Managing Inconsistent Databases Using Active Integrity Constraints

Sergio Flesca
DEIS Università della Calabria, Italy

Sergio Greco
DEIS Università della Calabria, Italy

Ester Zumpano
DEIS Università della Calabria, Italy

INTRODUCTION

Integrity constraints are a fundamental part of a database schema. They are generally used to define constraints on data (functional dependencies, inclusion dependencies, exclusion dependencies, etc.), and their enforcement ensures a semantically correct state of a database. As the presence of data inconsistent with respect to integrity constraints is not unusual, its management plays a key role in all the areas in which duplicate or conflicting information is likely to occur, such as database integration, data warehousing, and federated databases (Bry, 1997; Lin, 1996; Subrahmanian, 1994). It is well known that the presence of inconsistent data can be managed by “repairing” the database, that is, by providing consistent databases, obtained by a minimal set of update operations on the inconsistent original environment, or by consistently answering queries posed over the inconsistent database.

The motivation of this work stems from the observation that in repairing a database it is natural to express among a set of update requirements the preferred ones, that is, those actions which, besides making the database consistent, also maintain the preferred information. The novelty of our approach consists in the formalization of active integrity constraints, a flexible and easy mechanism for specifying the preferred updates, that is, the actions that should be performed if an integrity constraint is not satisfied. In some sense, active integrity constraints represent a restricted form of active rules sufficient to (declaratively) express database repairs but without the typical problems of active databases, such as termination and procedural interpretation of rules. Thus, in the general case, active integrity constraints can be thought of as a means to define an “intended” repairing strategy.

Recently, there have been several proposals considering the computation of repairs and queries over inconsistent databases (Arenas, Bertossi & Chomicki, 1999, 2000; Greco, Greco & Zumpano, 2001; Wijsen, 2003). Other works have investigated the updating of data and knowledge bases through the use of active rules and nonmonotonic formalisms. The application of the ECA (event-condition-action) paradigm of active databases to policies—collection of general principles specifying the desired behavior of systems—has been investigated in Chomicki, Lobo, and Naqvi (2003). In this work, the authors propose the introduction of active constraints to describe under which circumstances a set of actions cannot be executed simultaneously. In Alferes et al. (2000), the problem of updating knowledge bases represented by logic programs has been investigated. More specifically, the authors introduce the notion of updating a logic program $P$ by means of another logic program $U$ (denoted by $P \oplus U$) and a new paradigm, called dynamic logic programming, to model dynamic program update. The new paradigm has been further investigated in Alferes et al. (2002), where the language LUPS (Language for Dynamic Updates), designed for specifying changes to logic programs, has been proposed.

Nonmonotonic formalisms, such as Revision Programs (Marek, Pivkina & Truszczyński, 1998; Marek & Truszczyński, 1998), are based on the extension of the logic programming paradigm. Unlike earlier approaches in belief revision, where updates were represented in classical theories, revision programs are a collection of rules with nonclassical semantics; they can be interpreted as inference rules and are used to update interpretations. ECA languages based on revision programs have been proposed as well in Baral (1997).

The approach here proposed differs both from ECA languages, as only sets of actions making the input database consistent are allowed, and from revision programs, as actions can be enforced not only by the initial state of the database.
BACKGROUND

We assume that readers are familiar with relational and deductive databases (Abiteboul, Hull & Vianu, 1995; Ullman, 1988).

A (disjunctive Datalog) rule \( r \) is a clause of the form
\[
A_1 \lor \ldots \lor A_k \leftarrow B_1, \ldots, B_m, \text{not } B_{m+1}, \ldots, \text{not } B_n, \text{ k+m+n} \geq 0
\]
where \( A_1, \ldots, A_k, B_1, \ldots, B_n \) are atoms of the form \( p(t_1, \ldots, t_h) \), \( p \) is a predicate symbol of arity \( h \), and the terms \( t_1, \ldots, t_h \) are constants or variables (Eiter et al., 1998). The disjunction \( A_1 \lor \ldots \lor A_k \) is the head of \( r \), while the conjunction \( B_1, \ldots, B_m, \text{not } B_{m+1}, \ldots, \text{not } B_n \) 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 Herbrand Universe \( U_P \) of a program \( P \) is the set of all constants appearing in \( P \), and its Herbrand Base \( B_P \) is the set of all ground atoms constructed from the predicates appearing in \( P \) and the constants from \( U_P \). A term, (resp. an atom, a literal, a rule, or a program) is ground if no variables occur in it. A rule \( r' \) is a ground instance of a rule \( r \), if \( r' \) is obtained from \( r \) by replacing every variable in \( r \) with some constant in \( U_P \). We denote by \( \text{ground}(P) \) the set of all ground instances of the rules in \( P \). An interpretation \( M \) of \( P \) is any subset of \( B_P \).

