Preferred Repairs for Inconsistent Databases*

S. Greco, C. Sirangelo, I. Trubitsyna and E. Zumpano
DEIS – Università della Calabria
87030 Rende, Italy
{greco, sirangelo, irina, zumpano}@si.deis.unical.it

Abstract

The objective of this paper is to investigate the problems related to the extensional integration of information sources. In particular, we propose an approach for managing inconsistent databases, i.e. databases violating integrity constraints. The presence of inconsistent data can be resolved by "repairing" the database, i.e. by providing a computational mechanism that ensures obtaining consistent "scenarios" of the information or by consistently answer to queries posed on an inconsistent set of data. In this paper we consider preferences among repairs and possible answers by introducing a partial order among them on the base of some preference criteria. More specifically, preferences are expressed by considering polynomial functions applied to repairs and returning real numbers. The goodness of a repair is measured by estimating how much it violates the desiderata conditions and a repair is preferred if it minimizes the value of the polynomial function used to express the preference criteria. The main contribution of this work consists in the proposal of a logic approach for querying and repairing inconsistent databases that extends previous works by aallowing to express and manage preference criteria. The approach here proposed allows to express reliability on the information sources and is also suitable for expressing decision and optimization problems. The introduction of preference criteria strongly reduces the number of feasible repairs and answers; for special classes of constraints and functions it gives a unique repair and answer.

*Work partially supported by a MURST grants under the projects "D2I" and "Sistemi informatici Integrati a supporto di bench-marking di progetti ed interventi ad innovazione tecnologica in campo agro-alimentare". The first author is also supported by ICAR-CNR.

1 Introduction

The problem of dealing with inconsistent information has recently assumed additional relevance as it plays a key role in all the areas in which duplicate information or conflicting information is likely to occur [2, 6, 7, 16, 18, 23, 27]. In this paper we address the problem of managing inconsistent databases, i.e. databases violating integrity constraints. The fact that a database may be inconsistent with respect to a set of integrity constraints is not surprising if it is obtained as the result of an integration process over a set of knowledge bases.
The following example shows a typical case of inconsistency.

Example 1 Consider the database consisting of the three binary relations *Teaches(Professor, Course), Faculty(Professor, Faculty)* and *Course(Course, Faculty)* with the integrity constraint

$$\forall(P,C,F)[\,Teaches(P,C) \wedge Faculty(P,F) \supset Course(C,F)\,]$$

stating that if a professor P teaches a course C and P is in the faculty F, then the course C must belong to the faculty F. Assume there are two different sources of the databases: $D_1 = \{Teaches(t_1,c_1)$, $Teaches(t_2,c_2)$, $Faculty(t_1,f_1)$, $Course(c_1,f_1)\}$ and $D_2 = \{Teaches(t_1,c_1), Faculty(t_2,f_1), Course(c_2,f_2)\}$.
The two instances satisfy the constraint, but from their union we derive a relation which does not satisfy the constraint. □

The presence of inconsistent data can be resolved by "repairing" the database, i.e. by providing a computational mechanism that ensures to obtain consistent "scenarios" of the information (repairs) in an inconsistent environment or to consistently answer to queries posed on an inconsistent set of data. Informally, a repair for a possibly inconsistent database is a minimal set of insert and delete operations which makes the database consistent, whereas a consistent answer is a set of tuples derived from the database, satisfying all integrity constraints [3, 4, 12, 13, 25].

Thus the integration of, possibly inconsistent, databases must consider the possibility of constructing an integrated consistent database by replacing inconsistent tuples. For instance, for the integrated relation of the above example, it is possible to obtain a consistent database by i) deleting the tuple $Teaches(t_2, c_2)$, ii) deleting the tuple $Faculty(t_2, f_1)$, or iii) adding the tuple $Course(c_2, f_1)$. These three update operations are repairs that make the database consistent, but one should prefer a repair with respect to an alternative one. For instance, one could prefer a repair which minimize the number of deletion and insertion of tuples in the relation $Teaches$ and, in such a case, the second and third repair are preferred to the first one, or one should prefer repairs minimizing the set of deletions and in such a case the third repair is preferred to the first two repairs.

Example 2 Consider the relation $Teaches$ ($Name$, $Faculty$, $Course$) with the functional dependency $Name \rightarrow Faculty$. Assume to have three different sources for the relation $Teaches$ containing, respectively, the tuples $Teaches(john,\ science,\ databases)$, $Teaches(john,\ engineering,\ algorithms)$ and $Teaches(john,\ science,\ operating\_systems)$. The three different source relations satisfy the functional dependencies, but from their integration we derive the inconsistent relation $D = \{(john,\ engineering,\ algorithms),$ $(john,\ science,\ operating\_systems), (john,\ science,$ $databases)\}$.

The integrated relation can be repaired by applying a minimal set of update operations. In particular it admits two repairs: R_1 obtained by deleting the tuple $(john,$ $engineering,\ algorithms)$ and R_2 obtained by deleting the two tuples $(john,\ science,\ databases)$ and $(john,\ science,\ operating\_systems)$. $\Box$

In the presence of an alternative set of repairs one could express preferences on them, for instance, if, in the above example, preferred repairs are those minimizing the number of deletion and insertion of tuples then the repair R_1 preferred to the repair R_2.

In this paper we introduce a flexible mechanism that allows specifying preference criteria so that selecting among a set of feasible repairs the preferable ones, i.e. those better conforming to the specified criteria. Preference criteria introduce desiderata on how to update the inconsistent database in order to make it consistent and are expressed by means of a polynomial function, named *evaluation function*. The evaluation function "measures" the repairs with respect to the database and allows defining a partial order both among repairs and possible answers, so that allowing the selection of the feasible repairs which result preferred w.r.t. the evaluation function: the preferred repairs. Informally a preferred repair is a repair that better satisfies preferences, i.e. it minimizes the value of the evaluation function.

In the integration of two conflicting databases a simple way to remove inconsistencies consists in the definition of preference criteria such as a partial order on the source information, or in the use of the majority criteria [19], which in the presence of conflicts gives more credit to the information present in the majority of the knowledge bases.

