Query rewriting with filtering constraints [*†]

Eric Jui-Yi Kao
Stanford University
353 Serra Mall
Stanford, California, United States of America
erickao@cs.stanford.edu

Michael Genesereth
Stanford University
353 Serra Mall
Stanford, California, United States of America
genesereth@stanford.edu

ABSTRACT

A database may contain inaccurate information. When querying such a database, one may want to impose filtering constraints so that only answers produced from reasonable subsets of the data are presented. We call the problem of producing these answers with respect to a set of filtering constraints the Reasonable Query Answers (RQA) problem.

The general problem is NP-Hard in data-complexity, but we identify broad, tractable cases. We give a data-complexity map of the RQA problem over various classes of queries and constraints. In the intractable cases, we prove hardness. In the tractable cases, we provide query rewriting solutions.

1. INTRODUCTION

A database may become inconsistent with constraints which we expect totally accurate data to satisfy. In such cases, standard query semantics can produce unreasonable answers.

In this paper, we use constraints to filter out unreasonable answers. We define the *reasonable query answers*, analyze the complexity of computing such answers, and finally give query rewriting methods for finding these answers.

1.1 Why is there inconsistency in databases?

Databases are typically assumed to be consistent with the applicable integrity constraints because database management systems typically disallow updates that would create inconsistencies in the data. However, inconsistencies still arise in many scenarios:

Integration of autonomous data sources. When integrating data from autonomous sources, the result may violate constraints even if each autonomous source itself is consistent with all constraints. For instance, two sources of data may show two different birth years for the same person because at least one of the sources is incorrect (but we don't know which one). In addition to out-of-sync data and erroneous data, data from different sources may be inconsistent when combined due to semantic or fundamental disagreements. For example, two data sources may claim two different birth years for Julius Caesar because of genuine disagreement over which year he was born.

Unenforced constraints. A database system may fail to enforce consistency for many reasons. Some applicable constraints may have been unspecified at some point in the life of the database. A database may be a legacy system that does not support consistency enforcement. A database (especially a highly distributed database) may elect for efficiency reasons not to strictly enforce global consistency. Sometimes a database system does not support the enforcement of all types of applicable constraints.

Information preservation. Sometimes inconsistencies are allowed to exist in order to preserve information. For example, it may be desirable in a customers database to keep two different last names for the same customer while representatives attempt to contact the customer to determine which one is correct.

The data in table 1 will serve as example data. The BirthYear table associates each person with a birth year. Each person has at most one birth year, but in the example data to conflicting birth years are specified for Cody (and David) because they come from two sources, at least one of which is incorrect. The RecessionYear table gives the recessionary years in the United States.

Table 1: Example data

BirthYear

Person	Year
Adam	1980
Brian	1991
Cody	1984
Cody	1991
David	1980
David	1984

RecessionYear

Year
1980
1981
1982
1990
1991

Table 2: Example views

RecessionBorn

Person
Adam
Brian
Cody
David

View definition:

```
RecessionBorn(X) :-
BirthYear(X,Y),
RecessionYear (Y)
```

OfAge

Person
Adam
Cody
David

View definition:

```
OfAge(X) :-
BirthYear(X,Z),Z<1989
```

1.2 What is an unreasonable answer?

Inconsistencies are problematic in reasoning because, in classical logic, an inconsistency entails every sentence in the given language. In querying databases, the problem is less serious because the standard query semantics ignores integrity constraints and answers the query based on a model of the data (which is always

[*]This work was supported by the Collaborative Data Management Initiative.
[†]This work was supported by the National Science Foundation under award number IIS-0841152. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the National Science Foundation.

consistent because negative information is implicit). Nonetheless, the standard semantics can produce unreasonable answers which are justified only by unreasonable data that violates the constraints.

Consider the following scenario: For a study, a researcher would like to send surveys out to those who were born during a recessionary year in the United States and are also legally of age. So using the views defined in table 2, the researcher poses the following query: `ANS(X) = RecessionBorn (X) ∧ OfAge(X)`

Standard database semantics, which do not consider the constraints, would produce the answers {`Adam`, `Cody`, `David`}, the natural join between the two views. By digging down, we see that `RecessionBorn(Cody)` is derived from `BirthYear(Cody, 1991)` and `OfAge(Cody)` is derived from `BirthYear(Cody, 1984)`. But at least one of these two pieces of data is incorrect. We see that `Cody` is not a reasonable answer because whether we choose `BirthYear(Cody, 1984)` or `BirthYear(Cody, 1991)`, `Cody` does not satisfy the criteria intended by the query. We call such answers *unreasonable answers*.

The goal of this paper is to propose query answering semantics and algorithms that avoid unreasonable answers.

Consider the two conflicting tuples `BirthYear(David, 1980)` and `BirthYear(David, 1984)`. If we chose to believe `BirthYear(David, 1980)`, then `David` is an answer. But if we chose to believe `BirthYear(David, 1984)`, then `David` is not an answer. Since we have no information that leads us to believe one of the two conflicting tuples over the other, `David` is not an unreasonable answer.

1.3 Reasonable Query Answers

We can specify the constraint that each person has at most one birth year as a filtering constraint to filter out answers that depend on two different birthyears for the same person. Intuitively, the answer `Cody` depends on using both `BirthYear(Cody, 1984)` and `BirthYear(Cody, 1991)` together, so it should be filtered out by the filtering constraint $\forall X Y_1 Y_2(\neg \texttt{BirthYear}(X, Y_1) \vee \neg \texttt{BirthYear}(X, Y_2) \vee Y_1 = Y_2)$.

If we confine ourselves to conjunctive queries[1] and denial constraints[2], we can formalize the intuition as follows: Given filtering constraints C and a query Q, a tuple $\bar{t}$ is an RQA answer over database instance D if and only if there exists a $D^* \subseteq D$ such that D^* is consistent with C and $\bar{t}$ is a standard answer[3] over D^*.

For queries involving negation, the formal definition (to be given in section 2.3) is more complex, but the intuition remains the same.

Here, we briefly distinguish RQA from the well-known Consistent Query Answers [12, 3, 6] (CQA) and its dual, the possible answers from possible minimal-change repairs (possible answers for short). A more detailed comparison is deferred to section 9.

RQA takes a "credulous" approach under which reasonable but uncertain answers (e.g., `David` in the foregoing example) are admitted whereas CQA rejects such answers.

Unlike possible answers, RQA uses constraints in a "filtering-only" role rather than a "tuple-generating" role. As a consequence, the set of standard answers contains the set of RQA answers. In contrast, the possible answers semantics may generate many additional answers that are added in the repairs process. For example, if a constraint says every student must have at least one advising professor but a student has none listed because of missing information, then every professor is considered a possible answer as the student's advising professor.

A straightforward method for computing the reasonable query answers, as informally defined earlier, is to find and query each possible database. However, the number of consistent subsets of a database (even if restricted to maximal subsets) may be exponential in the size of the data. In fact, we will show that the problem of computing reasonable query answers is NP-Hard in data complexity[4].

1.4 Summary of contributions

In this paper, we map out the data complexity of the RQA problem over various classes of queries and constraints. In the intractable cases, we provide hardness proofs. In the tractable cases, we provide practical query rewriting solutions.

Main contributions:

- A formalization of reasonable answers. (RQA, Section 2)
- The NP-Hardness (in data complexity) of the RQA problem. (Section 3)
- An RQA query rewriting procedure for existential queries (E*) and universal constraints (A*). A polynomial data complexity procedure for computing RQA follows immediately. (Section 4)
- An RQA query rewriting procedure for E*A* queries[5] and sets of finitely-closed[6] universal constraints. A polynomial data complexity procedure for computing RQA follows immediately. (Section 7)

The results are summarized in the following table of data complexity of the RQA problem for several classes of queries and constraints:

	Constraint Set		
Query	Ground	Finitely-closed A*	A*
E*	P (♣)	P (♭)	P (♭)
E*A*	P (♣)	P (‡)	Open
A*E*	P (♣)	NP-Hard (♮)	NP-Hard (♮)
`CALC`	P (♣)	NP-Hard (♮)	NP-Hard (♮)

`CALC` denotes the class of relational calculus queries (defined in section 2.1.1). E*, E*A*, and A*E* denote, respectively, the $\exists^*$, $\exists^*\forall^*$, and $\forall^*\exists^*$ quantifier prefix fragments of relational calculus (defined in section 2.1.1). A ground constraint set is a set of ground (without variables, bound or free) relational calculus sentences.
The results denoted by ♮ are established by theorem 1 in section 3.
The results denoted by ♭ are established by theorem 2 in section 4.
The result denoted by ‡ is established by theorem 5 in section 7.
Finally, the results denoted by ♣ are straightforward and included as theorem 6 in section 8 for completeness.

2. FORMAL DEVELOPMENT

This section builds the necessary preliminaries and define reasonable query answers. The section also serves to fix the terminology and notations used throughout the paper.

2.1 Relational databases

We fix a countably infinite set **dom** as the *underlying domain*. A *constant* is an element of **dom**. A database schema S is a nonempty, finite set of relation names, each with a fixed arity. A database

[1] Existentially quantified conjunction of positive literals

[2] universally quantified disjunction of negative literals and equality literals

[3] That is, $D^* \models Q(\bar{t})$, as formally defined in section 2.1.2.

[4] Over the class of relational calculus queries and constraints.

[5] Relational calculus queries with $\exists^*\forall^*$ quantifier prefix

[6] A finitely-closed set of constraints is one whose closure under resolution is finite. (See section 5)

instance of a schema S is a finite set of facts (ground atoms) over S and **dom**.

We adopt relational calculus rather than relational algebra or Datalog to represent the queries because relational calculus allows for a simpler presentation of the results and the algorithms. This choice does not affect the substance of the results in any way. The class of domain independent relational calculus queries express the same query mappings as relational algebra [18].

2.1.1 Query syntax

A relational calculus query is a well-formed formula over S [1, pp 74-75, Definition 5.3.5]. We fix a countably-infinite set of variables VAR which is disjoint from **dom**. A *term* is a constant or a variable. A *database atom* over S is an expression of the form $R(t_1, \ldots, t_n)$ where R is an n-ary relation symbol in S and each t_i is a term. An *equality atom* is an expression of the form $t_1 = t_2$ or $t_1 \neq t_2$ where t_1, t_2 are terms. The *base formulas* are (database and equality) atoms over S. The well-formed formulas of the relational calculus over S are the base formulas and formulas of the form

1. $(\phi \wedge \psi)$, where ϕ and ψ are formulas over S;
2. $(\phi \vee \psi)$, where ϕ and ψ are formulas over S;
3. $\neg\phi$, where ϕ is a formula over S;
4. $\exists x\phi$, where x is a variable and ϕ is a formula over S;
5. $\forall x\phi$, where x is a variable and ϕ is a formula over S.

