Computing Query Answers with Consistent Support*

A Ph.D. project proposal

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

ABSTRACT

This paper describes a Ph.D. project addressing the problem of computing query answers from an inconsistent database.

Given a database instance, a set of integrity constraints, and a query, we say that an atom is an answer with consistent support if the atom is an answer on some consistent subset of the data. This notion of an answer is based on a "credulous" query semantics. It avoids answers that arise only from inconsistent subsets of the data; but, at the same time, it admits justifiable answers that do not persist under all reasonable repairs of the data. In this paper, we show that the problem of producing query answers with consistent support has non-polynomial combined complexity but polynomial data complexity. We provide an algorithm for rewriting queries such that for any database instance, the standard answers to the rewritten query are exactly the answers with consistent support to the original query.

Keywords

inconsistency, integrity constraints, query rewriting, data integration

1. INTRODUCTION

Integrity constraints form an important component of modern databases. Sometimes, the data in a database may violate the applicable integrity constraints. For example, a database may record two distinct dates of birth for a given person, one in May and another in August. Those records would violate the integrity constraint that each person must have at most one date of birth.

When the data recorded in a database violate the applicable integrity constraints, special care must be taken to avoid nonsensical answers from the database. In the foregoing example, the system must take care to prevent giving the answer that a person is born once in May and once in August.

*This work was supported primarily by the SGER Program of the National Science Foundation under Award Number IIS-0841152

Proceedings of the Third SIGMOD PhD Workshop on Innovative Database Research (IDAR 2009) June 28, 2009, Providence, RI, USA

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 an update 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 integrity constraints even if each autonomous source itself is consistent with all integrity constraints. For instance, two sources of data may show two different body heights for the same person because the two sources are out of sync (the person's body height changed) or because one body height was incorrectly entered. Each data source satisfies the functional dependency that each person has at most one body height, but together the integrated data violates the functional dependency. 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 some integrity constraints for many reasons. A data source may be a legacy system that does not support integrity constraints at all. A data source may elect not to enforce certain constraints for efficiency reasons. Sometimes a database system does not support all types of integrity constraints that are specified.

information preservation. Sometimes inconsistencies are allowed to exist in order to preserve information. For example, it may be desirable to keep two different body heights for the same person while further investigations take place to determine which one is correct.

1.2 Why do inconsistencies matter?

Inconsistencies are very 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 consistent because negative information is implicit). Nonetheless, the standard semantics can produce answers which are not justifiable by any data in the database that is possible according to the integrity constraints. Consider the following example:

Example 1.
Data:
institution <Person, Institution>
(person1, "Stanford University")

pursuing_degree <Person, degree name>
(person1, "MA")

department <Person, Dept>
(person1, "Computer Science")

Constraints:
(1) $institution(X, "StanfordUniversity'') \wedge$
$department(X, "ComputerScience'')$
$\rightarrow \neg\ pursuing\_degree(X, "MA'')$
(a student at the Stanford University Computer Science department cannot be pursuing the MA degree.)

The query q(X) :- institution("Stanford University"), department(X, "Computer Science"), pursuing_degree(X,"MA") (which students at Stanford University Computer Science department are pursuing the MA degree?) would produce the answer set $\{q(person1)\}$. Assuming that the integrity constraints correctly describe the domain, this answer does not make sense because constraint (1) specifies that no such student exists. For most applications, it would be desirable for the answers which are impossible given the constraints to be suppressed (or at least distinguished). It is important to distinguish between answers which are justified by consistent data and answers which arise only from inconsistent data. The next subsection gives a notion of query answers which captures the answers justified by consistent data.

1.3 QACS

A query answer with consistent support (QACS) is a query answer that arises from a consistent subset (weakening) of the data. In the preceding example (example 1), the set of QACS answers is empty because the answer $q(person1)$ arises only from data that violate the constraints.

Consider query answers with consistent support in the expanded example below.

