Inconsistency Management Policies

Maria Vanina Martinez[1], **Francesco Parisi**[2], **Andrea Pugliese**[2], **Gerardo I. Simari**[1], **V. S. Subrahmanian**[1]

[1] University of Maryland College Park
College Park, MD 20742, USA
{mvm,gisimari,vs}@cs.umd.edu

[2] Università della Calabria
Via Bucci − 87036 Rende (CS), Italy
{fparisi,apugliese}@deis.unical.it

Abstract

Though there is much work on how inconsistency in databases *should* be managed, there is good reason to believe that end users will want to bring their domain expertise and needs to bear in how to deal with inconsistencies. In this paper, we propose the concept of *inconsistency management policies* (IMPs). We show that IMPs are rich enough to specify many types of inconsistency management methods proposed previously, but provide end users with tools that allow them to use the policies that they want. Our policies are also capable of allowing inconsistency to persist in the database or of eliminating more than a minimal subset of tuples involved in the inconsistency. We present a formal axiomatic definition of IMPs and present appropriate complexity results, together with results linking different IMPs together. We extend the relational algebra (RA) to incorporate IMPs and present theoretical results showing how IMPs and classical RA operators interact.

Introduction

The management of inconsistent databases has been studied for many years by many different researchers (Grant 1978; Baral, Kraus, and Minker 1991; Benferhat, Dubois, and Prade 1997; Besnard and Schaub 1998; Arenas, Bertossi, and Chomicki 1999; Bohannon et al. 2005; Calì, Lembo, and Rosati 2003; Bertossi and Chomicki 2003; Chomicki 2007) However, almost all past approaches proceeded under the assumption that there was some epistemically correct way of resolving inconsistencies or reasoning in the presence of inconsistency. More recently, (Subrahmanian and Amgoud 2007) argued that inconsistency can often be resolved in different ways based on what the user wants, and they provided a mechanism to reason about maximal consistent subsets (also called "repairs" by (Arenas, Bertossi, and Chomicki 1999)) using objective functions where the user gets to choose an objective function.

To see why, let us consider the very simple salary table given below. This "Salary Example" will be used throughout the paper.

Copyright © 2008, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

	Name	*Salary*	*Tax_bracket*	*Source*
t_1	$John$	$70K$	15	s_1
t_2	$John$	$80K$	20	s_2
t_3	$John$	$70K$	25	s_3
t_4	$Mary$	$90K$	30	s_1

Let us assume that salaries are uniquely determined by names. In this case, a user may want to resolve the inconsistency about John's salary in many different ways. (C1) If he were considering John for a loan, he might want to choose the lowest possible salary of John to base his loan on. (C2) If he were assessing the amount of taxes John has to pay, he may choose the highest possible salary John may have. (C3) If he were just trying to estimate John's salary, he may choose some number between 70K and 80K (*e.g.*, the average of the three reports of John's salary) as the number. (C4) if he had different degrees of confidence in the sources that provided these salaries, he might choose a weighted mean of these salaries. (C5) He might choose not to resolve the inconsistency at all, but to just let it persist till he can clear it up. (C6) He might simply consider *all* the data about John unreliable and might want to ignore it till it can be cleared up – this is the philosophy of throwing away all contaminated data.[1] We are not aware of a single piece of past work that can handle all six reasonable possibilities mentioned above. (Baral, Kraus, and Minker 1991; Subrahmanian and Amgoud 2007; Arenas, Bertossi, and Chomicki 1999; Bohannon et al. 2005; Calì, Lembo, and Rosati 2003; Bertossi and Chomicki 2003; Chomicki 2007) can handle cases C1 and C2, but not the other cases.

Each of cases C1 through C6 reflects a *policy* that the user is using to resolve inconsistencies. Much as a carpenter has tools like hammers and saws, we propose to put data cleaning policies in the hands of users so that they can use them as tools, when appropriate, for reasoning about the data. *It is important to enable users to bring their application specific knowledge to bear when resolving inconsistencies.*

The contributions of this paper are as follows: we first define the concept of a *policy* for managing inconsistency. Our notion of an inconsistency management policy generalizes

[1] This is more likely to happen, for example, when there is a scientific experiment with inconsistent data or when there is a critical action that must be taken, but cannot be taken on the basis of inconsistent data.

(Arenas, Bertossi, and Chomicki 1999) by allowing policies to either remove inconsistency completely or to allow part or all of the inconsistency to persist. Our notion of a policy accounts for all six cases above, and many more. We start with policies applicable to a single functional dependency (FD for short) – one of the most common kinds of integrity constraints used in databases – and then extend policies to manage multiple FDs. We present two semantics to manage multiple FDs based on partial orderings on the importance of FDs. We present results showing that these semantics lead to various NP-hardness results. We then show that our policies can be embedded as operators within the relational algebra in two different ways – one where the policy is applied first (before other relational operators) and another where it is applied last. We study the interaction of these policy usage methods with other relational query operations and provide several interesting results.

Syntax and Notation

We assume the existence of relational schemas of the form $S(A_1, \ldots, A_n)$ (Ullman 1988) where the A_i's are attributes. Each attribute A_i has an associated domain, $dom(A_i)$. A *tuple* over S is a member of $dom(A_1) \times \cdots \times dom(A_n)$, and a set of such tuples is called a *relation*. We use $t[A_i]$ to denote the value of the A_i attribute of tuple t. We use $Attr(S)$ to denote the set of all attributes in S.

Given the relational schema $S(A_1, \ldots, A_n)$, a functional dependency (FD) *fd* over S is an expression of the form $A'_1, \ldots, A'_k \rightarrow A'_{k+1}, \ldots, A'_m$, where $\{A'_1, \ldots, A'_m\} \subseteq Attr(S)$. A relation R over the schema S *satisfies* the above FD iff $\forall t_1, t_2 \in R$, $t_1[A'_1] = t_2[A'_1] \wedge \ldots \wedge t_1[A'_k] = t_2[A'_k] \Rightarrow t_1[A'_{k+1}] = t_2[A'_{k+1}] \wedge \ldots \wedge t_1[A'_m] = t_2[A'_m]$. Without loss of generality, we assume that every functional dependency fd has exactly one attribute on the right-hand side (i.e., $k+1 = m$) and denote this attribute as $RHS(fd)$. Moreover, with a little abuse of notation, we write that fd is defined over R.

Definition 1 *Let R be a relation and $\mathcal{F}$ a set of functional dependencies. A* culprit *is a set $c \subseteq R$ not satisfying $\mathcal{F}$ such that $\forall\, c' \subset c$, c' satisfies $\mathcal{F}$.*

For instance, the culprits in the example of the Introduction are $\{t_1, t_2\}$ and $\{t_2, t_3\}$. We use $culprits(R, \mathcal{F})$ to denote the set of culprits in R w.r.t. $\mathcal{F}$.

Definition 2 *Let R be a relation and $\mathcal{F}$ a set of functional dependencies. Given two culprits $c, c' \in culprits(R, \mathcal{F})$, we say that c and c'* overlap*, denoted $c \triangle c'$, iff $c \cap c' \neq \emptyset$.*

Definition 3 *Let $\triangle^*$ be the reflexive transitive closure of relation $\triangle$. A* cluster *is a set $cl = \bigcup_{c \in e} c$ where e is an equivalence class of $\triangle^*$.*

In the example of the Introduction, the only cluster is $\{t_1, t_2, t_3\}$. We will denote the set of all clusters in R w.r.t. $\mathcal{F}$ as $clusters(R, \mathcal{F})$.

Inconsistency Management Policies

In this section, we introduce the concept of *policy* for managing inconsistency in databases violating a given set of functional dependencies. Basically, applying an inconsistency management policy on a relation results in a new relation with the intention of obtaining a lower degree of inconsistency.