We will sometimes use an additional logical connective $\rightarrow$. We view $\phi \rightarrow \psi$ as a syntactic abbreviation for $\neg\phi \vee \psi$. A query is said to be closed if it has no free variables.

We restrict queries to domain independent queries [21; 37; 1, pp 79, Definition 5.3.7].

2.1.2 Standard Query semantics

The standard model theoretic semantics is adopted for standard query answers over database instances. [1, pp 77-79, Definition 5.3.5]

Given D, ϕ, and a valuation ν which maps $\text{FREE}(\phi)$[7] to **dom**, D *entails* ϕ for ν, denoted $D \models \phi[\nu]$, if

1. $\phi = p(u)$ is an atom and $p(\nu(u)) \in D$;
2. $\phi = (s = s')$ is an equality atom and $\nu(s) = \nu(s')$;
3. $\phi = (s \neq s')$ is an equality atom and $\nu(s) \neq \nu(s')$;
4. $\phi = (\psi \wedge \xi)$ and $D \models \psi[\nu|_{\text{FREE}(\psi)}]$[8] and $D \models \xi[\nu|_{\text{FREE}(\xi)}]$;
5. $\phi = (\psi \vee \xi)$ and $D \models \psi[\nu|_{\text{FREE}(\psi)}]$ or $D \models \xi[\nu|_{\text{FREE}(\xi)}]$;
6. $\phi = \neg\psi$ and $D \not\models \psi[\nu]$ (i.e., $D \models \psi[\nu]$ does not hold);
7. $\phi = \exists x\psi$ and for some $c \in$ **dom**, $D \models \psi[\nu \cup \{x \mapsto c\}]$; or
8. $\phi = \forall x\psi$ and for each $c \in$ **dom**, $D \models \psi[\nu \cup \{x \mapsto c\}]$.

For a closed formula ϕ, $D \models \phi[\nu]$ is independent of ν. In this case, we simply omit the valuation.

For a set of closed relational calculus sentences Φ, we overload the notation, letting $D \models \Phi$ denote that $D \models \phi$ for every $\phi \in \Phi$.

Definition 1 (Standard answers) *We say that a tuple $\bar{t}$ is a standard answer to query $Q(\bar{x})$, over database instance D if $D \models Q(\bar{t})$.*

[7]For a formula ϕ, $\text{FREE}(\phi)$ denotes the free variables of ϕ.

[8]$\nu|_V$ for a variable set V denotes the restriction of ν to V.

2.1.3 Query classes

We define several subclasses of queries to be referred to in later discussion.

CALC (relational calculus) is the class of domain independent queries expressible by relational calculus.

E* queries is the subclass of CALC queries expressible by a formula consisting of a prefix of zero or more existential quantifiers followed by a quantifier-free part called the *matrix*.

CQ (conjunctive queries) is a subclass of E* queries expressible by an E* formula whose matrix is a conjunction of atoms.

CQ$\neg$ (conjunctive queries with negation) is a subclass of E* queries expressible by an E* formula whose matrix is a conjunction of literals (an atom or the negation of an atom).

E*A* queries is the subclass of CALC queries expressible by a formula consisting of a prefix of zero or more existential quantifiers followed zero or more universal quantifiers and ending with the quantifier-free matrix.

A*E* queries is the subclass of CALC queries expressible by a formula consisting of a prefix of zero or more universal quantifiers followed zero or more existential quantifiers and ending with the quantifier-free matrix.

2.1.4 Incomplete database

For defining a relaxation of a database instance, we use the notion of an incomplete database. An incomplete database of a schema S is a collection[9] of facts and negated facts over S and **dom**. We assume that an incomplete database never contains both a fact and its complement.[10]

The semantics of an incomplete database T is given by its set of instances. The set of instances of an incomplete database T is simply the set of (complete) database instances that satisfy T. Formally, for an incomplete database T,

$$\text{INSTANCES}(T) = \{B \text{ a database instance } : B \models \phi \text{ for each } \phi \in T\}$$

The standard query semantics for (complete) instances is naturally extended as follows to incomplete databases. For an incomplete database T and a query Q,

$$T \models^{\text{I}} Q \text{ if and only if } I \models Q \text{ for each } I \in \text{INSTANCES}(T)$$

The superscript 'I' is used to emphasize that we are dealing with incomplete databases.

Each (complete) database instance is semantically equivalent to an incomplete database that in obtained by making the implicit negative information explicit. We call the incomplete database thus obtained the explicit representation of a (complete) database instance. The explicit representation of B is defined as follows (the schema S is implicit): $\text{EXPLICIT}(B) =_{def}$
$B \cup \{\neg r(\bar{t}) \cdot r \in S, d_r \text{ is the arity of } r, \bar{t} \in \textbf{dom}^{d_r}, r(\bar{t}) \notin B\}$.[11]

It is clear that the definition of query answers over incomplete database is monotonic.

Proposition 1 *Let B_1,B_2 be incomplete databases such that $B_1 \subseteq B_2$. Then for any query Q and tuple of constants $\bar{t}$, if $B_1 \models^{\text{I}} Q(\bar{t})$ then $B_2 \models^{\text{I}} Q(\bar{t})$.*

[9]The collection may be infinite. Infinite collections are used only as intermediate concepts to define RQA. Computationally, these infinite collections are never constructed or directly handled.

[10]The concept of incomplete databases is more general, but for our purpose we consider only those incomplete databases representable as a collection of facts and negated facts.

[11]$\text{EXPLICIT}(B)$ is always infinite because B is finite and **dom** is infinite.

2.2 Constraints and inconsistency

A constraint is any closed relational calculus formula. A database instance is always consistent on its own. A database instance may be inconsistent with respect to a set of constraints. We adopt the following definition of inconsistency of a database instance with respect to a set of constraints.

Let B be a database instance. B is inconsistent with a constraint set C if and only if $B \models \neg c$ for some $c \in C$.

Otherwise, B is said to be consistent with C.

Similarly, for an incomplete database T, T is inconsistent with a constraint set C if and only if $T \models^{\mathrm{I}} \neg c$ for some $c \in C$.

We say that a set Φ of relational calculus sentences is *satisfiable* if there exists a database instance B such that $B \models \phi$ for every $\phi \in \Phi$. Otherwise, Φ is *unsatisfiable*.

2.3 Reasonable Query Answers

In order to discuss precisely the idea of using consistent data to support an answer, we define the notion of a *consistent relaxation* of a database instance.

Definition 2 *An incomplete database B' is a consistent relaxation of a database instance B, w.r.t. a set of constraints C, if and only if*

- $B' \subseteq \textsc{Explicit}(B)$ *(B' is a relaxation of B).*
- *B' is consistent with C*

The syntactic condition that $B' \subseteq \textsc{Explicit}(B)$ is equivalent to the semantic conditions that $B \in \textsc{Instances}(B')$.

Then we define a reasonable query answer as an answer that is supported by a consistent relaxation.

Definition 3 (Reasonable Query Answers) *A tuple $\bar{t}$ is a reasonable query answer (RQA) to a query $Q(\bar{x})$ over a database instance B with respect to constraints C if and only if there exists a consistent relaxation B' of B w.r.t. C such that $B' \models^{\mathrm{I}} Q(\bar{t})$.*[12]

This fact shall be denoted by $B \models_C Q(\bar{t})$

2.4 Properties of RQA

Here we discuss several simple properties of RQA.

Proposition 2 (Distribution over disjunction) *Let B be a database instance and Q_1,Q_2 each be a relational calculus query. Then for any tuple $\bar{t}$, $B \models_C Q_1(\bar{t}) \vee Q_2(\bar{t})$ if and only if $B \models_C Q_1(\bar{t})$ or $B \models_C Q_2(\bar{t})$.*

The next proposition states that RQA answers are a subset of standard answers.

Proposition 3 (Containment by Standard Answers) *Let B be a database instance, Q be a* CALC *query and $\bar{t}$ a tuple of constants. If $B \models_C Q(\bar{t})$ then $B \models Q(\bar{t})$.*

It is intuitive that if no constraints were specified, RQA reduces to standard answers.

[12]Some may worry that the definition allows for the selection of relaxations that relax more than necessary. In fact, the definition remains the same even if one restricted the relaxations considered to maximally strong consistent relaxations. This is because $\models^{\mathrm{I}}$ is monotonic. A relaxation that is weaker than necessary may "miss" some answers (that would be covered by another relaxation considered), but cannot "gain" answers.

Proposition 4 (Reduction under Empty Constraint) *Let $C = \{\}$. Then for any database instance B,* CALC *query Q, and a tuple of constants $\bar{t}$, $B \models_C Q(\bar{t})$ if and only if $B \models Q(\bar{t})$.*

3. PROBLEM COMPLEXITY

To investigate the data complexity of answering queries under RQA, we define the RQA decision problem with fixed constraints and query.

Problem: Let Q be a fixed, closed query. Let C be a fixed set of constraints. Given as input a database instance D, the problem is to decide whether $D \models_C Q$. Written as a set, the RQA decision problem is $\mathtt{RQA}_{Q,C} = \{D : D \models_C Q\}$, where D ranges over database instances.

We show that the RQA problem is NP-Hard by reduction from 3-CNF-SAT, a well-known NP-complete problem [32].

Let Φ be a 3 CNF formula.

We number each clause in Φ of literals from one to n.

For each instance of 3-CNF-SAT as described above, form the corresponding database instance as follows:

$B_\Phi = \{clause(i) : i \in \{1, \ldots, n\}\} \cup$
$\{r(i, p, +) :$ proposition p occurs positively in clause $i\} \cup \{r(i, p, -) :$ proposition p occurs negatively in clause $i\}$

We also specify a set of constraints C consisting of a fixed functional dependency
$\forall x, y, p, s_1, s_2 : \neg r(x, p, s_1) \vee \neg r(y, p, s_2) \vee s_1 \neq s_2$. (We can also use a simpler constraint $\forall x, y, p : \neg r(x, p, +) \vee \neg r(y, p, -)$.)

The formula Φ is satisfiable if and only if we can select a literal from each disjunction in such a way that no proposition and its negation are both selected. That condition holds if and only if $B_\Phi \models_C \forall x \exists p, s : (clause(x) \rightarrow r(x, p, s))$.

