On Monotonic Deductive Database Updating Under the Open World Assumption

Laurent Dominique

On Monotonic Deductive Database Updating under the Open World Assumption

Dominique Laurent

ETIS Laboratory - ENSEA / UCP / CNRS, Cergy-Pontoise, France
dominique.laurent@u-cergy.fr

Abstract. In this paper, we present and discuss our preliminary work on a deductive database model in which insertions and deletions are associated with time stamps. Although time stamps have been used for many purposes in traditional approaches to databases, no approach did investigate their impact in a deductive framework under the so called *Open World Assumption* (OWA).
To do so, we consider Datalog databases with negation in the body of the rules and define the semantics of such databases using a three valued logics. Relying on our previous work on database updating, we show that updates in our approach are performed in a deterministic way and preserve database consistency with respect to the rules. Moreover, contrary to standard approaches, we argue that our model is *monotonic* in the sense that through time, updates refine the database semantics, while never overriding results from past semantics. We relate our approach to standard updating approaches from the literature and we discuss implementation issues based on the graph database model.

Key words: Open World Assumption . Datalog with negation . Database semantics . Deductive database updating . Temporal databases . Graph databases.

1 Introduction

In this paper, we present and discuss our preliminary work on a deductive database model in which insertions and deletions are associated with time stamps. We propose a novel approach meant to take into account the needs of many current applications, specifically in the domains of data integration and data warehousing. Our approach has the following characteristics:

1. As usual when dealing with Datalog databases ([12]), in our approach, a database D is a pair (E, R) where E (respectively R) is called the *extension* (respectively *set of rules*) of D. However, whereas in standard approaches, the extension E is a set of ground facts, in our approach the extension E contains ground facts along with negated ground facts, referred to as *negative facts*. The role of negative facts is explained in the next item. The rules in R are standard Datalog rules with negation, as explained in [10, 12], and applying the rules in R to the extension E produces a set of positive and

negative facts that is called the *semantics of* D. The specificities of the semantics considered in our approach is introduced below.

2. As in our previous work ([3, 26]), and contrary to standard approaches to database updating ([5, 30, 32]) where *only* insertions are stored in the form of positive facts, we allow the presence of *negative* facts in the database, in order to also store the *deletions*. Moreover, similarly to [3, 26], the database updating process as defined in this paper is *deterministic* and *consistent*, in the sense that all updates are indeed processed, in a deterministic way, and they preserve consistency with respect to the rules present in the database.
3. We associate every positive or negative fact involved in a given update with a *time stamp*, as done in some models of *temporal databases* ([6, 11, 24]). We assume that we are given an infinite and totally ordered set to which these time stamps belong. We do not make any further hypothesis on this set, in particular on whether it is countable or not. Time stamps allow us to keep track of *all* updates along with their processing time. In this context, we define a preordering with respect to which updates are *monotonic*. Intuitively, the monotonicity property expresses the fact that updates refine the global semantics of a database, and preserve the semantics as computed in the past.
4. The database semantics is defined so as to reflect the *Open World Assumption* (OWA), contrary to most database models that assume the Closed World Assumption (CWA) ([29]). We argue that considering the OWA instead of the CWA is relevant in most applications related with data integration on the Web, for the following intuitive reason: when a fact does not appear in the answer to a query, this does *not* mean that this fact is *false*, but rather that it has not been searched properly. Therefore, in the absence of any other information such a fact will be considered *unknown*, instead of false. We refer to [8] for a more detailed discussion on this issue.

We illustrate our approach through the following example that will serve as a running example throughout the paper.

Example 1. We consider three atoms a, b and c, and a set of rules R containing the only rule $c \leftarrow a, \neg b$. In this setting, we assume that the rule $c \leftarrow a, \neg b$ should hold with the following meaning: at any time t, if a and $\neg b$ are true, then c is true.

Notice that this way of handling rules reflects the OWA, in the sense that the rule applies only when there is an *explicit evidence that its body holds*. As opposed to this remark, CWA approaches would interpret the rule as follows: if a is true and if b *cannot be proven as true*, then c is true.

We now consider the following changes of the database through time:

Starting time t_0. Initially, say at time t_0, the database extension E is supposed to be empty, meaning that a, b and c are all *unknown*. This implies that the rule does not apply and thus that the database semantics is empty. It should be noticed that to do so, we must consider a three valued logics where a formula can be either true, false or unknown. Moreover, it should be clear that unknown facts are not stored.

Time t_1 strictly greater than t_0. If at a given time t_1, a is inserted then the database extension E contains the only pair (a, t_1), implying that a is now *true* but that b remains unknown. Since the rule does not apply when a is true and b is unknown, the semantics of the database at time t_1 contains the only fact a, meaning that b and c remain unknown.

Time t_2 strictly greater than t_1. If at time t_2, b is inserted, then the database extension E contains the two pairs (a, t_1) and (b, t_2). In this case, at time t_2, a clearly remains true whereas b becomes true. Since the rule does not apply when a and b are true, the semantics of the database at time t_2 is $\{a, b\}$.

Time t_3 strictly greater than t_2. Assume now that at time t_3, b is deleted from the database. This update is achieved in our approach by *inserting* $(\neg b, t_3)$ into the database, which implies that the new database extension E is

$$E = \{(a, t_1), (b, t_2), (\neg b, t_3)\}.$$

Since t_1 is the largest time stamp stored in the database for a, we consider that a is still true at time t_3. Since $\neg b$ becomes true at time t_3, this overrides the fact that b was previously true. However, it should be noticed that storing (b, t_2) in the database state allows to keep track that b was true between times t_2 and t_3. On the other hand, the fact that at time t_3, a is true and b is false allows to apply the rule, implying that c is true. Therefore, at time t_3, the semantics of the database is as follows: $\{a, \neg b, c\}$.

Time t_4 strictly greater than t_3. As a last update, let us now consider that at time t_4, c is deleted. As in the previous case, this is achieved by inserting $(\neg c, t_4)$ into the database extension E, which thus becomes

$$E = \{(a, t_1), (b, t_2), (\neg b, t_3), (\neg c, t_4)\}.$$

At this stage, it is important to notice that we do *not* consider that this last update brings a contradiction to the rule, but rather, we consider that c becomes an *exception* to the rule. We refer in this respect to our previous work on database updates in the context of the Well-Founded semantics ([3, 26]). We simply recall here that in this approach, updates are given priority over the rules, thus modeling exceptions to the rules. In the case of our example, this implies that the intuitive meaning of the rule $c \leftarrow a, \neg b$ provided earlier has to be amended as follows: at any time t, if a and $\neg b$ are true, *and* if $\neg c$ does not hold in the database state, then c is true. As a consequence, at time t_4 the semantics of the database is the following: $\{a, \neg b, \neg c\}$. □

The paper is organized as follows: In Section 2 we introduce the basic definitions regarding our database model and in Section 3, we present our approach to database updating and study its main properties. In Section 4, we relate our model with standard models from the literature and we sketch possible ways of implementing our approach in the context of graph databases [4, 22]. Section 5 concludes the paper and suggests research issues related to this work.

