TLP **3** (3): 287–327, May 2003. © 2003 Cambridge University Press
DOI: 10.1017/S1471068402001631 Printed in the United Kingdom

On the abductive or deductive nature of database schema validation and update processing problems

ERNEST TENIENTE and TONI URPÍ

Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, Jordi Girona Salgado 1-3, Barcelona, Catalonia
(*e-mail:* {teniente,urpi}@lsi.upc.es)

Abstract

We show that database schema validation and update processing problems such as view updating, materialized view maintenance, integrity constraint checking, integrity constraint maintenance or condition monitoring can be classified as problems of either abductive or deductive nature, according to the reasoning paradigm that inherently suites them. This is done by performing abductive and deductive reasoning on the event rules (Olivé, 1991), a set of rules that define the difference between consecutive database states. In this way, we show that it is possible to provide methods able to deal with all these problems as a whole. We also show how some existing general deductive and abductive procedures may be used to reason on the event rules. In this way, we show that these procedures can deal with all database schema validation and update processing problems considered in this paper.

KEYWORDS: database updating, database validation, abduction, deduction

1 Introduction

It is largely agreed that databases should contain, at least, base facts, deductive rules (views), integrity constraints and, sometimes, conditions to monitor since these features, together with appropriate reasoning capabilities, facilitate program development and reuse (Grant and Minker, 1992). Base facts represent extensional information; while deductive rules, integrity constraints and conditions to monitor represent intentional information, i.e. information that can be inferred from the extensional one. In particular, deductive rules define common knowledge shared by different users; integrity constraints define conditions that each state of the database is required to satisfy and conditions to monitor define information whose changes must be notified to the user.

Database schema validation has become an important problem in database engineering, particularly since databases are able to define intentional information. Schema validation refers to the process of verifying whether a database schema correctly and adequately describes the user's intended needs and requirements (Adrion *et al*., 1982). In general, preventing possible flaws during schema design will prevent those flaws from materializing as execution time errors or inconveniences.

Among the typical problems related to database schema validation, we have satisfiability checking, redundancy of integrity constraints, view liveliness, etc. (Bry and Manthey, 1986; Levy and Sagiv, 1995; Decker *et al.*, 1996).

Databases must also include an update processing system that provides users with a uniform interface through which they can request different kinds of updates, e.g. updates of base facts or updates of derived facts. In the presence of intentional information, updating a database is not a simple task since many issues have to be taken into account (Abiteboul, 1988). Therefore, an important amount of research has been devoted to different database updating problems like view updating (Kakas and Mancarella, 1990; Guessoum and Lloyd, 1990; Teniente and Olivé, 1995; Console *et al.*, 1995; Decker, 1996; Lobo and Trajcevski, 1997), materialized view maintenance (Gupta and Mumick, 1995; Roussopoulos, 1998), integrity constraint checking (Sadri and Kowalski, 1988; Küchenhoff, 1991; Olivé, 1991; García *et al.*, 1994; Lee and Ling, 1996; Staudt and Jarke, 1996), integrity constraint maintenance (Moerkotte and Lockemann, 1991; Ceri *et al.*, 1994; Wüthrich, 1993; Schewe and Thalheim, 1994; Teniente and Olivé, 1995) or condition monitoring (Rosenthal *et al.,* 1989; Hanson et al., 1990; Qian and Widerchold, 1991; Baralis *et al.,* 1998).

Up to now, the general approach of the research related to database schema validation and update processing has been to provide specific methods for solving particular problems. Therefore, if we were interested in integrating these problems into current database technology, we should incorporate several methods into commercial database management systems. In our opinion, this is one of the main reasons of the difficulty of translating the huge amount of research in this area into practical applications.

Solving these problems requires reasoning about the effect of an update on the database. Therefore, all these methods are either explicitly or implicitly based on a set of rules that define the changes that occur in a transition from an old state of a database to a new one, which is obtained as a result of the application of a certain transaction consisting of a set of base fact updates.

Several authors have argued about the advantages of making explicit the rules that define changes on the database contents induced by the application of a transaction when dealing with database updating problems (Bry, 1990; Teniente and Urpí, 1995; Denecker and De Schreye, 1998). These rules allow to guarantee that the updated database is as close as possible to the old database (which is the traditional assumption considered in database updating) and provide a high level of expressiveness since they allow to reason jointly about both states involved in the update (which is specially useful when dealing with database updating problems).

On the other hand, the role of deduction and abduction as reasoning paradigms is widely accepted. For instance, deduction has been used for query processing, while abduction has been applied to fault diagnosis, planning or default reasoning. In the context of databases, abductive procedures have been proposed for view updating (Bry, 1990; Console *et al.*, 1995; Decker, 1996) or satisfiability checking (Denecker and De Schreye, 1998). However, we do not know precisely how many forms of reasoning are necessary for solving known database problems nor, in

general, which database problems can be considered of deductive or of abductive nature, according to the reasoning paradigm that is more naturally suited to solve them.

In this paper, we show that database schema validation and update processing problems can be classified as either of deductive or of abductive nature. This is done by considering the event rules (Olivé, 1991), a particular case of rules that define the exact difference among consecutive database states, and by showing that reasoning deductively or abductively on these rules allows us to naturally specify and handle database problems. As we will see, problems like materialized view maintenance, integrity constraint checking or condition monitoring will be considered as naturally deductive, while problems like view updating, integrity constraint maintenance or enforcing condition activation as naturally abductive.

This first result is an evolution of our earlier work (Teniente and Urpí, 1995), where we proposed two *ad hoc* procedures to reason about the event rules that allowed us to classify database updating problems. We extend this work by showing that, in fact, we do not need *ad hoc* procedures but that we can consider general reasoning paradigms like deduction and abduction. It follows from this result that general deductive and abductive procedures can be used to reason about the event rules and, hence, to deal with database schema validation and update processing problems.

We also show how some existing general deductive and abductive procedures may be used to reason on the event rules. In this way, we show that these procedures can be used to deal with all database schema validation and update processing problems considered in this paper. This is illustrated by means of examples and we also point out some additional benefits gained by these procedures when reasoning on the event rules. Note that our goal is not that of comparing existing procedures but to show how to use them to reason on the event rules.

Finally, we sketch how the event rules could be used to solve general abductive problems in addition to database schema validation and update processing problems.

Problems related to views, integrity constraints and conditions to be monitored will be encountered whenever we have a database able to deal with intentional information. We have developed our ideas for the particular case of deductive databases due to their clear and precise notation. However, our framework (and, thus, our conclusions) can be easily generalized to all kinds of databases containing views, integrity constraints and conditions like, for instance, relational, active, object or object-relational databases.

This paper is organized as follows. The next section reviews basic concepts of deductive databases. Section 3 shortly presents the concepts of event, transition rules and event rules. Section 4 defines deductive and abductive reasoning on the event rules. Section 5 describes the most important problems related to schema validation and update processing and classifies them as either of deductive or of abductive nature. Section 6 shows how to use general deductive and abductive procedures to reason on the event rules. Section 7 sketches the use of the event rules to solve general abductive problems. Finally, in section 8 we present our conclusions and point out future work.

2 Basic definitions and notation

We briefly review the basic concepts of deductive databases (Lloyd, 1987; Ullman, 1989). We consider a first order language with a universe of constants, a set of variables, a set of predicate names and no function symbols. We will use names beginning with a capital letter for predicate symbols and constants and names beginning with a lower case letter for variables.

A *term* is a variable symbol or a constant symbol. We assume that possible values for terms range over finite domains. If P is an m-ary predicate symbol and $t_1, \ldots, t_m$ are terms, then $\mathrm{P}(t_1, \ldots, t_m)$ is an *atom*. The atom is *ground* if every t_i $(i = 1, \ldots, m)$ is a constant. A *literal* is defined as either an atom or a negated atom.

A *fact* is a formula of the form: $P(t_1, \ldots, t_m) \leftarrow$, where $P(t_1, \ldots, t_m)$ is a ground atom. We will omit the arrow when denoting an atom.

A *deductive rule* is a formula of the form: $P(t_1, \ldots, t_m) \leftarrow L_1 \wedge \cdots \wedge L_n$, with $m \geqslant 0, n \geqslant 1$, where $P(t_1, \ldots, t_m)$ is an atom denoting the *conclusion* and $L_1, \ldots, L_n$ are literals representing *conditions*. $P(t_1, \ldots, t_m)$ is called the *head* and $L_1 \wedge \cdots \wedge L_n$ the *body* of the deductive rule. Variables in the conclusion or in the conditions are assumed to be universally quantified over the whole formula. The definition of a predicate P is the set of all rules in the database which have P in their head. We assume that the terms in the head are distinct variables.

An *integrity constraint* is a formula that every state of the database is required to satisfy. We deal with constraints in *denial* form: $\leftarrow L_1 \wedge \cdots \wedge L_n$, with $n \geqslant 1$, where the L_i are literals and all variables are assumed to be universally quantified over the whole formula. We associate to each integrity constraint an inconsistency predicate *Icn*, where n is a positive integer, and thus they have the same form as deductive rules. Then, we would rewrite the former denial as $Ic1 \leftarrow L_1 \wedge \cdots \wedge L_n$. We call them *integrity rules*. More general constraints can be transformed into denial form by applying the procedure described in Lloyd and Topor (1984).

We also assume that the database contains a distinguished derived predicate *Ic* defined by the n clauses: $Ic \leftarrow Icn$. That is, one rule for each integrity constraint *Ici*, $i = 1 \ldots n$, of the database. Note that *Ic* will only hold in those states of the database that violate some integrity constraint and that it will not hold for those states that satisfy all constraints.

A *condition to be monitored* is a formula of the form: $Cond(t_1, \ldots, t_m) \leftarrow L_1 \wedge \cdots \wedge L_n$, with $m \geqslant 0, n \geqslant 1$, where $Cond(t_1, \ldots, t_m)$ is an atom and $L_1, \ldots, L_n$ are literals. Moreover, variables in *Cond* and in $L_1, \ldots, L_n$ are assumed to be universally quantified over the whole formula. Each condition to be monitored corresponds to a derived predicate for which certain changes have to be notified to the database user.

A *deductive database* D is a tuple D = (EDB, IDB) where EDB is a set of facts, and IDB = DR ∪ IC ∪ Cond is a set of rules such that DR is a set of deductive rules, IC a set of integrity rules and Cond a set of conditions to be monitored. The set EDB of facts is called the *extensional* part of the deductive database and the set IDB of rules is called the *intentional* part. We say that a deductive database is consistent if predicate *Ic* does not hold on it, i.e. if no integrity constraint is violated.

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

We assume that deductive database predicates are partitioned into base and derived (view) predicates. A base predicate appears only in the extensional part and (possibly) in the body of deductive rules. A derived predicate appears only in the intentional part. Base and derived facts correspond to facts about base and derived predicates, respectively.

As usual, we require that the deductive database before and after any updates is *allowed*, that is, any variable that occurs in a deductive rule, integrity rule or condition to be monitored has an occurrence in a positive literal of the body of the rule. In this paper, we deal with hierarchical[1] databases. Note that, in particular, this kind of databases does not allow to express recursive rules.

3 The event rules

Intuitively, a database is a dynamic system that changes over time. Changes are effected by EDB updates. These updates define a transition from an old state of the database to a new updated one. In this sense, schema validation and update processing problems can be viewed as database state transition problems. It is possible to define a set of rules that explicitly defines the possible transitions in terms of the difference between consecutive database states. Reasoning about these rules will allow to reason jointly about both states involved in the transition and, thus, to reason about the effect of the updates.

The event rules explicitly define the difference between two consecutive database states. In the following, we shortly review the concepts and terminology of event rules, as presented in Olivé (1991), and we discuss the possible use of other sets of rules instead of the event rules.

3.1 Events

Let D^o be a deductive database, T a transaction and D^n the updated deductive database. We say that T induces a transition from D^o (the old state) to D^n (the new state). We assume for the moment that T consists of an unspecified set of base facts to be inserted and/or deleted.

Due to the deductive rules, T may induce other updates on some derived predicates. Let P be one of such predicates, and let P^o and P^n denote the same predicate evaluated in D^o and D^n, respectively. Assuming that a fact $P^o(\boldsymbol{C})$ holds in D^o, where $\boldsymbol{C}$ is a vector of constants, two cases are possible:

a.1. $P^n(\boldsymbol{C})$ also holds in D^n.
a.2. $P^n(\boldsymbol{C})$ does not hold in D^n.

and assuming that $P^n(\boldsymbol{C})$ holds in D^n, two cases are also possible:

b.1. $P^o(\boldsymbol{C})$ also holds in D^o.
b.2. $P^o(\boldsymbol{C})$ does not hold in D^o.

[1] Equivalent to nr-datalog-¬ rules, and with the same expressive power as the relational calculus (Abiteboul *et al.*, 1995).

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

In case a.2 we say that a deletion event occurs in the transition, and we denote it by $\delta P(\boldsymbol{C})$. In case b.2 we say that an insertion event occurs in the transition, and we denote it by $\iota P(\boldsymbol{C})$.

Formally, we associate to each predicate P an *insertion event predicate* ιP and a *deletion event predicate* δP, defined as:

$$(1)\ \forall \mathbf{x}\ (\iota P(\boldsymbol{x}) \leftrightarrow P^n(\boldsymbol{x}) \wedge \neg P^o(\boldsymbol{x}))$$
$$(2)\ \forall \mathbf{x}\ (\delta P(\boldsymbol{x}) \leftrightarrow P^o(\boldsymbol{x}) \wedge \neg P^n(\boldsymbol{x}))$$

where $\mathbf{x}$ is a vector of variables. From the above, we then have the equivalencies:

$$(3)\ \forall \mathbf{x}\ (P^n(\boldsymbol{x}) \leftrightarrow (P^o(\boldsymbol{x}) \wedge \neg \delta P(\boldsymbol{x})) \vee \iota P(\boldsymbol{x}))$$
$$(4)\ \forall \mathbf{x}\ (\neg P^n(\boldsymbol{x}) \leftrightarrow (\neg P^o(\boldsymbol{x}) \wedge \neg \iota P(\boldsymbol{x})) \vee \delta P(\boldsymbol{x}))$$

We say that an event ιP or δP is a base event if P is a base predicate. Otherwise, it is a derived event. If P is a base predicate, then ιP and δP facts represent insertions and deletions of base facts, respectively. Therefore, we assume from now on that a transaction T consists of an unspecified set of insertion and/or deletion base event facts. If P is a derived predicate, an integrity constraint or a condition to be monitored, ιP and δP facts represent induced insertions and induced deletions on P, respectively.

