# The View-Update Problem for Indefinite Databases

Luciano Caroprese[1], Irina Trubitsyna[1], Mirosław Truszczyński[2], and Ester Zumpano[1]

[1] DEIS, Università della Calabria, 87030 Rende, Italy
caroprese|irina|zumpano@deis.unical.it
[2] Department of Computer Science, University of Kentucky, Lexington, KY 40506, USA
mirek@cs.uky.edu

**Abstract.** This paper introduces and studies a declarative framework for updating views over *indefinite databases*. An indefinite database is a database with null values that are represented, following the standard database approach, by a single *null* constant. The paper formalizes views over such databases as indefinite *deductive* databases, and defines for them several classes of database repairs that realize view-update requests. Most notable is the class of *constrained repairs*. Constrained repairs change the database "minimally" and avoid making arbitrary commitments. They narrow down the space of alternative ways to fulfill the view-update request to those that are grounded, in a certain strong sense, in the database, the view and the view-update request.

## 1 Introduction

A typical database system is large and complex. Users and applications rarely can access the entire system directly. Instead, it is more common that access is granted in terms of a *view*, a *virtual* database consisting of relations defined by a query to the *stored and maintained* database. Querying a view does not present a conceptual problem. In contrast, another key task, *view updating*, poses major challenges.

*Example 1.* Let $D = \{q(a,b)\}$ be a database over relation symbols $q$ and $r$, where the relation $r$ has arity three and is currently empty. Let us consider the view over $D$ given by the Datalog program $P = \{p(X) \leftarrow q(X,Y), r(X,Y,Z)\}$. That view consists of a single unary relation $p$. Given the present state of $D$, the view is empty.

To satisfy the request that $p(a)$ holds in the view (as it is now, it does not), one needs to update the database $D$. Such update consists of executing update actions that specify facts to insert to and to delete from $D$. These update actions (in a simplified setting that we consider for now) are "signed" facts $+F$ and $-G$, where $+F$ stands for "insert F" and $-G$ stands for "delete G." In our case, the set of update actions $\{-q(a,b), +q(a,a), +r(a,a,a)\}$ is a correct update to $D$. Executing it on $D$ results in the database $D' = \{q(a,a), r(a,a,a)\}$, which has the desired property that $p(a)$ holds in the view determined by $P$. There are also other ways to satisfy the user's request, for instance: $\{+r(a,b,a)\}$ and $\{+q(c,d), +r(c,d,d)\}$, where $c$ and $d$ are any elements of the domain of the database. □

As this example suggests, view updating consists of translating a *view-update request*, that is, an update request against the view, into an *update*, a set of update actions

against the stored (extensional) database. The example highlights the basic problem of view updating. It may be (in fact, it is common), that a view-update request can be fulfilled by any of a large number of database updates. One of them has to be committed to and executed. Thus, developing methods to automate the selection process or, at least, aid the user in making the choice is essential and has been one of the central problems of view updating [9, 2, 13, 11]. That problem is also the focus of our paper.[1]

To restrict the space of possible database updates to select from, it is common to consider only those that accomplish the view-update request and are in some sense minimal. That reduces the space of updates. For instance, the update $\{-q(a,b), +q(a,a), +r(a,a,a)\}$ in Example 1 is not minimal. We can remove $-q(a,b)$ from it and what remains is still an update that once executed ensures that $p(a)$ holds in the view. Minimal updates fulfilling the view-update request are commonly called *repairs*.

Even imposing minimality may still leave a large number of candidate repairs. In Example 1, updates $\{+r(a,b,\psi)\}$, where $\psi$ is any domain element, are repairs, and there are still more. As long as we insist on the completeness of the repaired database, there is little one can do at this point but ask the user to *select* one.

The situation changes if we are willing to allow indefiniteness (incomplete information) in databases. Given the "regular" structure of the family of repairs above, one could represent it by a single *indefinite* repair, $\{+r(a,b,\psi)\}$ regarding $\psi$ as a distinct *null* value standing for some unspecified domain elements. The choice facing the user substantially simplifies as possibly infinitely many repairs is reduced to only one.

This approach was studied by Farré et al. [5], where the terminology of Skolem constants rather than null values was used. However, while seemingly offering a plausible solution to the problem of multiple repairs, the approach suffers from two drawbacks. First, the use of multiple Skolem constants (essentially, multiple null values) violates the SQL standard [15]. Second, the approach assumes that the initial database is complete (does not contain null values). Thus, while the approach might be applied once, iterating it fails as soon as an indefinite database is produced, since no guidance on how to proceed in such case is given.

We consider the view-update problem in the setting in which *both* the original and the updated databases are indefinite (may contain occurrences of null values) and the database domain is possibly infinite. To be compliant with the SQL standard, we allow a single null value only. We denote it by $\perp$ and interpret it as expressing the existence of unknown values for each attribute (with possibly infinite domain) it is used for [10]. Typically, a database is represented as a set of facts. We propose to represent indefinite databases by *two* sets of facts that we interpret by means of a two-level closed-world assumption tailored to the case of indefinite information. We interpret indefinite databases in terms of their *possible worlds*. We extend the possible-world semantics to the setting of views, which we formalize in terms of *indefinite deductive databases*, and apply the resulting formalism to state and study the view-updating problem.

[1] In some cases no update satisfies the view-update request. The question whether the lack of an appropriate update is caused by errors in the design of the view, in the extensional database, or in the view-update request is interesting and deserves attention, but is outside the scope of the present work.

We then turn attention to the problem of multiple repairs. In general, using null values to encode multiple repairs is still not enough to eliminate them entirely (cf. our discussion above), and some level of the user's involvement may be necessary. Therefore, it is important to identify principled ways to narrow down the space of repairs for the user to consider. We propose a concept of minimality tailored precisely to the new setting. The primary concern is to minimize the set of new constants introduced by an update. The secondary concern is to minimize the degree of the semantic change. The resulting notion of minimality yields the notion of a *repair*.

Our concept of minimality leads us also to the concept of a *relevant* repair, an update that introduces no new constants and minimizes the degree of change. In Example 1, $\{+r(a,b,\bot)\}$, $\{+r(a,a,\bot),+q(a,a)\}$ and $\{+r(a,c,\bot),+q(a,c)\}$, where $c$ is an element of the database domain other than $a$ and $b$, are all repairs. The first two are obviously relevant, the third one is not.

