SIGMOD: G: CAvSAT: A System for Query Answering over Inconsistent Databases

Akhil A. Dixit (Advisor: Prof. Phokion G. Kolaitis)  
akadixit@ucsc.edu  
University of California, Santa Cruz

1 PROBLEM AND MOTIVATION

Managing inconsistencies in databases is an old, but recurring, problem. An inconsistent database is a database that violates one or more integrity constraints, such as key constraints or functional dependencies. In the real-world, inconsistent databases arise due to various reasons and in several different contexts, including data warehousing and information integration, where dealing with inconsistency is regarded as a key challenge [10]. Inconsistencies are more of a norm than an exception, and they play a major role in a more general and broadly used term poor quality data, which, according to a report from IBM, cost $3.1 trillion to the US economy in 2016 [2]. Data cleaning is the main approach towards managing inconsistent databases (see the survey [26]). Some data cleaning approaches derive a consistent database by resolving all conflicts that exist in a given inconsistent database. This process often relies on clustering techniques, heuristics, and/or on particular domain knowledge. While data cleaning makes it possible to derive a single consistent database, this approach often involves making arbitrary choices for removing inconsistencies, and thus it is often ad hoc; for example, if a book has two different ISBN numbers stored in a database, there may not be an a priori reason to decide which one of the two should be removed.

An alternative, and arguably more principled, approach is the framework of database repairs and consistent query answering (CQA), introduced in [5]. Here, the inconsistencies are handled at query time by considering all possible repairs of the inconsistent database, where a repair of an inconsistent database I is a consistent database r that differs from I in a "minimal" way. The main algorithmic problem in this framework is to compute the consistent answers Cons(q, I) of a query q on a given database I, that is, the tuples that lie in the intersection of the results of q applied on each repair of I. Computing the consistent answers of a query q on I can be computationally harder than simply evaluating q on I, because an inconsistent database may have exponentially many repairs. In fact, the consistent answers exhibit a variety of computational behaviors even for simple conjunctive queries under key constraints, from SQL-rewritability to coNP-completeness.

There have been several attempts of building a system for CQA, including [7, 9, 15, 22, 23, 25, 28, 32, 33], but most of them remained as academic prototypes for various reasons. In particular, the ConQuer system [22, 23] is tailored to queries in the subclass (called C_forest) of SQL-rewritable queries. Other systems use logic programming [9, 25], compact representations of repairs [14], or reductions to solvers. Specifically, the system in [32] uses reductions to answer set programming, while the EQUIP system in [28] uses reductions to binary integer programming. It is fair to say, however, no comprehensive and scalable system for consistent query answering exists at present; this state of affairs has impeded the broader adoption of the framework of repairs and consistent answers as a principled alternative to data cleaning. The primary challenge in building a practical CQA system appears to be the high computational complexity of the consistent answers, and therefore the scalability of the system. In this work, we propose a novel approach based on boolean Satisfiability (SAT) to build a CQA system that scales well with the data, and supports broad classes of practical queries and integrity constraints. We also report experimental results obtained using our prototype CQA system CAvSAT (Consistent Answering via Satisfiability), implemented using this approach.

