

A Transformation-Based Approach to View Updating in Stratifiable Deductive Databases

Andreas Behrend, Rainer Manthey

University of Bonn, Institute of Computer Science III
Roemerstr. 164, D-53117 Bonn, Germany
{behrend,manthey}@cs.uni-bonn.de

Abstract In this paper we present a new rule-based approach for consistency preserving view updating in deductive databases. Based on rule transformations performed during schema design, fixpoint evaluations of these rules at run time compute consistent realizations of view update requests. Alternative realizations are expressed using disjunctive Datalog internally. The approach extends and integrates standard techniques for efficient query answering and integrity checking (based on transformation techniques and fixpoint computation, too). Views may be stratifiably recursive. The set-orientedness of the approach makes it potentially useful in the context of (commercial) SQL systems, too.

1 Introduction

In deductive databases, views are usually updated indirectly, via changes of the underlying base relations. If update requests for derived relations (view updates) are accepted at all, they cannot be directly executed. Instead, possible realizations of the view update (i.e. base relation updates inducing the requested change of the view) have to be determined and one of them chosen for execution, if alternatives exist. Realizations are (at least) expected to be consistency preserving. In "real life" SQL databases, view updating (VU) is heavily restricted in order to guarantee existence of a unique realization. In the context of Datalog and logic programming, however, the approach followed by most researchers is to systematically analyze all "reasonable" realizations and to leave the final choice to the user who has issued the request. We systematically compute alternative consistent realizations, too, restricting computation to those realizations which are (in a particular sense) "small enough" for being meaningful.

Our approach is transformation-based, i.e., we compute a set of VU rules from the rules of the given deductive database. VU requests as well as their realizations are represented by VU facts, either explicitly introduced (the request to be satisfied) or derived via the VU rules (intermediate requests or realizations). VU rules are expressed in disjunctive Datalog, disjunctive facts representing choices between alternative realizations. Computation of VU facts is performed by means of fixpoint computation (which can be viewed as a set-oriented version of model generation). We rely on fixpoint computation as the "engine" for view

updating as this will be the method of choice once the limited, but quite powerful forms of recursion defined in the SQL standard will at last be implemented by the commercial DBMS vendors. Furthermore, this choice enables us to employ, e.g., the Magic Sets approach (transformation-based, too) for optimizing evaluation of recursive queries and rules.

A related standard mode of inference driven by updates is update propagation (UP), i.e. the incremental computation of all consequences of a given base data change on derived data (induced updates). Whereas VU is inherently a top-down problem (from the request down to the realizations), UP is bottom-up (from the base updates up to the induced updates). There are various transformation- and fixpoint-based approaches to UP around, e.g. [3,6,11,13,17,19,22]. UP has been introduced for checking integrity over views as well as for adapting materialized views efficiently. In a VU context, UP is ideal for controlling integrity of the proposed realizations, too, and may in addition be used for computing any kind of further side effect caused by the chosen realization, as such side effects will certainly correspond to updates induced by the respective realization. We make use of "Magic Updates", our own variant of rule-based UP [6], which makes use of Magic Sets for ensuring goal-directedness of UP.

After an initial VU phase determining a first set of candidate realizations, and a subsequent first UP phase eliminating inconsistent branches of the resulting tree of alternatives, we try to identify cases where integrity violations caused by the resp. realization can be repaired by generating additional VU requests. Similar extensions to the initial VU request are used for compensating side effects sometimes contradicting the effects of the chosen realization steps, e.g. by introducing new derivations for a fact intended to disappear). The resulting overall process thus alternates between clearly distinguishable phases of "local" fixpoint computations using either VU rules or UP rules only. Phases are linked by additional transition rules triggering UP after a VU subprocess has stopped, or triggering further VU inferences (for compensation and repair) once UP has terminated. Termination is guaranteed unless no finite consistent realization exists.

An important advantage of our approach is that it uses other transformation- and fixpoint-based inference modes, present in a set-oriented (e.g. an SQL) context anyway (Magic Sets for efficient query evaluation, Magic Updates for efficient integrity checking and materialized view maintenance) rather than introducing a completely new style of computation, which would be hard to integrate with existing view-related DBMS services. From a theoretical perspective, covering all aspects of VU in a coherent formal manner (transformed internal rules as specifications, fixpoint operators as interpreters) appears a promising basis for investigating general problems and properties of view updating such as, e.g., termination, completeness, minimality and the like.

There are various well-known approaches to VU based on disjunctive fixpoints/model generation, e.g. [7,10,12]. Our approach is probably closest to that of Bry [7], combining the given application rules with an application-independent set of meta rules expressing the logical relation between consecutive database

states. Our (application-specific) VU rules can be seen as optimized versions of partial interpretations of Bry’s meta rules over the respective application rules, UP rules are incremental (and thus more efficient) specifications of differences between old and new state. Furthermore, set-oriented fixpoint computation replaces Bry’s instance-oriented model generator.

Another closely related rule-based approach to both, VU and UP, is the internal events method [18,22,23]. In this approach, the same transformed rules are applied for both, VU and UP purposes. SLDNF resolution is used for evaluating rules rather than fixpoint computation, thus limiting the approach in presence of recursion. In addition, the resolution engine has to maintain internal, temporary data structures for memorizing control information needed for finding realizations, thus leading to a rather ”heavy” computation engine not suitable in a set-oriented (SQL) context. This is the ”price to pay” for the formally elegant re-use of internal events rules (originally introduced for UP only) for finding realizations of VU requests. We prefer to keep the underlying runtime engine simple (and uniform) rather than the transformation process (applied only once during schema design). There are many other approaches to VU exhibiting similarities and differences to our approach if analyzed more deeply - such a discussion is beyond the scope of this paper.

In Section 2, we briefly summarize necessary basic notions and notations for stratifiable deductive databases and their extension to disjunctive databases. In Section 3 we recall our approach to update propagation ([6,11,17]), as UP is needed as a subprocess in our VU method and because the style of rule transformation used in the UP case serves as a model for the newly introduced VU rules. In Section 4, we first present these new transformations for computing the VU analysis rules, then we address the role of UP in view updating. Afterwards, we introduce the fixpoint algorithm ”driving” both, VU and UP rules according to an alternating control strategy. In Section 5, we briefly discuss pros and cons of the new approach.

2 Basic concepts

In this section, we recall basic concepts for stratifiable definite and indefinite (disjunctive) databases [5,16]. The fixpoint semantics of indefinite databases induces a model generation procedure that can handle our transformed VU rules in which alternative view update realizations are represented in form of disjunctions.

2.1 Deductive Databases

A Datalog *rule* is a function-free clause of the form $H_1 \leftarrow L_1 \wedge \dots \wedge L_m$ with $m \geq 1$ where H_1 is an atom denoting the rule’s head, and L_1, \dots, L_m are literals, i.e. positive or negative atoms, representing its body. We assume all deductive rules to be *safe*, i.e., all variables occurring in the head or in any negated literal of a rule must be also present in a positive literal in its body. If $A \equiv p(t_1, \dots, t_n)$

with $n \geq 0$ is a literal, we use $\text{vars}(A)$ to denote the set of variables occurring in A and $\text{pred}(A)$ to refer to the predicate symbol p of A . If A is the head of a given rule R , we use $\text{pred}(R)$ to refer to the predicate symbol of A . For a set of rules \mathcal{R} , $\text{pred}(\mathcal{R})$ is defined as $\cup_{r \in \mathcal{R}} \{\text{pred}(r)\}$. A *fact* is a ground atom in which every t_i is a constant.

A *deductive database* \mathcal{D} is a triple $\langle \mathcal{F}, \mathcal{R}, \mathcal{I} \rangle$ where \mathcal{F} is a finite set of facts (called *base facts*), \mathcal{I} is a finite set of integrity constraints and \mathcal{R} a finite set of rules such that $\text{pred}(\mathcal{F}) \cap \text{pred}(\mathcal{R}) = \emptyset$ and $\text{pred}(\mathcal{I}) \subseteq \text{pred}(\mathcal{F} \cup \mathcal{R})$. Within a deductive database \mathcal{D} , a predicate symbol p is called *derived* (view predicate), if $p \in \text{pred}(\mathcal{R})$. The predicate p is called *extensional* (or *base predicate*), if $p \in \text{pred}(\mathcal{F})$. The *state* of a database \mathcal{D} is defined as the set of all facts that can be derived by the rules \mathcal{R} including the facts in \mathcal{F} . An *integrity constraint* is represented by a ground atom which is required to be derivable in every state of the database. For the sake of simplicity of exposition, and without loss of generality, we assume that a predicate is either base or derived, but not both, which can be easily achieved by rewriting a given database. Additionally, we solely consider *stratifiable rules* [19] which do not allow recursion through negative predicate occurrences. Given a stratifiable deductive database $\mathcal{D} = \langle \mathcal{F}, \mathcal{R}, \mathcal{I} \rangle$, its semantics is defined by the perfect model $PM_{\mathcal{D}}$ [19] of $\mathcal{F} \cup \mathcal{R}$ which satisfies all integrity constraints \mathcal{I} , i.e., $\mathcal{I} \subseteq PM_{\mathcal{D}}$. Otherwise, the semantics of \mathcal{D} is undefined.