Definition 4 *An inconsistency management policy (IMP for short) for a relation R w.r.t. a set of functional dependencies $\mathcal{F}$ over R is a function $\gamma_{\mathcal{F}}$ from R to a relation $R' = \gamma_{\mathcal{F}}(R)$ that satisfies the following axioms:*

Axiom A1. *If $t \in R - \bigcup_{c \in culprits(R,\mathcal{F})} c$, then $t \in R'$. This axiom says that tuples that do not belong to any culprit cannot be eliminated or changed.*

Axiom A2. *If $t' \in R' - R$, then there exists a cluster c and a tuple $t \in c$ such that for each attribute A not appearing in any $fd \in \mathcal{F}$, $t.A = t'.A$. This axiom says that every tuple in R' must somehow be linked to a tuple in R.*

Axiom A3. *$\forall fd \in \mathcal{F}$, $|culprits(R, \{fd\})| \geq |culprits(R', \{fd\})|$. This axiom says that the IMP cannot increase the number of culprits.*

Axiom A4. *$|R| \geq |R'|$. This axiom says that the IMP cannot increase the cardinality of the relation.*

$\gamma_{\mathcal{F}}$ is a total *(resp.* partial*) inconsistency management policy if $\gamma_{\mathcal{F}}(R)$ satisfies all (resp. not all) FDs in $\mathcal{F}$. $\gamma_{\mathcal{F}}$ is a* singular *IMP iff $\mathcal{F}$ is a singleton.*

When $\mathcal{F} = \{fd\}$ we write γ_{fd} instead of $\gamma_{\{fd\}}$.

It is important to note that axioms (A1) through (A4) above are not meant to be exhaustive. They represent a minimal set of conditions that we believe any inconsistency management policy should satisfy. Specific policies may satisfy additional properties.

Singular IMPs

In this section, we introduce three types of IMPs: *tuple-based*, *value-based*, and *interval-based*. Later, we will extend these strategies to manage multiple FDs. Our goal is to allow the end user to choose a policy that best matches his needs.

Definition 5 *An IMP τ_{fd} for a relation R w.r.t. a functional dependency fd is said to be a* tuple-based policy *if each cluster $cl \in clusters(R, \{fd\})$ is replaced by $cl' \subseteq cl$ in $\tau_{fd}(R)$.*

Tuple-based IMPs generalize the well known notion of maximal consistent subsets (Baral, Kraus, and Minker 1991) and repairs (Arenas, Bertossi, and Chomicki 1999) by allowing a cluster to be replaced by any subset of the same cluster. Notice that tuple-based IMPs allow inconsistency to persist – a user may choose to retain all inconsistency (case C5) or retain part of the inconsistency. For instance, if the user believes only sources s_1, s_2 in the Salary Example, he might choose to replace the cluster $\{t_1, t_2, t_3\}$ by the cluster $\{t_1, t_2\}$ as shown below.

	Name	*Salary*	*Tax_bracket*
t_1	$John$	$70K$	15
t_2	$John$	$80K$	20
t_4	$Mary$	$90K$	30

(Baral, Kraus, and Minker 1991; Arenas, Bertossi, and Chomicki 1999) do not allow this possibility. Observe that this kind of policy can cause some information to be lost as a side effect. In our example, although the *Tax_bracket* 25 is not involved in any FD, it is lost when the policy is applied. We now introduce two kinds of policies that avoid this problem. The first kind of policy is based on the notion of *cluster simplification*.

Definition 6 *Given a cluster $cl \in clusters(R, \{fd\})$, cl' is a cluster* simplification of cl *iff $\forall t \in cl$, either $t \in cl'$ or there exists exactly one tuple $t' \in cl'$ obtained from tuple t by replacing $t[RHS(fd)]$ with $t''[RHS(fd)]$ where $t'' \in cl$.*

A simplification allows replacement of values in tuples in the same cluster (in the attribute associated with the right-hand side of an FD). This leads to the following kind of IMP.

Definition 7 *An IMP ν_{fd} for a relation R w.r.t. a functional dependency fd is said to be a* value-based policy *if each cluster $cl \in clusters(R, \{fd\})$ is replaced by a cluster simplification of cl in $\nu_{fd}(R)$.*

Thus a value-based IMP either leaves a cluster unchanged or reduces the number of distinct values for the attribute in the right-hand side of the functional dependency. A user may, for example, decide to use his knowledge that s_1 reflects more recent information than s_2 to reset the s_2 information to that provided by s_1. In this case, the relation returned by the value-based policy would be the one shown below.

	Name	*Salary*	*Tax_bracket*
t_1	$John$	$70K$	15
t_2	$John$	$70K$	20
t_3	$John$	$70K$	25
t_4	$Mary$	$90K$	30

The third kind of policy is called *interval-based*.

Definition 8 *An IMP ξ_{fd} for a relation R w.r.t. a functional dependency fd is said to be an* interval-based policy *if $\forall cl \in clusters(R, \{fd\})$, either $cl \subseteq \xi_{fd}(R)$ or $\xi_{fd}(R) = (R - \{t_1, \ldots, t_n\}) \cup \{t'_1, \ldots, t'_n\}$ where* (i) *$\{t_1, \ldots, t_n\} \subseteq cl$ and* (ii) *$\forall t'_i \in \{t'_1, \ldots, t'_n\}$ there is a tuple $t_i \in cl$ such that for all attributes $A \neq RHS(fd)$, $t'_i.A = t_i.A$, and $t'_i.RHS(fd) = v$ where v is any value in the interval $[\min_{t \in cl}(t[RHS(fd)]), \max_{t \in cl}(t[RHS(fd)])]$.*

The interval-based policy allows any tuple in a cluster to be replaced by a new tuple having a different value for attribute $RHS(fd)$.[2] For example, we may replace the values of the $Salary$ attribute of the tuples in cluster $\{t_1, t_2, t_3\}$ in the Salary example by a value equal to $73.33K$ (the mean of the three salary values for John). Or, if the reliability of sources s_1, s_2, s_3 are 1, 3, and 2, respectively, we might replace the values of the $Salary$ attribute with the weighted mean $(70K * 1 + 80K * 3 + 70K * 2)/6 = 75K$. Thus, the interval-based policy allows cases C3 and C4 in the Introduction to be handled.

[2] Another kind of policy could use the interval $[\min_{t \in cl}(t[RHS(fd)]), \max_{t \in cl}(t[RHS(fd)])]$ in the new tuple, as the value for attribute $RHS(fd)$. In order to store, for each attribute, an appropriate interval, this kind of policy would require an extension of the database schema. We do not consider them in this work.

Proposition 1 *The tuple-based, value-based, and interval-based policies satisfy axioms A1, A2, A3, and A4.*

Observe that, given a relation R and a functional dependency *fd* over R, for every tuple-based policy τ_{fd} we have that $\tau_{fd}(R) \subseteq R$. This does not hold for value-based and interval-based policies, as shown in the example for value-based policies, where tuple *(John, 70K, 20)* does not belong to the original relation.

Proposition 2 *Given a relation R over a schema S and a functional dependency $fd : A_1, \ldots, A_k \rightarrow B$ over R,*

1. *for each tuple-based policy τ_{fd}, there is a value-based policy ν_{fd} such that $\tau_{fd}(R) \subseteq \nu_{fd}(R)$; moreover, if $Attr(S) = \{A_1, \ldots, A_k, B\}$, then $\tau_{fd}(R) = \nu_{fd}(R)$.*
2. *for each value-based policy ν_{fd}, there is an interval-based policy ξ_{fd} such that $\nu_{fd}(R) = \xi_{fd}(R)$.*

Several authors (Lozinskii 1994; Grant and Hunter 2006; Hunter and Konieczny 2005; Grant and Hunter 2008) have proposed approaches to characterize how dirty a database is. The following result says that all the kinds of IMPs mentioned reduce the dirtiness or degree of inconsistency of a database independently of which of these approaches is considered.