2 BACKGROUND AND RELATED WORK

Integrity Constraints and Database Queries. Relational database schemas are often accompanied by a set of integrity constraints that impose semantic restrictions on the allowable instances. A functional dependency x → y on a relation R is an integrity constraint asserting that if two tuples agree on the attributes in x, then they must also agree on the attributes in y. Typically, the set of all attributes of relation R is denoted by Attr(R). A primary key (or simply, key) is a minimal subset x of Attr(R) such that the functional dependency x → Attr(R) holds. Denial constraints is a broad class of integrity constraints that contains keys and functional dependencies as special cases. A denial constraint prohibits a set of tuples that satisfy certain conditions from appearing together in a database instance; for example, a restriction on a university database that each instructor can teach at most three courses per semester can be expressed as a denial constraint but not using keys or functional dependencies alone.
As is well known, first-order logic has been successfully used as a database query language [16, 17]. In fact, it forms the core of SQL, the main commercial database query language. A conjunctive query is a query expressible by a first-order formula of the form \( q(\mathcal{E}) := \exists \mathcal{W}(R_1(x_1) \land \ldots \land R_m(x_m)) \), where each \( x_i \) is a tuple consisting of variables and constants, \( \mathcal{E} \) and \( \mathcal{W} \) are tuples of variables, and the variables \( x_1, \ldots, x_m \) appear in exactly one of \( \mathcal{E} \) and \( \mathcal{W} \). A conjunctive query with \( k \) free variables in \( \mathcal{E} \) is a \( k \)-ary query, while a conjunctive query with no free variables (i.e., all variables are existentially quantified) is a boolean query. Conjunctive queries are also known as select-project-join (SPJ) queries and are among the most frequently asked queries in databases. For example, the binary conjunctive query \( q(s, t) := \exists c (\text{Enrolls}(s,c) \land \text{Teaches}(t,c)) \) returns the set of all pairs \((s, t)\) such that student \( s \) is enrolled in a course taught by teacher \( t \), while the boolean conjunctive query \( q() := \exists x, y, z (E(x, y) \land E(y, z) \land E(z, x)) \) tests whether a graph with an edge relation \( E \) contains a triangle. In the real-world, database queries often involve aggregation functions such as \( \text{MIN}, \text{MAX}, \text{SUM}, \text{COUNT}, \) or \( \text{AVG} \) on top of a conjunctive query. For example, an aggregation query \( Q := \text{SELECT DISTINCT COUNT}(x) \text{ FROM (Enrolls}(s,c) \land \text{Teaches}(\text{Lee}, c)) \) returns the number of students enrolled in the courses taught by teacher Lee.

Database Repairs and Consistent Answers. Let \( \mathcal{R} \) be a database schema and \( \Sigma \) be a set of integrity constraints on \( \mathcal{R} \). An \( \mathcal{R} \)-instance \( I \) is consistent if \( I \models \Sigma \), i.e., \( I \) satisfies every constraint in \( \Sigma \); otherwise, \( I \) is inconsistent. A repair of an inconsistent instance \( I \) w.r.t. \( \Sigma \) is a consistent instance \( J \) that differs from \( I \) in a “minimal” way. Different notions of minimality give rise to different types of repairs (see [11] for a comprehensive survey). Here, we focus on subset repairs, the most extensively studied type of repairs. An instance \( J \) is a subset repair of an instance \( I \) if \( J \subseteq I \) (where \( I \) and \( J \) are viewed as sets of facts), \( J \models \Sigma \), and there exists no instance \( J' \) such that \( J' \models \Sigma \) and \( J \subseteq J' \subseteq I \). Arenas et al. [5] used repairs to give rigorous semantics to query answering on inconsistent databases. Specifically, assume that \( q \) is a query, \( I \) is an \( \mathcal{R} \)-instance, and \( J \) is a tuple of values. We say that \( I \) is a consistent answer to \( q \) on \( I \) w.r.t. \( \Sigma \) if \( I \models q \), for every repair \( J \) of \( I \). We write \( \text{Cons}(q, I, \Sigma) \) to denote the set of all consistent answers to \( q \) on \( I \) w.r.t. \( \Sigma \), i.e.,

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

If \( \Sigma \) is a fixed set of integrity constraints and \( q \) is a fixed first-order query, then the main computational problem associated with the consistent answers is: given an instance \( I \), compute \( \text{Cons}(q, I, \Sigma) \). If \( q \) is a boolean conjunctive query, then computing the consistent answers becomes the decision problem \( \text{CERTAINTY}(q, \Sigma) \) : given an instance \( I \), is \( q \) true on every repair \( J \) of \( I \) w.r.t. \( \Sigma \) ?

