Declarative Semantics of Production Rules for Integrity Maintenance

Luciano Caroprese, Sergio Greco, Cristina Sirangelo, and Ester Zumpano

DEIS, Univ. della Calabria, 87030 Rende, Italy
{caroprese, greco, sirangelo, zumpano}@deis.unical.it

Abstract. This paper presents a declarative semantics for the maintenance of integrity constraints expressed by means of production rules. A production rule is a special form of active rule, called active integrity constraint, whose body contains an integrity constraint (conjunction of literals which must be *false*) and whose head contains a disjunction of update atoms, i.e. actions to be performed if the corresponding constraint is not satisfied (i.e. is true). The paper introduces i) a formal declarative semantics allowing the computation of founded repairs, that is repairs whose actions are specified and supported by active integrity constraint, ii) an equivalent semantics obtained by rewriting production rules into disjunctive logic rules, so that repairs can be derived from the answer sets of the logic program and finally iii) a characterization of production rules allowing a methodology for integrity maintenance.

1 Introduction

Integrity constraints are logical assertions on acceptable (or consistent) database states, and specify properties of data that need to be satisfied by valid instances of the database [1]. When constraints are violated, for example during or at the end of the execution of a transaction, the repair of the database state is usually limited to fixed reversal actions, such as rolling back the current operation or the entire transaction [6]. Moreover, since 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 [17,18,24]. Thus, an improved approach to constraints enforcement allows definition of compensating actions that correct violation of constraints according to a well-defined semantics *(database repairs)* or allows computing *consistent answers*. Informally, the computation of repairs is based on the insertion and deletion of tuples so that the resulting database satisfies all constraints, whereas the computation of consistent answers is based on the identification of tuples satisfying integrity constraints and matching the goal.

The following example shows a situation in which inconsistencies occur.

Example 1. Consider the relation schema $mgr(Name, Dept, Salary)$ with the functional dependency $Dept \rightarrow Name$ which can be defined through the first order formula $\forall(E_1, E_2, D, S_1, S_2)\ [mgr(E_1, D, S_1) \wedge mgr(E_2, D, S_2) \supset E_1 = E_2]$.

S. Etalle and M. Truszczyński (Eds.): ICLP 2006, LNCS 4079, pp. 26–40, 2006.

Consider now the inconsistent instance: $\mathcal{DB} = \{mgr(john, cs, 1000),\ mgr(frank,\ cs, 2000)\}$. A consistent (repaired) database can be obtained by applying a minimal set of update operations; in particular it admits two repaired databases: $\mathcal{DB}_1 = \{mgr(frank, cs, 2000)\}$ obtained by applying the repair $\mathcal{R}_1 = \{-mgr(john, cs, 1000)\}$ (deleting the tuple $mgr(john, cs, 1000)$) and $\mathcal{DB}_2 = \{mgr(john, cs, 1000)\}$ obtained by applying the repair $\mathcal{R}_2 = \{-mgr(frank,\ cs, 2000)\}$ (deleting the tuple $mgr(frank,\ cs, 2000)$). □

The notion of integrity constraints and their automated maintenance has been investigated for many years. Several works have proposed the updating of data and knowledge bases through the use of active rules [6,7] and nonmonotonic formalisms [2,5,19,20]. Some approaches use ECA (event-condition-action) rules for checking and enforcing integrity constraints, whereas other approaches are based on simpler forms of rules, called CA (condition-action) or *production rules*, in which the event part is absent. Current DBMS languages offer the possibility of defining triggers, special ECA rules well-suited to automatically perform actions, in response to events that are taking place inside (or even outside) the database. However, the problem with active rules is the difficulty to understand the behavior of multiple triggers acting together [21,23]. Although many different proposals have been introduced over the years, at the moment there is no agreement on the integration of active functionalities with conventional database systems. A different solution based on the derivation of logic rules with declarative semantics has been recently proposed in several works. These proposals are based on the automatic generation of Datalog rules and on the computation of answer sets, from which repairs are derived [3,4,8,14,15,16,26]. All these work do not take into account the possibility to indicate the update operations making consistent the database.

This paper considers a special form of production rules called *active integrity constraints* (AIC). Active integrity constraints, recently proposed in [11], are special integrity constraints whose body consists of a conjunction of literals which should be *false* (denial constraint) and whose head contains the actions which should be performed if the body of the rule is true (i.e. the constraints defined in the body is not satisfied). AIC rules allow specification of the actions which should be performed to make the database consistent when integrity constraints are violated.

Example 2. Consider the database of Example 1 and the active constraint:

$\forall(E_1, E_2, D, S_1, S_2)[mgr(E_1, D, S_1) \wedge mgr(E_2, D, S_2) \wedge S_1 > S_2 \supset \neg mgr(E_1, D, S_1)]$

stating that *in case of conflicting tuples it is preferred to delete the one with greater salary.* The constraint suggests to update the database by deleting the tuple $mgr(frank, cs, 2000)$. This action, in the specific case, leads to taking into account only one of the two repairs, namely $\mathcal{R}_2$. □

Thus, active integrity constraints are production rules expressed by means of first order logic with declarative semantics. AIC allow the computation of "preferred" repairs, that is repairs whose actions are specified explicitly and are also supported.

Contributions
The novelty of the approach here proposed consists in the definition of a formal declarative semantics for *active integrity constraints*. The new semantics allows identification, among the set of all possible repairs, of the subset of *founded repairs* whose actions are specified in the head of rules and are "supported" by the database or by other updates. The computation of founded repairs can be done by checking whether for each repair all its update atoms are founded or by rewriting the constraints into a Datalog program and then computing its stable models; the founded repairs are obtained by selecting, for each stable model, the set of "update atoms".

The paper also studies the characteristic of AIC rules and show that for each production rule r (consisting of a body defining the integrity constraint and a head containing alternative actions which should be performed if the constraint is not satisfied), update head atoms not making the conjunction of body literals, defining an integrity constraint, false with respect to the repaired database (i.e. such that the body integrity constraint is satisfied), are useless [1]. This formal result confirms the intuition that for integrity maintenance general (E)CA rules are not necessary. Therefore, active integrity constraints can be thought as special CA rules with declarative semantics whose aim is to repair the database and to help consistently answering queries over inconsistent databases. As, in the general case, the existence of founded repairs is not guaranteed the class of universally quantified constraints under a different semantics in which actions are interpreted as preference conditions on the set of possible repairs ("preferable" semantics) is also investigated. Under such a semantics every database with integrity constraints admits repairs and consistent answers.

Finally, the paper studies the computational complexity and shows that computing founded and preferred repairs and answers is not harder than computing "standard" repairs and answers.

2 Preliminaries

Familiarity with relational database theory, disjunctive logic programming and computational complexity is assumed [1,9,13,22].