Some occurrences of non-nullary constants in a relevant repair may still be "ungrounded" or "arbitrary," that is, replacing them with another constant results in a repair. For instance, replacing in the relevant repair $\{+r(a,a,\bot),+q(a,a)\}$ the second and the forth occurrences of $a$ with a fresh constant $c$ yields $\{+r(a,c,\bot),+q(a,c)\}$, an update that is a repair. Intuitively, "arbitrary" occurrences of constants, being replaceable, are not forced by the information present in the view (deductive database) and in the view-update request. By restricting relevant repairs to those *without* arbitrary occurrences of constants we arrive at the class of *constrained* repairs. In the view-update problem considered in Example 1, there is only *one* constrained repair, $\{+r(a,b,\bot)\}$.

Finally, we study the complexity of the problems of the existence of repairs, relevant repairs and constrained repairs. We obtain precise results for the first two classes and an upper bound on the complexity for the last class.

To summarize, our main contributions are as follows. We propose a two-set representation of indefinite database that is more expressive than the standard one. We define the semantics and the operation of updating indefinite databases (Section 2). We define views over indefinite databases (indefinite deductive databases), and generalize the semantics of indefinite databases to views (Section 3). We state and study the view-update problem in the general setting when the initial and the repaired databases are indefinite. We propose a notion of minimality of an update and use it to define the concept of a repair. We address the problem of multiple repairs by defining relevant and constrained repairs (Section 4). We study the complexity of problems of existence of repairs, relevant repairs and constrained repairs (Section 5).

## 2 Indefinite Databases

We consider a finite set $\Pi$ of relation symbols and a set *Dom* of constants that includes a designated element $\bot$, called the *null value*. We define $Dom_d = Dom \setminus \{\bot\}$. Normally, we assume that *Dom* is an infinite countable set. However, for the sake of simplicity, in several of the examples the set *Dom* is finite.

Some predicates in $\Pi$ are designated as *base* (or *extensional*) predicates and all the remaining ones are understood as *derived* (or *intensional*) predicates. A *term* is a constant from *Dom* or a variable. An *atom* is an expression of the form $p(t_1,\ldots,t_k)$,

where $p \in \Pi$ is a *predicate symbol* of arity $k$ and $t_i$'s are terms. An atom is *ground* if it does not contain variables. We refer to ground atoms as *facts*. We denote the set of all facts by $At$. We call facts defined in terms of base and derived predicates *base facts* and *derived facts* respectively. A fact is *definite* if it does not contain occurrences of $\bot$. Otherwise, it is *indefinite*. Given a set $S$ of atoms, we define $Dom(S)$ (resp. $Dom_d(S)$) as the set of constants in $Dom$ (resp. $Dom_d$) occurring in $S$. For every two tuples of terms $t = (t_1, \ldots, t_k)$ and $t' = (t'_1, \ldots, t'_k)$ and every $k$-ary predicate symbol $p \in \Pi$, we write $t \preceq t'$ and $p(t) \preceq p(t')$ if for every $i$, $1 \leq i \leq k$, $t_i = t'_i$ or $t_i = \bot$. We say in such case that $t'$ and $p(t')$ are *at least as informative* as $t$ and $p(t)$, respectively. If, in addition, $t \neq t'$, we write $t \prec t'$ and $p(t) \prec p(t')$, and say that $t'$ and $p(t')$ are *more informative* than $t$ and $p(t)$. Sometimes, we say "at most as informative" and "less informative," with the obvious understanding of the intended meaning. We also define $t$ and $t'$ (respectively, $p(t)$ and $p(t')$) to be *compatible*, denoted by $t \approx t'$ (respectively, $p(t) \approx p(t')$), if for some $k$-tuple $s$ of terms, $t \preceq s$ and $t' \preceq s$. Finally, for a set $D \subseteq At$, we define

$$D^{\Downarrow} = \{a \mid \text{there is } b \in D \text{ s.t. } a \preceq b\} \qquad D^{\Uparrow} = \{a \mid \text{there is } b \in D \text{ s.t. } b \preceq a\}$$
$$D^{\approx} = \{a \mid \text{there is } b \in D \text{ s.t. } b \approx a\} \qquad D^{\sim} = D^{\approx} \setminus D^{\Downarrow}.$$

To illustrate, let $q$ be a binary relation symbol and $Dom = \{\bot, 1, 2\}$. Then:

$\{q(1,\bot)\}^{\Downarrow} = \{q(1,\bot), q(\bot,\bot)\}$ $\qquad$ $\{q(1,\bot)\}^{\Uparrow} = \{q(1,\bot), q(1,1), q(1,2)\}$

$\{q(1,\bot)\}^{\approx} = \{q(1,\bot), q(1,1), q(1,2), q(\bot,1), q(\bot,2), q(\bot,\bot)\}$

$\{q(1,\bot)\}^{\sim} = \{q(1,1), q(1,2), q(\bot,1), q(\bot,2)\}$.

We note also note that $D^{\approx} = (D^{\Uparrow})^{\Downarrow}$.

In the most common case, databases are finite subsets of $At$ that contain *definite* facts only. The semantics of such databases is given by the *closed-world assumption* or CWA [12]: a definite fact $q$ is *true* in a database $D$ if $q \in D$. Otherwise, $q$ is *false* in $D$. We are interested in databases that may contain indefinite facts, too. Generalizing, we will *for now* assume that an indefinite database is a finite set of possibly indefinite atoms. The key question is that of the semantics of indefinite databases.

Let $D$ be an indefinite database. Clearly, all facts in $D$ are true in $D$. In addition, any fact that is less informative than a fact in $D$ is also true in $D$. Indeed, each such fact represents an existential statement, whose truth is established by the presence of a more informative fact in $D$ (for instance, the meaning of $p(\bot)$ is that there is an element $c$ in the domain of the database such that $p(c)$ holds; if $p(1) \in D$, that statement is true). Summarizing, every fact in $D^{\Downarrow}$ is true in $D$.