Computational Complexity of Consistent Answers. By now, there has been an extensive study of CQA of conjunctive queries [11, 24, 27, 36, 37]. If \( \Sigma \) is a fixed finite set of denial constraints and \( q \) is a \( k \)-ary conjunctive query, where \( k \geq 2 \), then computing \( \text{Cons}(q, I, \Sigma) \) given an instance \( I \) is in coNP. Even for key constraints and boolean conjunctive queries, \( \text{CERTAINTY}(q, \Sigma) \) exhibits a variety of behaviors within coNP. The most definitive result to date is a trichotomy theorem [29–31]. This trichotomy theorem asserts that if \( q \) is a self-join-free (no repeated relation symbols) conjunctive query with one key constraint per relation, then \( \text{CERTAINTY}(q) \) is either SQL-rewritable, or in \( P \) but not SQL-rewritable, or coNP-complete. Moreover, there is a quadratic algorithm to decide, given such a query, which of the three cases of the trichotomy holds. It remains an open problem whether or not this trichotomy extends to arbitrary boolean conjunctive queries and to arbitrary denial constraints.

Boolean Satisfiability and SAT Solvers. Boolean Satisfiability (SAT) is the prototypical and the most widely studied NP-complete problem [18]. SAT asks the following: given a boolean formula, is it satisfiable? There has been extensive research about SAT-solving (see [12] for a survey), and this field of study has witnessed tremendous progress in the last few years (often referred as “SAT Revolution” [35]). Today’s SAT solvers are capable of solving SAT instances with millions of clauses and variables within a very short period of time. This success has resulted into widespread use of SAT solvers in industry as general-purpose problem-solving tools. Many real-world problems from a variety of domains such as scheduling, protocol design, software verification, model checking, and more can be naturally encoded as SAT instances, and solved quickly using solvers such as Glucose [8] and CaDiCaL [1] among others. Typically, a SAT solver takes a boolean formula in Conjunctive Normal Form (CNF) as an input and outputs a satisfying assignment (if one exists) or tells the formula is unsatisfiable. In an optimization variant of SAT, the clauses of the formula are assigned integer weights and the goal is to find an assignment that maximizes the sum of the weights of the satisfied clauses. This variant is known as Weighted MaxSAT, and modern solvers such as MaxHS [19] can solve Weighted MaxSAT instances very efficiently in practice.

3 APPROACH AND UNIQUENESS

We have been developing a comprehensive and scalable SAT-based system for CQA, namely, CAVSAT (Consistent Answers via Satisfiability). Figure 1 depicts its modular architecture. In the query pre-processor module, we leverage the extensive literature about the computational complexity of the consistent answers (e.g., [13, 27, 31, 38, 39]) to “route” the query to different modules tailored to handle the consistent answers of queries in a particular complexity class (SQL-rewritable, PTIME computable but not SQL-rewritable, and coNP-complete). The distinctive feature of CAVSAT is the SAT-solving module. Ours is the first CQA system that takes advantage of the progress made by the SAT-solving technology over the past few years and applies it to CQA. CAVSAT is also the first system to handle aggregation queries whose consistent answers (w.r.t. range semantics) are not SQL-rewritable. Furthermore, CAVSAT is flexible and extensible: when new classes of queries whose consistent answers are SQL-rewritable or tractable are identified, the system will be extended to incorporate these findings in the pre-processing stage, thus using a potentially more efficient evaluation strategy for computing the consistent answers of the queries at hand. Also, as the SAT-solving
technology progresses, newer solvers could be used to leverage such advances. The CAVSAT source code is available at the GitHub repository https://github.com/uccross/cavsat via a BSD-style open-source license.

![Diagram](image)

**Figure 1: Modular architecture of CAVSAT**

In what follows, we describe a reduction from CQA to WEIGHTED MAXSAT, which is used by CAVSAT in its SAT-solving module to compute the consistent answers to unions of conjunctive queries (UCQs) over databases that are inconsistent w.r.t. a set of arbitrary denial constraints.

### 3.1 Consistent Answers to UCQs via SAT Solving

Let $\mathcal{R}$ be a database schema, $\Sigma$ be a fixed finite set of denial constraints over $\mathcal{R}$, and $Q$ be a fixed union of conjunctive queries. Note that computing consistent answers to unions of conjunctive queries under denial constraints is still in coNP, but the consistent answers to a union $Q := q_1 \cup \ldots \cup q_k$ of conjunctive queries $q_1, \ldots, q_k$ is not, in general, equal to the union of the consistent answers to $q_1, \ldots, q_k$. In what follows, we give a polynomial-time reduction from CONS($Q, I, \Sigma$) to UnSAT. The reduction we give here relies on the notions of minimal violations and near-violations to the set of denial constraints that we introduce next.