Observe that the selection of the element which occurs a maximum number of times in the integrated knowledge bases is easily obtained by specializing the evaluation function to compute the cardinality of the deleted atoms, since a consistent scenario, obtained by performing the minimum number of delete operation, surely maintains the majority of the information which overlaps among the knowledge bases.

The application of the majority criteria [19] in the integration phase of the above example, eliminates the tuple $(john, engineering, algorithms)$; this corresponds to give preference to the repair R_1 with respect to the repair R_2, i.e. we express preference for the repaired databases which need a lesser number of updates to be consistent or equivalently we give credit to the information which occurs a greater number of times.

Note that, while integrity constraints can be considered as a query which must always be true after a modification of the database, the conditions expressed by the evaluation function can be considered as a set of desiderata which are satisfied if possible by a generic repair. The goodness of a repair is measured by estimating how much the updates to be performed on the inconsistent database respect the preference criteria or in other words how much the repaired database violates them.

The main contribution of this work consists in the definition of a logic approach for querying and repairing inconsistent databases that extends previous works by also considering techniques to express and manage preference criteria. The approach here proposed allows to express reliability on the information sources and moreover is also suitable for expressing decision and optimization problems. Obviously the introduction of preference criteria reduces the number of feasible repairs and answers, moreover for special cases of constraints it gives a unique repair and answer.

The rest of the paper is organized as follows. In Section 2 we present basic definitions on logic languages (Datalog, disjunctive Datalog and classical negation). In Section 3 we recall standard definitions on repairs and consistent answers and introduce preference criteria on repairs and answers. In Section 4 we present the rewriting of integrity constraints into disjunctive rules and show how preferred repairs and answers are computed. Finally, in Section 5, we present our conclusions.

2 Basic Definitions and Datalog Extensions

In this section we introduce preliminaries on deductive databases and integrity constraints. Integrity constraints define restrictions on the instance of relational databases. Deductive databases are defined by logical rules which are used to derive new knowledge starting from a given (relational) database. We present first the language Datalog and next two extensions: Disjunctive Datalog and Datalog with aggregates.

2.1 Datalog

We assume the existence of finite domains of constants, variables and predicate symbols. A term is either a variable or a constant. An atom is of the form $p(t_1, ..., t_n)$ where p is a predicate symbol and $t_1, ..., t_n$ are terms. A literal is either an atom A or its negation $not\ A$. A *Datalog program* (or, simply, a *program*) P is a finite set of rules. Each *rule* of P has the form $A \leftarrow A_1, ..., A_m$, where A is an atom (the *head* of the rule) and $A_1, ..., A_m$ are literals (the *body* of the rule). A ground rule with an empty body is called a *fact*.

Given a Datalog program P, the Herbrand universe for P, denoted H_P, is the set of all constants occurring in P; the Herbrand Base of P, denoted B_P, is the set of all possible ground atoms whose predicate symbols occur in P and whose arguments are elements from the Herbrand universe. A *ground instance* of a rule r in P is a rule obtained from r by replacing every variable X in r by a ground term in H_P. The set of ground instances of r is denoted by $ground(r)$; accordingly, $ground(P)$ denotes $\bigcup_{r \in P} ground(r)$. An interpretation I of P is a subset of B_P. A ground positive literal A (resp. negative literal $\neg A$) is true w.r.t. an interpretation I if $A \in I$ (resp. $A \notin I$). A conjunction of literals is true in an interpretation I if all literals are true in I. A ground rule is true in I if either the body conjunction is false or the head is true in I. A *(Herbrand) model* M of P is an interpretation that makes each ground instance of each rule in P $true$. A model M for P is minimal if there is no model N for P such that $N \subset M$.

Let I be an interpretation for a program P. The *immediate consequence operator* $T_P(I)$ is defined as the set containing the heads of each rule $r \in ground(P)$ s.t. the body of r is true in I. The semantics of a *positive* (i.e. negation-free) logic program P is given by the unique minimal model; this minimum model coincides with the least fixpoint $T_P^\infty(\emptyset)$ of T_P [20]. Generally, the semantics of logic programs with negation can be given in terms of total stable model semantics [9] which we now briefly recall.

Given a program P and an interpretation M, M is a (*total*) *stable model* of P if it is the minimum model of the positive program P^M defined as follows: P^M is obtained from $ground(P)$ by (i) deleting all rules which have some negative literal $\neg b$ in their body with $b \in M$, and (ii) removing all negative literals in the remaining rules. Logic programs may have zero, one or several stable models. Positive programs have a unique stable model which coincides with the minimum model [9].

Given a program P and two predicate symbols p and q, we write $p \rightarrow q$ if there exists a rule where q occurs in the head and p in the body or there exists a predicate s such that $p \rightarrow s$ and $s \rightarrow q$. A program is *stratified* if there exists no rule where a predicate p occurs in a negative literal in the body, q occurs in the head and $q \rightarrow p$ i.e. there is no recursion through negation [1]. Stratified programs have a unique stable model which coincides with the *stratified model*, obtained by partitioning the program into an ordered number of suitable subprograms (called 'strata') and computing the fixpoints of every stratum from the lowest one up [1].

Queries

Predicate symbols are partitioned into two distinct sets: *base predicates* (also called EDB predicates) and *derived predicates* (also called IDB predicates). Base predicates correspond to database relations defined over a given domain and they do not appear in the head of any rule whereas derived predicates are defined by means of rules. Given a database D, a predicate symbol r and a program $\mathcal{P}$, $D(r)$ denotes the set of r-tuples in D whereas $\mathcal{P}_D$ denotes the program derived from the union of $\mathcal{P}$ with the tuples in D, i.e. $\mathcal{P}_D = \mathcal{P} \cup \{r(t) \leftarrow \ \mid t \in D(r)\}$. In the following a tuple t of a relation r will also be denoted as a fact $r(t)$. The semantics of $\mathcal{P}_D$ is given by the set of its stable models by considering either their union (*possible semantics* or *brave reasoning*) or their intersection (*certain semantics* or *cautious reasoning*). A *query* Q is a pair $(g, \mathcal{P})$ where g is a predicate symbol, called the *query goal*, and $\mathcal{P}$ is a program. The answer to a query $Q = (g, \mathcal{P})$ over a database D, under the possible (resp. certain) semantics is given by $D'(g)$ where $D' = \bigcup_{M \in SM(\mathcal{P}_D)} M$ (resp. $D' = \bigcap_{M \in SM(\mathcal{P}_D)} M$).