2 Basic Definitions

2.1 Background

As seen in the introductory section, our approach deals with Datalog databases with negation ([9, 10, 12, 17]), and therefore we use the standard associated terminology. We recall in this respect from [12] that a *literal* g is an expression of one of the two forms $p(t_1, \ldots, t_n)$ or $\neg p(t_1, \ldots, t_n)$ where p is an n-ary predicate and $t_1, \ldots t_n$ are constants or variables; in the first case g is said to be *positive* and in the second case g is said to be *negative*. Moreover, a literal g is said to be *ground* when no variable occurs among its arguments, and in this case if g is positive (respectively negative) it is called a *fact* or a *positive fact* (respectively a *negative fact*).

In order to formally take *time stamps* into account, we assume that we are given an infinite and totally ordered set to which these time stamps belong. We do not make any further hypothesis on this set, in particular on whether it is countable or not. Time stamps are denoted by t possibly with primes or indices.

As seen in Example 1, we consider that ground literals associated with time stamps are stored in a database. Such an association is denoted as a pair (g, t), called a *t-literal*, where g is a ground literal and t is a time stamp.

The following example illustrates the notation and terminology introduced above in the context of Example 1.

Example 2. We first note that for the sake of simplification, in Example 1, ground literals are denoted by constants a, b and c. This simplification should be understood as a short hand for three ground literals, say $p_a(\alpha)$, $p_b(\beta)$ and $p_c(\gamma)$ (standing respectively for a, b and c) where p_a, p_b and p_c are three unary predicates and α, β and γ are three constants.

Denoting by E the set of t-literals considered at the last step described in Example 1, we have $E = \{(a, t_1), (b, t_2), (\neg b, t_3), (\neg c, t_4)\}$, meaning intuitively that in E:

- From time t_1 on, a is true.
- Between times t_2 and t_3, b is true, whereas, after time t_3, b is false.
- From time t_4 on, c is false.

We note that the fact that both b and $\neg b$ occur in E is not contradictory, but represents a change in the truth value of b (going from true to false). Moreover, since $t_1 < t_2$, the content of E gives no indication about the truth value of b between times t_1 and t_2. In this case, b is *unknown* in E. □

In order to formalize the remarks in Example 2, we first introduce the notion of *t-validity* as follows.

Definition 1. *Let X be a set of t-literals. For every ground literal g and every time stamp t, g is said to be* t-valid *in X if there exists a time stamp t' such that*

- *$t' \leq t$ and (g, t') is in X and*
- *for every t'' such that $t' \leq t'' \leq t$, $(\neg g, t'')$ is not in X.*

The set of all literals t-valid in X is denoted by $V(X,t)$.

The set X is said to be consistent *if for every time stamp t and every fact f, $V(X,t)$ does not contain f and $\neg f$.*

Applying Definition 1 to Example 2, it can seen that the set E is consistent. Moreover, a is t-valid in E for every $t \geq t_1$, whereas b is t-valid in E of every t such that $t_2 \leq t < t_3$. It is also easy to see that, for every t such that $t_2 \leq t < t_3$, we have $V(E,t) = \{a, b\}$.

As seen in Example 1, t-literals provide a simple and intuitive way of modeling updates while taking into account their associated time stamps. However, as will be seen later, this simple way of dealing with time can not be used when considering database semantics in our approach.

This is so because, when computing the database semantics at different time stamps, a ground literal g can successively be unknown, then true and then unknown again (which is not possible when considering database updates). Unfortunately, as shown in the forthcoming Example 6, it turns out that t-literals do not allow to express the last change, *i.e.,* that g becomes unknown.

In order to cope with this difficulty, we consider pairs of the form $\langle\, g, [t_1, t_2)\,\rangle$ where g is a ground literal and $[t_1, t_2)$ stands for the set of time stamps t such that $t_1 \leq t < t_2$; moreover, we use the notation $[t_1, \infty)$ to mean that the interval has an infinite upper bound.

Calling such a pair an *int-literal*, we define the notion of t-validity in a set of int-literals in much the same way as for sets of t-literals (see Definition 1).

Definition 2. *Let Y be a set of int-literals and t a time stamp. A ground literal g is said to be* t-valid in Y *if there exists $\langle g, I\rangle$ in Y such that t belongs to I. The set of all literals t-valid in Y is denoted by $V(Y,t)$.*

The set Y is said to be consistent *if for every time stamp t and every fact f, $V(Y,t)$ does not contain f and $\neg f$.*

We illustrate the notion of int-literal through the following example.

Example 3. Let us consider the following set Y, where as in Example 1, a, b and c are atoms and t_1, t_2, t_3 are distinct time stamps such that $t_1 < t_2 < t_3$:

$$Y = \{\langle a, [t_1, \infty)\,\rangle, \langle \neg b, [t_2, t_3)\,\rangle, \langle c, [t_2, t_3)\,\rangle, \langle b, [t_3, \infty)\,\rangle\}.$$

For t such that $t_2 \leq t < t_3$, we have $V(Y,t) = \{a, \neg b, c\}$, because in this case, $t \in [t_1, \infty)$ and $t \in [t_2, t_3)$ hold. On the other hand, for t' such that $t_3 \leq t'$, we have $V(Y,t') = \{a, b\}$, because now, $t \in [t_1, \infty)$ and $t \in b, [t_3, \infty)$ hold. □

Relating sets of t-literals with sets of int-literals, we notice that every set X of t-literals can be associated with a set $int(X)$ of int-literals as follows:

$$\begin{aligned} int(X) = \{\langle\, g, [t_1, t_2)\,\rangle \mid\ & (g, t_1) \in X \wedge (\neg g, t_2) \in X \wedge \\ & (\forall t)(t_1 \leq t < t_2 \Rightarrow (\neg g, t) \notin X)\} \cup \\ \{\langle\, g, [t_1, \infty)\,\rangle \mid\ & (g, t_1) \in X \wedge (\forall t)(t_1 \leq t \Rightarrow (\neg g, t) \notin X)\}. \end{aligned}$$

To illustrate this relationship in the context of Example 2, consider again the set $E = \{(a, t_1), (b, t_2), (\neg b, t_3), (\neg c, t_4)\}$. It is then easy to see that we have:

$$int(E) = \{\langle\, a, [t_1, \infty)\,\rangle, \langle\, b, [t_2, t_3)\,\rangle, \langle\, \neg b, [t_3, \infty)\,\rangle, \langle\, \neg c, [t_4, \infty)\,\rangle\}.$$

The following proposition states that t-validity in a set of t-literals and t-validity in its associated set of int-literals coincide, thus justifying the fact that we use the same terminology regarding t-validity in the two kinds of sets.

Proposition 1. *For every set X of t-literals and every time stamp t, we have $V(X, t) = V(int(X), t)$.*

Proof. 1. By Definition 1, g is t-valid in X for a given time stamp t if and only if X contains a pair (g, t') such $t' \leq t$ and X contains no pair $(\neg g, t'')$ such that $t'' \leq t' \leq t$. According to the definition of $int(X)$, this implies that g is t-valid in X for a given time stamp t if and only if $int(X)$ contains a pair $\langle g, I\rangle$ where I is a time interval whose lower bound is t' and whose upper bound is greater than t (*i.e.,* infinite if $\neg g$ does not occur in X associated with a time stamp greater than t_1, or finite otherwise). Therefore, g is t-valid in X for a given time stamp t if and only if $t \in I$, which by Definition 2 means that g is t-valid in $int(X)$. Therefore the proof is complete.

