Knowledge and Information Systems (2020) 62:1571–1609
https://doi.org/10.1007/s10115-019-01402-w

REGULAR PAPER

Consistent updating of databases with marked nulls

Jacques Chabin[1] · **Mirian Halfeld-Ferrari**[1] · **Dominique Laurent**[2]

Received: 21 December 2018 / Revised: 13 September 2019 / Accepted: 15 September 2019 /
Published online: 28 September 2019

Abstract
This paper revisits the problem of consistency maintenance when insertions or deletions are performed on a valid database containing marked nulls. This problem comes back to light in real-world linked data or RDF databases when blank nodes are associated with null values. This paper proposes solutions for the main problems one has to face when dealing with updates and constraints, namely update determinism, minimal change and leanness of an RDF graph instance. The update semantics is formally introduced and the notion of core is used to ensure a database as small as possible (i.e. the RDF graph leanness). Our algorithms allow the use of constraints such as tuple-generating dependencies, offering a way for solving many practical problems.

Keywords Updates · Null values · Constraints · TGD · Logical database · RDF

1 Introduction

Today, dealing with incomplete data is imperative, specially due to the increasing use of applications involving data integration and data exchange. After having been largely studied in relational databases (as for instance in Imielinski and Lipski Jr. [31], Grahne [22], Libkin [37], Reiter [49], Zaniolo [56]), incomplete information appears, nowadays, as an important issue in the semantic web domain [5,44,54].

Incompleteness can be of many kinds, and finding a trade-off between expressivity and the difficulty in query answering queries is still a challenging issue [1]. Our work focusses on a database perspective where incompleteness arises when values (such as attribute values, or class or property instances) are missing. In this setting, we follow Reiter [49] who has shown how to extend the relational data model accordingly, thus providing FOL (first-order logic) semantics to null values of type 'value exists but is currently unknown'. Indeed, in Reiter [49], databases are not presented as models but as theories (i.e. a set of formulas)—a point of view which proved to be very suited to provide an intuitively correct semantics for null values. Logical database approach has then been largely influenced by that work. To focus the discussion, let us consider an example, similar to the one used in Reiter's paper.

✉ Mirian Halfeld-Ferrari
mirian@univ-orleans.fr

1 LIFO EA, Université d'Orléans, INSA CVL, Orléans, France

2 ETIS, CNRS, Université Paris Seine, Cergy-Pontoise, France

Example 1.1 In a database storing researcher names, conferences and their attendances, the information that *Ann* and *Bob* are researchers and that *Ann* attends $VLDB'18$ can be stored through the formula

$$\mathsf{Researcher}(Ann) \wedge \mathsf{Researcher}(Bob) \wedge \mathsf{Conf}(VLDB'18) \ \wedge \ \mathsf{Attends}(Ann, VLDB'18)).$$

In order to store the fact that 'Bob attended a conference, but we do not know which one', Reiter proposed the following formula

$$(\exists x)(\mathsf{Conf}(x) \wedge \mathsf{Attends}(Bob, x))$$

which asserts the existence of a conference x to which Bob attended. Furthermore, by naming this conference with a null value N_1 and by writing

$$\mathsf{Conf}(N_1) \wedge \mathsf{Attends}(Bob, N_1),$$

Reiter linked the database terminology to the FOL in the following terms: *logicians call [the null value] a Skolem constant, providing a technical device for the elimination of existential quantifiers in proof theory*. □

The viewpoint illustrated in the above example is exactly the one we adopt in this paper. However, instead of dealing with query answering (as in Reiter [49] and also in the majority of the papers dealing with incomplete information, such as Imielinski and Lipski Jr. [31], Grahne [22], Libkin [37] and Zaniolo [56]), our proposal focusses on updates. Although updates on incomplete database have been the subject of several work such as Grahne [22] and Winslett [55], it is worth noting that the dynamic aspects of data are usually neglected in favour of querying features. Modelling database updates as updates on FOL theory has been a way to organise and explain different update approaches. In this context, the question becomes: *given a FOL theory* Th *and new information concerning changes on* Th, *how can we find a new theory* Th_1 *which integrates this new information?* Update semantics has been considered under different viewpoints varying not only with respect to applications but also with respect to the considered research domain (whether it is artificial intelligence, database, philosophy, etc.). No consensus has been established, and many different approaches are available. However, Winslett offers in Winslett [55] a classification identifying two semantic classes for FOL theory updating: the model-based approach (whose goal is to update the models of a theory) and the formula-based approach (whose goal is to update the set of formulas). The approach proposed in this paper falls in the latter category. Our choice to start this paper by a theoretical overview of our update semantic goals reveals our preoccupation in placing our work in this *wide picture* of updating policy.

To achieve our goal, we first recall the following basic aspects which are taken into account in our proposal: (i) $\mathsf{Th}_{\mathbb{D},\mathbb{C}}$ consists of a *unique* conjunctive formula $\mathbb{D}$ which represents the database instance and a set of axioms $\mathbb{C}$ which represents constraints (logical implications); (ii) $\mathsf{Th}_{\mathbb{D},\mathbb{C}}$ is *consistent* in the sense that any instantiation of the variables in $\mathbb{D}$ yields a model of $\mathbb{C}$; and (iii) the well-known closed world assumption (CWA) and unique name assumption (UNA) hold.

In this context, an update may change the conjunctive formula $\mathbb{D}$ (i.e.the database instance) to obtain a new consistent theory $\mathsf{Th}_{\mathbb{D}',\mathbb{C}}$ from the original one $\mathsf{Th}_{\mathbb{D},\mathbb{C}}$. The changes on $\mathbb{D}$, when they are possible, follow the basic principle that: (i) the insertion of a formula β in $\mathbb{D}$ transforms $\mathbb{D}$ in a new conjunctive formula $\mathbb{D}$' having β as a sub-formula[1] and (ii) the

[1] Informally speaking, a sub-formula γ of a FOL formula α is a string occurring in α which is itself a FOL formula.

deletion of a formula β from $\mathbb{D}$ transforms $\mathbb{D}$ into a new formula $\mathbb{D}$' of which β is not a sub-formula.

Example 1.2 In Example 1.1, the current database instance $\mathbb{D}$ is the FOL formula

$$\mathsf{Researcher}(Ann) \wedge \mathsf{Researcher}(Bob) \wedge \mathsf{Conf}(VLDB'18) \\ \wedge \mathsf{Attends}(Ann, VLDB'18) \wedge (\exists x)(\mathsf{Conf}(x) \wedge \mathsf{Attends}(Bob, x))$$

Moreover, assume now that $\mathbb{C}$ consists of the following axiom:

$$(\forall x, y)(\mathsf{Attends}(x, y) \Rightarrow \mathsf{Researcher}(x)) \quad (1)$$

Clearly, $\mathsf{Th}_{\mathbb{D},\mathbb{C}}$ is consistent (because Ann and Bob attend conferences and are researchers). Now, the insertion of $\mathsf{Attends}(Alice, VLDB'18)$ transforming formula $\mathbb{D}$ above into $\mathbb{D} \wedge \mathsf{Attends}(Alice, VLDB'18)$ renders the theory non-consistent, because (1) cannot be satisfied for any instantiation of the variable x occurring in $\mathbb{D}$. However, the insertion of $\mathsf{Attends}(Alice, VLDB'18) \wedge \mathsf{Researcher}(Alice)$ results in $\mathbb{D} \wedge \mathsf{Attends}(Alice, VLDB'18) \wedge \mathsf{Researcher}(Alice)$ giving as a result a new consistent theory. Therefore, in our approach, would a user ask for inserting $\mathsf{Attends}(Alice, VLDB'18)$, consistency is automatically preserved through the additional insertion of $\mathsf{Researcher}(Alice)$ as a side effect.

Similarly, the deletion from $\mathbb{D}$ of $\mathsf{Attends}(Ann, VLDB'18) \wedge \mathsf{Researcher}(Ann)$ results in the new conjunctive formula $\mathbb{D}'$

$$\mathsf{Researcher}(Bob) \wedge \mathsf{Conf}(VLDB'18) \wedge (\exists x)(\mathsf{Conf}(x) \wedge \mathsf{Attends}(Bob, x))$$

thus leading to a new consistent theory $\mathsf{Th}_{\mathbb{D}',\mathbb{C}}$. Notice that consistency would not hold when deleting from $\mathbb{D}$ the only atom $\mathsf{Researcher}(Ann)$. □

It should be clear from the above example that our constraints (as FOL axioms) follow a database perspective, in the sense that (i) they are never modified by updates and (ii) they always must be satisfied by the database instance (the other option being to see constraints as *inference rules* used in query processing).

On the other hand, as we focus on updates while keeping database consistency, we have to face the important problem of data redundancy, which has been the subject of many studies (see for instance [16,20]). This problem occurs because, given a consistent theory and updates, there could be different target consistent theories (differing from the original one only on formula $\mathbb{D}$). Then, the question of whether there is a best solution has been answered in Fagin et al. [16] by the definition of the *core*. The results of this work are applied in this paper to define the 'best' database instance resulting from an update. The following example shows the intuition of our reasoning.

Example 1.3 Let us consider the following theory where axioms (constraints) impose a conference to be assigned to a location for which a web page informing about visa regulations is available.

$$\mathbb{C}: (\forall x)(\mathsf{Conf}(x) \Rightarrow (\exists y)(\mathsf{Loc}(x, y))) \\ (\forall x, y)(\mathsf{Loc}(x, y) \Rightarrow (\exists z)(\mathsf{VisaReg}(y, z))) \\ \mathbb{D}: \mathsf{Conf}(VLDB'18) \wedge (\exists x, y)(\mathsf{Loc}(VLDB'18, x) \wedge \mathsf{VisaReg}(x, y))$$

Intuitively, $\mathsf{Th}_{\mathbb{D},\mathbb{C}}$ states that $VLDB'18$ is settled in a location which is currently unknown and that this location should be associated with some (currently unknown) visa regulation. The insertion in $\mathsf{Th}_{\mathbb{D},\mathbb{C}}$ of $\mathsf{Loc}(VLDB'18, Rio)$ requires to insert an associated information about visa regulations according to the second constraint. However, since the associated URL is still

unknown, the inserted formula is $(\exists u)(\mathsf{Loc}(VLDB'18, Rio) \wedge \mathsf{VisaReg}(Rio, u))$, yielding $\mathbb{D}_1^1$

$$\mathsf{Conf}(VLDB'18) \wedge (\exists x, y)(\mathsf{Loc}(VLDB'18, x) \wedge \mathsf{VisaReg}(x, y)) \\ \wedge (\exists u)(\mathsf{Loc}(VLDB'18, Rio) \wedge \mathsf{VisaReg}(Rio, u)).$$

Notice that, in $\mathbb{D}_1^1$, the last conjunct is a partial instantiation of the second one, implying that $\mathbb{D}_1^1$ contains redundancies. To avoid this situation, the result of the insertion is simply defined equivalent formula $\mathbb{D}_1$

$$\mathsf{Conf}(VLDB'18) \wedge (\exists u)(\mathsf{Loc}(VLDB'18, Rio) \wedge \mathsf{VisaReg}(Rio, u)).$$

As a last update, assume now the insertion of $\mathsf{VisaReg}(Rio, url1)$, meaning that the URL containing visa regulations is now identified as $url1$. A similar reasoning as above would then produce the following non-redundant instance $\mathbb{D}_2$

$$\mathsf{Conf}(VLDB'18) \wedge \mathsf{Loc}(VLDB'18, Rio) \wedge \mathsf{VisaReg}(Rio, url1).$$

According to Fagin et al. [16], simplifications as above refer to the *core* of an instance, shown as being a 'condensed' version that preserves the answers to queries.

Notice that this approach is not the one adopted by Reiter for whom a database instance as $\mathbb{D}_1^1$ above would be acceptable. But keeping the database size as small as possible was out of the scope of Reiter [49]. □

We emphasize that the above example illustrates current challenges considered in the RDF world (see [54]) and that examples from these contexts can be expressed in a FOL formalism, ensuring generic solutions.

The main goal of our paper is to propose an approach to update databases with incomplete information that not only generalizes *naive tables* [1] but also applies in most current new database models, such as graph database systems or RDFS triple store systems. To achieve this goal, we consider the following context, based on the generic FOL formalism:

- A database instance is a closed conjunctive formula built up with atoms, possibly containing existentially quantified variables.
- The database instance must satisfy a set of constraints, each of them being a closed universally quantified implication with possibly, existentially quantified variables in the right hand side. These constraints are also called TGDs (tuple-generating dependencies) in the literature.

In this context, the following issues are investigated.

How to deal with marked nulls?

There is no consensus on how database semantics should be extended to deal with nulls and, furthermore, more difficulties arise when considering non-relational database models such as graph databases. To cope with this issue, we consider a FOL formalism (as in Reiter [49], where existentially quantified variables appear in their skolemization form, as *marked nulls*, Imielinski and Lipski Jr. [31], Lipski Jr. [41]). In doing so, we can model not only naive tables, but also most database models that are based on FOL. However, it should be noticed that updating naive tables has been the object of very few investigations and, in any case, this issue has not been studied in the presence of TGDs.

How to avoid redundancies?

As shown in Example 1.3, updates with incomplete information can yield redundancies that, when dealing with real size databases, must be avoided. We argue in this respect that our update processing maintains the database instance as 'small' as possible, while preserving answers to queries as defined in Fagin et al. [16]. We achieve this important feature by considering only instances equal to their *core* (see [19]). Although the complexity of core computation is known to be high in general [20], we refer to Sect. 7 for possible optimizations in our context.

How to preserve consistency while avoiding to generate too many nulls?

When considering TGDs, non-redundant incomplete information may be generated, possibly in an infinite way, when maintaining database consistency. A simple example of this issue is illustrated in the following.

Example 1.4 Consider the following constraints about citations:

$$\begin{aligned}\mathbb{C}: \ & (\forall x, y)(\mathsf{Cites}(x, y) \Rightarrow \mathsf{Paper}(x)) \\ & (\forall x, y)(\mathsf{Cites}(x, y) \Rightarrow \mathsf{Paper}(y)) \\ & (\forall x)(\mathsf{Paper}(x) \Rightarrow (\exists y)(\mathsf{Cites}(x, y)))\end{aligned}$$

These constraints stipulate that citations deal with papers and that every paper must contain at least one citation. Due to the third constraint, inserting the formula $\mathsf{Cites}(P1, P2) \wedge \mathsf{Paper}(P1) \wedge \mathsf{Paper}(P2)$ implies that the formula $(\exists y_1)(\mathsf{Cites}(P2, y_1) \wedge \mathsf{Paper}(y_1))$ has to be inserted as well. We are then clearly entering an infinite processing requiring the insertion of a formula of the form $(\exists y_1 \ldots y_k \ldots)(\mathsf{Cites}(P2, y_1) \wedge \mathsf{Paper}(y_1) \wedge \ldots \wedge \mathsf{Cites}(y_{k-1}, y_k) \wedge \mathsf{Paper}(y_k) \wedge \ldots)$. □

To cope with this problem, it is usual to impose *restrictions* on the constraints so as to ensure the termination of the generation of nulls, known as the chase procedure (see [6,13,23,45]). In our approach, we make no restriction regarding the constraints. Instead, we define the notion of *degree* of a null as being the imbrication degree assigned to a null generated by applying a constraint (roughly speaking the degree is represented by the integer $k - 1$ in Example 1.4). Then, assuming a pre-defined maximal degree δ_{max}, we *reject* insertions that entail that a null has a degree higher than δ_{max}. In this way, *whatever the constraints,* we limit the spreading of nulls in the database instance. Referring to Example 1.4, we allow the constraints but, for any maximal degree δ_{max}, we reject the insertion. Notice, however, that the insertion of $\mathsf{Cites}(P1, P2) \wedge \mathsf{Cites}(P2, P1) \wedge \mathsf{Paper}(P1) \wedge \mathsf{Paper}(P2)$ is allowed.

How to keep update deterministic?