By CWA adapted for the setting of indefinite databases [10], facts that are not in $D^{\Downarrow}$ are *not* true in $D$, as $D$ contains no evidence to support their truth. Those facts among them that are compatible with facts in $D$ (in our notation, facts in $D^{\sim}$), might actually be true, but the database just does not know that. Of course, they may also be false, the database does not exclude that possibility either. They are regarded as *unknown*. By CWA again, the facts that are not compatible with any fact in $D$ are false in $D$, as $D$ provides no explicit evidence otherwise.

The simple notion of an indefinite database, while intuitive and having a clear semantics, has a drawback. It has a limited expressive power. For instance, there is no database $D$ to represent our knowledge that $p(1)$ is false and that there is some definite $c$ such that $p(c)$ holds (clearly, this $c$ is not 1). To handle such cases, we introduce a more general concept of an indefinite database, still using CWA to specify its meaning.

**Definition 1.** *An* indefinite database *(a* database*, for short) is a pair $\mathscr{I} = \langle D, E\rangle$, where $D$ and $E$ are finite sets of (possibly indefinite) facts.* □

The intended role of $D$ is to represent all facts that are true in the database $\langle D, E\rangle$, while $E$ is meant to represent *exceptions*, those facts that normally would be unknown, but are in fact false (and the database knows it). More formally, the semantics of indefinite databases is presented in the following definition.

**Definition 2.** *Let $\langle D, E\rangle$ be a database and let $q \in At$ be a fact. Then: (1) $q$ is* true *in $\langle D, E\rangle$, written $\langle D, E\rangle \models q$, if $q \in D^{\Downarrow}$; (2) $q$ is* unknown *in $\langle D, E\rangle$, if $q \in (D^{\approx} \setminus D^{\Downarrow}) \setminus E^{\Uparrow}$ $(= D^{\sim} \setminus E^{\Uparrow})$; (3) $q$ is* false *in $\langle D, E\rangle$, denoted $\langle D, E\rangle \models \neg q$, in all other cases, that is, if $q \notin D^{\approx}$ or if $q \in D^{\sim} \cap E^{\Uparrow}$.* □

The use of $E^{\Uparrow}$ in the definition (items (2) and (3)) reflects the property that if an atom $a$ is false then every atom $b$ at least as informative as $a$ must be false, too.

We denote the sets of all facts that are true, unknown and false in a database $\mathscr{I} = \langle D, E\rangle$ by $\mathscr{I}_t$, $\mathscr{I}_u$ and $\mathscr{I}_f$, respectively. Restating the definition we have:

$$\mathscr{I}_t = D^{\Downarrow}, \quad \mathscr{I}_u = D^{\sim} \setminus E^{\Uparrow}, \text{ and } \quad \mathscr{I}_f = (At \setminus D^{\approx}) \cup (D^{\sim} \cap E^{\Uparrow}).$$

*Example 2.* The knowledge that $p(c)$ holds for some constant $c$ and that $p(1)$ is false can be captured by the database $\langle \{p(\bot)\}, \{p(1)\}\rangle$. The database $\langle \{q(\bot,\bot), q(1,1)\}, \{q(1,\bot)\}\rangle$ specifies that the atoms $q(1,1)$, $q(1,\bot)$, $q(\bot,1)$ and $q(\bot,\bot)$ are true, that all definite atoms $q(1,d)$, with $d \neq 1$, are false, and that all other atoms $q(c,d)$ are unknown. While the fact that $q(\bot,\bot)$ is true follows from the fact that $q(1,1)$ is true, the presence of the former in the database is not redundant, that is, the database without it would have a different meaning. Namely, $q(\bot,\bot)$ makes all atoms $q(a,b)$ potentially unknown (with some of them true or false due to other elements of the database). □

**Definition 3.** *A set $W$ of* definite *facts is a* possible world *for a database $\mathscr{I} = \langle D, E\rangle$ if $\mathscr{I}_t \subseteq W^{\Downarrow}$ ($W$ "explains" all facts that are true in $\mathscr{I}$), $W \subseteq \mathscr{I}_t \cup \mathscr{I}_u$ (only definite facts that are true or unknown appear in a possible world).* □

Databases represent sets of possible worlds. For a database $\mathscr{I}$, we write $\mathscr{W}(\mathscr{I})$ to denote the family of all possible worlds for $\mathscr{I}$. Due to the absence of indefinite facts in $W \in \mathscr{W}(\mathscr{I})$, every fact in $At$ is either true (if it belongs to $W^{\Downarrow}$) or false (otherwise) w.r.t. $W$. Extending the notation we introduced earlier, for a possible world $W$ and $a \in At$ we write $W \models a$ if $a \in W^{\Downarrow}$ and $W \models \neg a$, otherwise. The following proposition shows that the semantics of a database can be stated in terms of its possible worlds.

**Proposition 1.** *Let $\mathscr{I}$ be a database and $q$ a fact. Then $q \in \mathscr{I}_t$ if and only if $W \models q$, for every $W \in \mathscr{W}(\mathscr{I})$, and $q \in \mathscr{I}_f$ if and only if $W \models \neg q$, for every $W \in \mathscr{W}(\mathscr{I})$.* □

*Updating* a database $\langle D, E\rangle$ consists of executing on it *update actions*: inserting a base fact $a$ into $D$ or $E$, and deleting a base fact $a$ from $D$ or $E$. We denote them by $+a^D$, $+a^E$, $-a^D$ and $-a^E$, respectively. For a set $U$ of update actions, we define $U^D_+ = \{a \mid +a^D \in U\}$, $U^E_+ = \{a \mid +a^E \in U\}$, $U^D_- = \{a \mid -a^D \in U\}$, and $U^E_- = \{a \mid -a^E \in U\}$. To be executable, a set $U$ of update actions must not contain *contradictory* update

actions: $+a^D$ and $-a^D$, or $+a^E$ and $-a^E$. A contradiction-free set $U$ of update actions is an *update*. We denote the set of all updates (in the fixed language we consider) by $\mathscr{U}$.

We now define the operation to update a database, that is, to apply an update to it.