Theorem 1 *Let $C = \{\forall x, y, p : \neg r(x, p, +) \vee \neg r(y, p, -)\}$. Let $Q = \forall x \exists p, s : (clause(x) \rightarrow r(x, p, s))$. The decision problem $\mathtt{RQA}_{Q,C}$ is NP-Hard.*

PROOF. Constructing B_Φ takes up to $4n$ insertions into an initially empty database. The construction takes time polynomial in the size of Φ. To finish the polynomial reduction, we show that a SAT instance Φ is satisfiable if and only if $B_\Phi \models_C Q$

($\Rightarrow$)

Suppose the SAT instance Φ is satisfiable. We can choose one literal out of each clause in Φ such that no literal and its complement are both chosen. Equivalently, we can choose for each i from 1 to n a database fact $r(i, p, s) \in B_\Phi$ where p is a SAT proposition and s is - or +, such that for no triple i,j and p, both $r(i, p, +)$ and $r(j, p, -)$ are chosen. Let the chosen set of database facts be B^*. B^* is consistent with C and $B^* \models^{\mathrm{I}} Q$.

($\Leftarrow$)

Suppose there exists $B^* \subseteq \textsc{Explicit}(B_\Phi)$ such that B^* is consistent with C and $B^* \models^{\mathrm{I}} Q$. $B^* \models^{\mathrm{I}} Q$ so for each i from 1 to n, there is a p and a s such that $r(i, p, s) \in B^*$ or $\neg clause(i) \in B^*$. But for all i from 1 to n, $\neg clause(i) \notin \textsc{Explicit}(B)$. And, for all $r(i, p, s) \in \textsc{Explicit}(B)$, p is a SAT proposition and s is + or -. So we can obtain a subset B^{**} by choosing for each i from 1 to n, $r(i, p, +) \in B^*$ or $r(i, p, -) \in B^*$, where p is a SAT proposition.

Let $M = \{p :$ there exists i such that $r(i, p, +) \in B^{**}\} \cup \{\neg p :$ there exists i such that $r(i, p, -) \in B^{**}\}$. M contains one literal from each clause of Φ. Furthermore, M does not contain a literal and its complement, because otherwise for some i,j and p, B^{**} and hence B^* contains both $r(i, p, +)$ and $r(j, p, -)$, then B^* is inconsistent with C, a contradiction. We conclude that Φ is satisfiable. $\square$

4. REWRITING E* QUERIES

First we define what is an RQA rewriting.

Definition 4 (RQA rewriting) *Given a relational calculus query Q and constraints C, Q' is an RQA rewriting of Q with respect to C if and only if for every database instance B, the RQA answers to Q are precisely the standard answers to Q' in B. That is,*

$B \models_C Q(\bar{t})$ *if and only if* $B \models Q'(\bar{t})$*, for every tuple* $\bar{t}$*.*

Before proceeding to the rewriting algorithm, we note that simply adding the entire C as a condition on the query does not produce a correct rewriting because the answer is always empty on any inconsistent database. However, we will use the idea of folding the constraints into the query.

First we make a basic observation.

Lemma 1 *Let $Q = p_1 \wedge \cdots \wedge p_k$ be a ground $\mathtt{CQ}\neg$ query (closed, quantifier-free query). Let C be a set of constraints. For any database instance B, $B \models_C Q$ if and only if $B \models Q$ and $\{Q\} \cup C$ is satisfiable.*

Example 1 $Q_1 = p(a) \wedge \neg q(b)$, $Q_2 = p(a) \wedge \neg q(c)$. $C = \{\neg p(a) \vee q(c)\}$. $B = \{p(a)\}$.

$B \models Q_1$ and $\{Q_1\} \cup C$ is satisfiable. We can check that $B \models_C Q_1$, in agreement with lemma 1.

$B \models Q_2$ but $\{Q_2\} \cup C$ is unsatisfiable. We can check that $B \not\models_C Q_2$, in agreement with lemma 1.

$\diamond$

The following corollary follows immediately from Lemma 1.

Corollary 1 *Let $Q(\bar{X}) = p_1 \wedge \cdots \wedge p_k$ be a quantifier-free $\mathtt{CQ}\neg$ query with free variables $\bar{X}$. Let C be a set of constraints. For any database instance B, $B \models_C Q(\bar{t})$ if and only if $B \models Q(\bar{t})$ and $Q(\bar{t}), C$ is satisfiable.*

These observations reduce the task of rewriting a quantifier free conjunctive query with negation to that of constructing a consistency condition of a query with respect to a set of constraints.

Definition 5 (Consistency Condition) *Let $Q(\bar{X}) = p_1 \wedge \cdots \wedge p_k$ be a quantifier-free CQ$\neg$ query with free variables $\bar{X}$. Let C be a set of constraints. We call a Boolean combination of equalities and disequalities $\Xi_C^Q(\bar{X})$ a consistency condition of Q with respect to C if for every tuple $\bar{t}$ such that $Q(\bar{t})$ is satisfiable, $\Xi_C^Q(\bar{t})$ evaluates to true if and only if $Q(\bar{t}), C$ is satisfiable.*

The following corollary follows immediately from Corollary 1.

Corollary 2 *Let $Q(\bar{X}) = p_1 \wedge \cdots \wedge p_k$ be a quantifier-free CQ$\neg$ query with free variables $\bar{X}$. Let C be a set of constraints. Let $\Xi_C^Q(\bar{X})$ be a consistency condition of Q with respect to C.*

For any database instance B and every tuple $\bar{t}$, $B \models_C Q(\bar{t})$ if and only if $B \models Q(\bar{t}) \wedge \Xi_C^Q(\bar{t})$.

Deferring the task of constructing such a consistency condition to the next section, we define a rewriting scheme for quantifier-free conjunctive queries with negation.

Let $Q(\bar{X}) = p_1 \wedge \cdots \wedge p_k$ be a quantifier-free CQ$\neg$ query with free variables $\bar{X}$. Let C be a set of constraints. Let $\Xi_C^Q(\bar{X})$ be a consistency condition of Q with respect to C.

Rewrite$[Q, C] =_{def} Q(\bar{X}) \wedge \Xi_C^Q(\bar{X})$.

Using lemma 2 (below) to pass through the existential quantifier, we generalize the rewriting to conjunctive queries with negation.

$$\text{Rewrite}[\exists \bar{X} M, C] =_{def} \exists \bar{X} \text{Rewrite}[M, C]$$

Lemma 2 *Let $Q(\bar{X}) = \exists \bar{Y} F(\bar{X}, \bar{Y})$ be a $\mathtt{CQ}\neg$ query with free variables $\bar{X}$. Then for any database B and any tuple $\bar{t}$, $B \models Q(\bar{t})$ if and only if there exists a tuple $\bar{t_2}$ such that $B \models F(\bar{t}, \bar{t_2})$.*

Finally, we generalize the rewriting to all E* queries using proposition 2 to distribute over disjunctions.

For a E* query $Q = Q_1 \vee \cdots \vee Q_k$, where each $Q_i \in CQ\neg$, define Rewrite$[Q, C] =_{def}$ Rewrite$[Q_1, C] \vee \cdots \vee$ Rewrite$[Q_k, C]$.

Here we summarize the rewriting procedure for E* queries: given a E* query $Q(\bar{x}) = \exists x_1 Q_1 \vee \cdots \vee \exists x_k Q_k$, where each Q_i is a quantifier-free formula in $CQ\neg$ and $\bar{x} = \text{FREE}(\exists x_1 Q_1 \vee \cdots \vee \exists x_k Q_k)$, Rewrite$[Q, C] =_{def} \exists x_1 (Q_1 \wedge \Xi_C^{Q_1}) \vee \cdots \vee \exists x_k (Q_k \wedge \Xi_C^{Q_k})$, where $\Xi_C^{Q_i}$ is a consistency condition of Q_i with respect to C.

The correctness of the procedure follows.

Theorem 2 *Let Q be a E* query. Then* Rewrite*$[Q, C]$ is an RQA rewriting of Q with respect to C.*

5. CONSTRUCTING A CONSISTENCY CONDITION

An obvious way to construct a consistency condition is to find out, for each tuple $\bar{t}$, whether $Q(\bar{t})$ is consistent with C. Then, construct an expression that enumerates all the tuples $\bar{t}$ such that $Q(\bar{t}), C$ is satisfiable. A problem immediately arises: if, say, there are no constraints at all, then every tuple of the right length would be enumerated and the resulting consistency condition. In this section, we describe a method to construct a more compact consistency condition.

5.1 Basic logic computations

First, we introduce some logic computations that will be useful.

A clause is a set of literals representing a disjunction of the literals. Free variables are interpreted to be universally quantified.

For a set of relational calculus A* formulae Φ, CLAUSIFY(Φ) converts Φ to a set of clauses that represent Φ. For example, $\Phi = \{\forall X(p(X, Y) \vee q(Y, a)), \forall Z(r(Z) \wedge p(b))\}$,
CLAUSIFY(Φ) = $\{\{p(X, Y), q(Y, a)\}, \{r(Z)\}, \{p(b)\}\}$

For a set of clauses R, and F a set of variables, DECLAUSIFY$_F(R)$ converts R to a relational calculus formula, leaving F as free variables. For example, $R = \{\{p(X, Y), q(Y, a)\}, \{r(Z)\}, \{p(b)\}\}$ and $F = \{Y\}$, DECLAUSIFY$_F(R) = \forall XZ((p(X, Y) \vee q(Y, a)) \wedge r(Z) \wedge p(b))$. As a shorthand, we will write simply DECLAUSIFY(R) for DECLAUSIFY$_{\{\}}(R)$.

For two clauses a and b, $a \amalg^{l_a, l_b} b$ is a resolvent of a and b on the literals l_a, l_b. More specifically,

$$a \amalg^{l_a, l_b} b =_{def} (a\sigma_a\sigma - l_a\sigma_a\sigma) \cup (b\sigma_b\sigma - l_b\sigma)$$

, where

- σ_a and σ_b are variable renamings such that $a\sigma_a$ and $b\sigma_b$ have no variable in common.
- l_a is a database literal in a,
- l_b is a database literal in b,
- $l_a\sigma_a$ and $\neg l_b\sigma_b$ are unifiable by most general unifier σ.

Sometimes we omit the specification of which literals to resolve on. In that case, we use $a \amalg b$ to represent any one of the valid resolvents. When we chain this operation without using parentheses to disambiguate order, we take the operator to be left associativity, e.g., $a_1 \amalg a_2 \amalg a_3 \amalg a_4$ = $(((a_1 \amalg a_2) \amalg a_3) \amalg a_4)$.

For a clause a, factor(a) gives a factor of a. That is, factor(a) $=_{def}$ $a\sigma$, where σ is a most general unifier of $l_1, l_2 \in a$.

RESCL[A] is the resolution and factoring closure of A. That is, RESCL[A] $=_{def}$ fixpoint$_{n\to\infty} ResCl_n[A]$, where $ResCl_n[A]$ is defined recursively as follows:

- $ResCl_0[A] = A$
- $ResCl_{n+1}[A] = \{a \amalg b : a, b \in ResCl_n[A]\} \cup \{factor(a) : a \in ResCl_n[A]\}$ [13]

Define the length of a clause c, denoted len(c), as the number of database literals in c. We also define the set of all clauses in a resolution closure, up to a length k. RESCL$_{\leq k}[A]$ $=_{def}$ $\{r \in$ RESCL$[A]$: len$(r) \leq k\}$.