**Definition 1. Minimal Violation.** Assume that $\Sigma$ is a set of denial constraints, $I$ is an $\mathcal{R}$-instance, and $S$ is a sub-instance of $I$. We say that $S$ is a minimal violation to $\Sigma$, if $S \not\models \Sigma$ and for every set $S' \subset S$, we have that $S' \models \Sigma$.

**Definition 2. Near-violation.** Assume that $\Sigma$ is a set of denial constraints, $I$ is an $\mathcal{R}$-instance, $S$ is a sub-instance of $I$, and $f$ is a fact of $I$. We say that $S$ is a near-violation to $\Sigma$ w.r.t. $f$, if $S \models \Sigma$ and $S \cup \{ f \}$ is a minimal violation of $\Sigma$. Special case: if $\{ f \}$ itself is a minimal violation to $\Sigma$, then we say that $f$ is exactly one near-violation w.r.t. $f$, and it is the singleton ($f_{true}$), where $f_{true}$ is an auxiliary fact.

**Reduction 1.** Given an $\mathcal{R}$-instance $I$, compute the following sets:

- $V$: the set of minimal violations to $\Sigma$ on $I$.
- $N^k$: the set of near-violations to $\Sigma$, on $I$, w.r.t. each tuple $f_i \in I$.
- $A$: the set of answers to $Q$ on $I$, i.e., the set $Q(I)$.
- $W^a$: the set of all minimal witnesses to $Q(a)$ on $I$, for each $a \in A$.

For each fact $f_i \in I$, introduce a boolean variable $x_i$. For the auxiliary fact $f_{true}$, let $x_{true} = true$. For each $N^k \in N^k$, introduce a boolean variable $y^k_j$, and for each $a \in A$, introduce a boolean variable $p_a$.

1. For each $V_j \in V$, construct a clause $a_j = \bigvee_{f_i \in V_j} \neg x_i$.
2. For each $a \in A$ and for each $W^a \in W^a$, construct a clause $\beta^a_j = \left( \bigwedge_{f_i \in W^a} \neg x_i \right) \land \neg p_a$.
3. For each $f_i \in I$, construct a clause $\gamma_i = x_i \lor \left( \bigvee_{N^k \in N^k} y^k_j \right)$.
4. For each variable $y^k_j$, construct an expression $\theta^k_j = y^k_j \leftrightarrow \left( \bigwedge_{f_a \in N^k} x_d \right)$.
5. Construct a boolean formula $\phi$ as follows:

$$\phi = \bigwedge_{i=1}^{|V|} a_i \land \bigwedge_{a \in A} \left( \bigwedge_{j=1}^{|W^a|} \beta^a_j \right) \land \bigwedge_{i=1}^{|A|} \left( \bigwedge_{j=1}^{|N^k|} \theta^k_j \right) \land \gamma_i$$

**Proposition 1.** Let $\phi$ be the boolean formula constructed using Reduction 1. There exists a satisfying assignment to $\phi$ in which a variable $p_a$ is set to true if and only if $a \not\models \text{Cons}(Q, I, \Sigma)$.

It follows from Proposition 1 that, if we find all $p_a$-variables such that they get set to true in at least one satisfying assignment to $\phi$, we can safely eliminate all their corresponding answers for being inconsistent. In practice, this can be done fast via a simple iterative algorithm mentioned in [20], that relies on a deployment of a state-of-the-art PARTIAL MAXSAT solver or a WEIGHTED MAXSAT solver to find satisfying assignments to $\phi$.