**Definition 4.** *Let $\mathscr{I} = \langle D, E\rangle$ be an indefinite database and $U$ an update. We define $\mathscr{I} \circ U$ as the database $\langle D', E'\rangle$, where $D' = (D \cup U^D_+) \setminus U^D_-$ and $E' = (E \cup U^E_+) \setminus U^E_-$.* □

## 3 Indefinite Deductive Databases

*Integrity constraints* (ICs, for short) are first-order sentences in the language $\mathscr{L}$ determined by the set of predicates $\Pi$ and by the set $Dom_d$ of definite constants. A database with integrity constraints is a pair $\langle \mathscr{I}, \eta\rangle$, where $\mathscr{I}$ is a database and $\eta$ is a set of ICs. Possible worlds can be regarded as interpretations of the language $\mathscr{L}$ with $Dom_d$ as their domain. This observation and Proposition 1 suggest a definition of the semantics of databases with ICs.

**Definition 5.** *Let $\langle \mathscr{I}, \eta\rangle$ be a database with ICs. A possible world $W \in \mathscr{W}(\mathscr{I})$ is a* possible world for a database $\langle \mathscr{I}, \eta\rangle$ *if $W$ satisfies every integrity constraint in $\eta$. We denote the set of possible worlds of $\langle \mathscr{I}, \eta\rangle$ by $\mathscr{W}(\mathscr{I}, \eta)$. A database with ICs, $\langle \mathscr{I}, \eta\rangle$, is* consistent *if $\mathscr{W}(\mathscr{I}, \eta) \neq \emptyset$. Otherwise, $\langle \mathscr{I}, \eta\rangle$ is* inconsistent.

*A fact $q$ is true in $\langle \mathscr{I}, \eta\rangle$ if $W \models q$, for every $W \in \mathscr{W}(\mathscr{I}, \eta)$; $q$ is false in $\langle \mathscr{I}, \eta\rangle$ if $W \models \neg q$, for every $W \in \mathscr{W}(\mathscr{I}, \eta)$; otherwise, $q$ is unknown in $\langle \mathscr{I}, \eta\rangle$.* □

*Example 3.* Let us consider the database $\mathscr{I} = \langle \{p(1), p(2), q(\bot)\}, \emptyset\rangle$ (there are no exceptions) and let $\eta = \{\forall X(q(X) \rightarrow p(X))\}$. Possible worlds of $\mathscr{I}$ include $\{p(1), p(2), q(1)\}$, $\{p(1), p(2), q(2)\}$ and $\{p(1), p(2), q(3)\}$. The first two satisfy the integrity constraint, the third one does not. Thus, only the first two are possible worlds of $\langle \mathscr{I}, \eta\rangle$. Since $p(1)$ belongs to all possible worlds of $\langle \mathscr{I}, \eta\rangle$, $p(1)$ is true in $\langle \mathscr{I}, \eta\rangle$. Further, $p(3)$ is false in every possible world of $\mathscr{I}$ and so also in every possible world of $\langle \mathscr{I}, \eta\rangle$. Thus, $p(3)$ is false in $\langle \mathscr{I}, \eta\rangle$. Lastly, we note that $q(1)$ and $q(2)$ are unknown in $\langle \mathscr{I}, \eta\rangle$, while $q(3)$ is false (due to $p(3)$ being false and the integrity constraint). □

We note that the possible-world semantics can capture additional information contained in integrity constraints. In Example 3, the semantics derives that $q(3)$ is *false* in $\langle \mathscr{I}, \eta\rangle$ even though this knowledge is not present in the database $\mathscr{I}$.

The concepts of an update and of the operation to execute an update on a database extend literally to the case of databases with ICs.

Following Ullman [16], *views* are safe Datalog$^\neg$ programs. We use the standard terminology and talk about (Datalog$^\neg$) *rules*, and *bodies* and *heads* of rules. A rule is *safe* if each variable occurring in the head or in a negative literal in the body also occurs in a positive literal in the body. A Datalog$^\neg$ program is safe if each rule is *safe*. We assume that views do not contain occurrences of $\bot$. The semantics of Datalog$^\neg$programs is given in terms of *answer sets* [6, 7]. A precise definition of that semantics is immaterial to our study and so we do not provide the details.

**Definition 6.** *An* indefinite deductive database *(from now, simply, a deductive database) is a tuple $\mathscr{D} = \langle \mathscr{I}, \eta, P\rangle$, where $\mathscr{I}$ is a database, $\eta$ is a set of integrity constraints, and*

*$P$ is a safe Datalog$^\neg$ program (the specification of a* view*) such that no predicate occurring in the head of a rule in $P$ is a base predicate.* □

Clearly, a deductive database with the empty view is a database with ICs, and a deductive database with the empty view and no ICs is simply a database.

**Definition 7.** *A deductive database $\mathscr{D} = \langle \mathscr{I}, \eta, P \rangle$ is* consistent *if $\langle \mathscr{I}, \eta \rangle$ is consistent and for every possible world $W \in \mathscr{W}(\mathscr{I}, \eta)$, the program $W \cup P$ has answer sets. We denote the family of all those answer sets by $\mathscr{W}(\mathscr{D})$ or $\mathscr{W}(\mathscr{I}, \eta, P)$. We call elements of $\mathscr{W}(\mathscr{D})$* possible worlds *of $\mathscr{D}$.* □

There is an alternative to our concept of consistency. One could define a deductive database $\langle \mathscr{I}, \eta, P \rangle$ as consistent if for *at least one* world $W \in \mathscr{W}(\mathscr{I}, \eta)$, the program $W \cup P$ has an answer set. That concept of consistency would allow situations where for some possible worlds of $\langle \mathscr{I}, \eta \rangle$, one of which could be a description of the real world, the view $P$ does not generate any meaningful virtual database. Our concept of consistency is more robust. It guarantees that the user can have a view of a database no matter how the real world looks like, that is, which of the possible worlds describes it.