Example 2.
Data:
institution <Person, Institution>
(person1, "Stanford University")
(person2, "Academy of Art University")

pursuing_degree <Person, degree name>
(person1, "MA")
(person2, "MS")

department <Person, Dept>
(person1, "Computer Science")
(person2, "Computer Science")

bayarea_institution <Institution>
("Stanford University")
("Academy of Art University")

name<Person, name>
(person1, "Alyssa")
(person2, "Alyssa")

Constraints:
(1) institution(X,"Stanford University") $\wedge$
department(X,"Computer Science") $\rightarrow$
$\neg$pursuing_degree(X,"MA")
(2) institution(X,"Academy of Art University") $\rightarrow$
$\neg$department(X,"Computer Science")

A = {institution(person1, "Stanford University"), pursuing_degree(person1, "MA"), department(person1, "Computer Science")} violates constraint (1).

B = {institution(person2, "Academic of Art University"), department(person2, "Computer Science")} violates constraint (2).

Removing any one element of A together with any one element of B from the data leads to a subset (or a weakening) of the data that is consistent with the constraints.

Now using this database, suppose I am trying to find a lost friend named Alyssa. I know that Alyssa is studying in the computer science department of a San Francisco Bay Area institution. So I decide to pose the following query and call the people in the result.

p(X) :- institution(X, Y), bayarea_institution(Y), department(X, "Computer Science"), name(X, "Alyssa")

Ignoring the integrity constraints, the answers are $person1$ and $person2$.

There is no consistent evidence that $person2$ is a computer science student in the SF Bay Area. If the information that she is in the computer science department is correct, then the information that she is at the Academy of Art University is incorrect, leaving us with no reason to believe she is a Bay Area student. On the other hand, if the information that he is at the Academy of Art University of correct, then the information that she is in the computer science department is incorrect, leaving us with no reason to believe she is a computer science student. So assuming that the constraints are reliable, there is no reason for me to consider $person2$ as a candidate for the friend I am looking for.

The answer $person2$ is excluded from the answers with consistent support. The complete set of QACS is $\{person1\}$.

1.4 A query rewriting approach

A straightforward method for computing the query answers with consistent support is to enumerate all the maximal consistent subsets of the data and take the union of the query answers obtained from each subset. However, as will be seen in example 3, the number of subsets considered may be exponential in the size of the data.

To produce a practical method for computing QACS, a different approach is required. In the Ph.D. project, I take a query rewriting approach to computing QACS.

Given a query q and integrity constraints C, the goal is to find a rewritten query Q such that for any database instance, the standard answers to the rewritten query Q are exactly the answers with consistent support to the original query q. Since the query rewriting is done independently of the data, the procedure for finding query answers with consistent support has polynomial data complexity for the class of queries considered. Moreover, if the same query is executed many times on different underlying data, the rewriting needs to be done only once. This feature is valuable because many queries are executed multiple times on changing data.

Furthermore, the query rewriting approach can be layered on top of an existing database system without any modifications to the existing system. So the cost of adoption is low and all the advantages of the existing database system (e.g., industrial-strength robustness, ability to handle massive datasets, purpose-tuned performance) are preserved.

2. PRELIMINARIES

In the following discussion, we assume a fixed database schema and a fixed infinite database domain D. Elements of D with different names are distinct (Unique Name Assumption).

A database instance is viewed as a finite collection of around atoms. Negative information is implicitly represented under the Closed World Assumption (CWA). We call the set of ground literals obtained by explicitly adding all the negative information to a database instance the *theory represented by* the database instance.

Integrity constraints on a database are first-order sentences over the database schema and domain. We assume that integrity constraints by themselves are consistent in the sense that there is a database instance which satisfies the integrity constraints.

The query class we consider is the class of first-order queries (equivalent to relational calculus and relational algebra [8]). Each query is expressed as a finite collection of flat DATALOG rules. For simplicity, we first consider a single flat DATALOG rule. As will be discussed later, all the results generalize readily to a finite collection of flat DATALOG rules.

Definition 1. A theory B' is a *weakening* of a database instance B, iff B' is a subset of the theory represented by B.

Definition 2. A theory B' is a *consistent weakening* of a database instance B, w.r.t. a set of integrity constraints IC, iff B' is a weakening of B and B' is consistent w.r.t. IC.

