Minimal and Consistent Evolution of Knowledge Bases*

Jorge Lobo and Goce Trajcevski
Department of EECS
University of Illinois at Chicago
851 S. Morgan Street
Chicago IL 60607
{jorge,goce}@guernica.eecs.uic.edu
tel:312-996-5340, fax:312-413-0024

Abstract

There are few important problems associated with the process of assimilating new knowledge in knowledge bases/deductive databases. One of them is maintaining the *consistency* of the data (usually specified by a set of Integrity Constraints). Another problem arises due to the fact that updates of intentionally specified relations through modifications of base facts are usually not unique. Keeping all the possibilities creates a storage problems and highly affects the efficiency of query processing. An important practical aspect is the domain issue. Transactions should be able to modify the active domain (add new constants in the knowledge base; delete some of the present) as opposed to domain independency of the regular query processing.Yet another subtle issue is the notion of minimality of the modifications - it is undesirable to insert or delete redundant set of facts in order to achieve the update request.

In this work we propose efficient procedures for bringing a knowledge base "up to date" i.e. a new state in which given update request is satisfied in such a manner that the *consistency* is maintained and the modifications with respect to the old state are *minimal*. We also address a few subtle issues of the problem of consistent updates which are important for practical purposes:
– we discuss a notion of minimality which is both defined as a natural partial order over possible models of the database and expresses a preference of maintaining as much as possible of the accumulated base knowledge. This requirement seems rational for many applications and yet it is hard to construct a set of integrity constraints that would specify it as a consistency requirement;
– we handle *safe negation* and *local* variables in the bodies of the rules. We also demonstrate how interaction with the user can be used to handle *recursive* definitions.

Keywords: Knowledge base, knowledge assimilation, view updates, consistency maintenance.

*This work has been partially supported by the NSF under grant No. IRI-9210220

1 Introduction

Knowledge bases are (partial) models of a given (mini)world which use some modeling mechanism to store, retrieve and update the relevant data about the world which they model. The valid/consistent states of the knowledge base (the world being modeled) are usualy specified by the set of *integrity constraints*. In case the undrelying modeling mechanism is based on the relational model [10], the semantics of the database is described by the *database schema*. Deductive databases generalize the relational ones because they include *deductive rules* which can be used to derive new facts from the base ones, explicitly/physically stored on some media. However, when a knowledge base needs to be brought up to date with respect to some intentionaly specified predicate (view update), the actual modifications must be performed on the base (extensional) predicates. There are few important problems which arise when a new knowledge is to be assimilated into a knowledge base:

• Given a current state of the knowledge base, the assimilation of the new fact might create a new updated state which is *inconsistent* with respect to the set of integrity constraint. Instead of rejecting the update request, in this work we propose efficient methods which will bring the knowledge base "up to date" and preserve its *consistency*. However, in order to ensure this, some extra modifications (insertions and/or deletions) might be required. Our method guarantees that the overall modifications of the knowledge base are *minimal*.

• Another important class of problems stems from the fact that, when the assimilation of new knowledge involves intentionaly specified predicates, the modifications of the base relations which accomplish the view update request are usually not unique[1]. For practical reasons (complicated syntax, data structures, efficiency of storage and query processing) [41, 49] it is desirable that the database is kept *definite*. Thus, most frequently, in case there are few possibilities to realize an update, only one is finally chosen, either non–deterministically, or with help from the user.

• Closely associated, in the sense of interaction with the user, are the issues of *flexibility* and the *domain*. As pointed in [9, 45], transactions should be able to modify the active domain of the database, which means some new constants that were not present before might appear after the update and some constants might get deleted (contrast this with *domain independency* of the regular query processing). Consider the following:

Example 1.1. Let the deductive database consist of:

[1]Some options might get eliminated due to integrity constraints satisfaction or other semantically/syntactically established minimality criteria but, in general, the problem of non–uniqueness remains.

$$\begin{array}{llllll}\textbf{IDB:} & p \leftarrow q(X), r(X) & \textbf{EDB:} & q(a) & \textbf{IC:} & \leftarrow q(X), s(X) \\ & p \leftarrow q(X), m(X) & & m(b) & & \\ & p \leftarrow n(X), s(X) & & & & \end{array}$$

If the update request is *insert* p, the user should be allowed to achieve it by inserting $q(c)$ and $r(c)$ or by inserting $q(d)$ and $m(d)$. The set of inserted facts is *minimal*– in a sense that no redundant information is inserted. However, if the user wants to accomplish the insertion of p by inserting $n(a)$ and $s(a)$, the system should be able to detect that, since $q(a)$ is present in the *EDB*, the IC is invalidated. Thus, the correct transaction[2] which would insert p should be: *insert* $\{n(a), s(a)\}$; *delete* $\{q(a)\}$. Moreover, if the insertion of p is attempted by inserting $q(b)$ and $r(b)$, satisfying the first rule in the definition of p, due to the presence of $m(b)$ in the *EDB* and the second rule in the definition of p, the system should be able to detect that the insertion of $q(b)$ is sufficient[3].

• Another subtle issue is the notion of *minimality*. Minimality is commonly defined using the *symmetric difference (SD)* order[4] between databases [2, 6, 13, 16, 37, 38]. However, there are occasions when *SD* does not seem appropriate.

Example 1.2. Consider a simplified instance of a deductive database, used for reasoning about relations between drugs and therapies:

$$IDB = \{\ \textit{add\_th(Drug,Therapy)} \leftarrow \textit{contains(Drug,Substance)}, \neg\ \textit{react(Substance,Substance\_1)}, \textit{uses(Substance\_1,Therapy)}.\ \}.$$

$$EDB = \{contains(a,c), contains(a,d), react(m,n), uses(b,g)\}.$$

Let the update request be *insert add_th(e,f)*. Clearly, the insertion is satisfied in $EDB' = (EDB \setminus \{react(m,n)\}) \cup \{contains(e,m), uses(n,f)\}$. Observe that the same task can be achieved in $EDB_n = EDB \cup \{contains(e,h), uses(l,f)\}$.

Within the *SD* order, both modifications are minimal. But despite the syntactical minimality of EDB', it requires a deletion of a tuple *react(m,n)*, which is an established fact that the two substances, m and n react. In general, it is not straightforward to construct a set of integrity constraints that would express a preference of maintaining the accumulated knowledge as long as possible. On the

[2] Instead of just rejecting the insertion of p via insertions of $n(a)$ and $s(a)$ because of the inconsistent updated state of the knowledge base

[3] In Section 3 we discuss the issue of extracting the information relevant for the update, from the current state of the database.

[4] Briefly, any transaction (a set of insertions and deletions) such that no subset of it can achieve the update is minimal (see Section 3).

other hand, a requirement like that seems rational in many applications[5]. The situation arises because in practice, the negation is often used to encode *exceptional* information but, it is deduced via default rules.

Efficient solutions to the problem of minimal view updates are also important for areas such as truth maintenance systems, hypothetical and non-monotonic reasoning, diagnosis and planning. The main contributions of our approach are:

– A view update algorithm that produces minimal changes, where the *minimality* is defined in terms of a natural partial order over the databases. The complexity of our method is within the same bounds as the methods that produce one solution [2, 6, 37, 38] but do not ensure minimality. We also demonstrate the completeness of our approach (i.e. any minimal solution can be generated).

– The algorithm applies to stratified deductive databases with *safe* negation which is, any variable that appears in a negative literal in the body of the rule, must also appear in some positive literal. For this case we demonstrate that we can handle *local* variables (i.e. the ones that appear only in the body of a rule). The existing solutions either do not consider negation at all [6, 37, 38]; or lack a formal treatment of the local variables [2, 13]. We incorporate the *integrity constraints maintenance* within the update process and guarantee the overall minimality without redundant work. Moreover, we demonstrate that through iterative interaction with the user we can obtain a minimal update for the case of recursive definitions as well.

– Since there could potentially be an exponential number of possible updates, instead of generating all of them and then make a selection at the end, we incrementally generate a sequence of insertions and deletions that will be applied to the *EDB* to create the updated database which maintains the satisfaction of the set of integrity constraints. During this process, questions will be asked to the user to resolve any ambiguity. By querying the user, we add flexibility and still guarantee that, modulo user's choice, we will not do any redundant work. Moreover, combining safe negation and interaction with the user we avoid high complexity problems. Sometimes, given a particular *EDB*, the interaction with the user is inevitable, if the update request needs values which do not appear in any tuple[6] (cf. [46, 33]). This would be the case if in the Example 1.1. above, the *EDB* were empty.

[5] Observe that the constants h and l are *not* from the active domain.

[6] If we allow for some type of *Skolem* constants, we introduce incompleteness in the database, consequently we loose its definiteness.

1.1 General remarks

In general, updates might not be achievable. Let, for example $IDB = \{p(X) \leftarrow c(X,Y), q(X,Z)\}$, $EDB = \{c(m,n)\}$ where c and d are the only base relations. Due to the presence of $q(X,Z)$ in the rule that defines $p(X)$ we can not insert any information about $p(X)$ with modifications applied only to the *EDB*. However, this situation is unlikely to occur in practice. In relational database terminology, even if the relation *q(_,_)* is empty, still its schema definition must be given before we define anything about *p(_)*. Another reason for rejection of an update request is that it might be impossible to generate a transaction that will satisfy both the request and the set of integrity constraints. As explained further, the way we construct the *update formula* and incorporate the integrity constraints will enable detection of impossibility in such cases. We start developing our results assuming no recursive definitions. Since non-recursive Datalog with negation has the same expressive power as relational algebra, our results can be carried over to the relational algebra case directly. Then we show how we obtain a minimal update for the case of recursive definitions through iterative interaction with the user.