3.2 Transition rules

Let us consider a derived predicate P of the database. The definition of P consists of the rules in the deductive database having P in the conclusion. Assume that there are $m (m \geqslant 1)$ such rules. For notation's sake, we rename predicate symbols in the conclusions of the m rules by $P_1, \ldots, P_m$, replace the implication by an equivalence and add the set of rules:

$$(5)\ P \leftarrow P_i \qquad i = 1 \ldots m$$

i.e. one rule defining P for each derived predicate P_i, $i = 1 \ldots m$.

Consider now one of the rules $P_i(\boldsymbol{x}) \leftrightarrow L_1 \wedge \cdots \wedge L_n$. When this rule is to be evaluated in the new state, its form is $P_i^n(\boldsymbol{x}) \leftrightarrow L_1^n \wedge \cdots \wedge L_n^n$, where L_j^n $(j = 1 \ldots n)$ is obtained by replacing the predicate Q of L_j by Q^n. Then, if we replace each literal in the body by its equivalent expression given in (3) or (4) we get a new rule which defines the new state predicate P_i^n in terms of old state predicates and events.

More precisely, if L_j^n is a positive literal $Q_j^n(\boldsymbol{x}_j)$ we apply (3) and replace it by:

$$\left(Q_j^o(\boldsymbol{x}_j) \wedge \neg\, \delta Q_j(\boldsymbol{x}_j)\right) \vee \iota Q_j(\boldsymbol{x}_j)$$

and if L_j^n is a negative literal $\neg Q_j^n(\boldsymbol{x}_j)$ we apply (4) and replace it by:

$$\left(\neg Q_j^o(\boldsymbol{x}_j) \wedge \neg \iota Q_j(\boldsymbol{x}_j)\right) \vee \delta Q_j(\boldsymbol{x}_j)$$

After distributing $\wedge$ over $\vee$, we get the set of *transition rules* for P_i^n. Notice that there are 2^{ki} such rules (where k_i is the number of literals in the P_i^n rule), and that literals in each rule can be of three types: old database literals, base event literals and derived event literals.

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

Example 1

Consider the rule $Cont_1(x) \leftrightarrow Sign(x) \wedge \neg Fail\text{-}ex(x)$ stating that contracted people are those who signed an agreement and did not failed the exam. In the new state, this rule has the form $Cont_1^n(x) \leftrightarrow Sign^n(x) \wedge \neg Fail\text{-}ex^n(x)$. Then, replacing $Sign^n(x)$ and $\neg Fail\text{-}ex^n(x)$ by their equivalent expressions given by (3) and (4) we get:

$$Cont_1^n(x) \leftrightarrow [(Sign^o(x) \wedge \neg \delta Sign(x)) \vee \iota Sign(x)] \wedge [(\neg Fail\text{-}ex^o(x) \wedge \neg \iota Fail\text{-}ex(x)) \vee \delta Fail\text{-}ex(x)]$$

and, after distributing $\wedge$ over $\vee$, we get the following transition rules:

$$Cont_1^n(x) \leftarrow Sign^o(x) \wedge \neg \delta Sign(x) \wedge \neg Fail\text{-}ex^o(x) \wedge \neg \iota Fail\text{-}ex(x)$$
$$Cont_1^n(x) \leftarrow Sign^o(x) \wedge \neg \delta Sign(x) \wedge \delta Fail\text{-}ex(x)$$
$$Cont_1^n(x) \leftarrow \iota Sign(x) \wedge \neg Fail\text{-}ex^o(x) \wedge \neg \iota Fail\text{-}ex(x)$$
$$Cont_1^n(x) \leftarrow \iota Sign(x) \wedge \delta Fail\text{-}ex(x)$$

Intuitively, it is not difficult to see that the first rule states that $Cont(X)$ will be true in the new state of the database if $Sign(X)$ was true in the old state, $Fail\text{-}ex(X)$ was false in the old state and no change of $Sign(X)$ and $Fail\text{-}ex(X)$ is given by the transition. In a similar way, the second rule states that $Cont(X)$ will be true in the new state if $Sign(X)$ was true and it has not been deleted and if $Fail\text{-}ex(X)$ has been deleted during the transition. A similar, intuitive, interpretation can be given for the third and fourth rules.

For simplicity of presentation, we will omit the subscript when a predicate P is defined by only one rule and we will omit the superscript ^{o} for denoting old database predicates.

3.3 Insertion and deletion event rules

Let P be a derived predicate. Insertion and deletion event rules of predicate P are defined, respectively, as:

$$(6)\ \iota P(\boldsymbol{x}) \leftarrow P^n(\boldsymbol{x}) \wedge \neg P^o(\boldsymbol{x})$$
$$(7)\ \delta P(\boldsymbol{x}) \leftarrow P^o(\boldsymbol{x}) \wedge \neg P^n(\boldsymbol{x})$$

where P refers to the current (old) database state and P^n refers to the transition rule of P. These event rules define the induced changes that happen in a transition from an old state of a database to a new, updated state. Note that they depend only upon the rules of the database, being independent of the stored facts and of any particular transaction.

We wish to point out that these rules can be intensively simplified, as described in Olivé (1991) and Urpí and Olivé (1992, 1994). By simplifying the event rules, we obtain a set of semantically equivalent rules, but with a lower evaluation cost. The automatic generation and simplification of the event rules has been implemented in a SunOS environment by means of Quintus Prolog. In this paper, we consider the simplified version of the event rules.

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

Definition 1
Let D = (EDB, IDB) be a deductive database. The *augmented database* associated to D is the tuple A(D) = (EDB, IDB*), where IDB* contains the rules in DR ∪ IC ∪ Cond and their associated simplified transition and event rules.

Example 2
Given the following database D = (EDB, IDB):

$$\text{Sign(John)}$$
$$\text{Fail-ex(John)}$$
$$\text{Cont(x)} \leftarrow \text{Sign(x)} \wedge \neg\, \text{Fail-ex(x)}$$

the corresponding augmented database A(D) is the following:

$$\text{Sign(John)}$$
$$\text{Fail-ex(John)}$$
$$\text{Cont(x)} \leftarrow \text{Sign(x)} \wedge \neg\text{Fail-ex(x)}$$
$$\iota\text{Cont(x)} \leftarrow \text{Sign(x)} \wedge \neg\delta\text{Sign(x)} \wedge \delta\text{Fail-ex(x)}$$
$$\iota\text{Cont(x)} \leftarrow \iota\text{Sign(x)} \wedge \neg\, \text{Fail-ex(x)} \wedge \neg\iota\text{Fail-ex(x)}$$
$$\iota\text{Cont(x)} \leftarrow \iota\text{Sign(x)} \wedge \delta\text{Fail-ex(x)}$$
$$\delta\text{Cont(x)} \leftarrow \text{Cont(x)} \wedge \iota\text{Fail-ex(x)}$$
$$\delta\text{Cont(x)} \leftarrow \delta\text{Cont(x)} \wedge \neg\, \text{Fail-ex(x)}$$

3.4 Using other rules instead of the event rules

We will use the event rules to provide the basis of our framework for specifying and handling schema validation and update processing problems. However, since our framework is only based on the definition of event given in (1) and (2) (see section 3.1), we could use any set of rules that defines the difference between consecutive states of the database, instead of the event rules, provided that they preserve the event definition. In this section, we show two different sets of rules that could had been used also.

Assume that we have the following deductive rules:

$$\text{P(x)} \leftarrow \text{Q(x)} \wedge \text{R(x)}$$
$$\text{R(x)} \leftarrow \text{S(x)}$$

If we simply consider the definition of transition and event rules without applying any simplification, we would have:

$$\iota\text{P(x)} \leftarrow \text{P}^n\text{(x)} \wedge \neg\text{P(x)}$$
$$\iota\text{R(x)} \leftarrow \text{R}^n\text{(x)} \wedge \neg\text{R(x)}$$
$$\delta\text{R(x)} \leftarrow \text{R(x)} \wedge \neg\text{R}^n\ \text{(x)}$$
$$\text{P}^n\text{(x)} \leftarrow \text{Q(x)} \wedge \neg\delta\text{Q(x)} \wedge \text{R(x)} \wedge \neg\delta\text{R(x)}$$
$$\text{P}^n\text{(x)} \leftarrow \text{Q(x)} \wedge \neg\delta\text{Q(x)} \wedge \iota\text{R(x)}$$
$$\text{P}^n\text{(x)} \leftarrow \iota\text{Q(x)} \wedge \text{R(x)} \wedge \neg\delta\text{R(x)}$$
$$\text{P}^n\text{(x)} \leftarrow \iota\text{Q(x)} \wedge \iota\text{R(x)}$$
$$\text{R}^n\text{(x)} \leftarrow \text{S(x)} \wedge \neg\delta\text{S(x)}$$
$$\text{R}^n\text{(x)} \leftarrow \iota\text{S(x)}$$

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

On the other hand, by adapting the rules generated by Küchenhoff (1991) to our terminology, we would obtain:

$$\iota P(x) \leftarrow Q(x) \wedge \neg\delta Q(x) \wedge \iota R(x) \wedge \neg P(x)$$
$$\iota P(x) \leftarrow \iota Q(x) \wedge R(x) \wedge \neg\delta R(x) \wedge \neg P(x)$$
$$\iota P(x) \leftarrow \iota Q(x) \wedge \iota R(x) \wedge \neg P(x)$$
$$\iota R(x) \leftarrow \iota S(x)$$
$$\delta R(x) \leftarrow \delta S(x)$$

Note that Küchenhoff's rules provide some simplifications with regards to the non-simplified event rules. For instance, insertion event rules about R do not check that R was false in the old state of the database. A similar simplification is given for deletion event rules about R. However, no simplification is applied for the rules defining events on P.

Finally, the simplified insertion event rules for P according to Urpí and Olivé (1992) are the following:

$$\iota P(x) \leftarrow Q(x) \wedge \neg\delta Q(x) \wedge \iota R(x)$$
$$\iota P(x) \leftarrow \iota Q(x) \wedge R(x) \wedge \neg\delta R(x)$$
$$\iota P(x) \leftarrow \iota Q(x) \wedge \iota R(x)$$
$$\iota R(x) \leftarrow \iota S(x)$$
$$\delta R(x) \leftarrow \delta S(x)$$

Note that, in addition to the simplifications already provided by Küchenhoff, these rules include also simplifications involving the event rules for P.

It is not difficult to see that the three sets of rules are semantically equivalent and define the same transitions between consecutive database states. The event definition is preserved in all cases. The only difference relies on the kind of optimizations that have been considered. In fact, we could also think about other sets of rules that incorporate additional optimizations (see, for instance, Urpí and Olivé (1994), which incorporates the knowledge provided by the integrity constraints into such set of rules). The differences among possible sets of rules imply advantages or inconveniences as far as efficiency is concerned, but not regarding the ability of solving the problems we deal with in this paper. Thus, our framework does not depend on any particular set of rules.

4 Reasoning on the event rules

There is a big amount of problems related to database schema validation and to update processing. Unfortunately, the general approach considered by previous research in this area has been to deal with each problem in an isolated way. So, it is unusual to find a method able to handle several of the previous problems. This limitation can be overcome by considering a set of rules that explicitly define the difference between two consecutive database states and by performing deductive and abductive reasoning on these rules.

The role of deduction and abduction as reasoning paradigms is widely accepted. *Deduction* is an analytic process based on the application of general rules to

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

particular cases, with the inference of a result. *Abduction* is another form of synthetic reasoning which infers the case from the rule and the result.

The event rules define the exact changes, either on base as on derived predicates, produced as a consequence of the application of a given transaction to a database state. Deductive and abductive reasoning can be performed on these rules. As we will see, performing deductive reasoning on the event rules defines changes on derived predicates induced by changes on base predicates. On the other hand, performing abductive reasoning on the event rules defines changes on base predicates needed to satisfy changes on derived predicates. In this way, reasoning deductively or abductively on the event rules provides the basis for solving database schema validation and update processing problems in a uniform way.

In fact, as stated in section 3.4, any set of rules that precisely defines the difference between consecutive database states could also be used. Due to our previous experience with the event rules, we have considered them in this paper.

4.1 Reasoning deductively on the event rules

Deduction is concerned with inferring consequences from facts via deductive rules. For instance, given a deductive rule $P(x) \leftarrow Q(x)$ and a fact Q(A), deduction infers P(A) as a consequence of Q(A). Thus, deductive reasoning is suitable among other things for finding correct answers to queries.

Definition 2
Let D = (EDB, IDB) be a deductive database and G a goal $L_1 \wedge \cdots \wedge L_n$. A *correct answer* to G over EDB is a substitution θ for variables of G such that $G\theta$ is a logical consequence of EDB $\cup$ IDB, i.e. EDB $\cup$ IDB $\models G\theta$.

Since event rules define the changes that occur in a transition from an old state of a database to a new one as a consequence of the application of a given transaction, by considering deduction in the context of the augmented database we can also define how to reason deductively on the event rules.

Definition 3
Let D = (EDB, IDB) be a deductive database, A(D) = (EDB, IDB*) its corresponding augmented database, T a transaction consisting of a set of ground base event facts, u a derived event. The *deduced consequences on* u due to the application of T is the set θ of correct answers to EDB $\cup$ IDB* $\cup$ T $\cup$ u. Note that each correct answer $\theta_i \in \theta$ defines a ground derived event $u\theta_i$ induced as a consequence of the application of T.

Thus, reasoning deductively on the event rules defines changes on derived predicates induced by changes on base predicates, since θ defines all ground facts about u induced by the application of T to the current state of the database.

As an example, given the database of Example 2 and the transaction $T = \{\delta \text{Fail-ex(John)}\}$, it is not difficult to see that reasoning deductively on the event rules allows to deduce that T induces ιCont(John), i.e. $\theta = \{x = \text{John}\}$.

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

4.2 Reasoning abductively on the event rules

Abduction is aimed at looking for hypothesis that explain a given observation, according to known laws usually specified by means of deductive rules. Abduction (in the absence of integrity constraints) is traditionally defined as follows: Given a set of sentences T (a theory presentation) and a sentence G (observation), the abductive task consists of finding a set of sentences Δ (abductive explanation for G) such that:

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

It is usually considered that Δ consists of atoms drawn from predicates explicitly indicated as abducible (those whose instances can be assumed when required). Therefore, an *abductive framework* is a pair $\prec T, Ab \succ$, where Ab is the set of abducible predicates, i.e. $\Delta \subseteq Ab$[2].