*Example 4.* Let $\mathscr{D} = \langle \mathscr{I}, \eta, P \rangle$ be a deductive database, where $\mathscr{I} = \langle \{p(\bot)\}, \emptyset \rangle$, $\eta = \emptyset$ and $P = \{t \leftarrow p(1), p(2), \mathit{not}\ t\}$, for some derived ground atom $t$. Every non-empty set $W \subseteq \{p(u) \mid u \in \mathit{Dom}_d\}$ is a possible world of $\langle \mathscr{I}, \eta \rangle$. In particular, the set $\{p(1), p(2)\}$ is a possible world of $\langle \mathscr{I}, \eta \rangle$. Since the program $P \cup \{p(1), p(2)\}$ has no answer sets, $\mathscr{D}$ is inconsistent (according to our definition). If $\mathscr{D}' = \langle \mathscr{I}, \{p(1) \wedge p(2) \rightarrow \bot\}, P \rangle$, then the integrity constraint in $\mathscr{D}'$ eliminates the offending possible world and one can check that for every possible world $W$ of $\langle \mathscr{I}, \{p(1) \wedge p(2) \rightarrow \bot\} \rangle$, $P \cup W$ has an answer set. Thus, $\mathscr{D}'$ is consistent.[2] □

The concept of an update extends in a natural way to deductive databases. If $U$ is an update, and $\mathscr{D} = \langle \mathscr{I}, \eta, P \rangle$, we define the result of updating $\mathscr{D}$ by $U$ by $\mathscr{D} \circ U = \langle \mathscr{I} \circ U, \eta, P \rangle$.

Next, we define the semantics of a deductive database $\mathscr{D} = \langle \mathscr{I}, \eta, P \rangle$, again building on the characterization given by Proposition 1.

**Definition 8.** *A fact $a \in At$ is* true *in a deductive database $\mathscr{D} = \langle \mathscr{I}, \eta, P \rangle$, denoted by $\mathscr{D} \models a$, if for every possible world $W \in \mathscr{W}(\mathscr{D})$, $W \models a$; $a$ is* false *in $\mathscr{D}$, denoted by $\mathscr{D} \models \neg a$, if for every $W \in \mathscr{W}(\mathscr{D})$, $W \models \neg a$; $a$ is* unknown *in $\mathscr{D}$, otherwise. We denote the truth value of $a$ in $\mathscr{D}$ by $v_{\mathscr{D}}(a)$.* □

*Example 5.* Let $\mathit{Dom} = \{\bot, 1, 2, 3\}$ and $\mathscr{D} = \langle \mathscr{I}, \eta, P \rangle$ be a deductive database, where $\mathscr{I} = \langle \{p(\bot)\}, \emptyset \rangle$, $\eta = \{p(2) \rightarrow \bot\}$ and $P = \{q(X) \leftarrow p(X)\}$.

We have $\mathscr{W}(\mathscr{I}, \eta) = \{\{p(1)\}, \{p(3)\}, \{p(1), p(3)\}\}$. Each of the possible worlds in $\mathscr{W}(\mathscr{I}, \eta)$, when extended with the view $P$, gives rise to a program that has answer sets. Thus, $\mathscr{D}$ is consistent. Moreover, the possible worlds for $\mathscr{D}$ are $\{p(1), q(1)\}$, $\{p(3), q(3)\}$ and $\{p(1), q(1), p(3), q(3)\}$ (in this case, one for each possible world in $\mathscr{W}(\mathscr{I}, \eta)$). It follows that $p(\bot)$ and $q(\bot)$ are true, $p(2)$ and $q(2)$ are false, and $p(1), q(1), p(3)$ and $q(3)$ are unknown in $\mathscr{D}$. □

[2] We point out that in the paper, we overload the notation $\bot$. We use it to denote both the single null value in the language and the falsity symbol in the first-order language used for integrity constraints. Since the meaning is always clear from the context, no ambiguity arises.

## 4 View Updating

In the *view update problem*, the user specifies a *request*, a list of facts the user learned (observed) to be true or false, and wants the stored database to be updated to reflect it.[3]

**Definition 9.** *A* request *over a deductive database $\mathscr{D}$ is a pair $\mathscr{S} = (\mathscr{S}_t, \mathscr{S}_f)$, where $\mathscr{S}_t$ and $\mathscr{S}_f$ are disjoint sets of facts requested to be true and false, respectively.* □

To fulfill a request we need an update which, when executed, yields a database such that the view it determines satisfies the request. We call such updates *weak repairs*.

**Definition 10.** *Let $\mathscr{D} = \langle \mathscr{I}, \eta, P \rangle$ be a deductive database and $\mathscr{S}$ a request. An update $U$ for $\mathscr{I}$ is a* weak repair *for $(\mathscr{D}, \mathscr{S})$ if $U$* fulfills *$\mathscr{S}$, that is, if for every $a \in \mathscr{S}_t$, $v_{\mathscr{D} \circ U}(a) = true$ and for every $a \in \mathscr{S}_f$, $v_{\mathscr{D} \circ U}(a) = false$.* □

We are primarily interested in updates that do not drastically change the database. One condition of being "non-drastic" is not to introduce new predicate or constant symbols. That leads us to the notion of a relevant weak repair.

**Definition 11.** *Let $\mathscr{D} = \langle \mathscr{I}, \eta, P \rangle$ be a deductive database and $\mathscr{S}$ a request. A constant is* relevant with respect to *$\mathscr{D}$* and *$\mathscr{S}$ if it occurs in $\mathscr{D}$, or $\mathscr{S}$, or if it is $\perp$. A predicate is* relevant with respect to *$\mathscr{D}$* and *$\mathscr{S}$ if it occurs in $\mathscr{D}$ or in $\mathscr{S}$. A weak repair $U$ for $(\mathscr{D}, \mathscr{S})$ is* relevant *if every constant and predicate occurring in $U$ is* relevant. □

More generally, a weak repair is "non-drastic" if it minimizes the change it incurs [14]. There are two aspects to the minimality of change: (1) minimizing the set of new predicate symbols and constants introduced by an update to the database (in the extreme case, no new symbols must be introduced, and we used that requirement to define relevant weak repairs above); (2) minimizing the change in the truth values of facts with respect to the database. Following the *Ockham's Razor* principle to avoid introducing new entities unless necessary, we take the minimality of the set of new symbols as a primary consideration. To define the resulting notion of change minimality, we assume that the truth values are ordered *false* $\leq$ *unknown* $\leq$ *true*. Further, for a deductive database $\mathscr{D}$, a request set $\mathscr{S}$ and an update $U \in \mathscr{U}$ we define $NC(\mathscr{D}, \mathscr{S}, U)$ as the set of non nullary constants that occur in $U$ and not in $\mathscr{D}$ and $\mathscr{S}$.