The rest of the paper is organized as follows. In Section 2 we recollect some necessary background. Section 3 presents the semantics and the algorithms for the updates. In Section 4 we present how maintenance of the integrity constraints' satisfaction is incorporated. Section 5 demonstrates how interaction with the user can be done and, in particular, used for updates in the presence of recursion. In Section 6 we discuss how our approach can be applied for abductive reasoning and we present some practical considerations. In Section 7 we compare our results with the existing methods. Finally, in Section 8 we summarize and propose directions for future work. The proofs of most of our results are given in the Appendix.

2 Preliminary background

For the databases under consideration, there is a given finite first order language L, defined over some proper alphabet [28, 27]. There are no function symbols. Predicate symbols of the language L are assumed to be partitioned into two disjoint sets: $L_{IP}-$ set of intentional predicates, and $L_{EP}-$ set of extensional predicates. *Terms, atoms, literals* and *formulae* are defined as in [28]. A formula is in *Conjunctive Normal Form* (CNF), if it is a conjunction of disjunctions of literals. It is in *Disjunctive Normal Form* (DNF) if it is a disjunction of conjunctions of literals. A literal in which every term is a constant is called *ground* literal. Formulae of the form $F \leftarrow G$ where $F \equiv f_1 \vee \ldots \vee f_m$, $G \equiv g_1 \wedge \ldots \wedge g_n$, and each f_i and g_j are atoms, are called *clauses* or *rules*, F is called the *head* and G the *body* of a rule. In *Horn* clauses, F is a single positive literal and all the literals in G are positive too. In particular, formulae of the form $F \leftarrow$ are called *facts.* A *definition* of a predicate P is a set of all the clauses that have P in their heads. The *EDB– Extensional Database*, consists of facts (obtained from the predicate symbols of L_{EP}); the *IDB– Intentional Database*, consists of rules which heads are obtained from the predicate symbols from L_{IP}. *Integrity constraints (IC)* are formulae which characterize *correct* states of the database. Often they are expressed in the form of *denials* $\leftarrow A_1, \ldots, A_m$, where each A_i is a positive or a negative literal. The declarative reading is *it is never the case that all the A_i's are true simultaneously in the database.* Equivalent form is $\neg A_1 \vee \ldots \vee \neg A_m$. We will assume that the negation is *safe* both in the *IDB* and *IC* which is: if some variable appears in a negative literal, it must appear in a positive literal too. We also assume that every variable that appears in the head of a rule also appears in the body. A *Deductive Database* is a triple *DB= (EDB,IDB,IC)*[7]. It is considered to be *definite* if the *EDB* consists of ground atoms and the bodies of the rules are conjunctions of literals [29, 44]. We assume standard definitions for *substitution*; *variants*; *composition* of substitutions; *(most general) unifiers*; application of a substitution θ to a literal L ($L(\theta)$) [28].

3 Updating stratified deductive databases

Because of the syntactic structure of deductive databases, there are no negated atoms that can be logical consequence of the database. However, negation is deduced by *default* rule $DB \vdash \neg p(\bar{t})$ *iff* $DB \not\vdash p(\bar{t})$. The effect of this definition in updates is that the insertion of $\neg p(\bar{t})$ into a database DB will produce a new database DB_{new} such that $DB_{new} \not\vdash p(\bar{t})$. Equivalently, using the terminology in [32], given a deletion request for a ground intentional atom $p(\bar{t})$, the *EDB* should be modified in such a manner that no *support* can be constructed for $p(\bar{t})$. Databases in which the rules are of the form $A \leftarrow L_1 \wedge \ldots \wedge L_k$, where each L_i can be a positive or negative literal are called *normal databases* [28]. Having normal databases means that sometimes the insertion of an atom might imply the insertion of *negative* information in the database.

An important class of normal databases are the *stratified* deductive databases which, briefly stated, do not allow recursion through negation in their *IDB*. The important property is that the extension of a fixpoint semantics, *iterated fixed points*, can be applied to generate a unique model [32].

[7]Throughout the rest of the paper we will assume that knowledge base is modeled by a deductive database.

3.1 Semantics for the updates

The rationale behind our update semantics is that we would like to preserve as many facts as possible from the current state of the *EDB* which is, in a sense, equivalent to preserving the accumulated base knowledge After a minimal set of base facts is selected for deletion, we would like to insert as few new facts as possible, in order to minimize the overall changes in the *EDB*.

Based on these minimizations of deletions and insertions, we can define a partial order among all possible interpretations[8] that reflects their *distance* from the current *EDB*. Then, in order to ensure minimality, all we have to do is to select one of the closest databases that satisfies the update request.

Let M be the current extensional database. If I and J denote some *EDB*'s with the same universe of discourse as the database under consideration, we will define a partial order based on M as follows:

Definition 3.1.1. $I \leq J$ *iff*

$(M \setminus I) \subset (M \setminus J)$ OR $[(M \setminus I) = (M \setminus J)$ AND $(I \setminus M) \subseteq (J \setminus M)]$ □

It is easy to verify that this relation is a partial order over the set of all possible *EDB*s. Given M, if we temporarily treat I and J as databases that are "candidates" as a result of the update:

• The first condition reflects the situation in which the set of facts which I needs to delete from the current *EDB* is a subset of the facts which J needs to delete. We *prefer* I because we want to minimize the amount of default information deduced by negation from the *EDB*.

• The other condition reflects the situation where we know that I needs to delete the same set of base facts from M as J. We can state that I is to be preferred over J iff the set of extra facts which J needs to insert is a superset of the ones in I.

For brevity, we will refer to the order above as P (*preserving*) order. Various types of minimal changes in revisions and updates of propositional knowledge–bases are given in [23, 26]. The minimality is also based on ordering among interpretations, following the Gärdenfors rationality postulates.

Another partial order that is used to define updates in stratified deductive databases is the *symmetric difference* (SD):

$I \leq J$ *iff* $(I \setminus M) \cup (M \setminus I) \subseteq (J \setminus M) \cup (M \setminus J)$

It is worthy noting that the P order is stricter than the SD order, in a sense that some *EDB*'s that are incomparable in SD can be ordered with respect to P, as the following example shows.

Example 3.1.1. Let $M = \{a, b, c\}$ and $I = \{a, b, c, d, e\}$, $J = \{b, c, d, f\}$. Then:

[8] In our framework possible interpretations are actually the possible *EDB*'s, since no modifications are allowed in the *IDB*. Because of this *AND* the non monotonic nature of deductive databases, we cannot trace the changes for all deducible facts.

$$(I \setminus M) \cup (M \setminus I) = \{d, e\} \not\subseteq (\not\supseteq)\{d, f, a\} = (J \setminus M) \cup (M \setminus J)$$

Hence $I \not\leq_{SD} J$ and $J \not\leq_{SD} I$. But, since $(M \setminus I) = \{\} \subset \{a\} = (M \setminus J)$ we have $I \leq_P J$. It can also be shown that if $I \leq_{SD} J$ then $I \leq_P J$. Thus, the P order restricts the number of possible *candidates* for the new databases. This result is not surprising since in the P order extra information is used to order the modifications: insertions take precedence over deletions. In the rest of this work we concentrate on developing the P order although similar results can be obtained for the SD order.

Definition 3.1.2. Given a database $DB = EDB \cup IDB$ and an update request of the form *insert* p (*insert* $\neg p$), where p is a ground atoms and $DB \not\vdash p$ ($DB \vdash p$) we will say that $DB_{new} = EDB_{new} \cup IDB$ is a minimal new database that satisfies the update request *iff* there exists no other $DB_o = EDB_o \cup IDB$ that also satisfies the request and $EDB_o \leq_P EDB_{new}$. □

Example 3.1.2. Consider the following intentional part of a deductive database:

IDB= $\{p \leftarrow r, \neg c, \neg d$; $p \leftarrow a, \neg b, \neg d$; $p \leftarrow b, \neg c, \neg a \quad r \leftarrow a$; $r \leftarrow b \quad s \leftarrow b, d\}$

If we want to *insert* p, we can write the disjunction of the bodies of all the rules from the *IDB* that have p in their head (i.e. the rules that define p) $(r \wedge \neg c \wedge \neg d) \vee (a \wedge \neg b \wedge \neg d) \vee (b \wedge \neg a \wedge \neg c)$. If the insertion of $\neg p$ is requested then we can move the negation symbols inside the formula to apply to single atoms: $(\neg r \vee c \vee d) \wedge (\neg a \vee b \vee d) \wedge (\neg b \vee a \vee c)$. Subsequently repeating the process of substitution of the definitions for the intentional predicates in the resulting formula and moving negation inside, we will obtain a formula for the update of p (resp. $\neg p$). In this example, the next step would be to replace r.

In general, let $\mathcal{F}(ur)$ denote a formula for an arbitrary update request ur. Clearly, $DB \vdash ur$ *iff* $DB \vdash \mathcal{F}(ur)$ because $\mathcal{F}(ur)$ is obtained from the definitions of the particular intentional atoms in the *IDB*. Let $\mathcal{F}_{DNF}(ur)$ denote a disjunctive normal form of the $\mathcal{F}(ur)$. Since we have substituted the definitions of all the intentional predicates while constructing the $\mathcal{F}(ur)$, if there exist disjuncts in $\mathcal{F}_{DNF}(ur)$ with occurrences of intentional predicates, we can freely omit them from further consideration, since the presence of intentional predicates in $\mathcal{F}_{DNF}(ur)$ means that there are no definitions for those particular predicates in the *IDB*. Consequently, none of them, in positive or negative form, can be satisfied through modifications of the *EDB* only. Let $\mathcal{F}'_{DNF}(ur)$ denote the formula obtained from the $\mathcal{F}_{DNF}(ur)$ after omitting all the disjuncts that have intentional atoms. Clearly, if $\mathcal{F}'_{DNF}(ur)$ is inconsistent, the update request can not be satisfied.