For determining the semantics of a stratifiable database, the iterated fixpoint computation can be used which constructs bottom-up a sequence of least Herbrand models according to a given stratification of the rules. As an example, consider the following consistent deductive database $\mathcal{D} = \langle \mathcal{F}, \mathcal{R}, \mathcal{I} \rangle$:

<u>\mathcal{R}</u>	<u>\mathcal{I}</u>	<u>\mathcal{F}</u>
$h(X, Y) \leftarrow p(X, Y) \wedge \neg e(X, Y)$	$ic_1 \leftarrow p(X, Y)$	$e(1, 2)$
$p(X, Y) \leftarrow e(X, Y)$	$ic_2 \leftarrow \neg aux$	$e(1, 4)$
$p(X, Y) \leftarrow e(X, Z) \wedge p(Z, Y)$	$aux \leftarrow p(X, X)$	$e(2, 3)$

Relation p is the transitive closure of e . Relation h is the set of derivable paths within p . Constraint ic_1 ensures that in every consistent database state at least one p -tuple exists. Constraint ic_2 is used to prevent cycles in p . Note that it is necessary to determine relation p before checking ic_2 due to the negative dependency. The last model computed by iterated fixpoint computation is represented by the set $\mathcal{F} \cup \{p(1, 2), p(1, 4), p(2, 3), p(1, 3)\} \cup \{ic_1, ic_2\}$ and coincides with the positive portion of the perfect model $PM_{\mathcal{D}}$ of \mathcal{D} .

2.2 Normalized Definite Rules

In this paper, we will recall a transformation-based approach to update propagation and introduce a related transformation-based approach to view updating, both leading to the generation of rather complex propagation rules if applied to an arbitrary set of stratifiable definite rules. In order to reduce syntactic complexity, our transformations will be expressed for *normalized rules*. Except for

the occurrence of 0-ary predicates, each normalized rule directly corresponds to one operation in relational algebra. Note that every given set of deductive rules can be systematically transformed into an equivalent set of normalized ones by unfolding [8]. A normalized definition of relation p from Section 2.1 is

$$\begin{array}{ll} p(X, Y) \leftarrow e(X, Y) & \text{aux}_1(X, Y) \leftarrow \text{aux}_2(X, Y, Z) \\ p(X, Y) \leftarrow \text{aux}_1(X, Y) & \text{aux}_2(X, Y, Z) \leftarrow e(X, Z) \wedge p(Z, Y) \end{array}$$

where the recursive rule for p is replaced by three new rules to separate the implicit union, projection and join operator, respectively.

2.3 Disjunctive Deductive Databases

A disjunctive Datalog rule is a function-free clause of the form $H_1 \vee \dots \vee H_m \leftarrow L_1 \wedge \dots \wedge L_n$ with $m, n \geq 1$ where the rule's head $H_1 \vee \dots \vee H_m$ is a disjunction of positive atoms, and the rule's body $L_1 \wedge \dots \wedge L_n$ consists of literals, i.e. positive or negative atoms. If $H \equiv H_1 \vee \dots \vee H_m$ is the head of a given rule R , we use $\text{pred}(R)$ to refer to the set of predicate symbols of H , i.e. $\text{pred}(R) = \{\text{pred}(H_1), \dots, \text{pred}(H_m)\}$. For a set of rules \mathcal{R} , $\text{pred}(\mathcal{R})$ is defined again as $\cup_{r \in \mathcal{R}} \text{pred}(r)$. A *disjunctive fact* $f \equiv f_1 \vee \dots \vee f_k$ is a disjunction of ground atoms f_i with $i \geq 1$. f is called *definite* if $i = 1$. In the following, we identify a disjunctive fact with a set of atoms such that the occurrence of a ground atom A within a fact f can also be written as $A \in f$. The set difference operator can then be used to exclude certain atoms from a disjunction while the empty set is interpreted as the boolean constant *false*.

A *disjunctive deductive database* \mathcal{DD} is a pair $\langle \mathcal{F}, \mathcal{R} \rangle$ where \mathcal{F} is a finite set of disjunctive facts and \mathcal{R} a finite set of disjunctive rules such that $\text{pred}(\mathcal{F}) \cap \text{pred}(\mathcal{R}) = \emptyset$. Again, stratifiable rules are considered only, that is, recursion through negative predicate occurrences is not permitted. In addition to the usual stratification concept for definite rules it is required that all predicates within a rule's head are assigned to the same stratum. Note that integrity constraints are omitted this time because our approach transforms a set of definite rules together with the given integrity constraints into a single set of disjunctive rules. This integration allows for providing a uniform approach to handling constraint violations as well as detecting erroneous derivation paths.

In contrast to a stratifiable definite database, there is no unique perfect model for a stratifiable disjunctive database. Instead, Minker et al. have shown that there is generally a set of perfect models which captures the semantics of a disjunctive database [14]. For their computation, an extended model generation approach based on a DNF representation of conclusions has been proposed in [21]. For the purpose of computing alternative VU realizations, however, the corresponding perfect model state based on CNF representation seems to be more appropriate. This is especially the case if an application scenario deals with substantially more definite than indefinite facts, this more compact representation allows for reducing the total number of derivations [5]. In following the positive portion of a perfect model state for a disjunctive database \mathcal{DD} is

denoted as $\mathcal{MS}_{\mathcal{DD}}$. As an example, consider the following stratifiable disjunctive database $\mathcal{DD} = \langle \mathcal{F}, \mathcal{R} \rangle$:

$$\begin{array}{ll}
 \underline{\mathcal{R}} & \underline{\mathcal{F}} \\
 s(\mathbf{X}) \vee t(\mathbf{X}) \leftarrow r(\mathbf{X}) \wedge \neg p(\mathbf{X}) & q(2) \\
 p(\mathbf{X}) \leftarrow b(\mathbf{X}, \mathbf{Y}) \wedge p(\mathbf{Y}) & b(1, 2) \\
 p(\mathbf{X}) \leftarrow q(\mathbf{X}) & r(\mathbf{a}) \vee r(\mathbf{b}) \vee r(\mathbf{c}) \vee r(1)
 \end{array}$$

Similar to the stratification concept in definite databases, it is necessary to postpone the determination of relations s and t because of the negative dependency to p . $\mathcal{MS}_{\mathcal{DD}}$ is then given by the set $\mathcal{F} \cup \{p(1), p(2)\} \cup \{s(a) \vee t(a) \vee r(b) \vee r(c) \vee r(1), s(b) \vee t(b) \vee r(a) \vee r(c) \vee r(1), s(c) \vee t(c) \vee r(a) \vee r(b) \vee r(1)\}$. Note that a model generation approach based on DNF would have to deal with 26 minimal models in order to compute the set of perfect models of \mathcal{DD} . A constructive fixpoint-based method for computing the perfect model state has been proposed by the authors in [5].

3 Update Propagation

Determining the consequences of base relation changes is essential for maintaining materialized views as well as for efficiently checking integrity. However, as in most cases an update will affect only a small portion of the database, it is rarely reasonable to compute the induced changes by comparing the entire old and new database states. Instead, update propagation methods have been proposed aiming at the efficient computation of implicit changes of derived relations resulting from explicitly performed updates of extensional facts [3,6,11,13,17,19,22]. In the context of deductive databases transformation-based approaches to UP have been studied which allow for an incremental computation of induced changes and utilize a uniform fixpoint computation mechanism. Bearing in mind the manifold benefits of these well-established approaches, it seems worthwhile to develop a similar, i.e., incremental and transformation-based, approach to the dual problem of view updating. Before doing so, we need to recall basic principles of UP in deductive databases.

In the following a specific method for UP is considered in more detail which fits well to the semantics of deductive databases introduced above. We will use the notion *update* to denote the 'true' changes caused by a transaction only; that is, we solely consider sets of updates where compensation effects (i.e., given by an insertion and deletion of the same fact or the insertion of facts which already existed, for example) have already been taken into account [11].