Proposition 3 *Consider a relation R, a functional dependency fd over R, and an IMP γ_{fd} that is either a tuple-based, value-based, or interval-based policy. The dirtiness of $\gamma_{fd}(R)$ is less than or equal to the dirtiness of R for any of the definitions of dirtiness given in (Lozinskii 1994; Grant and Hunter 2006; Hunter and Konieczny 2005; Grant and Hunter 2008).*

Multi-Dependency Policies

Suppose each $fd \in \mathcal{F}$ has a single-dependency policy associated with it (specifying how to manage the inconsistencies in the relation with respect to that FD). We assume that the system manager specifies a partial ordering $\leq_{\mathcal{F}}$ on the FDs, specifying their relative importance. Let $TOT_{\leq_{\mathcal{F}}}(\mathcal{F})$ be the set of all possible total orderings of FDs w.r.t. $\leq_{\mathcal{F}}$: this can be obtained by topological sorting.

Definition 9 *Given a relation R, a set of functional dependencies $\mathcal{F}$, a partial ordering $\leq_{\mathcal{F}}$, and an order $o = \langle fd_1, \ldots, fd_k \rangle \in TOT_{\leq_{\mathcal{F}}}(\mathcal{F})$, a* multi-dependency IMP *(MDIMP for short) for R w.r.t. o and $\mathcal{F}$ is a function $\mu^o_{\mathcal{F}}$ from a relation R to a relation $\gamma_{fd_k}(\ldots \gamma_{fd_2}(\gamma_{fd_1}(R))\ldots)$, where $\gamma_{fd_1}, \ldots, \gamma_{fd_k}$ are the singular dependency policies associated with $fd_1, \ldots, fd_k$, respectively.*

Basically, all that a total ordering does is to specify the order in which the conflicts are resolved. We start by resolving the conflict involving the first FD in the ordering, then the second, and so forth. However, different total orderings can lead to different results.

Example 1 *Consider the Salary Example presented in the Introduction and the set of FDs $\{fd_1, fd_2\}$ where fd_1 is*

Name → Salary and fd_2 is Name → Tax_bracket. Suppose the tuple-based policy τ_{fd_1} selects the tuple with the highest value of the Salary attribute (when inconsistency occurs), while τ_{fd_2} selects the lowest value of the Tax_bracket attribute. Under the total order $o = \langle fd_1, fd_2 \rangle$, we get $\{(John, 80K, 20), (Mary, 90K, 30)\}$ as the result. Note that after τ_{fd_1} is applied, the other policy is not, because there is no further inconsistency w.r.t. fd_2. Therefore, τ_{fd_1} is solely responsible for deciding what tuples are part of the final answer. Under the total order $o = \langle fd_2, fd_1 \rangle$, the result of applying the multi-dependency policy will be $\{(John, 70K, 15), (Mary, 90K, 30)\}$. Here, τ_{fd_2} decides which tuples are in the answer, causing the application of τ_{fd_1} to have no effect.
Now consider the set of FDs $\{fd_1, fd_3\}$ where fd_3 is Salary → Tax_bracket, and suppose the value-based policy ν_{fd_1} states that, in case of inconsistency, the highest value for attribute Salary should be preferred, while ν_{fd_3} states that the lowest value for attribute Tax_bracket should be preferred. In this case, depending on which order we choose, the result of applying the multi-dependency policy will be: $\{(John, 80K, 15), (Mary, 90K, 30)\}$ (for $\langle fd_1, fd_3 \rangle$), and $\{(John, 80K, 15), (John, 80K, 20), (Mary, 90K, 30)\}$ (for $\langle fd_3, fd_1 \rangle$).

It is clear that the order in which violations of FDs get resolved plays an important role in determining the semantics of our system. One semantics assumes that the user or the system administrator somehow chooses a fixed total ordering rather than a partial ordering. This then leads to the semantics specified in Definition 9. However, a natural question is whether we should say that a tuple is in the answer if it is present in the answer *irrespective* of which order is chosen. This is what we call the *Core* semantics below, and is analogous to cautious reasoning.

Definition 10 *Given a relation R, a set of functional dependencies $\mathcal{F}$ over R, and a partial ordering $\leq_{\mathcal{F}}$ on $\mathcal{F}$, the result of applying a policy under the* core semantics *is the set $Core(R, \mathcal{F}, \leq_{\mathcal{F}}) = \bigcap \{\mu^o_{\mathcal{F}}(R) \mid o \in TOT_{\leq_{\mathcal{F}}}(\mathcal{F})\}$.*

Intuitively, the *Core* semantics looks at all total orderings compatible with the associated partial ordering on $\mathcal{F}$. If every such total ordering causes a tuple to be in the result (according to Definition 9), then the tuple is returned in the answer. Of course, one may also be interested in the following analogous "Possibility" problem.

Problem 1 (Possibility Problem) *Given a relation R, a tuple $t \in R$, a set of functional dependencies $\mathcal{F}$ over R, and a partial ordering $\leq_{\mathcal{F}}$, does there exist a total ordering $o \in TOT_{\leq_{\mathcal{F}}}(\mathcal{F})$ such that $t \in \mu^o_{\mathcal{F}}(R)$?*

We now state three complexity results.

Theorem 1 *Given a relation R, a set of functional dependencies $\mathcal{F}$, a partial order $\leq_{\mathcal{F}}$ over $\mathcal{F}$, and a tuple $t \in R$:*

1. *Determining whether $t \in Core(R, \mathcal{F}, \leq_{\mathcal{F}})$ is coNP-complete.*
2. *Determining whether there is a total ordering $o \in TOT_{\leq_{\mathcal{F}}}(\mathcal{F})$ such that $t \in \mu^o_{\mathcal{F}}(R)$ is NP-complete.*
3. *If the arity of R is bounded, then the complexity of the problems (1) and (2) above is in PTIME.*

Proof *Statement 2.* (*Membership*) A polynomial size witness for this problem is a total ordering $o \in TOT_{\leq_{\mathcal{F}}}(\mathcal{F})$ such that $t \in \mu^o_{\mathcal{F}}(R)$. As any single FD policy can be computed in polynomial time, this witness can be verified in polynomial time by applying the policies one at a time, according to o, and finally checking whether $t \in \mu^o_{\mathcal{F}}(R)$.

(*Hardness*) We show a LOGSPACE reduction from 3SAT (Papadimitriou 1994). An instance of 3SAT is a pair $\langle U, \Phi \rangle$, where $U = \{p_1, p_2, \ldots, p_k\}$ is a set of propositional variables and Φ is a propositional formula of the form $C_1 \wedge \ldots \wedge C_n$ defined over U. Specifically, each C_i (with $1 \leq i \leq n$) is a clause containing exactly three (possibly negated) propositional variables in U.

We show how Φ can be encoded by an instance $\langle R, \mathcal{F}, \leq_{\mathcal{F}}, t' \rangle$ of our problem. Let S be the relational schema $S(A_1, B_1, V_1, \ldots, A_k, B_k, V_k, C, D, E)$ and $\mathcal{F}$ be the set of FDs $\{fd_{A,j} : A_j \rightarrow V_j, fd_{B,j} : B_j \rightarrow V_j \mid j \in [1..k]\} \cup \{fd_C : C \rightarrow D, fd_D : D \rightarrow E\}$. Consider the following tuple-based total policies associated with the FDs in $\mathcal{F}$: $\gamma_{fd_{A,j}}$ stating *choose the highest value of V_j*, $\gamma_{fd_{B,j}}$ stating *choose the lowest value of V_j* (with $j \in [1..k]$), γ_{fd_C} stating *delete the whole set of inconsistent tuples*, and γ_{fd_D} stating *delete the whole set of inconsistent tuples*. Assume that $\leq_{\mathcal{F}}$ states that $\forall j \in [1..k-1]$ and $Y \in \{A, B\}$, $fd_{Y,j} < fd_{Y,j+1}$ and $fd_{Y,k} < fd_C$, and $fd_C < fd_D$.