Let S denote the set of all the literals from $\mathcal{F}'_{DNF}(ur)$. This set has a crucial role in realizing the update request in a minimal manner. The literals that appear in it may or may not be present in the database, but that information can be obtained by regular queries. We will partition S into the following sets:

- The positive literals that are present in the *EDB*, $P_p = \{x_i \mid x_i \in S \wedge x_i \in EDB\}$
- The positive literals that are absent from the *EDB*, $P_a = \{x_i \mid x_i \in S \wedge x_i \notin EDB\}$
- The negative literals that are present in the *EDB*, $N_p = \{\neg y_i \mid \neg y_i \in S \wedge y_i \in EDB\}$, and
- The negative literals that are absent from the *EDB*, $N_a = \{\neg y_i \mid \neg y_i \in S \wedge y_i \notin EDB\}$

The main problem in realizing minimal updates is that, when variables are present, the generation of the set S, which consists of the ground facts relevant for the update, is not as straightforward as in the propositional case. On the other hand, many simple updates require instantiation of variables. Before presenting the algorithm for minimality, we discuss the process of instantiation of variables.

3.2 Formalizing the instantiation process

The main task of this section is to demonstrate that, if the negation is *safe*, the generation of the set S is always finite[9].

Example 3.2.1. Let:

$$\begin{array}{ll} IDB = & \{p(X,Y) \leftarrow q(X,Z), \neg r(Z,U), s(U,Y)\} \\ EDB = & \{q(a,c), q(a,d), r(m,n), s(g,b)\} \end{array}$$

and consider the following update requests:

- *insert p(e,f)*. The straightforward generation of the update formula would give $\mathcal{F}(p(e,f)) \equiv q(e,Z) \wedge \neg r(Z,U) \wedge s(U,f)$. Similarly, a request:
- *delete p(a,b)* would give $\mathcal{F}(\neg p(a,b) \equiv \neg q(a,Z) \vee r(Z,U) \vee \neg s(U,b))$.

Clearly, $EDB' = (EDB \setminus \{r(m,n)\}) \cup \{q(e,m), s(n,f)\}$ satisfies $\mathcal{F}(p(e,f))$ but the same task is achieved by $EDB_n = EDB \cup \{q(e,h), s(l,f)\}$ and $EDB_n \leq_P EDB'$. Similarly $EDB'' = EDB \setminus \{q(a,c), q(a,d), s(g,b)\}$ deletes $p(a,b)$, however $EDB_m = EDB \cup \{r(c,g), r(d,g)\}$ also satisfies the deletion request and $EDB_m \leq_P EDB''$.

This points out that the instantiation of the variables can not be done arbitrarily to guarantee the minimality. As discussed in [48], in accordance with Clark's completion [31, 34], the precise definition of a view can be taken as:

$$(\forall X,Y)(p(X,Y) \Leftrightarrow (\exists Z,U)(q(X,Z) \wedge \neg r(Z,U) \wedge s(U,Y)))$$

The variables that appear in the body but not in the head are existentially quantified within the scope of universally quantified variables of the head of a rule. Consequently, the generation of the

[9] Equivalently, given the current state of the *EDB* the set of constants from the *active domain* which are relevant for the update can be always determined and is finite.

update formula must involve the quantifiers too, and when negation is moved inside the formula, proper changes of quantifiers must be done. The completeness and decidability when using Clark's completion follow directly from the results given in [33], since we have assumed no recursive definitions, so far. Hence, for the request *insert p(e,f)*, the update formula with the quantifiers is:
$\mathcal{F}_q(p(e,f)) \equiv (\exists Z, U)(q(e,Z), \wedge \neg r(Z,U) \wedge s(U,f)))$
Given the current *EDB*, the values of Z and U for which $r(Z,U) \in EDB$ can always be determined. Thus, any other pair of values would make $\neg r(Z,U)$ *true* by default, which in this case coincides with the completion. Then, all we have to do is insert $q(e,Z)$ and $s(U,f)$ for such an instantiation of Z and U. In a similar manner in $\mathcal{F}_q(\neg p(a,b)) \equiv (\forall Z, U)(\neg q(a,Z) \vee r(Z,U) \vee \neg s(U,b))$ the universal quantification $(\forall Z, U)$ could take us into a potentially infinite instantiation without achieving the update at all. However, safeness of the rules enables us to deduce the finite set of values for Z and U for which $\neg q(a,Z) \vee \neg s(U,b)$ does not hold (i.e. the values for Z and U that make $q(a,Z)$ and $s(U,b)$ *true*). For those values, we can always force $r(Z,U)$, and for the rest of the potentially infinite universe, $\neg q(a,Z) \vee \neg s(U,b)$ will hold. We have the following:

Theorem 3.2.1. If the rules in the *IDB* are safe, the instantiation of variables and the generation of the set S of ground facts relevant for the update request, is always finite. □

Safe negation ensures that the queries posed to the current instance of the database are *range restricted* [46]. This, in turn, guarantees the domain independency of the queries, justifying Theorem 3.2.1.

Example 3.2.2. Consider the deductive database given by:

IDB: $p(X,Y) \leftarrow q(X,Z), s(Z,Y)$

$q(X,Y) \leftarrow t(X,c,Z), \neg u(Z,Y), v(W,Y)$

$q(X,Y) \leftarrow t(X,Z,d), o(Z,Y)$

$s(X,Y) \leftarrow m(X,b,Z), \neg o(Z,Y)$

EDB:	*t(a,a,d)*	*m(a,n,d)*	*o(b,n)*	*v(a,k)*
	t(a,b,d)	*m(a,n,c)*	*o(c,n)*	
	t(a,c,d)	*m(k,b,n)*	*o(d,b)*	
	t(a,c,m)	*m(p,b,d)*	*o(l,n)*	

Let the update request be *insert* $\neg p(a,b)$. We can freely assume that local variables are uniquely renamed. Then: $\mathcal{F}_q(\neg p(a,b)) \equiv (\forall Z)\{(\forall Z_2, Z_3)$

$$[(\neg t(a,c,Z_2) \vee u(Z_2,Z) \vee \neg v(Z_3,Z)) \wedge (\forall Z_4)(\neg t(a,Z_4,d) \vee \neg o(Z_4,Z)] \bigvee (\forall Z_5)[\neg m(Z,b,Z_5) \vee o(Z_5,b)]\}$$

Without further transformation of $\mathcal{F}_q(\neg p(a,b))$, we can see that the tuples of interest are:

$$S = \{\neg t(a,a,d), \neg t(a,b,d), \neg t(a,c,d), \neg t(a,c,m), \neg m(k,b,n),$$

$$\neg m(p,b,d), \neg o(b,n), \neg o(c,n), \neg v(a,k), u(d,k), u(m,k), o(n,b), o(d,b)\}$$

Splitting the set into a disjoint subsets, we will get:

$N_p = \{\neg t(a,a,d), \neg t(a,b,d), \neg t(a,c,d), \neg t(a,c,m), \neg m(k,b,n), \neg m(p,b,d), \neg o(b,n), \neg o(c,n), \neg v(a,k)\}$
$N_a = \{\}$ $P_p = \{o(d,b)\}$ $P_a = \{u(d,k), u(m,k), o(n,b)\}$

It is easy to check that { *insert o(n,b)* } is sufficient to realize the update request. All the possibilities to make *s(Z,b)* for the current values in the database *(the active domain)* will fail, therefore *p(a,b)* will no longer be derivable. There are few other possibilities to achieve the request, and the minimality is the subject we discuss next. Let us mention that none of the methods that we have found in the literature which attempt to produce a single output to accomplish an update request ([2, 6, 37, 38]) are capable of producing the output that we have presented above. [6, 37, 38] do not consider negation at all and [2] is not capable of handling negation and local variables as given in this example.

3.3 Minimizing the update

As we mentioned, in case $\mathcal{F}'_{DNF}(ur)$ is inconsistent, the update request cannot be satisfied because of the way the particular view is defined. Let $\mathcal{F}'_{DNF}(ur) \equiv T_1 \vee \ldots \vee T_m$, where $T_i = t^i_1 \wedge \ldots \wedge t^i_{n_i}$. If there exists some T_j for which there are some t^j_k and t^j_m such that $t^j_k = x$ and $t^j_m = \neg x$, that particular T_j can never be satisfied in any state of the *EDB*, so it does not need to be considered further. Let $\mathcal{F}''_{DNF}(ur)$ be a formula obtained from the $\mathcal{F}'_{DNF}(ur)$, after removing all the disjuncts like T_j above. Obvioulsy, if $\mathcal{F}''_{DNF}(ur)$ is empty then $\mathcal{F}'_{DNF}(ur)$ is inconsistent. For a given T_i, let $\mathcal{S}(T_i) = \{t^i_1, \ldots, t^i_{n_i}\}$. Now we can present the algorithm for minimal view updates *LGS*:

1. Construct $\mathcal{F}''_{DNF}(ur)$ for the given request.
2. If $\mathcal{F}''_{DNF}(ur)$ is *empty*

   update impossible– exit.
3. Else

   4. Instantiate the variables and obtain the set S.

   5. Split S into N_p, N_a, P_p, P_a.

   6. Apply minimization procedure *LG–min*

The minimization procedure *LG–min* is as follows:

While ($\exists T_i$ such that $\mathcal{S}(T_i) \subseteq S$ AND ($\exists \neg y_k \in N_p$) that is not marked

  Select a non marked $\neg y_m \in N_p$

  Mark $\neg y_m$

  If ($\exists T_j$ such that $\mathcal{S}(T_j) \subseteq (S \setminus \{\neg y_m\})$)

    Let $N_p = N_p \setminus \{\neg y_m\}$

    Let $S = S \setminus \{\neg y_m\}$

    Let $\mathcal{F}''_{DNF}(ur) = \bigvee_i T_i$ such that ($\not\exists t^i_l = \neg y_m$)

While ($\exists T_i$ such that $\mathcal{S}(T_i) \subseteq S$ AND ($\exists x_k \in P_a$) that is not marked)

  Select a non marked $x_h \in P_a$

  Mark x_h

  If ($\exists T_j$ such that $\mathcal{S}(T_j) \subseteq (S \setminus \{x_h\})$)

    Let $P_a = P_a \setminus \{x_h\}$

    Let $S = S \setminus \{x_h\}$

    Let $\mathcal{F}''_{DNF}(ur) = \bigvee_i T_i$ such that ($\not\exists t^i_l = x_h$)

Let $EDB_{new} = (EDB \setminus \{y_j \mid \neg y_j \in N_p\}) \cup P_a$

Termination- guaranteed since we have two bounded *while* loops.

