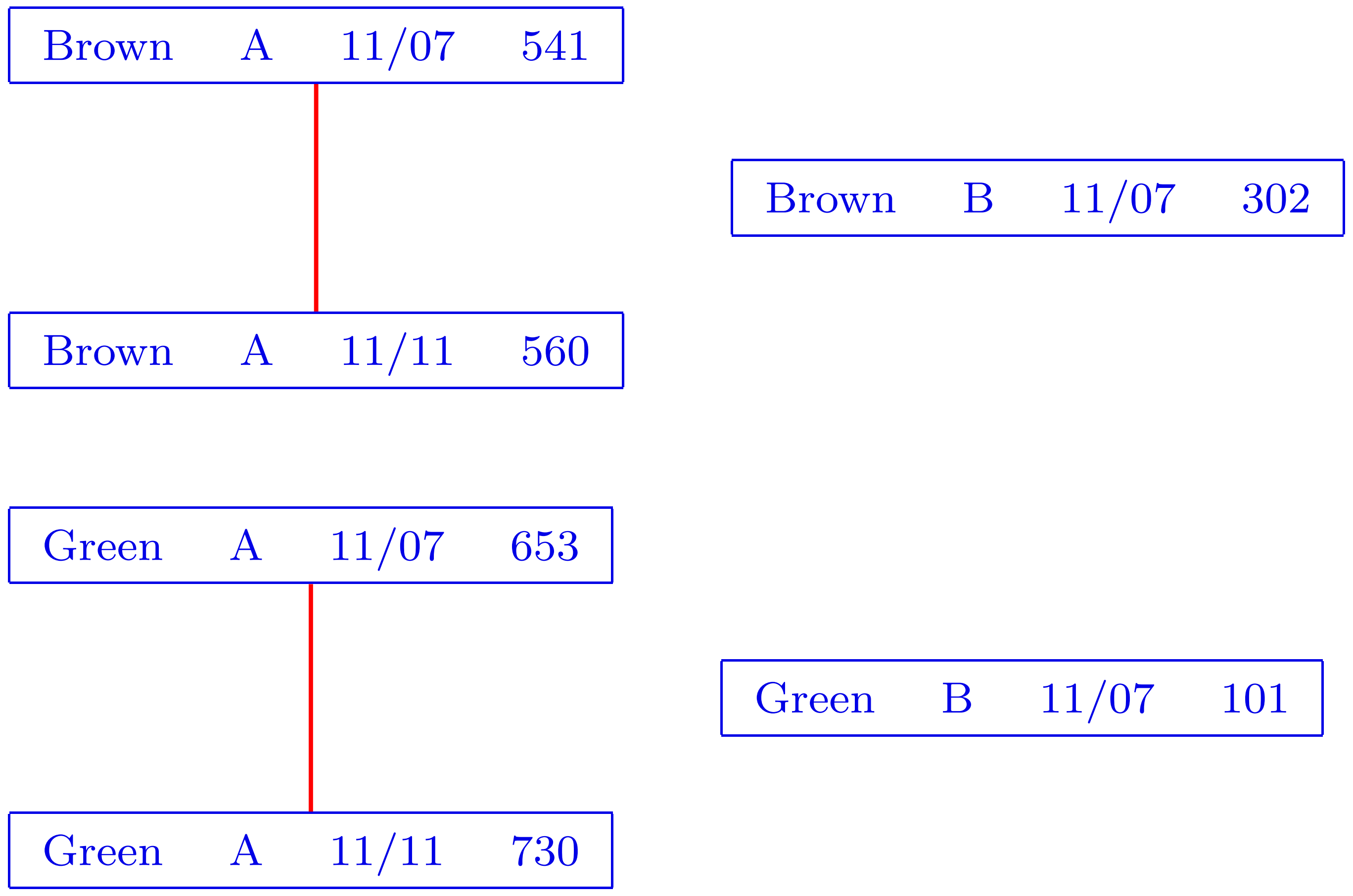
Representing all repairs

Not as a **formula** (as in belief revision) but as a **graph**.

A set of functional dependencies F, a database instance r.

Conflict graph:

- nodes: tuples in r
- edges: there is an edge (t_1, t_2) if there is a functional dependency $A \rightarrow B \in F$ such that $t_1[A] = t_2[A]$ and $t_1[B] \neq t_2[B]$.
- maximal independent sets: repairs


Brown A 11/07 541
Brown B 11/07 302
Brown A 11/11 560
Green A 11/07 653
Green B 11/07 101
Green A 11/11 730

Boyce-Codd Normal Form

Boyce-Codd Normal Form (BCNF):

Every functional dependency is a key dependency.

BCNF produces restrictions on the conflict graph that improve tractability.

Property: For one dependency in BCNF, the conflict graph is a union of disjoint cliques.

BCNF and `COUNT(*)` queries

Two functional dependencies.

Indirect approach:

- conflict graph is **claw-free** and **perfect**
- finding maximum independent sets in such graphs can be done in PTIME ($O(n^{5.5})$)

Direct approach:

- bipartite **clique graph**:
  - nodes: cliques in 1-dependency conflict graphs
  - edges: nonempty clique intersections
- finding a maximum matching in the clique graph
- overall complexity $O(n^{1.5})$.

Specifying repairs as logic programs [FQAS'00]

Logic programs with:

- negation in the body and the head
- disjunction
- exceptions (can be eliminated)

Scope:

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

A similar approach has been pursued by Greco and Zumpano [LPAR'00, CODAS'01].

Related work

Belief revision:

- revising database with integrity constraints
- revised theory changes with each database update
- emphasis on semantics (AGM postulates), not computation
- inference of ground literals using theorem proving techniques

Disjunctive information:

- repair $\equiv$ possible world
- using disjunctions to represent resolved conflicts
- constructing a single disjunctive instance
- query languages: representation-specific, relational algebra or calculus
- no tractable classes of aggregation queries

Future work

Broadening scope:

- SQL:
  - more general aggregation queries
  - nested subqueries
  - keys and foreign keys
- preferences:
  - source rankings
  - timestamps
- conflict resolution

New paradigms:

- query reformulation in information integration
- data cleaning
- XML

Algorithms and computational complexity:

- efficient algorithms for query processing in special cases
- lower bounds
- approximation

Selected papers:

1. M. Arenas, L. Bertossi, J. Chomicki, *"Consistent Query Answers in Inconsistent Databases,"* ACM Symposium on Principles of Database Systems (PODS), Philadelphia, May 1999.
2. M. Arenas, L. Bertossi, J. Chomicki, *"Specifying and Querying Database Repairs using Logic Programs with Exceptions,"* International Symposium on Flexible Query Answering Systems (FQAS), Warsaw, Poland, October 2000.
3. M. Arenas, L. Bertossi, J. Chomicki, *"Scalar Aggregation in FD-Inconsistent Databases,"* International Conference on Database Thory (ICDT), London, UK, January 2001. Full version accepted to a special issue of *Theoretical Computer Science.*
4. J. Chomicki, J. Marcinkowski, *On the Computational Complexity of Consistent Query Answers,"* submitted.