2.2 Disjunctive Datalog

A *(disjunctive Datalog) rule* r is a clause of the form

$$A_1 \vee \cdots \vee A_k \leftarrow B_1, \ldots, B_m, not\ B_{m+1}, \ldots, not\ B_n$$

where $k + m + n > 0$ and $A_1, \ldots, A_k, B_1, \ldots, B_n$ are atoms. The disjunction $A_1 \vee \cdots \vee A_k$ is the *head* of r, while the conjunction $B_1, \ldots, B_m, not\ B_{m+1}, \ldots, not\ B_n$ is the *body* of r. We also assume the existence of the binary built-in predicate symbols (comparison operators) which can only be used in the body of rules.

The (model-theoretic) semantics for positive programs $\mathcal{P}$ assigns to $\mathcal{P}$ the set of its *minimal models* $\mathcal{MM}(\mathcal{P})$ [21]. The more general *disjunctive stable model semantics* also applies to programs with (unstratified) negation [10, 8]. Disjunctive stable model semantics generalizes stable model semantics, previously defined for normal programs [9]. For any interpretation M, denote with $\mathcal{P}^M$ the ground positive program derived from $ground(\mathcal{P})$ by 1) removing all rules that contain a negative literal $not\ a$ in the body and $a \in M$, and 2) removing all negative literals from the remaining rules. An interpretation M is a (disjunctive) stable model of $\mathcal{P}$ if and only if $M \in \mathcal{MM}(\mathcal{P}^M)$. For general $\mathcal{P}$, the stable model semantics assigns to $\mathcal{P}$ the set $\mathcal{SM}(\mathcal{P})$ of its *stable models*. It is well known that stable models are minimal models (i.e. $\mathcal{SM}(\mathcal{P}) \subseteq \mathcal{MM}(\mathcal{P})$) and that for negation free programs minimal and stable model semantics coincide (i.e. $\mathcal{SM}(\mathcal{P}) = \mathcal{MM}(\mathcal{P})$). Observe that stable models are minimal models which are 'supported', i.e. their atoms can be derived from the program.

Extended disjunctive databases

Extended Datalog programs extend standard Datalog programs with a different form of negation, known as *classical* or *strong negation*, which can also appear in the head of rules [10, 17, 11]. An extended atom is either an atom, say A or its negation $\neg A$. An extended Datalog program is a set of rules of the form

$$A_1 \vee \cdots \vee A_k \leftarrow B_1, ..., B_m, not\ B_{m+1}, ..., not\ B_n \qquad k + n > 0$$

where $A_1, ..., A_k, B_1, ..., B_n$ are extended atoms. A (2-valued) interpretation I for an extended program $\mathcal{P}$ is a pair $\langle T, F \rangle$ where T and F define a partition of $\mathcal{B}_\mathcal{P} \cup \neg\mathcal{B}_\mathcal{P}$ and $\neg\mathcal{B}_\mathcal{P} = \{\neg A | A \in \mathcal{B}_\mathcal{P}\}$. An interpretation $I = \langle T, F \rangle$ is *consistent* if there is no atom A such that $A \in T$ and $\neg A \in T$. The semantics of an extended program $\mathcal{P}$ is defined by considering each negated predicate symbol, say $\neg p$, as a new symbol syntactically different from p and by adding to the program, for each predicate symbol p with arity n the constraint $\leftarrow p(X_1, ..., X_n), \neg p(X_1, ..., X_n)$[1]. In the following, for the sake of simplicity, we shall also use rules whose bodies may contain disjunctions. Such rules, called generalized (extended) disjunctive rules, are used as shorthands for multiple standard disjunctive rules. More specifically, a generalized disjunctive rule of the form

$$A_1 \vee ... \vee A_k \leftarrow (B_{1,1} \vee ... \vee B_{1,m_1}), ..., (B_{n,1} \vee ... \vee B_{n,m_n})$$

denotes the set of standard rules

$$A_1 \vee \cdots \vee A_k \leftarrow B_{1,i_1}, ..., B_{n,i_n} \qquad \forall j, i : 1 \leq j \leq n\ and\ 1 \leq i_j \leq m_j$$

[1]A rule with empty head is a constraint is satisfied only if its body is false.

2.3 Integrity constraints

Database schemata contain the knowledge on the structure of data, i.e. they define constraints on the form the data must have. Integrity constraints express semantic information on data, that is relationships that must hold among data in the theory and they are mainly used to validate database manipulations. Integrity constraints represent the interaction among data and define properties which are supposed to be explicitly satisfied by all instances over a given database schema. They are usually defined by first order rules or by means of special notations for particular classes such as keys and functional dependencies.

Definition 1 *An* integrity constraint *(or* embedded dependency*) is a formula of the first order predicate calculus of the form:*

$$(\forall X)\ [\ \Phi(X) \supset (\exists Z)\Psi(Y)\ ]$$

where X, Y and Z are sets of variables, Φ and Ψ are two conjunctions of literals such that X and Y are the distinct set of variables appearing in Φ and Ψ respectively, $Z = X - Y$ is the set of variables existentially quantified. □

In the definition above, the conjunction Φ is called the *body* and the conjunction Ψ the *head* of the integrity constraint. The semantics of the above constraints is that for every value of X which makes the formula $\Phi(X)$ true there must be an instance of Z which makes $\Psi(Y)$ true. Most of the dependencies developed in database theory are restricted cases of some of the above classes [15].

In this paper we consider *full* (or *universal*) *single-head* constraints, where Ψ is a literal or a conjunction of built-in literals (i.e. comparison operators). Therefore, an integrity constraint is a formula of the form: $(\forall X)\ [\ B_1 \wedge \cdots \wedge B_n \wedge not\, A_1 \wedge \cdots \wedge not\, A_m \wedge \phi \supset A_0\ ]$ where $A_1, ..., A_m, B_1, ..., B_n$ are base positive literals, ϕ is a conjunction of built-in literals, A_0 is a base positive atom or a conjunction of built-in atoms, X denotes the list of all variables appearing in $B_1, ..., B_n$; moreover the variables appearing in $A_0, ..., A_m$ and ϕ, also appear in $B_1, ..., B_n$.