Definition 1. Let $\mathcal{D} = \langle \mathcal{F}, \mathcal{R}, \mathcal{I} \rangle$ be a stratifiable deductive database. An update $u_{\mathcal{D}}$ is a pair $\langle u_{\mathcal{D}}^+, u_{\mathcal{D}}^- \rangle$ where $u_{\mathcal{D}}^+$ and $u_{\mathcal{D}}^-$ are sets of base facts with $\text{pred}(u_{\mathcal{D}}^+ \cup u_{\mathcal{D}}^-) \subseteq \text{pred}(\mathcal{F})$, $u_{\mathcal{D}}^+ \cap u_{\mathcal{D}}^- = \emptyset$, $u_{\mathcal{D}}^+ \not\subseteq \mathcal{F}$ and $u_{\mathcal{D}}^- \subseteq \mathcal{F}$. The atoms $u_{\mathcal{D}}^+$ represent insertions into \mathcal{D} , whereas $u_{\mathcal{D}}^-$ contains the facts to be deleted from \mathcal{D} .

We consider true updates only, i.e., ground atoms which are presently not derivable for atoms to be inserted, or are derivable for atoms to be deleted, respectively [11]. The notion *induced update* is used to refer to the entire set of facts in which the new state of the database (including derived facts) differs from the former after an update of base tables has been applied.

Definition 2. Let \mathcal{D} be a stratifiable database, $PM_{\mathcal{D}}$ the semantics of \mathcal{D} and $u_{\mathcal{D}}$ an update. Then $u_{\mathcal{D}}$ leads to an induced update $u_{D \rightarrow D'}$ from D to the updated database D' which is a pair $\langle u_{D \rightarrow D'}^+, u_{D \rightarrow D'}^- \rangle$ of sets of ground atoms such that $u_{D \rightarrow D'}^+ = PM_{D'} \setminus PM_D$ and $u_{D \rightarrow D'}^- = PM_D \setminus PM_{D'}$. The atoms $u_{D \rightarrow D'}^+$ represent the induced insertions, whereas $u_{D \rightarrow D'}^-$ consists of the induced deletions.

The task of update propagation is to systematically compute the set of all induced modifications in $u_{D \rightarrow D'}$ starting from the physical changes of base data. Technically, this is a set of delta facts for any affected relation which may be stored in corresponding delta relations. For each predicate symbol $p \in \text{pred}(\mathcal{D})$, we will use a pair of delta relations $\langle \Delta_p^+, \Delta_p^- \rangle$ representing the insertions and deletions induced on p by an update $u_{\mathcal{D}}$. The initial set of delta facts directly results from the given update u_D and represents the so-called *UP seeds*.

Definition 3. Let \mathcal{D} be a stratifiable deductive database and $u_D = \langle u_D^+, u_D^- \rangle$ a base update. The set of UP seeds $\text{prop_seeds}(u_D)$ with respect to u_D is defined as follows:

$$\text{prop_seeds}(u_D) := \{ \Delta_p^\pi(c_1, \dots, c_n) \mid p(c_1, \dots, c_n) \in u_D^\pi \text{ and } \pi \in \{+, -\} \}.$$

In the following we will recall a transformation-based approach to UP which uses the stratifiable rules given in a database schema and the UP seeds to derive *deductive propagation rules* for computing delta relations [6,11]. A propagation rule refers to at least one delta relation in its body in order to provide a focus on the underlying changes when computing induced updates. For showing the effectiveness of an induced update, however, references to the state of a relation before and after the base update has been performed are necessary. We call these states the old and the new state, respectively. The state relations are never completely computed but are queried with bindings from the delta relation in the propagation rule body and thus act as a test of effectiveness. For evaluating rule bodies, fixpoint computation combined with a Magic Sets transformation is used, as proposed by the authors in [6].

Definition 4 (Update Propagation Rules). Let \mathcal{R} be a normalized stratifiable deductive rule set. The set of UP rules for true updates with respect to \mathcal{R} is denoted by \mathcal{R}^Δ and is defined as the smallest set satisfying the following conditions:

1. For each rule of the form $p(\vec{x}) \leftarrow q(\vec{y}) \wedge r(\vec{z}) \in \mathcal{R}$ with $\text{vars}(p(\vec{x})) = (\text{vars}(q(\vec{y})) \cup \text{vars}(r(\vec{z})))$ four UP rules of the form

$$\begin{array}{ll} \Delta_p^+(\vec{x}) \leftarrow \Delta_q^+(\vec{y}) \wedge r^{\text{new}}(\vec{z}) & \Delta_p^-(\vec{x}) \leftarrow \Delta_q^-(\vec{y}) \wedge r(\vec{z}) \\ \Delta_p^+(\vec{x}) \leftarrow \Delta_r^+(\vec{z}) \wedge q^{\text{new}}(\vec{y}) & \Delta_p^-(\vec{x}) \leftarrow \Delta_r^-(\vec{z}) \wedge q(\vec{y}) \end{array}$$

are in \mathcal{R}^Δ . For $\vec{y} = \vec{z}$ these rules correspond to an intersection, and for $\vec{y} \neq \vec{z}$ to a join in relational algebra (RA).

2. For each rule of the form $p(\vec{x}) \leftarrow q(\vec{x}) \wedge \neg r(\vec{x}) \in \mathcal{R}$ four UP rules of the form

$$\begin{array}{ll} \Delta_p^+(\vec{x}) \leftarrow \Delta_q^+(\vec{x}) \wedge \neg r^{new}(\vec{x}) & \Delta_p^-(\vec{x}) \leftarrow \Delta_q^-(\vec{x}) \wedge \neg r(\vec{x}) \\ \Delta_p^+(\vec{x}) \leftarrow \Delta_r^-(\vec{x}) \wedge q^{new}(\vec{x}) & \Delta_p^-(\vec{x}) \leftarrow \Delta_r^+(\vec{x}) \wedge q(\vec{x}) \end{array}$$

are in \mathcal{R}^Δ . This kind of rules corresponds to the difference operator in RA.

3. For each two rules of the form $p(\vec{x}) \leftarrow q(\vec{x})$ and $p(\vec{x}) \leftarrow r(\vec{x})$ four UP rules of the form

$$\begin{array}{ll} \Delta_p^+(\vec{x}) \leftarrow \Delta_q^+(\vec{x}) \wedge \neg p(\vec{x}) & \Delta_p^-(\vec{x}) \leftarrow \Delta_q^-(\vec{x}) \wedge \neg p^{new}(\vec{x}) \\ \Delta_p^+(\vec{x}) \leftarrow \Delta_r^+(\vec{x}) \wedge \neg p(\vec{x}) & \Delta_p^-(\vec{x}) \leftarrow \Delta_r^-(\vec{x}) \wedge \neg p^{new}(\vec{x}) \end{array}$$

are in \mathcal{R}^Δ . This kind of rules corresponds to the union operator in RA. Note that the additional references to p and p^{new} are necessary for excluding alternative derivations.

4. a) For each relation p defined by a single rule $p(\vec{x}) \leftarrow q(\vec{y}) \in \mathcal{R}$ with $\text{vars}(p(\vec{x})) = \text{vars}(q(\vec{y}))$ two UP rules of the form

$$\Delta_p^+(\vec{x}) \leftarrow \Delta_q^+(\vec{y}) \qquad \Delta_p^-(\vec{x}) \leftarrow \Delta_q^-(\vec{y})$$

are in \mathcal{R}^Δ . These rules correspond to the special case where projections turns into permutation of variables.

- b) For each relation p defined by a single rule $p \leftarrow \neg q \in \mathcal{R}$ two UP rules of the form

$$\Delta_p^+ \leftarrow \Delta_q^- \qquad \Delta_p^- \leftarrow \Delta_q^+$$

are in \mathcal{R}^Δ . These rules cover the special case of negated 0-ary predicates.

5. For each relation p defined by a single projection rule $p(\vec{x}) \leftarrow q(\vec{y}) \in \mathcal{R}$ with $\text{vars}(p(\vec{x})) \supset \text{vars}(q(\vec{y}))$ two UP rules of the form

$$\Delta_p^+(\vec{x}) \leftarrow \Delta_q^+(\vec{y}) \wedge \neg p(\vec{x}) \qquad \Delta_p^-(\vec{x}) \leftarrow \Delta_q^-(\vec{y}) \wedge \neg p^{new}(\vec{x})$$

are in \mathcal{R}^Δ . Again, additional references to p and p^{new} are necessary for detecting alternative derivations.

As the selection operator of relational algebra is not fully expressible in Datalog, a corresponding transformation rule is missing in the above definition. However, Datalog can be easily extended by built-in terms which would make it relationally complete. In this paper we will not consider Datalog extensions for simplicity reasons although corresponding view update rules can be easily defined.