5.2 The construction

For each tuple $\bar{t}$, we may use a resolution-refutation procedure to determine whether $Q(\bar{t})$ is consistent with C. Intuitively, we could treat the free variables in $Q(\bar{X})$ as special parameters that stand in for constants. Then we may use a resolution-refutation process once to compute a condition on $\bar{X}$ that splits the consistent instances of $Q(\bar{X})$ from the inconsistent instances (with respect to C).

We define a new binary resolution with parameter variables F:

Let A, B be clauses. Let F be the subset of variables which are to be treated as parameters.

For two clauses a and b, $a \amalg_F^{l_a,l_b} b$ is a resolvent of a and b on literals l_a, l_b, with respect to parameters F. More specifically, $a \amalg_F^{l_a,l_b} b =_{def} (a\sigma_a\sigma - l_a\sigma_a\sigma) \cup (b\sigma_b\sigma - l_b\sigma) \cup \{X \neq X\sigma : X \in F\}$, where

- σ_a and σ_b are variable renamings such that $a\sigma_a$ and $b\sigma_b$ have no variable in common except variables in F and that variables in F are not renamed.
- l_a is a database literal in a,
- l_b is a database literal in b,
- $l_a\sigma_a$ and $\neg l_b\sigma_b$ are unifiable by most general unifier σ such that $F\sigma \subseteq (F \cup \mathbf{dom})$.

As in regular binary resolution, we sometimes omit the specification of the literals on which the resolution is done.

Example 2 Let $F = \{X, Y\}$.
$\{\neg p(Z), \neg q(Z)\} \amalg_F^{\neg p(Z), p(X)} \{p(X)\} = \{\neg q(X)\}$
$\{\neg q(X)\} \amalg_F^{\neg q(X), q(Y)} \{q(Y)\} = \{X \neq Y\}$ ◇

Define the following restricted resolution closure:

Definition 6 (Unit RRC) *Let $\leq$ be a total order on all of the database literals (the specific ordering does not matter). Let A be a set of unit clauses. Let B be a set of clauses. Let F be the set of free variables in A.*
RRC$_F[A, B]$ =
$\{b \amalg_F^{l_1,l_{a_1}} a_1 \amalg_F^{l_2,l_{a_2}} a_2 \amalg_F^{l_3,l_{a_3}} \cdots \amalg_F^{l_k,l_{a_k}} a_k :$
$a_1, \ldots, a_k \in A$ and $b \in B$ and $k =$ len(b)
and $l_{a_i} \in a_i$ and $l_i = \min_{\leq}(b_i)\}$,
where $b_i = \{l \in (b \amalg_F^{l_1,l_{a_1}} a_1 \amalg_F^{l_2,l_{a_2}} \cdots \amalg_F^{l_{i-1},l_{a_{i-1}}} a_{i-1}) :$ l is a database literal$\}$.

Example 3 $A = \{\{\neg p(x)\}, \{\neg q(y)\}\}$, $B = \{\{\neg p(z), \neg q(z)\}\}$. Assume $\neg p(z) \leq \neg q(z)$.

$\{\neg p(z), \neg q(z)\} \amalg_F^{\neg p(z), p(x)} \{p(x)\} = \{\neg q(x)\}$
$\{\neg q(x)\} \amalg_F^{\neg q(x), q(y)} \{q(y)\} = \{x \neq y\}$

RRC$_F[A, B]$ = $\{\{x \neq y\}\}$ ◇

Lemma 3 *Let $c_1, c_2, \ldots, c_k$ be clauses. Let F be a set of variables. Let σ be a variable substitution, mapping F to constants. If $c_1\sigma \amalg c_2\sigma \amalg \cdots \amalg c_k\sigma$ exists, call it r, then* SIMPLIFY(r) = SIMPLIFY($(c_1 \amalg_F c_2 \amalg_F \cdots \amalg_F c_k)\sigma$), *where* SIMPLIFY($c$) = $c - \{l \in c : l$ is a ground (dis)equality atom which evaluates to false$\}$.

If $c_1\sigma \amalg c_2\sigma \amalg \cdots \amalg c_k\sigma$ does not exist, then $(c_1 \amalg_F c_2 \amalg_F \cdots \amalg_F c_k)\sigma$ either does not exist or contains a ground (dis)equality atom that evaluates to true.

PROOF. (i) Proof by induction on k:
Base case:

Assume $c_1\sigma \amalg c_2\sigma$ exists, using most general unifier θ. Since no $x \in F$ appears in $(c_1\sigma, c_2\sigma)$, we can assume without loss of generality that (VAR $- F)\theta \subseteq$ (VAR $- F) \cup$ **dom**. Let $c_1 = d_1 \cup \{l_1\}$, $c_2 = d_2 \cup \{l_2\}$, and $c_1\sigma \amalg c_2\sigma = d_1\sigma\theta \cup d_2\sigma\theta$, where θ is the most general unifier of $(l_1\sigma, \neg l_2\sigma)$.

Then a most general unifier of $(l_1, \neg l_2)$ is $\gamma\theta$ for a substitution γ such that $\gamma\sigma = \sigma$. For example, we can take γ to be a minimal subset of $\{x \mapsto y : x\sigma = y\sigma\}$ that is a unifier.

$c_1 \amalg c_2 = d_1\gamma\theta \cup d_2\gamma\theta$.

$c_1 \amalg_F c_2 = (d_1\gamma\theta \cup d_2\gamma\theta) \cup \{x \neq x\gamma\theta : x \in F\}$. Given $x \in F$, $x\gamma\theta\sigma = x\gamma\sigma$ because $F\gamma \subseteq F \cup$ **dom** and θ is the identity mapping on $F \cup$ **dom**. Furthermore, $x\gamma\sigma = x\sigma$ because $\gamma\sigma = \sigma$.On $F \cup$ **dom**, $\gamma\theta\sigma = \gamma\sigma = \sigma$, so $x\sigma = x\gamma\theta\sigma$. Hence, SIMPLIFY$(\{x \neq x\gamma\theta : x \in F\}\sigma) = \{\}$.

Given $x \in F$, $x\gamma\theta\sigma = x\gamma\sigma$ because $F\gamma \subseteq F \cup$**dom** and θ is the identity mapping on $F \cup$ **dom**. Furthermore, $x\gamma\sigma = x\sigma$ because $\gamma\sigma = \sigma$. Finally, $x\sigma = x\sigma\theta$ because $(F \cup$ **dom**$)\sigma \subseteq$ **dom**, and θ is the identity map on **dom**. Given, $x \in$ (VAR $- F$), $x\gamma\theta\sigma = x\theta\sigma$ because γ is the identity map on VAR $- F$. Furthermore, $x\theta\sigma = x\theta = x\sigma\theta$ because (VAR $- F)\theta \subseteq$ (VAR $- F) \cup$ **dom** and σ is the identity map on (VAR $- F) \cup$ **dom**. We conclude that $\gamma\theta\sigma = \sigma\theta$.

We concludes that SIMPLIFY$((c_1 \amalg_F c_2)\sigma)$ = SIMPLIFY$((d_1\gamma\theta \cup d_2\gamma\theta)\sigma) \cup \{\}$ = SIMPLIFY$(d_1\sigma\theta \cup d_2\sigma\theta)$ = SIMPLIFY$((c_1\sigma \amalg c_2\sigma))$.

Induction hypothesis: If $c_1\sigma \amalg c_2\sigma \amalg \cdots \amalg c_k\sigma$ exists, call it r, then SIMPLIFY(r) =
SIMPLIFY$((c_1 \amalg_F c_2 \amalg_F \cdots \amalg_F c_k)\sigma))$

Let $r = c_1\sigma \amalg c_2\sigma \amalg \cdots \amalg c_k\sigma$. Let $s = (c_1 \amalg_F c_2 \amalg_F \cdots \amalg_F c_k)$

Induction step: We show that if $r \amalg c_{k+1}\sigma$ exists , then SIMPLIFY($r \amalg c_{k+1}\sigma$) =
SIMPLIFY$((s \amalg_F c_{k+1})\sigma)$

Assume $r \amalg c_{k+1}\sigma$ exists, using most general unifier θ. Let $r = d_1 \cup \{l_1\}$, $c_{k+1} = d_2 \cup \{l_2\}$, and $r \amalg c_{k+1}\sigma = d_1\theta \cup d_2\sigma\theta$, where θ is the most general unifier of $(l_1, \neg l_2\sigma)$.

Let $s = d_3 \cup \{l_3\}$, where SIMPLIFY$(d_3\sigma)$ = SIMPLIFY(d_1) and SIMPLIFY$(l_3\sigma)$ = SIMPLIFY(l_1).

Applying the same analysis as in (i) base case, using s in place of c_1 and c_{k+1} in place of c_2, we conclude that SIMPLIFY$(r \amalg c_{k+1}\sigma)$

[13] identifying different variable renaming of the same clause.

=
SIMPLIFY$((s \amalg_F c_{k+1})\sigma)$.

(ii) Proof by induction on k:

Base case: Let l_1, l_2 be literals and d_1, d_2 these sets of literals such that $c_1 = d_1 \cup \{l_1\}$ and $c_2 = d_2 \cup \{l_2\}$.

Assume $c_1\sigma \amalg c_2\sigma$, resolving on $(l_1\sigma, l_2\sigma)$, does not exist. That is, $(l_1\sigma, \neg l_2\sigma)$ cannot be unified. Assumed that $c_1 \amalg_F c_2$ exists, that is, $(l_1, \neg l_2)$ is unifiable by a most general unifier γ.