Definition 3. A ground literal $\bar{t}$ is an *answer with consistent support* (QACS) to a query $q(\bar{x})$ in a database instance B with respect to integrity constraints IC iff there exists a consistent weakening B' of B w.r.t. IC such that $\bar{t}$ is an answer to the query $q(\bar{x})$ in B'. That is,
$B \models_{IC} q(\bar{t})$ iff $B' \models q(\bar{t})$ for some consistent weakening B' of B.

3. COMPUTING THE QACS

The problem of computing the QACS answers subsumes the basic database query problem because, in the absence of integrity constraints, the QACS answers are exactly the standard answers to a query. Because the first-order query problem is known to be **PSPACE**-complete in combined complexity [18] , the problem of computing QACS answers given a query, a set of integrity constraints, and a database instance is **PSPACE**-hard in combined complexity.

However, with a fixed first-order query, the problem of determining whether an atom is an answer to the query in a given database instance can be done in time polynomial in the size of the data (in fact the problem is in **LOGSPACE** [18]). That is, with a fixed first-order query $Q(\bar{x}, \{(\bar{a}, B) | B \models Q(\bar{a})\} \in \mathbf{P}$. So we would hope for a procedure that computes QACS answers and also has polynomial data complexity.

Based on the definition, a simple method for computing the QACS is to compute the query answers from each maximal consistent weakening of the given database instance and report the union of all the answers. However, computing all maximal consistent weakenings of a given database instance is not feasible. Even with a single functional dependency, the number of maximal consistent weakenings can be exponential in the size of the original database instance.

Example 3. (taken from example 2.6 in [4])
Consider a family of relation instances r_n:

r_n<A,B>: (a_1, b_0),(a_1, b_1),(a_2, b_0),(a_2, b_1),...,(a_n, b_0), (a_n, b_1)
With a functional dependency $A \rightarrow B$, that is, $\neg r_n(x, y) \vee \neg r_n(x, z) \vee y = z$, each relation instance r_n has $2n$ tuples and 2^n maximal consistent weakenings.

Our approach to efficiently computing the QACS answers to a query is to augment the query with checks to block answers from being generated by an inconsistent subset of the data. Because the rewriting is done independently of the data, and the rewritten query can be evaluated in time polynomial in the size of the data, the resulting procedure for computing QACS answers has polynomial data complexity.

4. A REWRITING ALGORITHM

First we define a QACS rewriting for a collection of flat DATALOG rules.

Definition 4. Given a collection P of flat DATALOG rules and integrity constraints IC, P' is a QACS rewriting of P iff for every database instance B, the QACS answers of P in B w.r.t. to IC are exactly the standard answers of P' in B. That is,
$P \cup B \models_D a$ iff $P' \cup B \models_{IC} a$, for every ground atom a.

The rewriting algorithms presented here makes the following assumptions regarding the inputs:

> The input query is a flat DATALOG query.
>
> The integrity constraints are a set of function-free universal constraints (matrix is in CNF, no function, all variables universally quantified) given in clausal form.
>
> The equality built-in predicate (=) is allowed in queries and integrity constraints.

A flat DATALOG query is a collection of flat DATALOG queries. By the property that QACS distributes over disjunction, a QACS rewriting of a collection of flat DATALOG queries is simply the collection of the QACS rewritings of each flat DATALOG queries. For simplicity, the algorithms presented below assume the input query is a single flat DATALOG query.

4.1 No built-in predicates in constraints

An outline of the algorithm for the case when there are no built-in predicates in the constraints (see Algorithm 1 for more detailed pseudo-code):

> 1. Use resolution method to close the set of clauses under resolution, so that each inconsistent set of data must violate a single clause in the closed set of clauses.

2. Find all the maximally general substitutions to the query body that would cause the body to contradict a clause in the closed set of clauses.

3. Augment the query with (negated) equality literals to prevent precisely those substitutions found in step 2 from being used in binding the variables to data when evaluating the augmented query.

This example, continuing from example 2, illustrates how the algorithm works on concrete inputs.