We now emphasize that int-literals are strictly *more expressive* than t-literals, in the sense that there exist sets of int-literals that have *no* corresponding set of t-literals that preserves t-validity.

To see this, consider the set $Y = \{\langle\, g, [t_1, t_2)\,\rangle\}$, meaning that the ground literal g is t-valid for every t such that $t_1 \leq t < t_2$. Intuitively, this corresponds to the following: before t_1, g was unknown, between t_1 and t_2 g holds, and after t_2, g is again unknown. Now, if X is a set of t-literals such that, for every t, $V(X, t) = V(Y, t)$, then X must contain the t-pair (g, t_1) to state that g holds from t_1 on, but we can not express that neither g nor $\neg g$ hold from t_2 on.

As will be seen later on, int-literals are needed in our approach for defining the semantics of a database, whereas t-literals are used for defining the database extension, *i.e.,* the database content. We now introduce the following relation over sets of int-literals.

Definition 3. *Let Y_1 and Y_2 be two sets of int-literals. Y_1 is said to* refine *Y_2, denoted by $Y_1 \preceq Y_2$, if for every $\langle g_2, I_2\rangle$ in Y_2, there exists $\langle g_1, I_1\rangle$ in Y_1 such that $g_1 = g_2$ and $I_1 \subseteq I_2$.*

It is easy to see from Definition 3 that set inclusion implies refinement, in the sense that for all sets of int-literals Y_1 and Y_2, if $Y_2 \subseteq Y_1$ then $Y_1 \preceq Y_2$.

However, the converse does not hold because for $Y_1 = \{\langle\, g, [t_1, t_2)\,\rangle\}$ and $Y_2 = \{\langle\, g, [t_1, \infty)\,\rangle\}$ with $t_1 < t_2$, $Y_1 \preceq Y_2$ holds whereas Y_1 and Y_2 are not comparable with respect to set inclusion.

On the other hand, it is easy to see that the relation $\preceq$ is reflexive and transitive, implying that $\preceq$ is a pre-ordering. However this relation is not anti-symmetric and thus, not an ordering. The following example explains why the relation $\preceq$ is not anti-symmetric.

Example 4. Consider the set E of Example 2 and its associated set of int-literals $int(E) = \{\langle\, a, [t_1, \infty)\,\rangle, \langle\, b, [t_2, t_3)\,\rangle, \langle\, \neg b, [t_3, \infty)\,\rangle, \langle\, \neg c, [t_4, \infty)\,\rangle\}$ as given earlier.

Let $Y = int(E) \cup \{\langle\, \neg c, [t_5, t_6)\,\rangle\}$ where t_5 and t_6 are two time stamps such that $t_4 < t_5 < t_6$. In this case, according to Definition 3, $int(E)$ and Y are two distinct sets such that $int(E) \preceq Y$ and $Y \preceq int(E)$. Indeed:

- For every pair $\pi = \langle\, g, I\,\rangle$ in Y, either π belongs to $int(E)$ or $\pi = \langle\, \neg c, [t_5, t_6)\,\rangle$. In the first case, we trivially have a pair in $int(E)$ satisfying Definition 3 (namely π itself), and in the second case, $\langle\, \neg c, [t_4, \infty)\,\rangle$ satisfies Definition 3 because of the inclusion $[t_5, t_6) \subseteq [t_4, \infty)$. Thus, $int(E) \preceq Y$ holds.
- Conversely, $Y \preceq int(E)$ holds because $int(E) \subseteq Y$. □

The fact that the relation $\preceq$ is a pre-ordering but not an ordering raises the question of equivalent sets of int-literals, *i.e.,* sets Y_1 and Y_2 for which $Y_1 \preceq Y_2$ and $Y_2 \preceq Y_1$ hold. Example 4 suggests that such equivalent sets represent the same information in terms of t-validity. However, this point is left outside the scope of the present paper, and a complete study of this question is still needed.

2.2 Database and Database Semantics

As in standard approaches to Datalog databases with negation ([9, 10, 12, 17]), we consider that a database consists of an *extension* and a *set of rule*. However, in our approach, the extension is a set of t-literals (and not a set of facts) and the rules are standard Datalog rules with negation.

Definition 4. *A database D is a pair $D = (E, R)$ where E and R are respectively called the* extension *and the* rule set *of D. If $D = (E, R)$, then:*

- *E is a set of t-literals.*
- *R is a set of standard Datalog rules with negation, that is, rules of the form $r : h \leftarrow b_1, \ldots, b_n$ where*
  1. *for $i = 1, \ldots, n$, b_i is a literal (positive or negative) and the set of all b_i's ($i = 1, \ldots, n$) is called the* body *of the rule, denoted by $body(r)$,*
  2. *h is a positive literal, called the* head *of the rule, denoted by $head(r)$,*
  3. *all variables occurring in h are assumed to also occur in the body of the rule (*i.e., *rules are safe).*

Given a database $D = (E, R)$, the set $V(E, t)$ is called the state of D at time t *and is denoted by D_t. The database D is said to be* consistent *if for every time stamp t, D_t is consistent.*

As usual, the extension E of a given database $D = (E, R)$ as mentioned in Definition 4 is meant to contain the facts currently stored in the database as a result of the updates that have been processed so far.

Regarding database semantics, the presence of time stamps in the database allows for considering the database semantics at different points of time. To do so, given a database $D = (E, R)$, we associate D with the so-called *membership immediate consequence operator* ([10]) that we adapt so as to take into account

the presence in E of (i) *negative* facts and of (ii) time stamps. We address item (i) based on our previous work on updates ([3, 26]), whereas item (ii) is the subject of the remainder of the present section.

In the definition given next, we consider valuations of the variables occurring in rules, that is mappings associating every variable occurring in the rules with a constant. To this end, we use the following notation: if r is a rule in R and $inst$ an instantiation, then $inst(head(r))$ is the instantiation of the literal $head(r)$ and $inst(body(r))$ denotes the set of instantiations of the literals in $body(r)$.

Definition 5. *Let $D = (E, R)$ be a database and t a time stamp. The* membership immediate consequence operator associated to D, *denoted by $T_D^{\in}$ is a mapping associating every set X of t-literals with the following set:*

$$T_D^{\in}(X,t) = D_t \cup \{h \mid (\exists r \in R)(\ h = inst(head(r)) \wedge \\ inst(body(r)) \subseteq V(X,t) \wedge \neg h \notin D_t)\}.$$

It is easy to see that for every fixed time stamp t the membership immediate consequence operator $T_D^{\in}(\_, t)$ as defined above is monotonic and continuous. As a consequence this operator has a unique least fixed point obtained as the limit of the sequence $\left(T^k\right)_{k\geq 0}$ defined as follows:

- $T^0 = T_D^{\in}(\emptyset, t)$
- for every $k > 0$, $T^k = T_D^{\in}(T^{k-1}, t)$.