The UP rules defined above reference both, the old and new database state. We assume the old state to be present, thus only references to the new database are indicated by the adornment *new* while unadorned predicate symbols refer to the old state. For simulating the new database state from a given update and the old state, so called *transition rules* [11,18] are used:

Definition 5 (Update Propagation Transition Rules). *Let \mathcal{R} be a stratifiable deductive rule set. The set of UP transition rules for new state simulation with respect to \mathcal{R} denoted \mathcal{R}_τ^Δ is defined as the smallest set satisfying the following conditions:*

1. *For each n -ary extensional predicate symbol $p \in \mathbf{pred}(\mathcal{F})$, the direct transition rules*

$$\begin{aligned} p^{new}(x_1, \dots, x_n) &\leftarrow p(x_1, \dots, x_n) \wedge \neg \Delta_p^-(x_1, \dots, x_n) \\ p^{new}(x_1, \dots, x_n) &\leftarrow \Delta_p^+(x_1, \dots, x_n) \end{aligned}$$

are in \mathcal{R}_τ^Δ where the x_i are distinct variables.

2. *For each rule $H \leftarrow L_1 \wedge \dots \wedge L_n \in \mathcal{R}$, an indirect transition rule of the form*

$$\mathbf{new}(H) \leftarrow \mathbf{new}(L_1) \wedge \dots \wedge \mathbf{new}(L_n)$$

is in \mathcal{R}_τ^Δ where the mapping \mathbf{new} for a literal $A \equiv r(t_1, \dots, t_n)$ is defined as $\mathbf{new}(A) = r^{new}(t_1, \dots, t_n)$ and $\mathbf{new}(\neg A) = \neg \mathbf{new}(A)$.

Note that all adorned predicates are assumed to introduce new names not found in the database yet, e.g., $\forall p \in \mathbf{pred}(\mathcal{R} \cup \mathcal{F}) : \{p^{new}, \Delta_p^+\} \cap \mathbf{pred}(\mathcal{R} \cup \mathcal{F}) = \emptyset$. Neither the references to the new nor to the old database state have to be completely evaluated. Instead they are queried (e.g. by using Magic Sets) with respect to the bindings provided by the delta relation within the body of a propagation rule and thus, play the role of test predicates. As an example of this propagation approach, consider again the normalized rules for relation p from the example in Section 2.1. The UP rules \mathcal{R}^Δ with respect to insertions into e are as follows:

$$\begin{aligned} \Delta_p^+(X, Y) &\leftarrow \Delta_e^+(X, Y) \wedge \neg p(X, Y) & \Delta_{aux_2}^+(X, Y, Z) &\leftarrow \Delta_e^+(X, Z) \wedge p^{new}(Z, Y) \\ \Delta_p^+(X, Y) &\leftarrow \Delta_{aux_1}^+(X, Y) \wedge \neg p(X, Y) & \Delta_{aux_2}^+(X, Y, Z) &\leftarrow \Delta_p^+(Z, Y) \wedge e^{new}(X, Z) \\ & & \Delta_{aux_1}^+(X, Y) &\leftarrow \Delta_{aux_2}^+(X, Y, Z) \wedge \neg aux_1(X, Y). \end{aligned}$$

The transition rules \mathcal{R}_τ^Δ with respect to p are given by:

$$\begin{aligned} e^{new}(X, Y) &\leftarrow e(X, Y) \wedge \neg \Delta_e^-(X, Y) & p^{new}(X, Y) &\leftarrow e^{new}(X, Y) \\ e^{new}(X, Y) &\leftarrow \Delta_e^+(X, Y) & p^{new}(X, Y) &\leftarrow aux_1^{new}(X, Z) \\ & & aux_1^{new}(X, Y) &\leftarrow aux_2^{new}(X, Y, Z) \\ & & aux_2^{new}(X, Y, Z) &\leftarrow e^{new}(X, Z) \wedge p^{new}(Z, Y) \end{aligned}$$

Note that these rules can be determined at schema definition time and don't have to be recompiled each time a new transaction is applied. It is obvious that if \mathcal{R} is stratifiable, the rule set $\mathcal{R} \cup \mathcal{R}^\Delta \cup \mathcal{R}_\tau^\Delta$ will be stratifiable, too [11]. The following proposition states the correctness of this approach:

Proposition 1. Let $\mathcal{D} = \langle \mathcal{F}, \mathcal{R}, \mathcal{I} \rangle$ be a stratifiable database, $u_{\mathcal{D}}$ an update and $u_{\mathcal{D} \rightarrow \mathcal{D}'} = \langle u_{\mathcal{D} \rightarrow \mathcal{D}'}^+, u_{\mathcal{D} \rightarrow \mathcal{D}'}^- \rangle$ the corresponding induced update from \mathcal{D} to \mathcal{D}' . Let $\mathcal{D}^\Delta = \langle \mathcal{F} \cup \text{prop_seeds}(u_{\mathcal{D}}), \mathcal{R} \cup \mathcal{R}^\Delta \cup \mathcal{R}_\tau^\Delta \rangle$ be the transformed deductive database of \mathcal{D} . Then the delta relations defined by the UP rules \mathcal{R}^Δ correctly represent the induced update $u_{\mathcal{D} \rightarrow \mathcal{D}'}$. Hence, for each relation $p \in \text{pred}(\mathcal{D})$ the following conditions hold:

$$\begin{aligned} \Delta_p^+(\vec{t}) \in PM_{\mathcal{D}^\Delta} &\iff p(\vec{t}) \in u_{\mathcal{D} \rightarrow \mathcal{D}'}^+ \\ \Delta_p^-(\vec{t}) \in PM_{\mathcal{D}^\Delta} &\iff p(\vec{t}) \in u_{\mathcal{D} \rightarrow \mathcal{D}'}^- . \end{aligned}$$

Proof. cf. [11, p. 161-163].

4 View Updating

In contrast to update propagation, view updating aims at determining one or more base relation updates such that all given update requests with respect to derived relations are satisfied after the base updates have been successfully applied. In the following, we develop a transformation-based approach to incrementally compute such base updates. After introducing some basic definitions, in Sections 4.1 and 4.2 two rule forms are presented playing a similar role as the UP rules and UP transition rules from Section 3. As the application of these disjunctive rules is followed by an integrity check the overall process incorporates the UP rules from Section 3 and is described in Section 4.3.

Definition 6. Let $\mathcal{D} = \langle \mathcal{F}, \mathcal{R}, \mathcal{I} \rangle$ be a stratifiable deductive database. A VU request $\nu_{\mathcal{D}}$ is a pair $\langle \nu_{\mathcal{D}}^+, \nu_{\mathcal{D}}^- \rangle$ where $\nu_{\mathcal{D}}^+$ and $\nu_{\mathcal{D}}^-$ are sets of ground atoms representing the facts to be inserted into \mathcal{D} or deleted from \mathcal{D} , resp., such that $\text{pred}(\nu_{\mathcal{D}}^+ \cup \nu_{\mathcal{D}}^-) \subseteq \text{pred}(\mathcal{R})$, $\nu_{\mathcal{D}}^+ \cap \nu_{\mathcal{D}}^- = \emptyset$, $\nu_{\mathcal{D}}^+ \cap PM_{\mathcal{D}} = \emptyset$ and $\nu_{\mathcal{D}}^- \subseteq PM_{\mathcal{D}}$.

Note that we consider again true view updates only, i.e., ground atoms which are presently not derivable for atoms to be inserted, or are derivable for atoms to be deleted, respectively. A method for view updating determines sets of alternative updates satisfying a given request. A set of updates leaving the given database consistent after its execution is called *VU realization*.

Definition 7. Let $\mathcal{D} = \langle \mathcal{F}, \mathcal{R}, \mathcal{I} \rangle$ be a stratifiable deductive database and $\nu_{\mathcal{D}}$ a VU request. A VU realization is a base update $u_{\mathcal{D}}$ which leads to an induced update $u_{\mathcal{D} \rightarrow \mathcal{D}'}$ from \mathcal{D} to \mathcal{D}' such that $\nu_{\mathcal{D}}^+ \subseteq PM_{\mathcal{D}'}$ and $\nu_{\mathcal{D}}^- \cap PM_{\mathcal{D}'} = \emptyset$.

There may be infinitely many realizations and even realizations of infinite size which satisfy a given VU request. In our approach, a breadth-first search (BFS) is employed for determining a set of minimal realizations $\tau_{\mathcal{D}} = \{u_{\mathcal{D}}^1, \dots, u_{\mathcal{D}}^i\}$. Any $u_{\mathcal{D}}^i$ is minimal in the sense that none of its updates can be removed without losing the property of being a realization for $\nu_{\mathcal{D}}$. As each level of the search tree is completely explored, the result usually consists of more than one realization. If only VU realizations of infinite size exist, our method will not terminate.