We propose algorithms for inserting or deleting sets of atoms involving incomplete information (represented as nulls not occurring in the current database instance). As illustrated in the following example, deterministic updating is an issue for deletions, and we overcome this issue by assuming that choices can be made *a priori* at the design phase of constraints.

Example 1.5 Consider the following theory $\mathsf{Th}_{\mathbb{D},\mathbb{C}}$ where the constraint imposes that for any conference, all attendees must be registered.

$$\mathbb{C} : (\forall x, y)(\mathsf{Attends}(x, y) \wedge \mathsf{Conf}(y) \Rightarrow \mathsf{Registered}(x, y))$$
$$\mathbb{D} : \mathsf{Attends}(Bob, VLDB'18) \wedge \mathsf{Conf}(VLDB'18) \wedge \mathsf{Registered}(Bob, VLDB'18)$$

If Bob's registration is cancelled, at least one of the facts $\mathsf{Attends}(Bob, VLDB'18)$ or $\mathsf{Conf}(VLDB'18)$ has to be deleted along with $\mathsf{Registered}(Bob, VLDB'18)$, in order to maintain consistency. This situation yields three possible ways for processing the deletion, thus showing a case of non-deterministic update. In our approach, we assume that, for each constraint, the atom in the body to be deleted in case of the deletion of the head is known. In our example, a 'natural' choice would be the atom $\mathsf{Attends}(x, y)$. In this case, the deletion of $\mathsf{Registered}(Bob, VLDB'18)$ would imply deleting $\mathsf{Attends}(Bob, VLDB'18)$ in order to maintain consistency. In order to take into account this choice, we denote the chosen atom with '-' as an exponent. In our example, the constraint would be written as $(\forall x, y)(\mathsf{Attends}^{-}(x, y) \wedge \mathsf{Conf}(y) \Rightarrow \mathsf{Registered}(x, y))$. □

Additionally to determinism, the usual properties of minimal change and of monotonicity are also investigated in our framework.

Paper organization Section 2 provides the basics of our approach, using FOL, while database instance and database constraints are defined in Sect. 3. Then, Sect. 4 deals with the two ways constraints are applied in our updating processing: in a standard way for insertions and in an 'opposite' way that we call *backward* for deletions. Section 5 introduces our approach to updates by means of two algorithms, one for insertions and one for deletions. Properties of updates are studied in Sect. 6, while Sect. 7 positions our chase approach with regard to other chase versions and provides details on the core algorithm. Section 8 presents the results of some experiments and possible optimizations. In Sect. 9, related work are discussed and Sect. 10 concludes the paper.

2 Background: the core of a set of instantiated atoms

We assume that we have a standard FOL alphabet composed of three pairwise disjoint sets, namely: CONST, a set of constants, VAR, a set of variables, and PRED, a set of predicates, every predicate being associated with a positive integer called its arity. In this setting, a *term* is a constant or a variable and an atomic formula, or an atom, is a formula of the form $P(t_1, \ldots, t_n)$ where P is a predicate of arity n and $t_1, \ldots, t_n$ are terms. Every atom in which no variables occur is called a *fact*.

A *homomorphism* from a set of atoms A_1 to a set of atoms A_2 is a mapping h from the terms of A_1 to the terms of A_2 such that: (i) if $t \in$ CONST, then $h(t) = t$ and (ii) if $P(t_1, \ldots, t_n)$ is in A_1, then $P(h(t_1), \ldots, h(t_n))$ is in A_2.

A homomorphism associating every variable x in A_1 with a constant a occurring in A_2 is called a *valuation* of the variables in A_1. The set A_1 is *isomorphic* to the set A_2 iff there exists a homomorphism h_1 from A_1 to A_2 which admits an inverse homomorphism (from A_2 to A_1).

We let Φ be the set of all formulas ϕ of the form $(\exists \mathbf{X})(\varphi_1(\mathbf{X}_1) \wedge \ldots \wedge \varphi_n(\mathbf{X}_n))$ where $\mathbf{X}$ is a vector of variables made of all variables occurring in $\mathbf{X}_i$ $(i = 1, \ldots, n)$ and where for every $i = 1, \ldots, n$, $\varphi_i(\mathbf{X}_i)$ is an atomic formula in which the free variables are those in $\mathbf{X}_i$. Moreover, the set $\{\varphi_1(\mathbf{X}_1), \ldots, \varphi_n(\mathbf{X}_n)\}$ is denoted by $atoms(\phi)$.

Given ϕ in Φ, a *model* M of ϕ is a set of facts such that there exists a homomorphism from $atoms(\phi)$ to M. In such a setting, for all ϕ_1 and ϕ_2 in Φ, $\phi_1 \Rightarrow \phi_2$ holds if each model of ϕ_1 is a model of ϕ_2, and as usual, ϕ_1 and ϕ_2 in Φ are said to be *equivalent*, denoted by $\phi_1 \Leftrightarrow \phi_2$, if $\phi_1 \Rightarrow \phi_2$ and $\phi_2 \Rightarrow \phi_1$ both hold, that is, if ϕ_1 and ϕ_2 have the same models.

Lemma 2.1 *For all ϕ_1 and ϕ_2 in Φ, if there exists a homomorphism h from $atoms(\phi_2)$ to $atoms(\phi_1)$, then $\phi_1 \Rightarrow \phi_2$ holds.*

Proof Given a model M_1 of ϕ_1, there exists a homomorphism h_1 from $atoms(\phi_1)$ to M_1. Then, assuming that there exists a homomorphism h from $atoms(\phi_2)$ to $atoms(\phi_1)$ we have $h(atoms(\phi_2)) \subseteq atoms(\phi_1)$. Hence, by composition we obtain $h_1(h(atoms(\phi_2))) \subseteq h_1(atoms(\phi_1))$, and with $h_1(atoms(\phi_1)) \subseteq M_1$, we conclude that M_1 is also a model of ϕ_2. Thus, $\phi_1 \Rightarrow \phi_2$ holds. □

The following basic lemma is a consequence of Lemma 2.1.

Lemma 2.2 *For all ϕ_1 and ϕ_2 in Φ such that $atoms(\phi_1) \subseteq atoms(\phi_2)$, the equivalence $\phi_1 \Leftrightarrow \phi_2$ holds if and only if there exists a homomorphism h from $atoms(\phi_2)$ to $atoms(\phi_1)$.*

Proof By Lemma 2.1, assuming that there exists a homomorphism h from $atoms(\phi_2)$ to $atoms(\phi_1)$ implies that $\phi_1 \Rightarrow \phi_2$ holds. On the other hand, the inclusion $atoms(\phi_1) \subseteq atoms(\phi_2)$ shows that every model of ϕ_2 is also a model of ϕ_1, that is, $\phi_2 \Rightarrow \phi_1$ also holds.

Conversely, assuming that $\phi_1 \Leftrightarrow \phi_2$ holds, the proof relies on the fact that every ϕ in Φ has at least one model M defined by a valuation of variables h such that (i) for all variables x_1 and x_2 in ϕ, $h(x_1) \neq h(x_2)$ and (ii) for every fact A in M, $h^{-1}(A)$ contains exactly one atom in ϕ. To see this, let M be a model of ϕ defined by a valuation h such that, for all x_1 and x_2 occurring in ϕ, $h(x_1)$ and $h(x_2)$ are distinct constants not occurring in ϕ. By definition, h satisfies (i). Regarding point (ii), let A in M such that $h^{-1}(A) = \{\varphi_1, \varphi_2\}$. Then, φ_1 and φ_2 differ at least by one variable at a given position. Assume that at a given position p, x_1 occurs in φ_1 and not in φ_2. If the term in φ_2 at position p is a constant c, then it is not possible that $h(x_1) = c$ and so it is not possible that $h(\varphi_1) = h(\varphi_2)$. If the term in φ_2 at position p is a variable x_2, then according to (i), $h(x_1) \neq h(x_2)$ and so, it is not possible that $h(\varphi_1) = h(\varphi_2)$. On the other hand, if $h^{-1}(A) = \emptyset$, then it is clear that $M \backslash \{A\}$ is also a model of ϕ. Therefore, ϕ has at least one model M satisfying (i) and (ii).

Now let ϕ_1 and ϕ_2 be equivalent formulas in Φ such that $atoms(\phi_1) \subseteq atoms(\phi_2)$ and let M be a model of ϕ_1 satisfying points (i) and (ii). Since ϕ_1 and ϕ_2 are equivalent, M is also a model of ϕ_2. Let h_1 (respectively h_2) the valuation of variables in ϕ_1 (respectively in ϕ_2) such that $h_1(\phi_1) = M$ (respectively $h_2(\phi_2) \subseteq M$). Then, let h be defined for every variable x occurring in ϕ_2 by $h(x) = h_1^{-1}(h_2(x))$. Since h_1 satisfies (ii), h is well defined and for every φ in $atoms(\phi_2)$, $h(\varphi)$ is in $atoms(\phi_1)$. Thus, the proof is complete. □

If ϕ_1 and ϕ_2 are two formulas in Φ, ϕ_1 is said to be a *simplification* of ϕ_2, denoted by $\phi_1 \preceq \phi_2$, if (i) $\phi_1 \Leftrightarrow \phi_2$ holds, and (ii) $atoms(\phi_1) \subseteq atoms(\phi_2)$.

The relation $\preceq$ is a partial ordering. A simplification ϕ' of ϕ is said to be *minimal* if $\phi' \preceq \phi$ and there is no ϕ'' such that $\phi'' \prec \phi'$.

To illustrate the notion of simplification, consider the following very simple case where ϕ is the formula $(\exists x, y)(P(a, x) \wedge P(a, y))$. It is easy to see that $(\exists x)(P(a, x))$ and $(\exists y)(P(a, y))$ are two distinct but *equivalent* simplifications of ϕ. The following proposition shows that this always holds, i.e. that minimal simplifications are equal up to variable renaming.

Proposition 2.1 *If ϕ is a formula of Φ and ϕ_1 and ϕ_2 two* minimal *simplifications of ϕ, then $atoms(\phi_1)$ and $atoms(\phi_2)$ are isomorphic.*

Proof By Lemma 2.2, for $i = 1, 2$, there exists h_i such that for every φ in $atoms(\phi)$, $h_i(\varphi)$ is in $atoms(\phi_i)$. For $i = 1, 2$, if H_i is the restriction of h_i to the terms in ϕ_j ($j = 1, 2$ and $j \neq i$), we have $H_i(atoms(\phi_j)) \subseteq atoms(\phi_i)$.

We show that the inclusion is in fact an equality. Indeed, for $i = 1$ and $j = 2$, considering the morphism $h'_1 = h_1 \circ h_2$, for every φ in $atoms(\phi)$, implies that $h'_1(\varphi)$ is in $h_1(atoms(\phi_2))$. Therefore, by Lemma 2.2, the formula ϕ'_1 such that $atoms(\phi'_1) = h_1(atoms(\phi_2))$ is a simplification of ϕ. Consequently, assuming that $atoms(\phi'_1) \subset atoms(\phi_1)$ entails that $\phi'_1 \prec \phi_1$, which is a contradiction to the fact that ϕ_1 is a minimal simplification of ϕ. Recalling that $atoms(\phi'_1) = h_1(atoms(\phi_2))$, we obtain $h_1(atoms(\phi_2)) = atoms(\phi_1)$.

Since a similar reasoning holds for $i = 2$ and $j = 1$, and since for $i, j = 1, 2$ and $i \neq j$, $h_i(atoms(\phi_j)) = H_i(atoms(\phi_j))$, we obtain that $H_i(atoms(\phi_j)) = atoms(\phi_i)$, and the proof is complete. □

We recall that in the literature, the result of Proposition 2.1 follows from a similar result for graphs (see [30]), whereas our proof rather relies on FOL. Minimal simplifications are also called *cores* in the literature, and in the remainder of this paper, we shall follow this usual terminology and the core of a given formula ϕ is denoted by $core(\phi)$.

3 Consistent database with marked nulls

Database instance A database instance is basically a formula ϕ in Φ that cannot be simplified, i.e. such that $core(\phi) = \phi$. However, in order to simplify notation, we 'skolemize' formulas in Φ by replacing the variables with specific constants referred to as *Skolem constants* or as *(marked) nulls* and by omitting the existential quantifier. To do so, we assume an additional set of symbols in our alphabet, denoted by NULL, and assumed to be disjoint from the sets CONST and VAR. Skolem constants are then elements of the set NULL, and this induces that a term can be of one of the following types: either a constant, or a null, or a variable. Any atom of the form $P(t_1, \ldots, t_n)$ where for every $i = 1, \ldots, n$, t_i is in CONST $\cup$ NULL, is called an *instantiated atom* (not to be confused with a fact for which the terms t_i are in CONST).

Moreover, as usual, the transformed conjunctive formula is written as the *set* of its conjuncts. In other words, *a database instance is a set of instantiated atoms that can be written as* $atoms(Sk(\phi))$ *where* $Sk(\phi)$ *is the Skolem version of a formula* ϕ *in* Φ *such that* $core(\phi) = \phi$.

Example 3.1 Given ϕ defined by $(\exists x, y)(Q(a, x) \wedge S(a, x, y) \wedge Q(a, b))$, we have $\phi = core(\phi)$. Indeed, although $Q(a, x)$ can be mapped to $Q(a, b)$ there is no atom of the form $S(a, b, \_)$ to which we can map $S(a, x, y)$. Thus, the set $\{Q(a, N_1),\ S(a, N_1, N_2),\ Q(a, b)\}$ denotes the corresponding database instance, assuming that N_1 and N_2 are elements of the set NULL.

Now let ϕ' be defined by $(\exists x, y)(Q(a, x) \wedge R(a, y) \wedge S(a, x, y) \wedge Q(a, b) \wedge S(a, b, y))$. Since $core(\phi')$ is defined by $(\exists y)(Q(a, b) \wedge R(a, y) \wedge S(a, b, y))$, ϕ' cannot be seen as a database instance. In this case, the associated instance is $\{Q(a, b),\ R(a, N_1),\ S(a, b, N_1)\}$ where N_1 is in NULL. □

Constraints In database domain, constraints are usually established in order to ensure consistency and, thus, data quality. In this paper, we consider that constraints are implications known as *tuple-generating constraints* or TGDs for short [1], and of the generic form:

$$(\forall \mathbf{X}_1, \ldots, \mathbf{X}_k, \mathbf{Y}_1, \ldots, \mathbf{Y}_k)((L_1^-(\mathbf{X}_1, \mathbf{Y}_1) \wedge \ldots \wedge L_k(\mathbf{X}_k, \mathbf{Y}_k)) \\ \Rightarrow (\exists \mathbf{Z})(L(\mathbf{X}_1, \ldots, \mathbf{X}_k, \mathbf{Z})))$$

where $L(\mathbf{X}_1, \ldots, \mathbf{X}_k, \mathbf{Z})$ is an atom and for $i = 1, \ldots, k$, $L_i(\mathbf{X}_i, \mathbf{Y}_i)$ is an atom in which $\mathbf{X}_i$ (respectively $\mathbf{Y}_i$) is the vector of variables occurring in L_i and in L (respectively *not* in L). Removing quantifiers and denoting the left-hand side conjunction as $B(\mathbf{X}, \mathbf{Y})$ yields the simplified form $B(\mathbf{X}, \mathbf{Y}) \Rightarrow L(\mathbf{X}, \mathbf{Z})$.

The set of all atoms in $B(\mathbf{X}, \mathbf{Y})$ is denoted by $body(c)$, and the atom $L(\mathbf{X}, \mathbf{Z})$ is referred to as $head(c)$. Moreover, in order to ensure determinism of deletions (as discussed in Example 1.5), for every constraint c, a unique atom in $body(c)$ is identified using an hyphen character '-' as an exponent. In what follows, this atom is denoted by $body^-(c)$, with the convention that the exponent might be forgotten when $body(c)$ contains only one atom.

Constraints where no existentially quantified variables occur are said to be *full* (or *safe* according to Datalog terminology).