Disjunctive Databases. A *(disjunctive Datalog) rule* r is a clause of the form[2]

$$\bigvee_{i=1}^{p} A_i \leftarrow \bigwedge_{j=1}^{m} B_j, \bigwedge_{j=m+1}^{n} not\ B_j, \varphi \qquad p+m+n>0 \tag{1}$$

where $A_1, \cdots, A_p, B_1, \cdots, B_n$ are atoms of the form $p(t_1, ..., t_h)$, p is a *predicate* of arity h and the terms $t_1, ..., t_h$ are constants or variables, while φ is a conjunction of built-in atoms of the form $u\ \theta\ v$ where u and v are terms and θ is a comparison predicate.

[1] Under the declarative semantics here proposed.
[2] A literal can appear in a conjunction or in a disjunction at most once. The meaning of the symbols '$\wedge$' and ',' is the same.

An interpretation $\mathcal{M}$ for $\mathcal{P}$ is a model of $\mathcal{P}$ if $\mathcal{M}$ satisfies all rules in $ground(\mathcal{P})$ (the ground instantiation of $\mathcal{P}$). The (model-theoretic) semantics for a positive program $\mathcal{P}$ assigns to $\mathcal{P}$ the set of its *minimal models* $\mathcal{MM}(\mathcal{P})$, where a model $\mathcal{M}$ for $\mathcal{P}$ is minimal, if no proper subset of $\mathcal{M}$ is a model for $\mathcal{P}$. The more general *disjunctive stable model semantics* also applies to programs with (unstratified) negation [13]. Disjunctive stable model semantics generalizes stable model semantics, previously defined for normal programs [12]. For any interpretation $\mathcal{M}$, denote with $\mathcal{P}^{\mathcal{M}}$ 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 $A \in \mathcal{M}$, and 2) removing all negative literals from the remaining rules. An interpretation $\mathcal{M}$ is a stable model of $\mathcal{P}$ if and only if $\mathcal{M} \in \mathcal{MM}(\mathcal{P}^{\mathcal{M}})$. 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.

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; derived predicates are defined by means of rules. Given a database $\mathcal{DB}$ and a program $\mathcal{P}$, $\mathcal{P}_{\mathcal{DB}}$ denotes the program derived from the union of $\mathcal{P}$ with the facts in $\mathcal{DB}$, i.e. $\mathcal{P}_{\mathcal{DB}} = \mathcal{P} \cup \mathcal{DB}$. In the following a tuple t of a relation r will be also denoted as a fact $r(t)$. The semantics of $\mathcal{P}_{\mathcal{DB}}$ is given by the set of its stable models by considering either their union (*possibly semantics* or *brave reasoning*) or their intersection (*certain semantics* or *cautious reasoning*). A disjunctive Datalog *query* Q is a pair $(g, \mathcal{P})$ where g is a predicate symbol, called the *query goal*, and $\mathcal{P}$ is a disjunctive Datalog program. The answer to a disjunctive Datalog query $Q = (g, \mathcal{P})$ over a database $\mathcal{DB}$ (denoted by $Q(\mathcal{DB})$), under the possibly (resp. certain) semantics, is given by $\mathcal{DB}'(g)$ where $\mathcal{DB}' = \bigcup_{\mathcal{M} \in \mathcal{SM}(\mathcal{P}_{\mathcal{DB}})} \mathcal{M}$ (resp. $\mathcal{DB}' = \bigcap_{\mathcal{M} \in \mathcal{SM}(\mathcal{P}_{\mathcal{DB}})} \mathcal{M}$).

A disjunctive Datalog program $\mathcal{P}$ is said to be *semi-positive* if negation is only applied to database atoms. For a semi-positive program $\mathcal{P}$ and a database $\mathcal{DB}$, the set of stable models coincides with the set of minimal models containing as true database facts only those in $\mathcal{DB}$ (i.e. EDB database atoms not appearing in $\mathcal{DB}$ are assumed to be *false*). A (relational) query can be expressed by means of 'safe' non recursive Datalog, even though alternative equivalent languages such as relational algebra could be used as well [1,25].

3 Databases and Integrity Constraints

A database $\mathcal{DB}$ has an associated schema $\langle \mathcal{DS}, \mathcal{IC} \rangle$ defining the intentional properties of $\mathcal{DB}$: $\mathcal{DS}$ denotes the structure of the relations, while $\mathcal{IC}$ denotes the set of integrity constraints expressing semantic information over data.

3.1 Integrity Constraints

Definition 1. A (universally quantified or full) *integrity constraint* is a formula of the first order predicate calculus of the form:

$$(\forall\ X)[\ \bigwedge_{j=1}^{m} b_j(X_j),\ \varphi(X_0) \supset \bigvee_{j=m+1}^{n} b_j(X_j)\ ]$$

where b_j $(1 \leq j \leq n)$ is a predicate symbol, $\varphi(X_0)$ denotes a conjunction of built-in atoms, $X = \bigcup_{j=1}^{m} X_j$, $X_i \subseteq X$ for $i \in [0\,..\,n]$ and all existentially quantified variables appear once. □

The reason for considering constraints of the above form is that we want to consider range restricted constraints, i.e. constraints whose variables either take values from finite domains only or the exact knowledge of their values is not relevant [25]. Often our constraints will be written in a different form by moving literals from the right side to the left side and vice-versa. For instance, by rewriting the above constraint as denial we obtain:

$$(\forall\ X)[\ \bigwedge_{j=1}^{m} b_j(X_j),\ \bigwedge_{j=m+1}^{n} \mathit{not}\ b_j(X_j, Z_j),\ \varphi(X_0) \supset\ ].$$

In the following we assume that the set of integrity constraints $\mathcal{IC}$ is satisfiable, that is there exists a database instance $\mathcal{DB}$ satisfying $\mathcal{IC}$. For instance, by considering constraints of the above form with $m > 0$, the constraints are satisfied by the empty database.

3.2 Repairing and Querying Inconsistent Databases

In this section the formal definition of consistent database and repair is first recalled and, then, a computational mechanism is presented that ensures selecting repairs and consistent answers for inconsistent databases.

An update atom is in the form $+a(X)$ or $-a(X)$. A ground atom $+a(t)$ states that $a(t)$ will be inserted into the database, whereas a ground atom $-a(t)$ states that $a(t)$ will be deleted from the database. Given an update atom $+a(X)$ (resp. $-a(X)$) we denote as $Comp(+a(X))$ (resp. $Comp(-a(X))$) the literal *not* $a(X)$ (resp. $a(X)$). Given a set $\mathcal{U}$ of ground update atoms we define the sets $\mathcal{U}^+ = \{a(t) \mid +a(t) \in \mathcal{U}\}$, $\mathcal{U}^- = \{a(t) \mid -a(t) \in \mathcal{U}\}$ and $Comp(\mathcal{U}) = \{Comp(\pm a(t)) \mid \pm a(t) \in \mathcal{U}\}$. We say that $\mathcal{U}$ is *consistent* if it does not contain two update atom $+a(t)$ and $-a(t)$ (i.e. if $\mathcal{U}^+ \cap \mathcal{U}^- = \emptyset$). Given a database $\mathcal{DB}$ and a consistent set of update atoms $\mathcal{U}$, we denote as $\mathcal{U}(\mathcal{DB})$ the updated database $\mathcal{DB} \cup \mathcal{U}^+ - \mathcal{U}^-$.