4.1 Top-Down Computation of Realizations

Given a VU request $\nu_{\mathcal{D}}$, view updating methods usually determine further VU requests in order to find relevant base updates. Similar to delta relations for UP we will use the notion *VU relation* to access individual view updates with respect to the relations of our system. For each relation $p \in \mathbf{pred}(\mathcal{R} \cup \mathcal{F})$ we use the VU relation $\nabla_p^+(\vec{x})$ for tuples to be inserted into \mathcal{D} and $\nabla_p^-(\vec{x})$ for tuples to be deleted from \mathcal{D} . The initial set of delta facts resulting from a given VU request is again represented by so-called *VU seeds* similar to the UP seeds.

Definition 8. *Let \mathcal{D} be a deductive database and $\nu_D = \langle \nu_D^+, \nu_D^- \rangle$ a VU request. The set of VU seeds $\mathbf{vu_seeds}(\nu_D)$ with respect to ν_D is defined as follows:*

$$\mathbf{vu_seeds}(\nu_D) := \{ \nabla_p^\pi(c_1, \dots, c_n) \mid p(c_1, \dots, c_n) \in \nu_D^\pi \text{ and } \pi \in \{+, -\} \}.$$

Starting from the seeds, view updating methods as well as UP methods analyze the deductive rules of \mathcal{D} in order to find subsequent VU requests systematically. For finding the direct consequences of a given view update request, we employ so-called VU rules. These rules are used to perform a top-down analysis in a similar way as the bottom-up analysis implemented by the UP rules from Section 3. In order to enhance readability and to syntactically distinguish VU rules from UP rules, the notation $Head \leftarrow Body$ is changed to $Body \rightarrow Head$.

Definition 9 (View Update Rules). *Let \mathcal{R} be a normalized stratifiable deductive rule set. The set of VU rules for true view updates is denoted \mathcal{R}^∇ and is defined as the smallest set satisfying the following conditions:*

1. *For each rule of the form $p(\vec{x}) \leftarrow q(\vec{y}) \wedge r(\vec{z}) \in \mathcal{R}$ with $\mathbf{vars}(p(\vec{x})) = (\mathbf{vars}(q(\vec{y})) \cup \mathbf{vars}(r(\vec{z})))$ the following three VU rules are in \mathcal{R}^∇ :*

$$\begin{aligned} \nabla_p^+(\vec{x}) \wedge \neg q(\vec{y}) &\rightarrow \nabla_q^+(\vec{y}) & \nabla_p^-(\vec{x}) &\rightarrow \nabla_q^-(\vec{y}) \vee \nabla_r^-(\vec{z}) \\ \nabla_p^+(\vec{x}) \wedge \neg r(\vec{z}) &\rightarrow \nabla_r^+(\vec{z}) \end{aligned}$$

2. *For each rule of the form $p(\vec{x}) \leftarrow q(\vec{x}) \wedge \neg r(\vec{x}) \in \mathcal{R}$ the following three VU rules are in \mathcal{R}^∇ :*

$$\begin{aligned} \nabla_p^+(\vec{x}) \wedge \neg q(\vec{x}) &\rightarrow \nabla_q^+(\vec{x}) & \nabla_p^-(\vec{x}) &\rightarrow \nabla_q^-(\vec{x}) \vee \nabla_r^+(\vec{x}) \\ \nabla_p^+(\vec{x}) \wedge r(\vec{x}) &\rightarrow \nabla_r^-(\vec{x}) \end{aligned}$$

3. *For each two rules of the form $p(\vec{x}) \leftarrow q(\vec{x})$ and $p(\vec{x}) \leftarrow r(\vec{x})$ the following three VU rules are in \mathcal{R}^∇ :*

$$\begin{aligned} \nabla_p^-(\vec{x}) \wedge q(\vec{x}) &\rightarrow \nabla_q^-(\vec{x}) & \nabla_p^+(\vec{x}) &\rightarrow \nabla_q^+(\vec{x}) \vee \nabla_r^+(\vec{x}) \\ \nabla_p^-(\vec{x}) \wedge r(\vec{x}) &\rightarrow \nabla_r^-(\vec{x}) \end{aligned}$$

4. a) *For each relation p defined by a single rule $p(\vec{x}) \leftarrow q(\vec{y}) \in \mathcal{R}$ with $\mathbf{vars}(p(\vec{x})) = \mathbf{vars}(q(\vec{y}))$ the following two VU rules are in \mathcal{R}^∇ :*

$$\nabla_p^+(\vec{x}) \rightarrow \nabla_q^+(\vec{y}) \qquad \nabla_p^-(\vec{x}) \rightarrow \nabla_q^-(\vec{y})$$

b) For each relation p defined by a single rule $p \leftarrow \neg q \in \mathcal{R}$ the following two VU rules are in \mathcal{R}^∇ :

$$\nabla_p^+ \rightarrow \nabla_q^- \qquad \nabla_p^- \rightarrow \nabla_q^+$$

5. Assume without loss of generality that each projection rule in \mathcal{R} is of the form $p(\vec{x}) \leftarrow q(\vec{x}, Y) \in \mathcal{R}$ with $Y \notin \text{vars}(p(\vec{x}))$. Then the following two VU rules

$$\begin{aligned} \nabla_p^-(\vec{x}) \wedge q(\vec{x}, Y) &\rightarrow \nabla_q^-(\vec{x}, Y) \\ \nabla_p^+(\vec{x}) &\rightarrow \nabla_q^+(\vec{x}, c_1) \vee \dots \vee \nabla_q^+(\vec{x}, c_n) \vee \nabla_q^+(\vec{x}, c^{new}) \end{aligned}$$

are in \mathcal{R}^∇ where all c_i are constants from the Herbrand universe $\mathcal{U}_{\mathcal{D}}$ of \mathcal{D} and c^{new} is a new constant, i.e., $c^{new} \notin \mathcal{U}_{\mathcal{D}}$.

In contrast to the UP rules from Definition 4, no explicit references to the new database state are included in the above VU rules. The reason is that these rules are applied iteratively over several intermediate database states before the minimal set of realizations has been found. Hence, the apparent references to the old state really refer to the current state which is continuously modified while computing VU realizations. The need for a multi-level compensation of side effects, however, will be discussed in more detail in Section 4.2. Nevertheless, these predicates solely act as tests again queried with respect to bindings from VU relations and thus will never be completely evaluated. The following theorem states that for every realization for a given VU request a corresponding solution path will be started by the application of $\mathcal{R} \cup \mathcal{R}^\nabla$. That is, for all realizations at least one base update is found implying the completeness of our VU rules.

Theorem 1. *Let $\mathcal{D} = \langle \mathcal{F}, \mathcal{R}, \mathcal{I} \rangle$ be a stratifiable database, $\nu_{\mathcal{D}}$ a view update request and $\tau_{\mathcal{D}} = \{u_{\mathcal{D}}^1, \dots, u_{\mathcal{D}}^n\}$ the corresponding set of minimal realizations. Let $\mathcal{D}^\nabla = \langle \mathcal{F} \cup \text{vu_seeds}(\nu_{\mathcal{D}}), \mathcal{R} \cup \mathcal{R}^\nabla \rangle$ be the transformed deductive database of \mathcal{D} . Then the VU relations in $\mathcal{MS}_{\mathcal{D}^\nabla}$ with respect to base relations of \mathcal{D} correctly represent all direct consequences of $\nu_{\mathcal{D}}$. That is, for each realization $u_{\mathcal{D}}^i = \langle u_{\mathcal{D}}^{i+}, u_{\mathcal{D}}^{i-} \rangle \in \tau_{\mathcal{D}}$ the following condition holds:*

$$\exists p(\vec{t}) \in u_{\mathcal{D}}^{i+} : \nabla_p^+(\vec{t}) \in \mathcal{MS}_{\mathcal{D}^\nabla} \vee \exists p(\vec{t}) \in u_{\mathcal{D}}^{i-} : \nabla_p^-(\vec{t}) \in \mathcal{MS}_{\mathcal{D}^\nabla}.$$

Proof. (Sketch) Every realization $u_{\mathcal{D}}^i \in \tau_{\mathcal{D}}$ leads to an induced update $u_{\mathcal{D} \rightarrow \mathcal{D}^i}^+$ from \mathcal{D} to \mathcal{D}^i which is correctly represented by delta relations defined by UP rules \mathcal{R}^Δ according to Proposition 1. One can show that there is at least one derivation path from $u_{\mathcal{D}}^i$ to $\nu_{\mathcal{D}}$ using the rules $\mathcal{R} \cup \mathcal{R}^\Delta$ which is conversely present in $\mathcal{R} \cup \mathcal{R}^\nabla$, too. From this, the condition above directly follows. \square

Note that only VU rules resulting from projection rules may introduce new constants in order to find a realization. These constants are accessible subsequently when further view updates are to be determined. Recursion combined with projection may lead to an infinite solution path by introducing an infinite number of new constants. However, during the top-down analysis phase in which \mathcal{R}^∇ is applied to the facts of \mathcal{D} only one constant is introduced for all projection

rules. Further constants are introduced in subsequent iteration rounds for repairing undesired side effects. For example, the VU rules above do not restrict sequences of ∇ -relations to those leading to a consistent database state only. For repairing violated constraints corresponding new update requests are generated and a re-application of \mathcal{R}^∇ is necessary. The handling of erroneous 'solution' paths, however, will be discussed in more detail in Section 4.2.