Often we shall write our constraints in a different format by moving literals from the head to the body and vice versa. For instance, the above constraint could be rewritten as $(\forall X)\ [\ B_1 \wedge \cdots \wedge B_n \wedge \phi \supset A_0 \vee \cdots \vee A_m\ ]$ or in the form of rule with empty head, called *denial*: $(\forall X)\ [\ B_1 \wedge \cdots \wedge B_n \wedge not\, A_0 \wedge \cdots \wedge not\, A_m \wedge \phi \supset\ ]$ which is satisfied only if the body is false. Moreover, in some case we shall write the above constraint as

$$(\forall X)\ [\ B_1 \wedge \cdots \wedge B_n \wedge not\, A_0 \wedge \cdots \wedge not\, A_m \supset not(\varphi)\ ]$$

Example 3 *The integrity constraint $(\forall X)\ [\ p(X) \supset q(X) \vee r(X)\ ]$ states that the relation p must be contained in the union of the relations q and r. It could be rewritten as $(\forall X)\ [\ p(X) \wedge not\, q(X) \supset r(X)\ ]$ or as $(\forall X)\ [\ p(X) \wedge not\, q(X) \wedge not\, r(X) \supset\ ]$.* □

3 Preferred repairs and answers

In this section we recall the formal definition of consistent database and repair; we present a computational mechanism that ensures selecting preferred repairs and preferred answers for inconsistent database.

3.1 Preferred repairs for inconsistent databases

In this section we introduce a polynomial function through which expressing preferences criteria. The function introduces a partial order among repairs, so that allowing the evaluation of the goodness of a repair for an inconsistent database. Moreover we define preferred repairs as feasible repairs that are minimal with respect to the partial order.
Let us first recall the formal definition of consistent database and repair.

Definition 2 Given a database D and a set of integrity constraint $\mathcal{IC}$ on D, we say that D is *consistent* if $D \models \mathcal{IC}$, i.e. if all integrity constraints in $\mathcal{IC}$ are satisfied by D, otherwise it is *inconsistent*. □

Definition 3 Given a (possibly inconsistent) database D, a *repair* for D is a pair of sets of atoms (R^+, R^-) such that 1) $R^+ \cap R^- = \emptyset$, 2) $D \cup R^+ - R^- \models \mathcal{IC}$ and 3) there is no pair $(S^+, S^-) \neq (R^+, R^-)$ such that $S^+ \subseteq R^+, S^- \subseteq R^-$ and $D \cup S^+ - S^- \models \mathcal{IC}$. The database $D \cup R^+ - R^-$ will be called the *repaired database*. □

Thus, repaired databases are consistent databases which are derived from the source database by means of a minimal set of insertion and deletion of tuples. Given a repair R, R^+ denotes the set of tuples which will be added to the database, whereas R^- denotes the set of tuples of D which will be deleted. In the following, for a given repair R and a database D, $R(D) = D \cup R^+ - R^-$ denotes the application of R to D. Moreover, given a database schema DS, we denote with **D** the set of all possible database instance over DS.

Example 4 Assume we are given a database $D = \{p(a), p(b), q(a), \ q(c)\}$ with the *inclusion dependency* $(\forall\, X)\, [\, p(X) \supset q(X)\,]$. D is inconsistent since $p(b) \supset q(b)$ is not satisfied. The repairs for D are $R_1 = (\{q(b)\}, \emptyset)$ and $R_2 = (\emptyset, \{p(b)\})$ producing, respectively, the repaired databases $R_1(D) = \{p(a), p(b), q(a), q(c), q(b)\}$ and $R_2(D) = \{p(a), q(a), q(c)\}$. □

Definition 4 Given a (possibly inconsistent) database D over a fixed schema $\mathcal{DS}$, and a polynomial function $f : (\mathbf{D}, \mathbf{D}) \times \mathbf{D} \rightarrow \Re$. A repair R_1 is preferable to a repair R_2, w.r.t. the function f, written $R_1 \ll_f R_2$, if $f(R_1, D) \leq f(R_2, D)$. A repair R for D is said to be *preferred* w.r.t. the function f if there is no repair R' for D such that $R' \ll_f R$. A repaired database $D' = D \cup R^+ - R^-$ is said to be a *preferred database* if $R = (R^+, R^-)$ is a preferred repair. □

The above function f will be called *(repair) evaluation function* as it is used to evaluate a repair R with respect to a database D. A preferred database minimizes the value of the evaluation function f applied to the source database and repairs. Observe that, in the above definition, **D** denotes the domain of all possible database instances whereas $(\mathbf{D}, \mathbf{D})$ denotes the domain of all possible database updates. This means that the evaluation function f can be used to measure any possible modification of the input databases and not only to measure repairs, i.e. modification which make the database consistent. In the following, for the sake of simplicity, we only consider function which minimize the cardinality of a set.

Example 5 Consider the database $D = \{Teaches(t_1, c_1), Teaches(t_2, c_2), \ Faculty(t_1, f_1), Faculty(t_2, f_1), Course(c_1, f_1), Course(c_2, f_2)\}$ derived from the union of the databases D_1 and D_2 of Example 1 and the integrity constraint

$$\forall (P, C, F) [\, Teaches(P, C) \wedge Faculty(P, F) \supset Course(C, F)\,]$$

Let R be a repair for the database D, possible evaluation functions are:

- $f_1(R, D) = |R^+|$ computing the number of inserted atoms,
- $f_2(R, D) = |R^-|$ computing the number of deleted atoms,
- $f_3(R, D) = |R^-| + |R^+|$ computing the number of deleted and inserted atoms.

As seen in Example 1, there are three repairs for D: $R_1 = (\emptyset, \{Teaches(t_2, c_2)\})$, $R_2 = (\emptyset, \{Faculty(t_2, f_1)\})$ and $R_3 = (\{Course(c_2, f_1)\}, \emptyset)$. With respect to the above evaluation functions we have the following relations:

1. $R_1 \ll_{f_1} R_3$ and $R_2 \ll_{f_1} R_3$
2. $R_3 \ll_{f_2} R_1$ and $R_3 \ll_{f_2} R_2$

Thus, considering the minimization of the above evaluation functions we have that under the function f_2, R_3 is the unique preferred repair, under the function f_1, we have two preferred repairs: R_1 and R_2, and under the function f_3 all repairs are preferred. □

Given a database D, a set of integrity constraints $\mathcal{IC}$ and an evaluation function f, we denote with $\mathbf{R}(D, \mathcal{IC}, f)$ the set of preferred repairs for D. In the above example $\mathbf{R}(D, \mathcal{IC}, f_1) = \{R_1, R_2\}$, whereas $\mathbf{R}(D, \mathcal{IC}, f_2) = \{R_3\}$.