Let R be an instance of S defined as follows. Initially R is empty. Then, $\forall p_j \in U$ and $\forall C_i \in \Phi$,

- if making p_j *true* makes C_i *true* we add to R the tuple t such that $t[A_j] = t[B_j] = p_j$, $t[V_j] = 1$, $t[C] = C_i$, $t[D] = t[E] = 1$, and $\forall X \notin Attr(S) \setminus \{A_j, B_j, V_j, C, D, E\}, t[X] = k_1$ where k_1 is a new symbol;
- if making p_j *false* makes C_i *true* we add to R the tuple t such that $t[A_j] = t[B_j] = p_j$, $t[V_j] = 0$, $t[C] = C_i$, $t[D] = t[E] = 1$, and $\forall X \in Attr(S) \setminus \{A_j, B_j, V_j, C, D, E\}, t[X] = k_1$.

Moreover, $\forall C_i \in \Phi$ we add to R the tuple t such that $t[C] = C_i$, $t[D] = t[E] = 2$, and $\forall X \in Attr(S) \setminus \{C, D, E\}$, $t[X] = k_2$ where k_2 is new symbol. Finally, R also contains the tuple t' such that $t'[D] = 2$, $t'[E] = 3$ and $\forall X \in Attr(S) \setminus \{D, E\}$, $t'[X] = k_3$ where k_3 is new symbol.

We now prove that Φ is satisfiable iff there is a total ordering $o \in TOT_{\leq_{\mathcal{F}}}(\mathcal{F})$ such that $t' \in \mu^o_{\mathcal{F}}(R)$.

($\Rightarrow$) Assume that Φ is satisfiable. Let $U' \subseteq U$ be the set of propositional variables made *true* by a satisfying assignment for Φ. The total ordering $o \in TOT_{\leq_{\mathcal{F}}}(\mathcal{F})$ such that $t' \in \mu^o_{\mathcal{F}}(R)$ is obtained as follows.

For each $p_j \in U'$, o requires that $fd_{A,j} < fd_{B,j}$; this means that for the tuples t such that $t[A_j] = t[B_j] = p_j$, the value $t[V_j] = 1$ is chosen by $\gamma_{fd_{A,j}}$, and that $\gamma_{fd_{B,j}}$ will not have any effect on R. For each $p_j \in U \setminus U'$, o requires that $fd_{B,j} < fd_{A,j}$; this means that for the tuples t such that $t[A_j] = t[B_j] = p_j$, the value $t[V_j] = 0$ is chosen by $\gamma_{fd_{B,j}}$, and that $\gamma_{fd_{A,j}}$ will not have any effect on R. Observe that this suffices to define a total ordering o according the partial

ordering $\leq_{\mathcal{F}}$ (since the ordering for the other FDs is already defined by $\leq_{\mathcal{F}}$).

Let R_1 be the relation resulting from the application of the policies associated with the FDs $fd_{A,j}$ and $fd_{B,j}$ (with $j \in [1..k]$) according the above-specified order. Observe that $\forall p_j \in U'$, $\pi_{V_j}(\sigma_{A_j=p_j}(R_1)) = \{1\}$, and $\forall p_j \in U \setminus U'$, $\pi_{V_j}(\sigma_{A_j=p_j}(R_1)) = \{0\}$.

It is easy to see that the fact that Φ is satisfiable entails that $\pi_C(R_1) = \{C_1, \ldots, C_n\}$. Moreover, since $\forall C_i \in \Phi$, the relation R_1 also contains a tuple t such that $t[C] = C_i$ and $t[D] = 2$, there are n clusters w.r.t. fd_C (one for each C_i). Thus, the result of applying γ_{fd_C} to R_1 is a relation R_2 where each of these cluster is deleted (according the policy defined by γ_{fd_C}). Hence, the only tuple which remains in R_2 is t'. Finally, the application of the last policy γ_{fd_D} does not have any effect (since there are no inconsistent tuples w.r.t. fd_D), and t' results in $\mu^o_{\mathcal{F}}(R)$.

($\Leftarrow$) Assume now that there is a total ordering $o \in TOT_{\leq_{\mathcal{F}}}(\mathcal{F})$ such that $t' \in \mu^o_{\mathcal{F}}(R)$. According to the partial ordering $\leq_{\mathcal{F}}$, γ_{fd_D} must be the last policy applied on the relation. Thus, after applying all the others policies in in $\mathcal{F}$ there is no cluster w.r.t. fd_D (otherwise γ_{fd_D} would have deleted the whole cluster). The fact that there are no conflicting tuples in $\mu^o_{\mathcal{F}}(R)$ w.r.t. fd_D entails that there is no tuple $t \in \mu^o_{\mathcal{F}}(R)$ such that $t'[D] = 2$ and $t'[E] \neq 3$. Therefore, all the tuples t such that $t[C] = C_i$ and $t[D] = t[E] = 2$ must have been deleted by γ_{fd_C}, and this can happen only if there was at least a cluster for each C_i. Hence, after applying the policies associated with the FDs $fd_{A,j}$ and $fd_{B,j}$ (with $j \in [1..k]$) according to o, the resulting relation contains, $\forall C_i \in \Phi$, a tuple t such that $t[C] = C_i$. Let R_1 be such a relation. We define an assignment satisfying Φ as follows. For each variable p_j, if $\pi_{V_j}(\sigma_{A_j=p_j}(R_1)) = \{1\}$ then p_j is made *true* by such a assignment, otherwise (i.e., $\pi_{V_j}(\sigma_{A_j=p_j}(R_1)) = \{0\}$) p_j is made *false*.

Statement 1. (*Membership*) A polynomial size witness for the complement of this problem is a total ordering $o \in TOT_{\leq_{\mathcal{F}}}(\mathcal{F})$ such that $t \notin \mu^o_{\mathcal{F}}(R)$. As any single FD policy can be computed in polynomial time, this witness can be verified in polynomial time by applying the policies one at a time, according to o, and finally checking whether $t \notin \mu^o_{\mathcal{F}}(R)$.

(*Hardness*) The complement of the problem of determining whether $t \in \mathit{Core}(R, \mathcal{F}, \leq_{\mathcal{F}})$ is the problem of deciding whether there is a total ordering $o \in TOT_{\leq_{\mathcal{F}}}(\mathcal{F})$ such that $t \notin \mu^o_{\mathcal{F}}(R)$. We show a LOGSPACE reduction from the Possibility problem to the complement of our problem.

Let $\langle R_1, \mathcal{F}_1, \leq_{\mathcal{F}_1}, t_1\rangle$ be an instance of the problem of deciding whether there is a total ordering $o_1 \in TOT_{\leq_{\mathcal{F}_1}}(\mathcal{F}_1)$ such that $t_1 \in \mu^{o_1}_{\mathcal{F}_1}(R_1)$. We define an instance $\langle R_2, \mathcal{F}_2, \leq_{\mathcal{F}_2}, t_2\rangle$ of our problem as follows.

Given the relational schema $S_1(A_1, \ldots, A_n)$ of R_1, we define the relational schema S_2 of R_2 as $S_2(A_1, \ldots, A_n, B, C)$. Let R_2 be initially empty. For each tuple $t \in R_1 \setminus \{t_1\}$ we add to R_2 the tuple t' such that $t'[X] = t[X]$ $\forall X \in Attr(S_1)$ and $t'[B] = t'[C] = k_1$, where k_1 is a new symbol. Moreover, we add to R_2 the following tuples:

- t^*_1 such that $\forall X \in Attr(S_1)$, $t^*_1[X] = t_1[X]$, and $t^*_1[B] = k_2$, where k_2 is a new symbol, and $t^*_1[C] = 0$.
- t_2 such that $\forall X \in Attr(S_1)$, $t_2[X] = k_3$, where k_3 is a new symbol, $t_2[B] = k_2$, and $t_2[C] = 1$.