By considering these ideas in the context of the augmented database, we can also define how to reason abductively on the event rules. In this case, abducible predicates correspond to base event facts since this is the only possible way to physically update a database.

Definition 4

Let us consider a deductive database D = (EDB, IDB) and its corresponding augmented database A(D) = (EDB, IDB*). We can define an associated abductive framework $\prec$EDB $\cup$ IDB*, $Ab\succ$, where Ab corresponds to the set of base event predicates. Now, given a ground derived event u, we can define an *abductive explanation for* u in $\prec$EDB $\cup$ IDB*, $Ab\succ$ to be any set T_i consisting of ground facts about predicates in Ab such that:

$$-\ \text{EDB} \cup \text{IDB}^* \cup T_i \models u$$

An explanation T_i is *minimal* if no proper subset of T_i is also an explanation, i.e. if it does not exist any explanation T_j for u such that $T_j \subset T_i$.

The previous condition states that the update request is a logical consequence of the database updated according to T_i. Thus, abductive reasoning on the event rules defines changes on base predicates needed to satisfy a given change on a derived predicate.

As an example, given the database of Example 2 and the derived event ιCont(John), it is not difficult to see that $T = \{\delta\text{Fail-ex(John)}\}$ is a minimal abductive explanation for ιCont(John). That is, the insertion of Cont(John) is satisfied by the deletion of Fail-ex(John).

In general, the result of applying abductive reasoning may not be unique. That is, several sets T_i of base event facts that satisfy a derived event may exist. Each possible set T_i constitutes a possible transaction that applied to the database will accomplish the desired change on the derived predicate. Minimal explanations are usually of particular interest, specially when we deal with database updating problems since

[2] Here and in the rest of the paper we use Ab to indicate both the set of abducible predicates and the set of all their variable-free instances.

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

they allow to minimize the difference between the old and the new states of the database.

If no solution T_i is obtained then either the requested update cannot be satisfied only by changes on the EDB or the current database state already satisfies the intended effect of the request (e.g. an insertion of P is requested into a database that already satisfies P). Note that in the latter case we do not obtain a solution with the empty set since, when taking the event definition into account, an insertion of P can only be explained if the old state of the database does not satisfy P (resp. a deletion of P can only be explained if P is satisfied).

With this framework, the basic strategy of a proof procedure for computing T_i is the following. Given, for instance, an update request for inserting P, the update procedure tries to solve abductively the goal $\leftarrow \iota P$ in $\prec \text{EDB} \cup \text{IDB}^*, Ab \succ$ generating a set T_i of abducibles such that T_i satisfies the above condition. Before abducing a base event, we have to check that its definition is satisfied. That is, for abducing an event δP we require P to be true in the old state of the database while for ιP we require P to be false. The set T_i generated by abduction for an update request u constitutes a transaction that, applied to the current state of the EDB, will satisfy u.

Given an event (base or derived) u_i to be explained, abductive reasoning on the event rules can also be used to determine sets of base events that ensure that a certain derived event u_j is not induced by the explanations of u_i. In this case, the abductive interpretation defines changes on base predicates needed to satisfy that a certain change on a derived predicate does not occur as a consequence of the application of the explanations of u_i.

Definition 5

Let D be a deductive database D = (EDB, IDB), its corresponding augmented database A(D) = (EDB, IDB*) and its associated abductive framework $\prec$EDB $\cup$ IDB*, $Ab \succ$. Now, given a positive event u_i and a negative derived event $\neg u_j$ such that $u_i \wedge \neg u_j$ is allowed, we can define the abductive explanation for $u_i \wedge \neg u_j$ in $\prec$ EDB $\cup$ IDB*, $Ab \succ$ to be any set T_i consisting of ground facts about predicates in Ab such that:

$$- \text{EDB} \cup IDB^* \cup T_i \models u_i$$
$$- \text{EDB} \cup IDB^* \cup T_i \not\models u_j$$

The first condition states that u_i is a logical consequence of the database updated according to T_i, while the second states that u_j it is not. Note that if the explanations of u_i alone do not induce u_j, then they are already valid abductive explanations.

As an example, given the database of Example 2 and the positive event ιSign(Mary) and the negative event $\neg\iota$Cont(Mary), we have that $T = \{\iota\text{Sign(Mary)}, \iota\text{Fail-ex (Mary)}\}$ is a minimal abductive explanation for ιSign(Mary) $\wedge \neg\iota$Cont(Mary). Note that ιSign(Mary) alone would induce ιCont(Mary). However, adding ιFail-ex(Mary) to T does not induce ιCont(Mary) any more since Cont(Mary) will be false in the new database state.

Two special cases are of particular interest. First, when the negative derived event is $\neg\iota Ic$ it is guaranteed that the obtained explanations do not induce any insertion

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

of an integrity constraint. Thus, the new database state will be consistent if the old database state was already consistent. Secondly, if the positive event is base (i.e. a transaction T), abductive reasoning on the event rules determines possible sets S_i of ground base event facts that, appended to T, ensure that the application of any of the resulting transactions $T_i = S_i \cup T$ does not induce u_j. Note that if T alone does not induce u_j, then T itself is a valid transaction.

This framework can be easily generalized to reason abductively on sets of positive and negative events.

5 Deductive or abductive nature of database problems

Deduction and abduction provide a uniform way to reason about the event rules and, in general, about any set of rules that explicitly define the exact difference between two consecutive database states. Moreover, either views (i.e. derived predicates) or integrity constraints or conditions to be monitored are uniformly defined by means of deductive rules and they are only distinguished by the different semantics endowed to the head of the rule. Thus, a view defines common knowledge shared by different users, an integrity constraint defines a situation that must never happen and a condition to be monitored defines an information whose changes must be reported to the user.

Therefore, given a derived predicate P(x) defined by the rule $P(x) \leftarrow Q(x) \wedge \neg R(x)$, P can be expressed as:

$$\begin{aligned} &View(x) \leftarrow Q(x) \wedge \neg R(x) \\ &Ic1(x) \leftarrow Q(x) \wedge \neg R(x) \\ &Cond(x) \leftarrow Q(x) \wedge \neg R(x) \end{aligned}$$

according to the concrete semantics that we would like to endow to P.

Now, reasoning deductively or abductively on the event rules corresponding to View, Ic1 and Cond we may classify as naturally deductive or naturally abductive the database schema validation and update processing problems.

This is summarized in Figure 1. Each row corresponds to the form of reasoning to be applied to the event rules of P and to the relevant events about P (i.e. ιP, δP, $T \wedge \neg \iota P$ or $T \wedge \neg \delta P$; being T a transaction) to reason about. Each column considers a different semantics to be endowed to P. Finally, each resulting cell defines a possible database schema validation or update processing problem that can be specified in terms of that form of reasoning and of the considered semantics.

In the rest of this section, we briefly review database schema validation and update processing problems and we show how they can be handled by means of deductive or abductive reasoning, according to the classification provided in Figure 1.

5.1 Schema validation and update processing problems

The correct use of a database involves three different tasks: *Schema Validation*, to guarantee that the database schema satisfies the user's intended needs and requirements; *Query Processing*, to be able to give efficient and correct answers

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

		View	Ic	Cond
Deduced Consequences on *(deductive reasoning)*	ιP	Materialized view maintenance	IC checking	Condition Monitoring
	δP		Checking consistency restoration	
Abductive **Explanation for** *(abductive reasoning)*	ιP	View updating	Redundancy of Integrity Constraints	Enforcing condition activation
	δP	View liveliness	Repairing inconsistent DB Satisfiability Checking	Condition validation
	T ∧ ¬ιP	Preventing side effects	IC maintenance	Preventing condition activation
	T ∧ ¬δP		Maintaining DB inconsistency	

Fig. 1. Classification of database problems.

to the user' queries; and *Update Processing*, to be able to correctly perform updates to the database contents. In general, several problems may arise when dealing with each of these tasks (Teniente, 2000). We will consider here only the problems encountered during schema validation and update processing, since query processing is beyond the scope of this paper.

Example 3

Consider the following flawed database schema to be validated:

(DR.1)	Some-cand ← Cand(x)
(DR.2)	Emp(x) ← Cand(x) ∧ Cont(x)
(DR.3)	Cont(x) ← Sign(x) ∧ ¬Fail-ex(x)
(DR.4)	App(x) ← Cand(x)
(IC.1)	Ic1(x) ← App(x) ∧ Sign(x)
(IC.2)	Ic2(x) ← App(x) ∧ ¬Has-account(x)
(IC.3)	Ic3(x) ← ¬Some-cand
(IC.4)	Ic4(x) ← Cand(x) ∧ ¬App(x)
(IC.5)	Ic5(x) ← App(x)
(IC.6, . . . , IC.10)	Ic ← Ici, for i = 1 . . . 5
(Cond.1)	Cond1(x) ← Cand(x) ∧ ¬Cont(x)
(Cond.2)	Cond2(x) ← Emp(x) ∧ ¬Cont(x)

This schema defines four derived predicates (through deductive rules DR.1 to DR.4): Some-cand, Emp (employee), Cont (contracted person) and App (applicant). A person is an applicant if he is a candidate (Cand). A person is contracted if he signed an agreement (Sign) and he did not fail the exam. Employees are candidates that have a contract. Finally, Some-cand is true if the database contains, at least, one candidate.

The schema contains also five integrity constraints (defined by integrity rules IC.1 to IC.5). Ic1 states that it is not possible to have applicants that have signed an agreement. Ic2 states that it is not possible to be applicant and not to have an account. Ic3 states that there must be some candidate. Ic4 states that it is not possible to be candidate and not be applicant. Finally, Ic5 states that the database may not contain any applicant.

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

The schema contains also two conditions to be monitored. The first one is used to notify changes on the populations of applicants that do not have a contract and the second one changes on the populations of employees that do not have a contract.

5.2 Schema validation problems

In general, we cannot be completely sure that a certain database schema adequately describes the structure of the information that we want the database to contain. At first glance, we could perhaps detect that a certain deductive rule or integrity constraint is not precisely defined, as it might happen with IC.5 above, but it is very difficult to assess whether a certain schema does not present critical flaws. Detecting and removing flaws during schema design time will prevent these flaws from materializing as run-time errors or other inconveniences during operation time. Decker *et al.* (1996) identified several desirable properties that a database schema should satisfy.

5.2.1 Satisfiability checking

A database schema is *satisfiable* if there exists an EDB for which no integrity constraint is violated (Bry and Manthey, 1986), also mentioned in Bry *et al.* (1988) and Inoue *et al.* (1992). Clearly, a non-satisfiable schema is not useful, since it does not accept any extensional information.

As an example, the previous database schema is not satisfiable for any EDB. The empty EDB is not a proper EDB since it violates Ic3. Then, we need to consider an EDB with at least one candidate, let us say John. However, this is not enough since Ic4 would then be violated. So, John must also be an applicant but this is not possible since Ic5 impedes it. As a consequence of detecting that the previous schema is not satisfiable, we assume that the database designer decides to discard Ic3 and Ic5.

Satisfiability checking can be naturally specified as performing abductive reasoning on the event rules associated to δIc provided that Ic holds with an empty EDB. If there exists at least one abductive explanation for δIc in $\prec$EDB $\cup$ IDB*, $Ab\succ$, with IDB = DR $\cup$ IC, then the integrity constraints are satisfiable. Note that if Ic does not hold in the state corresponding to the empty EDB, all constraints are already satisfied in that state.

Note that, since satisfiability checking is to be determined at schema validation time, we are considering the empty EDB for checking this property. For the same reason, we will also use the empty EDB for checking other problems related to database schema validation.

5.2.2 Absolute redundancy of an integrity constraint

Intuitively, an integrity constraint is absolutely redundant if integrity does not depend upon it. That is, if it can never be violated. Obviously, an absolute redundant

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

integrity constraint is not useful since it does not add any additional information to the information already provided by the rest of the schema.

For instance, integrity constraint Ic4 is absolutely redundant since the deductive rule DR.4 prevents Ic.4 to be violated for any EDB. Therefore, the database designer has to modify the schema to remove this absolute redundancy. In this case, we assume that he decides to discard DR.4.

Given an integrity constraint Ic_i, *absolute redundancy* can be naturally specified as performing abductive reasoning on the event rules associated to ιIc_i. If there exists at least one abductive explanation for ιIc_i in $\prec$EDB $\cup$ IDB*, $Ab\succ$, with IDB = DR $\cup$ IC, then Ic_i is not absolutely redundant since it can be violated in some state of the database. In particular, in the database state that we obtain as a result of applying the obtained abductive explanation.

5.2.3 *View liveliness*

A derived predicate P (i.e. a view) is *lively* if there exists an EDB in which at least one fact about P is true. That is, predicates which are not lively correspond to views that are empty in each possible state of the database. Such predicates are clearly not useful and probably ill-specified. This definition of 'liveliness' essentially coincides with the definition of 'satisfiable' in Levy and Sagiv (1995).

For instance, predicate Emp as defined in Example 3 is not lively. The reason is that a fact Emp(X) requires Cand(X) and Sign(X) to be true at the same time. However, since nobody can be a candidate without being an applicant (Ic.4) and nobody can be an applicant and to have signed an agreement (Ic.1), no person X can be an employee. We assume that the database designer decides to correct this flaw by redefining Ic.1 as Ic1′(x) ← App(x) ∧ Sign(x) ∧ ¬ Has-account(x).

In our framework, provided that no fact about View holds in the empty EDB, *view liveliness* can be specified as reasoning abductively on the event rules of ιView(**x**). If there exists at least one abductive explanation for a certain event ιView(**X**) in $\prec$EDB $\cup$ IDB*, $Ab\succ$, then View is lively since it is possible to reach a state where at least a fact View(**X**) is true. Otherwise, View is not lively. Note that if some fact about View holds in the empty EDB, then View is already lively in that state and there is no reason to ask about this property.