Suppose for the sake of contradiction that for all $x \in F$, $x\sigma = x\gamma\sigma$. Define a mapping $\theta = \{x \mapsto x\gamma\sigma : x \in \text{VAR} - F\} \cup \{x \mapsto x : x \in F \cup \textbf{dom}\}$ For any $x, y \in F$, if $x\gamma = y\gamma$, then $x\gamma\sigma = y\gamma\sigma$, then $x\sigma = y\sigma$, then $x\sigma\theta = y\sigma\theta$. For $x, y \in \text{VAR} - F$, if $x\gamma = y\gamma$, then $x\sigma\gamma = y\sigma\gamma$, then $x\sigma\gamma\sigma = y\sigma\gamma\sigma$, then $x\sigma\theta = y\sigma\theta$. For $x \in F, y \in \text{VAR} - F$, if $x\gamma = y\gamma$, then $x\gamma = x\sigma\gamma = y\gamma$, then $x\sigma\gamma\sigma = y\gamma\sigma = y\sigma\gamma\sigma$, then $x\sigma\theta = y\sigma\theta$. For $x, y \in \textbf{dom}$, then $x\gamma = y\gamma$ implies $x = y$, hence $x\sigma\theta = y\sigma\theta$. For $x \in F, y \in \textbf{dom}$, $x\gamma = y\gamma$ implies $x\gamma\sigma = y\gamma\sigma$ implies $x\sigma = y$ implies $x\sigma = y\sigma$ implies $x\sigma\theta = y\sigma\theta$. For $x \in \text{VAR} - F, y \in \textbf{dom}$, $x\gamma = y\gamma$ implies $x\sigma\gamma = y\sigma\gamma$ (because σ is identity on all but F) implies $x\sigma\gamma\sigma = y\sigma\gamma\sigma = y\sigma = y\sigma\theta$ implies $x\sigma\theta = y\sigma\theta$. To summarize, for any $x, y \in \text{VAR} \cup \textbf{dom}$, if $x\gamma = y\gamma$ then $x\sigma\theta = y\sigma\theta$. That is, θ unifies $(l_1\sigma, \neg l_2\sigma)$, a contradiction.

We conclude that for some $x \in F$, $x\sigma = x\gamma\sigma$. Hence, $c_1 \amalg_F c_2$ contains a ground (dis)equality atom that evaluates to true.

Induction hypothesis: for all $i < k$, if $c_1\sigma \amalg c_2\sigma \amalg \cdots \amalg c_i\sigma$ does not exist, then $(c_1 \amalg_F c_2 \amalg_F \cdots \amalg_F c_i)\sigma$ either does not exist or contains a ground (dis)equality atom that evaluates to true.

Induction step: show that if $c_1\sigma \amalg c_2\sigma \amalg \cdots \amalg c_k\sigma$ does not exist, then $(c_1 \amalg_F c_2 \amalg_F \cdots \amalg_F c_k)\sigma$ either does not exist or contains a ground (dis)equality atom that evaluates to true.

If for some $i < k$, $c_1\sigma \amalg c_2\sigma \amalg \cdots \amalg c_i\sigma$ does not exist, then, by the induction hypothesis, $(c_1 \amalg_F c_2 \amalg_F \cdots \amalg_F c_i)\sigma$ either does not exist, in which case $(c_1 \amalg_F c_2 \amalg_F \cdots \amalg_F c_k)\sigma$ also fails to exist, or contains a ground (dis)equality atom that evaluates to true, in which case $(c_1 \amalg_F c_2 \amalg_F \cdots \amalg_F c_k)\sigma$ contains the same atom. So the induction step holds.

In the remaining case, $c_1\sigma \amalg c_2\sigma \amalg \cdots \amalg c_k\sigma$ exists, but $c_1\sigma \amalg c_2\sigma \amalg \cdots \amalg c_{k+1}\sigma$ does not exist.

Let $r = c_1\sigma \amalg c_2\sigma \amalg \cdots \amalg c_k\sigma$. Let $s = (c_1 \amalg_F c_2 \amalg_F \cdots \amalg_F c_k)\sigma$.

Applying part (i), SIMPLIFY(r) = SIMPLIFY$(s\sigma)$. Applying the same analysis as in (ii) base case, with s in place of c_1 and c_{k+1} in place of c_2, we conclude that if $s \amalg_F c_{k+1}$ exists, $s \amalg_F c_{k+1}$ contains a ground (dis)equality atom that evaluates to true.

□

The next lemma follow from Lee's classic subsumption theorem [36].

Lemma 4 *Given a finite set of ground literals L such that L is satisfiable. Given a set of clauses Ω. If Ω, L is unsatisfiable, then there exists $\omega \in$ RESCL$[\Omega]$ such that ω, L is unsatisfiable.*

Lemma 5 *Let $Q(\bar{X})$ be a quantifier-free CQ$\neg$ query with free variables $\bar{X}$. Let C be a set of constraints.*
DECLAUSIFY$_{\bar{X}}$[RRC$_{\bar{X}}$[CLAUSIFY(Q), RESCL[CLAUSIFY$[C]$]] *is a consistency condition of Q with respect to C.*

PROOF. Observe that $\bar{X}$ are the only free variables in (RRC$_{\bar{X}}$[CLAUSIFY(Q), RESCL[CLAUSIFY[C]]).

Let $\Xi^Q_C(\bar{X})$ = DECLAUSIFY$_{\bar{X}}$[RRC$_{\bar{X}}$[CLAUSIFY(Q), RESCL[CLAUSIFY$[C]$]].

Let $\bar{t}$ be a tuple.

($\Leftarrow$) Assume that $Q(\bar{t}), C$ is unsatisfiable and $Q(\bar{t})$ is satisfiable. Let σ be the variable substitution that maps each variable in $\bar{X}$ to the corresponding constant in $\bar{t}$. By Lemma 4, there exists $r \in$ RESCL[CLAUSIFY$[C]$] such that $r, Q(\bar{t})$ is unsatisfiable. There exist $q_1, q_2, \ldots, q_k \in$ CLAUSIFY$[Q(\bar{X})]$ such that $\{\}$ = $r \amalg q_1\sigma \amalg q_2\sigma \amalg \cdots \amalg q_k\sigma$, obeying the ordering $\leq$ on r[14]. Let $s = r \amalg_F q_1 \amalg_F q_2 \amalg_F \cdots \amalg_F q_k$. By Lemma 3, SIMPLIFY$(s\sigma)$ = $\{\}$. That is, $s\sigma$ evaluates to false. $s \in$ RRC$_{\bar{X}}$[CLAUSIFY(Q), RESCL[CLAUSIFY[C]]). Hence, $\Xi^Q_C(\bar{t})$ evaluates to false.

($\Rightarrow$) Assume that $\Xi^Q_C(\bar{t})$ evaluates to false. Let σ be the variable substitution that maps each variable in $\bar{X}$ to the corresponding constant in $\bar{t}$. There exists a clause $s \in$ RRC$_{\bar{X}}$[CLAUSIFY(Q), RESCL[CLAUSIFY[C]]) such that $s\sigma$ evaluates to false. That is, there exists $r \in$ RESCL[CLAUSIFY[C]] and $q_1, q_2, \ldots, q_k \in$ CLAUSIFY$[Q(\bar{X})]$ such that $s = r \amalg_F q_1 \amalg_F q_2 \amalg_F \cdots \amalg_F q_k$ and SIMPLIFY$(s\sigma)$ = $\{\}$.

By Lemma 3, $\{\}$ = $r \amalg q_1\sigma \amalg q_2\sigma \amalg \cdots \amalg q_k\sigma$. We conclude that $Q(\bar{t}), C$ is unsatisfiable. □

Example 4 Let $Q = p(X) \wedge q(Y), C = \{\forall Z(\neg p(Z) \vee \neg q(Z))\}$
The set of free variables of Q is $F = \{X, Y\}$.

Let A = CLAUSIFY$(Q) = \{\{\neg p(X)\}, \{\neg q(Y)\}\}$
Let B = RESCL[CLAUSIFY$(C)] = \{\{\neg p(Z), \neg q(Z)\}\}$

RRC$_F[A, B]$ = $\{\{Y \neq X\}, \{X \neq Y\}\}$ (see example 3)
DECLAUSIFY$_F$(RRC$_F[A, B]$) = $Y \neq X \wedge X \neq Y$ is a consistency condition of Q with respect to C. ◇

The above lemma gives an effective construction of a consistency condition when RESCL[CLAUSIFY[C]] is finite.

We make another observation to obtain an effective construction of a consistency condition even when RESCL[CLAUSIFY[C]] is infinite.

Lemma 6 *Given a tuple $\bar{t}$ and a CQ$\neg$ query $Q(\bar{X})$. Let* len$(Q(\bar{X}))$ *denote the number of database literals in $Q(\bar{X})$. If $Q(\bar{t}), c$ is unsatisfiable and $Q(\bar{t})$ is satisfiable, then there is a factor c_f of c such that* len$(c_f) \leq$ len$(Q(\bar{X}))$.

Lemma 7 *Let $Q(\bar{X})$ be a quantifier-free CQ$\neg$ query with free variables $\bar{X}$. Let the length of Q be k. Let C be a set of constraints.*
DECLAUSIFY*(*RRC*[*CLAUSIFY*(Q),*
RESCL$_{\leq k}$*[*CLAUSIFY*[C]]) is a consistency condition of Q with respect to C.*

Since RESCL$_{\leq k}$[CLAUSIFY[C]] is always finite and can be effectively computed[15], the foregoing lemma gives an effective construction of a consistency condition even when RESCL[CLAUSIFY[C]] is infinite.

[14] By the completeness of semi-ordered resolution [45].

[15] Let d be the finite subset of **dom** that occurs in C. Initially disregarding the equality literals, one can generate the finite set of all possible clauses over S and d, with the number of database literals that most k (up to variable renaming). Then for each of these equality-free clauses c, each pair of variables may be equated, disequated, or neither. So one can generate the finite set of all possible clauses whose database portion is c. Finally, each of the finitely many candidate clauses so generated can be tested for whether it follows from C under finite Herbrand logic.

5.3 Rewriting time and size

In this section, we analyze and discuss the running time of the rewriting procedure and the size of the output query.

First, we observe that the rewriting procedure appends to the input query a consistency condition consisting of (dis)equalities. The parts of the input query that refer to database relations are totally unchanged. Therefore, the asymptotic data complexity of the output query is no greater than that of the input query.[16]

Still, we will analyze and discuss the size the appended consistency condition. Let $Q(\bar{X})$ be a CQ$\neg$ query with free variables$\bar{X}$. Let $Q(\bar{X}) = \exists \bar{Y}(q_1 \wedge \cdots \wedge q_n)$, where each q_i is a literal. Let n_a be the greatest arity among the relations in Q. Let $Q^{cl} =$ CLAUSIFY$[Q]$.

Let C be a set of constraints. Let $R =$ RESCL$_{\leq n}$[CLAUSIFY$[C]$]. Let $k = |R|$. Let l be the length of the longest clause in R.

The consistency condition $\Xi^Q_C(\bar{X})$, computed as RRC$_{\bar{X}}[Q^{cl}, R]$, consists of at most kn^l (dis)equality clauses, each of which contains at most n_a (dis)equalities. For example, for C a set of k functional dependencies, the consistency condition consists of at most $n_a k n^2$ (dis)equality.

A consistency condition $\Xi^Q_C(\bar{X})$ can be found with at most kln^l unification attempts (see section 5). Each unification attempt takes time linear in the arity of the input literals [39]. So, given R, the total worst-case running time is $O(kln_a n^l)$. For example, for C a set of k functional dependencies, the rewriting takes at most $O(n_a k n^2)$ time.