Constraint satisfaction is defined as usual in this context: given a set I of instantiated atoms, I *satisfies* the constraint c, denoted as $I \models c$, if for every homomorphism h such that $h(body(c)) \subseteq I$, there is an homomorphism h' such that $h(body(c)) = h'(body(c))$ and $h'(head(c))$ belongs to I. Notice that here, by *homomorphism* we mean any mapping from the variables in c to the set of constants or nulls. If $\mathbb{C}$ is a set of constraints, I *satisfies* $\mathbb{C}$, denoted as $I \models \mathbb{C}$, if for every c in $\mathbb{C}$, $I \models c$.

Example 3.2 Let $\mathbb{C}$ be a set containing the following two constraints:

- $c_1 : \mathsf{Conf}(x_1) \Rightarrow \mathsf{Loc}(x_1, y_1)$
- $c_2 : \mathsf{Loc}(x_2, y_2) \Rightarrow \mathsf{VisaReg}(y_2, z_2)$

and consider the following sets of instantiated atoms:

$$\begin{aligned}
\mathfrak{D}_1 &= \{\mathsf{Conf}(VLDB'18), \mathsf{Loc}(VLDB'18, N_1), \mathsf{VisaReg}(N_1, N_2)\} \\
\mathfrak{D}_2 &= \{\mathsf{Conf}(VLDB'18), \mathsf{Loc}(VLDB'18, N_1), \mathsf{VisaReg}(N_1, N_2), \\
&\quad \mathsf{Loc}(VLDB'18, Rio)\} \\
\mathfrak{D}_3 &= \{\mathsf{Conf}(VLDB'18), \mathsf{Loc}(VLDB'18, N_1), \mathsf{VisaReg}(N_1, N_2), \\
&\quad \mathsf{Loc}(VLDB'18, Rio), \mathsf{VisaReg}(Rio, N_3)\}.
\end{aligned}$$

Notice that $\mathfrak{D}_1$ and $\mathfrak{D}_3$ satisfy $\mathbb{C}$, whereas $\mathfrak{D}_2$ does not, due to c_2. Moreover, it should be clear that $\mathfrak{D}_3$ contains redundancies since instantiating N_1 by Rio shows that $\mathsf{Loc}(VLDB'18, N_1)$ and $\mathsf{VisaReg}(N_1, N_2)$ are not in $core(\mathfrak{D}_3)$. □

The following lemma states that considering the core of a consistent set of instantiated atoms cannot destroy consistency.

Lemma 3.1 *Given a set of constraints $\mathbb{C}$, for every set of instantiated atoms I, if $I \models \mathbb{C}$, then $core(I) \models \mathbb{C}$.*

Proof Let $c : B(\mathbf{X}, \mathbf{Y}) \Rightarrow L(\mathbf{X}, \mathbf{Z})$ in $\mathbb{C}$ such that $core(I) \not\models c$. In this case, $core(I)$ contains all instantiated atoms in the instance $B(\alpha, \beta)$ of $body(c)$, but no atom of the form $L(\alpha, \gamma)$. However, since $core(I) \subseteq I$, we have that $B(\alpha, \beta) \subseteq I$, implying that I contains an atom of the form $L(\alpha, \gamma)$, because $I \models \mathbb{C}$. Thus, $core(I)$ contains an instance $L(\alpha', \gamma')$ of $L(\alpha, \gamma)$ where $\alpha' \neq \alpha$. We obtain a contradiction because in this case, $core(I)$ would contain atoms in $B(\alpha', \beta')$ instead of $B(\alpha, \beta)$. □

Databases We are now ready to formally define the notion of database as consisting of a set of instantiated atoms and a set constraints as follows. In our approach, only consistent databases are considered, i.e. database instances should satisfy the database constraints. Nulls play a central role in our database model; their impact in update strategies will be described in the next section.

Definition 3.1 (*Database*) A *database* Δ is a pair of the form $\Delta = (\mathfrak{D}, \mathbb{C})$ where $\mathfrak{D}$ is a finite set of instantiated atoms and $\mathbb{C}$ is a finite set of constraints such that (i) $core(\mathfrak{D}) = \mathfrak{D}$ and (ii) $\mathfrak{D} \models \mathbb{C}$.

Referring back to Example 3.2, it is easy to see that $\Delta_1 = (\mathfrak{D}_1, \mathbb{C})$ satisfies Definition 3.1, because $\mathfrak{D}_1 \models \mathbb{C}$ and $core(\mathfrak{D}_1) = \mathfrak{D}_1$. On the other hand, neither $\Delta_2 = (\mathfrak{D}_2, \mathbb{C})$ nor $\Delta_3 = (\mathfrak{D}_3, \mathbb{C})$ do satisfy Definition 3.1, because on the one hand $\mathfrak{D}_2 \not\models \mathbb{C}$ and on the other hand $core(\mathfrak{D}_3) \neq \mathfrak{D}_3$. However, $\Delta'_3 = (core(\mathfrak{D}_3), \mathbb{C})$ satisfies Definition 3.1, because $core(\mathfrak{D}_3) \models \mathbb{C}$.

4 Constraint application

Given a set of instantiated atoms I, and a set of constraints $\mathbb{C}$, constraints can be applied *forward* (i.e. from body to head) or *backward* (i.e.from head to body) to I in order to generate a new set of atoms I'. These two ways of applying constraints are precisely the subject of the present section.

4.1 Applying constraints forward

Applying forward a constraint $c : B(\mathbf{X}, \mathbf{Y}) \Rightarrow L(\mathbf{X}, \mathbf{Z})$ to a given set of instantiated atoms I means (i) check whether I contains an instance $B(\alpha, \beta)$ of $body(c)$ and then, if the answer is yes, (ii) if I contains no atom of the form $L(\alpha, \gamma)$, create new nulls $\mathbf{N}$ and add in I the instantiated atom $L(\alpha, \mathbf{N})$.

More generally, this process has been widely studied in the past decades [15,16,20,45,47] under the name of *chasing*. While checking whether a set of constraints has terminating chase is undecidable [13], weak acyclicity [15] has been established as a sufficient condition for testing whether a set of constraints has terminating chase. In this paper, we propose a practical solution which not only guarantees chase termination, but also allows for flexibility when handling nulls. To this end, we define a specific chasing operator that, as explained in Example 1.4, avoids infinite generation of nulls by associating every null N with an integer denoted by $\delta(N)$ and called the *degree of* N.

We assume that we are given a *maximal null degree* $\delta_{\max}$, and starting the iterative chasing process with null degrees equal to 0, when a new null N has to be inserted due to the application of a constraint c, then:

- If the instance of $body(c)$ contains a null N' such that $\delta(N') \geq \delta_{\max}$, then the constraint is not applied;
- Otherwise, $\delta(N)$ is set to the maximal degree occurring in the instance of $body(c)$ plus 1.

Formally, chasing and handling null degrees are defined using an operator called *forward operator* as follows.

Definition 4.1 (*Forward Operator*) Given a set of constraints $\mathbb{C}$ and a maximum degree of nulls δ_{max}, the associated forward operator, denoted by: T_{fw}, is defined for every set of instantiated atoms I by:

$$
\begin{aligned}
T_{\mathsf{fw}}(I) = I \cup \{h(L(\mathbf{X},\mathbf{Z})) \mid\ & (\exists c : B(\mathbf{X},\mathbf{Y}) \Rightarrow L(\mathbf{X},\mathbf{Z}) \in \mathbb{C})(\exists h) \\
& ((c \text{ is full}) \wedge (h(body(c)) \subseteq I))\} \\
\cup \{h'(L(\mathbf{X},\mathbf{Z})) \mid\ & (\exists c : B(\mathbf{X},\mathbf{Y}) \Rightarrow L(\mathbf{X},\mathbf{Z}) \in \mathbb{C})(\exists h) \\
& ((c \text{ is not full}) \wedge (h(body(c)) \subseteq I) \wedge \\
& (\forall h'')((h''(body(c)) = h(body(c))) \Rightarrow \\
& \quad h''(L(\mathbf{X},\mathbf{Z})) \notin I) \wedge \\
& (\text{for all nulls } N \text{ in } h(body(c)),\ \delta(N) < \delta_{max}) \wedge \\
& (h' \text{ extends } h \text{ so as } h'(\mathbf{Z}) \text{ are new nulls } N \text{ with} \\
& \delta(N) = 0 \text{ if } h(body(c)) \text{ contains no null, or} \\
& \delta(N) = N_{\max} + 1, \text{ where} \\
& N_{\max} = \max\{\delta(N') \mid N' \text{ occurs in } h(body(c))\}))\}.
\end{aligned}
$$

Given a finite set of instantiated atoms I, let $\left(T^k_{\mathsf{fw}}\right)_{k\in\mathbb{N}}$ be defined by:

- $T^0_{\mathsf{fw}} = I$ where for every null N in I, $\delta(N)$ has been set to 0;
- $T^{k+1}_{\mathsf{fw}} = T_{\mathsf{fw}}(T^k_{\mathsf{fw}})$.

The following lemma shows that this sequence has a limit which is computed after a *finite* number of steps. In what follows, this limit is denoted by $T^*_{\mathsf{fw}}(I)$.

Lemma 4.1 *Considering the sequence* $\left(T^k_{\mathsf{fw}}\right)_{k\in\mathbb{N}}$ *as defined above, there exists an integer* k_0 *such that, for every* $k \geq k_0$, $T^{k+1}_{fw} = T^k_{fw}$.

Proof Since $I \subseteq T_{\mathsf{fw}}(I)$, holds, we trivially have that for every $k \geq 0$, $T^k_{\mathsf{fw}} \subseteq T^{k+1}_{\mathsf{fw}}$. Therefore, the sequence $\left(T^k_{\mathsf{fw}}\right)_{k\in\mathbb{N}}$ is monotonous. As a consequence, either $T^{k+1}_{\mathsf{fw}} = T^k_{\mathsf{fw}}$ or T^{k+1}_{fw} contains at least one atom not in T^k_{fw}.

On the other hand, to compute T^{k+1}_{fw} from T^k_{fw}, according to Definition 4.1, two sets may be computed. The first set is built using the full constraints, which does not create new constants or nulls. Therefore, the number of atoms incurred by full constraints is finite, since I is finite. The second set is built using the constraints with an existential variable in their head. An infinite computation would necessitate to apply at least one rule an infinite number of times. However, since degrees of nulls are bounded, each such constraint can only be applied a finite number of times. Therefore, the number of atoms containing nulls is also finite.

Hence, there must exist k_0 such that $T_{\mathsf{fw}}(T^{k_0}_{\mathsf{fw}})$ generates no new atom, that is, by monotonicity, such that $T_{\mathsf{fw}}(T^{k_0}_{\mathsf{fw}}) = T^{k_0}_{\mathsf{fw}}$. Applying again monotonicity, this implies that for every $k \geq k_0$, $T^{k+1}_{\mathsf{fw}} = T^k_{\mathsf{fw}}$, which completes the proof. □

The following example illustrates computations of $T^*_{\mathsf{fw}}(I)$. In this example, as well as in the remainder of the paper, we use the notation N^d to specify that N is a null of degree d, i.e. N^d means that $\delta(N) = d$ holds.

Example 4.1 In the context of Examples 1.3 and 1.5, consider the set $\mathbb{C}$ of the following constraints:

- $c_1 : \mathsf{Attends}^-(x_1, y_1), \mathsf{Conf}(x_1) \Rightarrow \mathsf{Registered}(x_1, y_1)$
- $c_2 : \mathsf{Conf}^-(x_2) \Rightarrow \mathsf{Loc}(x_2, y_2)$
- $c_3 : \mathsf{Loc}^-(x_3, y_3) \Rightarrow \mathsf{VisaReg}(y_3, z_3)$

and $I = \{\mathsf{Attends}(Bob, VLDB'18), \mathsf{Conf}(VLDB'18)\}$.

For $\delta_{max} = 0$, the computation of $T^*_{\mathsf{fw}}(I)$ yields the following: first, $T_{\mathsf{fw}}(I)$ is set to $I \cup \{\mathsf{Registered}(Bob, VLDB'18), \mathsf{Loc}(VLDB'18, N^0_1)\}$ due to c_1 and c_2. Then, when computing $T_{\mathsf{fw}}(T_{\mathsf{fw}}(I))$, the only constraint to apply is c_3 with $\mathsf{Loc}(VLDB'18, N^0_1)$. However, as this atom contains a null whose degree is equal to $\delta_{\max}$, c_3 is not triggered, implying that $T_{\mathsf{fw}}(T_{\mathsf{fw}}(I)) = T_{\mathsf{fw}}(I)$ and thus that $T^*_{\mathsf{fw}}(I) = T_{\mathsf{fw}}(I)$. Notice that, in this case $T^*_{\mathsf{fw}}(I)$ does *not* satisfy $\mathbb{C}$.

Now, for $\delta_{max} = 2$, the computation of $T^*_{\mathsf{fw}}(I)$ yields first $T_{\mathsf{fw}}(I) = I \cup \{\mathsf{Registered}(Bob, VLDB'18), \mathsf{Loc}(VLDB'18, N^0_1)\}$ as above. Then, contrary to the previous case, c_3 is triggered for computing $T_{\mathsf{fw}}(T_{\mathsf{fw}}(I))$, thus generating $\mathsf{VisaReg}(N^0_1, N^1_2)$. As no further constraint applies, we obtain that $T^*_{\mathsf{fw}}(I) = I \cup \{\mathsf{Registered}(Bob, VLDB'18), \mathsf{Loc}(VLDB'18, N^0_1), \mathsf{VisaReg}(N^0_1, N^1_2)\}$. Notice that, in this case, $T^*_{\mathsf{fw}}(I)$ does satisfy $\mathbb{C}$. □

The following proposition states that T_{fw} allows for restoring consistency when no nulls N such that $\delta(N) \geq \delta_{max}$ are generated.

Proposition 4.1 *For every set of instantiated atoms I, if $T^*_{fw}(I)$ contains no null N such that $\delta(N) \geq \delta_{max}$, then $T^*_{fw}(I) \models \mathbb{C}$.*

Proof Suppose that $c : B(\mathbf{X}, \mathbf{Y}) \Rightarrow L(\mathbf{X}, \mathbf{Z})$ is a constraint not satisfied by $T^*_{\mathsf{fw}}(I)$. There exists a homomorphism h such that $h(body(c)) \subseteq T^*_{\mathsf{fw}}(I)$ satisfying the following:

(a) If c is full, then $h(head(c))$ is not in $T^*_{\mathsf{fw}}(I)$. However, by Definition 4.1, $h(head(c))$ belongs to $T_{\mathsf{fw}}(T^*_{\mathsf{fw}}(I)) = T^*_{\mathsf{fw}}(I)$, a contradiction.
(b) If c is not full, then for every h' such that $h'(boby(c)) = h(body(c))$, $h'(head(c))$ is not in $T^*_{\mathsf{fw}}(I)$. However, since every null N in $h(body(c))$ is such that $\delta(N) < \delta_{max}$, according to Definition 4.1, $T_{\mathsf{fw}}(T^*_{\mathsf{fw}}(I)) = T^*_{\mathsf{fw}}(I)$ contains an instantiated atom $h'(head(c))$ containing new nulls with a degree d such that $d \leq \delta_{max}$, a contradiction which completes the proof. □

Referring to Proposition 4.1, it is important to notice from Example 4.1 that when $T^*_{\mathsf{fw}}(I)$ contains nulls N such that $\delta(N) \geq \delta_{max}$, then $T^*_{\mathsf{fw}}(I) \models \mathbb{C}$ does not hold in general. In this paper, nulls N such that $\delta(N) \geq \delta_{\max}$ are said to be *disallowed*. We emphasize in this respect that when the underlying set of constraint is empty and $\delta_{\max}$ is strictly positive, then no disallowed nulls can occur. On the other hand, whatever the set of constraints, when $\delta_{\max} = 0$ no nulls are allowed in the database (as in Halfeld Ferrari and Laurent [28]).