This is precisely this least fixed point that we call the *semantics of D at time t*, which is denoted by $sem_t(D)$.

Example 5. In the context of Example 1, and according to Definition 4, we denote by $D = (E, R)$ the database obtained after the last given update. Thus, as seen in Example 2, $E = \{(a, t_1), (b, t_2), (\neg b, t_3), (\neg c, t_4)\}$ and $R = \{c \leftarrow a, \neg b\}$.

We note that, for the sake of simplification, we consider here the simple case where no variables occur in the rules. As a consequence, each rule is equal to its unique instantiation. Based on Definition 5, we now illustrate the computations of $sem_t(D)$ for $t = t_1, \ldots, t_4$.

Since $D_{t_1} = \{a\}$, we have in this case: $T^0 = T^1 = \{a\}$ because the rule of R does not apply. Therefore, $sem_{t_1}(D) = \{a\}$. For similar reasons, we have $sem_{t_2}(D) = \{a, b\}$ because $D_{t_2} = \{a, b\}$, which again prevents the rule of R from applying. Now, the computation of $sem_{t_3}(D)$ is as follows:

1. We have $D_{t_3} = \{a, \neg b\}$, and thus $T^0 = \{a, \neg b\}$.
2. Using the rule in R, we obtain $T^1 = \{a, \neg b\} \cup \{c\} = \{a, \neg b, c\}$, because $\neg c$ is not in D_{t_3}.
3. As no further ground literal is generated when computing T^3, we obtain that $sem_{t_3}(D) = \{a, \neg b, c\}$.

On the other hand, the computation of $sem_{t_4}(D)$ is as follows:

1. We have $D_{t_4} = \{a, \neg b, \neg c\}$, and thus $T^0 = \{a, \neg b, \neg c\}$.

2. Then, since $\neg c$ is in D_{t_4}, the rule in R does not generate c in T^1. Thus, we obtain $T^1 = T^0$.

Therefore, we have $sem_{t_4}(D) = \{a, \neg b, \neg c\}$. □

The following proposition states that, for every time stamp t occurring in a consistent database D, the semantics of D at time t contains the extension of D and that this semantics is consistent.

Proposition 2. *Let $D = (E, R)$ be a consistent database. For every time stamp t occurring in E:*

1. *$D_t \subseteq sem_t(D)$.*
2. *The set $sem_t(D)$ is a consistent set of ground literals.*

Proof. 1. By definition of $T_D^{\in}$, for every set X of t-literals, the set D_t is a subset of $T_D^{\in}(X)$. Therefore, for every $k \geq 0$, D_t is a subset of T^k, which entails that D_t is a subset of $sem_t(D)$.
2. Since D is assumed to be consistent, for every time stamp t occurring in E, D_t is a consistent, *i.e.,* D_t does not contain a fact f along with its negation $\neg f$. On the other hand, by Definition 4, as R contains only rules whose head is a positive literal, literals that belong to $sem_t(D)$ but not to D_t are positive facts. Thus, assuming that $sem_t(D)$ is not consistent implies that there exists a fact f in $sem_t(D) \setminus D_t$ such that $\neg f$ belongs to D_t. Since this is not possible because of the definition of $T_D^{\in}$ in Definition 5, the proof is complete.

An important remark regarding OWA and database semantics as defined above is now in order. We first recall that defining semantics for Datalog databases with negation had been the subject of important research efforts in the past (see [9] for a survey of this topic). Among these semantics, we cite the Kripke-Kleene semantics as defined in [14] and the Well-Founded semantics introduced in [17]. We focus on these two semantics because they can easily be adapted to our context in much the same way as done above for the $T_D^{\in}$ operator (we refer to our previous work in [3, 26] regarding the case of the Well-Founded semantics).

Our choice of defining database semantics using the membership immediate consequence operator is motivated by our assumption that working under the OWA is preferable to working under the CWA. Indeed, in our approach, no negative fact is obtained by the semantic operator, since no rule can explicitly generate a negative fact. This means that, contrary to CWA, we make no particular explicit or implicit assumption regarding negative facts. On the other hand, the approaches in [14] and in [17] work under CWA because:

- In [14], when computing the considered operator for a given set X of ground literals, a negative fact $\neg f$ is obtained when every instantiated rule whose head is f has a body containing a contradiction with respect to X. Consequently, when f is the head of no instantiated rule, then $\neg f$ is deduced, which means that CWA is assumed.

– In [17], when computing the greatest set of unfounded facts for a given set X of ground literals, a negative fact $\neg f$ is obtained when, for every instantiated rule whose head is f, the body contains either a contradiction with respect to X, or a fact already found as being unfounded. As above, this implies that $\neg f$ is deduced when f is the head of no instantiated rule, which again means that CWA is assumed.

However, it should be noticed that, under the hypothesis that CWA is preferable to OWA, choosing one of the two semantics mentioned above would not basically change the way our approach is constructed; only the computations of the database semantics $sem_t(D)$ would change, but all theoretical results based on the output of these computations would still hold.

Now, given a database D, and based on the semantics defined at different time stamps occurring in D, we address the issue of defining the *global semantics* of D. However, the following example shows that defining this global semantics as the union of all semantics at all time stamps occurring in D is not correct.

Example 6. In the context of Example 1, let us now consider the database $D' = (E', R)$ where $E' = \{(a, t_1), (\neg b, t_2), (b, t_3)\}$ (where $t_1 < t_2 < t_3$) and R contains the single rule $c \leftarrow a, \neg b$. Computations similar to those in Example 5 yield the following: $sem_{t_1}(D') = \{a\}$, $sem_{t_2}(D') = \{a, \neg b, c\}$ and $sem_{t_3}(D') = \{a, b\}$.

Consequently, the global semantics of D' should be the set S of all t-literals that can be built up using the three sets above, namely:

$$S = \{(a, t_1), (a, t_2), (\neg b, t_2), (c, t_2), (a, t_3), (b, t_3)\}.$$

However, we argue that S is not the appropriate set to represent the global semantics of D' for the following two reasons:

1. The pairs (a, t_2) and (a, t_3) are redundant because removing these two pairs from S does not change the fact that a is t-valid for every t such that $t_1 \leq t$.
2. More importantly, S is *not* correct regarding the t-validity of c. Indeed, considering the semantics $sem_{t_2}(D')$ et $sem_{t_3}(D')$, c is t-valid for $t_2 \leq t < t_3$, whereas for $t \geq t_3$, c is no longer t-valid. On the other hand in S, c is clearly t-valid for every t such that $t \geq t_2$, thus for $t \geq t_3$. □

Referring back to our previous discussion about the expressiveness of t-literals with respect to that of int-literals, Example 6 shows that the global semantics can *not* be expressed using t-literals. This is so because in the semantics, ground literals may become unknown after being true or false, whereas, as will be seen in the next section, this is not possible when dealing with the database extension.

In order to cope with the difficulty raised in Example 6, the *global semantics* of a given database $D = (E, R)$ is defined below using int-literals. In this definition, as well as in the remainder of this paper, we assume that the first time stamp related to D is t_0 and that, at time t_0, E is equal to the empty set.