**Definition 12.** *Let $\mathscr{D} = \langle \mathscr{I}, \eta \rangle$ be a database with integrity constraints. For updates $V, U \in \mathscr{U}$, we define $U \sqsubseteq V$ if: $NC(\mathscr{D}, \mathscr{S}, U) \subset NC(\mathscr{D}, \mathscr{S}, V)$, or $NC(\mathscr{D}, \mathscr{S}, U) = NC(\mathscr{D}, \mathscr{S}, V)$ and for every* base atom *$a$*

1. *if $v_{\mathscr{D}}(a) = true$, then $v_{\mathscr{D} \circ U}(a) \geq v_{\mathscr{D} \circ V}(a)$*
2. *if $v_{\mathscr{D}}(a) = false$, then $v_{\mathscr{D} \circ V}(a) \geq v_{\mathscr{D} \circ U}(a)$*
3. *if $v_{\mathscr{D}}(a) = unknown$, then $v_{\mathscr{D} \circ U}(a) = unknown$ or $v_{\mathscr{D} \circ V}(a) = v_{\mathscr{D} \circ U}(a)$.*

*We also define $U \sqsubset V$ if $U \sqsubseteq V$ and $V \not\sqsubseteq U$.* □

[3] We do not allow requests that facts be *unknown*. That is, we only allow definite requests. While there may be situations when all the user learns about the fact is that it is unknown, they seem to be rather rare. In a typical situation, the user will learn the truth or falsity of a fact.

We now define the classes of *repairs* and *relevant repairs* as subclasses of the respective classes of weak repairs consisting of their $\sqsubseteq$-minimal elements.

**Definition 13.** *Let $\mathscr{D} = \langle \mathscr{I}, \eta, P \rangle$ be a deductive database and $\mathscr{S}$ a request. A (*relevant*)* repair *for $(\mathscr{D}, \mathscr{S})$ is a $\sqsubseteq$-minimal (relevant) weak repair for $(\mathscr{D}, \mathscr{S})$.* □

We note that the existence of (weak) repairs does not guarantee the existence of relevant (weak) repairs. The observation remains true even if the view is empty.

*Example 6.* Let $\mathscr{D} = \langle \mathscr{I}, \eta, P \rangle$, where $\mathscr{I} = \langle \emptyset, \emptyset \rangle$, $\eta = \emptyset$ and $P = \{t \leftarrow p(x), q(x)\}$. If the request is $(\{t\}, \emptyset)$, then each repair is of the form $\{+p(i)^D, +q(i)^D\}$, for some $i \in Dom_d$. None of them is relevant. (We note that $\{+p(\bot)^D, +q(\bot)^D\}$ is not a (weak) repair. The database resulting from the update would admit possible worlds of the form $\{p(i), q(j)\}$, where $i \neq j$. Clearly, the corresponding possible world of the view over any such database does not contain $t$ and so the update does not fulfill the request.) □

Some relevant constants are not "forced" by the database and the request, that is, can be replaced by other constants. If such constants are present in a relevant (weak) repair, this repair is *arbitrary*. Otherwise, it is *constrained*. A formal definition follows.

**Definition 14.** *Let $\mathscr{D} = \langle \mathscr{I}, \eta, P \rangle$ be a deductive database and $\mathscr{S}$ a request. A relevant (weak) repair $U$ for $(\mathscr{D}, \mathscr{S})$ is* constrained *if there is no non-nullary constant $a$ in $U$ such that replacing some occurrences of $a$ in $U$ with a constant $b \neq a$ ($b$ might be $\bot$), results in a weak repair for $(\mathscr{D}, \mathscr{S})$.* □

*Example 7.* Let $\mathscr{D} = \langle \mathscr{I}, \eta, P \rangle$, where $\mathscr{I} = \langle \{p(1), h(2)\}, \emptyset \rangle$, $\eta = \emptyset$ and $P = \{t \leftarrow p(X), q(X);\ \ s \leftarrow r(X)\}$. Let us consider the request $\mathscr{S} = (\{s, t\}, \emptyset)$. The updates $\mathscr{R}_i = \{+q(1)^D, +r(i)^D\}$, $i \in \{\bot, 1, 2\}$, and $\mathscr{R}'_i = \{+q(2)^D, +p(2)^D, +r(i)^D\}$, $i \in \{\bot, 1, 2\}$, are relevant weak repairs. One of them, $\mathscr{R}_\bot$, is constrained. Indeed, replacing in $\mathscr{R}_\bot$ the unique occurrence of a non-nullary constant (in this case, 1) with any other constant does not yield a weak repair. On the other hand, $\mathscr{R}_i$, $i \in \{1, 2\}$, and $\mathscr{R}'_i$, $i \in \{\bot, 1, 2\}$, are not constrained. Indeed, replacing with 3 the second occurrence of 1 in $\mathscr{R}_1$, or the occurrence of 2 in $R_2$, or both occurrences of 2 in $\mathscr{R}'_i$ in each case results in a weak repair. Also weak repairs $\mathscr{R}_i = \{+q(1)^D, +r(i)^D\}$ and $\mathscr{R}'_i = \{+q(2)^D, +p(2)^D, +r(i)^D\}$, $i \in \{3, \ldots\}$, are not constrained as they are not even relevant. □

We stress that, in order to test whether a relevant (weak) repair $\mathscr{R}$ is constrained, we need to consider every *subset of occurrences* of non-nullary constants in $\mathscr{R}$. For instance, in the case of the repair $\mathscr{R}_1 = \{q(1), r(1)\}$ from Example 7, the occurrence of the constant 1 in $q(1)$ is constrained by the presence of $p(1)$. Replacing that occurrence of 1 with 3 does not result in a weak repair. However, replacing the occurrence of 1 in $r$ with 3 gives a weak repair and shows that $\mathscr{R}_1$ is not constrained.