Let us review the Example 3 with the flaws corrected up to now:

(DR.1)	Some-cand ← Cand(x)
(DR.2)	Emp(x) ← Cand(x) ∧ Cont(x)
(DR.3)	Cont(x) ← Sign(x) ∧ ¬Fail-ex(x)
(IC.1')	Ic1(x) ← App(x) ∧ Sign(x)∧ ¬Has-account(x)
(IC.2)	Ic2(x) ← App(x) ∧ ¬ Has-account(x)
(IC.4)	Ic4(x) ← Cand(x) ∧ ¬App (x)
(IC.6, IC.7, IC.9)	Ic ← Ici, for i = 1, 2, 4.
(Cond.1)	Cond1(x) ← Cand(x) ∧ ¬Cont (x)
(Cond.2)	Cond2(x) ← Emp(x) ∧ ¬Cont (x)

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

5.2.4 Relative redundancy of integrity constraints

Relative redundancy is similar to absolute redundancy but, in this case, an integrity constraint (or a set of constraints) is relatively redundant if it is always satisfied in all states that satisfy the rest of the constraints. Again, such a redundancy should be detected and redundant constraints should not be considered during update processing.

In our example, we can see that Ic1′ is relatively redundant, since it is entailed by Ic2. Therefore, we assume that the database designer decides to discard Ic.1′ since the resulting database will admit the same consistent states.

Given an integrity constraint Ic_i, *relative redundancy* can be naturally specified as performing abductive reasoning on the event rules associated to ιIc_i. Ic_i is not relatively redundant if there exists at least one abductive explanation for $\leftarrow \iota Ic_i \wedge \neg \iota \mathrm{Ic}$ in $\prec \mathrm{EDB} \cup \mathrm{IDB}^*, Ab \succ$, with $\mathrm{IDB} = \mathrm{DR} \cup \mathrm{IC} - \{\mathrm{Ic} \leftarrow Ic_i\}$.

5.2.5 Condition validation

Condition validation refers to the problem of determining whether it is possible to change the contents of a certain condition Cond(x). That is, to determine whether exists at least one transaction that, if applied to the database, could activate a certain condition $\iota\mathrm{Cond}(\mathbf{X})$ or $\delta\mathrm{Cond}(\mathbf{X})$. Clearly, a condition that can never be activated is probably ill-specified since the active behaviour of the database does not depend on it. This can be useful, for instance, to provide the database designer with a tool for validating certain aspects of the condition definition and, hence, of the active behaviour of the database. This problem is very important in the context of active databases since this technology is mainly based on the extensive use of conditions to be monitored, which are the core of Condition-Action (CA) and Event-Condition-Action (ECA) rules (Widom and Ceri, 1996).

For instance, condition Cond2 as defined in Example 3 is not valid, since no insertion and no deletion can be induced on it. The reason is that, by the deductive rule DR.2, employees must have a contract and, then, it is not possible to have employees without a contract. So, we assume that the database designer decides to discard condition Cond2.

In a similar way that view liveliness, changes induced in a given condition, Cond(x), can be specified as reasoning abductively on the event rules of $\iota\mathrm{Cond}(\mathbf{x})$ and $\delta\mathrm{Cond}(\mathbf{x})$. If there exists at least one abductive explanation for a certain event $\iota\mathrm{Cond}(\mathbf{X})$ or $\delta\mathrm{Cond}(\mathbf{X})$ in $\prec \mathrm{EDB} \cup \mathrm{IDB}^*, Ab \succ$, then the condition can be activated.

5.3 Update processing problems

Once the database schema is validated, we are ready to perform updates to the database contents. Several problems will arise when processing the requested updates (Teniente and Urpí, 1995). To illustrate these problems, we consider the schema we have previously validated and we will assume that the database contains several base facts. Note that now the schema is satisfiable, all predicates are lively, and no integrity constraint is either absolutely nor relatively redundant.

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

Example 4

The following database will be considered to deal with update problems related to views and integrity constraints:

(F.1)	Sign(John)
(F.2)	Fail-ex(John)
(DR.1)	Some-cand ← Cand(x)
(DR.2)	Emp(x) ← Cand(x) ∧ Cont(x)
(DR.3)	Cont(x) ← Sign(x) ∧ ¬Fail-ex(x)
(IC.2)	Ic2(x) ← App(x) ∧ ¬Has-account(x)
(IC.4)	Ic4(x) ← Cand(x) ∧ ¬App (x)
(IC.7, IC.9)	Ic ← Ici, for i = 2, 4.
(Cond.1)	Cond1(x) ← Cand(x) ∧ ¬Cont (x)

5.3.1 *Integrity constraint checking*

There exists a large cumulative effort in the field of *integrity constraint checking* (Sadri and Kowalski, 1988; Küchenhoff, 1991; Olivé, 1991; García *et al*., 1994; Lee and Ling, 1996; Staudt and Jarke, 1996). Given a consistent database and a transaction (i.e. a set of insertions and deletions of base facts), integrity constraint checking is devoted to incrementally, i.e. efficiently, determine whether the application of this transaction to the current database violates some integrity constraint. In this case, the transaction is rejected since, otherwise, its application would lead to an inconsistent database state.

Given a transaction T, *integrity constraint checking* can be naturally specified in our framework as performing deductive reasoning on the event rules associated to ιIc, provided that Ic does not hold. The deduced consequences on ιIc are either the identity substitution or no correct answer of EDB ∪ IDB* ∪ T ∪ ¬ιIc exists. In the first case, T induces an insertion of Ic and, therefore, it must be rejected because it violates some integrity constraint. Otherwise, T does not violate any integrity constraint and it can be successfully applied. As it happens with materialized view maintenance, efficiency of the process is ensured since reasoning about the transaction and the event rules allows to compute only the updates induced by this transaction.

As an example, assume that the database of Example 4 also contains the facts App(Peter) and Has-account(Peter). Reasoning deductively on the event rules of Ic2 we could determine that the transaction T = {δHas-account(Peter)} induces the insertion of Ic2(Peter), and thus of Ic, and would lead the database to an inconsistent state.

5.3.2 *Integrity constraint maintenance*

The main drawback of integrity constraint checking is that the user may not know which changes to the transaction are needed to guarantee that its application does not violate any integrity constraint. *Integrity constraint maintenance* is aimed at overcoming this drawback: given a consistent database state and a transaction T that violates some integrity constraint, the problem is to find *repairs*, i.e. an additional

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

set of insertions and/or deletions of base facts to be appended to T such that the resulting transaction T' satisfies all integrity constraints. In general, there may be several repairs and the user must select one of them. Eventually, if no such repair exists then the original transaction must be rejected. Several methods have been proposed to deal with this problem (Moerkotte and Lockemann, 1991; Ceri *et al.*, 1994; Wüthrich, 1993; Schewe and Thalheim, 1994; Teniente and Olivé, 1995).

Given a consistent database state and a transaction T, *integrity constraint maintenance* can be specified in our framework as performing abductive reasoning on the goal $\leftarrow T \wedge \neg\iota\text{Ic}$. Thus, possible abductive explanations for $\text{T} \wedge \neg\iota\text{Ic}$ in $\prec \text{EDB} \cup \text{IDB}^*, Ab \succ$, with $\text{IDB} = \text{DR} \cup \text{IC}$, correspond to the possible transactions T', $\text{T} \subseteq \text{T}'$, that maintain database consistency.

As an example, consider again the database of Example 4, and assume that the transaction $\text{T} = \{\iota\text{App(Claire)}\}$ wants to be applied to the database. Reasoning abductively on $\iota\text{App(Claire)} \wedge \neg\iota\text{Ic}$ we obtain the transaction $\text{T}' = \{\iota\text{App(Claire)}, \iota\text{Has-account(Claire)}\}$ which satisfies the original transaction and maintains the database consistent.

5.3.3 *View updating*

View updating is concerned with determining how a request to update a view, i.e. to update the contents of a derived predicate, must be appropriately translated into updates of the underlying base facts. In general, several translations may exist and the user must select one of them. This problem has attracted much research during the last years in deductive databases (Kakas and Mancarella, 1990; Guessoum and Lloyd, 1990; Teniente and Olivé, 1995; Console *et al.*, 1995; Decker, 1996; Lobo and Trajcevski, 1997), and it has been already identified as an abductive problem (Console *et al.*, 1995; Decker, 1996; Denecker and De Schreye, 1998; Inoue and Sakama, 1999).

View updating can be naturally specified as performing abductive reasoning on the event rules of $\iota\text{View}(\mathbf{X})$ or $\delta\text{View}(\mathbf{X})$, where View($\mathbf{X}$) is the derived fact to be inserted or deleted, respectively. The abductive explanation for $\iota\text{View}(\mathbf{X})$ defines possible sets of base fact updates (i.e. transactions) that satisfy the insertion of View($\mathbf{X}$), while the abductive explanation for $\delta\text{View}(\mathbf{X})$ defines possible sets of base fact updates that satisfy the deletion of View($\mathbf{X}$).

For instance, in Example 4 reasoning abductively on the event rules of Cont we can determine that the view update request $\iota\text{Cont(John)}$ is satisfied by the transaction $\text{T} = \{\delta\text{Fail-ex(John)}\}$.

In principle, it may happen that some translations corresponding to a given view update request do not satisfy the integrity constraints. For this reason, view updating is usually combined with problems related to integrity constraints. Possible ways of performing this combination will be explained in section 5.5.

5.3.4 *Materialized view maintenance*

A view can be materialized by explicitly storing its contents in the extensional database. This can be useful, for instance, to improve efficiency of query processing.

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

Given a transaction, *materialized view maintenance* consists of incrementally (i.e. efficiently) determining which changes are needed to update accordingly the materialized views (see Gupta and Mumick (1995) and Roussopoulos (1998) for a state-of-the-art reports).

Given a transaction T and a materialized view View(**x**), *materialized view maintenance* can be naturally specified as performing deductive reasoning on the event rules associated to ιView(**x**) and δView(**x**). That is, deduced consequences for ιView(**x**) and for δView(**x**) correspond, respectively, to the insertions and to the deletions to be performed on View(**x**). Efficiency of the process is ensured, since reasoning about the transaction and the event rules allows to incrementally compute only the updates induced by this transaction.

For instance, if we assume that predicate Cont(x) in Example 4 is materialized, reasoning deductively on the event rules of Cont we can determine that the transaction $T = \{\delta$Fail-ex(John)$\}$ induces the insertion of Cont(John) in the materialized view.

5.3.5 *Preventing side effects*

Due to the deductive rules, undesired updates may be induced on some derived predicates when applying a transaction. We say that a side effect occurs when this happens. The problem of *preventing side effects* (Teniente and Olivé, 1995) is concerned with determining a set of base fact updates which, appended to a given transaction, ensure that the application of the resulting transaction to the current state of the database will not induce the undesired side effects. In general, several solutions may exist and the user must select one of them.

Ensuring that a transaction T will not induce an insertion or a deletion of a derived fact View(**X**) can naturally be specified as reasoning abductively on $\{T \wedge \neg\iota$View(**X**)$\}$ or on $\{T \wedge \neg\delta$View(**X**)$\}$, respectively. The former defines sets T′ of base fact updates, which are supersets of T, needed to guarantee that the insertion of View(**X**) is not induced by T, while the latter defines sets T′ of base fact updates, again supersets of T, needed to satisfy that the deletion of View(**X**) is not induced.

For instance, reasoning abductively on ιSign(Mary) $\wedge \neg\iota$Cont(Mary) we can prevent that the transaction $T = \{\iota$Sign(Mary)$\}$ will not induce the insertion of Cont(Mary). This is done by considering $T' = \{\iota$Sign(Mary), ιFail-ex(Mary)$\}$ instead of T, which is also given by this abductive interpretation.

5.3.6 *Condition monitoring*

Condition monitoring refers to the problem of incrementally monitoring the changes on a condition induced by a transaction that consists of a set of base fact updates (Rosenthal *et al.*, 1989; Hanson *et al.*, 1990; Qian and Widerhold, 1991; Baralis *et al.*, 1989).

As an example, applying the transaction $T = \{\iota$Cand(Peter)$\}$ to the database of Example 4 would induce ιCond1(Peter). That is, due to the application of T, Peter would be a candidate without a contract.

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

In our framework, changes induced in a condition Cond(x), are specified as performing deductive reasoning on the events rules associated to ιCond(**x**) and δCond(**x**). The former, ιCond(**x**), defines the changes meaning that x satisfy the condition after the application of the transaction, but not before. δCond(**x**) defines the changes meaning that x satisfy the condition before the application of the transaction, but not after.

5.3.7 Enforcing condition activation

Enforcing condition activation refers to the problem of obtaining the possible transactions that, if applied to the current state of the database, would induce an activation of a given condition. For instance, the transaction $T_1 = \{\iota\text{Cand(Peter)}\}$ would induce the condition ιCond1(Peter).

In our framework, enforcing condition activation is specified reasoning abductively on ιCond(**X**) or δCond(**X**), where both correspond to the conditions to be enforced. The former defines possible transactions that will induce **X** to satisfy the condition after their application, but not before. The latter, defines possible transactions that will induce **X** not to satisfy the condition after their application.

5.3.8 Preventing condition activation

This problem is close to the problem of preventing side effects but considering conditions to be monitored instead of views. Given a transaction T, the problem of *preventing condition activation* is to find an additional set of insertions and/or deletions of base facts to be appended to T such that the resulting transaction T′ guarantees that no changes in the condition would occur as a consequence of the application of T′. In general, several resulting transactions may exist and the user should select one of them.

5.4 Updates to an inconsistent database

Sometimes it could be useful to allow for intermediate inconsistent database states, i.e. states where some integrity constraint is violated. This may happen, for instance, to reduce the number of times that integrity constraint enforcement is performed. In this case, three new problems related to update processing arise.

5.4.1 Checking consistency restoration

Given an inconsistent database state and a transaction that consists of a set of base fact updates, the problem of *checking consistency restoration* is to incrementally check whether these updates restore the database to a consistent state.

Checking consistency restoration can be specified as performing deductive reasoning on δIc, provided that Ic holds. In this case, deduced consequences on δIc are also either the identity substitution or no correct answer exists. If the identity substitution is obtained, then the transaction induces a deletion of Ic and, therefore, restores the database to a consistent state.

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

5.4.2 Repairing an inconsistent database

Given an inconsistent database state, the problem of *repairing an inconsistent database* is to obtain a set of updates of base facts, i.e. a transaction, that restore the database to a consistent state. In general, several solutions may exist and the database administrator should select one of them.

The problem of *repairing an inconsistent database* can be specified as performing abductive reasoning on the event rules associated to δIc, provided that Ic holds. Given an EDB that violates some integrity constraint, abductive explanations for δIc in $\prec$EDB $\cup$ IDB*, $Ab\succ$, with IDB = DR $\cup$ IC, correspond to the possible transactions that would induce a deletion of Ic and that, therefore, would restore database consistency.