Definition 6. *Let $D = (E, R)$ be a database and $t_1, \ldots, t_n$ all time stamps occurring in E such that $t_1 < \ldots < t_n$. The* global semantics *of D, denoted by $SEM(D)$, is the set of all int-literals $\langle g, I \rangle$ satisfying one of the following two items:*

– $I = [t_i, t_j)$, *where*
  1. $i, j \in \{1, \ldots, n\}$, $i < j$, *and*
  2. $g \notin sem_{t_{i-1}}(D)$, $g \notin sem_{t_j}(D)$, *and*
  3. $(\forall k \in \{1, \ldots, n\})(i \leq k < j \Rightarrow g \in sem_{t_k}(D))$;
– $I = [t_i, \infty)$, *where*
  1. $i \in \{1, \ldots, n\}$, *and*
  2. $g \notin sem_{t_{i-1}}(D)$, *and*
  3. $(\forall k \in \{1, \ldots, n\})(i \leq k \Rightarrow g \in sem_{t_k}(D))$.

Applying Definition 6 to the database D' of Example 6 yields the following global semantics:

$$SEM(D') = \{\langle\, a, [t_1, \infty)\,\rangle, \langle\, \neg b, [t_2, t_3)\,\rangle, \langle\, c, [t_2, t_3)\,\rangle, \langle\, b, [t_3, \infty)\,\rangle\}.$$

It can be seen that $SEM(D')$ correctly represents the information conveyed by the sets $sem_{t_i}(D')$ $(i = 1, 2, 3)$ in the sense that for every ground literal g and every time stamp t_i $(i = 1, 2, 3)$, g is t_i-valid in $SEM(D')$ if and only if g is in $sem_{t_i}(D')$. The following proposition shows that this property holds in general, for any time stamp t.

Proposition 3. *Let $D = (E, R)$ be a database. For every ground literal g and every time stamp t, g is t-valid in $SEM(D)$ if and only if g is in $sem_t(D)$.*

Proof. Using the same notation as in Definition 6, let g be a ground literal such that g belongs to $sem_t(D)$. Denoting by k the least index such that $t_k \leq t$ and g is in $sem_{t_p}(D)$ for $p \geq k$ and $t_p \leq t$, Definition 6 implies that $SEM(D)$ contains an int-literal $\langle g, I\rangle$ such that $I = [t_k, t_j)$ with $t_k \leq t < t_j$, or $I = [t_k, \infty)$. In both cases, by Definition 2, we have that g is t-valid in $SEM(D)$.

Conversely, let us assume that g is t-valid in $SEM(D)$. In this case, by Definition 2, $SEM(D)$ contains an int-literal $\langle g, I\rangle$ such that t is in I. By Definition 6, I is either $[t_i, t_j)$ or $[t_i, \infty)$ and g belongs to $sem_{t_i}(D)$. Let k be in $\{1, \ldots, n-1\}$ such that $[t_k, t_{k+1})$ or $[t_k, \infty)$ is the least interval I_0 included in I and containing t. It is easy to see that such a k always exists and is unique. Moreover, we have the following:
(i) Applying again Definition 6, g belongs to $sem_{t_k}(D)$.
(ii) I_0 contains no time stamp from $\{t_1, \ldots, t_n\}$ other than t_k, implying that for every time stamp q in I_0, $sem_q(D) = sem_{t_k}(D)$.
Hence, $sem_t(D) = sem_{t_k}(D)$, which implies that g is in $sem_t(D)$. Therefore, the proof is complete.

3 Updates

In this section, we define the two standard update operations insert and delete in our model, and then, we study their basic properties.

Definition 7. *Let $D = (E, R)$ be a database and t_c a time stamp strictly greater than any time stamp occurring in E (t_c can be referred to as the current time stamp). For every fact f*

- *the insertion of f in D results in the database denoted by $ins(f, t_c, D) = (E^f_{t_c}, R)$, where $E^f_{t_c} = E \cup \{(f, t_c)\}$;*
- *the deletion of f from D results in the database denoted by $del(f, t_c, D) = (E^{\neg f}_{t_c}, R)$, where $E^{\neg f}_{t_c} = E \cup \{(\neg f, t_c)\}$.*

In the literature ([24]), time stamps stored in temporal databases can be of two kinds, namely *processing time* or *validity time*. While processing time refers to the time when the update has been performed in the system, validity time refers to the time the update should be taken in to account in the database semantics. Many examples can be found in the literature, this topic lying beyond the scope of this paper, we refer to [24] in this respect.

Although our approach can deal with any of these two kinds of time stamps, the fact that in Definition 7 the stored time stamps refer to the current time means that processing time is considered. We note however that considering validity time instead of processing time does not raise any particular difficulty. Moreover, dealing with the two kinds of time stamps in our model should be possible but we do not investigate further this issue in this paper.

On the other hand, as stated by Definition 7, in our approach as well as in our previous work [3, 26], updates are *insert-only* operations, even in the case of deletion. We note that keeping track of deletions is not new, since this is common practice in DBMSs and in data warehouse systems. However, the impact of deletions on database semantics has never been addressed as we do in our approach.

We now state the main properties of our updating approach. As an immediate consequence of Definition 7 and Proposition 2, the proposition below states that updates are always valid and preserve database consistency.

Proposition 4. *For every consistent database $D = (E, R)$, every time stamp t_c strictly greater than any time stamp occurring in E, and every fact f:*

1. *$ins(f, t_c, D)$ and $del(f, t_c, D)$ are consistent.*
2. *Moreover, the following holds:*
   - *$f \in sem_{t_c}(ins(f, t_c, D))$, and*
   - *$\neg f \in sem_{t_c}(del(f, t_c, D))$.*

It is important to notice that Proposition 4(1) implies that, in our approach all databases are consistent. Indeed, every database is obtained through updates, starting from the empty database which is trivially consistent. Therefore in the remainder of this paper, we always refer to *consistent* databases, even when the word 'consistent' is omitted.

Notice however in this respect that Proposition 4(1) holds because we consider that only *one* update at a time is possible. It is easy to extend Definition 7 so as to consider a *set* of insertions and deletions, all associated with the same time stamp. In that case however, database consistency is ensured if updates in this set are 'globally consistent', meaning that no fact is inserted and deleted at the same time.

On the other hand, Proposition 4(2) shows that updates are *always* performed, in the sense that an inserted fact becomes true and a deleted fact becomes false in the updated database. We emphasize that traditional approaches to database updating fail to satisfy this property, in particular in the case of deletion.

Next, we illustrate this important feature of our approach in the context of Example 1.

Example 7. We recall that in Example 1 the only rule in R is $c \leftarrow a, \neg b$. Thus, when considering $D = (E, R)$ where $E = \{(a, t_1), (b, t_2), (\neg b, t_3)\}$, the deletion of c from D at time t_4 is problematic in traditional approaches, as explained below:

- As c occurs in the head of a rule, it could be considered as an *intentional* fact on which updates are not allowed. In this case, which is that of traditional approaches to deductive databases [12], the deletion is simply rejected.
- Assuming that the deletion is not rejected, it should be noticed that c has *never* been inserted. Consequently, approaches to updates that define a deletion as a removal from the extension would leave the database unchanged, thus making the deletion impossible to process.
- Another option (as in [5]) consists in modifying the database extension so as to prevent from triggering the rule. In our example, this would lead to two possible updates: either delete a or insert b. This is a typical case of non determinism that traditional approaches fail to take into account in general.