Although only one new constant is introduced for projection rules per iteration round, the considered number of constants within the corresponding disjunctive facts is still impractically high. However, it is usually not necessary to consider all constants from \mathcal{U}_D as proposed in Definition 9. For example, only those constants which are also provided by the codomain of q have to be considered. In fact, the set of constants can be even further reduced without losing the completeness of our approach. Additionally, the handling of the existentially quantified new constants c^{new} can be optimized as, e.g., proposed in [2]. We will refrain from a discussion of implementation issues, however, as the focus of this paper is to discover general properties of view updating methods rather than efficiency considerations.

We illustrate via an example the basic mechanism behind the application of \mathcal{R}^∇ . Consider the following deductive database $\mathcal{D} = \langle \mathcal{F}, \mathcal{R}, \mathcal{I} \rangle$ with $\mathcal{F} = \{r_2(2), s(2)\}$, $\mathcal{I} = \{ic(2)\}$ and the normalized deductive rules \mathcal{R} :

$$\begin{array}{ll} p(X) \leftarrow q_1(X) & q_1(X) \leftarrow r_1(X) \wedge s(X) \\ p(X) \leftarrow q_2(X) & q_2(X) \leftarrow r_2(X) \wedge \neg s(X) \\ ic(2) \leftarrow \neg au(2) & au(X) \leftarrow q_2(X) \wedge \neg q_1(X) \end{array}$$

The corresponding set of VU rules with respect to the VU request $\nu_D^+ = \{p(2)\}$ is then given by:

$$\begin{array}{ll} \nabla_p^+(X) \rightarrow \nabla_{q_1}^+(X) \vee \nabla_{q_2}^+(X) & \nabla_{q_2}^+(X) \wedge \neg r_2(X) \rightarrow \nabla_{r_2}^+(X) \\ \nabla_{q_1}^+(X) \wedge \neg r_1(X) \rightarrow \nabla_{r_1}^+(X) & \nabla_{q_2}^+(X) \wedge s(X) \rightarrow \nabla_s^-(X) \\ \nabla_{q_1}^+(X) \wedge \neg s(X) \rightarrow \nabla_s^+(X) & \end{array}$$

Applying these rules using disjunctive fixpoint computation leads to two alternative updates $u_{\mathcal{D} \rightarrow \mathcal{D}_1}^+ = \{r_1(2)\}$ and $u_{\mathcal{D} \rightarrow \mathcal{D}_2}^- = \{s(2)\}$ induced by the derived disjunction $\nabla_{r_1}^+(2) \vee \nabla_s^-(2)$. Obviously, the second update represented by $\nabla_s^-(2)$ would lead to an undesired side effect. In order to provide a complete method, however, such erroneous/incomplete paths must be also explored and side effects repaired if possible.

4.2 Bottom-Up Assessment of Realizations

Determining whether a computed update will lead to a consistent database state or not can be done by applying a bottom-up UP process at the end of the top-down phase leading to an irreparable constraint violation with respect to $\nabla_s^-(2)$:

$$\nabla_s^-(2) \Rightarrow \Delta_{q_2}^+(2) \Rightarrow \Delta_p^+(2), \Delta_{au}^+(2) \Rightarrow \Delta_{ic}^-(2) \rightsquigarrow false$$

In order to see whether the violated constraint can be repaired, the subsequent view update request $\nu_{\mathcal{D}^2}^+ = \{ic(2)\}$ with respect to \mathcal{D}^2 ought to be answered. The application of \mathcal{R}^∇ yields

$$\begin{aligned} & \Rightarrow \nabla_{q_2}^-(2), \nabla_{q_2}^+(2) \rightsquigarrow false \\ \nabla_{ic}^+(2) \Rightarrow \nabla_{aux}^-(2) \Downarrow & \\ & \Rightarrow \nabla_{q_1}^+(2) \Rightarrow \nabla_s^+(2), \nabla_s^-(2) \rightsquigarrow false \end{aligned}$$

showing that this request cannot be satisfied as inconsistent subsequent view update requests are generated on this path. In the following, erroneous derivation paths will be indicated by the keyword *false*. To eliminate those paths, we will employ the operation **reduce** which extracts the occurrences of *false* from given disjunctions. The reduced set of updates - each of them leading to a consistent database state only - represents the set of realizations $\tau_{\mathcal{D}} = \{\langle u_{\mathcal{D} \rightarrow \mathcal{D}^1}^+, \emptyset \rangle\}$ with $u_{\mathcal{D} \rightarrow \mathcal{D}^1}^+ = \{r_1(2)\}$.

An induced deletion of an integrity constraint predicate can be seen as a side effect of an 'erroneous' VU. Similar side effects, however, can be also found when induced changes to the database caused by a VU request may include derived facts which had been actually used for deriving this view update. This effect is shown in the following example for a non-normalized deductive database $\mathcal{D} = \langle \mathcal{R}, \mathcal{F}, \mathcal{I} \rangle$ with $\mathcal{R} = \{h(X) \leftarrow p(X) \wedge q(X) \wedge i, i \leftarrow p(X) \wedge \neg q(X)\}$, $\mathcal{F} = \{p(1)\}$, and $\mathcal{I} = \emptyset$. Given the VU request $\nu_{\mathcal{D}}^+ = \{h(1)\}$, the overall evaluation scheme for determining the only realization $\tau_{\mathcal{D}} = \{\langle u_{\mathcal{D} \rightarrow \mathcal{D}^1}^+, \emptyset \rangle\}$ with $u_{\mathcal{D} \rightarrow \mathcal{D}^1}^+ = \{q(1), p(c^{new_1})\}$ would be as follows:

$$\begin{aligned} & \Rightarrow \nabla_p^+(c^{new_1}) \\ \nabla_h^+(1) \Rightarrow \nabla_q^+(1) \Rightarrow \Delta_q^+(1) \Rightarrow \Delta_i^- \Rightarrow \nabla_i^+ \Downarrow & \\ & \Rightarrow \nabla_q^-(1), \nabla_q^+(1) \rightsquigarrow false \end{aligned}$$

The example shows the necessity of compensating side effects, i.e., the compensation of the 'deletion' Δ_i^- (that prevents the 'insertion' $\Delta_h^+(1)$) caused by the tuple $\nabla_q^+(1)$. In general the compensation of side effects, however, may in turn cause additional side effects which have to be 'repaired'. Thus, the view updating method must alternate between top-down and bottom-up phases until all possibilities for compensating side effects (including integrity constraint violations) have been considered, or a solution has been found. Therefore, we introduce so-called *VU transition rules* which are used to restart the VU analysis:

Definition 10 (View Update Transition Rules). *Let $\mathcal{D} = \langle \mathcal{F}, \mathcal{R}, \mathcal{I} \rangle$ be a stratifiable database, \mathcal{R}^∇ the set of corresponding VU rules, $Rel_{\mathcal{D}}$ the set of all predicate symbols occurring in $\langle \mathcal{F}, \mathcal{R}, \mathcal{I} \rangle$. The set of VU transition rules is defined as the smallest rule set satisfying the following conditions:*

1. For each n -ary predicate symbol $p \in Rel_{\mathcal{D}}$ the following rule is in \mathcal{R}_τ^∇

$$\nabla_p^+(x_1, \dots, x_n) \wedge \nabla_p^-(x_1, \dots, x_n) \rightarrow false$$

where x_i ($i = 1, \dots, n$) are distinct variables. This kind of rule is employed for detecting and deleting erroneous derivation paths.

2. For each ground literal $ic(\vec{c}) \in \mathcal{I}$ the following rule is in \mathcal{R}_τ^∇ :

$$\Delta_{ic}^-(\vec{c}) \rightarrow \nabla_{ic}^+(\vec{c})$$

These rules are used to repair violated integrity constraints.

3. For each rule $\nabla_p^\pi(\vec{x}) \wedge q(\vec{y}) \rightarrow \nabla_q^\nabla(\vec{y}) \in \mathcal{R}^\nabla$ with $\pi \in \{+, -\}$ the following rule is in \mathcal{R}_τ^∇

$$\nabla_p^\pi(\vec{x}) \wedge \neg q(\vec{y}) \wedge \Delta_q^+(\vec{y}) \rightarrow \nabla_q^-(\vec{y})$$

whereas for each rule $\nabla_p^\pi(\vec{x}) \wedge \neg q(\vec{y}) \rightarrow \nabla_q^+(\vec{y}) \in \mathcal{R}^\nabla$ a rule of the form

$$\nabla_p^\pi(\vec{x}) \wedge q(\vec{y}) \wedge \Delta_q^-(\vec{y}) \rightarrow \nabla_q^+(\vec{y})$$

is in \mathcal{R}_τ^∇ . These rules identify side effects and are employed for initiating a compensation attempt.