**Optimizations in Reduction 1.** The preceding reduction is generic, in the sense that it supports a broad class of integrity constraints, namely, arbitrary denial constraints. If we restrict the database schema to have only key constraints, however, a more efficient reduction can be applied [20]. Moreover, in real-life applications, a large portion of the inconsistent database is often consistent, and we leverage this fact by efficiently pre-computing the consistent answers that come from the consistent part of the database. This can be done with simple SQL queries that involve grouping on the key attributes of each relation. By doing so, we need boolean variables corresponding to only those tuples that contribute to the answers that are not pre-computed. This significantly reduces the size of the SAT instances produced by Reduction 1.

### 3.2 Consistent Answers to Aggregation Queries via SAT Solving

Let $\mathcal{R}$ be a database schema with one key constraint per relation, and $Q$ be the aggregation query

$$Q := \text{SELECT } f \text{ FROM } T(\bar{u}, w),$$

where $f$ is one of COUNT(+), COUNT(-), SUM($w$), MIN($w$), or MAX($w$), and $T(\bar{u}, w)$ is a relation expressed by a self-join-free conjunctive query on $\mathcal{R}$. Unlike a conjunctive query, an aggregation query is likely to return different answers in different repairs of an inconsistent database, and there may not be a
single consistent answer to it as per the preceding definition of Cons(q, I, Σ). In order to obtain more informative answers in such a case, Arenas et al. [6] introduced an extended notion of consistent query answers, called range semantics. The consistent answer to an aggregation query Q on an inconsistent database I w.r.t. the range semantics is a pair (glb, lub) of answers, i.e., the tightest range of answers, such that, glb ≤ q(J) ≤ lub holds for every repair J of I.

Ongoing work involves enabling CAvsAT to compute consistent answers to aggregation queries using SAT-solving. In what follows, we show how to compute the consistent answers (w.r.t. range semantics) to aggregation queries involving MIN function over databases with one key constraint per relation. Slight modifications can be made to Reduction 2 for the queries with other aggregation functions MAX, SUM, and COUNT.

Let Q := SELECT MIN(w) FROM T(u, w). Simply evaluating the query Q on I returns the glb-answer in Cons(Q, I, Σ). To see why this is true, observe that, i) no repair of I can produce a smaller answer to Q than the one returned by Q(I), and ii) there exists a repair R of I such that Q(R) = Q(I).

For computing the lub-answer in Cons(Q, I, Σ), we give a polynomial-time reduction from Cons(Q, I, Σ) to Weighted Partial MaxSAT.

Reduction 2. Given an Ρ-instance I, construct a Weighted Partial MaxSAT instance φ as follows. For each tuple f_i of I, introduce a boolean variable x_i. Let G be the set of key-equal groups of tuples of I, and W = {W_1, · · · , W_m} denote the set of minimal witnesses to a conjunctive query q on I, where q(w) := ∃u T(u, w) ∧ (w IS NOT NULL). For the set W, assume for 1 ≤ i ≤ m, we have that q(W_i) ≥ q(W_i+1).

- For each G_j ∈ G, construct a hard clause a_j = ∨ f_i∈G_j x_i.
- Construct a hard clause b_j = ∨ f_i∈W_j ¬x_i, and let its weight be c.
- Construct a Weighted Partial MaxSAT instance
  φ = \left( \bigwedge_{j=1}^{G} a_j \right) \land \left( \bigwedge_{j=1}^{W} b_j \right).

Proposition 2. Let φ be a maximum satisfying assignment to the Weighted Partial MaxSAT instance φ constructed using Reduction 2. Let b_j be a clause such that φ(b_j) = false, and there is no j’ < j such that φ(b_j’) = false. Then, q(W_j) is the lub-answer in Cons(Q) on I.

4 RESULTS AND CONTRIBUTIONS

In the first set of experiments, we compared the performance of the SAT-solving module of CAvsAT against the SQL-rewritings of seven conjunctive queries chosen from the literature [28], over a synthetically generated database (1 million tuples/relation with 10% inconsistency w.r.t. primary keys). For queries q_1, · · · , q_7, we computed the SQL-rewritings using the algorithm of Koutris and Wijsen [31] (referred as KW-SQL-rewriting). Since the queries q_1, · · · , q_4 happen to be in the class C_forest, we computed additional SQL-rewritings for them using the algorithm implemented in the CQA system ConQuer [22] (referred as ConQuer-SQL-rewriting). Figure 2 shows that for the queries q_1, · · · , q_4 in the class C_forest, the performance of the SAT-solving module was slightly worse, but comparable, to their ConQuer-SQL-rewritings. For all seven queries q_1, · · · , q_7, however, SAT-based approach significantly outperformed KW-SQL-rewritings, as the database engine hit the two hours time-out while evaluating each KW-SQL-rewriting.