Definition 2. Given a database $\mathcal{DB}$ and a set of integrity constraints $\mathcal{IC}$, a *repair* for $\langle \mathcal{DB}, \mathcal{IC} \rangle$ is a consistent set of update atoms $\mathcal{R}$ such that 1) $\mathcal{R}(\mathcal{DB}) \models \mathcal{IC}$, 2) there is no consistent set of update atoms $\mathcal{U} \subset \mathcal{R}$ such that $\mathcal{U}(\mathcal{DB}) \models \mathcal{IC}$.

Repaired databases are consistent databases, derived from the source database by means of a minimal set of update operations. Given a database $\mathcal{DB}$ and a set

of integrity constraints $\mathcal{IC}$, the set of all possible repairs for $\langle \mathcal{DB}, \mathcal{IC} \rangle$ is denoted as $\mathbf{R}(\mathcal{DB}, \mathcal{IC})$. Observe that the set of possible repairs in the case of constraints containing not range-restricted variables could be infinite.

Given a set of (universally quantified) constraints $\mathcal{IC}$, an integrity constraint $r \in \mathcal{IC}$ and a domain Dom, a ground instantiation of r with respect to Dom can be obtained by replacing variables with constants of Dom and eliminating the quantifier $\forall$. The set of ground instances of r is denoted by $ground(r)$, whereas $ground(\mathcal{IC}) = \bigcup_{r \in \mathcal{IC}} ground(r)$ denotes the set of ground instances of constraints in $\mathcal{IC}$. Clearly, for any set of universally quantified constraints $\mathcal{IC}$, the cardinality of $ground(\mathcal{IC})$ is polynomial in the size of the database (and in the size of Dom).

Theorem 1. *Let $\mathcal{DB}$ be a database, $\mathcal{IC}$ a set of full integrity constraints and $\mathcal{R}$ a repair for $\langle \mathcal{DB}, \mathcal{IC} \rangle$. For each $\pm a(t) \in \mathcal{R}$, let $\mathcal{R}' = \mathcal{R} - \{\pm a(t)\}$, there exists in $ground(\mathcal{IC})$ a ground integrity constraint $\phi \wedge Comp(\pm a(t)) \supset$ such that $\mathcal{R}'(\mathcal{DB}) \models \phi$.* □

The above theorem states that each update atom of a repair is necessary to satisfy at least a ground integrity constraint.

Definition 3. Given a database $\mathcal{DB}$ and a set of integrity constraints $\mathcal{IC}$, an atom A is *true* (resp. *false*) with respect to $\mathcal{IC}$ if A belongs to all repaired databases (resp. there is no repaired database containing A). The atoms which are neither *true* nor *false* are *undefined*. □

Thus, true atoms appear in all repaired databases, whereas undefined atoms appear in a non empty proper subset of repaired databases.

Definition 4. Given a database $\mathcal{DB}$ and a relational query $Q = (g, \mathcal{P})$, the *consistent answer* of the query Q on the database $\mathcal{DB}$, denoted as $Q(\mathcal{DB}, \mathcal{IC})$, gives three sets, denoted as $Q(\mathcal{DB}, \mathcal{IC})^+$, $Q(\mathcal{DB}, \mathcal{IC})^-$ and $Q(\mathcal{DB}, \mathcal{IC})^u$. These contain, respectively, the sets of g-tuples which are *true* (i.e. belonging to $\bigcap_{\mathcal{R} \in \mathbf{R}(\mathcal{DB}, \mathcal{IC})} Q(\mathcal{R}(\mathcal{DB}))$), *false* (i.e. not belonging to $\bigcup_{\mathcal{R} \in \mathbf{R}(\mathcal{DB}, \mathcal{IC})} Q(\mathcal{R}(\mathcal{DB}))$) and *undefined* (i.e. set of tuples which are neither *true* nor *false*). □

3.3 Repairing and Querying Through Stable Models

As shown in [15,16], a relational query over databases with standard constraints can be rewritten into disjunctive query over the same database without constraints. More specifically, it is obtained from the union of the non recursive Datalog query and the disjunctive rules derived from the constraints.

Given a database $\mathcal{DB}$ and a set of integrity constraints $\mathcal{IC}$, the technique derives from $\mathcal{IC}$ a disjunctive program $\mathcal{DP}(\mathcal{IC})$. The repairs for $\mathcal{DB}$ can be derived from the stable models of $\mathcal{DP}(\mathcal{IC}) \cup \mathcal{DB}$, whereas the consistent answers for a query $(g, \mathcal{P})$ can be derived from the stable models of $\mathcal{P} \cup \mathcal{DP}(\mathcal{IC}) \cup \mathcal{DB}$.

Definition 5. Let c be a (range restricted) full integrity constraint of the form

$$(\forall\ X)[\ \bigwedge_{j=1}^{m} b_j(X_j),\quad \bigwedge_{j=m+1}^{n} not\ b_j(X_j),\ \varphi(X_0) \supset\ ] \tag{2}$$

where $X = \bigcup_{j=1}^{m} X_j$ and $X_i \subseteq X$ for $i \in [0..n]$. We denote as $dj(c)$ the rule

$$\bigvee_{j=1}^{m} -b_j(X_j) \vee \bigvee_{j=m+1}^{n} +b_j(X_j) \leftarrow \bigwedge_{j=1}^{m} (b_j(X_j) \vee +b_j(X_j)),\ \bigwedge_{j=m+1}^{n} (not\ b_j(X_j) \vee -b_j(X_j)),\ \varphi(X_0)$$

Given a set $\mathcal{IC}$ of full integrity constraints, we define $\mathcal{DP}(\mathcal{IC}) = \{\ dj(c) \mid c \in \mathcal{IC}\ \} \cup \{\leftarrow -b(X), +b(X) \mid b \text{ is a predicate symbol}\}$. □

In the above definition the variable X in the constraint $\leftarrow -b(X), +b(X)$ denotes a list of k distinct variables with k equal to the arity of b.

Given a database $\mathcal{DB}$, a set of full integrity constraints $\mathcal{IC}$ and a stable model $\mathcal{M}$ of $\mathcal{DP}(\mathcal{IC}) \cup \mathcal{DB}$, the set of update atoms $\mathcal{R}(\mathcal{M}) = \{\pm a(t) \mid \pm a(t) \in \mathcal{M}\}$ is a repair for $\mathcal{DB}$.