Moreover, we denote with f_0 any constant evaluation function (e.g. $f_0(R, D) = 0$, the function returning the value 0). $\mathbf{R}(D, \mathcal{IC}, f_0)$ denotes the set of all feasible repairs for D as no preference is introduced.

3.2 Preferred answers for queries over inconsistent databases

A (relational) query over a database defines a function from the database to a relation. It can be expressed by means of alternative equivalent languages such as relational algebra, 'safe' relational calculus or 'safe' non recursive Datalog [24]. In the following we shall use Datalog. Thus, a query is a pair $(g, \mathcal{P})$ where $\mathcal{P}$ is a safe non-recursive Datalog program and g is a predicate symbol specifying the output (derived) relation. Observe that relational queries define a restricted case of disjunctive queries. The reason for considering relational and disjunctive queries is that, as we shall show next, relational queries over databases with constraints can be rewritten into extended disjunctive queries over databases without constraints.

Definition 5 Given a database D, a set of integrity constraints $\mathcal{IC}$, and an evaluation function f, an atom A is *true* (resp. *false*) with respect to $\mathcal{IC}$ and f, if A belongs to all preferred repaired databases (resp. there is no preferred repaired database containing A). The atoms which are neither true nor false are *undefined*. □

Thus, true atoms appear in all preferred repaired databases whereas undefined atoms appear in a proper subset of preferred repaired databases. Given a database D, a set of integrity constraints $\mathcal{IC}$ and an evaluation function f, the application of $\mathcal{IC}$ to D (under f), denoted by $\mathcal{IC}_f(D)$, defines three distinct sets of atoms: the set of true atoms $\mathcal{IC}_f(D)^+$, the set of undefined atoms $\mathcal{IC}_f(D)^u$ and the set of false atoms $\mathcal{IC}_f(D)^-$.

Given a database D, a set of integrity constraint $\mathcal{IC}$, an evaluation function f and a query $Q = (g, \mathcal{P})$, the application of the query Q to the database D with constraint $\mathcal{IC}$ under the function f is denoted by $Q(D, \mathcal{IC}, f)$.

Definition 6 The answer to a query $Q(D, \mathcal{IC}, f)$ consists of three sets, denoted as $Q(D, \mathcal{IC}, f)^+$, $Q(D, \mathcal{IC}, f)^-$ and $Q(D, \mathcal{IC}, f)^u$, containing, respectively, the sets of g-tuples which are $true$ (i.e. belonging to $Q(D')$ for all preferred repaired databases D'), *false* (i.e. not belonging to $Q(D')$ for all preferred repaired databases D') and *undefined* (i.e. set of tuples which are neither true nor false). □

Example 6 Consider the integrated databases of Example 2, the query $Q = (q, P)$ where P consists of the following rule: $q(Y) \leftarrow Teaches(john, Y, Z)$ and the evaluation function f measuring the cardinality of the repairs. The preferred answer to the query Q over the inconsistent databases (w.r.t. the constraint defined by the functional dependency), gives the unique result $\{science\}$. □

Theorem 1 *Given a database D, a set of integrity constraint $\mathcal{IC}$, an evaluation function f and a query $Q = (g, \mathcal{P})$, then*

1. $\boldsymbol{R}(D, \mathcal{IC}, f) \subseteq \boldsymbol{R}(D, \mathcal{IC}, f_0)$
2. $Q(D, \mathcal{IC}, f_0) \subseteq Q(D, \mathcal{IC}, f)$ □

The above theorem states that the introduction of preference criteria reduces the number of repairs and enlarges the answer (i.e. reduces the set of undefined elements of the answer).

4 Managing Inconsistent Databases

We present a technique which permits us to compute repairs and consistent answers for possibly inconsistent databases. The technique is based on the generation of a disjunctive program $\mathcal{DP}(\mathcal{IC})$ derived from the set of integrity constraints $\mathcal{IC}$. The repairs for the database can be generated from the stable models of $\mathcal{DP}(\mathcal{IC})$ whereas the computation of the consistent answers of a query $(g, \mathcal{P})$ can be derived by considering the stable models of the program $\mathcal{P} \cup \mathcal{DP}(\mathcal{IC})$ over the database D.

Integrity constraints express semantic information over data, i.e. relationships that must hold among data in the theory and they are mainly used to validate database transactions. Integrity constraints represent the interaction among data and define properties which are supposed to be satisfied by all instances over a given database schema explicitly. They are usually defined by first order rules or by means of special notations for particular classes such as keys and functional dependencies. Generally, a database D has associated a schema $\mathcal{DS} = \langle Rs, \mathcal{IC} \rangle$ which defines the intentional properties of D: Rs denotes the structure of the relations whereas $\mathcal{IC}$ contains the set of integrity constraints.

Definition 7 Let c be a universally quantified constraint of the form

$$(\forall X)[B_1 \wedge ... \wedge B_n \wedge \varphi \supset A_1 \vee \cdots \vee A_m \,]$$

where $B_1, ..., B_n, A_1, ..., A_n$ are positive atoms and φ is a conjunction of built-in atoms. Then, $dj(c)$ denotes the extended disjunctive rule

$$\begin{aligned} \neg\, B'_1 \vee ... \vee \neg\, B'_n \vee\ A'_1 \vee ... \vee\ A'_m \quad \leftarrow \\ (B_1 \vee B'_1), ..., (B_n \vee B'_n), \varphi, \\ (not\ A_1 \vee \neg A'_1), ..., (not\ A_m \vee \neg A'_m) \end{aligned}$$

where C'_i denotes the atom derived from C_i by replacing the predicate symbol p with the new symbol p_d. □

Example 7 Consider the following integrity constraints:

