Consistent Query Answering via SAT Solving

Akhil A. Dixit   Phokion G. Kolaitis

University of California, Santa Cruz
Computer Science and Engineering Department

Appeared in the Proceedings of the SAT 2019, Lisbon, Portugal, July 2019

The Relational Data Model

- Relational Database: A collection $(R_1, \cdots, R_m)$ of finite relations (tables)
- Relational Query Languages
  - Relational Calculus: (Safe) First-Order Logic
  - SQL: The standard commercial database query language based on relational calculus

Conjunctive Queries

A Conjunctive Query (CQ) is a query specified by a first-order formula of the form

$$\exists y_1, \cdots, \exists y_m\, \varphi(x_1, \cdots, x_n, y_1, \cdots, y_m)$$

where $\varphi(x_1, \cdots, x_n, y_1, \cdots, y_m)$ is a conjunction of atoms.

Example

Schema: $Enrolls(student, course)$, $Teaches(professor, course)$

Query in relational calculus:

$\textsc{Taught-By}(x_1, x_2) : \exists y \big(Enrolls(x_1, y) \wedge Teaches(x_2, y) \big)$

SQL expression for TAUGHT-BY:

SELECT *Enrolls.student*, *Teaches.professor*
FROM *Enrolls, Teaches*
WHERE *Enrolls.course = Teaches.course*

Integrity Constraints and Inconsistent Databases

Key Constraint $R : X \rightarrow Y$,
If two tuples in R agree on X, then they also agree on Y, where Y is the set of attributes of R that are not in X.

Inconsistent Database: A database $\mathcal{I}$ that violates Σ.

- Inconsistencies are unavoidable in the real world

Example

S : $Employees(\underline{EmpID}, Name, Role, Salary)$

$\Sigma : EmpID \rightarrow Name, Role, Salary$

#	<u>*EmpID*</u>	*Name*	*Role*	*Salary*
1	111	Bob Williams	Software Engineer	$90,000
2	112	John Smith	Software Engineer	$100,000
3	112	Alice Brown	Software Intern	$50,000

Coping with Inconsistent Databases

Two different approaches:

- Data Cleaning: Inconsistencies are removed by modifying (adding, deleting, updating) the tuples in the relations.
  - Main approach in the industry
  - More engineering than science

Coping with Inconsistent Databases

Two different approaches:

- Data Cleaning: Inconsistencies are removed by modifying (adding, deleting, updating) the tuples in the relations.
  - Main approach in the industry
  - More engineering than science
- Database Repairs: A principled framework for coping with inconsistent databases without “cleaning” dirty data.

Database Repairs and Consistent Answers

Definition (Arenas, Bertossi, Chomicki – 1999)

Σ a set of integrity constraints and $\mathcal{I}$ an inconsistent database.
Database instance $\mathcal{J}$ is a subset-repair of $\mathcal{I}$ w.r.t. Σ if

- $\mathcal{J} \subset \mathcal{I}$
- $\mathcal{J} \models \Sigma$ (i.e., $\mathcal{J}$ is consistent)
- there is no $\mathcal{J}'$ such that $\mathcal{J}' \models \Sigma$ and $\mathcal{J} \subset \mathcal{J}' \subset \mathcal{I}$.

Definition (Arenas, Bertossi, Chomicki - 1999)

Σ a set of integrity constraints, q a query, and $\mathcal{I}$ an inconsistent database. The consistent answers to q on $\mathcal{I}$ w.r.t. Σ is the set

$$\textsc{Cons}(q, \mathcal{I}, \Sigma) = \bigcap \ \{q(\mathcal{J}) : \mathcal{J} \text{ is a repair of } \mathcal{I} \text{ w.r.t. } \Sigma\}.$$

Database Repairs and Consistent Answers

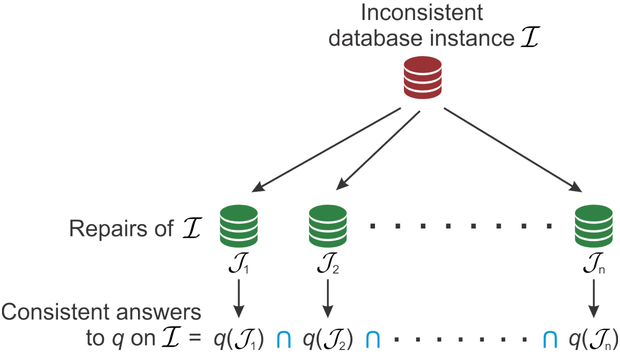


Figure: $\textsc{Cons}(q, \mathcal{I}, \Sigma)$

Consistent Query Answering (CQA)