Complexity- Let $n = \mid S \mid (= O(\mid EDB \mid))$. S could have been obtained from $\mathcal{F}(ur)$ itself and it is the step of transforming the $\mathcal{F}(ur)$ into $\mathcal{F}_{DNF}(ur)$ that makes the overall complexity exponential in the size of the *IDB*. After the $\mathcal{F}''_{DNF}(ur)$ has been obtained, in every iteration through the *while* loop we check if $S \setminus \{t\}$ still satisfies$\mathcal{F}''_{DNF}(ur)$, where $t \in N_p$ for the first *while* loop and $t \in P_a$ for the second one. This test is equivalent to testing if there exists some $T_i = t^i_1 \wedge \ldots \wedge t^i_{n_i}$ such that $(\forall t^i_j) t^i_j \neq t$. If $N = max_i\{n_i\}$, and if m is the total number of disjuncts of $\mathcal{F}'_{DNF}(ur)$, the worst case complexity is $O(mnN)$, where m is the factor that is exponential in the size of the *IDB*. It will be difficult to do this better since we can show that the problem of minimal updates is *NP*– hard with respect to the size of the *IDB*. In Section 7 we discuss the complexity issue in comparison with other approaches (although, except for [6], none of the works gives a formal analysis of the complexity). Observe that in order to generate a candidate which is minimal in the *SD*– order, the two *while* loops can be combined in a single one, which means that the worst case complexity is still the same. The issue of user's interference is reflected in the *select* statements of the procedure *LG–min*. In Section

5 we present an example in which we discuss the interaction with the user when updating recursively defined predicates.

Correctness- The following holds:

Theorem 3.3.1. Let $DB_{new} = EDB_{new} \cup IDB$ and $DB_{new} \vdash ur$. EDB_{new} is P-minimal *iff* Algorithm *LGS* produces it as output. □

Example 3.3.1. For the deductive database in Example 3.1.2,

$\{p \leftarrow r, \neg c, \neg d \; ; \; p \leftarrow a, \neg b, \neg d; \; p \leftarrow b, \neg c, \neg a \qquad r \leftarrow a \; ; \; r \leftarrow b \qquad s \leftarrow b, d\}$

assume that the *EDB*= $\{b, c, d, m, n\}$ and consider the update request: *insert p*.

Then we have: $\mathcal{F}(p) = ((a \vee b) \wedge \neg c \wedge \neg d) \vee (a \wedge \neg b \wedge \neg d) \vee (b \wedge \neg a \wedge \neg c))$;

$$\mathcal{F}''_{DNF}(ur) \equiv (a \wedge \neg c \wedge \neg d) \vee (b \wedge \neg c \wedge \neg d) \vee (a \wedge \neg b \wedge \neg d) \vee (b \wedge \neg a \wedge \neg c))$$

$S = \{a, b, \neg b, \neg c, \neg d, \neg a\} \qquad P_p = \{b\}; \; P_a = \{a\} \; ; \; N_a = \{\neg a\}$ and $N_p = \{\neg b, \neg c, \neg d\}$

There are two possible outcomes, depending on the selections in the *while* loops:

1.) If $\neg c$ is the first element selected in the first *while* loop, then it turns out that $\mathcal{F}_{DNF''}(ur) \equiv a \wedge \neg b \wedge \neg d$, and there is nothing that can be done in the second *while* loop, so the algorithm will produce $EDB_{new} = \{a, c, m, n\}$.

2.) Any other selection in the first *while* loop will reduce the formula to $\mathcal{F}''_{DNF}(ur) \equiv b \wedge \neg c \wedge \neg a$ which in turn means that in the second *while* loop a can be removed, producing the final output $EDB_{new} = \{b, d, m, n\}$.

Note that in case the structure of the $\mathcal{F}(ur)$ is such that it has no intentional predicates and also $(P_p \cap \neg N_p) = \{\} = (P_a \cup \neg N_a)$, there is no need to transform $\mathcal{F}(ur)$ into $\mathcal{F}_{DNF}(ur)$. The reason is that we are guaranteed that no T_i will contain intentional predicates and no T_i will be of the contradictory form $T_i = \ldots \wedge a \wedge \ldots \wedge \neg a \wedge \ldots$, requiring insertion and deletion of the same (set of) tuple(s). This, in turn, implies that for those particular cases, the exponential complexity of the Algorithm *LGS* can be avoided. Another interesting detail, concerning the interaction with the user is that instead of asking her/him to select ground facts, a higher level choice could have been offered. After constructing $\mathcal{F}(p)$, the user could have been asked to select a particular disjunct. The check for minimality will, of course, require the rest of the $\mathcal{F}(p)$, but this kind of interaction is particularly important when the user should instantiate some variables. Also, as we will demonstrate in Section 5, this is useful when an update is required for a recursively defined predicates.

4 Integrity constraints

Two key issues that are considered for efficient maintenance of integrity constraints satisfaction are :
1. The assumption that the database is in a correct state before the transaction is applied.
2. Subject to a particular transaction, only the constraints relevant to the transaction need to be checked.
Our approach combines several ideas presented in the literature: 1.) We use a transformation of the database (Kowalski and Sadri [39]) to integrate and check the constraints; 2.) We restrict the checking to the instances of relevant integrity constraints (Bry, Decker and Manthey [17]); 3.) In case some constraint might be invalidated, the repair is embedded within the update process itself in order to insure the overall minimality (Moerkotte and Lockemann [20]).

Definition 4.1. (cf. [17]) A constraint IC_i is *relevant* to an update of the atom A *iff* the complement of the atom A is unifiable with a literal in IC_i. □

For each relevant integrity constraint it is sufficient to check its *simplified instance* [17]. Briefly, given an integrity constraint $IC_a \equiv\leftarrow A_1(\overline{X_1}), \ldots A_i(\overline{X_i}), \ldots, A_{n_a}(\overline{X_{n_a}})$, and an update *insert* $A_i(\overline{c_i})$, the simplified instance of IC_a is obtained by ommiting $A_i(\overline{X_i})$ and propagating the binding substitutions for $\overline{c_i}$ among $\overline{X_1}, \ldots, \overline{X_{n_a}}$.

Before any particular update of $A_i(\overline{X_i})$ is requested, we can realize that the relevant sub–portion of IC_a for $A_i(\overline{X_i})$ will be $IC_a^{A_i} \equiv\leftarrow A_1(\overline{X_1}), \ldots, A_{i-1}(\overline{X_{i-1}}), A_{i+1}(\overline{X_{i+1}}), \ldots, A_{n_a}(\overline{X_{n_a}})$, and upon a request, all that needs to be done is the proper instantiation of variables in order to obtain the simplified instance. Observe that, given the set of integrity constraints, the sub–portion of any relevant integrity constraint for a particular axiom, such as $IC_a^{A_i}$ for A_i above, can be pre–computed[10]. We can always construct an equivalent database in which the sets of sub–portions of relevant integrity constraints are attached to extensional axioms and their negated forms. The work done in semantic query optimization [25, 51] uses *residues* to indicate the sub–portions of interest from the relevant integrity constraints.

Example 4.1. Consider an *IDB* with a single view: $p(X,Y) \leftarrow q(X,Z), r(X,Y,Z)$ and assume that the set of integrity constraints consists of:

$$IC_1 : \leftarrow q(X,Y), r(X,Y,d) \qquad \text{and} \qquad IC_2 : \leftarrow s(X,Y), \neg r(X,Y,X)$$

where *q,r,s* are base relations. After the compilation, the following equivalent form is generated:

[10] It might be the case that a single integrity constraint contributes to more than one simplified instance for updates of a certain literal. Hence, the literal will have several sub–portions relevant to it.