1. $(\forall X)\ [\, p(X) \supset s(X) \vee q(X)\,]$
2. $(\forall X)\ [\, q(X) \supset r(X)\,]$

and the database D containing the facts $p(a), p(b), s(a)$ and $q(a)$. The derived generalized extended disjunctive program is defined as follows:

$$\begin{aligned} \neg p_d(X) \vee s_d(X) \vee q_d(X) \leftarrow (p(X) \vee p_d(X)), \\ (not\ s(X) \vee \neg s_d(X)), (not\ q(X) \vee \neg q_d(X)) \\ \neg q_d(X) \vee r_d(X) \leftarrow (q(X) \vee q_d(X)), (not\ r(X) \vee \neg r_d(X)) \end{aligned}$$

The above rules can be now rewritten in standard form by eliminating body disjunctions. Let $\mathcal{P}$ be the corresponding extended disjunctive Datalog program, the computation of the program $\mathcal{P}_D$, derived from the union of $\mathcal{P}$ with the facts in D, gives the following stable models:

$$\begin{aligned} M_1 &= D \cup \{\neg p_d(b), \neg q_d(a)\},\\ M_2 &= D \cup \{\neg p_d(b), r_d(a)\},\\ M_3 &= D \cup \{\neg q_d(a), s_d(b)\},\\ M_4 &= D \cup \{r_d(a), s_d(b)\},\\ M_5 &= D \cup \{q_d(b), \neg q_d(a), r_d(b)\},\\ M_6 &= D \cup \{q_d(b), r_d(a), r_d(b)\}. \end{aligned}$$

□

Thus, $\mathcal{DP}(\mathcal{IC})$ denotes the set of generalized disjunctive rules derived from the rewriting of $\mathcal{IC}$, $\mathcal{DP}(\mathcal{IC})_D$ denotes the program derived from the union of the rules in $\mathcal{DP}(\mathcal{IC})$ with the facts in D and $\mathcal{SM}(\mathcal{DP}(\mathcal{IC})_D)$ (resp. $\mathcal{MM}(\mathcal{DP}(\mathcal{IC})_D)$) denotes the set of stable (resp. minimal) models of $\mathcal{DP}(\mathcal{IC})_D$. Recall that every stable model is consistent, according to the definition of consistent set given in Section 2, since it cannot contain two atoms of the form A and $\neg A$.

A *functional dependency* $X \to Y$ over a relation p can be expressed by a formula of the form

$$(\forall(X,Y,Z,U,V))[\, p(X,Y,U) \wedge p(X,Z,V) \supset Y = Z \,]$$

where X, Y, Z, U, V are lists of variables and X, Y, Z may be empty lists. An *inclusion dependency* is of the form

$$(\forall(X,Y))[\, p_1(X_1) \wedge ... \wedge p_n(X_n) \supset q(Y) \,]$$

where $X_1, ..., X_n$ and Y are lists of variables and $Y \subseteq X_1 \cup ... \cup X_n$.

4.1 Computing preferred database repairs

Every stable model can be used to define a possible repair for the database by interpreting new derived atoms (denoted by the subscript “d”) as insertions and deletions of tuples. Thus, if a stable model M contains two atoms $\neg p_d(t)$ (derived atom) and $p(t)$ (base atom) we deduce that the atom $p(t)$ violates some constraint and, therefore, it must be deleted. Analogously, if M contains the derived atoms $p_d(t)$ and does not contain $p(t)$ (i.e. $p(t)$ is not in the database) we deduce that the atom $p(t)$ should be inserted in the database. We now formalize the definition of repaired database.

Definition 8 Given a database D and a set of integrity constraint $\mathcal{IC}$ over D and letting M be a stable model of $\mathcal{DP}(\mathcal{IC})_D$, then, $\mathcal{R}(M) = (\ \{p(t) \mid p_d(t) \in M \ \wedge \ p(t) \notin D\},\ \{p(t) \mid \neg p_d(t) \in M \ \wedge \ p(t) \in D\}\ )$. □

Theorem 2 *[12]*
Given a database D and a set of integrity constraints $\mathcal{IC}$ on D, then

1. *(Soundness) for every stable model M of $\mathcal{DP}(\mathcal{IC})_D$, $\mathcal{R}(M)$ is a repair for D;*
2. *(Completeness) for every database repair S for D there exists a stable model M for $\mathcal{DP}(\mathcal{IC})_D$ such that $S = \mathcal{R}(M)$.* □

Example 8 Consider the database of Example 4. The rewriting of the integrity constraint $(\forall X)[\ p(X) \supset q(X)\ ]$ produces the disjunctive rule

$$\begin{aligned} r : \neg p_d(X) \vee q_d(X) \quad \leftarrow \quad & (p(X) \vee p_d(X)),\\ & (not\ q(X) \vee \neg q_d(X)). \end{aligned}$$

which can be rewritten in the simpler form

$$r' : \neg p_d(X) \vee q_d(X) \quad \leftarrow \quad p(X), not\ q_d(X).$$

since the predicates p_d and $\neg q_d$ do not appear in the head of any rule. The program P_D, where P is the program consisting of the disjunctive rule r' and D is the input database, has two stable models $M_1 = D \cup \{\ \neg p_d(b)\}$ and $M_2 = D \cup \{\ q_d(b)\}$. The derived repairs are $\mathcal{R}(M_1) = (\{q(b)\}, \emptyset)$ and $\mathcal{R}(M_2) = (\emptyset, \{p(b)\})$ corresponding, respectively, to the insertion of $q(b)$ and the deletion of $p(b)$.

Theorem 3
Let D be a database, $\mathcal{IC}$ a set of functional dependencies and f an evaluation function measuring the cardinality of repairs. Then,

1. *(Soundness) for every stable model M of $\mathcal{DP}(\mathcal{IC})_D$ with minimum cardinality, $\mathcal{R}(M)$ is a preferred repair (w.r.t. f) for D;*
2. *(Completeness) for every preferred database repair S for D there exists a stable model M for $\mathcal{DP}(\mathcal{IC})_D$ with minimal cardinality such that $\mathcal{R}(M) = S$.*
3. *$\mathcal{R}(M)$ can be computed in polynomial time.* □

Theorem 4
Let D be a database, $\mathcal{IC}$ a set of full inclusion dependencies and f an evaluation function measuring the number of insertions (resp. deletions). Then

1. *there is a unique preferred (w.r.t. f) repair S and there exists a unique stable model M such that $\mathcal{R}(M) = S$;*
2. *$M^+ = \emptyset$ (resp. $M^- = \emptyset$);*
3. *$\mathcal{R}(M)$ can be computed in polynomial time.* □

4.2 Computing preferred answers

We now consider the problem of computing a preferred (consistent) answers without modifying the (possibly inconsistent) database. We assume that tuples, contained in the database or implied by the constraints, may be *true*, *false* or *undefined*.