5.4.3 Maintaining database inconsistency

Given an inconsistent database state and a transaction T, the problem of *maintaining database inconsistency* is to obtain an additional set of base fact updates to be appended to the original transaction to guarantee that the resulting database state remains inconsistent.

Maintaining database inconsistency can be specified as performing abductive reasoning on the goal $\leftarrow T \wedge \neg\delta$Ic, provided that Ic holds, with an abductive framework $\prec$EDB $\cup$ IDB*, $Ab\succ$, with IDB = DR$\cup$IC. Although we do not see for the moment any practical application of this problem, it can be naturally classified and specified in the framework we propose in this paper.

5.5 Combining different problems

In previous sections, we have assumed that deductive or abductive reasoning is performed on the event rules associated to a single derived event predicate. However, this framework can be easily extended to consider several derived events instead of only one. Deductive or abductive reasoning on a set of derived events is performed by considering the conjunction of all derived events in the set as the goal to be reasoned about. For instance a view update request consists, in general, of a set of insertions and/or deletions to be performed on derived predicates, e.g. $u = \iota$P(a) $\wedge$ ιQ(b) $\wedge \delta$S(c) stands for the request of inserting P(a) and Q(b) and deleting S(c), being P, Q and S derived predicates. In this case, translations that satisfy u correspond to the abductive explanations for ιP(a) $\wedge$ ιQ(b) $\wedge \delta$S(c) in $\prec$EDB $\cup$ IDB*, $Ab\succ$.

Moreover, we would like to notice that deductive problems can be naturally combined. All of them share a common starting-point (a transaction that consists of a set of base fact updates) and aim at the same goal (to define the changes on derived predicates induced by this transaction). The same reasons allow the combination of abductive problems. Therefore, we can specify more complex database updating problems of deductive or of abductive by considering possible combinations of the problems specified in section 5.

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

For instance, given a transaction T, a materialized view View, a condition to be monitored Cond and the integrity constraint predicate Ic, we could combine materialized view maintenance, integrity constraints checking and condition monitoring by reasoning deductively on $\iota\text{View}(\mathbf{x}) \wedge \delta\text{Cond}(\mathbf{y}) \wedge \neg\iota\text{Ic}$. Deduced consequences on $\iota\text{View}(\mathbf{x}) \wedge \delta\text{Cond}(\mathbf{x}) \wedge \neg\iota\text{Ic}$ correspond to the values $\mathbf{x}$ and $\mathbf{y}$ that cause an insertion of View, satisfy the condition δCond as a consequence of the application of T, and such that T does not violate any integrity constraint.

In a similar way, we could also combine view updating with integrity constraints maintenance by reasoning abductively on $\iota\text{View(a)} \wedge \neg\iota\text{Ic}$. In this case, abductive explanations for $\iota\text{View(a)} \wedge \neg\iota\text{Ic}$ correspond to the translations that satisfy both the insertion of View(a) and that do not violate any integrity constraint.

Furthermore, we could also combine deductive and abductive problems. Note that the result of performing abductive reasoning is exactly the starting-point for performing deductive reasoning, that is, a transaction that consists of a set of base fact updates. Therefore, we could first deal with abductive problems and, immediately after, use the obtained result for dealing with the deductive ones.

For instance, we could be interested on distinguishing between integrity constraints to be maintained and integrity constraints to be checked, and on combining view updating with the treatment of both kinds of constraints. In this case, we should first reason abductively on the view update request and the set of integrity constraints to be maintained and, later on, to consider the resulting transactions and reason deductively on the set of integrity constraints to be checked to reject those resulting transactions that violate some constraint in this set.

Finally, we would also like to notice that in our approach for the specification of database updating problems does not change when considering other kinds of updates like insertions or deletions of deductive rules. In this case, we should first determine the changes on the transition and event rules caused by the update and apply then our approach.

6 Using existing procedures to reason on the event rules

Our framework to classify database schema validation and updating processing problems is based on the existence of a set of rules, like the event rules, that define the exact difference between consecutive database states. By performing deductive and abductive reasoning on these rules, we can deal with all these problems in a uniform way. Therefore, our framework does not rely on the concrete method we use to perform either deductive or abductive reasoning. However, candidate methods to be used must satisfy several conditions.

Given a method able to perform deductive reasoning in a certain class of deductive databases, it should satisfy two requirements to tackle the deductive problems described in the previous section:

(a) The class of deductive databases considered by the method must allow expressing at least the goals required to define these problems.
(b) The method must obtain all correct answers that satisfy the request.

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

Similarly, given a method able to perform abductive reasoning in a certain class of deductive databases, it should satisfy two requirements to tackle the abductive problems described in the previous section:

(a) The class of deductive databases considered by the method must allow expressing at least the goals required to define these problems.
(b) For schema validation problems, if there exists some explanation for a given request the method obtains such explanation (but not necessarily several or even all of them). For updating problems, the method must be complete, i.e. it must obtain all possible explanations that satisfy the request.

In this section we show that there exists already some procedures able to compute each different form of reasoning on the event rules and we illustrate them by means of some examples. We show, in this way, the applicability of our approach. We would like to mention, however, that our aim is not that of comparing existing procedures but just to show they are able to handle several problems.

6.1 Using SLDNF to reason deductively on the event rules

Standard SLDNF resolution is a possible way for reasoning deductively on the event rules. Given an augmented database $A(D) = (EDB, IDB^*)$, a transaction T and a derived event u, the deduced consequences on u due to T correspond to the successful SLDNF derivations of the goal $\leftarrow u$ that result in a computed answer θ_i when considering the input set $EDB \cup IDB^* \cup T$.

Nevertheless, other proof procedures could be used instead of SLDNF resolution like, for instance, bottom-up computation of the event rules. To motivate our discussion and without loss of generality, we will assume that SLDNF resolution is used to reason deductively on the event rules. The following example illustrates how to perform deductive reasoning on the event rules.

Example 5

Consider again Example 2 and assume a transaction T which consists of the deletion of the base fact Fail-ex(John), in our notation $T = \{\delta\text{Fail-ex(John)}\}$. The corresponding deductive framework is:

$EDB \cup IDB^*$: Sign(John)
Fail-ex(John)
$\text{Cont(x)} \leftarrow \text{Sign(x)} \wedge \neg\text{Fail-ex(x)}$
$\iota\text{Cont(x)} \leftarrow \text{Sign(x)} \wedge \neg\delta\text{Sign(x)} \wedge \delta\text{Fail-ex(x)}$
$\iota\text{Cont(x)} \leftarrow \iota\text{Sign(x)} \wedge \neg\text{Fail-ex(x)} \wedge \neg\iota\text{Fail-ex(x)}$
$\iota\text{Cont(x)} \leftarrow \iota\text{Sign(x)} \wedge \delta\text{Fail-ex(x)}$
$\delta\text{Cont(x)} \leftarrow \delta\text{Sign(x)} \wedge \neg\text{Fail-ex(x)}$
$\delta\ \text{Cont(x)} \leftarrow \text{Cont(x)} \wedge \iota\text{Fail-ex(x)}$
T: δFail-ex(John)

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

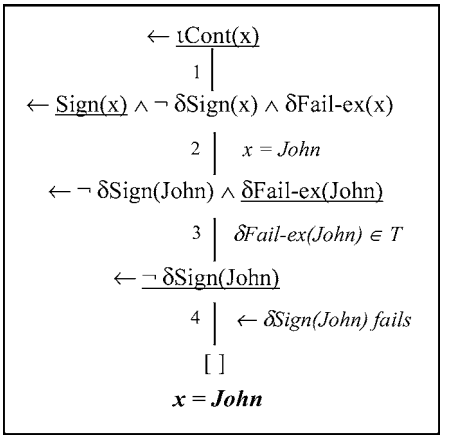

Fig. 2. Computing answers for ← ιCont(x).

The SLDNF refutation of Figure 2 shows that $T = \{\delta \text{Fail-ex(John)}\}$ induces the insertion of Cont(John), i.e. $\theta = \{x = \text{John}\}$. Note that the SLDNF tree rooted at ← ιCont(x) succeeds with a computed answer x = John. That means that the deletion of Fail-ex(John) induces an insertion of Cont(John). Selecting other rules at step 1 of this tree does not result on any other successful branch and, thus, no other insertion of Cont is induced due to T.

6.2 Using abductive procedures to reason abductively on the event rules

We show now how the Events Method (Teniente and Olivé, 1995), Inoue and Sakama's method (Inoue and Sakama 1998, 1999) and SLDNFA (Denecker and De Schreye, 1998) may be used to perform abductive reasoning on the event rules and, hence, to deal with schema validation and updating processing problems. Other existing procedures could have been used as well like, for instance, Kakas and Mancarella (1990), which was the first attempt to use abduction in a database context. However, we have just considered some of the most recent proposals since they can be understood in some sense as an evolution of the initial ones.

A detailed discussion on the specific features and limitations of other (abductive) methods to perform view updating and integrity maintenance can be found in Mayol and Teniente (1999).

6.2.1 The Events Method

The Events Method (Teniente and Olivé, 1995) takes the event rules explicitly into account to obtain all possible minimal sets T_i on the EDB that satisfy a given update request on the IDB. It extends the SLDNF proof procedure to obtain all possible transactions T_i and it has been proved to be sound and complete for stratified databases (Teniente and Olivé, 1995). Soundness of the method guarantees that the obtained transactions satisfy the update request, while completeness ensures that it obtains all minimal transactions.

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

In this method, an update request u is a conjunction of positive and negative events (base and derived). Positive events correspond to updates that must be effectively performed during the transition from the old state of the database to the new state, while negative events correspond to updates that may not happen during this transition.

Let D be a deductive database, A(D) its augmented database, u an update request and T_i a set of base events. In the Events Method, T_i satisfies the request u if, using SLDNF resolution, the goal $\leftarrow u$ succeeds from input set $A(D) \cup T_i$. Each set T_i is obtained by having some failed SLDNF derivation of $A(D) \cup \leftarrow u$ succeed. The possible ways in which a failed derivation may succeed correspond to the different sets T_i that satisfy the request. If no T_i is obtained, then it is not possible to satisfy the requested update by changing only the EDB.

Although the event rules define the exact difference between consecutive database states, making a failed SLDNF derivation succeed does not always guarantee the generation of minimal solutions only. Therefore, the Events Method includes also a final step to discard obtained non-minimal solutions. We must note that the Events Method may not terminate in the presence of recursive rules because it may enter into an infinite loop. Moreover, as far as efficiency is concerned, it provides certain limitations on the treatment of rules with existential variables.

Let u be an update request. In the Events Method, a transaction T satisfies u if there is a constructive derivation from $(\leftarrow u \varnothing \varnothing)$ to $([\,]\ T\ C)$. The transaction T contains the base event facts to be applied, while the condition set C contains base events (subgoals in the general case) that would invalidate the update request if applied.

A constructive derivation is defined as follows (for convenience, let G/L stand for the goal obtained from a goal G by dropping a selected occurrence of literal L in G).

Definition 6

A *constructive derivation* from $(G_1\ T_1\ C_1)$ to $(G_n\ T_n\ C_n)$ via a safe computation rule R (Lloyd, 1987) is a sequence:

$$(G_1\ T_1\ C_1), (G_2\ T_2\ C_2), \ldots, (G_n\ T_n\ C_n)$$

such that for each $i \geqslant 1$, G_i has the form $\leftarrow L_1 \wedge \cdots \wedge L_k$, $R(G_i) = L_j$ and $(G_{i+1}\ T_{i+1}\ C_{i+1})$ is obtained according to one of the following rules:

(A1) If L_j is a positive literal and it is not a base event, then $G_{i+1} = S$, where S is the resolvent of some clause in A(D) with G_i on the selected literal L_j, $T_{i+1} = T_i$ and $C_{i+1} = C_i$.

(A2) If L_j is a positive base event 'ιP' (resp. 'δP'), there is a substitution σ such that $P\sigma$ does not hold (resp. $P\sigma$ holds) in the current database and there is a consistency derivation from $(C_i\ T_i \cup \{L_j\sigma\}\ C_i)$ to $(\{\}\ T'\ C')$ then $G_{i+1} = G_i\sigma/L_j\sigma$, $T_{i+1} = T'$ and $C_{i+1} = C'$.
Note that if $C_i = \varnothing$ or $L_j\sigma \in T_i$ then $G_{i+1} = G_i\sigma \backslash L_j\sigma$, $T_{i+1} = T_i \cup \{L_j\sigma\}$ and $C_{i+1} = C_i$.

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

(A3) If L_j is negative and there is a consistency derivation from $(\{\leftarrow \neg L_j\}\ T_i\ C_i)$ to $(\{\}\ T'\ C')$, then $G_{i+1} = G_i/L_j$, $T_{i+1} = T'$ and $C_{i+1} = C'$.

Rule (A1) is an SLDNF resolution step where A(D) acts as input set. In rule (A2), the selected base event is included in the transaction set T_i to get a successful derivation for the current branch, provided that the event does not violate any of the conditions in C_i. In rule (A3), we get the next goal if we can ensure consistency for the selected literal.

Definition 7

A *consistency derivation* from $(F_1\ T_1\ C_1)$ to $(F_n\ T_n\ C_n)$ via a safe computation rule R is a sequence:

$$(F_1\ T_1\ C_1), (F_2\ T_2\ C_2), \ldots, (F_n\ T_n\ C_n)$$

such that for each $i \geqslant 1$, F_i has the form $H_i \cup F'_i$, where $H_i = \leftarrow L_1 \wedge \cdots \wedge L_k$ and, *for some* $j = 1 \ldots k$, $(F_{i+1}\ T_{i+1}\ C_{i+1})$ is obtained according to one of the following rules:

(B1) If L_j is a positive literal and it is not a base event, then $F_{i+1} = S' \cup F'_i$, $T_{i+1} = T_i$ and $C_{i+1} = C_i$; where S' is the set of all resolvents of clauses in A(D) with H_i on the selected literal L_j and $[] \notin S'$. Note that, if no input clause in A(D) can be unified with L_j, then $S' = \varnothing$ and $F_{i+1} = F'_i$.

(B2) If L_j is a positive base event, then $F_{i+1} = S' \cup F'_i$, $T_{i+1} = T_i$ and $C_{i+1} = C_i \cup \{H_i\}$; where S' is the set of all resolvents of clauses in T_i with H_i on the selected literal L_j and $[] \notin S'$. If L_j is ground then $C_{i+1} = C_i$.