*Example 8.* Let $Dom = \{\bot, 1, 2, \ldots\}$ and $\mathscr{D} = \langle \mathscr{I}, \eta, P \rangle$, where $\mathscr{I} = \langle \{q(1,2), s(1,2,3)\}, \emptyset \rangle$, $\eta = \emptyset$ and $P = \{p(X) \leftarrow q(X,Y), r(X,Y,Z);\ \ r(X,Y,Z) \leftarrow s(X,Y,Z), t(X,Y,Z)\}$. Let us consider the request $\mathscr{S} = (\{p(1)\}, \emptyset)$. In this case, our approach yields a unique constrained repair $\mathscr{R} = \{+t(1,2,3)^D\}$. It recognizes that thanks to $s(1,2,3)$ simply inserting $t(1,2,3)$ guarantees $r(1,2,3)$ to be true and, consequently, ensures the presence of $p(1)$ in the view. There are other repairs and other relevant repairs, but only the one listed above is constrained. □

We observe that every (relevant, constrained) weak repair contains a (relevant, constrained) repair.

**Proposition 2.** *Let $\mathscr{D} = \langle \mathscr{I}, \eta, P \rangle$ be a deductive database and $\mathscr{S}$ a request. A (relevant, constrained) repair for $(\mathscr{D}, \mathscr{S})$ exists if and only if a (relevant, constrained) weak repair for $(\mathscr{D}, \mathscr{S})$ exists.* □

## 5 Complexity

Finally, we discuss the complexity of problems concerning the existence of (weak) repairs of types introduced above. The results we present here have proofs that are non-trivial despite rather strong assumptions we adopted.

We assume that the sets of base and derived predicate symbols, the set of integrity constraints $\eta$ and the view $P$ are fixed. The only varying parts in the problems are a database $\mathscr{I}$ and a request $\mathscr{S}$. That is, we consider the *data complexity* setting. Moreover, we assume that $Dom = \{\bot, 1, 2, \ldots\}$, and take $=$ and $\leq$, both with the standard interpretation on $\{1, 2, \ldots\}$, as the *only* built-in relations. We restrict integrity constraints to expressions of the form: $\exists X(\forall Y(A_1 \wedge \ldots \wedge A_k \rightarrow B_1 \vee \ldots \vee B_m))$, where $A_i$ and $B_i$ are atoms with no occurrences of $\bot$ constructed of base and built-in predicates, and where every variable occurring in the constraint belongs to $X \cup Y$, and occurs in some atom $A_i$ built of a base predicate.

We start by stating the result on the complexity of deciding the consistency of an indefinite database with integrity constraints. While interesting in its own right, it is also relevant to problems concerning the existence of repairs, as one of the conditions for $U$ to be a repair is that the database that results from executing $U$ be consistent.

**Theorem 1.** *The problem to decide whether a database $\langle \mathscr{I}, \eta \rangle$ has a possible world (is consistent) is NP-complete.* □

We now turn attention to the problem of checking request satisfaction. Determining the complexity of that task is a key stepping stone to the results on the complexity of deciding whether updates are (weak) repairs that are necessary for our results on the complexity of the existence of (weak) repairs. However, checking request satisfaction turns out to be a challenge even for very simple classes of views. In this paper, we restrict attention to the case when $P$ is a safe definite (no constraints) acyclic (no recursion) Horn program, although we obtained Proposition 3 in a more general form.

**Proposition 3.** *The problem to decide whether a ground atom $t$ is true in a deductive database $\langle \mathscr{I}, \eta, P \rangle$, where $\langle \mathscr{I}, \eta \rangle$ is consistent and $P$ is a safe Horn program, is in the class co-NP.* □

Next, we consider the problem to decide whether a ground atom $t$ is false in a deductive database $\langle \mathscr{I}, \eta, P \rangle$. We state it separately from the previous one as our present proof of that result requires the assumption of acyclicity.

**Proposition 4.** *The problem to decide whether a ground atom $t$ is false (ground literal $\neg t$ is true) in a deductive database $\langle \mathscr{I}, \eta, P \rangle$, where $\langle \mathscr{I}, \eta \rangle$ is consistent and $P$ is an acyclic Horn program, is in the class co-NP.* □

With Propositions 3 and 4 in hand, we move on to study the complexity of the problems of the existence of weak repairs. First, we establish an upper bound on the complexity of checking whether and update is a (relevant) weak repair.

**Proposition 5.** *Let $\mathscr{D} = \langle \mathscr{I}, \eta, P \rangle$, where $\eta$ is a set of integrity constraints, P an acyclic Horn program, U an update and $\mathscr{S}$ a request set. The problem of checking whether an update U is a weak repair (relevant weak repair) for $(\mathscr{D}, \mathscr{S})$ is in $\Delta_2^P$.* □

With the results above, we can address the question of the complexity of the existence of repairs. The first problem concerns weak repairs and stands apart from others. It turns out, that deciding the existence of a weak repair is NP-complete, which may seem at odds with Proposition 5 (an obvious non-deterministic algorithm guesses an update $U$ and checks that it is a weak repair apparently performing a "$\Sigma_2^P$ computation"). However, this low complexity of the problem is simply due to the fact that there are no relevance, constrainedness or minimality constraints are imposed on weak repairs. Thus, the question can be reduced to the question whether there is a "small" database $\mathscr{J}$, in which the request holds. The corresponding weak repair consists of deleting all elements from $\mathscr{I}$ and "repopulating" the resulting empty database so that to obtain $\mathscr{J}$.

**Theorem 2.** *Let $\mathscr{D} = \langle \mathscr{I}, \eta, P \rangle$, where $\eta$ is a set of integrity constraints, and P an acyclic Horn program, and let $\mathscr{S}$ be a request set. The problem of deciding whether there is a weak repair for $(\mathscr{D}, \mathscr{S})$ is NP-complete.* □

As noted, the case of the existence of weak repairs is an outlier and deciding the existence of (weak) repairs of other types is much harder (under common assumptions concerning the polynomial hierarchy).

**Theorem 3.** *Let $\mathscr{D} = \langle \mathscr{I}, \eta, P \rangle$, where $\eta$ is a set of integrity constraints, and P an acyclic Horn program, and let $\mathscr{S}$ be a request set. The problems of deciding whether there is a relevant weak repair and whether there is a relevant repair for $(\mathscr{D}, \mathscr{S})$ are $\Sigma_2^P$-complete.* □

The last result concerns constrained (weak) repairs. It provides an upper bound on the complexity of the problem of deciding the existence of constrained repairs. We conjecture that the upper bound is in fact tight but have not been able to prove it. We leave the problem for future work.