Let $\mathcal{F}_2$ be $\mathcal{F}_1 \cup \{fd : B \rightarrow C\}$, γ_{fd} be a tuple-based total policy stating that *the lowest value of C must be chosen*, and $\leq_{\mathcal{F}_2}$ be the partial order consisting of the relations in $\leq_{\mathcal{F}_1}$ and $\forall fd' \in \mathcal{F}_1$, $fd' < fd$.

We now prove that there is $o_1 \in TOT_{\leq_{\mathcal{F}_1}}(\mathcal{F}_1)$ such that $t_1 \in \mu^{o_1}_{\mathcal{F}_1}(R_1)$ iff there is $o_2 \in TOT_{\leq_{\mathcal{F}_2}}(\mathcal{F}_2)$ such that $t_2 \notin \mu^{o_2}_{\mathcal{F}_2}(R_2)$.

($\Rightarrow$) Assume that there is a total ordering $o_1 \in TOT_{\leq_{\mathcal{F}_1}}(\mathcal{F}_1)$ such that $t_1 \in \mu^{o_1}_{\mathcal{F}_1}(R_1)$. We can define $o_2 \in TOT_{\leq_{\mathcal{F}_2}}(\mathcal{F}_2)$ such that $t_2 \notin \mu^{o_2}_{\mathcal{F}_2}(R_2)$ as follows: o_2 is equal to o_1 plus $fd' < fd$ where fd' is the last FD in o_1. The fact that $t_1 \in \mu^{o_1}_{\mathcal{F}_1}(R_1)$ implies that the tuple $t^*_1 \in R_2$ will be in $\mu^{o_1}_{\mathcal{F}_2}(R_2)$. Thus, as $t_2[B] = t^*_1[B]$ and $t_2[C] > t^*_1[C]$ the policy γ_{fd} deletes t_2 from R_2. Hence, $t_2 \notin \mu^{o_2}_{\mathcal{F}_2}(R_2)$.

($\Leftarrow$) Assume now that there is a total ordering $o_2 \in TOT_{\leq_{\mathcal{F}_2}}(\mathcal{F}_2)$ such that $t_2 \notin \mu^{o_2}_{\mathcal{F}_2}(R_2)$. As only γ_{fd} can delete t_2, this implies that before applying γ_{fd} the tuple t^*_1 was in the result of $\mu^{o_1}_{\mathcal{F}_2}(R_2)$ (where o_1 is equal to o_2 except the ordering relationships involving fd). Hence, $t_1 \in \mu^{o_1}_{\mathcal{F}_1}(R_1)$.

Statement 3. Assuming that the arity of R is bounded by a constant b, the cardinality of $\mathcal{F}$ is bounded by 2^b, and the number of possible ordering in $TOT_{\leq_{\mathcal{F}}}(\mathcal{F})$ is bounded by the factorial of 2^b, which is still a constant w.r.t. the cardinality of R. Thus, since any single FD policy can be computed in polynomial time, checking whether there is total ordering $o \in TOT_{\leq_{\mathcal{F}}}(\mathcal{F})$ such that $t \in \mu^o_{\mathcal{F}}(R)$ (or equivalently $t \notin \mu^o_{\mathcal{F}}(R)$) and determining whether $t \in \mathit{Core}(R, \mathcal{F}, \leq_{\mathcal{F}})$ are in PTIME. □

It should be noted that we assume that policies can be computed in polynomial time. We do not consider NP-hard policies such as, *i.e.*, among a set V of inconsistent (possibly negative) values choose a nonempty subset $V' \subset V$ such that $\sum_{v \in V'} v = 0$. F We do not specify a possible semantics which returns $\bigcup\{\mu^o_{\mathcal{F}}(R) \mid o \in TOT_{\leq_{\mathcal{F}}}(\mathcal{F})\}$, since this can yield a relation with sources of inconsistency that were not present before the application of the multi-dependency policy. In the following, we show an example of how such a situation can arise.

Example 2 *Consider the following relation R:*

	Name	*Salary*	*Tax_bracket*
t_1	*John*	70*K*	*15*
t_2	*John*	80*K*	*20*

Let fd_1 be Name $\rightarrow$ Salary, and fd_2 be Salary $\rightarrow$ Tax_bracket. Suppose we have two interval-based policies ξ_{fd_1} and ξ_{fd_2}, both stating that conflicting values must be replaced by their mean. Assuming that fd_1 and fd_2 are incomparable w.r.t. $\leq_{\mathcal{F}}$, then there are two possible total orders: $\langle fd_1, fd_2\rangle$ and $\langle fd_2, fd_1\rangle$. In the first case, the result of applying the corresponding multi-dependency policy is $R' = \{(John, 75K, 17.5)\}$, whereas in the second

case the result is $R'' = \{(John, 75K, 15), (John, 75K, 20)\}$. *It is easy to see that* $|culprits(R' \cup R'', \{fd_1, fd_2\})| > |culprits(R, \{fd_1, fd_2\})|$.

Extensions of Classical Relational Algebra Operators with Multi-Dependency Policies

In this section, we study the relationship between IMPs and classical relational algebra operators. We suggest adding to the relational algebra operators that combine the classical relational operators with our IMPs. Each relational operator can be augmented by an IMP by either applying IMP first and then applying the operator, or the other way around. The following definition formalizes this.

Definition 11 *Given two relations* R_1 *and* R_2*, a binary relational operator op, two sets* $\mathcal{F}_1, \mathcal{F}_2$ *of functional dependencies defined over* R_1 *and* R_2*, respectively, and two multi-dependency policies* $\mu_{\mathcal{F}_1}$ *and* $\mu_{\mathcal{F}_2}$*,*

1. *a* policy-first inconsistency management operator *(*policy-first *operator) is defined as the* 7*-ary operator* $\omega_{\textit{policy-first}}(R_1, R_2, op, \mathcal{F}_1, \mathcal{F}_2, \mu_{\mathcal{F}_1}, \mu_{\mathcal{F}_2}) = op(\mu_{\mathcal{F}_1}(R_1), \mu_{\mathcal{F}_2}(R_2))$*;*
2. *a* policy-last inconsistency management operator *(*policy-last *operator) is defined as the* 7*-ary operator* $\omega_{\textit{policy-last}}(R_1, R_2, op, \mathcal{F}_1, \mathcal{F}_2, \mu_{\mathcal{F}_1}, \mu_{\mathcal{F}_2}) = \mu_{\mathcal{F}_1 \cup \mathcal{F}_2}(op(R_1, R_2))$.

Observe that the use of a policy-last operator requires the union of the given sets of functional dependencies to be handled by a multi-dependency policy that takes into account both sets. The definition of policy-first and policy-last inconsistency management operators for unary relational operators is straightforward. In the rest of this section, we study these *policy-first* and *policy-last* operators for several relational algebra operators.

Policy-first vs. *Policy-last* Operators for Projection

In the case of a *policy-last* projection operator, the projection is applied first and then the policy. In order for the inconsistency management operation to make sense, the projected attributes must include all attributes from the left-hand side of the functional dependencies involved in the multi-dependency policy, and at least one of the attributes from the right-hand side of each dependency. This is necessary to ensure that there will still be enough data to be able to identify the inconsistent tuples regarding all functional dependencies involved in the policy.

Example 3 *Consider relation* R *in our Salary Example and the functional dependency fd* : $Name \rightarrow Salary$*. Suppose also that we have a tuple-based policy* τ_{fd} *for fd that states that in case of inconsistency the tuple with the highest salary must be preferred. Let us consider a projection* $\pi_{salary}(R)$*. If* τ_{fd} *is applied first to* R *then we obtain* $\{80K, 90K\}$*. Otherwise, if* $\pi_{salary}(R)$ *is applied first, we no longer have inconsistency w.r.t. fd (and therefore no clusters at all), thus the application of* τ_{fd} *has no effect and the result is* $\{70K, 80K, 90K\}$.