- Consistent Query Answering (CQA): The problem of computing $\textsc{Cons}(q, \mathcal{I}, \Sigma)$
- CQA is often studied for Conjunctive Queries
- CQA for conjunctive queries can be intractable

Example

$\mathbf{S} : R(\underline{A}, B)$ $\qquad \Sigma : A \rightarrow B$

$\mathcal{I} =$

#	$\underline{A}$	B
1	a_1	b_1
2	a_1	b_2
3	a_2	b_1
4	a_2	b_2

- Four (subset) repairs of $\mathcal{I}$ are $\{1, 3\}, \{1, 4\}, \{2, 3\}$, and $\{2, 4\}$
- Exponentially many repairs, in general

More on the Complexity of CQA

- Boolean CQs: Conjunctive queries with no free variables.
- The decision problem $\textsc{Certainty}(q, \Sigma)$ asks:
  Given a database $\mathcal{I}$, is $\textsc{Cons}(q, \mathcal{I}, \Sigma)$ true?

Example

S : $R(\underline{A}, B),\ S(\underline{C}, D)$ $\qquad$ $\Sigma : A \to B,\ \ C \to D$

- $\textsc{Path}()$: $\exists x, y, z(R(\underline{x}, y) \land S(\underline{y}, z))$
  $\textsc{Certainty}(\textsc{Path})$ is SQL-rewritable.
- $\textsc{Circle}()$: $\exists x, y(R(\underline{x}, y) \land S(\underline{y}, x))$
  $\textsc{Certainty}(\textsc{Circle})$ is in P, but not SQL-rewritable.
- $\textsc{Sink}()$: $\exists x, y, z(R(\underline{x}, z) \land S(\underline{y}, z))$
  $\textsc{Certainty}(\textsc{Sink})$ is in coNP-complete.

The Trichotomy for CERTAINTY($q, \mathcal{I}, \Sigma$)

Theorem (Koutris and Wijsen - 2015, 2017)

If Σ is a set of key constraints with one key per relation and q is a Boolean self-join-free conjunctive query, then one of the following holds.

- CERTAINTY(q, Σ) is SQL-rewritable.
- CERTAINTY(q, Σ) is in P, but not SQL-rewritable.
- CERTAINTY(q, Σ) is coNP-complete.

Moreover, this trichotomy is decidable in quadratic time.

More on the Koutris-Wijsen Trichotomy

- SQL-rewritable: For a query q, there is another query q' such that $q'(\mathcal{I}) = \textsc{Cons}(q, \mathcal{I}, \Sigma)$. For example,

  S: E (id, name, city)

  q = SELECT *E1.name* FROM *E* AS *E1* WHERE *E1.city* = *'London'*;

  q′ = SELECT *E1.name* FROM *E* AS *E1* WHERE *E1.city='London'*
  AND NOT EXISTS (SELECT * FROM *E* AS *E2* WHERE *E2.id* = *E1.id*
  AND *E2.city* != *'London'*);

More on the Koutris-Wijsen Trichotomy

- SQL-rewritable: For a query q, there is another query q' such that $q'(\mathcal{I}) = \textsc{Cons}(q, \mathcal{I}, \Sigma)$. For example,

  S: E (<u>id</u>, name, city)

  q = SELECT *E1.name* FROM *E* AS *E1* WHERE *E1.city* = *'London'*;

  q' = SELECT *E1.name* FROM *E* AS *E1* WHERE *E1.city*=*'London'*
  AND NOT EXISTS (SELECT * FROM *E* AS *E2* WHERE *E2.id* = *E1.id*
  AND *E2.city* != *'London'*);

- In P, but not SQL-rewritable: $\textsc{Cons}(q, \mathcal{I}, \Sigma)$ can be efficiently computed, but that algorithm cannot be expressed in SQL

More on the Koutris-Wijsen Trichotomy

- SQL-rewritable: For a query q, there is another query q' such that $q'(\mathcal{I}) = \textsc{Cons}(q, \mathcal{I}, \Sigma)$. For example,