(B3) If L_j is a negative literal, $\neg L_j$ is not a base event, $k > 1$ and there is a consistency derivation from $(\{\leftarrow \neg L_j\}\ T_i\ C_i)$ to $(\{\}\ T'\ C')$, then $F_{i+1} = \{H_i \backslash L_j\} \cup F'_i$, $T_{i+1} = T'$ and $C_{i+1} = C'$.

(B4) If L_j is a negative base event, $\neg L_j \notin T_i$ and $k > 1$, then $F_{i+1} = \{H_i \backslash L_j\} \cup F'_i$, $T_{i+1} = T_i$ and $C_{i+1} = C_i$.

(B5) If L_j is negative, and there is a constructive derivation from $(\{\leftarrow \neg L_j\}\ T_i\ C_i)$ to $([]\ T'\ C')$, then $F_{i+1} = F'_i$, $T_{i+1} = T'$ and $C_{i+1} = C'$.

Rules (B1) and (B2) are SLDNF resolution steps where A(D) or T act as input set, respectively. Rules (B3) and (B4) allow to continue with the current branch by ensuring that the selected literal L_j is consistent with respect to T_i and C_i. In rule (B5) the current branch is dropped if there exists a constructive derivation for the negation of the selected literal.

Consistency derivations do not rely on the particular order in which selection rule R selects literals, since in general, all possible ways in which a conjunction $\leftarrow L_1 \wedge \cdots \wedge L_k$ can fail should be explored. Each one may lead to a different transaction. As a result of this formulation, non-minimal solutions may be obtained. They are discarded by means of a simple procedure that rejects those that are a superset of the minimal ones.

Example 6

Consider again the same database as in Example 5 and assume now that the update request $\imath$Cont(John) is requested. The corresponding abductive framework is:

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

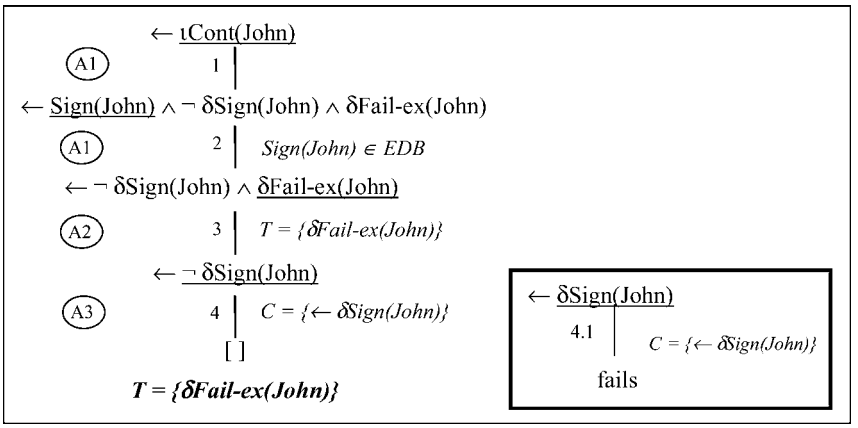

Fig. 3. Constructive derivation for ← ιCont(John).

EDB ∪ IDB*:	Sign(John)
	Fail-ex(John)
	Cont(x) ← Sign(x) ∧ ¬Fail-ex(x)
	ιCont(x) ← Sign(x) ∧ ¬δSign(x) ∧ δFail-ex(x)
	ιCont(x) ← ιSign(x) ∧ ¬Fail-ex(x) ∧ ¬ιFail-ex(x)
	ιCont(x) ← ιSign(x) ∧ δFail-ex(x)
	δCont(x) ← δSign(x) ∧ ¬ Fail-ex(x)
	δCont(x) ← Cont(x) ∧ ιFail-ex(x)
Ab:	{ιSign, δSign, ιFail-ex, δFail-ex}

Figure 3 shows that the abductive interpretation of the event rules of Cont provided by the Events Method computes the abductive explanation T = {δFail-ex(John)} for ιCont(John). This is done by performing a constructive derivation rooted at ← ιCont(John). Circled labels appearing at the left of the derivation are references to the rules of the Events Method we have just defined.

The Events Method starts from the update request ← ιCont(John) and uses SLDNF resolution pursuing the empty clause. Steps 1 and 2 are standard SLDNF resolution steps. At step 3, an abducible fact is selected. Then, it is included in the input set T and used as input clause if we want to get a successful derivation for ← ιCont(John).

Finally, at step 4, a negative base event literal is selected. Then, its corresponding subsidiary derivation must be considered, which is shown enclosed by the bold box. To ensure failure of this derivation it must be guaranteed that δSign(John) will not be included into T later on during the derivation process. This is achieved by means of an auxiliary set C that contains conditions to be satisfied during the whole derivation process. These conditions correspond to some of the goals reached in subsidiary derivations, as shown at step 4.1 of Figure 3, where the condition ← δSign(John) is included in C. Hence, before adding a base event to T we must enforce that it does not falsify any of the conditions of C.

Once the empty clause is reached in the primary derivation, the abductive procedure finishes and T gives the base events that, applied to the EDB, will satisfy

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

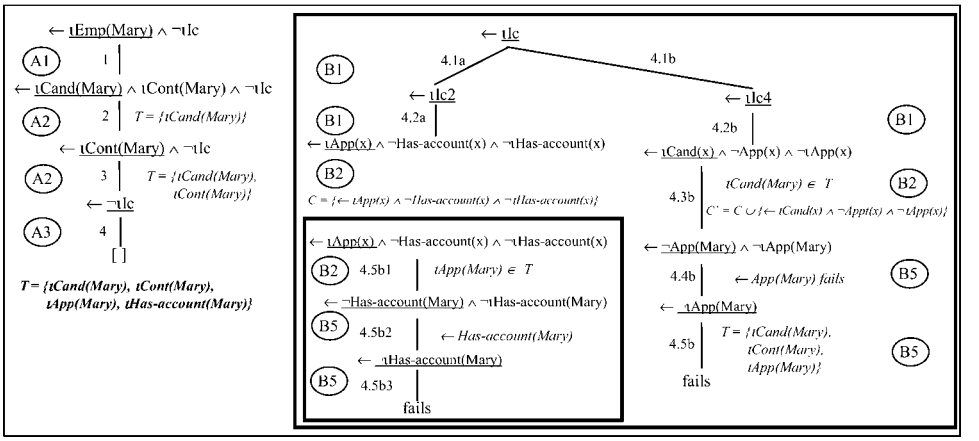

Fig. 4. Constructive derivation for ← ιEmp(Mary) ∧ ¬ ιIc.

the requested update. From this derivation we have that $T = \{\delta\text{Fail-ex(John)}\}$. Then, the request for inserting Cont(John) can be achieved by deleting the fact Fail-ex(John).

Example 7

Consider now the database of Example 4. The abductive framework relevant to this example is:

$\text{EDB} \cup \text{IDB}^*$: Sign(John)
Fail-ex(John)
Some-cand ← Cand(x)
Emp(x) ← Cand(x) ∧ Cont(x)
Cont(x) ← Sign(x) ∧ ¬Fail-ex(x)
Ic2(x) ← App(x) ∧ ¬Has-account(x)
Ic4(x) ← Cand(x) ∧ ¬App(x)
Ic ← Ic2(x)
Ic ← Ic4(x)
ιEmp(x) ← ιCand(x) ∧ ιCont(x)
ιIc ← ιIc2(x)
ιIc ← ιIc4(x)
ιIc2(x) ← ιApp(x) ∧ ¬Has-account(x) ∧ ¬ιHas-account(x)
ιIc4(x) ← ιCand(x) ∧ ¬App(x) ∧ ¬ιApp(x)

Ab: $\{\iota\text{Cand}, \delta\text{Cand}, \iota\text{Cont}, \delta\text{Cont}, \iota\text{Sign}, \delta\text{Sign}, \iota\text{Fail-ex}, \delta\text{Fail-ex}\ \iota\text{App}, \delta\text{App}, \iota\text{Has-account}, \delta\text{Has-account}\}$

Assume that we want to insert the derived fact Emp(Mary) without violating any integrity constraint. In this case, the abductive interpretation of ιEmp(Mary) ∧ ¬ιIc defines the possible sets T_i, $\{\iota\text{Emp(Mary)}\} \subseteq T_i$, that do not induce ιIc. This is shown in Figure 4.

As it happens in the Example 6, the Events Method starts from the update request ← ιEmp(Mary) ∧ ¬ιIc and uses SLDNF resolution pursuing the empty clause. In this case, base event facts ιCand(Mary) and ιCont(Mary) are included in

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

the translation set T_i during steps 2 and 3 of the primary constructive derivation. At step 4, the subsidiary consistency derivation (enclosed by the bold box) rooted at ιIc must fail finitely to get the empty clause in the constructive derivation. Steps 4.1a and 4.2a of this subsidiary derivation are SLDNF resolution steps. After this step, a failed goal is reached and it is included in the condition set C to guarantee that latter additions to T do not make it succeed.

Steps 4.1b to 4.4b are SLDNF resolution steps, with the extension that the goal is included in C at step 4.3b. At step 4.5b, we have, in turn, a subsidiary constructive derivation (not shown in the previous figure) which is handled in the same way as the primary constructive one. This derivation causes the inclusion of ιApp(Mary) into T. Moreover, it must be ensured that this inclusion does not violate any of the conditions in C. This is done by means of the subsidiary derivation rooted at $\leftarrow \iota$App(x) $\wedge \neg$Has-account(x) $\wedge \neg\iota$ Has-account(x) which, in turn, requires the inclusion of ιHas-account(Mary) into T (this is done in the constructive derivation associated to step 4.5b3, which is not shown in Figure 4).

Once the empty clause is reached, the abductive procedure finishes and T contains the base events that satisfy the requested update. From this derivation we have that $T_i = \{\iota$Cand(Mary), ιCont(Mary), ιApp(Mary), ιHas-account(Mary)$\}$. This is the only solution that satisfies the requested update in this example.

6.2.2 *Inoue and Sakama's Method*

In this proposal (Inoue and Sakama, 1998, 1999), an Abductive Logic Program (ALP) is defined as a pair $\prec$P, A$\succ$ where P is a normal logic program and A is a set of abducible atoms. Given an ALP, Inoue and Sakama define a set of production rules, its *transaction program* τP, that declaratively specifies addition and deletion of abductive hypothesis. Abductive explanations are then computed by the fixpoint of the transaction program using a bottom-up model generation procedure.

They consider two possible kinds of explanations: positive and negative. Given an ALP $\prec$P, A$\succ$ and an observation G, a set of hypothesis E is a *positive explanation* for G if P $\cup$ E $\models$ G and P $\cup$ E is consistent. Similarly, a set of hypothesis F is a *negative explanation* for G if P$\backslash$F $\models$ G and P$\backslash$F is consistent.

Moreover, they apply abduction also to 'unexplain' an observation (look for anti-explanations). Given a normal logic program P and an observation G, a set of hypothesis E is a *positive anti-explanation* for G if P $\cup$ E $\not\models$ G and E is consistent. Similarly, a set of hypothesis F is a *negative anti-explanation* for G if P$\backslash$F $\not\models$ G and E is consistent.

Their method has been shown sound and complete for covered acyclic normal logic programs. A normal logic program P is covered if, for every rule in P, all variables in the body appear in the head. In particular, this restriction does not allow having non-ground integrity constraints neither existential variables in the body of rules.

There is a clear correspondance between looking for (negative) explanations that explain/unexplain an observation and the kind of updates we can consider with the

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

event rules. In particular, a positive explanation (anti-explanation) E is a set of event facts to be inserted (i.e. a set of base insertion events) while a negative explanation (anti-explanation) F is a set of event facts to be deleted (i.e. a set of base deletion events). Moreover, to explain an observation G that does not hold in the current database corresponds to consider the update request ιG, to unexplain G (which holds in the current database) corresponds to consider the update request δG.

We may use Inoue and Sakama's method to reason abductively on the event rules. We have to:

1. Consider the ALP $\prec$P*, A$\succ$ where P* is the augmented database of P and A is the set of base event facts (both insertion and deletion).
2. Include the following rules in the transaction program τP*:
   – For each base fact q that belongs to the EDB, we have to include the rule: in(ιq) $\rightarrow$ false.
   – For each base fact q that does not belong to the EDB, we have to include the rule: in(δq) $\rightarrow$ false
   These rules are needed to distinguish event predicates from database ones and to guarantee that an event can be successfully applied.
   Furthermore, since events (and events facts) are handled in this way, we also require to apply rule 3 of the definition of transaction program as defined in page 346 of (Inoue and Sakama, 1999) only to base facts, and not to apply it to events facts. This rule states that for any atom A that does not appear in the head of any rule in P we must introduce the production rule $out(A) \rightarrow \varepsilon$ and if A is not abducible we must introduce also $in(A) \rightarrow false$.
3. To perform abductive reasoning on a derived event fact ιp (resp. δp), we have to *explain* ιp (resp. δp). On the other hand, to perform abductive reasoning on a negative derived event fact $\neg\iota$p (resp. $\neg\delta$p), we have to *unexplain* $\neg\iota$p (resp. $\neg\delta$p).

As a result of applying Inoue and Sakama's method according to the previous transformations we obtain several explanations $\prec E_i, F_i \succ$. For each such explanation $\prec E_i, F_i \succ$, E_i corresponds to the abductive explanation that satisfies the request while F_i contains base events that may not belong to E_i.

Note that the sets F_i play a similar role than the condition set in the events method. However, due to the restrictions imposed by Inoue and Sakama's method, each F_i contains only base event facts while the condition set contains more general goals.

Example 8 (adapted from Example 3.2 of Inoue and Sakama (1999)) illustrates the use of Inoue and Sakama's method to perform abductive reasoning on the event rules.

Example 8

Let $\prec$P, A$\succ$ be an ALP where:

P:	P $\leftarrow P \wedge \neg$A
	Q $\leftarrow \neg$C
	C $\leftarrow$
A :	A, C

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

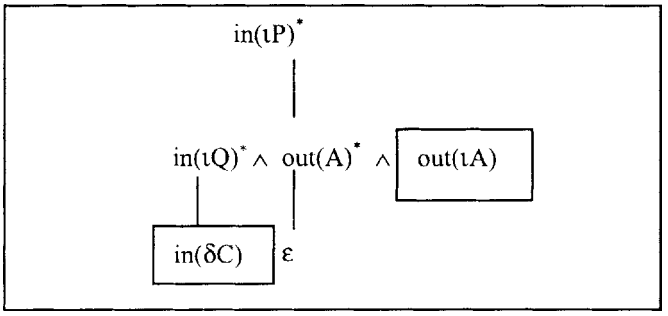

Fig. 5. Computing explanations for ← ιP.