□

We now turn to the monotonicity properties of our approach. The following proposition states in this respect that past semantics can be safely recovered from any updated database.

Proposition 5. *For every database $D = (E, R)$, every fact f and every time stamp t such that $t < t_c$ (where t_c stands for any time stamp strictly greater than any time stamp occurring in E), we have:*

$$sem_t(D) = sem_t(ins(f, D, t_c)) = sem_t(del(f, D, t_c)).$$

Proof. Since t_c is assumed to be strictly greater than any other time stamp t occurring in D, the states at time t of D, $ins(f, D, t_c)$ and $del(f, D, t_c)$ are equal, in other words, $D_t = (ins(f, D, t_c))_t = (del(f, D, t_c))_t$. This implies that the corresponding semantics at t are the same, and thus the proof is complete.

We illustrate Proposition 5 in the context of Example 1 as follows.

Example 8. Let $D'' = (E'', R)$ be the database such that $E'' = \{(a, t_1), (b, t_2)\}$ and $R = \{c \leftarrow a, \neg b\}$. Then, it is easy to see that $D''_{t_2} = (del(b, D'', t_3))_{t_2} = (del(c, del(b, D'', t_3), t_4))_{t_2} = E''$. Thus:

$$sem_{t_2}(D'') = sem_{t_2}(del(b, D'', t_3)) = sem_{t_2}(del(c, del(b, D'', t_3), t_4)) = \{a, b\}.$$

□

Proposition 5 shows that the database semantics at a given time t_1 is preserved in any forthcoming state at time t_2 ($t_2 > t_1$) obtained through updates. This means intuitively that past semantics is preserved, while more and more such past semantics become available through time.

As a second result regarding monotonicity, the following proposition states that our approach to updating is monotonic with respect to the pre-ordering $\preceq$, meaning intuitively that updates always refine database semantics.

Proposition 6. *For every database $D = (E, R)$, every fact f and every time stamp t_c strictly greater than all time stamps occurring in E, we have:*

$$SEM(ins(f, D, t_c)) \preceq SEM(D) \quad \textit{and} \quad SEM(del(f, D, t_c)) \preceq SEM(D).$$

Proof. In this proof, we again assume that the time stamps occurring in D are $t_1, \ldots, t_n$ suct that $t_1 < \ldots < t_n$. Therefore, for every int-literal $\langle g, I\rangle$ in $SEM(D)$, either I is of the form $[t_i, t_j)$ with $t_i < t_n$ and $t_j \leq t_n$, or I is of the form $[t_i, \infty)$ with $t_i \leq t_n$. Moreover, the time stamps occurring in either of the databases $ins(f, D, t)$ and $del(f, D, t)$ are such that $t_1 < \ldots < t_n < t_c$.

Now, let $\langle g, I\rangle$ be an int-literal in $SEM(D)$. Recalling from Proposition 5 that for every time stamp t such that $t < t_c$, we have $sem_t(D) = sem_t(ins(f, D, t_c)) = sem_t(del(f, D, t_c))$, we consider the following two cases, depending on the form of the interval I:

1. If I is $[t_i, t_j)$, then $\langle g, I\rangle$ is in $SEM(ins(f, D, t_c))$ and in $SEM(del(f, D, t_c))$, because in this case, for every t in I we have $t < t_n < t_c$.
2. If I is $[t_i, \infty)$, then we distinguish the two cases whereby g is or not in $sem_{t_c}(ins(f, D, t_c))$ or $sem_{t_c}(del(f, D, t_c))$. If g is in, then $\langle g, I\rangle$ is unchanged in $SEM(ins(f, D, t_n))$ or in $SEM(del(f, D, t_c))$.
   If g is not in $sem_{t_c}(ins(f, D, t_c))$ or $sem_{t_c}(del(f, D, t_c))$, then in the global semantics of the updated database, $\langle g, [t_i, \infty)\rangle$ is changed into $\langle g, [t_i, t_c)\rangle$ and again two cases occur:
   - If $\neg g$ is in $sem_{t_c}(ins(f, D, t_c))$ or $sem_{t_c}(del(f, D, t_c))$ then $\langle \neg g, [t_c, \infty)\rangle$ appears in the global semantics of the updated database.
   - If $\neg g$ is not in $sem_{t_c}(ins(f, D, t_c))$ or $sem_{t_c}(del(f, D, t_c))$ then no new int-literal appears in the global semantics of the updated database.

Therefore, in any case, assuming that $\langle g, I\rangle$ is an int-literal in $SEM(D)$ implies that $SEM(ins(f, D, t_c))$ and $SEM(del(f, D, t_c))$ contain an int-literal $\langle g, I'\rangle$ such that $I' \subseteq I$. As a conclusion, by Definition 3 we obtain $SEM(ins(f, D, t_c)) \preceq SEM(D)$ and $SEM(del(f, D, t_c)) \preceq SEM(D)$ and thus, the proof is complete.

The following example illustrates Proposition 6 in the context of Example 1.

Example 9. As in Example 8, let us consider the database $D'' = (E'', R)$ where $E'' = \{(a, t_1), (b, t_2)\}$ and $R = \{c \leftarrow a, \neg b\}$, along with the deletion of b from D'' at time t_3. In this case, it can be seen that we have:

$$SEM(D'') = \{\langle a, [t_1, \infty)\rangle, \langle b, [t_2, \infty)\rangle\} \text{ and}$$
$$SEM(del(b, D'', t_3)) = \{\langle a, [t_1, \infty)\rangle, \langle b, [t_2, t_3)\rangle, \langle \neg b, [t_3, \infty)\rangle, \langle c, [t_3, \infty)\rangle\}.$$

Therefore, by Definition 3, we indeed have $SEM(del(b, D'', t_3)) \preceq SEM(D'')$. □

To conclude this section, we emphasize that Proposition 5 and Proposition 6 can be summarized as follows: *updating a database D refines its global semantics while preserving its past semantics at any time before this update.*

4 Discussion

In this section, we relate our approach to earlier work and then, we sketch the issue of implementation in the context of graph databases.

4.1 Comparison with Related Work

As noticed earlier, our approach heavily relies on previous work in various research domains, namely temporal databases, database semantics and database updating. We thus comment further how our approach relates to previous work in these domains.

Temporal databases have been the subject of many research efforts during the last three decades, and providing a survey of this important work is beyond the scope of the present paper. We simply mention here three broad research areas related to our present work: (*i*) temporal logics ([11, 16]), (*ii*) deductive temporal databases ([6]), and (*iii*) relational temporal databases ([24]).

Clearly, our approach falls in the second area mentioned above, where time labels are associated to formulas, as in [16]. In this context, we even consider the simplest case where labels cannot be combined and where rules do not involve time.

Our approach is quite different than those dealing with temporal deductive databases [6, 11] in which logical temporal operators are defined and used for expressing temporal queries. This is so because our goal is not to define a new temporal database model, but rather to define a framework assuming OWA, in which updating is monotonic. However, it is important to notice that the issue of non basic temporal queries in our approach should be investigated in the context of OLAP queries ([13]).