  S: E (id, name, city)

  q = SELECT *E1.name* FROM *E* AS *E1* WHERE *E1.city* = *'London'*;

  q' = SELECT *E1.name* FROM *E* AS *E1* WHERE *E1.city='London'*
  AND NOT EXISTS (SELECT * FROM *E* AS *E2* WHERE *E2.id* = *E1.id*
  AND *E2.city* != *'London'*);

- In P, but not SQL-rewritable: $\textsc{Cons}(q, \mathcal{I}, \Sigma)$ can be efficiently computed, but that algorithm cannot be expressed in SQL

- CoNP-complete: No efficient algorithm can compute $\textsc{Cons}(q, \mathcal{I}, \Sigma)$, unless P = NP.

Satisfiability (SAT)

Definition

Boolean Formula is an expression built from variables, constants, and the Boolean operators conjunction ($\wedge$), disjunction ($\vee$), and negation ($\neg$).

Definition

Boolean SAT Problem: Given a Boolean formula, is it satisfiable?

Example

$\phi = (a \vee b \vee c) \wedge (\neg d) \wedge (b \vee \neg c \vee d) \wedge (\neg a \vee c) \wedge (d \vee e \vee \neg f) \wedge (f)$
Satisfying assignment: $(a, b, c, d, e, f) = (0, 1, 1, 0, 1, 1)$

- Boolean SAT is the most fundamental and most extensively studied NP-complete problem.

Modern SAT Solvers

- SAT Revolution (Biere, Heule, van Maaren, Walsh - 2009):
  "From 100 variables and 200 clauses (in early 1990s) to 1,000,000+ variables and 5,000,000+ clauses in 20 years"
- Advanced algorithms like conflict-driven clause learning and look-ahead techniques, pre-processing and in-processing optimizations
- Widely used as general purpose problem-solving tools
- Mostly open-source, e.g., Lingeling, Glucose, MaxHS

Motivation...

...To build a CQA System

- Research on CQA has not penetrated the industry yet
- Industry relies on ad-hoc data cleaning techniques
- Partly because no comprehensive CQA system exists

...To use SAT Solvers

- Current logic-based approaches to CQA have limitations in terms of the size of the database
- Recent advancements in SAT solving technology
- Natural reductions from the complement of CQA to variants of SAT for broad classes of queries and integrity constraints

Applications of Repairs and CQA

Data Exchange (ten Cate, Halpert, Kolaitis - 2016)

- Conflicting information in the source instance is undesirable for query answering
- New frameworks based on the notion of exchange-repairs

Applications of Repairs and CQA

Data Exchange (ten Cate, Halpert, Kolaitis - 2016)

- Conflicting information in the source instance is undesirable for query answering
- New frameworks based on the notion of exchange-repairs

Description Logic (Bienvenu and Bourgaux - 2017)

- Various semantics to inconsistency-tolerant query answering in context to ontology-based data access
- Query answering systems such as CQAPri for querying inconsistent knowledge bases

Applications of Repairs and CQA

Data Exchange (ten Cate, Halpert, Kolaitis - 2016)

- Conflicting information in the source instance is undesirable for query answering
- New frameworks based on the notion of exchange-repairs

Description Logic (Bienvenu and Bourgaux - 2017)

- Various semantics to inconsistency-tolerant query answering in context to ontology-based data access
- Query answering systems such as CQAPri for querying inconsistent knowledge bases

Fairness in ML (Salimi, Rodriguez, Howe, Suciu - 2019)

- Underlying training data often reflects discrimination
- Notion of repairing the training data for fairness guarantees