The following example illustrates that considering disallowed nulls has an impact on the computation of the core.

Example 4.2 Consider a database dealing with assistant professors and PhD students, as people who teach and do research in certain domains. Let us suppose that the following set is obtained by applying constraints forward with $\delta_{max} = 1$. (We refer to Example 5.1 for the details of this application.)

$$I = \{\mathsf{AssistProf}(Alice), \mathsf{Researcher}(Alice), \mathsf{Teaches}(Alice, N^0_1), \mathsf{PhD}(Alice), \mathsf{doesResearchIn}(Alice, N^0_2), \mathsf{Teaches}(Alice, N^1_3)\}.$$

When computing the core of I, two homomorphisms are possible, namely:

– h_1 such that $h_1(N^0_1) = N^1_3$, which gives

$$core(I) = \{\mathsf{AssistProf}(Alice), \mathsf{Researcher}(Alice), \mathsf{PhD}(Alice), \mathsf{doesResearchIn}(Alice, N^0_2), \mathsf{Teaches}(Alice, N^1_3)\}$$

– h_2 such that $h_2(N_3^1) = N_1^0$, which gives

$$core(I) = \{\mathsf{AssistProf}(Alice), \mathsf{Researcher}(Alice), \mathsf{PhD}(Alice), \\ \mathsf{doesResearchIn}(Alice, N_2^0), \mathsf{Teaches}(Alice, N_1^0)\}$$

Although these two sets are isomorphic (as stated earlier in Proposition 2.1), the first one contains disallowed nulls, contrary to the second one. Consequently, in our approach, the second one is accepted, whereas the first one *is not*. □

In order to take this situation into account, we consider a *restricted* version of the core, called R_core, descarding homomorphisms that associate disallowed nulls with allowed ones. These homomorphisms, called *restricted homomorphisms*, are defined as follows.

Definition 4.2 (*Restricted homomorphism*) A homomorphism h is said to be restricted if for all nulls N_i and N_j such that $h(N_i) = N_j$, it holds that if $\delta(N_i) < \delta_{\max}$, then $\delta(N_j) < \delta_{\max}$.

The following lemma, whose proof follows from the definitions and Proposition 2.1, shows the relationship between the core and the restricted core.

Lemma 4.2 *Given a set of instantiated atoms I and a maximal null degree $\delta_{\max}$,*

– *if $R\_core(I)$ contains no disallowed nulls, then $R\_core(I)$ and $core(I)$ are equal up to a null renaming;*
– *if $R\_core(I)$ contains disallowed nulls, then for every homomorphism h leading to $core(I)$, $h(I)$ contains disallowed nulls.*

4.2 Applying constraints backward

Applying backward a constraint $c : B(\mathbf{X}, \mathbf{Y}) \Rightarrow L(\mathbf{X}, \mathbf{Z})$ to a given a set of instantiated atoms I with respect to another set of instantiated atom J means:

1. Check whether I contains an instance $L(\alpha, \gamma_1)$ of $head(c)$ and whether $J \backslash I$ contains an instance $B(\alpha, \beta_1)$ of $body(c)$. If the answer is yes, then proceed to the step in the following.
2. If $J \backslash I$ contains no atom of the form $L(\alpha, \gamma_2)$ where $\gamma_1 \neq \gamma_2$, add to I the atom $A(\alpha_0, \gamma_0)$, where $A(\alpha_0, \gamma_0)$ is the instantiation of $body^-(c)$.

This processing is defined through a *backward operator* as follows.

Definition 4.3 (*Backward Operator*) Given a set of constraints $\mathbb{C}$ and two finite sets of instantiated atoms I and J, $T_{\mathsf{bw}}(I, J)$ is defined as follows:

$$T_{\mathsf{bw}}(I, J) = I \cup \{A \mid (\exists c \in \mathbb{C})(\exists h)((A = h(body^-(c))) \wedge \\ (h(body(c)) \subseteq (J \backslash I)) \wedge (h(head(c)) \in I) \wedge \\ (\forall h')((h'(body(c)) = h(body(c))) \Rightarrow h'(head(c)) \notin (J \backslash I)))\}.$$

Given two finite sets of instantiated atoms I and J, let $\left(T_{\mathsf{bw}}^k\right)_{k \in \mathbb{N}}$ be defined by:

– $T_{\mathsf{bw}}^0 = I$;
– $T_{\mathsf{bw}}^{k+1} = T_{\mathsf{bw}}(T_{\mathsf{bw}}^k, J)$.

The following lemma shows that this sequence has a limit which is computed after a *finite* number of steps. In what follows, this limit is denoted by $T_{\mathsf{bw}}^*(I, J)$.

Lemma 4.3 *Considering the sequence* $\left(T^k_{bw}\right)_{k\in\mathbb{N}}$ *as defined above, there exists an integer* k_0 *such that, for every* $k \geq k_0$, $T^{k+1}_{bw} = T^k_{bw}$.

Proof Since $I \subseteq T_{\mathsf{bw}}(I, J)$ holds, we trivially have that for every $k \geq 0$, $T^k_{\mathsf{bw}} \subseteq T^{k+1}_{\mathsf{bw}}$. Therefore, the sequence $\left(T^k_{\mathsf{bw}}\right)_{k\in\mathbb{N}}$ is monotonous. As a consequence, either $T^{k+1}_{\mathsf{bw}} = T^k_{\mathsf{bw}}$ or T^{k+1}_{bw} contains at least one atom not in T^k_{bw}. Moreover, since $T_{\mathsf{bw}}(I, J)$ is a subset of J, J is a finite set, and $T_{\mathsf{bw}}(I, J) \subseteq T_{\mathsf{bw}}(T_{\mathsf{bw}}(I, J), J)$, the total number of iterations is finite.

Hence, there must exist k_0 such that $T_{\mathsf{bw}}(T^{k_0}_{\mathsf{fw}}, J)$ generates no new atom, that is, by monotonicity, such that $T_{\mathsf{bw}}(T^{k_0}_{\mathsf{bw}}, J) = T^{k_0}_{\mathsf{bw}}$. Applying again monotonicity this implies that for every $k \geq k_0$, $T^{k+1}_{\mathsf{bw}} = T^k_{\mathsf{bw}}$, which completes the proof. □

The following proposition states that the limit $T^*_{\mathsf{bw}}(I, J)$ allows to restore consistency of J.

Proposition 4.2 *For all finite sets of instantiated atoms* I *and* J, *it holds that* $J \backslash T^*_{bw}(I, J) \models \mathbb{C}$.

Proof Suppose that there exists c which is not satisfied by $J\backslash T^*_{\mathsf{bw}}(I, J)$. Then, there exists a homomorphism h such that $h(body(c)) \subseteq J\backslash T^*_{\mathsf{bw}}(I, J)$ and $h(head(c)) \notin J\backslash T^*_{\mathsf{bw}}(I, J)$. In this case, by Definition 4.3, $h(body^-(c))$ would appear in $T_{\mathsf{bw}}(T^*_{\mathsf{bw}}(I, J), J)$, thus in $T^*_{\mathsf{bw}}(I, J)$. Hence, $h(body(c)) \subseteq J\backslash T^*_{\mathsf{bw}}(I, J)$ does not hold, which is a contradiction. Therefore, the proof is complete. □

Example 4.3 Referring to Example 4.1, let $\mathbb{C} = \{c_1, c_2, c_3\}$ defined by:

- $c_1 : \mathsf{Attends}^-(x_1, y_1), \mathsf{Conf}(x_1) \Rightarrow \mathsf{Registered}(x_1, y_1)$
- $c_2 : \mathsf{Conf}(x_2) \Rightarrow \mathsf{Loc}(x_2, y_2)$
- $c_3 : \mathsf{Loc}(x_3, y_3) \Rightarrow \mathsf{VisaReg}(y_3, z_3)$

Different computations of $T^*_{\mathsf{bw}}(I, J)$ are presented as follows for the following set J:

$$J = \{\mathsf{Attends}(Bob, VLDB'18), \mathsf{Conf}(VLDB'18), \mathsf{Registered}(Bob, VLDB'18), \mathsf{Loc}(VLDB'18, Rio), \mathsf{VisaReg}(Rio, url1), \mathsf{VisaReg}(Rio, url2)\}.$$

Considering first $I_1 = \{\mathsf{Registered}(Bob, VLDB'18)\}$, c_1 applies backward, producing $T_{\mathsf{bw}}(I_1, J) = I_1 \cup \{\mathsf{Attends}(Bob, VLDB'18)\}$. Since no other constraint applies, $T^*_{\mathsf{bw}}(I_1, J) = \{\mathsf{Registered}(Bob, VLDB'18), \mathsf{Attends}(Bob, VLDB'18)\}$.

For $I_2 = \{\mathsf{VisaReg}(Rio, url1)\}$, c_3 is considered as follows: $(J\backslash I_2)$ contains one instance of $body(c_3)$ associated with the two instances of $head(c_3)$, among which one is in $(J\backslash I_2)$, namely $\mathsf{VisaReg}(Rio, url2)$. Hence, according to Definition 4.3, we have $T^*_{\mathsf{bw}}(I_2, J) = I_2$ because no constraint other than c_3 applies.

For $I_3 = \{\mathsf{VisaReg}(Rio, url1), \mathsf{VisaReg}(Rio, url2)\}$, $(J\backslash I_3)$ contains an instance of $body(c_3)$, namely $\mathsf{Loc}(VLDB'18, Rio)$, associated with the two instances of $head(c_3)$ that belong to I_2. Thus, according to Definition 4.3, we have $T_{\mathsf{bw}}(I_3, J) = I_3 \cup \{\mathsf{Loc}(VLDB'18, Rio)\}$. Applying c_2 backward yields $T_{\mathsf{bw}}(T_{\mathsf{bw}}(I_3, J)) = T_{\mathsf{bw}}(I_3, J) \cup \{\mathsf{Conf}(VLDB'18)\}$. As no other constraint applies backward, $T^*_{\mathsf{bw}}(I_3, J) = I_3 \cup \{\mathsf{Loc}(VLDB'18, Rio), \mathsf{Conf}(VLDB'18)\}$. □

We point out that the backward operator T_{bw} does not require to consider null degrees, contrary to the case of the forward operator T_{fw}. In the next section, we show how the two operators T_{fw} and T_{bw} are used in our update processing.

5 Update processing

An update in our approach is a deterministic processing that modifies the database instance according to the following guidelines: given a database $\Delta = (\mathfrak{D}, \mathbb{C})$ and a set I of instantiated atoms such that nulls occurring in I do not occur in $\mathfrak{D}$:

Inserting I in Δ means generating a database $\Delta' = (\mathfrak{D}', \mathbb{C})$ such that $\mathfrak{D}'$ contains an instance of every atom of I. More precisely:

1. Compute first $T^*_{\mathsf{fw}}(\mathfrak{D} \cup I)$ and then the restricted core of the output.
2. If disallowed nulls occur, then the insertion is rejected and Δ is not changed. Otherwise $\Delta' = (\mathfrak{D}', \mathbb{C})$ where $\mathfrak{D}'$ is the instance defined above is returned.

Deleting I from Δ means generating a database $\Delta' = (\mathfrak{D}', \mathbb{C})$ such that $\mathfrak{D}'$ contains no atom isomorphic to an atom in I. More precisely, under the restriction that in I *every null occurs in one single atom*:

1. Delete from $\mathfrak{D}$ all atoms isomorphic to an atom of I. Denoting by $\mathfrak{D}_1$ the resulting set of instantiated atoms, compute $T^*_{\mathsf{fw}}(\mathfrak{D}_1)$ and then the restricted core of the output.
2. The computation of the restricted core can imply that atoms isomorphic to atoms in I are back in the output or that constraint satisfaction cannot be restored due to disallowed nulls. Denoting by $ToDel$ the set of all these atoms, compute $T^*_{\mathsf{bw}}(ToDel, \mathfrak{D}_1)$. After removing all these atoms, the resulting instance $\mathfrak{D}'$ is the core of the result of this last removal.

The details of update operations are introduced in the following.

5.1 Insertion processing

Given a database $\Delta = (\mathfrak{D}, \mathbb{C})$ and a maximal null degree δ_{max}, the insertion in Δ of a set of instantiated atoms iRequest is defined as the output of Algorithm 1. Notice that, as earlier mentioned, insertions might be rejected due to a null degree greater than or equal to δ_{max}. Different situations are illustrated by the following example.

Algorithm 1: $Insert(\Delta, \mathsf{iRequest})$

Input: The database $\Delta = (\mathfrak{D}, \mathbb{C})$, the maximal degree of nulls δ_{max} and iRequest, a set of instantiated atoms sharing no nulls with $\mathfrak{D}$
Output: The updated database $\Delta' = (\mathfrak{D}', \mathbb{C})$

```
for all null N occurring in 𝔇 ∪ iRequest do
    δ(N) := 0
𝔇1 := T*fw(𝔇 ∪ iRequest)
𝔇1 := R_core(𝔇1)
if max{δ(N) | N occurs in 𝔇1} < δmax then
    𝔇' := 𝔇1
else
    𝔇' := 𝔇 // The insertion is rejected
return Δ' = (𝔇', ℂ)
```

Example 5.1 In the context of Example 4.2, let $\Delta = (\mathfrak{D}, \mathbb{C})$ be a database where $\mathfrak{D} = \emptyset$ and $\mathbb{C}$ is the set of constraints below which state the following: c_1 and c_2 state, respectively, that

assistant professors are researchers and that a PhD student does some research in a scientific domain, and c_3 and c_4 state that doing research implies teaching at least a course.

- $c_1 : \mathsf{AssistProf}(x_1) \Rightarrow \mathsf{Researcher}(x_1)$
- $c_2 : \mathsf{PhD}(x_2) \Rightarrow \mathsf{doesResearchIn}(x_2, y_2)$
- $c_3 : \mathsf{doesResearchIn}(x_3, y_3) \Rightarrow \mathsf{Teaches}(x_3, z_3)$
- $c_4 : \mathsf{Researcher}(x_4) \Rightarrow \mathsf{Teaches}(x_4, z_4)$

We apply Algorithm 1 for $\mathsf{iRequest} = \{\mathsf{AssistProf}(Alice), \mathsf{PhD}(Alice)\}$, first for $\delta_{\max} = 0$ and then for $\delta_{\max} = 1$.

In the first case, the computation of $T^*_{\mathsf{fw}}(\mathfrak{D} \cup I)$ on line 3 first produces $T^1_{\mathsf{fw}}(\mathfrak{D} \cup \mathsf{iRequest}) = \mathsf{iRequest} \cup \{\mathsf{Researcher}(Alice), \mathsf{doesResearchIn}(Alice, N^0_1)\}$. Then, when computing $T^2_{\mathsf{fw}}(\mathfrak{D} \cup \mathsf{iRequest})$, c_3 is not applied forward because its instantiated body contains N^0_1 whose degree is equal to $\delta_{\max}$. However, c_4 applies forward, generating $\mathsf{Teaches}(Alice, N^0_2)$. We thus obtain

$$T^*_{\mathsf{fw}}(\mathfrak{D} \cup I) = \mathsf{iRequest} \cup \{\mathsf{doesResearchIn}(Alice, N^0_1), \mathsf{Teaches}(Alice, N^0_2)\}.$$

Since the computation of the restricted core on line 4 does not change this set, the test on line 5 fails, and so the insertion is rejected. This shows that when $\delta_{\max} = 0$, nulls are not accepted in the database instance, as was the case in our previous work [28].