4. For each n -ary predicate symbol $p \in \mathcal{F}$ the following rule is in \mathcal{R}_τ^∇

$$\begin{aligned} \nabla_p^+(x_1, \dots, x_n) \wedge \neg p(x_1, \dots, x_n) &\rightarrow \Delta_p^+(x_1, \dots, x_n) \\ \nabla_p^-(x_1, \dots, x_n) \wedge p(x_1, \dots, x_n) &\rightarrow \Delta_p^-(x_1, \dots, x_n) \end{aligned}$$

where x_i ($i = 1, \dots, n$) are distinct variables. These rules initiate the bottom-up consequence analysis after the top-down analysis phase has been finished. Note that an effectiveness test is needed as the accumulated VU request will not represent true updates anymore after the first iteration round [11].

The rules in \mathcal{R}_τ^∇ make sure that erroneous solutions are evaluated to *false* and side effects are repaired. The following theorem establishes the correctness of the view update rules \mathcal{R}^∇ and \mathcal{R}_τ^∇ .

Theorem 2. Let $\mathcal{D} = \langle \mathcal{F}, \mathcal{R}, \mathcal{I} \rangle$ be a stratifiable database, $\nu_{\mathcal{D}}$ a view update request and $\tau_{\mathcal{D}} = \{u_{\mathcal{D}}^1, \dots, u_{\mathcal{D}}^n\}$ the corresponding set of minimal realizations. Let $\mathcal{D}^\nabla = \langle \mathcal{F} \cup \mathbf{vu_seeds}(\nu_{\mathcal{D}}), \mathcal{R} \cup \mathcal{R}^\nabla \rangle$ be the transformed deductive database of \mathcal{D} for determining the direct consequences of $\nu_{\mathcal{D}}$ and $\mathcal{D}^{\nabla\Delta} = \langle \mathcal{MS}_{\mathcal{D}^\nabla}, \mathcal{R} \cup \mathcal{R}^\Delta \cup \mathcal{R}_\tau^\Delta \cup \mathcal{R}_\tau^\nabla \rangle$ be the database for determining compensating VU requests. Then the VU facts ∇_{new}^π in $\mathcal{MS}_{\mathcal{D}^\nabla\Delta}$ which are not contained in $\mathcal{MS}_{\mathcal{D}^\nabla}$ correctly represent all indirect consequences of $\nu_{\mathcal{D}}$. That is, for every realization $u_{\mathcal{D}}^i \in \tau_{\mathcal{D}}$ which has not been completely determined by $\mathcal{MS}_{\mathcal{D}^\nabla}$ exists a fact f in ∇_{new}^π with $\pi \in \{+, -\}$ such that $u_{\mathcal{D}}^i$ is also a realization for $\mathbf{vu_seeds}(\nu_{\mathcal{D}}) \cup f$.

Proof. (Sketch) Since $u_{\mathcal{D}}^i$ is a realization which has not yet completely determined by $\mathcal{MS}_{\mathcal{D}^\nabla}$, there must be side effects that inhibit the derivation of one or more delta facts in $\mathbf{vu_seeds}(\nu_{\mathcal{D}})$ or lead to the violation of an integrity constraint. Compensation paths for violated constraints are correctly started by the corresponding rules in \mathcal{R}_τ^∇ . The other side effects must be handled by introducing new VU request which lead to the generation of at least one of the missing base updates in $u_{\mathcal{D}}^i$. From Theorem 1 it follows that the rules \mathcal{R}^∇ are correct and complete. Thus, the missing VU request result from VU rules where the

Algorithm 1 BFS determination of view update realizations

Input: normalized stratifiable deductive database $\mathcal{D} = \langle \mathcal{F}, \mathcal{R}, \mathcal{I} \rangle$,
transformed rule sets $\mathcal{R}^\nabla, \mathcal{R}_\tau^\nabla, \mathcal{R}^\Delta, \mathcal{R}_\tau^\Delta$ w.r.t. \mathcal{R} and \mathcal{I} ,
view update request $\nu_{\mathcal{D}} = \langle \nu_{\mathcal{D}}^+, \nu_{\mathcal{D}}^- \rangle$
Output: set of realizations $\tau_{\mathcal{D}} = \{u_{\mathcal{D} \rightarrow \mathcal{D}^1}, \dots, u_{\mathcal{D} \rightarrow \mathcal{D}^n}\}$ for $\nu_{\mathcal{D}}$

```
 $i$  := 0;  
 $F_0^\nabla$  := vu_seeds( $\nu_{\mathcal{D}}$ );  
 $F_0$  :=  $\mathcal{F}$ ;  
 $\tau_{\mathcal{D}}$  :=  $\emptyset$ ;  
repeat  
   $i$  :=  $i + 1$ ;  
   $MS_{D_i^\nabla}$  :=  $\mathcal{MS}_{\langle F_{i-1} \cup F_{i-1}^\nabla, \mathcal{R} \cup \mathcal{R}^\nabla \rangle_n}$ ;  
   $MS_{D_i^\nabla \Delta}$  :=  $\mathcal{MS}_{\langle MS_{D_i^\nabla}, \mathcal{R} \cup \mathcal{R}^\Delta \cup \mathcal{R}_\tau^\Delta \cup \mathcal{R}_\tau^\nabla \rangle_n}$ ;  
   $MS_i$  := reduce( $MS_{D_i^\nabla \Delta}$ );  
   $F_i^\nabla$  := get_ $\nabla$ ( $MS_i$ );  
   $F_i^\Delta$  := get_ $\Delta$ ( $MS_i$ );  
   $F_i$  := update( $\mathcal{F}, F_i^\Delta$ );  
   $\tau_{\mathcal{D}}$  := get_realizations( $MS_i, \nu_{\mathcal{D}}$ );  
until ( $\tau_{\mathcal{D}} \neq \emptyset$ )  $\vee$  ( $false \in MS_{D_i^\nabla \Delta} \vee (F_{i-1}^\nabla = F_i^\nabla)$ )  
return  $\tau_{\mathcal{D}}$ ;
```

body VU literal was satisfied but not the side literal with reference to the current database state. As for every rule in \mathcal{R}^∇ a corresponding rule is present in \mathcal{R}_τ^∇ which react to the changed database state, all possible new VU request are determined. Correctness follows from the correctness of \mathcal{R}^∇ and $\mathcal{R} \cup \mathcal{R}^\Delta \cup \mathcal{R}_\tau^\Delta$. Completeness follows from the fact, that all possible new VU request from \mathcal{R}^∇ are considered in \mathcal{R}_τ^∇ , too. \square

Having the rules for the direct and indirect consequences of a given VU request, we can now define a general application scheme for systematically determining VU realizations.

4.3 Overall Organization of View Updating

The top-down analysis rules $\mathcal{R} \cup \mathcal{R}^\nabla$ and the bottom-up consequence analysis rules $\mathcal{R} \cup \mathcal{R}^\Delta \cup \mathcal{R}_\tau^\Delta \cup \mathcal{R}_\tau^\nabla$ are iteratively applied using the general evaluation scheme depicted in Algorithm 1. Note that these disjunctive rules are stratifiable such that the computation of the perfect model state can be applied. The algorithm continuously extends the sets of alternative base updates by new ones until a realization for the given VU request has been found.

During the initialization phase, the operator **vu_seeds** is applied to the given VU request $\nu_{\mathcal{D}}$ for determining the initial ∇ -seed facts similar to the propagation seeds from Definition 3. In the following iteration phase the VU rules and UP rules are consecutively employed and their iterated fixpoint model state $MS_{D_i^\nabla}$

respectively $MS_{D_i^{\nabla\Delta}}$ is determined. The **reduce** operation eliminates all occurrences of *false* from disjunctive facts in $MS_{D_i^{\nabla\Delta}}$ in order to discard erroneous derivation paths. If the fact *false* has been derived, no solution for $\nu_{\mathcal{D}}$ could be found and the algorithm terminates. Operation **get_** ∇ accumulates previous as well as new compensating VU requests. If no new compensation requests are generated by \mathcal{R}_r^{∇} , no more alternative solution paths must be explored and the iteration stops. Operation **get_** Δ accumulates all determined base relation updates over all iteration round. These updates are then applied to \mathcal{F} such that the new compensation requests are processed with respect to the new (possibly disjunctive) database state. Based on the minimal models $\mathcal{D}^1, \dots, \mathcal{D}^n$ of MS_i satisfying the VU request $\nu_{\mathcal{D}}$, the operator **get_realizations** constructs the updates $u_{\mathcal{D} \rightarrow \mathcal{D}^j}$ which cause the transition from \mathcal{D} to \mathcal{D}^j . The correctness of Algorithm 1 directly follows from Theorem 1 and Theorem 2.