Observe that, for every database $\mathcal{DB}$, set of integrity constraints $\mathcal{IC}$, query $Q = (g, \mathcal{P})$ and repaired database $\mathcal{DB}'$ (i) each atom $A \in Q(\mathcal{DB}, \mathcal{IC})^+$ belongs to each stable model of $\mathcal{P} \cup \mathcal{DB}'$ (soundness) and (ii) each atom $A \in Q(\mathcal{DB}, \mathcal{IC})^-$ does not belong to any stable model of $\mathcal{P} \cup \mathcal{DB}'$ (completeness).

4 Active Integrity Constraints

This section presents an extension of integrity constraints that allows the specification for each constraint of the actions which can be performed to make the database consistent. For simplicity of presentation we only consider universally quantified variables, although the framework can be applied also to constraints with existentially quantified variables appearing once in body literals.

4.1 Syntax and Semantics

Definition 6. A (universally quantified) *Active Integrity Constraint* (*AIC*) is of the form

$$(\forall\ X)[\ \bigwedge_{j=1}^{m} b_j(X_j),\quad \bigwedge_{j=m+1}^{n} not\ b_j(X_j),\ \varphi(X_0) \supset \bigvee_{i=1}^{p} \pm a_i(Y_i)\ ] \tag{3}$$

where $X = \bigcup_{j=1}^{m} X_j$, $X_i \subseteq X$ for $i \in [0\,..\,n]$ and $Y_i \subseteq X$ for $i \in [1\,..\,p]$. □

In the above definition the conditions $X = \bigcup_{j=1}^{m} X_j$, $X_i \subseteq X$ for $i \in [0\,..\,n]$ and $Y_i \subseteq X$ for $i \in [1\,..\,p]$ guarantee variables to be range restricted. Given an AIC $r = (\forall X)[\Phi \supset \Psi]$, Φ is called body of r (and is denoted by $Body(r)$), whereas Ψ is called head of r (and is denoted by $Head(r)$).

Example 3. Consider the relation manager mgr of Example 1. The active constraint of Example 2 states that in case of conflicting tuples (i.e. there are two managers managing the same department) we prefer to repair the database by deleting the one having a higher salary, whereas the constraint

$$\forall(E_1,E_2,D,S_1,S_2)[mgr(E_1,D,S_1), mgr(E_2,D,S_2), E_1 \neq E_2 \supset \\ -mgr(E_1,D,S_1) \vee -mgr(E_2,D,S_2)]$$

states that between two different managers of the same department we do not have any preference and, therefore, one of them, selected nondeterministically, can be deleted. □

AICs are constraints specifying actions which can be performed to obtain repairs. Given an AIC r of the form (3) $St(r)$ denotes the standard constraint

$$(\forall\ X)\,[\ \bigwedge_{j=1}^{m} b_j(X_j),\quad \bigwedge_{j=m+1}^{n} not\ b_j(X_j),\ \varphi(X_0) \supset\ ] \tag{4}$$

derived from r by removing the head update atoms. Moreover, for a set of active integrity constraints $\mathcal{IC}$, $St(\mathcal{IC})$ denotes the corresponding set of standard integrity constraints, i.e. $St(\mathcal{IC}) = \{St(r) \mid r \in \mathcal{IC}\}$.

Definition 7. A repair for a database $\mathcal{DB}$ and a set of AICs $\mathcal{IC}$ is any repair for $\langle \mathcal{DB}, St(\mathcal{IC})\rangle$. □

Note that not all repairs contain atoms which can be derived from the active integrity constraints. Thus, we can identify a class of repairs, called *founded*, whose actions can be "derived" from the active integrity constraints.

Example 4. Consider the database $\mathcal{DB} = \{movie(Marshall, Chicago, 2002), director(Stone)\}$ and the constraint

$$\forall(D,T,A)\,[\,movie(D,T,A) \wedge not\ director(D) \supset +director(D)\ ]$$

There are two feasible repairs $\mathcal{R}_1 = \{-movie(Marshall, Chicago, 2002)\}$ and $\mathcal{R}_2 = \{+director(Marshall)\}$, but only $\mathcal{R}_2$ contains updates derived from the active integrity constraint. □

In the following the definition concerning the truth value of ground atoms and ground update atoms, with respect to a database $\mathcal{DB}$,a consistent set of update atoms $\mathcal{U}$ and a founded repair are provided.

Definition 8. Given a database $\mathcal{DB}$ and a consistent set of update atoms $\mathcal{U}$, the truth value of

- a positive ground literal $a(t)$ is *true* w.r.t. $(\mathcal{DB},\mathcal{U})$ if $a(t) \in \mathcal{U}(\mathcal{DB})$,
- a negative ground literal $not\, a(t)$ is *true* w.r.t. $(\mathcal{DB},\mathcal{U})$ if $a(t) \notin \mathcal{U}(\mathcal{DB})$,
- a ground update atom $\pm a(t)$ is *true* w.r.t. $(\mathcal{DB},\mathcal{U})$ if $\pm a(t) \in \mathcal{U}$,
- built-in atoms, conjunctions and disjunctions of literals is given in the standard way,
- a ground AIC $\phi \supset \psi$ is *true* w.r.t. $(\mathcal{DB},\mathcal{U})$ if ϕ is *false* w.r.t. $(\mathcal{DB},\mathcal{U})$. □

Definition 9. Let $\mathcal{DB}$ be a database, $\mathcal{IC}$ a set of AICs and $\mathcal{R}$ a repair for $\langle \mathcal{DB}, \mathcal{IC} \rangle$.

- A ground update atom $\pm a(t) \in \mathcal{R}$ is *founded* if there exists $r \in ground(\mathcal{IC})$ s.t. $\pm a(t)$ appears in $Head(r)$ and $Body(r)$ is *true* w.r.t. $(\mathcal{DB}, \mathcal{R} - \{\pm a(t)\})$. We say that $\pm a(t)$ is *supported* by r.
- A ground rule $r \in ground(\mathcal{IC})$ is *applied* w.r.t. $(\mathcal{DB}, \mathcal{R})$ if there exists $\pm a(t) \in \mathcal{R}$ s.t. $\pm a(t)$ appears in $Head(r)$ and $Body(r)$ is *true* w.r.t. $(\mathcal{DB}, \mathcal{R} - \{\pm a(t)\})$. We say that r *supports* $\pm a(t)$.
- $\mathcal{R}$ is *founded* if all its atoms are *founded*.
- $\mathcal{R}$ is *unfounded* if it is not founded. □

The set of founded update atoms in $\mathcal{R}$ with respect to $\langle \mathcal{DB}, \mathcal{IC} \rangle$ is denoted as $Founded(\mathcal{R}, \mathcal{DB}, \mathcal{IC})$, whereas $Unfounded(\mathcal{R}, \mathcal{DB}, \mathcal{IC}) = \mathcal{R} - Founded(\mathcal{R}, \mathcal{DB}, \mathcal{IC})$. The set of applied rules in $ground(\mathcal{IC})$ is denoted as $Applied(\mathcal{R}, \mathcal{DB}, \mathcal{IC})$, whereas $Unapplied(\mathcal{R}, \mathcal{DB}, \mathcal{IC}) = ground(\mathcal{IC}) - Applied(\mathcal{R}, \mathcal{DB}, \mathcal{IC})$. Thus, update atoms of founded repairs are inferable by means of AICs. Given a database $\mathcal{DB}$ and a set of AICs $\mathcal{IC}$, $\mathbf{FR}(\mathcal{DB}, \mathcal{IC})$ denotes the set of founded repairs for $\langle \mathcal{DB}, \mathcal{IC} \rangle$. Clearly, the set of founded repairs is contained in the set of repairs ($\mathbf{FR}(\mathcal{DB}, \mathcal{IC}) \subseteq \mathbf{R}(\mathcal{DB}, St(\mathcal{IC}))$).