![Figure 2: Performance of the SAT-solving module of CAvsAT against the SQL-rewriting approaches](image)

Next, we considered fourteen additional conjunctive queries whose consistent answers are coNP-complete or in P but not SQL-rewritable. Figure 3 shows that the time required for constructing the Weighted MAXSAT instances dominates over the time taken to solve them and eliminate inconsistent answers. The solver took comparatively more time for the queries that have higher number of free variables or joins. Observe that even for the queries with intractable consistent answers, the performance of the SAT-solving module did not degrade too much compared to the SQL-rewritable queries.

![Figure 3: Performance of the SAT-solving module for conjunctive queries of varying data complexity](image)
data have been previously used for evaluating data cleaning systems, such as HoloClean [34]. We evaluated the performance of the SAT-solving module on five conjunctive queries $Q_1, \cdots, Q_5$ and one union $Q_6$ of conjunctive queries. The optimizations mentioned in Section 3.1 were not applied during this set of experiments. We observed that the SAT solver took more time to compute consistent answers to query $Q_6$ due to high number of joins, and the union $Q_6$ due to high number of minimal witnesses (Figure 4). All aforementioned experiments were carried out on a machine running on Intel Core i7 2.7 GHz, 64 bit Ubuntu 16.04, with 8 GB RAM. The precise definitions of the queries used in the experiments and additional experimental results can be found in our paper [21].

![Figure 4: Performance of the SAT-solving module on real-world data](image)

The findings from Figure 2 suggest a potential difference between theory and practice, since the study of SQL-rewritability of the consistent answers was motivated from having an efficient evaluation of consistent answers using the database engine alone. To investigate further, we conducted experiments on large TPC-H benchmark databases (12 million tuples/relational with 5% inconsistency w.r.t. primary keys) and fifteen TPC-H-inspired queries, on Microsoft Azure SQL Database (200 DTUs). Queries $Q_{11}, \cdots, Q_{15}$ were in $C_{\text{forest}}$, thus we computed their consistent answers using ConQuer’s algorithm [22]. Table 1 shows the comparison of the evaluation time in seconds for computing the consistent answers. The SAT-solving approach outperformed the ConQuer SQL-rewriting for five out of nine queries, once again highlighting the gap between theory and practice. For queries $Q_{13}$ and $Q_{18}$, however, the reason behind the poor performance of the SAT-solving module appears to be the high number of joins between relatively large relations causing significant increase in the size of the SAT instances. The column potential answers refers to the time taken to directly evaluate the input query on the inconsistent database and helps understand the overhead incurred due to computing consistent answers using both SAT-based and SQL-rewriting approaches.

### Conclusions

We designed and implemented CAvSAT, the first SAT-based system for consistent query answering. CAvSAT leverages a set of natural reductions from the complement of CQA to SAT and its variants. The system is currently capable of handling unions of conjunctive queries and arbitrary denial constraints. Ongoing work involves computing consistent answers to aggregation queries w.r.t. range semantics. Our experimental evaluation provides evidence that a SAT-based approach is indeed promising, and can give rise to a comprehensive and scalable system for CQA. The next step in this investigation is to carry out an extensive comparative evaluation of CAvSAT with other systems for CQA that can handle the queries whose consistent answers are not SQL-rewritable, in particular, with systems that use reduction-based methods [28, 32]. A longer-term and more ambitious endeavor is to carry out a comparison between CAvSAT and a data cleaning system such as HoloClean [34] on various fronts including performance and semantic guarantees. This is indeed a challenging endeavor because it will require first to develop a methodology for carrying out a meaningful comparison between two fundamentally different approaches to handle inconsistent databases, namely, data cleaning and consistent query answering.

### REFERENCES