Example 4.
Constraints:
(1) {¬institution(X,"Stanford University"),
¬department(X,"Computer Science"),
¬pursuing_degree(X,"MA")}
(2) {¬institution(X,"Academy of Art University"),
¬department(X,"Computer Science")}

Closing under resolution does not add any additional clauses.

query 1: p(V) :- institution(V, Y), bayarea_institution(Y), department(V, "Computer Science"), name(V, "Alyssa")

Maximally general substitutions that would cause the body of the query to contradict a clause in the closed set of constraint clauses:
{"Academy of Art University" → Y}

rewritten query:
p'(V) :- institution(V, Y), bayarea_institution(Y), department(V, "Computer Science"), name(V, "Alyssa"),
Y ≠ "Academy of Art University"
Evaluating this rewritten query on the data in example 2 indeed gives the query answers with consistent support:
$\{p'(person1)\}$

query 2: p(V) :- institution(V, Y), bayarea_institution(Y), department(V, Z), technical_department(Z), name(V, "Alyssa")

Maximally general substitutions that would cause the body of the query to contradict a clause in the closed set of constraint clauses:
{"Academy of Art University" → Y, "Computer Science" → Z}

rewritten query:
p'(V) :- institution(V, Y), bayarea_institution(Y), department(V, Z), technical_department(Z), name(V, "Alyssa") ∧
¬("Academy of Art University" = Y ∧ "Computer Science" = Z)

4.2 Allowing built-in predicates

The outline given in section 4.1 assumes that there are no built-in predicates in the constraints. The more general algorithm which allows the built-in equality predicate in the constraints is outlined below (see Algorithm 1 for more detailed pseudo-code).

Step 1. Use resolution method to close the set of clauses under resolution.

Step 2. Treat the clause in the closure as a disjunction of two portions, the database predicates portion $\delta(\bar{x})$ and the built-in predicates portion $\theta(\bar{x})$. Find all the maximally general substitutions to the query body that would cause the body to contradict the database predicates portion of a clause in the closure. Associate with each such substitution the residue remaining after resolving away the database predicates portion of a constraint clause using the query body.

Step 3. For each substitution found in Step 2, augment the query with (negated) equality literals and the associated residue so data bindings that match the substitution and also violate the associated residue are disallowed in evaluating the augmented query.

Examples of how the algorithm works: (built-in equality predicate)

Example 5.
constraints:
¬department(X,"Computer Science") ∨¬pursuing_degree(X,Y) ∨ Y = "BS" ∨ Y = "MS" ∨ Y = "PhD"
(a computer science student must be pursuing a BS, an MS, or a PhD)

query:
q(T) :- department(T,U), technical_department(U), pursuing_degree(T,V), graduate_degree(V)

Maximally general substitutions that would cause the body of the query to contradict (the database predicates portion of) a clause in the closed set of constraint clauses:
{"Computer Science" → U}
residue: V = "BS" ∨ V = "MS" ∨ V = "PhD"

rewritten query:
q'(T) :- department(T,U), technical_department(U), pursuing_degree(T,V), graduate_degree(V), **(U ≠ "Computer Science" ∨ (V = "BS" ∨ V = "MS" ∨ V = "PhD"))**

Another example of how the algorithm works (with built-in equality predicate expressing functional dependency).

Example 6.
Constraints:
¬pursuing_degree(X,Y) ∨¬pursuing_degree(X,Z) ∨ Y = Z
(each person may pursue at most one degree concurrently)

query:
q(R) :- pursuing_degree(R,S),graduate_degree(S), pursuing_degree(R,T),professional_degree(T) (asks for people who are pursuing a graduate degree and also pursuing a, possibly distinct, professional degree)

Maximally general substitutions that would cause the body of the query to contradict (the database predicates portion of) a clause in the closed set of constraint clauses:
{empty substitution} residue: S=T

rewritten query:
q(R) :- pursuing_degree(R,S),graduate_degree(S), pursuing_degree(R,T),professional_degree(T), S = T

Algorithm 1 QACS-Rewrite$[q$:- $\phi(\bar{x}), IC]$

1: $R := IC$ closed under resolution
2: $S :=$ Inconsistent-Substitutions$[\phi(\bar{x}), R]$
3: **return** $q'(\bar{x})$:- $\phi(\bar{x}) \wedge \bigwedge_{\langle\sigma,r\rangle\in S}[r \vee \bigvee_{(t\to u)\in\sigma} t \neq u]$