$p(X,Y) \leftarrow q(X,Z), r(X,Y,Z)$

$q(X,Y)$	$\{\neg r(X,Y,Z) \vee \neg(Z=d)\}$
$r(X,Y,Z)$	$\{\neg(Z=d) \vee \neg q(X,Y)\}$
$s(X,Y)$	$\{r(X,Y,X)\}$
$\neg r(X,Y,Z)$	$\{\neg(Z=X) \vee \neg s(X,Y)\}$

The equivalent clausal form $\{\leftarrow r(X,Y,Z),(Z=d)\}$ could have been used too. Given this kind of semantic information, let the update request be: *delete p(a,b)*. The update formula becomes:

$\mathcal{F}^{IC}_{q-DNF} \equiv (\forall Z)\{(\neg q(a,Z) \quad \bigvee \quad (\neg r(a,b,Z) \wedge \neg s(a,b)) \quad \bigvee \quad (\neg r(a,b,Z) \wedge \neg(X=Z))\}$

We can see that $(\neg q(a,Z))$, in the context of the update *delete p(a,b)*, cannot affect any integrity constraint. Both possibilities in which $(\neg r(a,b,Z))$ can affect IC_2 are taken into account. Observe that a straightforward approach of generating an update formula from the view definitions separately and extending it with a conjunction of the set of integrity constrains would have created much more overhead.

Before we give the complete version of the view update procedure, we need to specify the requirements for the set of integrity constraints which ensure termination and completeness:

Rq1 Integrity constraints are expressed in terms of extensional literals[11].

Rq2 The set of integrity constraints is resolution complete (i.e. no new clauses can be derived by resolving any of the existing constraints in the set).

Example 4.2. Let $IC_1 \equiv\leftarrow a(X), \neg b(X)$ and $IC_2 \equiv\leftarrow b(X), \neg c(X)$. This particular set of integrity constraints is not resolution complete, because a third condition that the database must satisfy, $IC_3 \equiv\leftarrow a(X), \neg c(X)$, can be derived by resolving IC_1 and IC_2 on b. In this context, in Example 4.1. above, we could have derived an additional integrity constraint $\leftarrow s(X,Y), q(X,Y)$.

The generation of an equivalent set of integrity constraints expressed with only extensional literals (Rq.1) and completing the set of integrity constraints (Rq.2) can be done as a pre–processing. Then the compilation can be applied to generate the sub–portions of the relevant integrity constraints of defining axioms for extensional atoms (and their negations) for the database in which (Rq.1) and (Rq.2) are satisfied[12].

Now, the algorithm *IC-LGS* can be specified as follows:

[11] This requirement is essentially equivalent to the requirement of no recursion. In case some IC_i has an intentional predicate, it can always be unfolded into one or more integrity constraints expressed purely by extensional atoms.

[12] The works in [39] and [13, 5] also assume some pre-processing of the database

1. Construct $\mathcal{F}_{q-DNF}$ for the update request.
2. Substitute every literal with the conjunction of that particular literal
   and the simplified instances of all the relevant
   integrity constraints from its defining axiom.
3. Transform the formula into $DNF - \mathcal{F}^{IC}_{q-DNF}$.
4. Get $\mathcal{F}^{IC}_{Cq-DNF}$– the consistent subformula of $\mathcal{F}^{IC}_{q-DNF}$
   by ommiting all the inconsistent disjuncts.
5. If $\mathcal{F}^{IC}_{Cq-DNF}$ is *empty*
   6. *impossible to satisfy request and the set of integrity constraints*– exit
7. Else
   8. Generate the S set
   9. Split it into N_p, N_a, P_p, P_a
   10. Apply the *LG–min* procedure.

Lemma 4.1. If the database satisfies the requirements Rq1 and Rq2, the output of the algorithm *IC-LGS* does not violate any integrity constraint. □

Theorem 4.1. Given an update request, algorithm *IC-LGS* will produce an EDB_{new} as an output *iff* it is minimal and satisfies the set of integrity constraints. □

Observe that Theorem 4.1. is a direct consequence of Theorem 3.3.1. and Lemma 4.1.

5 Recursive definitions

In general it is difficult to have determinism when updating a recursively defined predicate, because the *boundedness* of recursion (i.e. its equivalence to a finite union of non recursive conjunctive rules) is in many cases *undecidable*. Decidability has been proven for restricted subclasses [24]. In our method, updating a recursively defined predicate is handled simply through iterative interactions with the user. Generating the $\mathcal{F}''_{DNF}$ as described in Section 3 may not terminate. Thus, the moment a predicate is identified as recursive, a partial $\mathcal{F}''_{DNF}$ is presented to the user and she/he is asked to select one of the disjuncts which satisfaction will be enforced. In case the disjunct consists entirely of base relations, upon instantiation of variables, the minimality is checked with respect to the other disjuncts and the current instance of the database. If the user selects a disjunct with recursively defined views, the whole process is repeated. Finally, minimality is checked again given the current instance of the database and the set of ground facts obtained during the interaction with respect to the other disjuncts of the update formula[13].

IDB:

r(X,Y) <- p(X,Z), q(Z,Y).

r(X,Y) <- p(X,W), s(W,Y,Z), q(Z,Y).

t(X,Y) <- r(X,Z), q(Z,Y).

t(X,Y) <- s(X,Z,Y), u(Y,Z).

u(X,Y) <- p(X,Z), p(Z,Y).

u(X,Y) <- q(X,Z), t(Z,Y).

EDB:

p(a,b).	q(c,b).	s(a,c,b).
p(a,c).	q(i,c).	s(b,c,n).
p(a,n).	q(j,i).	s(c,n,b).
p(b,i).	q(n,c).	s(d,w,n).
p(c,d).	q(w,n).	
p(d,b).		
p(d,c).		
p(d,d).		
p(i,c).		
p(m,b).		
p(n,j).		

Figure 1: The database before the update

Example 5.1. Consider an instance of a database given in Figure 1, where *u(X, Y)* is recursively defined. Assume that the update request is *insert u(m,n)*. First partial $\mathcal{F}''_{DNF}$ will contain the following disjuncts (assume that the local variables are uniquely renamed):

`[p(m,Z),p(Z,n)]`

`[q(m,U),p(U,V),s(V,W,K),q(K,W),q(W,n)]`

`[q(m,U),p(U,L),q(L,W),q(W,n)]`

`[q(m,U),s(U,M,n),u(n,M)]`

Assume that the 4th disjunct is selected. Since u is recursively defined, it is detected and reported to the user, after which he/she is asked to instantiate the variables Assume that the set of facts provided by user is `q(m,d);s(d,a,n)`, in that order. This is enough to instantiate `u(n,a)`, but it is not a consequence of the current instance of the database augmented by `q(m,d);s(d,a,n)`. Hence, the whole new cycle is started for the insertion of `u(n,a)`:

`[p(n,Z),p(Z,a)]`

`[q(n,U),p(U,V),s(V,W,K),q(K,W),q(W,a)]`

`[q(n,U),p(U,L),q(L,W),q(W,a)]`

`[q(n,U),s(U,M,a),u(a,M)]`

Upon the instantiation `q(n,w); s(w,c,a)`, `u(a,c)` still cannot be derived and the whole process is repeated in order to insert it:

`[p(a,Z),p(Z,c)]`

[13]Another possibility is to have the user specify the number of iterations and, if there are not enough facts provided before the bound is exceeded, force the user to choose a disjunct with only base relations in the last iteration.

```
[q(a,U),p(U,V),s(V,W,K),q(K,W),q(W,c)]
[q(a,U),p(U,L),q(L,W),q(W,c)]
[q(a,U),s(U,M,c),u(c,M)]
```

Suppose the user selected the 4th candidate again. When asked for instantiation, assume that she/he gives the facts: `q(a,k); s(k,b,c)` which propagates automatically to `u(c,b)`. Given the current instance of the database plus the set of facts obtained in the interaction so far, `u(c,b)` is a consequence, so there is no need of further interaction (sufficient set of facts is provided). The base facts collected so far are: `[q(m,d),s(d,a,n),q(n,w),s(w,c,a),q(a,k),s(k,b,c)]` and none of them is present in the database. But, observe that both `p(d,d)` and `s(d,w,n)` hold in the current instance of the database. The minimization procedure will detect that from all the facts obtained during the interaction, inserting the tuples `[q(m,d),q(n,w)]` is enough to satisfy the second disjunct, consequently the whole update request.

The example above is actually the way our PROLOG - *minimizer* implementation [21] will behave for the given update request. We have also implemented a user interface which, given a schema description, automatically generates the selection menus and communication boxes (say, for variable instantiation) [36]. The generality of a system like this is that it is independent of a particular application domain. Treating updates as queries with side effects [1], makes our system very attractive for applications which require some form of *hypothetical reasoning*. Namely, creating a transaction does not have to imply *commit*. Users should have an opportunity for reasoning like "what would be resulting modifications if $p(\overline{V})$ is to become true/false", based on the selections in the update process. This kind of problems are discussed in the next section.

6 Abduction and Cooperative Updates

Abduction is a form of non monotonic logical inference which is used to find a plausible explanation(s) for a data/observation with respect to a given knowledge. Identified by Pierce [7] as a form of syntactic reasoning of the case *from a rule and a result*, it can be formalized as an inverse modus–ponens:

$$\frac{p \Rightarrow q, q}{p}$$

Abduction has been studied by many researchers in AI and recently as a belief revision mechanism [35, 26]. In logic based approaches [3, 18, 40], abduction is defined in a following manner:

• Given a set of sentences T (a theory presentation), and a sentence G (observation), the abductive task can be characterized as finding a set of sentences Δ (abductive explanations for G) such that:

1.) $T \cup \Delta \models G$.

2.) $T \cup \Delta$ is consistent.

However, since in knowledge bases, integrity constraints (IC) are used to describe consistent states, the consistency requirement above can be formulated as:

2'.) $T \cup \Delta$ satisfies IC.