Example 5. Consider the following set of AICs $\mathcal{IC}$

$$\forall(E, P, D)[\, mgr(E, P),\ prj(P, D),\ not\ emp(E, D) \supset\ +emp(E, D)\,]$$

$$\forall(E, D_1, D_2)[\, emp(E, D_1),\ emp(E, D_2),\ D_1 \neq D_2 \supset -emp(E, D_1) \vee -emp(E, D_2)\,]$$

The first constraint states that every manager E of a project P carried out by a department D must be an employee of D, whereas the second one says that every employee must be in only one department. Consider now the database $\mathcal{DB} = \{mgr(e_1, p_1),\ prj(p_1, d_1),\ emp(e_1, d_2)\}$. There are three repairs for $\mathcal{DB}$: $\mathcal{R}_1 = \{-mgr(e_1, p_1)\}$, $\mathcal{R}_2 = \{-prj(p_1, d_1)\}$ and $\mathcal{R}_3 = \{+emp(e_1, d_1), -emp(e_1, d_2)\}$. $\mathcal{R}_3$ is the only founded repair as only the update atoms $+emp(e_1, d_1)$ and $-emp(e_1, d_2)$ are derivable from $\mathcal{IC}$. □

Theorem 2. *Let $\mathcal{DB}$ be a database, $\mathcal{IC}$ a set of AICs and $\mathcal{R}$ a founded repair for $\langle \mathcal{DB}, \mathcal{IC} \rangle$. For each ground AIC $r = \phi \supset \psi \in Applied(\mathcal{R}, \mathcal{DB}, \mathcal{IC})$, let $\mathcal{U}$ be the set of ground update atoms appearing in ψ, then $\mathcal{U}(\mathcal{DB}) \not\models \phi$ (i.e. $\mathcal{U}(\mathcal{DB}) \models St(r)$).* □

The above theorem states that for each ground applied constraint there must be among the true update head atoms, at least one atom $\pm a(y)$ which makes the body of the active constraint false with respect to the repaired database, i.e. the body must contain a literal $Comp(\pm a(y))$. Observe that, if for each ground AIC $r = \phi \supset \psi \in ground(\mathcal{IC})$ the set $\mathcal{U}$ of ground update atoms appearing in ψ is such that $\mathcal{U}(\mathcal{DB}) \models \phi$ (i.e. $\mathcal{U}(\mathcal{DB}) \not\models St(r)$), no founded repair exists.

Corollary 1. *Given a database $\mathcal{DB}$ and a set of AICs $\mathcal{IC}$, an update atom $\pm a(t)$ can belong to a founded repair only if there exists $r \in ground(\mathcal{IC})$ such that $\pm a(t)$ appears in $Head(r)$ and $Comp(\pm a(t))$ appears in $Body(r)$.* □

Definition 10. Given a ground AIC $r = \phi \supset \psi$, $Core(r)$ denotes the ground AIC $\phi \supset \psi'$, where ψ' is obtained by deleting from ψ any update atom $\pm a(t)$ such that $Comp(\pm a(t))$ does not appear in ϕ. Given a set $\mathcal{IC}$ of AICs, $Core(\mathcal{IC}) = \{Core(r) \mid r \in ground(\mathcal{IC})\}$. □

Theorem 3. *Given a database $\mathcal{DB}$ and a set $\mathcal{IC}$ of AICs,*

$$\mathbf{FR}(\mathcal{DB}, \mathcal{IC}) = \mathbf{FR}(\mathcal{DB}, Core(\mathcal{IC})).$$

The above results (Theorem 2, Corollary 1 and Theorem 3) state that every head update atom $\pm a(t)$ not repairing the body (i.e. such that the body does not contain a literal $Comp(\pm a(t))$) is useless and can be deleted. This is an important result suggesting that active rules (with the declarative semantics here proposed), used to repair databases, should have a specific form: the head update atoms must repair the database so that the body of the active constraint is *false* (i.e. the constraint is satisfied).

Example 6. Consider the database $\mathcal{DB} = \{a, b\}$ and the set $\mathcal{IC} = \{a \supset -b, b \supset -a\}$ of AICs. The unique repair for $\langle \mathcal{DB}, \mathcal{IC} \rangle$ is $\mathcal{R} = \{-a, -b\}$, but it is not founded. Intuitively, if we apply $a \supset -b$, b is deleted from $\mathcal{DB}$, so $b \supset -a$ cannot be applied. If we apply $b \supset -a$, a is deleted from $\mathcal{DB}$, so $a \supset -b$ cannot be applied. □

From the previous results, in the following, only AICs in which for each head update atom $\pm a(t)$, there exists in the body a corresponding complementary literal $Comp(\pm a(t))$, are considered.

Theorem 4. *Let $\mathcal{DB}$ be a database, $\mathcal{IC}$ a set of AICs and $\mathcal{R}$ a founded repair for $\langle \mathcal{DB}, \mathcal{IC} \rangle$. For each $\pm a(t) \in \mathcal{R}$ there exists a ground AIC $\rho \in Core(\mathcal{IC})$ s.t. $\pm a(t)$ is the unique update atom in $\mathcal{R}$ supported by ρ.* □

Next example shows how AICs can be used to express classical hard problems.

Example 7. Graph coloring. The following set of constraints $\mathcal{IC}$ checks if the coloring of a (possibly partially colored) graph, defined by means of the relations *node* and *edge*, can be completed by using only the two colors *red* and *blue*.

$$\forall(X)[\, node(X),\ not\, col(X, red),\ not\, col(X, blue),\ not\, col(X, yellow) \supset +col(X, red) \vee +col(X, blue)\,]$$

$$\forall(X, Y, C)[\, edge(X, Y),\ col(X, C),\ col(Y, C) \supset -col(X, C) \vee -col(Y, C)\,]$$

The two constraints state that colored nodes can be (re-)colored with one of two available colors. □

Observe that in the above example if the head update atoms are removed from the second constraint, as colored nodes cannot be re-colored, the expressed problem consists in completing the coloring of the graph. Assuming that the input

graph is not colored, the classical 3-coloring problem constraints can be defined by the following constraints:

$$\forall(X)[\,node(X),\ not\,col(X,red),\ not\,col(X,blue),\ not\,col(X,yellow) \supset \\ +col(X,red) \vee +col(X,blue) \vee +col(X,yellow)\,]$$

$$\forall(X,Y,C)[\,edge(X,Y),\ col(X,C),\ col(Y,C) \supset\,]$$

It is worth noting that the same problem cannot be expressed using not founded repairs as a repair can also be obtained by deleting nodes from the input graph. The problem with active integrity constraints is that the existence of founded repairs, in the general case, is not guaranteed. Thus, Section 5 will present a different semantics where founded repairs can be considered as repairs which are preferable with respect to the not founded ones as they contain only actions derived from the active constraints.