Let D be a database, $\mathcal{IC}$ a set of integrity constraints and f an evaluation function, then we denote with $\mathcal{PSM}_f(\mathcal{DP}(\mathcal{IC})_D)$ the set of preferred stable models of $\mathcal{DP}(\mathcal{IC})_D$ with respect to the function $f(\mathcal{R}(M), D)$.

From the results of Section 3.1 we derive

$$\begin{array}{l}\mathcal{IC}_f(D)^+ = \\ \quad \{\, p(t) \in D \mid \nexists M \in \mathcal{PSM}_f(\mathcal{LP}(\mathcal{IC})_D) \ s.t. \neg p_d(t) \in M \,\} \cup \\ \quad \{\, p(t) \notin D \mid \forall M \in \mathcal{PSM}_f(\mathcal{LP}(\mathcal{IC})_D) \ s.t. \ p_d(t) \in M \,\},\end{array}$$

$$\begin{array}{l}\mathcal{IC}_f(D)^- = \\ \quad \{\, p(t) \notin D \mid \nexists M \in \mathcal{PSM}_f(\mathcal{LP}(\mathcal{IC})_D) \ s.t. \ p_d(t) \in M \,\} \cup \\ \quad \{\, p(t) \in D \mid \forall M \in \mathcal{PSM}_f(\mathcal{LP}(\mathcal{IC})_D) \ s.t. \neg p_d(t) \in M \,\},\end{array}$$

$$\begin{array}{l}\mathcal{IC}_f(D)^u = \\ \quad \{p(t) \in D \mid \exists M_1, M_2 \in \mathcal{PSM}_f(\mathcal{LP}(\mathcal{IC})_D) \ s.t. \\ \qquad \neg p_d(t) \in M_1 \,, \neg p_d(t) \notin M_2 \,\} \ \cup \\ \quad \{p(t) \notin D \mid \exists M_1, M_2 \in \mathcal{PSM}_f(\mathcal{LP}(\mathcal{IC})_D) \ s.t. \\ \qquad p_d(t) \in M_1 \,, p_d(t) \notin M_2 \,\}.\end{array}$$

Observe that the sets $\mathcal{IC}_f(D)^+$, $\mathcal{IC}_f(D)^-$ and $\mathcal{IC}_f(D)^u$ are disjoint and that $\mathcal{IC}_f(D)^+ \cup \mathcal{IC}_(fD)^-$ defines a set of consistent atoms. We are now in the position to introduce the computation of (preferred) consistent answer.

The preferred consistent answer for the query $Q = (g, \mathcal{P})$ over the database D under constraints $\mathcal{IC}$ is as follows:

$$\begin{array}{l}Q(D, \mathcal{IC}, f)^+ = \\ \quad \{g(t) \in D \mid \ \nexists M \in \mathcal{PSM}_f((\mathcal{P} \cup \mathcal{LP}(\mathcal{IC}))_D) \ s.t. \neg g_d(t) \in M \,\} \cup \\ \quad \{g(t) \notin D \mid \forall M \in \mathcal{PSM}_f((\mathcal{P} \cup \mathcal{LP}(\mathcal{IC}))_D) \ s.t. \ g_d(t) \in M \,\},\end{array}$$

$$\begin{array}{l}Q(D, \mathcal{IC}, f)^u = \\ \quad \{\, g(t) \in D \mid \exists M_1, M_2 \in \mathcal{PSM}_f((\mathcal{P} \cup \mathcal{LP}(\mathcal{IC}))_D) \ s.t. \\ \qquad \neg g_d(t) \in M_1 \,, \neg g_d(t) \notin M_2 \,\} \cup \\ \quad \{\, g(t) \notin D \mid \exists M_1, M_2 \in \mathcal{PSM}_f((P \cup \mathcal{LP}(\mathcal{IC}))_D) \ s.t. \\ \qquad g_d(t) \in M_1 \,, g_d(t) \notin M_2 \,\}.\end{array}$$

whereas the set of atoms which are neither true nor undefined can be assumed to be false.

Theorem 5
Let D be a database, $Q = (g, \mathcal{P})$ a query, f a polynomial evaluation function and D' a repaired database preferred (w.r.t. f). Then

1. *each atom $A \in Q(D, \mathcal{IC}, f)^+$ belongs to the stable model of $\mathcal{P}_{D'}$ (soundness)*
2. *each atom $A \in Q(D, \mathcal{IC}, f)^-$ does not belong to the stable model of $\mathcal{P}_{D'}$ (completeness).* □

Example 9 Consider the database of Example 5, the integrity constraint:

$$\forall(P, C, F)[\,Teaches(P, C) \land Faculty(P, F) \supset Course(C, F)\,]$$

and the evaluation functions $f_1(R, D) = |R^+|$ and $f_2(R, D) = |R^-|$ computing respectively the number of inserted and deleted atoms. The program P_D has three stable models: $M_1 = D \cup \{\neg Teaches_d(t_2, c_2)\}$, $M_2 = D \cup \{\neg Faculty_d(t_2, f_1)\}$ and $M_3 = D \cup \{Course_d(c_2, f_1)\}$. Considering the evaluation function $f_1(R, D) = |R^+|$ the set of preferred models consists of M_1 and M_2; therefore, the atoms which are true and undefined are: $\mathcal{IC}_{f_1}(D)^+ = \{Teaches(t_1, c_1), Faculty(t_1, f_1), Course(c_1, f_1), Course(c_2, f_2)\}$ and $\mathcal{IC}_{f_1}(D)^u = \{Teaches(t_2, c_2), Faculty(t_2, f_1)\}$. Considering the evaluation function $f_2(R, D) = |R^-|$, M_3 is the unique preferred model; thus, the set of undefined atoms is empty, whereas the set of true atoms is $\mathcal{IC}_{f_2}(D)^+ = \{Teaches(t_1, c_1),\ Teaches(t_2, c_2),\ Faculty(t_1, f_1),\ Faculty(t_2, f_1),\ Course(c_1, f_1),\ Course(c_2, f_2),\ Course(c_2, f_1)\}$. The answer to the query $(Teaches, \emptyset)$ is $\{(t_1, c_1)\}$ under the evaluation function f_1 and $\{(t_1, c_1), (t_2,\, c_2)\}$ under the evaluation function f_2. The answer to the query $(Course, \emptyset)$ is $\{(c_1, f_1),\ (c_2, f_2)\}$, under the function f_1 and $\{\ (c_1, f_1), (c_2, f_2),\ (c_2, f_1)\}$ under the function f_2. □