**Theorem 4.** *Let $\mathscr{D} = \langle \mathscr{I}, \eta, P \rangle$, where $\eta$ is a set of integrity constraints, and P an acyclic Horn program, and let $\mathscr{S}$ be a request set. The problems of deciding whether there is a constrained weak repair and whether there is a constrained repair for $(\mathscr{D}, \mathscr{S})$ are in $\Sigma_3^P$.* □

## 6 Discussion and conclusion

We presented a declarative framework for view updating and integrity constraint maintenance for indefinite databases. The framework is based on the notion of an indefinite deductive database. In our approach, the indefiniteness appears in the extensional

database and is modeled by a single null value, consistent with the standards of database practice (a condition not followed by earlier works on the view-update problem over indefinite databases). We defined a precise semantics for indefinite deductive databases in terms of possible worlds. We used the framework to formulate and study the view-update problem. Exploiting the concept of minimality of change introduced by an update, we defined several classes of repairs, including relevant and constrained repairs, that translate an update request against a view into an update of the underlying database. Finally, we obtained several complexity results concerning the existence of repairs.

Our paper advances the theory of view updating in three main ways. First, it proposes and studies the setting where extensional databases are indefinite both before and after an update. While *introducing* indefiniteness to narrow down the class of potential repairs was considered before [5], the assumption there was that the initial extensional database was complete. That assumption substantially limits the applicability of the earlier results. Second, our paper proposes a more expressive model of an indefinite extensional database. In our model databases are determined by two sets of facts. The first set of facts specifies what is true and provides an upper bound to what might still be unknown. By CWA, everything else is false. The second set of facts lists exceptions to the "unknown range," that is, facts that according to the first set might be unknown but are actually false (exceptions). Third, our paper introduces two novel classes or repairs, relevant and constrained, that often substantially narrow down possible ways to fulfill an update request against a view. Relevant repairs do not introduce any new constants and minimize change. Constrained repairs in addition do not involve constants that are in some precise sense "replaceable" and, thus, not grounded in the problem specification.

We already discussed some earlier work on view updating in the introduction as a backdrop to our approach. Expanding on that discussion, we note that the view-update problem is closely related to abduction and is often considered from that perspective. Perhaps the first explicit connection between the two was made by Bry [1] who proposed to use deductive tools as a means for implementing the type of abductive reasoning required in updating views. That idea was pursued by others with modifications that depended on the class and the semantics of the views. For instance, Kakas and Mancarella [9] exploited in their work on view updates the abductive framework by Eshghi and Kowalski [4] and were the first to consider the stable-model semantics for views. Neither of the two works mentioned above was, however, concerned with the case of updates to views over indefinite databases. Console et al. [2] studied the case in which *requests* can involve variables. These variables are replaced by null values and, in this way, null values eventually end up in repaired databases. However, once there, they loose their null value status and are treated just as any other constants. Consequently, no reasoning over null values takes place, in particular, they have no special effect on the notion of minimality. None of the papers discussed studied the complexity of the view-update problem. Instead, the focus was on tailoring resolution-based deductive reasoning tools to handle abduction. Some results on the complexity of abduction for logic programs were obtained by Eiter, Gottlob and Leone [3]. However, again the setting they considered did not assume incompleteness in extensional databases.

Our paper leaves several questions for future work. First, we considered restricted classes of views. That suggests the problem to extend our complexity results to the

full case of Horn programs and, later, stratified ones. Next, we considered a limited class of integrity constraints. Importantly, we disallowed tuple-generating constraints. However, once they are allowed, even a problem of repairing consistency in an extensional database becomes undecidable. A common solution in the database research is to impose syntactic restrictions on the constraints [8]. That suggests considering view-updating in the setting in which only restricted classes of constraints are allowed.

## Acknowledgments

The third author was supported by the NSF grant IIS-0913459. We are grateful to Sergio Greco and Leopoldo Bertossi for helpful discussions.

## References

1. François Bry. Intensional updates: Abduction via deduction. In *Proceedings of ICLP 1990*, pages 561–575, Cambridge, MA, 1990. MIT Press.
2. Luca Console, Maria Luisa Sapino, and Daniele Theseider Dupré. The role of abduction in database view updating. *J. Intell. Inf. Syst.*, 4(3):261–280, 1995.
3. Thomas Eiter, Georg Gottlob, and Nicola Leone. Abduction from logic programs: Semantics and complexity. *Theor. Comput. Sci.*, 189(1-2):129–177, 1997.
4. K. Eshghi and R. A. Kowalski. Abduction compared with negation by failure. In *Proceedings of ICLP 1989*, pages 234–254, Cambridge, MA, 1989. MIT Press.
5. Carles Farré, Ernest Teniente, and Toni Urpí. Handling existential derived predicates in view updating. In *Proceedings of ICLP 2003*, volume 2916 of *LNCS*, pages 148–162. Springer, 2003.
6. Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proceedings of ICLP/SLP 1988*, pages 1070–1080, 1988.
7. Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Comput.*, 9(3/4):365–386, 1991.
8. Sergio Greco, Francesca Spezzano, and Irina Trubitsyna. Stratification criteria and rewriting techniques for checking chase termination. *PVLDB*, 4(11):1158–1168, 2011.
9. A. C. Kakas and P. Mancarella. Database updates through abduction. In *Proceedings of the sixteenth international conference on Very large databases*, pages 650–661, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.
10. Leonid Libkin. A semantics-based approach to design of query languages for partial information. In *Semantics in Databases*, volume 1358 of *LNCS*, pages 170–208. Springer, 1995.
11. Enric Mayol and Ernest Teniente. Consistency preserving updates in deductive databases. *IEEE TDKE*, 47(1):61–103, 2003.
12. R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and data bases*, pages 55–76. Plenum Press, 1978.
13. Ernest Teniente and Antoni Olivé. Updating knowledge bases while maintaining their consistency. *VLDB J.*, 4(2):193–241, 1995.
14. Stephen Todd. Automatic constraint maintenance and updating defined relations. In *IFIP Congress*, pages 145–148, 1977.
15. Can Türker and Michael Gertz. Semantic integrity support in sql: 1999 and commercial (object-)relational database management systems. *VLDB J.*, 10(4):241–269, 2001.
16. Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume I*. Computer Science Press, 1988.