The relevant rules of P* are the following:

P: $\iota P \leftarrow \iota Q \wedge \neg A \wedge \neg \iota A$

$\iota Q \leftarrow \delta C$

and the new abducibles atoms are δC, ιA, ιC, δA.

Then, the subset of τP^* obtained for the previous rules becomes:

$$in(\iota P) \rightarrow in(\iota Q) \wedge out(A) \wedge out(\iota A)$$
$$in(\iota Q) \rightarrow in(\delta C)$$

Now, to perform abductive reasoning on ιP we have to compute the explanations for ιP. Figure 5 shows the behaviour of Inoue and Sakama's method in this case. Note that, as a result, we have obtained the minimal explanations for ιP: $\prec E, F \succ = \prec \{in(\delta C)\}, \{out(\iota A)\} \succ$. That means that to insert P we should delete C.

From the previous example, we can conclude that Inoue and Sakama's method is not only applicable to the update problems mentioned by Inoue and Sakama (1998, 1999), mainly view updating and satisfiability checking, but also to the rest of schema validation and update processing problems in covered acyclic databases.

Moreover, by explicitly considering the event rules, this method is able to perform more precise requests. It is not difficult to see that in Inoue and Sakama (1999), an anti-explanation of P takes two different cases into account: whether P is false in the old state of the database and it is not inserted during the transition and whether P is deleted during the transition. By considering events about P we can be more specific since we can just look for anti-explanations of ιP (which corresponds only to the second case). A similar claim can be made when looking for explanations. In fact, this distinction would allow them also to be able to define and handle dynamic integrity constraints. For instance, the dynamic constraint "it may not be inserted Joan as an employee and deleted the Sales department at the same time" may be specified as: $\leftarrow \iota Emp(Joan) \wedge \delta Dept(Sales)$.

Finally, we must note that, in fact, Inoue and Sakama's method would not need to use the event rules to obtain the solutions of Example 8. However, we believe that the use of these rules could help this method to relax the restrictions it imposes on the programs it deals with. We will see in the next section other advantages provided by the event rules when more general conditions have to be taken into account.

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

6.2.3 The SLDNFA Method

Denecker and De Schreye (1998) propose SLDNFA, an extension of SLDNF resolution, to deal with abduction in abductive logic programs with negation. Given an abductive logic program and a query Q_o to be explained, an SLDNFA computation can be understood as a process of deriving formulas of the form $\forall (Q_o \leftarrow \Psi)$, where Ψ is obtained from the unsolved goals of the SLDNFA computation.

In fact, SLDNFA distinguishes two kinds of abductive solutions. A *ground abductive solution* for $\leftarrow Q_o$ is a triple (Σ', Δ, θ) with Δ a finite set of ground abducible atoms and θ a substitution of the variables of Q_o, both based on the alphabet Σ', such that $P + \Delta \models \forall(\theta\ (Q_o))$. Similarly, an *abductive solution* for $\leftarrow Q_o$ wrt P^A is an open formula Ψ containing only equality and abducible predicates such that $P^A \models \forall (Q_o \leftarrow \Psi)$ and $\exists(\Psi)$ is satisfiable wrt P^A. A ground abductive solution can be considered as a special case of an abductive solution since many ground abductive solutions can be drawn from Ψ, in general.

SLDNFA has been proved to be sound and complete for failure. Soundness ensures that the obtained solutions are correct since they entail the initial query and are consistent. Completeness for failure guarantees that if there exists a failed SLDNFA-tree for an initial query, then the query has no abductive solutions. However, this does not imply that SLDNFA generates all ground abductive solutions or all ground solutions satisfying some minimality criteria, as already pointed out by Denecker and De Schreye (1998, p. 138). Two variants of SLDNFA, namely $SLDNF^o$ and $SLDNFA_+$, are defined for which stronger completeness results are proved.

SLDNFA can also be used to perform abductive reasoning on the event rules. To do it, we should first rewrite event and transition rules to guarantee that they explicitly state the event definition and, thus, that the events are correctly handled by SLDNFA. That is, for each event literal $\iota Q(x)$ appearing in the body of an event or transition rule we should add the literal $\neg Q(x)$ to that rule, while for each event literal $\delta Q(x)$ we should add the literal $Q(x)$. This rewriting is equivalent to the addition of new rules in Inoue and Sakama's method to guarantee that an event can be successfully applied.

Then, we should take the Augmented Database (corresponding to the rewritten rules) as the abductive logic program P^A, where SLDNFA is applied and consider base event predicates as the only abducible predicates.

Denecker and De Schreye already mention the applicability of SLDNFA (and its variants) to other problems and, in particular, to satisfiability checking. From the use of SLDNFA to perform abductive reasoning on the event rules, we can conclude that SLDNFA is also applicable to the rest of schema validation and update processing problems. Furthermore, dynamic integrity constraints can be easily specified by means of event and transition rules and, thus, they could also be handled by SLDNFA.

Example 9 (a simplified version of the example in Denecker and De Schreye (1998, p. 119)) illustrates the use of SLDNFA to perform abductive reasoning on the event rules.

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

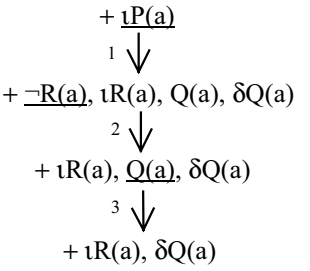

Fig. 6. SLDNFA-refutation for ← ιP(a).

Example 9

Consider the following deductive database:

$$Q(a)$$
$$P(x) \leftarrow R(x) \wedge \neg Q(x)$$

The abductive logic program P^A corresponding to the previous database is:

(*I*.1)	$\iota P(x) \leftarrow R(x) \wedge \neg \delta R(x) \wedge Q(x) \wedge \neg \delta Q(x)$
(*I*.2)	$\iota P(x) \leftarrow \neg R(x) \wedge \iota R(x) \wedge \neg Q(x) \wedge \neg \iota Q(x)$
(*I*.3)	$\iota P(x) \leftarrow \neg R(x) \wedge \iota R(x) \wedge Q(x) \wedge \delta Q(x)$
Abducibles:	$\{\iota R, \delta R, \iota Q, \delta Q\}$

An SLDNFA-refutation to find abductive solutions for ιP(a) (which corresponds to the initial query ← ιP(a)) is shown in Figure 6. The ground abductive answer generated by this refutation is $(\Sigma,\{\iota R(a), \delta Q(a)\},\varepsilon)$. This answer states that the insertion of P(a) may be achieved by the insertion of R(a) and the deletion of Q(a).

Use of the event rules also provides other contributions to SLDNFA. For instance, it allows SLDNFA to obtain solutions that would not be generated otherwise. For instance, SLDNFA (as defined in Denecker and De Schreye (1998)) would not obtain any solution to the request insert P(a) in Example 9. The reason is that SLDNFA considers only positive explanations, but no negative explanations (according to the terminology of Inoue and Sakama (1999)). Therefore, it cannot generate the deletion of Q(a) which is required to insert P(a). We have shown in Figure 6 that the use of the event rules allows SLDNFA to obtain such kind of solutions, since abducing events corresponds always to the generation of positive explanations although an event may correspond to a deletion of a database atom.

Another limitation that is overcome by the use of the event rules is that a direct application of SLDNFA to integrity constraint maintenance is not incremental. That is, it does not exploit the fact that the old database satisfies the integrity constraints, but rechecks blindly all of them (Denecker and De Schreye, 1998, p. 158). Again, such situation can also be overcome if SLDNFA is used in conjunction with the event rules, as shown in Example 10.

Example 10

Consider the following deductive database:

$$Q(b)\ R(b)\ Q(c)\ R(c)\ Q(d)\ R(d)$$
$$Ic \leftarrow Q(x) \wedge \neg R(x)$$

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

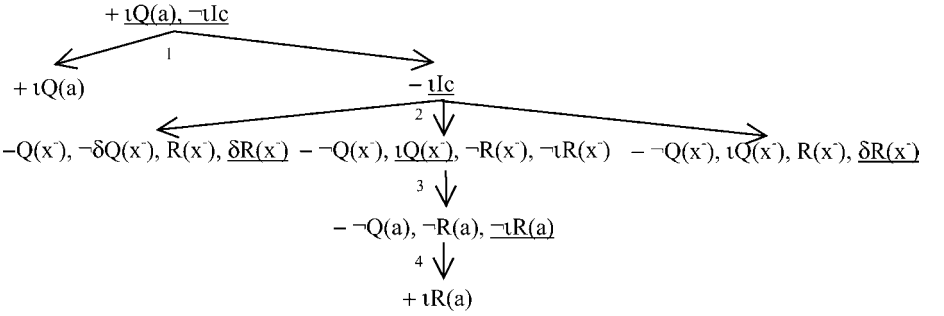

Fig. 7. SLDNFA-refutation for $\leftarrow \iota Q(a) \wedge \neg \iota Ic$.

The abductive logic program corresponding to the previous database is:

(*I*.1)	$\iota Ic \leftarrow Q(x) \wedge \neg \delta Q(x) \wedge R(x) \wedge \neg \delta R(x)$
(*I*.2)	$\iota Ic \leftarrow \neg Q(x) \wedge \iota Q(x) \wedge \neg R(x) \wedge \neg \iota R(x)$
(*I*.3)	$\iota Ic \leftarrow \neg Q(x) \wedge \iota Q(x) \wedge R(x) \wedge \delta R(x)$
Abducibles:	$\{\iota Q, \delta Q, \iota R, \delta R\}$

An SLDNFA-refutation to find abductive solutions for $\iota Q(a)$ that do not violate integrity constraints (which corresponds to the initial query $\leftarrow \iota Q(a) \wedge \neg \iota Ic$) is shown in Figure 7.

The ground abductive answer generated by this refutation is $(\Sigma, \{\iota Q(a), \iota R(a)\}, \varepsilon)$. Note that, in this case, integrity constraint maintenance has taken into account that integrity constraints are satisfied before the update, since it has only considered the events that could induce a violation of Ic (and not all database facts involved in the definition of Ic, as SLDNFA would do in the absence of the event rules).

6.2.4 How many methods do we really need to deal with database problems?

In general, research related to database problems is still looking for methods able to deal only with specific problems. However, we have shown that all problems are either of abductive or deductive nature and, thus, they can be formulated in terms of just two forms of reasoning. Therefore, we may conclude from our results that at most two different procedures are enough to handle all of them.

In fact, it could also be argued that just one single general procedure would be enough since it has been shown that either abduction as well as deduction can be realized in terms of the other. For instance, Bry (1990) and Bry *et al.* (1999) have proposed a procedure that allows to perform abductive reasoning by means of deduction, while Denecker and De Schreye (1998) and Inoue and Sakama (1999) have shown that their abductive procedures can also be used for deduction. Clearly, this strengthens our claim that we do not need a different method to deal with each schema validation and update processing problem.

The discussion about whether it would be better to have just one single method or two is far beyond the scope of this paper since it requires to be further investigated. From our intuition, we think that it will be difficult to define a single method which is as efficient to perform abductive reasoning as it is to perform deductive reasoning

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

since, as we have shown in the paper, the nature of the corresponding problems is intrinsically abductive or deductive. In this sense, we believe that it is difficult to incorporate the optimizations required to efficiently deal with deductive problems into a method based on abductive reasoning (and the other way around). However, this is still an open problem and it is a challenger research.

7 Using the event rules to perform general abductive reasoning

We have shown how the event rules can be used to deal with several schema validation and update processing problems and we have illustrated how several deductive and abductive procedures can be used to reason on them. In this section, we sketch how the event rules can be used also to solve general abductive problems in addition to the database problems considered before. Obviously, we must take into account that whenever we use the event rules we obtain solutions that minimize the difference between the old and the new sates and that this is not necessarily a requirement that the abductive explanations must satisfy in general.

We illustrate, by means of the following example, how could we use the event rules to deal with a typical abductive task like fault diagnosis and how the expected solutions to this problem can be abduced from these rules.

Example 11

The relevant part of the abductive framework of this example is the following:

$EDB \cup IDB^*$: Lamp(L1)
Battery(C1,B1)
Faulty-lamp $\leftarrow$ Lamp(x) $\wedge$ Broken(x)
Backup(x) $\leftarrow$ Battery(x,y) $\wedge \neg$Unloaded(y)
Faulty-lamp $\leftarrow$ Power-failure(x) $\wedge \neg$Backup(x)
Unloaded(x) $\leftarrow$ Dry-cell(x)
ιFaulty-lamp $\leftarrow$ Faulty-lamp^n $\wedge$ $\neg$Faulty-lamp
Faulty-lamp^n $\leftarrow$ ιPower-failure(x) $\wedge$ Backup(x)$\wedge \delta$Backup(x)
Faulty-lamp^n $\leftarrow$ ιPower-failure(x) $\wedge \neg$Backup(x) $\wedge \neg\iota$Backup(x)
Faulty-lamp^n $\leftarrow$ Lamp(x) $\wedge \neg\delta$Lamp(x) $\wedge \iota$Broken(x)
δBackup(x) $\leftarrow$ Battery(x,y) $\wedge \iota$Unloaded(y) $\wedge$ Backup^n(x)
ιUnloaded(x) $\leftarrow$ ιDry-cell(x)

Ab: $\{\iota$Broken, δBroken, ιPower-failure, δPower-failure, ιDry-cell, δDry-cell$\}$

Within this framework, we can use the event rules to detect that a faulty lamp problem is caused by a broken lamp or by a power failure of a circuit without backup, that is, a loaded battery. To do it, we have selected the Events Method among the three possible abductive procedures considered in section 6.2.

Figure 8 shows how, given the goal $\leftarrow \iota$Faulty-lamp and $EDB \cup IDB^*$, the Events Method obtains the abductive solution $T = \{\iota$Power-failure(C1), ιDry-cell(B1)$\}$, which is a possible solution for having a faulty lamp.