An interpretation \( M \) for \( P \) is a model of \( P \) if \( M \) satisfies each rule in \( \text{ground}(P) \). The (model-theoretic) semantics for a positive program, say \( P \), assigns to \( P \) the set of its minimal models \( \text{MM}(P) \), where a model \( M \) for \( P \) is minimal, if no proper subset of \( M \) is a model for \( P \) (Minker, 1982). The more general disjunctive stable model semantics also applies to programs with (unstratified) negation (Gelfond & Lifschitz, 1991).

For any interpretation \( I \), denote with \( P^I \) the ground positive program derived from \( \text{ground}(P) \) (1) by removing all rules that contain a negative literal not \( a \) in the body and \( a \in I \), and (2) by removing all negative literals from the remaining rules. An interpretation \( M \) is a (disjunctive) stable model of \( P \) if and only if \( M \in \text{MM}^+(P^I) \). For general \( P \), the stable model semantics assigns to \( P \) the set \( \text{SM}(P) \) of its stable models. It is well known that stable models are minimal models (i.e., \( \text{SM}(P) = \text{MM}(P) \)) and that for negation free programs, minimal and stable model semantics coincide (i.e., \( \text{SM}(P) = \text{MM}(P) \)). Observe that stable models are minimal models which are “supported”; that is, their atoms can be derived from the program. An alternative semantics which overcomes some problems of stable model semantics has been recently proposed in Greco (1999).

INTEGRITY CONSTRAINTS

Integrity constraints express semantic information over data, that is, relationships that must hold among data in the theory. Generally, integrity constraints, denoted as IC, represent the interaction among data and define properties which are supposed to be explicitly satisfied by all instances over a given database schema. Therefore, they are mainly used to validate database transactions.

Definition 1

A full (or universal) integrity constraint is a formula of the first order predicate calculus of the form:

\[
(\forall X) \left[ B_1 \land \ldots \land B_n \land \varphi \supset A_1 \lor \ldots \lor A_m \lor \psi_1 \lor \ldots \lor \psi_k \right]
\]

where \( A_1, \ldots, A_m, B_1, \ldots, B_n \) are base positive literals, \( \varphi, \psi_1, \ldots, \psi_k \) are built-in literals, \( X \) denotes the list of all variables appearing in \( B_1, \ldots, B_n \) and it is supposed that variables appearing in \( A_1, \ldots, A_m, \varphi, \psi_1, \ldots, \psi_k \) also appear in \( B_1, \ldots, B_n \).

In the definition above, the conjunction \( B_1 \land \ldots \land B_n \) is called the body, and the disjunction \( A_1 \lor \ldots \lor A_m \lor \psi_1 \lor \ldots \lor \psi_k \) the head of the integrity constraint. Moreover, an integrity constraint is said to be positive if no negated literals occur in it. Classical definitions of integrity constraints only consider positive nondisjunctive constraints, called embedded dependencies (Kanellakis, 1991).

Often we shall write our constraints in a different format by moving literals from the head to the body and vice versa.

REPAIRING INCONSISTENT DATABASES

Intuitively, a repair for a (possibly inconsistent) database \( D \) is a minimal consistent set of insert and delete operations which makes \( D \) consistent, whereas a consistent answer for a query consists of two sets containing, respectively, the maximal set of true and undefined atoms which match the query goal; atoms which are neither true nor undefined can be assumed to be false.

More formally:
Related Content

On U.S. Homeland Security and Database Technology
[www.igi-global.com/article/homeland-security-database-technology/3323?camid=4v1](www.igi-global.com/article/homeland-security-database-technology/3323?camid=4v1)

Behavioral Aspects of Data Production and Their Impact on Data Quality
[www.igi-global.com/article/behavioral-aspects-data-production-their/51119?camid=4v1](www.igi-global.com/article/behavioral-aspects-data-production-their/51119?camid=4v1)

The Impact of Ideology on the Organizational Adoption of Open Source Software
[www.igi-global.com/chapter/impact-ideology-organizational-adoption-open/39354?camid=4v1](www.igi-global.com/chapter/impact-ideology-organizational-adoption-open/39354?camid=4v1)

Implementing Relational Database Systems: Implications for Administrative Cultures and Information Resource Management
[www.igi-global.com/chapter/implementing-relational-database-systems/6207?camid=4v1](www.igi-global.com/chapter/implementing-relational-database-systems/6207?camid=4v1)