Kakas and Mancarella [2] use an abductive framework to describe view updates in a very natural manner. Teniente [12] also demonstrates how abduction can be used to handle update requests in a consistency–preserving mode (with respect to a given set of integrity constraints). The problem of updating intentional predicates can be tackled as selecting and explanation/hypothesis if the update request is understood (as we mentioned in Section 5) as: " what would be the resulting modifications if $p(\overline{V})$ is to become true/false". In our scenario, the set of *abducibles* corresponds to the set of *EDB* predicates. We have already demonstrated the flexibility of our method for the purpose of selecting a hypothesis which reflects users' needs/preferences (cf. [9, 8]). Below we present an example which shows how our method can be used to perform abductive reasoning in practical situations.

Example 6.1. Consider an international export–import company with the *IDB* part of its deductive database given as:

```
import(Product) ← available(Product,Country), ¬ ban_import(Product),
                    ¬ ban_trade(Country).
available(Product,Country) ← manufactures(Product,Country,Price), profitable(Price).

ban_product(Product) ← contains(Product,Ingredient,Percentage),
     limits_fda(Ingredient,P_low,P_high), ¬ in_range(Percentage,P_low,P_high).

ban_trade(Country) ← sanctions(Country).
ban_trade(Country) ← in_region(Country,Region),troubled(Region).
```

Assume, for simplicity, that the predicates not mentioned in the head of the rules are extensional[14]. Now, assume that at some point user requests *insert import(prod-1)*. Based on the definition of *import*, the following formula is generated:

[14] In practice, however, *profitable* is very likely to be defined via some other rules that reflect the current situation on the market and/or experts predictions.

$(\exists Country)\{(\exists Price)[manufactures(prod_1, Country, Price) \wedge profitable(Price, prod_1)] \bigwedge$
$(\forall In, Pc, P_l, P_h)[\neg\ contains\ (prod_1, In, Pc) \vee \neg\ limits\_fda\ (In, P_l, P_h) \vee\ in\_range\ (Pc, P_l, P_h)] \bigwedge$
$[\neg\ sanctions(Country)\ \wedge (\forall Region)[\neg\ in\_region(Country, Region) \vee \neg\ troubled(Region)\ ]]\}$

The existing approaches would require deletion of some tuple from *sanctions, limits-fda, troubled.* As much as it is syntactically correct, in practice that would be an impossible requirement. Namely, although any company might access those databases in a *Read-Only* manner, they are likely to be controlled by the government (some of them even by the *UN*) for modification purposes. On the contrary, our approach, as demonstrated in Section 3, is capable of extracting the values of existentially quantified variable (like *Country*) that appear in negated atoms[15]. Moreover, generating all the possibilities affects the efficiency. Another important issue is the ability for the user to specify some preferences about his favorite hypothesis beforehand. Namely, since our approach essentially constructs a formula for the update request, it can deal with the *composite* requests of the form $p(\overline{a}) \wedge (\neg) q(\overline{b}) \ldots$. In the context of the Example 6.1. above this means that the user has an additional flexibility of saying *I would like* `prod-1` *from a country different than* `country-1`.

7 Comparison with other methods

Much research has been done for the problem of view updates. Guessoum and Lloyd [4] offer an approach for view updates in knowledge bases in which they deal with negative information concluded through some form of closed world reasoning. Their method allows modifications in the intentional database.

The works presented in [2, 6, 37, 38] concentrate on generating one out of many possible transactions that would satisfy given update request. Unlike our method, [6, 37, 38] treat procedural and declarative semantics of view updates for the restricted class of Horn deductive databases.

Tomasic [6] suggests the assignment of annotation to deductive rules in order to limit the amount of derivation but, a unique annotation cannot guarantee the minimality of every update. If negation is present (not covered in [6]) it can be shown that even a unique annotation will still result in exponential complexity in the size of the *IDB.* Atzeni and Torlone [37] propose querying the user any time an ambiguity arises, which is essentially a more flexible annotation (on-the-fly). In [38] they extend the method to deal with integrity constraints restricted to functional dependencies. The

[15] Clearly, in this particular case, the only possibility is that the desired product is bought from a country which is not on the list of banned ones for trading and is not in a troubled region (provided *fda* limits are satisfied, and the price is profitable).

algorithms presented ensure the generation of *potential* updates which are minimal only if there is a unique way to satisfy the request. The work does not treat negation or consider the issue of local variables.

Kakas and Mancarella [2] consider negation (locally stratified [50]) and use abduction to generate an output that realizes a given request. Essentially, they distinguishing two steps in realizing the update request. In the first step, the request is translated into several sets Δ_i, each containing sufficient requirements that the *EDB* should satisfy. The second step involves the application of the update, as indicated by one Δ_i on the *EDB*. As pointed out in [13], unnecessary work might be done because the current state of the *EDB* is not taken into account, and a non-minimal output may be produced. The authors do not consider formally the efficiency and the minimality of updates in the presence of the set of integrity constraints. Also, the work lacks a formal treatment of local variables. There are very few applications where local variables are not needed.

The update problem in general first order theories is discussed in [4, 41]. The approaches face problems such as complicated syntax of the database (disjunction of tuples), exponential growth of the size of the database, which makes the problem intractable and complicates query processing. Extension of deductive databases to *Disjunctive Deductive Databases* [27] retains those problems, and the work handles only ground (propositional) cases. [42, 43, 47] treat declarative specifications of updates and the evolution of the database without providing any procedures to be used in practice.

The methods below try to generate all the minimal transactions that satisfy a view update request, which number may be $O(2^{(O(IDB))})$[6].

Teniente and Olivé [13] extend the *event* calculus introduced by Olivé in [5]. The method augments the database with *transition* and *events* rules, exponentially increasing the number of rules in the *IDB*. This enables *SLDNF* to be used in order to generate all the possible transactions that satisfy an update request and the set of integrity constraints. The use of *SLDNF* makes the method crucially dependent on the constants that appear in the current active domain, not addressing the issues of local variables and interaction with the user The authors consider the conjunction of the *IC*–set and the update request which might require too much overhead. Our work is close in spirit to their approach in [14] where they consider the problem of a combination of view updating and consistency maintenance, and consider the issue of local variables' instantiation. Mayol and Teniente [11] describe the results concerning the efficiency of updating consistent deductive databases. The concept of augmented database is used by Teniente [12] to define an abductive framework for handling consistency_preserving updates.

Bry [16] also uses an abductive approach for updating logic programs. Instead of specifying the updates extensionally, as changes to perform, they are specified intentionally – as expected effects. This, in a sense, extends the database notion of a view updates. The work describes a theorem prover (Satchmo) and a set of meta–clauses, which exhaustively generate all finite minimal realizations of an intentional update, not addressing the issue of efficiency. Interaction with the user is not considered.

Abduction in Horn–clause knowledge bases is considered by Hinkelmann [30]. The author presents how rewriting of knowledge base (extension of query optimization techniques) can be used both for finding abductive explanation(s) and checking the consistency, without considering a possible *repair* (cf. [20]) that corresponds to the update.

Decker's method [22] handles deletion requests by having each non–failed derivation in the *SLDNF* tree fail. The updates for insertions are obtained by enforcing the success of some failed derivation. The completeness is relative, in a sense that there may exists a valid update that cannot be generated. Given $\{p \leftarrow r \wedge \neg q;\ q \leftarrow r \wedge \neg s\}$, the request *insert* p cannot be handled.

8 Conclusions and future work

In this work we have presented an approach for updating intentionally specified predicates in knowledge bases. We have demonstrated that the minimality of the update can be ensured and the satisfaction of integrity constraints can be maintained efficiently, generating only one possible solution. The semantics for minimality of updates that we have proposed and the correctness of our algorithms qualify our approach to the update problem as *semantically acceptable* [15][16]. Our approach is also applicable for *composite* update requests of the form *insert* $(\wedge_{i,j}(\vee_{i,j})p_i, \neg q_j)$ for which overall minimality (and maintaining satisfaction of the set of integrity constraints) is guaranteed. As we mentioned, it is straightforward to construct an algorithm which can offer a selection of the partial order for minimality (i.e. *SD* or *P*).

There are few immediate extensions of our work. From practical aspects, it will considerably increase generality and flexibility if our meta updater [21] is extended with cooperative answering module as described in [19]. With respect to Example 6.1. (export–import company) if the user specifies a *price* which is profitable, it may be the case that the only country which manufactures the product for an acceptable price is on the list of the countries which are banned for trading.

[16] This means that if the request is an insertion of a ground intentional atom which is already *true*, or a deletion of the one which is *false*, the update will not change the database, and in case modifications are needed, they are non–redundant.

Rejecting an update request as impossible to achieve is undesirable. An intelligent system should explain the reason to the user, and try to offer another options that would satisfy the update request. In this particular example the system should offer an option of relaxing the price before rejecting the request. Merging an update request with integrity constraints and user constraints/preferences in a cooperative manner can build a new information, but we need more formal study of what cooperativity means in terms of updates (as queries with side effects). In this spirit, it would be interesting to investigate to what extent the information provided by the user (preferences) can be used to *optimize* the extraction of relevant information from the current state of the *EDB*. Another challenging issue, closely related to *active* databases is the incorporation of *dynamic* integrity constraints satisfaction during the generation of the updates. It is worthy noting that sometimesm, the *minimality* itself might not be the crucial issue. In many applications there might be another preference criteria among possible solutions to the update/belief revision problem. Investigating rational criteria not based on the notion of *minimality* is another interesting research problem.

Acknowledgments: The authors would like to thank Ernest Teniente for his valuable comments and suggestions.

References

[1] A.Boner and M.Kiffer. Transaction logic programming. Technical Report CSRI–270, University of Toronto, 1995. 1995 revision.