All E* queries can be written as a union of CQ$\neg$ queries. These queries are rewritten one CQ$\neg$ queries at a time, so its straightforward to extend the analysis in this section to E* queries.

6. REWRITING A* QUERIES

The technique for rewriting E* queries, as presented in section 4, does not extend to deal with universal quantified queries because the technique is based on treating each instance of the quantifier-free part separately. With universal quantification, the technique fails because multiple instances of the quantifier-free part may interact to produce an inconsistency.

Example 5 $Q = \forall X \neg p(X)$. $C = \{p(a) \vee p(b)\}$. No single instance of $p(X)$ is inconsistent with C, but $\neg p(a)$ and $\neg p(b)$ are together inconsistent with C. $\diamond$

In this section, we obtain a rewriting procedure for A* queries by using a restricted form of resolution between the query and the constraints to derive all additional conditions a database must satisfy to answer affirmatively to an A* query.

First, we give an example to suggest the rewriting procedure.

Example 6 Query:
$Q = \forall$ S ($\neg$p(S) $\vee \neg$q(S)). Clausal form: $\{\neg p(S), \neg q(S)\}$.
Constraints:
$C = \{\forall S(q(S) \vee S = a)\}$. Clausal form: $\{q(S), S = a\}$.

We can use C to strengthen Q, finding what must also be entailed if Q is entailed and C is not violated. Intuitively, if for all S $\neg$p(S) or $\neg$q(S), but $q(S)$ holds unless $S = a$, then we had better have $p(S)$ unless $S = a$. Resolution gives us exactly what we want in this case: $\{\neg p(S), \neg q(S)\}$ II $\{q(S), S = a\} = \{\neg p(S), S = a\}$, ie., $\forall S(p(S) \vee S = a)$. $\diamond$

To define the rewriting procedure, we first extend the definition of restricted resolution closure (as defined earlier in section 5.2) to non-unit clauses in the first argument. Define RRC$[A, B]$ recursively as the following:

[16]Assuming straightforward evaluation.

Definition 7 (RRC) *Let $\leq$ be a total order on all of the database literals (the specific ordering does not matter). Let A be a set of clauses. Let B be a set of clauses.*
RRC$[A, B]$ =
$\{b \amalg^{l_1,l_{a_1}} a_1 \amalg^{l_2,l_{a_2}} a_2 \amalg^{l_3,l_{a_3}} \cdots \amalg^{l_k,l_{a_k}} a_k :$
$a_1, \ldots, a_k \in A$ and $b \in B$ and $k =$ len(b) and $l_{a_i} \in a_i$ and
$l_i = \min_{\leq}(b_i)\}$, *where*
$b_i = \{l \in b \amalg^{l_1,l_{a_1}} a_1 \amalg^{l_2,l_{a_2}} \cdots \amalg^{l_{i-1},l_{a_{i-1}}} a_{i-1} :$
l is a database literal that descends from $b\}$.

For a closed A* query Q and a set of A* constraints C, we define a rewriting of Q with respect to C as follows:
Rewrite$[Q, C] =_{def}$
DECLAUSIFY[RRC[CLAUSIFY[Q], RESCL[CLAUSIFY[C]]]].

Example 7 Query:
$Q = \forall x(p(x, a) \rightarrow q(x, b))$.
$F = \{b\}$.
Constraints:
$C = \{\forall x_1 x_2 x_3(q(x_1, b) \wedge q(x_2, b) \wedge q(x_3, b) \rightarrow (x_1 = x_2 \vee x_2 = x_3 \vee x_1 = x_3))\}$
We rewrite Q with respect to C as follows:
CLAUSIFY$[Q] = \{(q_1 =)\{\neg p(x, a), q(x, b)\}\}$.
CLAUSIFY$[C] =$
$\{(c_1 =)\{\neg q(x_1, b), \neg q(x_2, b), \neg q(x_3, b), x_1 = x_2, x_2 = x_3, x_1 = x_3\}\}$. Assume $\neg q(x_1, b) \leq \neg q(x_2, b) \leq \neg q(x_3, b)$.
$c_1 \amalg^{\neg q(x_1,b),q(x,b)} q_1 =$
$\{\neg p(x_1, a), \neg q(x_2, b), \neg q(x_3, b), x_1 = x_2, x_2 = x_3, x_1 = x_3\}$
$c_1 \amalg q_1 \amalg^{\neg q(x_2,b),q(x,b)} q_1 =$
$\{\neg p(x_1, a), \neg p(x_2, a), \neg q(x_3, b)$
$, x_1 = x_2, x_2 = x_3, x_1 = x_3\}$
$c_1 \amalg q_1 \amalg q_1 \amalg^{\neg q(x_3,b),q(x,b)} q_1 =$
$\{\neg p(x_1, a), \neg p(x_2, a), \neg p(x_3, a)$
$, x_1 = x_2, x_2 = x_3, x_1 = x_3\}$
RRC[CLAUSIFY$[Q]$, CLAUSIFY$[C]] =$
$\{q_1, c_1 \amalg^{\neg q(x_1,b),q(x,b)} q_1 \amalg^{\neg q(x_2,b),q(x,b)} q_1 \amalg^{\neg q(x_3,b),q(x,b)} q_1\}$
Finally, Rewrite$[Q, C] =_{def}$
DECLAUSIFY(RRC[CLAUSIFY$[Q]$, CLAUSIFY$[C]]) =$
$\forall x(p(x, a) \rightarrow q(x, b)) \wedge$
$\forall x_1 x_2 x_3$
$(p(x_3, a) \wedge p(x_3, a) \wedge p(x_3, a) \rightarrow (x_1 = x_2 \vee x_2 = x_3 \vee x_1 = x_3))$
$\diamond$

The termination of RRC$[A, B]$ is clear for finite sets of sentences A, B. We conclude that the rewriting procedure terminates on finitely-closed constraints C (ie., RESCL[CLAUSIFY$[C]$] is finite).

Lemma 8 *Let A, B each be a finite set of function-free clauses. Then* RRC$[A, B]$ *terminates.*

In the next theorem, we show the correctness of the rewriting.

Theorem 3 *Let Q be a closed relational calculus query in the A^* fragment. Let C be a set of constraints in the A^* fragment of relational calculus. Then $D \models_C Q$ if and only if $D \models Rewrite[Q, C]$.*

PROOF. Let $C_R =$ RESCL[CLAUSIFY$[C]$].
Let $R =$ RRC[CLAUSIFY$[Q], C_R]$.
We prove the lemma that $D \models_C Q$ if and only if $D \models r$ for every $r \in R$. The theorem follows immediately.
Notation: Let l be a literal. $\bar{l}$ denotes the complement of l. Let c be a clause (a set of literals). $\bar{c}$ denotes the set $\{\bar{l} : l \in c\}$.
($\Leftarrow$) Given $D \models R$. CLAUSIFY$[Q] \subseteq R$, so $D \models Q$.

Let M be a minimal subset of EXPLICIT(D) that implies Q. M implies every clause in $Q^{cl} = \text{CLAUSIFY}(Q)$. By minimality, for every $b \in M$, there is an instantiation q^i of $q \in Q^{cl}$ such that $(q^i - b) \cap M$ is empty.

Let f be a function that maps each $b \in M$ to one such q^i.

Suppose M is inconsistent with C. There is a $d \in C_R$ with an instance d^i is such that $\bar{d}^i \subset M$. Take the derivation of the empty clause from d and $\bar{d}^i$[17] but replace each $b \in \bar{d}^i$ by $f(b)$. We derive a ground clause $c = \bigcup_{b \in \bar{d}^i} [f(b) - b]$. By construction of f, $c \cap M$ is empty. c is an instance of a clause in R, so by assumption, M implies c. $M \cap c$ cannot be empty. This is a contradiction.

We conclude that M is consistent with C. $M \models Q$ by the way M is chosen. From the definition, we conclude $D \models_C Q$.

($\Rightarrow$)

Given $D \models_C Q$.

Let D^* be a subset of EXPLICIT(D) such that $D^* \models^{\text{I}} Q$ and D^* is consistent with C.

Suppose for the sake of contradiction that $D \not\models R$. Then there is a clause $r \in R$ such that there is an instantiation r^i of r such that r^i is disjoint from EXPLICIT(D). Consider a derivation tree of r^i from ground instances $q_1, q_2, \ldots, q_k$ of a clause $q \in \text{CLAUSIFY}[Q]$ and C_R. By the assumption that $D^* \models^{\text{I}} Q$, each q_i has nonempty intersection with D^*.

For each q_i, let f be a function that maps each q_i to a ground literal in $q_i \cap D^*$. Let $Q_i^* = \{q_1, q_2, \ldots, q_k\}$ be the collection of q_i's used in the derivation of r^i. In the derivation tree of r^i, if there is a $c \in C_R$ such that every resolution step on c is against a literal $f(q_i)$, then c contradicts $\{f(q_i) : q_i \text{ used in derivation}\} \subseteq D^*$. This is a contradiction against the assumption that D^* is consistent with C.

Otherwise, there exists $f(q_i)$ that is in r^i, contradicting the assumption that r^i is disjoint from EXPLICIT(D). We conclude $D \not\models R$. □

6.1 Rewriting time and size

In this section, we analyze and discuss the running time of the rewriting procedure and the size of the output query.

Let $Q(\bar{X})$ be a closed A* query $\forall \bar{X}(Q_1 \wedge \cdots \wedge Q_n)$, where each Q_i is a disjunction of literals. Let m be the length of the largest Q_i. Let n_a be the greatest arity among the relations in Q.

Let C be a set of constraints. Let $R = \text{RESCL}[\text{CLAUSIFY}[C]]$. Let $k = |R|$. Let l be the length of the longest clause in R.

The rewriting Rewrite$[Q, C]$, computed as RRC$[Q^{cl}, R]$, adds to Q at most $k(mn)^l$ independent conjuncts, each of which is a disjunction of at most ml literals. For example, for C a set of k functional dependencies, Rewrite$[Q, C] = Q \wedge Q_1^r \wedge \cdots \wedge Q_h^r$, where h is at most $k(mn)^2$, each Q_i^r of length at most $2m$.

The rewriting Rewrite$[Q, C]$ can be found with at most $kl(mn)^l$ unification attempts. Each unification attempt takes time linear in the arity of the input literals [39]. So, given R, the total worst-case running time is $O(kln_a(mn)^l)$. For example, for C a set of k functional dependencies, the rewriting takes at most $O(n_a k(mn)^2)$ time.

7. REWRITING E*A* QUERIES

As a springboard to rewriting E*A* queries,we first treat the rewriting of A* queries with free variables.

We treat free variables as parameters and the modified binary resolution rule (as defined in section 5.2) to deal with them.