5 Discussion

In the previous chapter we introduced a new method for consistency preserving view updating. In our systematic approach, the top-down analysis phase and the bottom-up consequence analysis are separated showing the duality of update propagation and view updating analysis. Our method is suited for being implemented in a database context because of the transformation-based approach and the close relationship of normalized rules to relational algebra operators. In order to deal with alternative realizations, however, an extension to disjunctive databases becomes necessary. Further implementation details remain undiscussed, too, even though various possibilities for enhancing efficiency exist. For example, the employed rules can be further improved by incorporating transformation-based query optimization methods such as Magic Sets [4] or by a delayed evaluation of existentially quantified variables in projections [2].

Our method illustrates further interesting properties relevant for view updating in general. It is well-known that the view update problem: "Does there exist a VU realization for \mathcal{D} satisfying $\nu_{\mathcal{D}}$?" is undecidable given an arbitrary stratifiable deductive database and an arbitrary VU request. This can be shown, e.g., by reducing the query containment problem for positive deductive rules to a VU problem with respect to a stratifiable database. Since the query containment problem is undecidable for positive Datalog with at least one derived relation having more than one attribute, the VU problem must be undecidable, too [9,15]. Note that undecidability is not caused by the combination of deductive rules and integrity constraints but rather by the combination of rules with a view update which corresponds to an existential condition. The main reason for undecidability of the VU problem is that there are cases where only infinite fact bases are able to satisfy a given VU request.

The existence of infinite solutions, of course, is not a proof of undecidability of the VU problem but already indicates a fundamental problem of every view updating method. The infinite number of possible solutions usually can be handled by using a breadth-first search and cycle tests within the derivation tree.

However, if the problem has an infinite solution only, these approaches need infinite time to compute this solution, too. Usually recursive rule sets have been identified as a source for infinite solutions. But our algorithm indicates that even a non-recursive rule set can lead to an infinite realization set in case that the compensation of side effects lead to a continuous introduction of new constants. As an example, consider the following non-recursive example having an infinite model only that satisfies the given request $\nu_{\mathcal{D}}^+ = \{h\}$:

$$h \leftarrow e(X, Y) \wedge \neg i$$

$$\begin{array}{ll} i \leftarrow e(Y, X) \wedge e(Z, X) \wedge \neg eq(Y, Z) & j(X) \leftarrow e(X, Y) \wedge e(X, Z) \wedge \neg eq(Y, Z) \\ i \leftarrow e(X, Y) \wedge \neg j(X) & eq(X, X) \leftarrow e(X, Y) \\ i \leftarrow e(X, Y) \wedge \neg j(Y) & eq(Y, Y) \leftarrow e(X, Y) \end{array}$$

Relation e must contain an infinite number of facts in order to satisfy the VU request $\nu_{\mathcal{D}}^+ = \{h\}$. In this example, relation e corresponds to a mapping from the second attribute Y to the first attribute X such that each value of X is assigned to at least two values of Y and each value of Y is also value of X .

But from Algorithm 1 it can be directly followed that if no projection is employed within compensation paths of $\mathcal{R}^{\nabla} \cup \mathcal{R}^{\Delta}$, termination can be assured. In the example above, the path $\nabla_h^+(\vec{x}) \rightarrow \nabla_e^+(\vec{y}) \rightarrow \Delta_e^+(\vec{y}) \rightarrow \Delta_i^+ \rightarrow \Delta_h^-(\vec{x})$ violates this condition as it contains a (π sign changing) negation and a projection rule which together represents a possibly infinite generator of new constants. Although this - quite restrictive - condition uses the UP rules introduced above, it can be also formulated by means of the original deductive rules. The class of deductive databases for which the VU problem is decidable, however, may still include recursion, projections and arbitrary n-ary derived relations. This condition offers a general approach to finding better static criteria as it depends closely on the chosen propagation method and the way of optimizing them. This includes dynamic criteria taking into account the specific instance $\nu_{\mathcal{D}}$ as well as the current database state for optimizing the propagation rules.

Although our set-oriented approach fits well into a database context, it cannot be generally preferred over instance-oriented methods. In fact, approaches based on SLDNF [18,22] or tableaux calculus [1] may perform better in cases where only a small number of alternative solution paths have to be exploited. In general, however, none of the approaches can be preferred over the other as similar optimization effects can be incorporated in all those methods. As our search tree is completely materialized till a solution has been found, the space and time complexity for computing a finite model grows linear with the size of this solution. Because Ackermann's function can be modelled by stratifiable Datalog rules without function terms, however, the growth of constants in such a model can reach the same order of magnitude as the computation of Ackermann terms [8].

References

1. ARAVINDAN, C., BAUMGARTNER, P.: *Theorem Proving Techniques for View Deletion in Databases*. Journal of Symbolic Computation 29(2): 119–147 (2000).
2. ABDENNADHER, S., SCHÜTZ, H.: *Model Generation with Existentially Quantified Variables and Constraints*. ALP/HOA 1997: 256-272.
3. BRY, F., DECKER, H., MANTHEY, R.: *A Uniform Approach to Constraint Satisfaction and Constraint Satisfiability in Deductive Databases*. EDBT 1988: 488-505.
4. BEHREND, A.: *Soft stratification for magic set based query evaluation in deductive databases*. PODS 2003, New York, June 9–12, pages 102-110.
5. BEHREND, A.: *A Fixpoint Approach to State Generation for Stratifiable Disjunctive Deductive Databases*. Accepted for ADBIS 2007.
6. BEHREND, A., MANTHEY, R.: *Update Propagation in Deductive Databases Using Soft Stratification*. ADBIS 2004: 22-36.
7. BRY, F.: *Intensional Updates: Abduction via Deduction*. ICLP 1990: 561-575.
8. ECKERT, H.: *Ein regelbasierter Ansatz zur Analyse von Sichtenänderungswünschen in stratifizierbaren deduktiven Datenbanken*. Master Thesis, University of Bonn, 2004.
9. FARRÉ, C., TENIENTE, E., URPI, T.: *Query Containment Checking as a View Updating Problem*. DEXA 1998: 310-321.
10. GRANT, J., HORTY, J., LOBO, J., MINKER, J.: *View Updates in Stratified Disjunctive Databases*. JAR 11(2): 249-267 (1993).
11. GRIEFAHN, U.: *Reactive Model Computation - A Uniform Approach to the Implementation of Deductive Databases*. Dissertation, University of Bonn, 1997. http://www.cs.uni-bonn.de/idb/publications/diss_griefahn.ps.gz
12. INOUE, K., SAKAMA, C.: *A Fixpoint Characterization of Abductive Logic Programs*. JLP 27(2): 107-136 (1996).
13. KÜCHENHOFF, V.: *On the Efficient Computation of the Difference Between Consecutive Database States*. DOOD 1991: 478-502.
14. LOBO, J., MINKER, J., RAJASEKAR, A.: *Foundations of Disjunctive Logic Programming*. MIT Press, 1992.
15. LEVY, A., MUMICK, I., SAGIV, Y., SHMUELI, O.: *Equivalence, Query-Reachability, and Satisfiability in Datalog Extensions*. PODS 1993: 109-122.
16. LLOYD87, J. W.: *Foundations of Logic Programming (2nd Edition)*. Springer, Berlin, 1987.
17. RAINER MANTHEY. *Reflections on some fundamental issues of rule-based incremental update propagation*. DAISD 1994: 255-276.
18. OLIVÉ, A.: *Integrity Constraints Checking In Deductive Databases*. VLDB 1991: 513-523.
19. PRZYMUSINSKI T.: *Every Logic Program Has a Natural Stratification And an Iterated Least Fixed Point Model*. PODS 1989: 11-21.
20. SAGIV, Y.: *Optimizing Datalog Programs*. Foundations of Deductive Databases and Logic Programming 1988: 659-698.
21. D. SEIPEL, J. MINKER, AND C. RUIZ: *Model Generation and State Generation for Disjunctive Logic Programs*. Journal of Logic Programming 32(1): 49-69 (1997).
22. TENIENTE, E., OLIVÉ, A.: *The Events Method for View Updating in Deductive Databases*. EDBT 1992: 245-260.
23. TENIENTE, E., URPI, T.: *On the abductive or deductive nature of database schema validation and update processing problems*. TPLP 3(3): 287-327 (2003).