Algorithm 2 Inconsistent-Substitutions$[\phi(\bar{x}), R]$

1: $S := \emptyset$
2: **for all** $c \in R$ **do**
3: **for all** set of unit resolutions U using $\phi(\bar{x})$ on c **do**
4: $r :=$ the residue of c after applying U
5: **if** the residue r contains only built-in predicates **then**
6: $\sigma :=$ the substitution into $\phi(\bar{x})$ in applying U
7: $S := S \cup \{\langle\sigma, r\rangle\}$
8: **end if**
9: **end for**
10: **end for**
11: **return** S

5. EVALUATION OF THE REWRITING ALGORITHM

5.1 Finite resolution closure of constraints

Termination of the rewriting algorithm requires that the set of constraint clauses have finite closure under resolution. The requirement for the resolution closure of the constraints to be finite can be guaranteed by the following syntactic restriction on the integrity constraints:

Definition 5. A set of integrity constraints IC is acyclic iff there exists a function f from each literal in the database to the natural numbers such that for every integrity constraint $l_1(\bar{x_1})\vee l_2(\bar{x_2})\vee\cdots\vee l_k(\bar{x_i}) \in IC$, and every i and j $(1 \leq i, j \leq k)$, if $i \neq j$ then $f(\neg l_i) > f(l_j)$. ($\neg l_i$ is the complementary literal of l_i.)

An acyclic set of integrity constraints has finite closure under resolution. This is the same condition as the necessary and sufficient condition for syntactic finiteness to guarantee termination of the query rewriting procedure proposed by Arenas, Bertossi, and Chomicki for consistent query answers (CQA) [1, 3, 4]. That the same condition guarantees termination in both our query rewriting algorithm for QACS and the query rewriting algorithm for CQA is not surprising because both procedures rely critically on resolution. However, the query rewriting algorithm for CQA requires additional restrictions on the query to guarantee completeness.

5.2 Reusing computations

The algorithm computes the resolution closure of the constraints as a first step. The result of this computation can be reused to rewrite any number of queries as long as the integrity constraints remain unchanged (changes to the data also do not matter because the whole rewriting algorithm is independent of the data). This is a very powerful feature because most database systems have integrity constraints that change very infrequently. Typically, the integrity constraints are written at the time of schema design and left unchanged throughout the life of the database. So the cost of computing the resolution closure of the constraints may be amortized over the life of the database.

5.3 Overhead above original query

The rewriting approach gives a procedure for computing QACS with polynomial data complexity because the rewriting is done independently of the data. However, the degree of the polynomial depends on the specific query considered. It is important to consider how long it takes to evaluate the rewritten query compared to how long it takes to evaluate the original query.

Because the rewriting procedure merely augments the original query with a boolean combination of non-equalities, the join structure of the rewritten query is exactly the same as that of the original query. So it is to be expected that evaluating the rewritten query takes about the same amount of time as does evaluating the original query.

6. NEXT STEPS

The rewriting algorithm given here represents an important first step in computing query answers with consistent support. This section outlines the next steps in the Ph.D. project.

6.1 Expanding query and constraint classes

The rewriting algorithm currently restricts the language of integrity constraints to universal constraints. Universal constraints express most major classes of database constraints, but one notable class of integrity constraints that requires existential quantification is the class of referential integrity constraints [4]. In future work, I plan to extend the rewriting algorithm to a broader constraint language that allows for existential quantification.

I also plan to expand the class of queries and constraints allowed in other dimensions, including recursive queries and additional built-in predicates. Understanding the more complex interaction between an expanded class of queries and an expanded class of constraints will be key to achieving these goals.

6.2 As-needed resolution

A positive feature of the rewriting algorithm given is that the resolution closure of the integrity constraints may be computed only once and reused for rewriting different queries. An alternative is to perform resolution on the constraints only as needed by an input query. The alternative has the advantage that Set of Support resolution strategy [19] can be used with the body of the query (in clauses) as the set of support. The strategy would prevent resolving constraints needlessly with each other.