Comprehensive CQA System using SAT Solvers

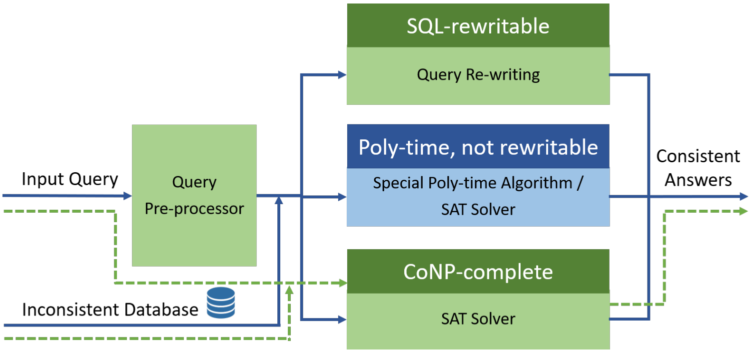


Figure: Envisioned architecture of a modular CQA system

Complement of CERTAINTY(q) to SAT

1. Let g_j be a set of tuples of I that share a key j. For all tuples $f_i \in g_j$, construct a clause u_j such that

$$u_j = \bigvee_{f_i \in g_j} x_{f_i}$$

2. For each minimal witness w_k to q on I, construct a clause v_k such that

$$v_k = \bigvee_{f_i \in w_k} \neg x_{f_i}$$

3. Construct a conjunctive Boolean formula

$$\phi = (u_1 \wedge u_2 \wedge ... \wedge u_m) \wedge (v_1 \wedge v_2 \wedge ... \wedge v_n)$$

Complement of CERTAINTY(q) to SAT

1. Let g_j be a set of tuples of I that share a key j. For all tuples $f_i \in g_j$, construct a clause u_j such that

$$u_j = \bigvee_{f_i \in g_j} x_{f_i}$$

2. For each minimal witness w_k to q on I, construct a clause v_k such that

$$v_k = \bigvee_{f_i \in w_k} \neg x_{f_i}$$

3. Construct a conjunctive Boolean formula

$$\phi = (u_1 \wedge u_2 \wedge ... \wedge u_m) \wedge (v_1 \wedge v_2 \wedge ... \wedge v_n)$$

Fact: ϕ is satisfiable if and only if q is false on some repair of I.

A Short Demo of CAvSAT

Beyond sjfBCQ and Primary Key Constraints

- Primary keys are an important but limited class of integrity constraints

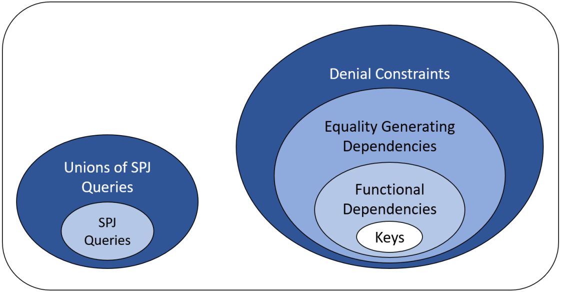


Figure: Classes of integrity constraints and database queries

- Preceding reduction extends naturally for UCQs over databases having arbitrary denial constraints

Experimental Setup

Phase I:

- Synthetically generated databases (1M tuples/relation)
- One key constraint per relation
- Varying inconsistency (5% to 15%)
- 21 conjunctive queries from the literature

Experimental Setup

Phase I:

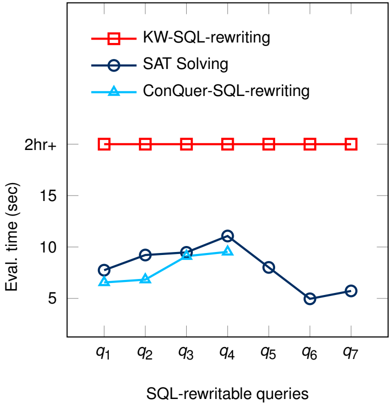
- Synthetically generated databases (1M tuples/relation)
- One key constraint per relation
- Varying inconsistency (5% to 15%)
- 21 conjunctive queries from the literature

Phase II:

- Real-world data about food and safety inspections of restaurants in Chicago and New York (up to 230K tuples/relation)
- Arbitrary key and functional dependency constraints
- Up to 25% inconsistency
- Five conjunctive queries, one union of conjunctive queries

Experimental Results (Phase I)

- KW-SQL-rewriting: The state-of-the-art generic algorithm for SQL-rewriting (Koutris, Wijsen – 2017)
- ConQuer-SQL-rewriting: A special algorithm suitable for a subclass C_{forest} of SQL-rewritable queries (Fuxman, Fazli, Miller – 2005)