4.2 Rewriting into Logic Programs

The technique introduced in [15,16] cannot be applied to AICs. Consider for instance the database $\mathcal{DB} = \{a, b\}$ and the set $\mathcal{IC}$ containing AICs $a \supset -a$ and $a, b \supset -b$. The database $\mathcal{DB}$ is inconsistent and the unique repairs is $\mathcal{R} = \{-a\}$. Moreover, the program $\mathcal{DP}(\mathcal{IC})$ consists of the rules $-a \leftarrow (a \vee +a)$ and $-b \leftarrow (a \vee +a), (b \vee +b)$. The program $\mathcal{DP}(\mathcal{IC}) \cup \mathcal{DB}$ has a unique stable model $\mathcal{M} = \{-a, -b, a, b\}$ from which we derive the set of updates $\mathcal{R}(\mathcal{M}) = \{-a, -b\}$ which is not a repair.

A different technique will now be shown which generalizes the one proposed in [15,16] so that repairs can be produced by logic programs derived from rules defining integrity constraints. It is worth noting that the presence of existentially quantified variables in negated body literals, does not allow the generation of a possibly infinite number of repairs as the logic rules derived from the rewriting of constraints are *safe* [25].

Given a set $\{ic_1, \ldots, ic_r, \ldots, ic_k\}$ of ground AICs and a ground update atom $+a(t)$ (resp. $-a(t)$), we use the following notation:

- $a^+(t, r)$ (resp. $a^-(t, r)$) means that the update $+a(t)$ (resp. $-a(t)$) is performed by ic_r. We call $a^+(t, r)$ (resp. $a^-(t, r)$) a *marked update atom.*
- $\overline{a^+}(t, r)$ (resp. $\overline{a^-}(t, r)$) means that the update $+a(t)$ (resp. $-a(t)$) is performed by a ground AIC different from ic_r.

Definition 11. Given a database $\mathcal{DB}$, a set $\mathcal{IC} = \{ic_1, \ldots, ic_k\}$ of ground AICs, and a founded repair $\mathcal{R} = \{\pm a_1(t_1), \ldots, \pm a_n(t_n)\}$ for $\langle \mathcal{DB}, \mathcal{IC} \rangle$,

- a *marked founded repair* derived from $\mathcal{R}$ is a set of marked update atoms $\mathcal{MR} = \{a_1^{\pm}(t_1, r_1), \ldots, a_n^{\pm}(t_n, r_n)\}$ s.t.
  - $\forall\ a_i^{\pm}(t_i, r_j) \in \mathcal{MR}$, $\pm a_i(t_i)$ is supported by ic_j,
  - $r_i \neq r_j$ for $i, j \in [1..n]$ and $i \neq j$.
- the mapping between $\mathcal{R}$ and the set of marked founded repairs derived from $\mathcal{R}$ is defined by means of a (multivalued) *marking function* γ. □

Thus, $\gamma(\mathcal{R})$ denotes the set of marked founded repairs derived from $\mathcal{R}$ (it is here assumed that the database and the active integrity constraints are understood). We define the set of marked founded repairs for $\langle \mathcal{DB}, \mathcal{IC} \rangle$: $\mathbf{MFR}(\mathcal{DB}, \mathcal{IC}) = \bigcup_{\mathcal{R} \in \mathbf{FR}(\mathcal{DB}, \mathcal{IC})} \gamma(\mathcal{R})$.

Example 8. Consider the database $\mathcal{DB} = \{a, b\}$ and the set $\{ic_1, ic_2\} = \{a \supset -a,\ \ a \wedge b \supset -a\}$ of AICs. There exists only the founded repair $\mathcal{R} = \{-a\}$. As the update atom $-a$ is supported by both AICs, there are two possible marked founded repairs derived from $\mathcal{R}$: $\mathcal{MR}_1 = \{a^-(1)\}$ and $\mathcal{MR}_2 = \{a^-(2)\}$ stating, respectively, that the deletion of the atom a is associated with the first and second constraints. □

The existence of at least a marked founded repair for each founded repair is guaranteed by the following corollary.

Corollary 2. *Given a database $\mathcal{DB}$ and a set $\mathcal{IC}$ of ground AICs, for each founded repair $\mathcal{R}$, $\gamma(\mathcal{R}) \neq \emptyset$.* □

The following definition shows how active integrity constraints are rewritten into Datalog programs.

Definition 12. Let $\mathcal{IC}$ be a set of AICs and $\{ic_1, \ldots, ic_r, \ldots, ic_k\}$ its ground version w.r.t. a database $\mathcal{DB}$, where

$$ic_r = \bigwedge_{j=1}^{m} b_j(x_j),\ \ \bigwedge_{j=m+1}^{n} not\ b_j(x_j),\ \varphi(x_0) \supset \bigvee_{i=1}^{p} \pm a_i(y_i).$$

We define $Rew(ic_r) = Rew^0(ic_r) \cup Rew^1(ic_r) \cup Rew^2(ic_r)$, where $Rew^0(ic_r)$ is the set of rules

$$\begin{array}{lll}
\mathbf{r.1}: & \bigvee_{i=1}^{p} a_i^{\pm}(y_i, r) \leftarrow \bigwedge_{j=1}^{m} \widetilde{b_j}(x_j, r),\ \bigwedge_{j=m+1}^{n} not\ \widetilde{b_j}(x_j, r),\ \varphi(x_0) & \\
\mathbf{r.2.j}: & \widetilde{b_j}(x_j, r) \leftarrow (b_j(x_j),\ not\ \overline{b_j^-}(x_j, r)) \vee \overline{b_j^+}(x_j, r) & j \in [1..n] \\
\mathbf{r.3.i}: & a_i^{\pm}(y_i, l) \leftarrow a_i^{\pm}(y_i, r),\ 1 \leq l \leq k,\ r \neq l & i \in [1..p]
\end{array}$$