Carrying one step further the idea of as-needed resolution, a stop-anytime algorithm may be devised to augment the original query as the resolution takes place, producing more and more accurate rewritings as more resolution is completed. Such an algorithm is useful in situations where resolution of the integrity constraints takes a very long time or fails to terminate (because the termination conditions are not satisfied).

6.3 View maintenance

Often, a query is used to instantiate a view that is maintained. In such a situation, it is desirable to avoid re-evaluating the query from scratch whenever the data changes. Much work has been done to reduce the amount of incremental computation required to maintain a view when the

underlying data changes. One particular approach by Orman uses a differential relational calculus to compute the change to a view given the change to the base relations [14]. I plan to explore a similar approach for maintaining views which are query answers with consistent support.

6.4 Changing constraints

Consider the situation that the integrity constraints on a database are amended. Constraints may be added and removed. It would be desirable in such a case to avoid computing query rewritings and answers with consistent support from scratch. I plan to investigate data structures and algorithms to reuse old computations in computing new rewritings and new answers in response to a change in the integrity constraints.

7. RELATED WORK

The idea of reasoning from consistent subsets of inconsistent information is well known [13, 9]. The notion of query answers with consistent support is a variation on existential Ω-entailment [12].

There is a large body of work on producing consistent query answers from databases that may contain inconsistencies [1, 10, 2, 7], 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 the goal of this Ph.D. project.

Shekhar Pradhan investigated using annotated logic programs to give databases argumentation capacity in answering queries [17, 16]. However, that body of work do not result in a query rewriting procedure which rewrites the query in the original query language. Moreover, that body of work cannot be directly applied to find the QACS answers because the straightforward transformation of integrity constraints into contestations would result in contested annotated logic programs without a model. The kind of computation proposed in this paper would be required to transform integrity constraints into a contested annotated logic program that computes the set of query answers with consistent support.

Possibilistic, probabilistic, and fuzzy databases [15, 6, 5] are also related to the problem considered in this Ph.D. project. One key feature that distinguishes the problem considered in this Ph.D. project is that we start with a standard relational model without possibility or probabilistic annotations on each tuple. Instead, the admissibility of a group of tuples in answering a query is decided by considering the integrity constraints.

The query rewriting algorithms in this paper are based on recent work in inconsistency-tolerant reasoning [11]. However, the algorithm proposed in this paper specifically addresses the problem of query rewriting and handles built-in predicates.

Acknowledgment

I would like to thank Michael Genesereth for advising my thesis. I am also grateful to Michael Kassoff, Rada Chirkova, and Mary-Anne Williams for valuable discussions.

8. REFERENCES

[1] M. Arenas, L. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. pages 68–79, 1999.

[2] 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.

[3] L. Bertossi. Consistent query answering in databases. *SIGMOD Rec.*, 35(2):68–76, 2006.

[4] L. Bertossi and J. Chomicki. Query answering in inconsistent databases. In Saake and R. van der Meyden, editors, *Logics for Emerging Applications of Databases*.

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

[6] 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.

[7] J. Chomicki and J. Marcinkowski. On the computational complexity of minimal-change integrity maintenance in relational databases. In *Inconsistency Tolerance, Springer LNCS 3300*, pages 119–150. Springer, 2004.

[8] 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.

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

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

[11] T. L. Hinrichs, J.-Y. Kao, and M. Genesereth. Inconsistency-tolerant reasoning with classical logic and large databases. In *Symposium on Abstraction, Reformulation, and Approximation (to appear)*, 2009.

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

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

[14] L. V. Orman. Transaction repair for integrity enforcement. *IEEE Trans. Knowl. Data Eng.*, 13(6):996–1009, 2001.

[15] 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.

[16] S. Pradhan. Argumentation databases. In *Logic Programming*.

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

[18] M. Vardi. Complexity of relational query languages. In *14th ACM STOC Symp*, pages 137–146, 1982.

[19] L. Wos, G. A. Robinson, and D. F. Carson. Efficiency and completeness of the set of support strategy in theorem proving. *J. ACM*, 12(4):536–541, 1965.