5 Conclusions

In this paper we have proposed a logic programming based framework for managing possibly inconsistent databases. The main contribution of this work consists in the definition of a logic approach for querying and repairing inconsistent databases that extends previous works by also considering techniques to express and manage preferences among repairs and possible answers. Preference criteria can be introduced to specify desiderata on how to update the inconsistent database in order to make it consistent and are expressed by means of *evaluation functions*, i.e. polynomial functions that are applied to repairs and return real numbers. The evaluation function defines a partial order both among repairs and possible answer, thus it represents a flexible mechanism for selecting among a set of feasible repairs those better conforming to the specified criteria. The goodness of a repair is measured by estimating how much it violates the desiderata conditions and a repair is "preferred" if it minimizes the value of the polynomial function used to express the preference criteria. A further important characteristic related to the introduction of preference criteria is the reduction of feasible repairs and answers, which let, for special cases of constraints, to unique repair and answer.

References

[1] Abiteboul S., Hull R., Vianu V. *Foundations of Databases*. Addison-Wesley, 1994.

[2] Argaval, S., Keller, A. M., Wiederhold, G., Saraswat, K., Flexible Relation: an Approach for Integrating Data from Multiple, Possibly Inconsistent Databases. *ICDE*, 1995.

[3] Arenas, M., Bertossi, L., Chomicki, J., Consistent query Answers in inconsistent databases. *PODS*, pp. 68–79, 1999.

[4] Arenas, M., Bertossi, L., Chomicki, J., Specifying and Querying Database repairs using Logic Programs with Exceptions. *FQAS*, pp. 27-41, 2000.

[5] Baral, C., Kraus, S., Minker, J., Combining Multiple Knowledge Bases. *TKDE*, 3(2), pp. 208-220, 1991.

[6] Bry, F., Query Answering in Information System with Integrity Constraints, *IFIP WG 11.5 Working Conf. on Integrity and Control in Inform. System*, 1997.

[7] Dung, P. M., Integrating Data from Possibly Inconsistent Databases. *CoopIS*, pp. 58-65, 1996.

[8] Eiter, T., Gottlob, G., Mannila, H., Disjunctive Datalog. *TODS*, 22(3), pp. 364–418, 1997.

[9] Gelfond, M., Lifschitz, V. The Stable Model Semantics for Logic Programming, *ICLP*, pp. 1070–1080, 1988.

[10] Gelfond, M., Lifschitz, V., Classical Negation in Logic Programs and Disjunctive Databases, *NGC*, No. 9, pp. 365–385, 1991.

[11] Greco, S., Saccà, D., Negative Logic Programs. *NACLP*, pp. 480-497, 1990.

[12] Greco, S., Zumpano, E., Querying Inconsistent Database *LPAR*, pp. 308-325, 2000.

[13] Greco, G., Greco, S., Zumpano, E., A Logic Programming Approach to the Integration, Repairing and Querying of Inconsistent Databases. *ICLP*, 2001.

[14] Grant, J., Subrahmanian, V. S., Reasoning in Inconsistent Knowledge Bases, *TKDE*,7(1), pp. 177-189, 1995.

[15] Kanellakis, P. C., Elements of Relational Database Theory. *Handbook of Theoretical Computer Science*, Vol. 2, J. van Leewen (ed.), North-Holland, 1991.

[16] Kifer, M., Li, A., On the Semantics of Rule-Based Expert Systems with Uncertainty. *Int. Conf. on Database Theory* pp. 102-11, 1988.

[17] Kowalski, R. A., Sadri, F., Logic Programs with Exceptions. *NGC*, 9(3/4), pp. 387-400, 1991.

[18] Lin, J., A Semantics for Reasoning Consistently in the Presence of Inconsistency. *AI*, 86(1), pp. 75-95, 1996.

[19] Lin, J., and Mendelzon, A. O., Knowledge Base Merging by Majority, in *Dynamic Worlds: From the Frame Problem to Knowledge Management*, R. Pareschi and B. Fronhoefer (eds.), Kluwer, 1999.

[20] Lloyd, J., *Foundation of Logic Programming*. Spinger-Verlag, 1987.

[21] Minker, J., On Indefinite Data Bases and the Closed World Assumption, *6-th Conf. on Automated Deduction*, pp. 292–308, 1982.

[22] Sakama, C., Inoue, K., Priorized logic programming and its application to commonsense reasoning. *AI*, No. 123, pp. 185-222, 2000.

[23] Subrahmanian, V. S., Amalgamating Knowledge Bases. *ACM ToDS*, 19(2), pp. 291-331, 1994.

[24] Ullman, J. K., *Principles of Database and Knowledge-Base Systems*, Vol. 1, Computer Science Press, 1988.

[25] Wijsen, J., Condensed representation of database repairs for consistent query answering, *ICDT*, pp. 378-393, 2003.

[26] Wang, X, You, J. H., Yuan, L. Y., Nonmonotonic reasoning by monotonic inferences with priority conditions. *Proc. Int. Workshop on Nonmonotonic Extensions of Logic Programming*. pp. 91-109, 1996.

[27] Yan, L.L., Ozsu, M.T., Conflict Tolerant Queries in Aurora. *CoopIS*, 1999; pp. 279–290.

[28] Zang, Y., Foo, N., Answer sets for prioritized logic programs. *(ILPS*, pp. 69-83, 1997.