$Rew^1(ic_r)$ is the set of rules

$$\begin{array}{lll}
\mathbf{r.4}: & \leftarrow \bigwedge_{j=1}^{m} \widehat{b_j}(x_j),\ \bigwedge_{j=m+1}^{n} not\ \widehat{b_j}(x_j),\ \varphi(x_0) & \\
\mathbf{r.5.j}: & \widehat{b_j}(x_j) \leftarrow (b_j(x_j),\ not\ -b_j(x_j)) \vee +b_j(x_j) & j \in [1..n]
\end{array}$$

and $Rew^2(ic_r)$ is the set of rules

$$\begin{array}{lll}
\mathbf{r.6.i}: & \pm a_i(y_i) \leftarrow a_i^{\pm}(y_i, r) & i \in [1..p] \\
\mathbf{r.7.i}: & \leftarrow +a_i(y_i), -a_i(y_i) & i \in [1..p].
\end{array}$$

We define $Rew^u(\mathcal{IC}) = \bigcup_{i=1}^{k} Rew^u(ic_i)$, with $u \in \{0, 1, 2\}$, and $Rew(\mathcal{IC}) = \bigcup_{i=1}^{k} Rew(ic_i)$. □

The rules in $Rew^0(\mathcal{IC})$ are used to compute stable models corresponding to sets of updates, whereas the rules in $Rew^1(\mathcal{IC})$ and in $Rew^2(\mathcal{IC})$ check that the stable models of $Rew^0(\mathcal{IC})$ define (consistent) database repairs. Intuitively,

the atom $\widetilde{b_j}(x_j, r)$ states that the atom $b_j(x_j)$ is present in the database if ic_r doesn't perform any update actions, whereas the atom $\widehat{b_j}(x_j)$ expresses the fact that the atom $b_j(x_j)$ is present in the database after all update action have been performed. Rule **r.1** declares that if the constraint $St(ic_r)$ is violated before any update actions is performed by ic_r, ic_r has to perform an update action. The denial **r.4** (the original integrity constraint defined over the updated database) is added in order to guarantee that the updated database satisfies ic_r.

Proposition 1. Given a database $\mathcal{DB}$, a set $\mathcal{IC}$ of AICs, a model $\mathcal{M}$ of $\mathcal{DB} \cup Rew(\mathcal{IC})$ and an atom $a^{\pm}(t, r) \in \mathcal{M}$, $\mathcal{M}$ does not contain any atom $a^{\pm}(t, l)$ with $r \neq l$. □

Definition 13. Given an interpretation $\mathcal{M}$, we denote as $UpdateAtoms(\mathcal{M})$ the set of update atoms in $\mathcal{M}$ and as $MarkedUpdateAtoms(\mathcal{M})$ the set of marked update atoms in $\mathcal{M}$. Given a set S of interpretations, we define $UpdateAtoms(S) = \{UpdateAtoms(\mathcal{M}) \mid \mathcal{M} \in S\}$ and $MarkedUpdateAtoms(S) = \{MarkedUpdateAtoms(\mathcal{M}) \mid \mathcal{M} \in S\}$. □

Next theorem shows the equivalence between (marked) founded repairs and stable models, restricted to (marked) update atoms.

Theorem 5. *Given a database $\mathcal{DB}$ and a set $\mathcal{IC}$ of AICs*
1. $\mathbf{MFR}(\mathcal{DB}, \mathcal{IC}) = MarkedUpdateAtoms(\mathcal{SM}(Rew(\mathcal{IC}) \cup \mathcal{DB}))$,
2. $\mathbf{FR}(\mathcal{DB}, \mathcal{IC}) = UpdateAtoms(\mathcal{SM}(Rew(\mathcal{IC}) \cup \mathcal{DB}))$. □

It is worth noting that given a stable model $\mathcal{M}$ of $Rew(\mathcal{IC}) \cup \mathcal{DB}$ and a marked update atom $a^{\pm}(t, r) \in \mathcal{M}$, $\mathcal{M}$ does not contain any other atom $b^{\pm}(v, r)$ different from $a^{\pm}(t, r)$. In fact, $a^{\pm}(t, r)$ and $b^{\pm}(v, r)$ can be inferred only by the rule **r.1**, and $\mathcal{M}$ is not minimal if it contains both atoms. From this observation it follows that rule **r.1** can be rewritten using exclusive disjunction in the head, i.e.

$$\mathbf{r.1}: \bigoplus_{i=1}^{p} a_i^{\pm}(y_i, r) \leftarrow \bigwedge_{j=1}^{m} \widetilde{b_j}(x_j, r), \quad \bigwedge_{j=m+1}^{n} not\ \widetilde{b_j}(x_j, r),\ \varphi(x_0)$$

Data Complexity

Theorem 6. *Let $\mathcal{DB}$ be a database and $\mathcal{IC}$ a set of active integrity constraints. The problem of checking if there exists a founded repair $\mathcal{R}$ for $\mathcal{DB}$ is Σ_2^p-complete.* □

The consistent founded answer to a relational query $Q = (g, \mathcal{P})$ over a database $\mathcal{DB}$ with active constraints $\mathcal{IC}$ (denoted by $Q(\mathcal{DB}, \mathcal{IC})$), is obtained by first computing the set $\mathbf{FR}(\mathcal{DB}, \mathcal{IC})$ of founded repairs for $\mathcal{DB}$ and, then, considering the intersection $\bigcap_{\mathcal{R} \in \mathbf{FR}(\mathcal{DB}, \mathcal{IC})} Q(\mathcal{R}(\mathcal{DB}))$.

Theorem 7. *Let $\mathcal{DB}$ be a database and $\mathcal{IC}$ a set of active integrity constraints. The problem of checking whether a ground atom g belongs to all repaired databases obtained by means of founded repairs is Π_2^p-complete.* □

For single head active integrity constraints the complexity is in the first level of the polynomial hierarchy.

5 Preferred Repairs and Answers

In this section we define an approach that always permits us to obtain a consistent repaired database. In particular, we interpret the actions in the head of constraints as indication of the operations the user prefers to perform to make the database consistent.

Definition 14. Let $\mathcal{DB}$ be a database, $\mathcal{IC}$ a set of active integrity constraints and $\mathcal{R}_1, \mathcal{R}_2$ two repairs for $\mathcal{DB}$. Then, $\mathcal{R}_1$ is *preferable* to $\mathcal{R}_2$ ($\mathcal{R}_1 \sqsupset \mathcal{R}_2$) w.r.t. $\mathcal{IC}$, if $Unfounded(\mathcal{R}_1, \mathcal{DB}, \mathcal{IC}) \subset Unfounded(\mathcal{R}_2, \mathcal{DB}, \mathcal{IC})$. Moreover, $R_1 \sqsupset R_2$ if $R_1 \sqsupseteq R_2$ and $R_2 \not\sqsupseteq R_1$. A repair $\mathcal{R}$ is said to be *preferred* w.r.t. $\mathcal{IC}$ if there is no repair $\mathcal{R}'$ such that $\mathcal{R}' \sqsupset \mathcal{R}$. □