Now, for $\delta_{\max} = 1$, the computation of $T^*_{\mathsf{fw}}(\mathfrak{D} \cup I)$ on line 3 is as follows:

$$\begin{aligned}
T^1_{\mathsf{fw}}(\mathfrak{D} \cup \mathsf{iRequest}) &= \mathsf{iRequest} \cup \{\mathsf{Researcher}(Alice), \mathsf{doesResearchIn}(Alice, N^0_1)\} \\
T^2_{\mathsf{fw}}(\mathfrak{D} \cup \mathsf{iRequest}) &= T^1_{\mathsf{fw}}(\mathfrak{D} \cup \mathsf{iRequest}) \\
&\quad \cup \{\mathsf{Teaches}(Alice, N^0_2), \mathsf{Teaches}(Alice, N^1_3)\} \\
T^3_{\mathsf{fw}}(\mathfrak{D} \cup \mathsf{iRequest}) &= T^2_{\mathsf{fw}}(\mathfrak{D} \cup \mathsf{iRequest}).
\end{aligned}$$

Thus, $T^*_{\mathsf{fw}}(\mathfrak{D} \cup I) = T^2_{\mathsf{fw}}(\mathfrak{D} \cup I)$, and the computation of the restricted core on line 4 is as discussed in Example 4.2, that is, the atom $\mathsf{Teaches}(Alice, N^1_3)$ is discarded as an instance of $\mathsf{Teaches}(Alice, N^0_2)$. Since in the produced set all nulls have a degree strictly less than $\delta_{\max}$, the test on line 5 succeeds, and so the insertion is performed producing the following database instance:

$$\begin{aligned}
\mathfrak{D}' = \{&\mathsf{AssistProf}(Alice), \mathsf{Researcher}(Alice), \mathsf{Teaches}(Alice, N_2), \\
&\mathsf{PhD}(Alice), \mathsf{doesResearchIn}(Alice, N_1)\}.
\end{aligned}$$

□

We refer to Sect. 7 regarding the relationship of our insertion processing and the chase procedures as summarized in Onet [45]. In particular, we show that, when an insertion is not rejected and when chase procedures terminate, our approach is similar to the procedure of core chase.

5.2 Deletion semantics

Before considering how deletions are performed, their semantics should be clarified by considering three particular aspects: (i) the meaning of a deletion of an atom with a null value, (ii) the possibility of considering linked nulls in the deletion request and (iii) the determinism of deletions.

Concerning aspect (i), in our approach, a deletion expressed as 'delete $P(a, N)$' means 'delete all atoms over P where a is associated with a null'. We draw attention on the fact

that this update should *not* be understood as 'delete all atoms of the form $P(a, \_)$'. To avoid confusions, we do *not* consider updates whose expression involves the placeholder '_'.

Concerning aspect (ii), the presence of nulls in the atoms to be deleted raises subtle issues, as illustrated by the following example.

Example 5.2 In the context of Example 1.5, let $\Delta_1 = (\mathfrak{D}_1, \mathbb{C}_1)$ where $\mathbb{C}_1 = \emptyset$ and $\mathfrak{D}_1 = \{\mathsf{Attends}(Bob, VLDB'18)\}$.

If the atoms in $I_1 = \{\mathsf{Attends}(Bob, VLDB'18), \mathsf{Conf}(VLDB'18)\}$ have to be deleted, then any database system will output the database instance $\mathfrak{D}'_1 = \emptyset$.

Now, as a case where nulls are involved, let $\Delta_2 = (\mathfrak{D}_2, \mathbb{C}_1)$ where $\mathfrak{D}_2 = \{\mathsf{Attends}(Bob, N_1)\}$. Considering that the atoms in $I_2 = \{\mathsf{Attends}(Bob, N_2), \mathsf{Conf}(N_2)\}$ have to be deleted, the following question arises: *should the database instance be changed nor not?* The following two answers are then possible:

- Referring to the previous case, the answer is `yes` because I_2 contains an atom isomorphic to the one in $\mathfrak{D}_2$, meaning that the result is $\mathfrak{D}'_2 = \emptyset$.
- Referring to FOL semantics, I_2 stipulates that we are deleting all pairs of atoms stating that Bob attends a conference whose name is unknown. Since $\mathfrak{D}_2$ contains no such pair, the deletion should not change the database instance. Thus, the answer to the question above is `no`!

To keep deletion semantics for null and non-null atoms intuitively similar, while avoiding questions as above, we do not allow nulls to have several occurrences in the set of atoms to be deleted. That is, in our case, the set I_2 above is changed to $I = \{\mathsf{Attends}(Bob, N_2), \mathsf{Conf}(N_3)\}$.

It is worth noting that the above question also concerns: (1) a choice between an 'or' semantics (delete if it matches *this OR that*) or an 'and' semantics (delete if it matches *this AND that*), together with (2) the verification whether the required deletion englobes a whole partition in the database instance (e.g. the situation where we have I_2 and an instance containing $\mathsf{Attends}(Bob, N_1)$, $\mathsf{Conf}(N_1)$ and $\mathsf{Loc}(N_1, Rio)$). Since the 'and' semantics raises further important issues, this point lies out of the scope of the present paper. □

Finally, regarding aspect (iii), we recall from Sect. 4.2 that the backward operator T_{bw} definition is based on the fact that for every constraint c, a literal $L_i(\mathbf{X}_i, \mathbf{Y}_i)$ in $B(\mathbf{X}, \mathbf{Y})$ of c has been marked with '-' as its exponent, meaning that:

- When the database contains the atom $L_i(\alpha_i, \beta_i)$ along with $L(\alpha, \gamma)$,
  if $L(\alpha, \gamma)$ has to be deleted and $\mathfrak{D}\backslash\{L(\alpha, \gamma)\}$ does not satisfy c,
  then $L_i(\alpha_i, \beta_i)$ is deleted to restore constraint satisfaction.

Deletion is thus deterministic because when restoring consistency, each constraint applied backward generates a unique side effect. It should be noticed, as argued later in Sect. 9, that this way of ensuring the determinism of deletions can be related to what is called *active rules* in the literature [50].

5.3 Deletion processing

Algorithm 2 implements deletions and is illustrated in the following examples. We recall that, as argued in Example 5.2, we assume that in the sets of atoms to be deleted, every null occurs once.

Example 5.3 In this example, we consider $\Delta = (\mathfrak{D}, \mathbb{C})$ where $\mathbb{C}$ is as in Example 4.3, that is:

Algorithm 2: Delete(Δ, dRequest)

Input: The database $\Delta = (\mathfrak{D}, \mathbb{C})$, the maximal degree of nulls δ_{max} and dRequest, a set of instantiated atoms where nulls occur only once and do not occur in $\mathfrak{D}$
Output: The updated database $\Delta' = (\mathfrak{D}', \mathbb{C})$
1: $ToDel := \{A \in \mathfrak{D} \mid (\exists \varphi \in \mathsf{dRequest})(\varphi \text{ is isomorphic to } A)\}$
2: **for all** null N occurring in $\mathfrak{D} \backslash ToDel$ **do**
3: $\quad \delta(N) := 0$
4: $\mathfrak{D}_1 := T^*_{\mathsf{fw}}(\mathfrak{D} \backslash ToDel)$
5: $\mathfrak{D}_1 := R\_core(\mathfrak{D}_1)$
6: $Disallowed := \{A \in \mathfrak{D}_1 \mid A \text{ contains a null } N \text{ such that } \delta(N) \geq \delta_{max}\}$
7: $Back := \{A \in \mathfrak{D}_1 \mid (\exists \varphi \in ToDel)(\varphi \text{ is isomorphic to } A)\}$
8: **if** $Disallowed \cup Back = \emptyset$ **then**
9: $\quad \mathfrak{D}' := \mathfrak{D}_1$
10: **else**
11: $\quad \mathfrak{D}_2 := \mathfrak{D}_1 \backslash T^*_{\mathsf{bw}}((Disallowed \cup Back), \mathfrak{D}_1)$
12: $\quad \mathfrak{D}' := R\_core(\mathfrak{D}_2)$
13: **return** $\Delta' = (\mathfrak{D}', \mathbb{C})$

– $c_1 : \mathsf{Attends}^{-}(x_1, y_1), \mathsf{Conf}(y_1) \Rightarrow \mathsf{Registered}(x_1, y_1)$
– $c_2 : \mathsf{Conf}(x_2) \Rightarrow \mathsf{Loc}(x_2, y_2)$
– $c_3 : \mathsf{Loc}(x_3, y_3) \Rightarrow \mathsf{VisaReg}(y_3, z_3)$

Moreover, we set δ_{max} to be equal to 1 and we consider $\mathfrak{D}$ as follows:

$$\mathfrak{D} = \{\mathsf{Attends}(Bob, VLDB'18), \mathsf{Conf}(VLDB'18), \mathsf{Registered}(Bob, VLDB'18), \mathsf{Loc}(VLDB'18, Rio), \mathsf{VisaReg}(Rio, url1), \mathsf{VisaReg}(Rio, url2)\}.$$

We first apply Algorithm 2 for $\mathsf{dRequest} = \{\mathsf{Registered}(Bob, VLDB'18)\}$. We have $ToDel = \mathsf{dRequest}$ and on line 4, $\mathfrak{D}_1 = \mathfrak{D}$ because when applying T_{fw} to $\mathfrak{D} \backslash \mathsf{dRequest}$, $\mathsf{Registered}(Bob, VLDB'18)$ is generated. Then, the restricted core computation on line 5 does not change $\mathfrak{D}_1$, meaning that we have $Disallowed = \emptyset$ (since no nulls are present) and $Back = \{\mathsf{Registered}(Bob, VLDB'18)\}$. Therefore, the test on line 8 fails and $T^*_{\mathsf{bw}}(\{\mathsf{Registered}(Bob, VLDB'18)\}, \mathfrak{D}_1)$ is computed on line 11. By Example 4.3, the result is $\{\mathsf{Registered}(Bob, VLDB'18), \mathsf{Attends}(Bob, VLDB'18)\}$ and so, on line 11, $\mathfrak{D}_2$ is set to $\mathfrak{D} \backslash \{\mathsf{Registered}(Bob, VLDB'18), \mathsf{Attends}(Bob, VLDB'18)\}$. Since no null is involved, the computation of the restricted core on line 12 produces $\mathfrak{D}' = \mathfrak{D}_2$, which is the updated database instance.

Now, we apply Algorithm 2 for $\mathsf{dRequest} = \{\mathsf{VisaReg}(Rio, url1)\}$. As above, $ToDel = \mathsf{dRequest}$, but on line 4, we have $\mathfrak{D}_1 = \mathfrak{D} \backslash \mathsf{dRequest}$, because $head(c_3)$ has two instances in $\mathfrak{D}$ matching with $\mathsf{Loc}(VLDB'18, Rio)$. Then, it is easy to see that the sets $Disallowed$ and $Back$ are set to $\emptyset$, and so the result $\mathfrak{D}' = \mathfrak{D}_1$ is output on line 9. That is, the deletion is processed by removing $\mathsf{VisaReg}(Rio, url1)$ from $\mathfrak{D}$.

As another deletion, let $\mathsf{dRequest} = \{\mathsf{VisaReg}(Rio, url1), \mathsf{VisaReg}(Rio, url1)\}$. As in the previous two cases, $ToDel = \mathsf{dRequest}$, but on line 4, $\mathfrak{D}_1$ is set to $(\mathfrak{D} \backslash ToDel) \cup \{\mathsf{VisaReg}(Rio, N^0_1)\}$ due to c_3. Then, as in the previous case, the sets $Disallowed$ and $Back$ are set to $\emptyset$, and the resulting updated instance $\mathfrak{D}_1$ is output on line 9. That is, the updated database instance $\mathfrak{D}'$ is processed by removing $\mathsf{VisaReg}(Rio, url1)$ and $\mathsf{VisaReg}(Rio, url2)$ and adding $\mathsf{VisaReg}(Rio, N^0_1)$. We point out that in this case $\mathfrak{D}' \subseteq \mathfrak{D}$ does not hold.

As a last deletion in this example, let $\mathsf{dRequest} = \{Loc(VLDB'18, Rio)\}$, which again is equal to $ToDel$ on line 1 of Algorithm 2. Then, on line 4, $\mathfrak{D}_1$ is set to $(\mathfrak{D} \backslash ToDel) \cup \{\mathsf{Loc}(VLDB'18, N^0_1), \mathsf{VisaReg}(N^0_1, N^1_2)\}$ due to c_2 and c_3. Since $\mathsf{Loc}(VLDB'18, N^0_1)$ can-

not be instantiated, the restricted core computation on line 5 does not change $\mathfrak{D}_1$, and so, on line 6 we have $Disallowed = \{\mathsf{VisaReg}(N_1^0, N_2^1)\}$ because $\delta_{\max} = 1$. On line 7 $Back$ is set to $\emptyset$, and the test on line 8 fails implying that $T^*_{\mathsf{bw}}(\{\mathsf{VisaReg}(N_1^0, N_2^1)\}, \mathfrak{D}_1)$ is computed on line 11, producing $\{\mathsf{VisaReg}(N_1^0, N_2^1), \mathsf{Loc}(VLDB'18, N_1^0), \mathsf{Conf}(VLDB'18)\}$. Since $\mathfrak{D}_2$ as computed on line 11 contains no null, the restricted core computation on line 12 does not change $\mathfrak{D}_2$. Consequently, $\mathfrak{D}'$ is obtained by removing from $\mathfrak{D}$ the atoms $\mathsf{Loc}(VLDB'18, Rio)$ and $\mathsf{Conf}(VLDB'18)$. □

In the following two examples, we show that the core computations on lines 5 and 12 in Algorithm 2 are necessary in order to properly implement deletions.

Example 5.4 As for the core computation on line 5 of Algorithm 2, let $\Delta = (\mathfrak{D}, \mathbb{C})$ where $\mathfrak{D} = \{P(a, b),\ P(a, N_1),\ P(a', N_1)\}$ and where $\mathbb{C} = \emptyset$ (which satisfies that $R\_core(\mathfrak{D}) = \mathfrak{D}$). We also set $\delta_{\max} = 1$. Since $\mathbb{C}$ is empty, this implies that all null degrees are equal to 0. Hence, no null can be disallowed and thus, all homomorphisms are restricted.

For $\mathsf{dRequest} = \{P(a', N)\}$, we have $ToDel = \{P(a', N_1)\}$ and $T^*_{\mathsf{fw}}(\mathfrak{D} \backslash \mathsf{dRequest}) = \mathfrak{D} \backslash \{P(a', N_1)\}$. Therefore, on line 4, we have $\mathfrak{D}_1 = \{P(a, b),\ P(a, N_1)\}$, and thus, at this stage we do not have that $R\_core(\mathfrak{D}_1) = \mathfrak{D}_1$, since $R\_core(\mathfrak{D}_1) = \{P(a, b)\}$. This is what is computed on line 5 and returned by Algorithm 2, since in that case, the sets $Disallowed$ and $Back$ are empty.

We point out from this example that even when no constraints are considered in Δ, the core computations are necessary in order to keep the size of the database instance as small as possible. □

Example 5.5 To illustrate the core computation on line 12 of Algorithm 2, consider that $\delta_{\max} = 1$ and let $\Delta = (\mathfrak{D}, \mathbb{C})$ where $\mathfrak{D} = \{P(a, N_1),\ Q(N_1),\ Q(b),\ R(a, N_2)\}$ and $\mathbb{C} = \{c_1, c_2\}$ defined as follows:

- $c_1 : P(x_1, y_1) \Rightarrow Q(y_1)$
- $c_2 : P(x_2, y_2) \Rightarrow R(x_2, z_2)$

For $\mathsf{dRequest} = \{R(a, N)\}$, we have $ToDel = \{R(a, N_2)\}$, and due to c_2, $\mathfrak{D}_1 = T^*_{\mathsf{fw}}(\mathfrak{D} \backslash ToDel)$ is $(\mathfrak{D} \backslash \{R(a, N_2^0)\}) \cup \{R(a, N_3^0)\}$. Therefore, on line 6, the set $Disallowed$ is set to $\emptyset$, and on line 7, the set $Back$ is set to $\{R(a, N_3^0)\}$. Since $T^*_{\mathsf{bw}}(\{R(a, N_3^0)\}, \mathfrak{D}_1) = \{R(a, N_3^0),\ P(a, N_1^0)\}$, on line 11 we have $\mathfrak{D}_2 = \{Q(N_1^0),\ Q(b)\}$. Thus, the core computation of line 12 returns $R\_core(\mathfrak{D}_2) = \{Q(b)\}$, and so Algorithm 2 outputs $\Delta' = (\mathfrak{D}', \mathbb{C})$ where $\mathfrak{D}' = \{Q(b)\}$. □

5.4 Complexity remarks

Checking whether all previously satisfied TGDs are still satisfied after an update is an NP-complete problem in the general case (see [19,47]).