On the Practicality of KW-SQL-rewriting

Schema

```
R1(A,B,C), R3(A,B), R4(A,B,C)
```

Query

```
SELECT R1.C FROM R1, R3, R4
WHERE R1.B=R3.A AND R3.B=R4.A
```

KW-SQL-Rewriting

```
SELECT DISTINCT free_R1.C FROM R1 AS free_R1
WHERE EXISTS(
    SELECT * FROM R1 AS s1
    WHERE NOT EXISTS(
        SELECT * FROM R1 AS r1
        WHERE NOT EXISTS(
            SELECT * FROM R3 AS s2
            WHERE NOT EXISTS(
                SELECT * FROM R3 AS r2
                WHERE NOT EXISTS(
                    SELECT * FROM R4 AS s3
                    WHERE NOT EXISTS(
                        SELECT * FROM R4 AS r3
                        WHERE ((( (s1.A=r1.A)
                        AND (s2.A=r2.A)) AND (s3.A=r3.A))
                        AND (((r2.B!=r3.A) OR (r1.B!=r2.A))
                        OR (r1.C!=free_R1.C)))
                    )
                )
            )
        )
    )
)
```

Experimental Results (Phase I)

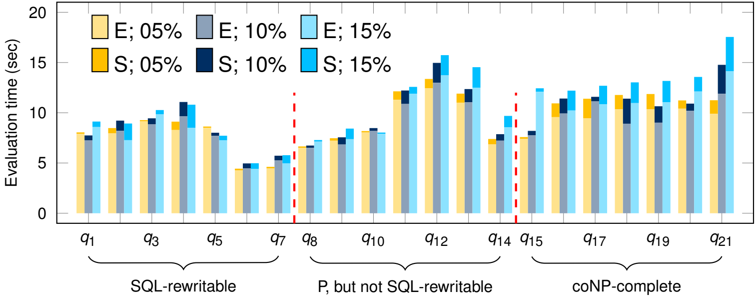


Figure: Time to compute consistent answers using MaxHS v3.0

Experimental Results (Phase II)

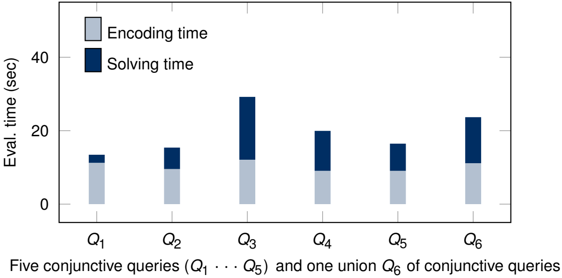


Figure: Time to compute consistent answers using MaxHS v3.0

Synopsis and Outlook

Summary

- The framework of repairs provides a scientific approach to cope with inconsistent databases
- While much progress has been made in theory, no comprehensive CQA system exists in practice
- Given the advancements in SAT solving, CAvSAT seems to be a promising approach

Future work

- Investigate the gap between theory and practice w.r.t. CQA
- Conjunctive queries with aggregate functions
- Performance comparison of CAvSAT with existing reduction-based CQA systems

Conjunctive Queries Used On Synthetic Data

FO-rewritable:

$q_1(z) : R_1(\underline{x}, y, z), R_2(\underline{y}, v, w)$

$q_2(z, w) : R_1(\underline{x}, y, z), R_2(\underline{y}, v, w)$

$q_3(z) : R_1(\underline{x}, y, z), R_3(\underline{y}, v), R_2(\underline{v}, u, d)$

$q_4(z, d) : R_1(\underline{x}, y, z), R_3(\underline{y}, v), R_2(\underline{v}, u, d)$

$q_5(z) : R_1(\underline{x}, y, z), R_4(\underline{y, v}, w)$

$q_6(z) : R_1(\underline{x}, y, z), R_2(\underline{x'}, y, w), R_5(\underline{x}, y, d)$

$q_7(z) : R_1(\underline{x}, y, z), R_2(\underline{y}, x, w), R_5(\underline{x}, y, d)$

In P, but not FO-rewritable:

$q_8(z, w) : R_1(\underline{x}, y, z), R_2(\underline{y}, x, w)$

$q_9(z) : R_1(\underline{x}, y, z), R_2(\underline{y}, x, w), R_4(\underline{y}, u, d)$

$q_{10}(z, w, d) : R_1(\underline{x}, y, z), R_2(\underline{y}, x, w), R_4(\underline{y}, u, d)$