[2] A.C.Kakas and P.Mancarella. Database updates through abduction. In *Proceedings of the 16th VLDB Conference*, 1990.

[3] R.A.Kowalski A.C.Kakas and F.Toni. Abductive logic programming. *Journal of Logic and Computation*, 2(6):719–770, 1992.

[4] A.Guessoum and J.W.Lloyd. Updating knowledge bases. Technical Report TR-89-05, Dept. of Computer Science - University of Bristol, 1989.

[5] A.Olivé. Integrity constraint checking in deductive databases. In *Proceedings of the 17th Conference on VLDB*, 1991.

[6] A.Tomasic. View update translation via deduction and annotation. In *Second International Conference on Database Theory– ICDT'88*, pages 338–352, 1988.

[7] C.S.Pierce. *Collected Papers of Charles Sanders Pierce*, volume 2. Harvard University Press. 1931–1958.

[8] D.Poole. On the comparison of theories: Preferring the most specific explanation. In *Proceedings of the 9th IJCAI*, pages 143–148, 1986.

[9] D.Poole. Variable in hypotheses. In *Proceedings of the 10th IJCAI*, pages 905–909, 1987.

[10] E.F.Codd. A relational model of data for large shared banks. *Communications ACM*, 13(6):377–387, 1970.

[11] E.Mayol and E.Teniente. Towards an efficient method for updating consistent deductive databases. In *Proceedings of the BIWIT'95*. IEEE Computer Society Press, 1995.

[12] E.Teniente. An abductive framework to handle consistency preserving updates in deductive databases. In *Proceedings of the ICLP'95 Joint Workshop*, pages 111–125, 1995.

[13] E.Teniente and A.Olivé. The events method fo view update in deductive databases. In C.Delobel A.Pirotte and G.Gottlob, editors, *Advances in Database Technology – EDBT*, pages 245–260. Springer–Verlag, 1992.

[14] E.Teniente and A.Olivé. Updating knowledge bases while maintaining their consistency. *VLDB Journal*, 4(2), 1995.

[15] F.Banchilon and N.Spyratos. Update semantics of relational views. In *ACM Transactions on Database Systems*, pages 557–575, 1981.

[16] F.Bry. Intentional updates: Abduction via deduction. In D.Warren and P.Sceredi, editors, *Proceedings of the 7th International Conference on Logic Programming*. Morgan - Kaufmann, 1990.

[17] H.Decker F.Bry and R.Manthey. A uniform approach to constraint satisfaction and constraint satisfiability in deductive databases. In *Advances in Database Technology -EDBT'88*, pages 488–505, 1988.

[18] F.Toni. *Abductive Logic Programming.* PhD thesis, Imperial College, London, 1995.

[19] T. Gaasterland. *Generating Cooperative Answers in Deductive Databases.* PhD thesis, University of Maryland, 1992.

[20] G.Moerkotte and P.C.Lockemann. Reactive consistency control in deductive databases. *ACM TODS*, pages 670–702, 1991.

[21] G.Trajcevski. Minimality of view updates in deductive databases. Master's thesis, University of Illinois at Chicago, 1994.

[22] H.Decker. Drawing updates from derivations. In *Third International Conference on Database Theory–ICDT'90*, pages 437–454, 1990.

[23] H.Katsuno and A.O.Mendelzon. Propositional knowledge base revision and minimal change. Technical Report KRR-TR-90-3, University of Toronto, 1990.

[24] J.F.Naughton and Y.Sagiv. A decidable class of bounded recursion. In *Proceedings of the 6th International Symposium ACM - PODS*, pages 227–235, 1987.

[25] J.Grant and J.Minker. Integrity constraints in knowledge based systems. Technical Report 89 - 39, University of Maryland, 1989.

[26] J.Lobo and C.Uzcategui. Abductive change operators. submitted to Fundamenta Informaticae, 1996.

[27] J.Lobo, J.Minker, and A.Rajasekar. *Foundations of Disjunctive Logic Programming.* MIT Press, 1992.

[28] J.W.Lloyd. *Foundations of Logic Programming.* Symbolic Computation Series. Springer - Verlag, 1984.

[29] J.W.Lloyd and R.W.Topor. A basis for deductive database systems. *Journal of Logic Programming*, 3:55–67, 1986.

[30] K.Hinkelmann. Knowledge-base rewriting for bottom-up abduction and integrity checking. In *Proceedings of the ICLP'95 Joint Workshop*, pages 127–141, 1995.

[31] K.L.Clark. Negation as a failure. In H.Gallaire and J.Minker, editors, *Logic and Data Bases.* Plenum Press, 1978.

[32] K.R.Apt, H.Blair, and A.Walker. Towards a theory of declarative knowledge. In J.Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann, 1988.

[33] K.R.Apt and M.Bezem. Acyclic programs (extended abstract). In D.Warren and P.Szeredi, editors, *Proceedings of the 7th International Conference on Logic Programming*, pages 617–634, 1990.

[34] M.Genesereth and N.I.Nilsson. *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann, 1988.

[35] A.C.Nayak M.Pagnucco and N.Y.Foo. Abductive reasoning, belief expansion and nonmonotonic consequence. In *Proceedings of the ICLP'95 Joint Workshop*, pages 143–158, 1995.

[36] N.Grover. Automatic generation of user interfaces for view updates. University of Illinois at Chicago, Master's Project, 1994.

[37] P.Atzeni and R.Torlone. Solving ambiguities in updating deductive databases. In J.Demetrovics B.Thalheim and H. D.Gerhardt, editors, *Symposium on Mathematical Foundations of Database and Knowledge Base Systems*, pages 104–118. Springer–Verlag, 1991.

[38] P.Atzeni and R.Torlone. Updating deductive databases with functional dependencies. In *2nd International Conference on Deductive and Object–Oriented Databases – DOOD'91*, pages 278–291, 1991.

[39] R.A.Kowalski and F.Sadri. Knowledge representation without integrity constraint. Technical report, Dept. of Computing - Imperial College of Science and Technology, London, 1988.

[40] R.A.Kowalski and K.Eshgi. Abduction compared with negation by failure. In *Proceedings of the 6th ICLP*, 1989.

[41] R.Fagin, G.M.Kuper, J.D.Ullman, and M.Y.Vardi. Updating logical databases. *Advances in Computing Research*, 3:1–18, 1986.

[42] R.Reiter. On formalizing database updates: Preliminary report. In C.Delobel A.Pirotte and G.Gottlob, editors, *Advances in Database Technology–EDBT*, pages 10–20. Springer–Verlag, 1992.

[43] R.Reiter. How a database should progress and why. In *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning*, 1994.

[44] R.W.Topor and E.A.Sonenberg. On domain independent databases. In J.Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 217–240. Morgan Kaufmann, 1988.

[45] S.Abiteboul. Updates, a new frontier. In *Second International Conference on Database Theory– ICDT'88*, pages 1–18, 1988.

[46] S.Abiteboul, R.Hull, and V.Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[47] S.Manchanda. Declarative expression of deductive database updates. In *Proceedings of the 8th International Symposium ACM - PODS*, pages 93–100, 1989.

[48] S.Manchanda and D.S.Warren. A logic-based language for database updates. In J.Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 363–394. Morgan Kaufmann, 1988.

[49] T.Eiter and G.Gottlob. Complexity aspects of various semantics for disjunctive databases. In *Proceedings of the 12th ACM-PODS Symposium*, pages 158–168, 1993.

[50] T.Przymusinski. On the declarative semantics of deductive databases and logic programs. In J.Minker, editor, *Foundations of Deductive Databases and Logic Programming.* Morgan Kaufmann, 1987.

[51] U.S.Chakravarthy, J.Grant, and J.Minker. Foundations of semantic query optimization for deductive databases. In *Foundations of Deductive Databases and Logic Programming*, pages 243–273. Morgan Kaufmann, 1987.

9 Appendix

Theorem 3.2.1. If the rules in the *IDB* are safe, the instantiation of variables and the generation of the set S of ground facts relevant for the update request, is always finite. □

Proof: The proof is inductive on the number of strata in which an intentional predicate is defined. We will assume that the stratum for the facts in the *EDB* is 0.

Base case (stratum 1) Observe one of the definitions of $p(\overline{X})$ where $\overline{X}$ denotes a vector of variables. Without loss of generality, we can assume that the variables in the body of the particular rule are arranged as shown:

$$p(\overline{X}) \leftarrow \wedge_i q_i(\overline{X_i}, \overline{Z_i}) \wedge_j \neg r_j(\overline{X_j}, \overline{Z_j})$$

where $\overline{X_i}, \overline{X_j}$ are sub vectors of $\overline{X}$, and $\overline{Z_i}, \overline{Z_j}$ are sub vectors of $\overline{Z}$, the vector of local variables. Since negation is safe, every $z_j \in \overline{Z_j}$ must appear in some $\overline{Z_i}$ of the given rule.

If the update request is *insert* $p(\overline{a})$ the formula becomes:

$$\mathcal{F}_q(p(\overline{a})) \equiv \ldots \bigvee (\exists \overline{Z})(\wedge_i q_i(\overline{a_i}, \overline{Z_i}) \wedge_j \neg r_j(\overline{a_j}, \overline{Z_j})) \bigvee \ldots$$

Clearly, the values for $\overline{Z_j}$ for which $\neg r_j(\overline{a_j}, \overline{Z_j})$ is true in the EDB can be finitely determined, and applying any other substitution for the variables in $\overline{Z}$ that makes $q_i(\overline{a_i}, \overline{Z_i})$ *true* will make $\neg r_j(\overline{a_j}, \overline{Z_j})$ *true* by default (completion).