In Algorithm 1, the most expensive computations concern the chase (performed on line 3) and the core (performed on line 4). Similarly, in Algorithm 2, the most expensive computations appear on lines 4 and 11 (where the forward or the backward operator is applied) and on lines 5 and 12 (where the core is computed). We refer to Sect. 7 for a detailed discussion on our core algorithm together with its computation.

Here, following a similar reasoning as in Halfeld Ferrari and Laurent [28], we consider complexity aspects of the computing $T^*_{\mathsf{fw}}(I)$ for a given instance I. Denoting by α the maximal arity of predicates, the number of constants occurring in I is bounded by $\alpha \times |I|$, and so the

number of possible instantiations of an atom $L(x_1, \ldots, x_\alpha)$ is $(\alpha \times |I|)^\alpha$. Thus, for a constraint c with m_c atoms in its body, the number of its possible instantiations is $((\alpha \times |I|)^\alpha)^{m_c}$ or just $O|I|^{\alpha.m_c}$. This implies that, for one constraint c, the number of possible atoms generated by the application of T_{fw} on c and I is $O|I|^{\alpha.m_c}$. Hence, applying T_{fw} on $\mathbb{C}$ and I generates a new set of atoms in time $O(|I|^{\alpha.m})$, where m is the maximal number of atoms in the bodies of the constraints in $\mathbb{C}$. As by Lemma 4.1, constraints are applied at most $(k_0 + 1)$ times, the complexity of computing $T^*_{\mathsf{fw}}(I)$ is in $O(|I|^{\alpha.m.(k_0+1)})$. Consequently, the computation of $T^*_{\mathsf{fw}}(\mathfrak{D} \cup \mathsf{iRequest})$ (line 3 of Algorithm 1) is in $O(|\mathfrak{D} \cup \mathsf{iRequest}|^{\alpha.m.(k_0+1)})$.

In Algorithm 2, a similar reasoning shows that the complexity of computing line 4 is $O(|\mathfrak{D} \backslash ToDel|^{\alpha.m.(k_0+1)})$. Moreover, the complexity of applying the operator T_{bw} is similar to that obtained for the forward operator, except that T_{bw} operates on rules having only one atom in the body and in the head. Thus, the complexity of computing $T^*_{\mathsf{bw}}((Disallowed \cup Back), \mathfrak{D}_1)$ (line 11) is $O(|Disallowed \cup Back|^{\alpha.(k_0+1)})$ where k_0 is the integer in Lemma 4.3.

To sum up the above remarks, it can be stated that the generation of side effects for insertions and deletions in our approach is polynomial with respect to the sizes of $\mathfrak{D}$, iRequest and dRequest.

6 Update properties

In this section, we investigate the properties of the updates as defined in the previous section. To this end, given $\Delta = (\mathfrak{D}, \mathbb{C})$ and the insertion of iRequest or the deletion of dRequest, we denote by $\Delta' = (\mathfrak{D}', \mathbb{C})$ the result of the update and we consider the following issues:

Effectiveness This property states that the output of any update is indeed a database, i.e. Δ' satisfies Definition 3.1. Moreover, when the update is not rejected, the property ensures that the update is effective, meaning that in case of insertion, $\mathfrak{D}'$ contains an instance of every atom in iRequest and in case of deletion, $\mathfrak{D}'$ contains no atom isomorphic to an atom in dRequest.

Determinism The way an insertion (respectively a deletion) is processed depends only on Δ and on the set iRequest (respectively dRequest), i.e. no choice has to be made during the processing.

Monotonicity This property requires an ordering over instances to state that after any insertion (respectively any deletion), $\mathfrak{D}'$ is greater (respectively less) than or equal to $\mathfrak{D}$. Example 5.3 shows that, in our approach, set inclusion cannot be considered, as in standard approaches. Instead we compare instances according to a relation denoted by $\sqsubseteq$, based on homomorphisms and to be defined in the following.

Minimal change Intuitively, minimal change states that updates should modify the database instance as few as possible. However, to check this property, all instances have to be considered, which is in general non-tractable. Even if this would be feasible, there exist in general several distinct instances satisfying minimal change, which is in contradiction to the property of determinism. Instead, minimal change in our approach is expressed as follows: the insertion of iRequest (respectively the deletion of dRequest) satisfies the minimal change property if, when 'forgetting' one of the specified side effects, at least one constraint is not satisfied.

It should be clear that our update processing is deterministic because no choice has to be made when running Algorithms 1 or 2. We, however, recall that deletions are deterministic thanks to the choice of an atom in the bodies of constraints, a choice made at design phase

and not during update processing. We now define the relation according to which instances are compared.

Definition 6.1 (*Comparison of sets of instantiated atoms*) Let I_1 and I_2 be two sets of instantiated atoms. We say that $I_1 \sqsubseteq I_2$ holds if there exists a homomorphism h such that $h(I_1) \subseteq I_2$.

Notice that $\sqsubseteq$ is a partial pre-ordering and that $\sqsubseteq$ generalizes set inclusion, because $I_1 \subseteq I_2$ implies $I_1 \sqsubseteq I_2$. Moreover, it is easy to see that if ϕ_1 and ϕ_2 are the formulas in Φ associated, respectively, with I_1 and I_2, then $I_1 \sqsubseteq I_2$ holds if and only if so does $\phi_1 \Rightarrow \phi_2$. As a consequence, for every set of instantiated atoms I, $I \sqsubseteq R\_core(I)$ and $R\_core(I) \sqsubseteq I$ both hold.

Regarding insertions, it should be clear that given $\Delta = (\mathfrak{D}, \mathbb{C})$ and a set iRequest, if Algorithm 1 returns $\Delta' = (\mathfrak{D}', \mathbb{C})$ where $\mathfrak{D}' = \mathfrak{D}$, then the update is *not* effective when $\mathfrak{D}$ contains no instance of an atom in iRequest. However, such an update is trivially deterministic, monotonic and satisfies minimal change. Knowing that it has already been stated that our update processing is deterministic, the following proposition shows that when $\mathfrak{D}' \neq \mathfrak{D}$ (implying that the insertion is not rejected), the insertion satisfies all properties listed above.

Moreover, in this proposition, minimal change is stated by considering side effects as those instantiated atoms not in iRequest (up to null renaming) are inserted so as to maintain consistency.

Proposition 6.1 *Let $\Delta = (\mathfrak{D}, \mathbb{C})$ be a database and iRequest a finite set of facts. If Algorithm 1 returns $\Delta' = (\mathfrak{D}', \mathbb{C})$ where $\mathfrak{D}' \neq \mathfrak{D}$, then $\mathfrak{D}'$ satisfies the following statements*:

1. *Effectiveness: (a) for every φ in iRequest, $\mathfrak{D}'$ contains an instance of φ, (b) $R\_core(\mathfrak{D}') = \mathfrak{D}'$ and (c) $\mathfrak{D}' \models \mathbb{C}$.*
2. *Monotonicity: $\mathfrak{D} \sqsubseteq \mathfrak{D}'$.*
3. *Minimal change: For every φ in $\mathfrak{D}'$ not in $\mathfrak{D}$ and not isomorphic to an atom in iRequest, $\mathfrak{D}' \setminus \{\varphi\} \not\models \mathbb{C}$.*

Proof See ("Appendix A"). □

The following proposition gives basic properties of Algorithm 2, namely deletions are effective and monotonic. However, as will be argued later, deletions do not satisfy the requirement of minimal change. We notice in this respect that, for deletions, side effects as those instantiated atoms not in dRequest (up to null renaming) have been deleted so as to maintain consistency.

Proposition 6.2 *Let $\Delta = (\mathfrak{D}, \mathbb{C})$ be a database and dRequest a finite set of instantiated atoms where every null occurs at most once. Algorithm 2 returns $\Delta' = (\mathfrak{D}', \mathbb{C})$ where $\mathfrak{D}'$ satisfies the following statements*:

1. *Effectiveness: (a) for every φ in dRequest, $\mathfrak{D}'$ contains no atom isomorphic to φ, (b) $R\_core(\mathfrak{D}') = \mathfrak{D}'$, and (c) $\mathfrak{D}' \models \mathbb{C}$.*
2. *Monotonicity: $\mathfrak{D}' \sqsubseteq \mathfrak{D}$.*

Proof See ("Appendix B"). □

The following example shows that our approach does not satisfy minimal change in the sense that deletions might be performed with fewer changes.

Example 6.1 Consider the set $\mathbb{C}$ containing the following two constraints

– $c_1 : P^-(x_1, y_1), Q(y_1, z_1) \Rightarrow R(x_1, z_1)$
– $c_2 : Q(x_2, y_2) \Rightarrow S(x_2, t_2)$

along with $\Delta = (\mathfrak{D}, \mathbb{C})$ where $\mathfrak{D} = \{P(a, b), Q(b, c), R(a, c), S(b, N_1)\}$. For $\delta_{\max} = 1$ and $\mathsf{dRequest} = \{R(a, c), S(b, N)\}$, according to Algorithm 2, we have first $ToDel = \{R(a, c), S(b, N_1)\}$, implying that $T^*_{\mathsf{fw}}(\mathfrak{D} \backslash ToDel) = \{P(a, b), Q(b, c), R(a, c), S(b, N_2^0)\}$. In other words, $\mathfrak{D}_1$ is equal to $\mathfrak{D}$ up to a renaming of N_2. Hence, $R\_core(\mathfrak{D}_2) = \mathfrak{D}_2$, showing that $Disallowed = \emptyset$ and that $Back = \{R(a, c), S(b, N_2^0)\}$. Thus, $T^*_{\mathsf{bw}}(Disallowed \cup Back, \mathfrak{D}_1) = \{R(a, c), S(b, N_2^0), P(a, b), Q(b, c)\}$ and so $\mathfrak{D}' = \emptyset$.

However, it is easy to see that the deletion of $P(a, b)$ is not necessary. Indeed, $\Delta'' = (\mathfrak{D}'', \mathbb{C})$ where $\mathfrak{D}'' = \{P(a, b)\}$ is an acceptable result of the deletion because atoms in $\mathsf{dRequest}$ are not in $\mathfrak{D}''$, $\mathfrak{D}'' \models \mathbb{C}$ and $R\_core(\mathfrak{D}'') = \mathfrak{D}''$. □

The following proposition shows nevertheless that when the bodies of constraints contain one atom, our approach to deletions *does* satisfy minimal change.

Proposition 6.3 *Let $\Delta = (\mathfrak{D}, \mathbb{C})$ be a database such that all constraints in $\mathbb{C}$ have one literal in their body, and $\mathsf{dRequest}$ a finite set of atoms. Let $\Delta' = (\mathfrak{D}', \mathbb{C})$ be the database returned by the call of Algorithm 2 with Δ and $\mathsf{dRequest}$ as input. Then, for every φ in $((\mathfrak{D} \backslash ToDel) \backslash \mathfrak{D}')$, $\mathfrak{D}' \cup \{\varphi\} \not\models \mathbb{C}$.*

Proof Since φ is in $((\mathfrak{D} \backslash ToDel) \backslash \mathfrak{D}')$, line 11 in Algorithm 2 has been run, and φ is in $T^*_{\mathsf{bw}}(Disallowed \cup Back, \mathfrak{D}_1)$ and in $\mathfrak{D}$. By definition of T_{bw}, there exist $k > 0$, $c : B(\mathbf{X}, \mathbf{Y}) \Rightarrow L(\mathbf{X}, \mathbf{Z})$ in $\mathbb{C}$ and h such that $\varphi = h(B(\mathbf{X}, \mathbf{Y})) = B(\alpha, \beta)$, $h(B(\mathbf{X}, \mathbf{Y})) \in (\mathfrak{D} \backslash T^k_{\mathsf{bw}}(Disallowed \cup Back, \mathfrak{D}_1))$ and all atoms in $\mathfrak{D}$ of the form $L(\alpha, \_)$ are in $T^k_{\mathsf{bw}}(Disallowed \cup Back, \mathfrak{D}_1)$. As a consequence, $\mathfrak{D}_2 \cup \{\varphi\}$ contains $h(body(c))$ but no atom of the form $L(\alpha, \_)$, and so, $\mathfrak{D}_2 \cup \{\varphi\} \not\models \mathbb{C}$.

Let us now assume that $\mathfrak{D}' \cup \{\varphi\}$, i.e. $R\_core(\mathfrak{D}_2) \cup \{\varphi\}$, satisfies $\mathbb{C}$. Since $R\_core(\mathfrak{D}_2) \subseteq \mathfrak{D}_2$, $R\_core(\mathfrak{D}_2) \cup \{\varphi\}$ contains no atom of the form $L(\alpha, \_)$, and thus, our assumption entails that $R\_core(\mathfrak{D}_2) \cup \{\varphi\}$ contains no atom of the form $B(\alpha, \_)$. This is a contradiction to the fact that φ is such an atom that clearly belongs to $R\_core(\mathfrak{D}_2) \cup \{\varphi\}$. Therefore, the proof is complete. □

7 Chasing versions and core computation

This section first offers a discussion about our chase approach, with regard to different chase versions available in the literature, and then provides details on the core computation, which is the most complex step in our algorithms.

7.1 Chasing: our choice of controlling null propagation

It is well known that when chasing an instance with respect to TGDs, the process might not terminate due to the specific form of the constraints; Example 1.4 shows such a case. The usual way to address this issue is to identify those rules that might lead to a non-terminating chase so as to discard them. However, as the problem is known to be *undecidable*, sufficient conditions ensuring termination have been proposed in the literature [45]. In our approach, we avoid such restrictions through the introduction of the parameter $\delta_{\max}$, thus offering the possibility of dealing with any kind of constraints while avoiding infinite processing.

More precisely, we show that: (i) for computations where δ_{max} is not reached, our chasing operator T_{fw} corresponds to a well-defined chase semantics and (ii) for situations where constraints meet the conditions ensuring chase termination, we can determine δ_{max} so that all null degrees are less than δ_{max}.

Positioning our Chase Procedure We first recall from Onet [45] that various chasing versions have been proposed in the literature, under the names of standard, oblivious, semi-oblivious and core chase. We show here that our chasing approach as defined through the operator T_{fw} is closely related to the standard-chase and the core-chase procedures. To this end, we define a new operator T as follows, for every set I of instantiated atoms:

$$\begin{aligned} T(I) = I \cup \{h(L(\mathbf{X},\mathbf{Z})) \mid\ & (\exists c : B(\mathbf{X},\mathbf{Y}) \Rightarrow L(\mathbf{X},\mathbf{Z}) \in \mathbb{C})(\exists h) \\ & ((c \text{ is full}) \wedge (h(body(c)) \subseteq I))\} \\ \cup \{h'(L(\mathbf{X},\mathbf{Z})) \mid\ & (\exists c : B(\mathbf{X},\mathbf{Y}) \Rightarrow L(\mathbf{X},\mathbf{Z}) \in \mathbb{C})(\exists h) \\ & ((c \text{ is not full}) \wedge (h(body(c)) \subseteq I) \wedge \\ & (\forall h'')((h''(body(c)) = h(body(c))) \Rightarrow \\ & h''(L(\mathbf{X},\mathbf{Z})) \notin I))\} \end{aligned}$$

Roughly speaking, T is a simplified version of T_{fw} in which all considerations about null degrees are omitted. In other words, T can be seen as a 'parallelized' version (i.e. constraints are applied in a breadth first manner) of a step of the *standard-chase procedure*, where constraints are applied in a depth first manner [45]. Thus, when standard chase terminates, the following sequence:

- $T^0 = I$
- for every $k > 0$, $T^k = T(T^{k-1})$

has a limit, denoted by $T^*(I)$, which is precisely the result of the standard-chase procedure applied to I.