Define RRC$_F[A, B]$ the same way as RRC$[A, B]$ (definition 7) except that the $\amalg_F$ operation is used in place of the $\amalg$ operation.

[17]The derivation can be chosen to obey the ordering $\leq$ on d, by the completeness of semi-ordered resolution [45].

For an open A* query Q, with set of free variables $\mathcal{F}$, and a set of A* constraints C, we extend the rewriting of closed queries as follows:

Rewrite$[Q, C] =_{def}$

DECLAUSIFY$_F$[RRC$_F$[CLAUSIFY[Q], RESCL[CLAUSIFY[C]]]].

Example 8 Query:

$Q = \forall x(p(x, a) \rightarrow q(x, y))$.

$F = \{y\}$.

Constraints:

$C = \{\forall x_1 x_2 x_3(q(x_1, b) \wedge q(x_2, b) \wedge q(x_3, b) \rightarrow (x_1 = x_2 \vee x_2 = x_3 \vee x_1 = x_3))\}$

We rewrite Q with respect to C as follows:

CLAUSIFY$[Q] = \{(q_1 =)\{\neg p(x, a), q(x, y)\}\}$.

CLAUSIFY$[C] =$

$\{(c_1 =)\{\neg q(x_1, b), \neg q(x_2, b), \neg q(x_3, b), x_1 = x_2, x_2 = x_3, x_1 = x_3\}\}$. Assume $\neg q(x_1, b) \leq \neg q(x_2, b) \leq \neg q(x_3, b)$.

$c_1 \amalg_F^{\neg q(x_1,b), q(x,y)} q_1 =$

$\{\neg p(x_1, a), \neg q(x_2, b), \neg q(x_3, b), x_1 = x_2, x_2 = x_3, x_1 = x_3, y \neq b\}$

$c_1 \amalg_F q_1 \amalg_F^{\neg q(x_2,b), q(x,y)} q_1 =$

$\{\neg p(x_1, a), \neg p(x_2, a), \neg q(x_3, b)$

$, x_1 = x_2, x_2 = x_3, x_1 = x_3, y \neq b\}$

$c_1 \amalg_F q_1 \amalg_F q_1 \amalg_F^{\neg q(x_3,b), q(x,y)} q_1 =$

$\{\neg p(x_1, a), \neg p(x_2, a), \neg p(x_3, a)$

$, x_1 = x_2, x_2 = x_3, x_1 = x_3, y \neq b\}$

RRC$_F$[CLAUSIFY[Q], CLAUSIFY[C]] =

$\{q_1, c_1 \amalg_F^{\neg q(x_1,b), q(x,y)} q_1 \amalg_F^{\neg q(x_2,b), q(x,y)} q_1 \amalg_F^{\neg q(x_3,b), q(x,y)} q_1\}$

Finally, Rewrite$[Q, C] =_{def}$

DECLAUSIFY$_F$(RRC$_F$[CLAUSIFY[Q], CLAUSIFY[C]]) =

$\forall x(p(x, a) \rightarrow q(x, y)) \wedge$

$\forall x_1 x_2 x_3$

$(p(x_3, a) \wedge p(x_3, a) \wedge p(x_3, a) \rightarrow (x_1 = x_2 \vee x_2 = x_3 \vee x_1 = x_3 \vee y \neq b))$ ◇

Theorem 4 *Let Q be an open relational calculus query, with free variables F, in the A^* fragment. Let C be a set of constraints in the A^* fragment of relational calculus. Then $D \models_C Q$ if and only if $D \models Rewrite_F[Q, C]$.*

PROOF. Follows from theorem 3 and lemma 3. □

Theorem 5 *Let Q be a relational calculus query, with existentially quantified variables $\bar{x}$, in the E^*A^* fragment. Let C be a set of constraints in the A^* fragment of relational calculus. Let $Q = \exists \bar{x} Q'$ where Q' is in the A^* fragment. Then $D \models_C Q$ if and only if $D \models \exists \bar{x} Rewrite_{\bar{x}}[Q', C]$.*

PROOF. Follows from theorem 4 and lemma 2. □

The running time analysis for rewriting A* queries (section 6.1) still applies. The output size is as in the analysis for rewriting A* queries, except that each conjunct Q_i^r may contain up to $|\bar{x}|$ additional equality literals.

8. GROUND CONSTRAINTS

In the special case of ground constraints (closed, quantifier-free CALC sentences), the number of maximally strong consistent relaxations is bounded by the constraint set, independently of the input database instance. We give a polynomial data complexity algorithm that constructs and queries each maximal consistent relaxation to find the reasonable query answers to any CALC query.

Let B be a database instance. Let C be a set of ground constraints. C may be written in disjunctive normal form as a sentence

$\phi = \phi_1 \vee \cdots \vee \phi_k$, where each ϕ_i is a conjunction of ground literals. Let m be the length of the longest ϕ_i.

For each conjunction ϕ_j, construct a possible, incomplete database B_j that is same as B except that it is agnostic regarding each conjunct l that is in ϕ_j and contradicts B. To state it precisely, first we define for a database instance B and a finite set of ground literals L, the operation of relaxing B by L: $B -^* L = \text{EXPLICIT}(B) - L$. Let $B_j = B -^* L_j$, where $L_j = \{\neg l : l \text{ is a literal in } \phi_j\}$.

Finally, $\bar{t}$ is an RQA answers to any CALC query Q if and only if for some B_j, $B_j \models^{\text{I}} Q(\bar{t})$. That $B_j \models^{\text{I}} Q(\bar{t})$ can be tested in polynomial time[18], say at most $p(|B|)$. Therefore, we can test $B_j \models^{\text{I}} Q(\bar{t})$ for each j in polynomial time as a function of $|B|$. Hence we have the following theorem.

Theorem 6 *Let Q be a relational calculus query. Let C be a set of ground relational calculus constraints. The decision problem $\{(B, \bar{t}) : B \models_C Q(\bar{t})\}$ is in PTIME (data complexity), where B ranges over database instances and $\bar{t}$ ranges over tuples.*

9. RELATED WORK

The idea of reasoning from consistent subsets of inconsistent information is well known [38, 20]. The notion of reasonable query answers is a variation on existential Ω-entailment [33].

There is a large body of work on producing consistent query answers from inconsistent databases [12, 3, 25, 6, 14, 16, 17, 4, 5, 10, 30][19], including some that use a query rewriting approach. However, that body of work is concerned with producing answers according to a notion that is different from that considered in this paper (see section 9.1).

Shekhar Pradhan investigated using annotated logic programs to give databases argumentation capacity in answering queries [42, 44, 43]. The semantics proposed turn out to be similar to CQA. The body of work does not give a query rewriting procedure which rewrites the query into a standard database query language (e.g., relational algebra, relational calculus, Datalog, SQL).

Data exchange [22, 23, 24, 26, 27, 9, 2, 40, 34] and data integration with target constraints [28, 13, 19] also study the answering queries while accounting for constraints. One major difference between these two bodies of work and the answering of queries from inconsistent databases is that Data exchange is primarily concerned with under-constrained systems: given a source instance, a set of source-to-target dependencies, and a set of target dependencies, find a "best" target instance among many that satisfy the dependencies. In contrast, answering queries from inconsistent databases is primarily concerned with over-constrained systems: given a set of constraints and an instance which violates the constraints, produce reasonable query answers. A straightforward application of the data exchange framework to such a setting would simply find that there are no adequate target instances and no meaningful way to derive query answers.

Consistency and integrity constraints for XML has also been studied[35, 7, 8].

Various fuzzy database systems [41, 15, 11] are proposed to deal with incomplete, imprecise and uncertain information. The work presented in this paper is distinguished from fuzzy database systems in several ways: 1) it is intended to deal with overconstrained systems, 2) it does not require extending the standard relational model with tuple or attribute annotations, and 3) it gives a light weight solution which requires no modification to the existing infrastructure.

The idea of rewriting a query by appending a consistency condition as defined in definition 5 appeared in the authors' recent work in inconsistency-tolerant reasoning [29]. However, the paper addresses a different answer semantics and applies to a much smaller class of constraints.

A preliminary and restricted version of this rewriting algorithm for E* queries appeared in a Ph.D. project proposal at a Ph.D. workshop [31]. This paper greatly expands the class of queries and constraints handled and reports new complexity results.

9.1 Comparison with CQA and Possible Answers

A natural way to model the uncertainty of having inconsistent data is to view an inconsistent database instance as a collection of possible, consistent databases.

To obtain a possible database, one might choose to repair the original database instance (adding and removing information) or to relax the original database instance (only removing information).

For example, consider the case that each student has at least one advising professor, but a particular student has none listed. Then each possible minimal-change repair guesses a particular professor to be an advising professor to the student (adding information). A relaxation, on the other hand, simply becomes agnostic regarding whether any given professor is an advising professor to the student.[20]

After modeling and inconsistent database instance as a collection of possible, consistent databases, one may decide to admit the certain answers (admit answers that arise in all the possible databases) or to admit the possible answers (admit answers that arise in some possible database).

Consistent Query Answers [12, 3, 6] (CQA, a well known inconsistency-tolerant query semantics) gives the certain answers from the possible repairs.

Another query semantics that has been discussed in literature gives possible answers from the possible repairs. It's been referred to simply as "possible answers". In this discussion, we will call it possible answers from repairs (PAR).

The Reasonable Query Answers (RQA) gives possible answers from the possible relaxations.

We summarize the relationship between these three answer semantics in the following table:

	possible answers	certain answers
relaxations	**RQA**	
repairs	PAR	CQA

In many situations, such as looking at an integrated database to identify candidates, one would like to include uncertain answers. Recall the scenario and accompanying data (table 1) given in the introduction. Consider the answer `David` to the query `ANS(X) :- RecessionBorn (X), OfAge(X).` The answer is consistently derived from the tuple `BirthYear(David, 1980).`

[18]For example, $B -^* L \models^{\text{I}} Q(\bar{t})$ can be tested by running a straightforward top-down nonrecursive Datalog evaluation of the $B \models Q(\bar{t})$, intervening to discard any branch that uses a literal in L.

[19]in some cases, the authors do not refer to the notion of answers considered as consistent query answers. Nevertheless, the notions generally fit the description of certain answers over a collection of possible repairs. [14] considers incompleteness as well as inconsistencies.

[20]One may ask about notions of repairs that allow only tuple deletion? This notion has the problem that there may be no tuple-deletion repairs that satisfy a constraint. Furthermore, the deletion of a tuple to obtain a repair is also an addition of information — the implicit information that a deleted fact is false.