$q_{11}(z) : R_1(\underline{x}, y, z), R_2(\underline{y}, x, w)$

$q_{12}(v, d) : R_3(\underline{x}, y), R_6(\underline{y}, z), R_1(\underline{z}, x, d), R_4(\underline{x, u}, v)$

$q_{13}(v) : R_3(\underline{x}, y), R_6(\underline{y}, z), R_7(\underline{z}, x), R_4(\underline{x, u}, v)$

$q_{14}(d) : R_3(\underline{x}, y), R_6(\underline{y}, z), R_1(\underline{z}, x, d), R_7(\underline{x}, u)$

CoNP-complete:

$q_{15}(z) : R_1(\underline{x}, y, z), R_2(\underline{x'}, y, w)$

$q_{16}(z, w) : R_1(\underline{x}, y, z), R_2(\underline{x'}, y, w)$

$q_{17}(z) : R_1(\underline{x}, y, z), R_2(\underline{x'}, y, w), R_4(\underline{y}, u, d)$

$q_{18}(z, w) : R_1(\underline{x}, y, z), R_2(\underline{x'}, y, w), R_4(\underline{y}, u, d)$

$q_{19}(z, w, d) : R_1(\underline{x}, y, z), R_2(\underline{x'}, y, w), R_4(\underline{y}, u, d)$

$q_{20}(z) : R_1(\underline{x}, y, z), R_2(\underline{x'}, y, w), R_4(\underline{y}, u, d), R_3(\underline{u}, v)$

$q_{21}(z, w) : R_1(\underline{x}, y, z), R_2(\underline{x'}, y, w), R_4(\underline{y}, u, d), R_3(\underline{u}, v)$

Queries Used On Real-world Data

$$
\begin{aligned}
Q_1() :& \text{NY\_Restaurants}(x, y, z, w, v) \wedge \text{CH\_Restaurants}(x, y', z', w', v') \\
Q_2(x) :& \text{NY\_Restaurants}(x, y, z, w, v) \wedge \text{CH\_Restaurants}(x, y', z', w', v') \\
Q_3(x) :& \text{NY\_Restaurants}(x, y, z, w, v) \wedge \text{CH\_Restaurants}(x, y', z', w', v') \\
& \wedge \text{NY\_Insp}(y, q, r, s, t) \wedge \text{CH\_Insp}(y', q', r, s', t') \\
Q_4(x, y) :& \text{CH\_Restaurants}(x, y, z, w, v) \wedge \text{CH\_Insp}(y, q, r, s, \text{`Pass'}) \\
Q_5(x, v) :& \text{CH\_Restaurants}(x, y, z, w, v) \wedge \text{NY\_Restaurants}(x, y', z', w', v') \\
& \wedge \text{NY\_Insp}(y', \text{`Not Critical'}, q, r, s) \\
Q_6(x) :& \text{CH\_Restaurants}(x, y, z, w, v) \wedge \text{CH\_Insp}(y, q, r, s, \text{`Fail'}) \\
& \cup \text{NY\_Restaurants}(x, y, z, w, v) \wedge \text{NY\_Insp}(y, q, r, s, \text{`Fail'})
\end{aligned}
$$

The Notion of Possible Answers

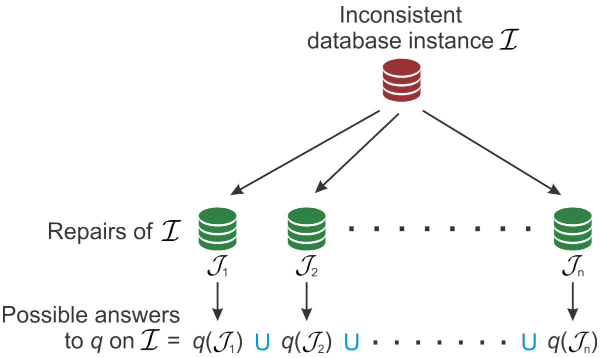


Figure: Possible answers to q on $\mathcal{I}$

The Notion of Potential Answers

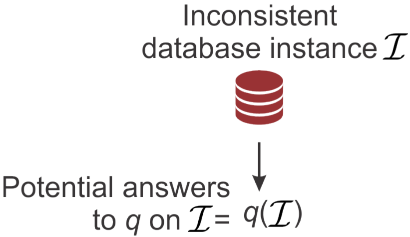


Figure: Potential answers to q on $\mathcal{I}$