Another important remark regarding our way of dealing with time is that point wise time stamps are not expressive enough for defining the database semantics. This point is not new, but an illustration of the following basic results known for many years: the algebra for time intervals proposed in [2] has strong expressiveness properties, at the cost of being undecidable (as shown in [20]), whereas dealing with time through point wise time stamps is decidable (as shown in [16]). We also mention that this issue has been the subject of more recent work in [31], in the context of relational temporal databases. Relating the work in [31] regarding time granularity with our approach is a non trivial open issue that should be investigated.

As for temporal databases, our goal here is not to survey all approaches to database updating that have been published during the last four decades.

Instead, the very basic point that we would like to stress is that, contrary to standard logics, all update approaches proposed so far are non-monotonic, in the sense that updating a database may invalidate previous knowledge (whereas in standard logics introducing new hypotheses does not invalidate theorems). It is commonly argued that this property is a consequence of CWA, which, as mentioned earlier, is not suitable for many current applications. We refer to [8] for more detailed motivations on why considering OWA.

It is also important to recall that our approach is based on a three-valued logics thus allowing for considering *unknown* facts, additionally to true and false facts. This framework was also considered in previous work on database semantics assuming CWA (see [9, 10]), and the two semantics defined respectively in [14] (Kripke-Kleen semantics) and in [17] (Well-Founded semantics) are among the most popular. Considering OWA implies that we consider here the simpler operator known as *membership immediate consequence operator*.

However, we recall that it is possible to consider in our approach any of the two standard CWA semantics as defined in [14] and in [17], and that, in either of these two cases, Proposition 5 and Proposition 6 still hold. This means that monotonicity is not a consequence of the choice of the database semantics, and thus, monotonicity is not a consequence of choosing OWA rather than CWA. Instead, this means that monotonicity is a consequence of keeping track of all updates along with their associated time stamps. We think that more work is needed for further investigating this important point.

Additionally to the issues mentioned above regarding temporal OLAP queries and monotonicity, the following extensions are worth investigating:

- As we consider a three-valued logics, new types of updates are possible, namely updates that would allow a fact to become unknown after being true or false. Notice that this is not possible according to Definition 7, although this can happen for facts that are deduced by the rules (remember the case of fact c in Example 6). Such new types of updates should be carefully investigated because their intuitive semantics and their impact on the semantics are not clear (at least to the author of this paper).
- Going one step further, considering a database model in which inconsistencies are possible is an issue that has been the subject of many research efforts (see for instance [18, 28]) and, as argued in [8], consistency is an important but open issue under the OWA. We think that tackling this problem using the four-valued logics introduced in [7] offers promising perspectives.
- Another relevant issue is to extend our approach so as to take into account constraints such as functional dependencies. Notice that updating in the presence of constraints has been the subject of many research papers, among which we cite [3, 32] in the context of deductive databases, and our previous work in [25] in the context of relational databases.
- The last issue that we would like to mention is related to the exceptions to the rules, inspired by the work in [21, 27] in the context of association rule mining ([1]). In this context, the goal is to generate rules that are 'almost' satisfied by the underlying data set, in the sense that the quality measure

of confidence allows to keep the number of exceptions to the rules below a given threshold. It is shown in [27] that association rule mining can be adapted to the case of mining Datalog rules with negation whose number of exceptions is also kept below a given threshold. It is clear that this work also applies to our approach, thus allowing to generate new rules when the currently existing rules have too many exceptions due to deletions.

4.2 Possible Implementation using Graph Databases

In addition to the theoretical issues listed above, one important work to achieve is implementing our approach. Although this could be done using traditional frameworks dealing with temporal deductive databases (see for instance in [24]), we think that, in our context, it is more appropriate to consider novel data models, and more specifically the graph database model [4, 22]. This is so because all novel data models recently introduced, known as NoSQL, have been designed to better scale up given the data size in many current applications, and also to better handle the flexibility of the schema of the data.

As a first possible environment for implementing our approach, we cite RDF[1], a standard model in which the stored triples are seen as two vertices and one edge linking them in a graph. The work in [19] investigates a temporal extension of this model, in which every RDF triple is associated with a time stamp. As it seems that our approach can be easily 'embedded' in that of [19], we are planning to shortly investigate this issue. As another interesting work related to ours, in [15] the authors propose a global approach to updating an RDF knowledge base, seen as a Datalog program. Therefore, combining the approaches in [15] and in [19] with our work seems to be a very promising research direction.

On the other hand, in a more general graph database model, data are represented in terms of vertices and edges, not always stored as triples as in the case of RDF. Consequently, implementing our model in such a framework means that every vertex and every edge stored in the database is associated with a set of pairs of the form (t, upd) where t is a time stamp and upd is a value representing the type of the corresponding update, *i.e.,* either an insertion or a deletion. Considering a multi-relational graph data model, as for example the one of Neo4j (see `http://neo4j.com`), these pairs could simply be attributes or properties associated to the vertices and to the edges. We notice that, in such a setting, querying the database according to time stamp values requires to visit the whole graph, which is costly. An intuitively appealing example of such query is to retrieve the latest update processed against the database.

Another interesting option would be to consider time stamps as vertices and store an update as an edge connecting its time stamp to the 'object' it involves. Notice that in this case, the query mentioned above can be efficiently answered because it simply requires to retrieve the vertex representing the largest time stamp in the database, and from this vertex to go through its associated link

[1] RDF stands for 'Resource Description Framework' and is a W3C Recommendation, see `https://www.w3.org/standards/techs/rdf#w3c_all`

leading to the 'object' involved in the update. However, although this works when the 'object' is a vertex, this is not the case when the 'object' is an edge, because in a graph, an edge can not be connected by another edge to a vertex.

To cope with this difficulty, an extended graph database model dealing with *hyper graphs* is required, because such a model allows to connect as many vertices as needed through *hyper edges* that are defined as sets of vertices. For example, if v_1 and v_2 are vertices representing respectively an employee and a department, inserting that at time t employee v_1 becomes a member of department v_2 is performed by storing the edge $\{t, v_1, v_2\}$, associated with a label indicating that the update is an insertion. Since HyperGraphDB [23] (see `http://hypergraphdb.org`) handles hyper graphs, implementing our approach using this software will be investigated in the next future.

5 Concluding Remarks

The work presented in this paper results in a monotonic approach to database updating under the Open World Assumption (OWA), combining well known previous work on temporal database and on deductive database. In our approach, database semantics is defined in a three valued logics, in order to take into account that OWA was preferred over CWA. We recall in this respect that OWA was preferred in order to take into account the specificities of most current applications involving social networks, data mining or data warehousing. We also emphasize again that, in our approach, all updates are performed in a deterministic way and preserve database consistency with respect to the rules in the database, because rules can have exceptions.

Another important property of our approach is monotonicity of update operations, in the sense that updates refine database semantics while preserving the past semantics (*i.e.,* the semantics of any past database state can be recovered even after an arbitrary number of updates). We also argued that this result is basically a consequence of the fact that we store updates associated with a time stamp to keep track of the history of the updates.