The situation of this example can arise in the presence of both value and interval-based IMPs as well. The following theorem provides necessary and sufficient conditions for which it does not matter if a policy is applied before or after the projection operator.

Theorem 2 *Let* R *be a relation and fd* : $A_1, \ldots, A_k \rightarrow B$ *be a functional dependency over* R*. Let* $X \subseteq Attr(R)$ *be a superset of* $\{A_1, \ldots, A_k, B\}$*. For each (tuple-based, value-based, or interval-based) IMP* γ_{fd} *it holds that is the case that* $\gamma_{fd}(\pi_X(R)) = \pi_X(\gamma_{fd}(R))$ *iff*

1. $\gamma_{fd}(\pi^m_X(R)) = \pi^m_X(\gamma_{fd}(R))$*, and*
2. $\gamma_{fd}(\pi^m_X(R)) = \gamma_{fd}(\pi_X(R))$*,*

where π^m *is the multi-set projection operator (*i.e.*, the projection operator that returns multi-sets rather than sets).*

Proof ($\Leftarrow$) From condition (1) and (2) we obtain that $\gamma_{fd}(\pi_X(R)) = \pi^m_X(\gamma_{fd}(R))$. Clearly the first term of this equality is a set, since the application of the policy γ_{fd} on the relation $\pi_X(R)$ is a relation. Hence, $\pi^m_X(\gamma_{fd}(R))$ is also a set. This implies that it is equal to $\pi_X(\gamma_{fd}(R))$.
($\Rightarrow$) Assuming that either condition (1) or (2) is false it is easy to see that $\gamma_{fd}(\pi_X(R)) \neq \pi_X(\gamma_{fd}(R))$. □

The former condition requires that the policy does not depend on the values of attributes not involved in the functional dependency with which the policy is associated. This property is valid for all the IMPs C1-C6 of the Introduction and, intuitively, for a large class of IMPs. The latter condition requires that the policy does not depend on duplicate values of an attribute. This property is valid for IMPs C1, C2, C5, and C6 shown in the Introduction and for a large class of IMPs like those stating *choose the values lower/greater than (or equal to) a constant*, or *choose all values except the lowest/hightest ones*, and so on. The following example shows what can happen when a policy depends on duplicate values.

Example 4 *Consider relation* R *shown in the Introduction and the functional dependency fd* : $Name \rightarrow Salary$*. Let* γ_{fd} *be the policy* C3*, and* X *be the set* $\{Name, Salary\}$*. Thus,* $\gamma_{fd}(\pi_X(R)) = \{(John, 75K), (Mary, 90K)\}$*, whereas* $\pi_X(\gamma_{fd}(R)) = \{(John, 73.33K), (Mary, 90K)\}$*. Hence none of these sets is subset of the other one.*

Policy-first vs. *Policy-last* Operators for Selection

We start by showing that, for unrestricted policies, the order in which the policy and the selection operator are applied makes a difference.

Example 5 *Consider the following relation* R *and the functional dependency fd* : $Name \rightarrow Salary$*.*

	Name	*Salary*
t_1	$John$	$70K$
t_2	$John$	$80K$
t_3	$John$	$90K$

Let γ_{fd} *be the tuple-based policy* "choose all values except the lowest one". *It is easy to see that* $\sigma_{Salary \geq 80K}(\gamma_{fd}(R)) = \{t_2, t_3\}$*, whereas* $\gamma_{fd}(\sigma_{Salary \geq 80K}(R)) = \{t_3\}$*. Now assume that the policy*

is "choose the lowest value" *and the selection condition is the same used above. We have* $\sigma_{Salary \geq 80K}(\gamma_{fd}(R)) = \emptyset$, *whereas* $\gamma_{fd}(\sigma_{Salary \geq 80K}(R)) = \{t_2\}$.

Thus, in general, neither $\sigma_C(\gamma_{fd}(R)) \not\subseteq \gamma_{fd}(\sigma_C(R))$ nor $\sigma_C(\gamma_{fd}(R)) \not\supseteq \gamma_{fd}(\sigma_C(R))$. Moreover, as a consequence of Proposition 2, this is valid also for the value-based and interval-based approaches. The following proposition identifies a kind of tuple-based policy for which the order is irrelevant.

Proposition 4 *Let R be a relation and fd be a functional dependency over R. Given a tuple-based MDIMP γ_{fd} and a selection condition C, it holds that $\gamma_{fd}(\sigma_C(R)) = \sigma_C(\gamma_{fd}(R))$ if γ_{fd} is equivalent to σ_{C_γ} for some selection condition C_γ.*

Policy-first vs. *Policy-last* Operators for Cartesian Product

In this section we discuss the use of the *policy-first* and *policy-last* strategies with the cartesian product operator.

The following definition identifies a class of (tuple-based, value-based, or interval-based) IMPs which yield the same result when applied to a multi-set S of tuples or on a multi-set which contains the same proportion of tuples w.r.t. S.

Definition 12 *An IMP γ is said to be* ratio-invariant *iff for any multi-set $S = \{t_1, t_2 \ldots, t_n\}$ of tuples and for any integer $k > 0$ it is the case that $\gamma(S) = \gamma(S')$, where S' is the multi-set $S' = \{t_1^i, t_2^i \ldots, t_n^i \mid t_j^i = t_j$ where $t_j \in S$ and $i \in [1..k]\}$*

Observe that all IMPs C1-C6 described in the Introduction are ratio-invariant. For instance, policy C1 is ratio-invariant because the minimum value of a multi-set is independent of the number of elements having the same value. The same happens with the weighted mean used by C4: the weighted mean of the values in $\{v_1, v_2 \ldots, v_n\}$ is the same of that of $\{v_1^i, v_2^i \ldots, v_n^i \mid v_j^i = v_j$ where $v_j \in S$ and $i \in [1..k]\}$ (if $w_j^i = w_j$). On the other hand, a policy stating that *"if there are at least K occurrences of a value V, then choose V, otherwise choose 0"* is not ratio-invariant.

The following theorem provides results regarding the use of (ratio-invariant) IMPs with the cartesian product operator.

Theorem 3 *Let R_1 and R_2 be relations, and fd_1 and fd_2 be functional dependencies over R_1 and R_2, respectively. For any pair of ratio-invariant (tuple-based, value-based, or interval-based) IMPs γ_{fd_1} and γ_{fd_2}, it is the case that*

1. $\gamma_{fd_1}(R_1 \times R_2) = \gamma_{fd_1}(R_1) \times R_2$
2. $\gamma_{fd_2}(R_1 \times R_2) = R_1 \times \gamma_{fd_2}(R_2)$
3. $\gamma_{fd_1}(\gamma_{fd_2}(R_1 \times R_2)) = \gamma_{fd_1}(R_1) \times \gamma_{fd_2}(R_2)$
4. $\gamma_{fd_1}(\gamma_{fd_2}(R_1 \times R_2)) = \gamma_{fd_2}(\gamma_{fd_1}(R_1 \times R_2))$.

Proof We assume that the schemas of R_1 and R_2 do not have attributes in common, thus the schema of $R_1 \times R_2$ is the union of the schemas of R_1 and R_2. Moreover, with a little abuse of notation, we allow the application of γ_{fd_1} (resp. γ_{fd_2}) to $R_1 \times R_2$, even if they are defined for R_1 (resp. R_2).

We now prove the first case, the second case can be proved by analogous reasoning. Since fd_1 works only on the attributes of R_1 and is ratio-invariant, it holds that $\gamma_{fd_1}(\pi^m_{Attr(R_1)}(R_1 \times R_2)) = \gamma_{fd_1}(R_1^M) = \gamma_{fd_1}(R_1)$, where π^m is the multi-set projection operator, and R_1^M is the multi-set relation obtained by copying $M = |R_2|$ times every tuple of R_1. As $R_1 \times R_2 = \{t_1.t_2 \mid t_1 \in R_1 \wedge t_2 \in R_2\}$, it is easy that $\gamma_{fd_1}(R_1 \times R_2)$ is equivalent to is the set $\{t_1'.t_2 \mid t_1' \in \gamma_{fd_1}(R_1) \wedge t_2 \in R_2\}$, which is $\gamma_{fd_1}(R_1) \times R_2$.