On the other hand, another variant of the chase procedure is known as *core chase procedure* and can be defined as follows. For every I, let $T_c(I) = core(T(I))$ and let $core\_chase(I)$ be the limit of the sequence defined by

- $T_c^0 = I$
- for every $k > 0$, $T_c^k = T_c(T_c^{k-1})$

when this limit exists. If the limit doest not exist, $core\_chase(I)$ is set to $\perp$.

The relation between the two versions of the chase procedure as set in Fagin et al. [15] shows that when standard chase terminates, $core\_chase(I) = core(T^*(I))$. Thus, the following proposition holds.

Proposition 7.1 *Let I be a set of instantiated atoms such that the computation of $T^*_{fw}(I)$ yields no nulls whose degree is greater than or equal to $\delta_{\max}$. Then, $core\_chase(I) \neq \perp$ and $core\_chase(I) = core(T^*_{fw}(I))$.*

Proof Since the computation of $T^*_{\mathsf{fw}}(I)$ yields no nulls whose degree is greater than or equal to $\delta_{\max}$, we have $T^*_{\mathsf{fw}}(I) = T^*(I)$ and thus $core\_chase(I) = core(T^*(I)) = core(T^*_{\mathsf{fw}}(I))$, and the proof is complete. □

Chase Termination Conditions and $\delta_{\max}$ We recall that the threshold $\delta_{\max}$ has been introduced to cope with two important issues when dealing with TGDs in practice:

1. *Avoid non-terminating computations of $T_{\mathsf{fw}}(\mathfrak{D})$.* This issue is the most important one because it is known that termination of the chase procedure is *non-decidable* in general

(see [45]). In the literature, sufficient conditions for termination of the chase procedure have been proposed, and in this section, we relate one of these suffisent conditions to the value of $\delta_{\max}$.

2. *Avoid any non-suitable spreading of nulls throughout the database instance.* Even when the constraints are such that termination can be ensured, it is likely that an instance containing a huge number of nulls produced by cascading constraint applications is of very limited interest.

Focussing on the first item above, we now recall the notion of weak acyclicity of the dependency graph of $\mathbb{C}$ as introduced in Fagin et al. [15]. The dependency graph of $\mathbb{C}$, denoted by $DG(\mathbb{C})$, is a labelled directed graph defined as follows:

- the vertices of $DG(\mathbb{C})$ are all pairs (P, i) where P is an n-ary predicate occurring in a constraint c of $\mathbb{C}$ and i is an integer such that $1 \leq i \leq n$ representing a position in the argument of P.
- an edge occurs between vertices (P, i) and (Q, j) if there exists a constraint $c : B(\mathbf{X}, \mathbf{Y}) \Rightarrow L(\mathbf{X}, \mathbf{Z})$ in $\mathbb{C}$ such that $body(c)$ contains an atom over P for which position i is a variable x in $\mathbf{X}$, Q is the predicate L and one of the following holds:
  (a) x occurs in $L(\mathbf{X}, \mathbf{Z})$ at position j,
  (b) there exists a variable z in $\mathbf{Z}$ occurring at position j in $L(\mathbf{X}, \mathbf{Z})$.

In case (a), the edge is said to be *universal* and in case (b) the edge is said to be *existential*. The graph $DG(\mathbb{C})$ is said to be *weakly acyclic* if it contains no cycle involving an existential edge.

It has been shown in Fagin et al. [15] that when the dependency graph is weakly acyclic, standard chase always terminates. Using this result in our context, we show the following proposition.

Proposition 7.2 *Let $\mathbb{C}$ be such that $DG(\mathbb{C})$ is weakly acyclic and K be the maximum number of existential edges occurring in a path of $DG(\mathbb{C})$. If $\delta_{\max} > K$, then the computation of $T^*_{fw}(I)$ yields no nulls whose degree is greater than or equal to $\delta_{\max}$.*

Proof By definition of T_{fw}, when $\delta(N)$ is set to k, this means that there exists a path in $DG(\mathbb{C})$ involving k existential edges. Thus, the result follows. □

As a consequence of Propositions 7.1 and 7.2, it turns out that, in our approach, when the dependency graph of $\mathbb{C}$ is weakly acyclic, $\delta_{\max}$ can be chosen so as all insertions are accepted, producing the same instance as if the core-chase procedure were applied. On the other hand, we recall that sets of constraints $\mathbb{C}$ whose dependency graph is *not* weakly acyclic are accepted in our approach, at the cost of rejecting some insertions.

7.2 Computing the core

As earlier mentioned, computing cores is necessary in our approach to keep the database instance as small as possible and to avoid the presence of useless, and thus confusing, nulls in the database. In what follows, we give details about core computations and its complexity.

Given a set I of instantiated atoms, the computation of $R\_core(I)$ is based on a partition $\Pi(I)$ of I whose construction can be explained based on the so-called *Gaifman graph of the nulls of I* (see [16]). This graph, denoted by $G(I)$, is defined as follows:

- the vertices of $G(I)$ are all the nulls occurring in I,

– (N, N') is an edge of $G(I)$ if N and N' both occur in one atom in I.

The connected components of $G(I)$, say $G_1, \ldots, G_n$, allow to partition I into $n+p$ pairwise disjoint subsets $I_1, \ldots, I_n, I_{n+1}, \ldots, I_{n+p}$ where, for $j = 1, \ldots, n$, I_j contains all atoms in I in which nulls in G_j occur and where $I_{n+1}, \ldots, I_{n+p}$ are all pairwise distinct singletons, each of them containing an atom in I with no nulls. Then, denoting by $\Pi(I)$, respectively, $\overline{\Pi}(I)$, the set $\{I_1, \ldots, I_n\}$, respectively, $\{I_1, \ldots, I_n, I_{n+1}, \ldots, I_{n+p}\}$, Algorithm 3 shows how $R\_core(I)$ is computed. This algorithm can be seen as a simplification of the one given in Fagin et al. [16], because the chase computation is not part of our algorithm. Notice, however, that the computation of the homomorphism is given in more details than in Fagin et al. [16], in order to better explain the optimizations we focus on.

Algorithm 3: Compute_R_Core

Input: A set I of instantiated atoms, the associated partition $\Pi(I)$ and a maximal degree δ_{max}.
Output: $R\_core(I)$ (the last obtained Γ) and the modified partition $\Pi(R\_core(I))$.
1: **for all** π in $\Pi(I)$ **do**
2: Compute $hom(\pi) = \{h \mid h(\pi) \neq \pi \wedge h(\pi) \subseteq I \wedge h \ is \ restricted\}$
3: $\Gamma := I$
4: $H := \{hom(\pi) \mid \pi \in \Pi(I) \wedge hom(\pi) \neq \emptyset\}$
5: **while** $H \neq \emptyset$ **do**
6: Choose an element $hom(\pi)$ in H
7: **if** exists h in $hom(\pi)$ such that $h(\pi) \subseteq \Gamma$ **then**
8: Choose one such h in $hom(\pi)$
9: $\Gamma := (\Gamma \backslash \pi) \cup h(\pi)$
10: $H := H \backslash \{hom(\pi)\}$
11: $\Pi := \{\pi \cap \Gamma \mid \pi \in \Pi(I) \wedge (\pi \cap \Gamma) \neq \emptyset\}$
12: **return** Γ, Π

It is important to note that the loop on line 1 can be computed in parallel because any two sets in $\Pi(I)$ share no nulls. Such parallel processing would thus result in a fast global computation, in particular when the sizes of the sets π are small.

However, the loop on line 5 cannot be computed by just choosing one homomorphism in each set $hom(\pi)$ and combining these chosen homomorphisms into one global homomorphism. To see this, consider $I = \{P(a, N_1), P(a, N_2)\}$ where for the sake of simplification no disallowed nulls occur, implying that null degrees are omitted and that all homomorphisms are restricted. We have $\Pi(I) = \overline{\Pi}(I) = \{\{P(a, N_1)\}, \{P(a, N_2)\}\}$ and h_1 and h_2 such that $h_1(N_1) = N_2$ and $h_2(N_2) = N_1$ are found on line 2, possibly in parallel. Thus, one global homomorphism would simply exchange the nulls, and thus, I would not be changed. This is not correct because, as found by Algorithm 3, $R\_core(I)$ is $\{P(a, N_1)\}$ (or equivalently $\{P(a, N_2)\}$). The following example illustrates Algorithm 3 in a less simple case, where again, no disallowed nulls are present and thus where null degrees are omitted.

Example 7.1 Assuming that all homomorphisms in this example are restricted, let $I = \{P(a), P(N_1), Q(a, b), Q(a, N_2), R(a, N_3), S(a, b, N_3), S(a, N_2, N_3)\}$. Then, $\Pi(I) = \{\pi_1, \pi_2\}$ where $\pi_1 = \{P(N_1)\}$ and $\pi_2 = \{Q(a, N_2), R(a, N_3), S(a, b, N_3), S(a, N_2, N_3)\}$, whereas $\overline{\Pi}(I) = \Pi(I) \cup \{\{P(a)\}, \{Q(a, b)\}\}$.

On line 2 of Algorithm 3, it is found that $hom(\pi_1) = \{h_1\}$ where h_1 is defined by $h_1(N_1) = a$ and $hom(\pi_2) = \{h_2\}$ where h_2 is defined by $h_2(N_2) = b$ and $h_2(N_3) = N_3$. In the loop line 5, choosing π_1 and then π_2, Γ is first set to $(I \backslash \{P(N_1)\}) \cup \{P(a)\} = (I \backslash \{P(N_1)\})$, and then changed to $(\Gamma \backslash \{Q(a, N_2), R(a, N_3), S(a, b, N_3), S(a, N_2, N_3)\}) \cup \{Q(a, b), R(a, N_3),$

$S(a, b, N_3)\}$. Hence, $R\_core(I) = \{P(a), Q(a, b), R(a, N_3), S(a, b, N_3)\}$, and as $\pi_1 \cap \Gamma = \emptyset$ and $\pi_2 \cap \Gamma = \{R(a, N_3), S(a, b, N_3)\}$, $\Pi(R\_core(I)) = \{\{R(a, N_3), S(a, b, N_3)\}\}$. □

Complexity It has been shown in Fagin et al. [20] that the complexity of Algorithm 3 is equal to that of computing the homomorphisms on line 2, which itself is in $O(|I|^b)$ where b is the maximum number of nulls occurring in the sets of I_j of $\Pi(I)$. We also recall from Gottlob [20] that, in our context, this complexity can be brought down to $O(|I|^{(b/2+2)})$.

8 Experimental study

We have implemented a prototype of our approach which can be uploaded together with all the details of our tests (see [10]). The implementation, written in JAVA, allows running all examples provided in the present paper and serves as a proof of concepts to our updating methods. Even if the mentioned implementation deals with facts stored in main memory, it allows us to offer a preliminary study on the trade-off between the gain of expression power and efficiency in updating. Our tests were run on an Intel(R) Core(TM) i7-6600U CPU 2.60GHz x 4; 16 GB of RAM. Time results correspond to the average time of 5 executions of each test.

8.1 Synthetic example

As a preliminary test, we run our algorithms on a synthetic example, reproducible according to instructions available in DBOrleans-Team [10]. To provide a fair idea of the impact of allowing more complex constraints in consistency maintenance of a database, we propose the following evolving testing scenario:

1. The initial database instance $\mathfrak{D}$ is fixed, with a fixed set of instantiated atoms (nulls may exist) over 525 distinct predicates.
2. Each test consists in performing a fixed number of updates (K) on the initial instance $\mathfrak{D}$.
3. Three different sets of constraints, denoted by $\mathbb{C}_{0np}$, $\mathbb{C}_{1np}$ and $\mathbb{C}_{2np}$, give rise to three different tests. The difference between these tests is the increasing number of side effects, i.e. each test is performed on a set of constraints which evolves in the following way:

   $\mathbb{C}_{0np}$ In this first test, there is no null propagation, i.e. a constraint generates a null-atom, but this new atom cannot match the body of another constraint. More precisely, each constraint has the form $A_i(x, y) \Rightarrow B_i(y, z)$ for $i \in [1, K]$. This scenario corresponds to the restrictions imposed on constraints in Alves et al. [25].

   $\mathbb{C}_{1np}$ In the second test, each generated null propagates generating a new side effect. The set of constraints is composed by subsets containing two constraints of the form $A_i(x, y) \Rightarrow B_i(y, z)$ and $B_i(x, y) \Rightarrow C_i(x, y, z)$ for $i \in [1, K]$.

   $\mathbb{C}_{2np}$ In the third test, the set of constraints is composed by subsets containing three constraints of the form $A_i(x, y) \Rightarrow B_i(y, z)$, $B_i(x, y) \Rightarrow C_i(x, y, z)$ and $C_i(x, y, z) \Rightarrow D_i(x, y, z, u)$ for $i \in [1, K]$.

   Tests are built by increasing the side effect chain triggered by an atom A_i. Moreover, each new generated side effect corresponds to an atom with a new null, containing as well the nulls generated by the previous constraint of the chain. For instance, in $\mathbb{C}_{1np}$,

the insertion of $A_1(a, b)$ generates $B_1(a, N_1)$ and $C_1(a, N_1, N_2)$. In $\mathbb{C}_{2np}$, this insertion will generate $B_1(a, N_1)$, $C_1(a, N_1, N_2)$ and $D_1(a, N_1, N_2, N_3)$.

4. Each insertion is a set of facts on predicates A_i for $i \in [1, K]$. The constants appearing in such facts are randomly chosen from the active domain of the database instance together with new nulls. Similarly, a deletion is a set of facts on the predicate appearing at the end of the side effect chain considered in the corresponding test, e.g. for the second test, deletions are on predicate C_i.
5. Besides constraints with only one atom in the body, we also offer some tests with constraints having two atoms in the body. Three tests are performed in this scenario:

   $\mathbb{C}^{2B}_{0np}$ In this test, the set of constraints is composed of constraints of the form $A_i(x, y), A_{i+1}(x, z) \Rightarrow AA_i(x, z)$.

   $\mathbb{C}^{2B}_{1np}$ In this test, the set of constraints is composed of subsets containing two constraints, one of the form $A_i(x, y), A_{i+1}(x, z) \Rightarrow AA_i(x, z)$ and the other of the form $A_i(x, y), A_{i+1}(y, z) \Rightarrow AB_i(x, u)$ for $i \in [1, K-1]$.

   $\mathbb{C}^{2B}_{2np}$ In this third test, In this test, the set of constraints is composed of subsets containing the two constraints of $\mathbb{C}^{2B}_{1np}$ together with a third constraint of the form $AA_i(x, y), AB_i(x, z) \Rightarrow AC_i(x, z, u)$.

   In this scenario, insertions are sets containing instantiated atoms on the predicates A_i and A_{i+1}.

Our tests offer a preliminary evaluation of how update performance is impacted by the use of complex constraints such as TGDs.

(A) The first impact is, naturally, the possibility of performing an *automatic* verification for applications requiring such complex constraints.

(B) The second impact concerns the time needed for processing updates. We analyse time results for insertions and deletions separately, on the basis of the scenario described above: starting with a very simple case, where each update triggers a single constraint and has one side effect, we continue, as explained above, by increasing the complexity of our constraints, the null propagation and the number of generated nulls.

(B.1) Table 2 shows our results for insertions. Insertions are done according to the explanations given in item 4 and, thus, concern the worst case where all constraints are triggered. The results are promising: the gain of expression power does not compromise the performance. To better analyse the situation, we compare the results obtained for $\mathbb{C}_{0np}$ with the other results in Table 2:

(a) From $\mathbb{C}_{0np}$ to $\mathbb{C}_{1np}$: while the number of triggered constraints is doubled, the increase of total time for the insertion of 75 atoms is only of 6.4%. Similarly, from $\mathbb{C}_{0np}$ to $\mathbb{C}_{2np}$, while the number of triggered constraints is tripled the increase of the total insertion time is only of 1.7%.