As this paper reports on preliminary work, many issues remain open and need to be investigated in the future. We recap below all issues that have been listed earlier in the paper:

- Implementation: It has been suggested just above that this very important issue should be addressed in graph database models, and more specifically involving RDF triples or hyper graphs.
- Extend the rules by incorporating time variables, and similarly, take constraints into account, possibly involving time.
- Define possible extensions regarding new kinds of updates (that would involve several facts at a time and/or that would allow for a fact to become unknown after being true or false) or new semantics in a four valued logics (that would allow to consider inconsistencies in the database).
- Study new OLAP queries involving time stamps in the context of our approach.

– Apply data mining techniques in order to generate rules that would take updates into account (in the sense that the number of exceptions to the rules is minimized).

Acknowledgement

The author wishes to thank the anonymous referees whose comments and suggestions helped improve a preliminary version of the paper.

References

1. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A.I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 309–328. AAAI-MIT Press, 1996.
2. James F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983.
3. Mirian Halfeld Ferrari Alves, Dominique Laurent, and Nicolas Spyratos. Update rules in datalog programs. *J. Log. Comput.*, 8(6):745–775, 1998.
4. Renzo Angles and Claudio Gutierrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1):1:1–1:39, 2008.
5. Paolo Atzeni and Riccardo Torlone. Updating intensional predicates in datalog. *Data Knowl. Eng.*, 8:1–17, 1992.
6. Marianne Baudinet, Jan Chomicki, and Pierre Wolper. Temporal deductive databases. In *Temporal Databases*, pages 294–320. Benjamin/Cummings, 1993.
7. Nuel D. Belnap. A useful four-valued logic. In J. M. Dunn and G. Epstein, editors, *Modern Uses of Multiple-Valued Logic*. D. Reidel, 1977.
8. Mike Bergman. The Open World Assumption: Elephant in the room. In *AI3:::Adaptative Information*, pages 1–11. Available at: `www.mkbergman.com/852/the-open-world-assumption-elephant-in-the-room/`, 2009.
9. Nicole Bidoit. Negation in rule-based database languages: A survey. *Theor. Comput. Sci.*, 78(1):3–83, 1991.
10. Nicole Bidoit and Christine Froidevaux. Negation by default and unstratifiable logic programs. *Theor. Comput. Sci.*, 78(1):86–112, 1991.
11. Nicole Bidoit and Matthieu Objois. Temporal query languages expressive power: μTL versus T-WHILE. In *12th International Symposium on Temporal Representation and Reasoning (TIME 2005)*, pages 74–82. IEEE Computer Society, 2005.
12. Stefano Ceri, Georg Gottlob, and Letizia Tanca. *Logic Programming and Databases*. Springer, 1990.
13. Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and OLAP technology. *SIGMOD Rec.*, 26(1):65–74, March 1997.
14. Melvin Fitting. A Kripke-Kleene semantics for logic programs. *J. Log. Program.*, 2(4):295–312, 1985.
15. Giorgos Flouris, George Konstantinidis, Grigoris Antoniou, and Vassilis Christophides. Formal foundations for RDF/S KB evolution. *Knowl. Inf. Syst.*, 35(1):153–191, 2013.
16. Dov M. Gabbay. Introduction to labelled deductive systems. In Dov M. Gabbay and Franz Guenthner, editors, *Handbook of Philosophical Logic: Volume 17*, chapter 3, pages 179–266. Springer, 2013.

17. Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf. The well-founded semantics for general logic programs. *J. ACM*, 38(3):620–650, 1991.
18. Gianluigi Greco, Sergio Greco, and Ester Zumpano. A logical framework for querying and repairing inconsistent databases. *IEEE Trans. Knowl. Data Eng.*, 15(6):1389–1408, 2003.
19. Claudio Gutierrez, Carlos A. Hurtado, and Alejandro A. Vaisman. Introducing time into RDF. *IEEE Trans. Knowl. Data Eng.*, 19(2):207–218, 2007.
20. Joseph Y. Halpern and Yoav Shoham. A propositional modal logic of time intervals. *J. ACM*, 38(4):935–962, 1991.
21. Farhad Hussain, Huan Liu, Einoshin Suzuki, and Hongjun Lu. Exception rule mining with a relative interestingness measure. In Takao Terano, Huan Liu, and Arbee L. P. Chen, editors, *Knowledge Discovery and Data Mining, Current Issues and New Applications, 4th Pacific-Asia Conference, PADKK, Proceedings*, volume 1805 of *Lecture Notes in Computer Science*, pages 86–97. Springer, 2000.
22. Emil Eifrem Ian Robinson, Jim Webber. *Graph Databases, 2nd Edition New Opportunities for Connected Data.* O'Reilly Media, 2015.
23. Borislav Iordanov. Hypergraphdb: A generalized graph database. In Heng Tao Shen, Jian Pei, M. Tamer Özsu, Lei Zou, Jiaheng Lu, Tok Wang Ling, Ge Yu, Yi Zhuang, and Jie Shao, editors, *Web-Age Information Management - WAIM 2010 International Workshops: IWGD 2010, XMLDM 2010, WCMT 2010, Revised Selected Papers*, volume 6185 of *Lecture Notes in Computer Science*, pages 25–36. Springer, 2010.
24. Christian S. Jensen and Richard T. Snodgrass. Temporal data management. *IEEE Trans. Knowl. Data Eng.*, 11(1):36–44, 1999.
25. Dominique Laurent, Viet Phan Luong, and Nicolas Spyratos. The use of deleted tuples in database, querying and updating. *Acta Inf.*, 34(12):905–925, 1997.
26. Dominique Laurent, Viet Phan Luong, and Nicolas Spyratos. Updating intensional predicates in deductive databases. *Data Knowl. Eng.*, 26(1):37–70, 1998.
27. Dominique Laurent and Christel Vrain. Learning query rules for optimizing databases with update rules. In Dino Pedreschi and Carlo Zaniolo, editors, *Logic in Databases, International Workshop LID'96, Proceedings*, volume 1154 of *Lecture Notes in Computer Science*, pages 153–172. Springer, 1996.
28. Yann Loyer, Nicolas Spyratos, and Daniel Stamate. Hypothesis-based semantics of logic programs in multivalued logics. *ACM Trans. Comput. Log.*, 5(3):508–527, 2004.
29. Raymond Reiter. On closed world data bases. In *Logic and Data Bases*, pages 55–76, 1977.
30. Raymond Reiter. On formalizing database updates: Preliminary report. In Alain Pirotte, Claude Delobel, and Georg Gottlob, editors, *Advances in Database Technology - EDBT'92, 3rd International Conference on Extending Database Technology Proceedings*, volume 580 of *Lecture Notes in Computer Science*, pages 10–20. Springer, 1992.
31. Paolo Terenziani and Richard T. Snodgrass. Reconciling point-based and interval-based semantics in temporal relational databases: A treatment of the telic/atelic distinction. *IEEE Trans. Knowl. Data Eng.*, 16(5):540–551, 2004.
32. Riccardo Torlone. Update operations in deductive databases with functional dependencies. *Acta Inf.*, 31(6):573–600, 1994.