The Events Method would also obtain the solution $T = \{\iota$Broken(L1)$\}$ by considering the rule Faulty-lamp^n(x) $\leftarrow$ Lamp(x) $\wedge \delta$Lamp(x) $\wedge \iota$Broken(x) at step 2 of

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

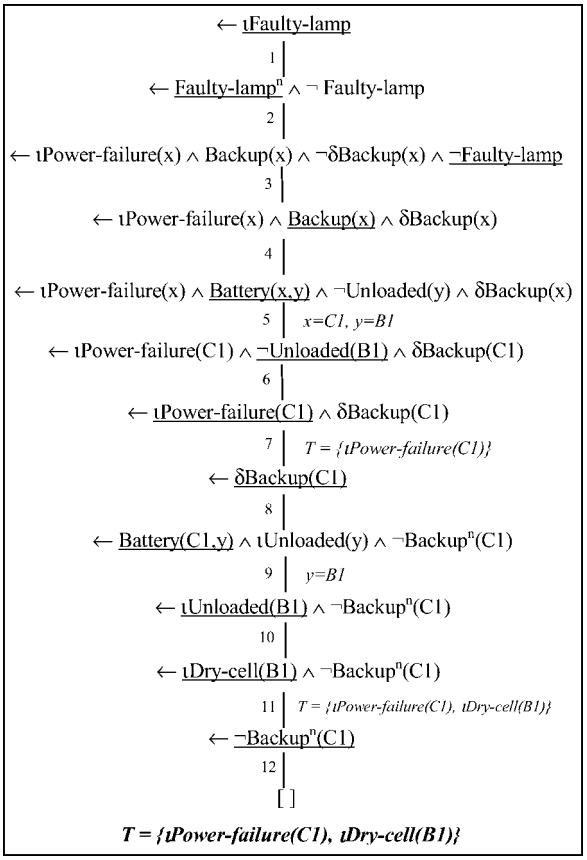

Fig. 8. Constructive derivation for ← ιFaulty-lamp.

Figure 8, and other possible solutions, like for instance T = {ιPower-failure(C2)}, by considering the rule Faulty-lamp^n(x) ← ιPower-failure(x) ∧ Backup(x) ∧¬ιBackup(x) also at step 2. The resulting derivations are not shown in the tree above.

8 Conclusions and further work

We have shown that database schema validation and update processing problems can be classified into problems of either deductive or abductive nature according to the reasoning paradigm that is more adequate to solve them. This has been done by making explicit the exact changes that occur in a transition between two consecutive states of the database by means of the event rules (Olivé, 1991) and by performing deductive and abductive reasoning on these rules. In this way, we have distinguished between *deductive problems*, concerned with computing the changes on derived predicates induced by a transaction, and *abductive problems*, concerned with determining the possible transactions that satisfy a set of changes on derived predicates. We have also shown that deductive and abductive problems can be combined, thus defining more complex update problems.

https://doi.org/10.1017/S1471068402001631 Published online by Cambridge University Press

Thus, problems like materialized view maintenance, integrity constraint checking or condition monitoring are considered as naturally deductive, while problems like view updating, integrity constraint maintenance or enforcing condition activation as naturally abductive.

By taking only a unique set of rules and two forms of reasoning into account to specify and deal with all these problems, we have shown that it is possible to provide general methods able to uniformly deal with several database updating problems at the same time. This suggests that future research in this field should be aimed at providing general methods instead of proposing specific methods for solving a particular problem like has been traditionally done in the past. Moreover, all these problems could be uniformly integrated into a database update processing system.

We have also shown how some existing general deductive and abductive procedures may be used to reason on the event rules. In this way, we have shown that these procedures can be used to deal with all database problems considered in this paper. This has been illustrated by means of examples and some additional benefits gained by these procedures when reasoning on the event rules have also been pointed out. Moreover, we have sketched how the event rules could be used to solve general abductive problems in addition to database schema validation and update processing problems.

The results presented in this paper may be extended at least in three different directions. First, our framework could be generalized to databases that allow recursive rules. Second, to deepen in the study of the application of the event rules to current abductive procedures to provide an efficient implementation of abductive reasoning on the event rules. Third, the advantages and inconveniences of using the event rules to perform general abductive reasoning should be further investigated.

Acknowledgements

This work has been partially supported by the CICYT PRONTIC program project TIC97-1157.

References

Abiteboul, S. (1988) Updates, a new frontier. In: Gyssens, M., Paredaens, J. and Gucht, D. V. (eds.), *ICDT'88, 2nd International Conference on Database Theory: Lecture Notes in Computer Science 326*, pp. 1–18. Bruges, Belgium. Springer-Verlag.

Abiteboul, S., Hull, R. and Vianu, V. (1995) *Foundations of Databases*. Addison-Wesley.

Adrion, W. R., Branstad, M. A. and Cherniavsky, J. C. (1982) Verification, validation, and testing of computer software. *ACM Computing Surveys*, **14**(2), 159–192.

Baralis, E., Ceri, S. and Paraboschi, S. (1998) Compile-time and runtime analysis of active behaviors. *Transactions of Knowledge and Data Engineering*, **10**(3), 353–370.

Bry, F. (1990) Intensional updates: Abduction via deduction. In: Warren, D. H. D. and Szeredi, P. (eds.), *Proceedings Seventh International Conference on Logic Programming*, pp. 561–575. Jerusalem, Israel. The MIT Press.

Bry, F. and Manthey, R. (1986) Checking consistency of database constraints: a logical basis. In: Chu, W. W., Gardarin, G., Ohsuga, S. and Kambayashi, Y. (eds.), *Twelfth International Conference on Very Large Data Bases*, pp. 13–20. Kyoto, Japan. Morgan Kaufmann.

Bry, F., Decker, H. and Manthey, R. (1988) A uniform approach to constraint satisfaction and constraint satisfiability in deductive databases. In: Schmidt, M. M. J. W. and Ceri, S. (eds.), *Proceedings International Conference on Extending Database Technology* (*EDBT '88*): *Lecture Notes in Computer Science 303*, pp. 488–505. Venice, Italy. Springer-Verlag.

Bry, F., Eisinger, N., Schütz, H. and Torge, S. (1998) Sic: Satisfiability checking for integrity constraints. In: Fraternali, P., Geske, U., Ruiza, C. and Seipel, D. (eds.), *6th International Workshop on Deductive Databases and Logic Programming* (*DDLP'98*), pp. 25–36.

Ceri, S., Fraternali, P., Paraboschi, S. and Tanca, L. (1994) Automatic generation of production rules for integrity maintenance. *ACM Transactions on Database Systems*, **19**(3), 367–422.

Console, L., Sapino, M. L. and Dupré, D. T. (1995) The role of abduction in database view updating. *Journal Intelligent Information Systems*, **4**(3), 261–280.

Decker, H. (1996) An extension of SLD by abduction and integrity maintenance for view updating in deductive databases. In: Maher, M. (ed.), *Proceedings Joint International Conference and Symposium on Logic Programming*, pp. 157–169. MIT Press.

Decker, H., Teniente, E. and Urpí, T. (1996) How to tackle schema validation by view updating, *Proceedings of the International Conference on Extending Database Technology* (*EDBT'96*): *Lecture Notes in Computer Science 1057*, pp. 535–549. Avignon, France. Springer-Verlag.

Denecker, M. and De Schreye, D. D. (1998) SLDNFA: An abductive procedure for abductive logic programs. *Journal of Logic Programming*, **34**(2), 111–167.

García, C., Celma, M., Mota, L. and Decker, H. (1994) Comparing and synthesizing integrity checking methods for deductive databases. In: Elmagarmid, A. K. and Neuhold, E. (eds.), *Proceedings 10th International Conference on Data Engineering*, pp. 214–222. IEEE Press.

Grant, J. and Minker, J. (1992) The impact of logic programming on databases. *Communications of the ACM*, **35**(3), 66–81.

Guessoum, A. and Lloyd, J. W. (1990) Updating knowledge bases. *New Generation Computing*, 71–89.

Gupta, A. and Mumick, I. S. (1995) Maintenance of materialized views: Problems, techniques and applications. *IEEE Quarterly Bulletin on Data Engineering*, Special Issue on Materialized Views and Data Warehousing, **18**(2), 3–18.

Hanson, E. N., Chaabouni, M., Kim, C. and Wang, Y. (1990) A predicate matching algorithm for database rule systems. In: Garcia-Molina, H. and Jagadish, H. V. (eds.), *Proceedings ACM SIGMOD International Conference on Management of Data*, Atlantic City, NJ. (*SIGMOD Record* (*ACM Special Interest Group on Management of Data*), **19**(2) 271–280.)

Inoue, K. and Sakama, C. (1998) Specifying transactions for extended abduction. In: Cohn, A. G., Schubert, L. and Shapiro, S. C. (eds.), *Proceedings 6th International Conference on Principles of Knowledge Representation and Reasoning* (*KR-98*), pp. 394–405. Morgan Kaufmann.

Inoue, K. and Sakama, C. (1999) Computing extended abduction through transaction programs. *Annals of Mathematics and Artificial Intelligence*, **25**(3–4), 339–367.

Inoue, K., Koshimura, M. and Hasegawa, R. (1992) Embedding negation as failure into a model generation theorem prover. In: Kapur, D. (ed.), *Proceedings 11th International Conference on Automated Deduction* (*CADE-11*): *Lecture Notes in Artificial Intelligence 607*, pp. 400–415. Saratoga Springs, NY. Springer-Verlag.

Kakas, A. C. and Mancarella, P. (1990) Database updates through abduction. In: McLeod, D., Sacks-Davis, R. and Schek, H.-J. (eds.), *16th International Conference on Very Large Data Bases*, pp. 650–661. Brisbane, Australia. Morgan Kaufmann.

Küchenhoff, V. (1991) On the efficient computation of the difference between consecutive database states. In: Delobel, C., Kifer, M. and Masunaga, Y. (eds.), *Proceedings Deductive and Object–Oriented Databases* (*DOOD'91*)*: Lecture Notes in Computer Science 566*, pp. 478–502. Berlin, Germany. Springer-Verlag.

Lee, S. Y. and Ling, T. W. (1996) Further improvements on integrity constraint checking for stratifiable deductive databases. In: Vijayaraman, T. M., Buchmann, A. P., Mohan, C. and Sarda, N. L. (eds.), *VLDB'96, Proceedings 22th International Conference on Very Large Data Bases*, pp. 495–505. Mumbai (Bombay), India. Morgan Kaufmann.

Levy, A. Y. and Sagiv, Y. (1995) Semantic query optimization in Datalog programs. *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 163–173. San Jose, CA.

Lloyd, J. W. (1987) *Foundations of Logic Programming, Second Edition*. Springer-Verlag.

Lloyd, J. W. and Topor, R. W. (1984) Making PROLOG more expressive. *Journal of Logic Programming*, **1**(3), 225–40.

Lobo, J. and Trajcevski, G. (1997) Minimal and consistent evolution in knowledge bases. *Journal of Applied Non-Classical Logics*, **7**(1–2), 117–146.

Mayol, E. and Teniente, E. (1999) A survey of current methods for integrity constraint maintenance and view updating. *First International Workshop on Evolution and Change in Data Management ECDM99*, pp. 62–73.

Moerkotte, G. and Lockemann, P. C. (1991) Reactive consistency control in deductive databases. *ACM Transactions on Database Systems*, **16**(4), 670–702.

Olivé, A. (1991) Integrity constraints checking in deductive databases. In: Lohman, G. M., Sernadas, A. and Camps, R. (eds.), *17th International Conference on Very Large Data Bases*, pp. 513–523. Barcelona, Catalonia. Morgan Kaufmann.

Qian, X. and Widerhold, G. (1991) Incremental recomputation of active relational expressions. *Transactions on Knowledge and Data Engineering*, **3**(3), 337–341.

Rosenthal, A., Chakravarthy, S., Blaustein, B. T. and Blakeley, J. A. (1989) Situation monitoring for active databases. In: Apers, M. G. and Wiederhold, G. (eds.), *Very Large Data Bases: Proceedings 15th International Conference on Very Large Data Bases*, pp. 455–464. Amsterdam, The Netherlands. Morgan Kaufmann.

Roussopoulos, N. (1998) Materialized views and data warehouses. *SIGMOD Record* (*ACM Special Interest Group on Management of Data*), **27**(1), 21–26.

Sadri, F. and Kowalski, R. A. (1988) A theorem proving approach to database integrity. In: Minker, J. (ed.), *Foundations of Deductive Databases and Logic Programming*, pp. 313–362. Morgan Kaufmann.

Schewe, K. D. and Thalheim, B. (1994) Achieving consistency in active databases. *Proceedings 4th International Workshop on Research Issues in Data Engineering – Active Database Systems*, pp. 71–76.

Staudt, M. and Jarke, M. (1996) Incremental maintenance of externally materialized views. In: Vijayaraman, T. M., Buchmann, A. P., Mohan, C. and Sarda, N. L. (eds.), *Proceedings 22nd International Conference on Very Large Data Bases*, Mumbai (Bombay), pp. 75–86. India. Morgan Kaufmann.

Teniente, E. (2000) Deductive databases. In: Piattini, M. and Díaz, O. (eds.), *Advanced Database Technology and Design*, pp. 91–136. Artech House.

Teniente, E. and Olivé, A. (1995) Updating knowledge bases while maintaining their consistency. *The VLDB Journal*, **4**(2), 193–241.

Teniente, E. and Urpí, T. (1995) A common framework for classifying and specifying deductive database updating problems. In: Yu, P. S. and Chen, A. L. P. (eds.), *Proceedings 11th International Conference on Data Engineering*, pp. 173–183. IEEE Press.

Ullman, J. D. (1989) *Principles of Database and Knowledge-base Systems*. Computer Science Press.

Urpí, T. and Olivé, A. (1992) A method for change computation in deductive databases. In: Yuan, L.-Y. (ed.), *18th International Conference on Very Large Data Bases*, pp. 225–237. Vancouver, Canada. Morgan Kaufmann.

Urpí, T. and Olivé, A. (1994) Semantic change computation optimization in active databases. *Proceedings 4th International Workshop on Research Issues in Data Engineering – Active Database Systems*, pp. 19–27.

Widom, J. and Ceri, S. (1996) *Active Database Systems: Triggers and Rules For Advanced Database Processing*. Morgan Kaufmann.

Wüthrich, B. (1993) On updates and inconsistency repairing in knowledge bases. *International Conference on Data Engineering*, pp. 608–615. IEEE Press.