As to the third case, since $\gamma_{fd_2}(R_1 \times R_2) = R_1 \times \gamma_{fd_2}(R_2)$, as a consequence of the first statement we obtain $\gamma_{fd_1}(R_1 \times \gamma_{fd_2}(R_2)) = \gamma_{fd_1}(R_1) \times \gamma_{fd_2}(R_2)$.

The result of the fourth case follows from the fact that $\gamma_{fd_1}(\gamma_{fd_2}(R_1 \times R_2)) = \gamma_{fd_1}(R_1) \times \gamma_{fd_2}(R_2)$ and $\gamma_{fd_2}(\gamma_{fd_1}(R_1 \times R_2)) = \gamma_{fd_1}(R_1) \times \gamma_{fd_2}(R_2)$. □

Policy-first vs. *Policy-last* Operators for Union

In this section we discuss the use of the *policy-first* and *policy-last* strategies with the union operator. We consider the case when the union is performed between two relations having the same schema and the functional dependency is defined over this schema. We start by showing a case where different results are obtained when the policy is applied before or after the union.

Example 6 *Consider the relational schema $S(Name, Salary)$ and the functional dependency $fd : Name \rightarrow Salary$. Assume that relation R_1 is $\{t_1 = (John, 75K), t_2 = (John, 80K)\}$ and R_2 is $\{t_3 = (John, 70K), t_4 = (John, 85K), t_5 = (Mary, 90K)\}$. Let γ_{fd} be the tuple-based policy* "choose the highest value of Salary". *It is easy to see that $\gamma_{fd}(R_1) = \{t_2\}$, $\gamma_{fd}(R_2) = \{t_4, t_5\}$, and $\gamma_{fd}(R_1 \cup R_2) = \{t_4, t_5\}$. Hence, $\gamma_{fd}(R_1 \cup R_2) \not\supseteq \gamma_{fd}(R_1) \cup \gamma_{fd}(R_2)$. Now assume that the policy is* "choose all the values except the lowest one". *We obtain $\gamma_{fd}(R_1) = \{t_2\}$, $\gamma_{fd}(R_2) = \{t_4, t_5\}$, but in this case $\gamma_{fd}(R_1 \cup R_2) = \{t_1, t_2, t_4, t_5\}$. Hence, $\gamma_{fd}(R_1 \cup R_2) \not\subseteq \gamma_{fd}(R_1) \cup \gamma_{fd}(R_2)$.*

Similar cases can be identified for value-based and interval-based policies. The following theorem presents necessary and sufficient conditions for which it does not matter if a policy is applied before or after the union.

Theorem 4 *Let R_1 and R_2 be relations over the relational schema S, and $fd : X \rightarrow B$ be a functional dependency where $X \subseteq Attr(S)$. For any (tuple-based, value-based, or interval-based) IMPs γ_{fd} it holds that $\gamma_{fd}(R_1 \cup R_2) = \gamma_{fd}(R_1) \cup \gamma_{fd}(R_2)$ iff $\pi_X(R_1) \cap \pi_X(R_2) = \emptyset$.*

Proof ($\Leftarrow$) If $\pi_X(R_1) \cap \pi_X(R_2) = \emptyset$ then $clusters(R_1, \{fd\}) \cap clusters(R_2, \{fd\}) = \emptyset$, and $clusters(R_1 \cup R_2, \{fd\}) = clusters(R_1, \{fd\}) \cup clusters(R_2, \{fd\})$. Thus, the application of γ_{fd} to $R_1 \cup R_2$ yields the same result as when it is applied first on the portion of $R_1 \cup R_2$ corresponding with R_1 and then on the portion corresponding to R_2.

($\Rightarrow$) Reasoning by contradiction, assume that $\pi_X(R_1) \cap \pi_X(R_2) \neq \emptyset$. It is easy to show that there is a policy γ such that $\gamma_{fd}(R_1 \cup R_2) \neq \gamma_{fd}(R_1) \cup \gamma_{fd}(R_2)$. □

Related Work

There has been a tremendous amount of work in inconsistency management since the 60s and 70s when paraconsistent logics where introduced (da Costa 1974; Blair and Subrahmanian 1989) and logics of inconsistency (Belnap 1977; Grant 1978) were developed. The four valued logic in (Belnap 1977) was used for handling inconsistency in logic programming (Blair and Subrahmanian 1989) and extended to the case of bilattices (Fitting 1991). Subsequently, frameworks such as default logic (Reiter 1980), maximal consistent subsets (Baral, Kraus, and Minker 1991), inheritance networks (Touretzky 1986), and others were used to generate multiple plausible consistent scenarios (often called "extensions"), and methods to draw inferences were developed that looked at truth in all (or some) extensions. Kifer and Lozinskii (Kifer and Lozinskii 1992) extended annotated logics of inconsistency developed by Blair and Subrahmanian (Blair and Subrahmanian 1989) to handle a full first order case, while (Thirunarayan and Kifer 1993) developed similar extensions to handle inheritance networks. Argumentation methods (Amgoud and Cayrol 2002) were used to reason about how certain arguments defeated others.

Methods to clean data and/or provide consistent query answers in the presence of inconsistent data are also quite common (Jermyn, Dixon, and Read 1999; Schallehn and Sattler 2003; Chomicki 2007; Bohannon et al. 2005). For instance, (Chomicki 2007) addresses the basic concepts and results of the area of consistent query answering (in the standard model-theoretic sense). They consider universal and binary integrity constraints, denial constraints, functional dependencies, and referential integrity constraints. (Bohannon et al. 2005) presents a cost-based framework that allows finding "good" repairs for databases that exhibit inconsistencies in the form of violations to either functional or inclusion dependencies. They propose heuristic approaches to constructing repairs based on equivalence classes of attribute values; the algorithms presented are based on greedy selection of least repair cost, and a number of performance optimizations are also explored. The technique based on query rewriting introduced in (Arenas, Bertossi, and Chomicki 1999) for quantifier-free conjunctive queries and binary universal constraints was extended in (Fuxman and Miller 2005; Fuxman, Fazli, and Miller 2005) to work for a subclass of conjunctive queries in the presence of key constraints. The complexity of the consistent query answer problem was investigated in (Calì, Lembo, and Rosati 2003) in the presence of both functional and inclusion dependencies, and further studied in (Chomicki and Marcinkowski 2005) in the presence of denial constraints and inclusion dependencies. The notion of consistent answer was extended to the case of aggregate queries in (Arenas et al. 2003), where the evaluation of consistent answers of aggregate queries was investigated in the presence of functional dependencies.

The logic-based frameworks in (Arenas, Bertossi, and Chomicki 2003; Greco, Greco, and Zumpano 2003; Barceló and Bertossi 2003) assume that tuple insertions and deletions are the basic primitives for repairing inconsistent data. Repairs also consisting of value-update operations were considered in (Franconi et al. 2001; Bertossi et al. 2005; Wijsen 2003; 2005; Bohannon et al. 2005; Flesca, Furfaro, and Parisi 2005; 2007). In particular, (Wijsen 2003; 2005; Flesca, Furfaro, and Parisi 2005) investigated the complexity of the consistent query answering problem when the basic primitive for repairing data is the attribute-value update, whereas (Franconi et al. 2001; Bertossi et al. 2005; Bohannon et al. 2005; Flesca, Furfaro, and Parisi 2007) focused on the problem of computing repairs rather than computing consistent answers.

There are also several important works (Lozinskii 1994; Hunter and Konieczny 2005; Grant and Hunter 2006) on measuring the amount of inconsistency in a database or knowledge base.