Example 9. Consider the integrity constraint of Example 2 with the database $\mathcal{DB} = \{mgr(john, b, 1000), mgr(frank, b, 2000), mgr(mary, c, 1000), mgr(rosy, c, 2000)\}$. There are four repairs $\mathcal{R}_1 = \{-mgr(john, b, 1000), -mgr(mary, c, 1000)\}$, $\mathcal{R}_2 = \{-mgr(john, b, 1000), -mgr(rosy, c, 2000)\}$, $\mathcal{R}_3 = \{-mgr(frank, b, 2000), -mgr(mary, c, 1000)\}$ and $\mathcal{R}_4 = \{-mgr(frank, b, 2000), -mgr(rosy, c, 2000)\}$. The order relation is $\mathcal{R}_2 \sqsupset \mathcal{R}_1$, $\mathcal{R}_3 \sqsupset \mathcal{R}_1$, $\mathcal{R}_4 \sqsupset \mathcal{R}_2$ and $\mathcal{R}_4 \sqsupset \mathcal{R}_3$. Therefore, we have only one preferred model which is also founded (namely $\mathcal{R}_4$). Assume now to also have the constraint

$$not\ mgr(rosy, c, 2000) \supset$$

declaring that the tuple $mgr(rosy, c, 2000)$ must be in $\mathcal{DB}$. In such a case we have only the two repairs $\mathcal{R}_1$ and $\mathcal{R}_3$ and the preferred one is $\mathcal{R}_3$ which is not founded. □

The relation $\sqsupset$ is a *partial order* as it is irreflexive, asymmetric and transitive. The set of preferred repairs for a database $\mathcal{DB}$ and a set of active integrity constraints $\mathcal{IC}$ is denoted by $\mathbf{PR}(\mathcal{DB}, \mathcal{IC})$. Clearly, the relation between preferred, founded and standard repairs is as follows: $\mathbf{FR}(\mathcal{DB}, \mathcal{IC}) \subseteq \mathbf{PR}(\mathcal{DB}, \mathcal{IC}) \subseteq \mathbf{R}(\mathcal{DB}, \mathcal{IC})$. The next theorem states the precise relation between preferred, founded and general repairs.

Theorem 8. *Let $\mathcal{DB}$ be a database and $\mathcal{IC}$ a set of active integrity constraints, then*

$$\mathbf{PR}(\mathcal{DB}, \mathcal{IC}) \begin{cases} = \mathbf{FR}(\mathcal{DB}, \mathcal{IC}) & if\ \mathbf{FR}(\mathcal{DB}, \mathcal{IC}) \neq \emptyset \\ \subseteq \mathbf{R}(\mathcal{DB}, \mathcal{IC}) & if\ \mathbf{FR}(\mathcal{DB}, \mathcal{IC}) = \emptyset \end{cases}$$

□

Obviously, as the existence of repairs is guaranteed, the existence of a preferred repair is guaranteed too. We conclude by presenting a result on the computational complexity of computing preferred repairs and answers.

Theorem 9. *Let $\mathcal{DB}$ be a database and $\mathcal{IC}$ a set of active integrity constraints*

1. *checking if there exists a preferred founded repair $\mathcal{R}$ for $\mathcal{DB}$ is Σ_2^p-complete;*
2. *checking whether a ground atom belongs to all preferred repairs is Π_2^p-complete.* □

The above theorem states that computing preferred repairs and answers is not harder than computing standard or founded repairs and answers.

References

1. Abiteboul S., Hull R., and Vianu V., *Foundations of Databases*. Addison-Wesley, 1994.
2. Alferes J. J., J. A. Leite, Pereira L. M., Przymusinska H., and Przymusinski T.C., Dynamic updates of non-monotonic knowledge bases. *JLP*, 45(1-3), 43–70, 2000.
3. Arenas M., Bertossi L., and Chomicki J., Consistent query answers in inconsistent databases. *Proc. PODS*, 68–79, 1999.
4. Arenas M., Bertossi L., and Chomicki J., Specifying and querying database repairs using logic programs with exceptions. *Proc. FQAS*, 27–41, 2000.
5. Baral C., Embedding revision programs in logic programming situation calculus. *Journal of Logic Programming*, 30(1), 83–97, 1997.
6. Ceri S., Widom J., Deriving Production Rules for Constraint Maintenance, *VLDB*, 566-577, 1990.
7. Chomicki J., Lobo J., and Naqvi S. A., Conflict resolution using logic programming. *IEEE TKDE*, 15(1), 244–249, 2003.
8. Chomicki J., Marcinkowski J., Minimal-change integrity maintenance using tuple deletions. *Information & Computation*, 197(1-2), 90-121, 2005.
9. Eiter T., Gottlob G., and Mannila H., Disjunctive datalog. *ACM TODS*, 22(3), 364–418, 1997.
10. Flesca, S., Greco, S., Declarative semantics for active rules, *TPLP*, 1(1), 43-69, 2001.
11. Flesca S., Greco S., Zumpano E., Active integrity constraints. *PPDP*, 98-107, 2004.
12. Gelfond M. and Lifschitz V. The stable model semantics for logic programming. *Proc. ICLPS*, 1070–1080, 1988.
13. Gelfond M. and Lifschitz V. Classical negation in logic programs and disjunctive databases. *New generation Computing*, 9(3/4), 365–385, 1991.
14. Grant J. and Subrahmanian V. S., Reasoning in inconsistent knowledge bases. *IEEE TKDE*, 7(1), 177–189, 1995.
15. Greco S., and Zumpano E., Querying Inconsistent Databases. *LPAR*, 2000.
16. Greco G., Greco S., and Zumpano E., A Logical Framework for Querying and Repairing Inconsistent Databases. *IEEE TKDE*, 15(6), 1389-1408, 2003.
17. Kifer M. and Li A., On the semantics of rule-based expert systems with uncertainty. *Proc. ICDT*, 102–117, 1988.
18. Lin J., A semantics for reasoning consistently in the presence of inconsistency. *Artificial Intelligence*, 86(1), 75–95, 1996.
19. Marek V. W., Pivkina I., and Truszczynski M., Revision programming = logic programming + integrity constraints. In *Computer Science Logic*, 73–98, 1998.
20. Marek V. W. and Truszczynski M., Revision programming. *TCS*, 190(2), 241–277, 1998.
21. May W., Ludascher B., Understanding the Global Semantics of Referential Actions using Logic Rules, *ACM TODS* 27(4), 343-397, 2002.
22. Papadimitriou, C. H., *Computational Complexity*. Addison-Wesley, 1994.
23. Paton N. W., Diaz O., Active Database Systems, *ACM Computing Surveys*, 31(1), 63-103, 1999
24. Subrahmanian V. S., Amalgamating knowledge bases. *ACM TKDE*, 19(2), 291–331, 1994.
25. Ullman J. K., *Principles of Database and Knowledge-Base Systems*. Computer Science Press, 1998.
26. Wijsen J., Condensed representation of database repairs for consistent query answering. *ICDT*, 378–393, 2003.