But the data also asserts another birth year,1984 for David. If we chose `BirthYear(David, 1984)` and discarded `BirthYear(David, 1980)`, then `David` no longer satisfies the query. So the answer `David` has consistent support but is not guaranteed. In our scenario, there is little harm in sending a survey to David because the survey would include a section that verifies the real birth year satisfies the criteria set out by the study. Restricting to certain answers forces the researcher to pass over reasonable but uncertain candidates like David, forfeiting potentially valuable responses.

Adopting PAR, in the other hand, leads to answers that come from information added to the original data through guessing. Recall the example that a student has no advising professor listed. PAR would guess every professor as a possible answer as an advising professor to the student.

RQA allows uncertain answers by admitting the possible answers but avoids additional guessed information by considering relaxations rather than repairs of the original database instance.

10. CONCLUSIONS FUTURE WORK

In this paper, we define reasonable query answers and rewriting procedures to compute these answers. Here we discuss an interesting extension.

Recall from the introductory scenario the two conflicting tuples `BirthYear(David, 1980)` and `BirthYear(David, 1984)`. If we chose to believe `BirthYear(David, 1980)`, then `David` is an answer. But if we chose to believe `BirthYear(David, 1984)`, then `David` is not an answer. Since we have no information that leads us to believe one of the two conflicting tuples over the other, `David` is not an unreasonable answer.

Now suppose we have information that `BirthYear(David, 1984)` is more trustworthy than `BirthYear(David, 1980)`, then we would choose to believe `BirthYear(David, 1984)` over `BirthYear(David, 1980)` and consequently refuse to consider `David` a "good" answer. We refer to such answers as *defeated answers*. In some applications, we would also like to avoid defeated answers.

We can add a single condition in the definition of RQA to filter out consistent relaxations that contain literals defeated by more trustworthy literals.

We obtain the following definition.

Definition 8 (Reasonable Undefeated Query Answers) *Assume a (possibly empty) partial order $\succ$ on the set of ground atoms and negated ground atoms ($p \succ q$ means p is more trustworthy than q). A tuple $\bar{t}$ is an reasonable undefeated query answers (RQA$^{\succ}$) to a query $Q(\bar{x})$ over a database instance B with respect to constraints C if and only if there exists a consistent relaxation B' of B w.r.t. C such that*

- *for each $p \in B'$, there does not exist a set of literals $\{q_1, \ldots, q_k\} \subseteq \text{EXPLICIT}[B]$ such that $\{p, q_1, \ldots, q_k\} \cup C$ is unsatisfiable and $q_i \succ p$ for every $i \in \{1, \ldots, k\}$, and*
- $B' \models^{\text{I}} Q(\bar{t})$.

In fact, rewriting procedures for RQA are flexible enough to accommodate the new definition. A conceptually simple tweak to the definition of RRC_F turns rewriting procedures for RQA into rewriting procedures for RQA$^{\succ}$. The details will be covered in future work.

Acknowledgments

We are grateful to Jeff Ullman, Jennifer Widom, Michael Kassoff, Rada Chirkova, Mary-Anne Williams, Carl Hewitt, Lukasz Golab, attendees of Stanford Logic Group and Stanford Infolab seminars, and anonymous referees for their valuable feedback.

11. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. Foundations of databases, 1995.

[2] F. Afrati, C. Li, and V. Pavlaki. Data exchange in the presence of arithmetic comparisons. In *EDBT '08: Proceedings of the 11th international conference on Extending database technology*, pages 487–498, New York, NY, USA, 2008. ACM.

[3] M. Arenas, L. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *PODS '99 Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 68–79, 1999.

[4] M. Arenas, L. Bertossi, and J. Chomicki. Answer sets for consistent query answering in inconsistent databases. *Theory Pract. Log. Program.*, 3(4):393–424, 2003.

[5] M. Arenas, L. Bertossi, J. Chomicki, X. He, V. Raghavan, and J. Spinrad. Scalar aggregation in inconsistent databases. *Theor. Comput. Sci.*, 296(3):405–434, 2003.

[6] M. Arenas, L. E. Bertossi, and J. Chomicki. Query evaluation in almost consistent databases using residues. In *SCCC*, pages 8–14. IEEE Computer Society, 1998.

[7] M. Arenas and L. Libkin. Xml data exchange: consistency and query answering. In *PODS '05: Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 13–24, New York, NY, USA, 2005. ACM.

[8] M. Arenas and L. Libkin. Xml data exchange: Consistency and query answering. *J. ACM*, 55(2):1–72, 2008.

[9] P. Barceló. Logical foundations of relational data exchange. *SIGMOD Rec.*, 38(1):49–58, 2009.

[10] L. Bertossi and L. Bravo. Consistent query answers in virtual data integration systems. In *IN INCONSISTENCY TOLERANCE, SPRINGER LNCS 3300*, pages 42–83. Springer, 2005.

[11] P. Bosc, M. Galibourg, and G. Hamon. Fuzzy querying with sql: extensions and implementation aspects. *Fuzzy Sets Syst.*, 28(3):333–349, 1988.

[12] F. Bry. Query answering in information systems with integrity constraints. In S. Jajodia, W. List, G. W. McGregor, and L. Strous, editors, *IICIS*, volume 109 of *IFIP Conference Proceedings*, pages 113–130. Chapman Hall, 1997.

[13] A. Calì, D. Calvanese, G. D. Giacomo, and M. Lenzerini. Data integration under integrity constraints. In *CAiSE '02: Proceedings of the 14th International Conference on Advanced Information Systems Engineering*, pages 262–279, London, UK, 2002. Springer-Verlag.

[14] A. Calì, D. Lembo, and R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *PODS '03: Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 260–271, New York, NY, USA, 2003. ACM.

[15] R. Cavallo and M. Pittarelli. The theory of probabilistic databases. In *VLDB '87: Proceedings of the 13th*

International Conference on Very Large Data Bases, pages 71–81, San Francisco, CA, USA, 1987. Morgan Kaufmann Publishers Inc.

[16] J. Chomicki and J. Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Inf. Comput.*, 197(1-2):90–121, 2005.

[17] J. Chomicki and J. Marcinkowski. On the computational complexity of minimal-change integrity maintenance in relational databases. In L. E. Bertossi, A. Hunter, and T. Schaub, editors, *Inconsistency Tolerance*, volume 3300 of *Lecture Notes in Computer Science*, pages 119–150. Springer, 2005.

[18] E. F. Codd. Relational completeness of data base sublanguages. *In: R. Rustin (ed.): Database Systems: 65-98, Prentice Hall and IBM Research Report RJ 987, San Jose, California*, 1972.

[19] O. M. Duschka, M. R. Genesereth, and A. Y. Levy. Recursive query plans for data integration. *The Journal of Logic Programming*, 43(1):49 – 73, 2000.

[20] M. Elvang-Gøransson and A. Hunter. Argumentative logics: Reasoning with classically inconsistent information. *Data Knowl. Eng.*, 16(2):125–145, 1995.

[21] R. Fagin. Horn clauses and database dependencies. *J. ACM*, 29(4):952–985, 1982.

[22] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. In *ICDT '03: Proceedings of the 9th International Conference on Database Theory*, pages 207–224, London, UK, 2002. Springer-Verlag.

[23] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

[24] R. Fagin, P. G. Kolaitis, and L. Popa. Data exchange: getting to the core. In *PODS '03: Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 90–101, New York, NY, USA, 2003. ACM.

[25] A. Fuxman and R. J. Miller. First-order query rewriting for inconsistent databases. *J. Comput. Syst. Sci.*, 73(4):610–635, 2007.

[26] G. Gottlob and A. Nash. Data exchange: computing cores in polynomial time. In *PODS '06: Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 40–49, New York, NY, USA, 2006. ACM.

[27] G. Gottlob and A. Nash. Efficient core computation in data exchange. *J. ACM*, 55(2):1–49, 2008.

[28] G. Grahne and A. O. Mendelzon. Tableau techniques for querying information sources through global schemas. In *ICDT '99: Proceedings of the 7th International Conference on Database Theory*, pages 332–347, London, UK, 1999. Springer-Verlag.

[29] T. L. Hinrichs, J.-Y. Kao, and M. Genesereth. Inconsistency-tolerant reasoning with classical logic and large databases. In *Proc. of the eighth Symposium on Abstraction, Reformulation, and Approximation*, 2009.

[30] P. Jackson. Applications of annotated predicate calculus to querying inconsistent databases. In *Computational Logic - CL2000*, pages 926–941. Springer, Berlin / Heidelberg, 2000.

[31] J.-Y. Kao. Computing query answers with consistent support - a ph.d. project proposal. In *Proceedings of the Third SIGMOD PhD Workshop on Innovative Database Research (IDAR 2009) (to appear)*, 2009.

[32] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[33] M. Kassoff and M. Genesereth. Predicalc: A logical spreadsheet management system. *Knowl. Eng. Rev.*, 22(3):281–295, 2007.

[34] P. G. Kolaitis, J. Panttaja, and W.-C. Tan. The complexity of data exchange. In *PODS '06: Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 30–39, New York, NY, USA, 2006. ACM.

[35] R. Krishnamurthy, R. Kaushik, and J. F. Naughton. Xml views as integrity constraints and their use in query translation. In *ICDE '05: Proceedings of the 21st International Conference on Data Engineering*, pages 693–704, Washington, DC, USA, 2005. IEEE Computer Society.

[36] C.-T. Lee. *A completeness theorem and a computer program for finding theorems derivable from given axioms*. PhD thesis, University of California, Berkeley, 1967.

[37] J. A. Makowsky. Characterizing data base dependencies. In *Proceedings of the 8th Colloquium on Automata, Languages and Programming*, pages 86–97, London, UK, 1981. Springer-Verlag.

[38] R. Manor and N. Rescher. On inferences from inconsistent information. *Theory and Decision 1*, pages 179–219, 1970.

[39] M. S. Paterson and M. N. Wegman. Linear unification. In *STOC '76: Proceedings of the eighth annual ACM symposium on Theory of computing*, pages 181–186, New York, NY, USA, 1976. ACM.

[40] R. Pichler and V. Savenkov. Towards practical feasibility of core computation in data exchange. *Theor. Comput. Sci.*, 411(7-9):935–957, 2010.

[41] H. Prade and C. Testemale. Generalizing database relational algebra for the treatment of incomplete or uncertain information and vague queries. *Information Sciences*, 34:115–143, 1984.

[42] S. Pradhan. Contestations and constraints for databases. In *International Conference on Innovative Applications of PROLOG*, 2001.

[43] S. Pradhan. Argumentation databases. In *Logic Programming*, volume 2916 of *LNCS*, pages 178–193. Springer, 2003.

[44] S. Pradhan. Connecting databases with argumentation. In *Web Knowledge Management and Decision Support*, volume 2543 of *LNCS*, pages 170–185. Springer, 2003.

[45] R. Reiter. Two results on ordering for resolution with merging and linear format. *J. ACM*, 18:630–646, October 1971.