Conclusions

None of the past approaches to inconsistency management is capable of handling cases C3, C4, C5, C6 introduced in the Introduction because past approaches adhere to three important tenets: first, that no "new" data should be introduced into the database; second, that as much of the original data as possible should be retained and third, that consistency must be restored.

Though we agree these are desirable goals, the fact remains that users in specific application domains often know a lot more about the intricacies of *their* data than a database designer who has never seen the data. In many of these cases, the end-user wants to resolve inconsistencies by taking *his knowledge of the data into account.* Tools for managing inconsistent data today do not support such users.

For example, there are many cases where end-users might actually want to introduce "seemingly new" data – in case C3 and C4, the user wants to take an average or weighted average of salaries. This may be what the user or his company determines is appropriate for his application domain. Should he be stopped from doing this by database designers who do not know the application *a priori*? No.

Likewise, consider case C6. When conducting a scientific experiment (biological, atmospheric etc.), inconsistent data might be collected for any number of reasons (faulty measurements, incorrectly mixed chemicals, environmental factors). Should the results of the experiments be based on *dirty* data? Some scientists at least would argue "No" (perhaps for some experiments) and eliminate the dirty data and repeat all, or parts, of the experiment. Databases should provide support for decisions users want to make.

In response to these needs, we introduce the concept of *inconsistency management policies* in this paper as functions satisfying a *minimal* set of axioms. We propose several IMPs that satisfy these axioms, and study relations between them in the simplified case where only one functional dependency is present. We show that when multiple FDs are present, multiple alternative semantics can result. We prove results on the complexity of some of these semantics and related problems. Fortunately, as long as we assume that relations have schemas of bounded size, these problems end up in PTIME. We introduce new versions of the relational algebra that are augmented by inconsistency management policies which are applied either before the operator or after. We develop theoretical results on the resulting extended relational

operators that could, in principle, be used in the future for query optimization.

Future work will focus on several problems. In this paper, we have provided a theoretical definition of an IMP, but not given a policy specification language – such a language is needed. The equivalence theorems in the paper could form the basis for query optimization in inconsistent DBs, but such query optimizers need to be developed. Cost models for our policy-first and policy-last operations need to be developed to build query plans.

Acknowledgments

The authors gratefully acknowledge funding support for this work provided by the AFOSR through the Laboratory for Computational Cultural Dynamics (LCCD) under grants FA95500610405 and FA95500510298, the ARO under grant DAAD190310202, and the NSF under grant 0540216. Any opinions, findings or recommendations in this document are those of the authors and do not necessarily reflect the views of sponsors.

References

Amgoud, L., and Cayrol, C. 2002. A reasoning model based on the production of acceptable arguments. *Ann. of Math. and Artif. Intel.* 34(1):197–215.

Arenas, M.; Bertossi, L. E.; and Chomicki, J. 1999. Consistent query answers in inconsistent databases. In *PODS*, 68–79.

Arenas, M.; Bertossi, L. E.; and Chomicki, J. 2003. Answer sets for consistent query answering in inconsistent databases. *TPLP* 3(4-5):393–424.

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

Baral, C.; Kraus, S.; and Minker, J. 1991. Combining multiple knowledge bases. *IEEE TKDE* 3(2):208–220.

Barceló, P., and Bertossi, L. E. 2003. Logic programs for querying inconsistent databases. In *PADL*, 208–222.

Belnap, N. 1977. A useful four valued logic. *Modern Uses of Many Valued Logic* 8–37.

Benferhat, S.; Dubois, D.; and Prade, H. 1997. Some syntactic approaches to the handling of inconsistent knowledge bases: A comparative study part 1: The flat case. *Studia Logica* 58(1):17–45.

Bertossi, L. E., and Chomicki, J. 2003. Query answering in inconsistent databases. In *Logics for Emerging Applications of Databases*, 43–83. Springer.

Bertossi, L. E.; Bravo, L.; Franconi, E.; and Lopatenko, A. 2005. Complexity and approximation of fixing numerical attributes in databases under integrity constraints. In *DBPL*, 262–278.

Besnard, P., and Schaub, T. 1998. Signed systems for paraconsistent reasoning. *J. Autom. Reas.* 20(1):191–213.

Blair, H. A., and Subrahmanian, V. S. 1989. Paraconsistent logic programming. *Theor. Comp. Sci.* 68(2):135–154.

Bohannon, P.; Fan, W.; Flaster, M.; and Rastogi, R. 2005. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*, 143–154.

Calì, A.; Lembo, D.; and Rosati, R. 2003. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *PODS*, 260–271.

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

Chomicki, J. 2007. Consistent query answering: Five easy pieces. In *ICDT*, 1–17.

da Costa, N. 1974. On the theory of inconsistent formal systems. *Notre Dame J. of Formal Logic* 15(4):497–510.

Fitting, M. 1991. Bilattices and the semantics of logic programming. *J. of Logic Progr.* 11(1-2):91–116.

Flesca, S.; Furfaro, F.; and Parisi, F. 2005. Consistent query answers on numerical databases under aggregate constraints. In *DBPL*, 279–294.

Flesca, S.; Furfaro, F.; and Parisi, F. 2007. Preferred database repairs under aggregate constraints. In *SUM*, 215–229.

Franconi, E.; Palma, A. L.; Leone, N.; Perri, S.; and Scarcello, F. 2001. Census data repair: a challenging application of disjunctive logic programming. In *LPAR*, 561–578.

Fuxman, A., and Miller, R. J. 2005. First-order query rewriting for inconsistent databases. In *ICDT*, 337–351.

Fuxman, A.; Fazli, E.; and Miller, R. J. 2005. Conquer: Efficient management of inconsistent databases. In *SIGMOD*, 155–166.

Grant, J., and Hunter, A. 2006. Measuring inconsistency in knowledgebases. *J. of Intel. Inf. Syst.* 27(2):159–184.

Grant, J., and Hunter, A. 2008. Analysing inconsistent first-order knowledge bases. *Artif. Intel.* 172(8-9):1064–1093.

Grant, J. 1978. Classifications for inconsistent theories. *Notre Dame J. of Formal Logic* 19(3):435–444.

Greco, G.; Greco, S.; and Zumpano, E. 2003. A logical framework for querying and repairing inconsistent databases. *IEEE TKDE* 15(6):1389–1408.

Hunter, A., and Konieczny, S. 2005. Approaches to measuring inconsistent information. In *Inconsistency Tolerance*, 191–236.

Jermyn, P.; Dixon, M.; and Read, B. J. 1999. Preparing clean views of data for data mining. In *ERCIM Work. on Database Res.*, 1–15.

Kifer, M., and Lozinskii, E. L. 1992. A logic for reasoning with inconsistency. *J. of Autom. Reas.* 9(2):179–215.

Lozinskii, E. L. 1994. Resolving contradictions: A plausible semantics for inconsistent systems. *J. of Autom. Reas.* 12(1):1–31.

Papadimitriou, C. M. 1994. *Computational complexity*. Addison-Wesley.

Reiter, R. 1980. A logic for default reasoning. *Artif. Intel.* 13(1-2):81–132.

Schallehn, E., and Sattler, K. 2003. Using similarity-based operations for resolving data-level conflicts. In *BNCOD*, volume 2712, 172–189.

Subrahmanian, V. S., and Amgoud, L. 2007. A general framework for reasoning about inconsistency. In *IJCAI*, 599–504.

Thirunarayan, K., and Kifer, M. 1993. A theory of non-monotonic inheritance based on annotated logic. *Artif. Intel.* 60(1):23–50.

Touretzky, D. 1986. *The mathematics of inheritance systems*. Morgan Kaufmann.

Ullman, J. D. 1988. *Principles of Database and Knowledge-Base Systems, Volume I*. Computer Science Press.

Wijsen, J. 2003. Condensed representation of database repairs for consistent query answering. In *ICDT*, 378–393.

Wijsen, J. 2005. Database repairing using updates. *ACM TODS* 30(3):722–768.