If the request is *delete* $p(\overline{b})$ we have:

$$\mathcal{F}_q(\neg p(\overline{b})) \equiv \ldots \bigwedge (\forall \overline{Z})(\vee_i \neg q_i(\overline{b}, \overline{Z_i}) \vee_j r_j(\overline{b_j}, \overline{Z_j})) \bigwedge \ldots$$

Again, since the negation is safe, any variable z_k that appears in $\overline{Z_j}$ must appear in some $\overline{Z_i}$. From the current state of the EDB all such z_k for which $\neg q_i(\overline{b_i}, \overline{Z_i})$ is *false* (i.e. the tuples for which $q_i(\overline{b_i}, \overline{Z_i})$ holds) can be finitely determined. Inserting those tuples of $r_j(\overline{b_j}, \overline{Z_j})$ plus the default reasoning for the rest of the potentially infinite universe of $(\forall \overline{Z})$ will surely satisfy the update request.

Inductive hypothesis Assume that Theorem 3.2.1. is valid for any intentional predicate which stratum is $\leq n$.

Inductive step Let s be an intentional predicate from the stratum $n+1$. Then the definitions of s must be of the form:

$$s(\overline{X}) \leftarrow \wedge_i q_{s_i}(\overline{X_i}, \overline{Z_i}) \wedge_j \neg r_{s_j}(\overline{X_j}, \overline{Z_j})$$

In case the request is *insert* $s(\overline{a})$, we have:

$$\mathcal{F}_q(s(\overline{a})) \equiv \ldots \bigvee (\exists \overline{Z})(\wedge_i q_{s_i}(\overline{a_i}, \overline{Z_i}) \wedge_j \neg r_{s_j}(\overline{a_j}, \overline{Z_j})) \bigvee \ldots$$

Since the database is stratified *stratum* r_{s_j} < *stratum* s. Since we have assumed that there are no recursive definitions, either *stratum* q_{s_i} < *stratum* s, or q_{s_i} can be finitely substituted in terms of some $q_{s_i}^{m_k}$ which strata are $\leq n$. Thus, we fall into the case of the inductive hypothesis.

The case of update request *delete* $s(\overline{b})$ is similar, in terms of the form of $\mathcal{F}_q(\neg s(\overline{b}))$.

This completes the proof of the Theorem 3.2.1. $\square$.

Theorem 3.3.1. Let $DB_{new} = EDB_{new} \cup IDB$ and $DB_{new} \vdash ur$. EDB_{new} is P-minimal *iff* Algorithm LGS produces it as output. $\square$

Let $\neg A = \{\neg a \mid a \in A\}$ where $\neg(\neg x) = x$. Before we prove the theorem, let us observe a property that we will state as a separate lemma.

Lemma 3.3.2. If $EDB_m = (EDB \setminus D_m) \cup I_m$ is a minimal extensional database for a given update request and a current state $\mathcal{D} = EDB \cup IDB$, then $D_m \subseteq \neg N_p$ and $I_m \subseteq P_a$. $\square$

Proof: Assume the opposite, and let $\neg D_m \cap S = C_d$; $\neg D_m \setminus S = S_d$; $I_m \cap S = C_i$ and $I_m \setminus S = S_i$. EDB_m must satisfy at least one of the disjuncts in $\mathcal{F}_{DNF}(ur)$. But since all the members of S are obtained from $\mathcal{F}(ur) \equiv \mathcal{F}_{DNF}(ur)$, obviously $EDB_c = (EDB \setminus C_d) \cup C_i$ is sufficient, which contradicts the assumption of minimality of the EDB_m (unless $S_d = S_i = \{\}$). $\square$

Proof:(of Theorem 3.3.1.)**:**($\rightarrow$) Let $EDB_{new} = (EDB \setminus D_n) \cup I_n$ be the output of Algorithm. LGS. Assume there exists some $EDB' = (EDB \setminus D') \cup I'$ such that $EDB' \cup IDB \vdash \mathcal{F}(ur)$ and $EDB' \leq_P EDB_{new}$. Then either:

1. $EDB \setminus EDB' \subset EDB \setminus EDB_{new}$.

2. or

   - $(EDB \setminus EDB' = EDB \setminus EDB_{new})$ AND
   - $(EDB' \setminus EDB \subseteq EDB_{new} \setminus EDB)$.

Since the EDB is definite, $EDB \setminus EDB' = D'$; $EDB \setminus EDB_{new} = D_n$; $EDB' \setminus EDB = I'$ and $EDB_{new} \setminus EDB = I_n$.

Assume (1.) holds, which is equivalent to $D' \subset D_{new}$. From the construction of algorithm *LGS*, if EDB_{new} is an output, it must have been the case that at a certain instance of the first *while* loop, every element of the $\neg D_n$ appears in every disjunct of the current $\mathcal{F}''_{DNF}(ur)$. Let $d \in D_n \setminus D'$. Since $d \in D_n$, it must be the case that $\neg d$ appears in every disjunct of the current $\mathcal{F}''_{DNF}(ur)$. But, since $d \notin D'$, it follows that $(EDB \setminus D') \cup P_a \not\vdash \mathcal{F}''_{DNF}(ur)$ and, according to Lemma 3.3.2. $(EDB \setminus D') \cup I' \not\vdash \mathcal{F}''_{DNF}(ur)$ which contradicts the assumption that $EDB' = (EDB \setminus D') \cup I'$ can satisfy the update request.

In case (2.) holds, we have $D' = D_n$ and $I' \subseteq I_n$. Assume $I' \subset I_n$. If $EDB_{new} = (EDB \setminus D_n) \cup I_n$ is an output, it must have been the case that at a certain instance of the second *while* loop, every $x_i \in I_n$ appeared in every disjunct of the current form of the $\mathcal{F}''_{DNF}(ur)$. Let $x \in I_n \setminus I'$. Since $x \in I_n$ it must appear in every disjunct of the $\mathcal{F}''_{DNF}(ur)$, consequently $EDB' = (EDB \setminus D_n) \cup I' \not\vdash \mathcal{F}''_{DNF}(ur)$ and we have a contradiction as in the previous case (unless $I' = I_n$).

($\longleftarrow$) Let $EDB' = (EDB \setminus D') \cup I'$ be a minimal one, such that $EDB' \cup IDB \vdash \mathcal{F}(ur)$. Due to Lemma 3.3.2. it must be the case that $\neg D' \subseteq N_p$ and $I' \subseteq P_a$. Since we have already proved the soundness of the algorithm, all we have to do is observe a run which selects the elements of $N_p \setminus \neg D'$ in the first *while* loop, and the elements of $P_a \setminus I'$ in the second one.

This completes the proof of Theorem 3.3.1. □.

Lemma 4.1. If the database satisfies the requirements Rq1 and Rq2, the output of the algorithm *IC-LGS* does not violate any integrity constraint. □

Proof: (Sketch) Let the set of integrity constraints be $\mathcal{IC} = \{IC_1, \ldots IC_N\}$ and assume that Rq1 and Rq2 are satisfied. Consider an insertion of an extensional atom, say q which set of relevant integrity constraints is given by $\{(\neg R_1^{q1} \vee \ldots \vee \neg R_{n1}^{q1}), \ldots, (\neg R_1^{qk} \vee \ldots \vee \neg R_{nk}^{qk})\}$, where each R_g^h is a literal. According to Algorithm *IC-LGS*, the following formula is obtained:

$$\mathcal{F}^{IC}_{q-DNF} \equiv \ldots \bigvee \{q \wedge \neg R_{j1}^{q1} \wedge \neg R_{j2}^{q2} \ldots \wedge \neg R_{jn}^{qn}\} \bigvee \ldots$$

Assume that $\mathcal{F}^{IC}_{q-DNF}$ is satisfied exactly by inserting the disjunct above $D_j \equiv q \wedge \neg R_{j1}^{q1} \wedge \neg R_{j2}^{q2} \ldots \wedge \neg R_{jn}^{qn}$ and that there exists some $IC_u \equiv \neg N_1 \vee \ldots \vee \neg N_n \vee P_1 \vee \ldots \vee P_p$ which is violated. If the database were in the correct state before the update, the only reason for violation of IC_u is the insertion of some N_j or deletion of some P_j, or a combination of possibly more of them, enforced by D_j. Consider the case of insertion of N_j being the reason for violation of IC_u ($DB_{new} \not\models \neg N_j$). The only reason that N_j appears in D_j above is that there exists some integrity constraint of the form: $IC_i \equiv \neg q \vee N_j \vee Rest_i$. But then, due to Rq2, since IC_u and IC_i can be resolved, their resolvent would generate some $IC_l \equiv \neg q \vee \neg N_1 \vee \ldots \vee \neg N_{j-1} \vee \neg N_{j+1} \vee \ldots \vee \neg N_n \vee P_1 \vee \ldots \vee P_p \vee Rest_i$. Now, IC_l becomes relevant for the insertion of q, consequently D_j must have an atom from it (other than q). If any $\neg N_k$ or P_k is a member of D_j, there is nothing to prove because IC'_us satisfaction is enforced. In case D_j contains some atom from the $Rest_i$, since we have already assumed that N_j is in D_j we reach a contradiction on the assumption that insertion of q is realized through D_j. Namely, from the structures of IC_i and IC_l it is obvious that in $\mathcal{F}^{IC}_{q-DNF}$ there will be another disjunct, say D_m which completelly subsumes D_j and does not contain N_j. Thus, Algorithm LGS as a last step of IC–LGS would have never allowed for N_j to be inserted, consequently, IC_u will remain satisfied. The other cases can be analyzed in a simmilar manner. $\square$