(b) Experiments with constraints with more than one atom in the body (lines 4–6 of Table 2) show that the insertion time increases, respectively, by 0.4%, 4.7% and 13.7%, when compared to $\mathbb{C}_{0np}$.

(c) Unsurprisingly, the core is the most time-expensive part of our algorithm, responding, here, for more than 80% of the total computation time. Inserting a set of atoms with linked null values in an instance with an important number of null values renders the search for instantiations expensive, because the algorithm looks for instantiating partitions and not only one atom. In our synthetic database, 44% of the instantiated atoms have null values. Section 8.3 proposes some optimizations which should soften this problem.

Moreover, the core computation deserves some attention, even if it is independent of constraint complexity. In our examples, the forward operator always produces atoms with nulls.[2] The core procedure tries to instantiate these atoms, but as the original database instance has 44% of its atoms with null values, such instantiations are rare. The situation is illustrated by the results on column $|\mathfrak{D}'|$ of Table 2 which show that all computed side effects are inserted in the database, meaning that no core reduction is possible. However, to complete the example, consider now a different insertion scenario:

- Take the set of constraints and database instance of line 3 in Table 2.
- Consider iRequest as containing the 75 atoms as before plus 25 *facts* corresponding to instantiations of atoms appearing in the database instance and in the head of some constraints.

In this new scenario, our results are: $|T^*_{\mathsf{fw}}(\mathfrak{D} \cup \mathsf{iRequest})| = 243$, $|\mathfrak{D}'| = 49,888$, $t_{T^*_{\mathsf{fw}}} = 0.197s$ and $t_{Core} = 2.256s$. In this case, the core eliminates 75 atoms (sine $(243+49,710) - 49,888 = 75$), thus resulting in a significant simplification.

(B.2) Table 3 shows our results for deletions. The tests consider only the worst case of deletion, i.e. when $Back$ is not empty, (line 12 of Algorithm 2) which requires computations involving the operator T_{bw} and the restricted core.

Deleting atoms with a null value is much more expensive than inserting them, since the deletion requires a database traversal to determine all possible null instantiations while the insertion needs just one instantiation. For instance, considering a database instance $\mathfrak{D} = \{A(a, N), B(a, c), B(a, b), B(a, d)\}$, the constraint $B(x, y) \Rightarrow A(x, z)$, and the deletion of $A(a, N_1)$, the algorithm has to find all the instantiations of $B(a, \_)$ in the database instance. This operation is expensive in the current implementation because data are stored in regular files. This should be significantly improved in the new version (under construction) where data are stored using a database management system (DBMS). In this case, thanks to the optimized query processing in any DBMS, the instantiations of $B(a, \_)$ can be efficiently retrieved. Notice that a similar situation is described in Halfeld Ferrari et al. [25] and Halfeld Ferrari [27], where only constraints as those in $\mathbb{C}_{0np}$ are considered.

Table 3 shows that, once again, even if our tests concern only the worst case, the results are promising, because the gain of expression power does not compromise the performance.

8.2 URBS example

Recalling that our approach is well adapted to RDF data sources, we now illustrate such an application. As explained in Halfeld Ferrari and Laurent [28], our constraints can be settled to express RDF/S semantic constraints. For instance, in Halfeld Ferrari and Laurent [28] and Halfeld Ferrari et al. [27], this is done by using the formalism of Flouris et al. [19] and classifying predicates into two sets: one concerning schema and another concerning instances. Then, general constraints of RDF/S are written.

In this paper, we test our approach over the RDF data set used for tests in Halfeld Ferrari et al. [25], which had been generated by importing data from the Curitiba Urbanization Company (URBS). We recall that Curitiba (Brazil) is known by its mass transport corridors and mobility solutions using bus rapid transit (BRT) systems since the 1970s [48]. The complete system, according to Curitiba's Institute of Research and Urban Planning (IPPUC), includes about 482 routes, distributed among 9940 bus stops, 23 bus terminals and 17 categories of streets

[2] Recall that our forward operator (Definition 4.1) does not compute atoms which are already in the database instance.

Table 1 Notation used for results of the synthetic tests

Notation	Meaning
$\lvert\mathfrak{D}\rvert$	Database size, i.e. number of instantiated atoms before updates
$\lvert\mathfrak{D}'\rvert$	New database size , i.e. number of instantiated atoms after updates
$\lvert\mathsf{iRequest}\rvert$	Number of required insertions
$\lvert\mathbb{C}\rvert$	Number of constraints
$\lvert T^*_{\mathsf{fw}}(\mathfrak{D} \cup \mathsf{iRequest})\rvert$	Number of atoms generated by the forward operator
$\lvert T^*_{\mathsf{bw}}((Disallowed \cup Back), \mathfrak{D}_1)\rvert$	Number of atoms generated by the backward operator on line 11 of Algorithm 2
$t_{T^*_{\mathsf{fw}}}$	Time for computing side effects by applying the forward operator
t_{Core}	Time for computing the core
t_{total}	Total time for the insertion of iRequest

Table 2 Results for insertions: examples consider different constraint sets on a database instance $\mathfrak{D}$ such that $\lvert\mathfrak{D}\rvert = 49{,}710$. The set iRequest is fixed and contains 75 instantiated atoms. Results are presented using the notation shown in Table 1

	Set $\mathbb{C}$	$\lvert\mathbb{C}\rvert$	$\lvert T^*_{\mathsf{fw}}(\mathfrak{D} \cup \mathsf{iRequest})\rvert$	$\lvert\mathfrak{D}'\rvert$	$t_{T^*_{\mathsf{fw}}}$ (s)	t_{Core} (s)	t_{total} (s)
1	$\mathbb{C}_{0np}$	75	109	49,819	0.1138	2.22	2.3338
2	$\mathbb{C}_{1np}$	150	143	49,853	0.1822	2.3002	2.4824
3	$\mathbb{C}_{2np}$	225	177	49,887	0.2148	2.1638	2.3786
4	$\mathbb{C}^{2B}_{0np}$	75	205	49,915	0.1908	2.1526	2.3434
5	$\mathbb{C}^{2B}_{1np}$	150	209	49,919	0.3424	2.1034	2.4458
6	$\mathbb{C}^{2B}_{2np}$	225	213	49,923	0.5698	2.0842	2.654
7	$\mathbb{C}_{2np} \cup \mathbb{C}^{2B}_{2np}$	450	315	50,025	0.7558	2.188	2.9436

Table 3 Results for deletions: examples consider different constraint sets on a database instance $\mathfrak{D}$ such that $\lvert\mathfrak{D}\rvert = 49,710$. The set dRequest is fixed and contains 10 instantiated atoms. Results are presented using the notation shown in Table 1

	Set $\mathbb{C}$	$\lvert\mathbb{C}\rvert$	$\lvert T^*_{\mathsf{bw}}((Disallowed \cup Back), \mathfrak{D}_1)\rvert$	t_{total} (s)
1	$\mathbb{C}_{0np}$	75	20	3.3892
2	$\mathbb{C}_{1np}$	150	55	3.5998
3	$\mathbb{C}_{2np}$	225	43	3.779
4	$\mathbb{C}^{2B}_{0np}$	75	20	3.6824
5	$\mathbb{C}^{2B}_{1np}$	150	20	3.8172
6	$\mathbb{C}^{2B}_{2np}$	225	30	4.1518
7	$\mathbb{C}_{2np} \cup \mathbb{C}^{2B}_{2np}$	450	30	4.3212

Table 4 Constraints for the RDF database concerning Curitiba Urbanization Company

Constraints	
c_1	$MiniBus(XX) \Rightarrow CardOnly(XX)$
c_2	$Expressline(XX) \Rightarrow StartStop(XX, XY)$
c_3	$ExpressLineStop(XX) \Rightarrow FastBoarding(XX)$
c_4	$ExpressVehicle(XX, XY) \Rightarrow Expressline(XX)$
c_5	$ExpressLineStop(XX, XY) \Rightarrow ExpressStop(XY)$
c_6	$ExpressLineStop(XX, XY) \Rightarrow ExpressLine(XX)$
c_7	$MADRUGUEIRO(XX), NightLine(XY, XX) \Rightarrow ExpressLineStop(XX, XY)$
c_8	$Alimentador(XY, XX) \Rightarrow AlocaExpresso(XX, XZ)$
c_9	$AlocaExpreso(XX, XY) \Rightarrow ExpressLineStop(XX, XZ)$
c_{10}	$ConventionalLineStop(XX, XY), ExpressLineStop(XZ, XX) \Rightarrow NovoMobiliario(XY, XV)$
c_{11}	$DowntownLineStop(XY, XX) \Rightarrow DowntownVehicle(XZ, XX)$

Table 5 Results for insertions in the RDF database URBS: examples consider different constraint sets on a database instance $\mathfrak{D}$ such that $|\mathfrak{D}| = 115{,}910$. The set iRequest is fixed and contains 10 instantiated atoms. Results are presented using the notation shown in Table 1

Set $\mathbb{C}$	$\lvert\mathbb{C}\rvert$	$\lvert T^*_{\mathsf{fw}}(\mathfrak{D} \cup \mathsf{iRequest})\rvert$	$\lvert\mathfrak{D}'\rvert$	$t_{T^*_{\mathsf{fw}}}$ (s)	t_{Core} (s)	t_{total} (s)
$\mathbb{C}_1$	7	27	115,937	23.4146	0.128	23.5334
$\mathbb{C}_2$	12	27	115,937	72.000	0.73	72.0735

[35]. Different types of routes, such as express routes, inter-district and local lines, or feeder, are implemented.

In this paper, each RDF triple of the form (sPo) (i.e. subject–predicate–object) is translated into an atom of the form $P(s, o)$. Notice also that we do not use special predicates as in Flouris et al. [19], Halfeld Ferrari and Laurent [28] and Halfeld Ferrari et al. [25] (such as PI, standing for property instance, or CI, standing for class instance). Instead, we perform a translation in order to obtain only atoms on application predicates. For example, the instance of property ExpressLineStop, expressed in RDF as $PI(a, b,$ ExpressLineStop$)$, is translated into ExpressLineStop(a, b). Our choice is only motivated by performance issues. Indeed, considering predicates PI and CI, finding, for instance, that $PI(a, N_1,$ ExpressLineStop$)$ and $PI(a, b,$ ExpressLineStop$)$ unify would require testing all atoms on PI, whereas in our implementation, only atoms on ExpressLineStop are tested.

Table 5 shows the result of our tests on URBS data. Based on Table 4, two sets of constraints are considered, namely $\mathbb{C}_1 = \{c_1, c_2, c_3, c_4, c_5, c_6\}$ and $\mathbb{C}_2 = \mathbb{C}_1 \cup \{c_7, c_8, c_9, c_{10}, c_{11}\}$. The set $\mathbb{C}_1$ contains simple constraints, that is, constraints with no null propagation, as in Halfeld Ferrari et al. [25]. The set $\mathbb{C}_2$ is obtained by adding more sophisticated constraints to the previous set. This new set contains constraints with more than one atom in the body (c_7 and c_{10}) and allows for null propagation. In this way, it is possible to express constraints establishing, for instance, that we should allocate express lines (c_9) to join lines doing connections between districts and the town terminals (c_8). The iteration between constraints c_8 and c_9, not possible in Halfeld Ferrari et al. [25], is allowed in the present approach, thus representing an important gain in the expression power of constraints.

8.3 Possible optimizations

Our experiments have shown that the core computations may become expensive. This is why we now discuss possible optimizations of the core computation in our current implementation, based on the fact that given $\Delta = (\mathfrak{D}, \mathbb{C})$, we have that $R\_core(\mathfrak{D}) = \mathfrak{D}$.

In the case of an insertion, as can be seen from Algorithm 1, one core computation is necessary for the set $T^*_{\mathsf{fw}}(\mathfrak{D} \cup \mathsf{iRequest})$, which we denote as $\mathfrak{D} \cup I$ where I contains the atoms in iRequest along with the side effects of the insertion. We notice that, although $\mathfrak{D}$ and iRequest have no nulls in common, I may contain nulls occurring in $\mathfrak{D}$. As a consequence, the partition $\Pi(\mathfrak{D} \cup I)$ is not equal to $\Pi(\mathfrak{D}) \cup \Pi(I)$. However, assuming that $\Pi(\mathfrak{D})$ is available, the computation of $\Pi(\mathfrak{D} \cup I)$ can be optimized, as compared with a computation from scratch. This is so because every partition block associated with a connected component of $G(\mathfrak{D} \cup I)$ that occurs in $G(\mathfrak{D})$ has not to be modified.

Assuming that $\Pi(\mathfrak{D} \cup I)$ has been computed, and referring to line 2 of Algorithm 3, since we have $R\_core(\mathfrak{D}) = \mathfrak{D}$, if the block π of $\Pi(\mathfrak{D} \cup I)$ being considered was already in $\Pi(\mathfrak{D})$, then, clearly, a homomorphism exists only if one atom in π can be instantiated by one atom in I. Therefore, this narrows the search space for the computation of $hom(\pi)$.

Turning now to deletions, by Algorithm 2, two core computations are involved, and both address an instance that can be written as $\mathfrak{D} \backslash I$ where I is a subset of $\mathfrak{D}$. In this case, $\Pi(\mathfrak{D})$ has first to be modified so as to generate $\Pi(\mathfrak{D} \backslash I)$. This can be optimized because the only blocks π to be modified are such that $\pi \cap I \neq \emptyset$, in which case π is changed into $\pi \backslash I$. Every such block $\pi \backslash I$ that gets empty is removed and the others have to be checked because they might have to be split. To see this, let $\mathfrak{D} = \{P(N_1), Q(N_1, N_2), R(N_2)\}$ and $I = \{Q(N_1, N_2)\}$, in which case $\Pi(\mathfrak{D}) = \{\mathfrak{D}\}$. Then, the block of $\Pi(\mathfrak{D})$ is changed to $\{P(N_1), R(N_2)\}$ that has clearly to be split into two blocks.

Assuming that $\Pi(\mathfrak{D} \backslash I)$ has been computed, and referring to line 2 of Algorithm 3, the only blocks to process are those that do not belong to $\Pi(\mathfrak{D})$, which in general shortens the execution of the loop.

9 Related work

Semantic for nulls Incomplete data in the relational model have been the subject of different work. In Reiter [49], we find the proposal for a first-order interpretation of the null value together with query answering algorithms. At the same time, proposals on relational theory have emerged. In particular, the landmark paper [31] introduced the notion of Table (a relation containing nulls) and of representation system (detailing two of them considered of 'practical interest'). Assuming that a table contains only constants and variables (representing missing information), the first representation system deals with Codd Tables [9] where variables occur at most once. However, it has been noticed in Imielinski and Lipski Jr. [31] that 'no representation system based on Codd Tables can support join and projection at a time'. The second representation system is based on naive Tables, allowing different marked variables (or nulls) occurring more than once. It is shown in Imielinski and Lipski Jr. [31] that this system allows arbitrary conjunctive queries. A third representation system, based on the idea of conditional table and considered mainly of theoretical interest, is also described in Imielinski and Lipski Jr. [31]. Since then, considerable work has been done on querying incomplete databases as, for instance, in Grahne [22], Zaniolo [56], Fagin et al. [17] and Libkin [38].

Updating incomplete data Updates on incomplete databases have drawn less attention than queries. We mention (i) the work of Abiteboul and Grahne based on conditional tables (see [1]); (ii) the work in Fagin et al. [18] which discusses the fact that the result of an update is dependent upon the way the logical database is constructed; (iii) the survey in Winslett [55] which offers a clear distinction between model-based and formula-based approaches to updating logical theories.

However, the interest of updates on incomplete data sources is now stemming from needs on the web semantic domain [12]. Work, such as Nikolaou and Koubarakis [44], extends to the RDF world concepts introduced in Imielinski and Lipski Jr. [31] and explores SPARQL query evaluation. Moreover, when De Giacomo et al. [11] propose an ontology update management, it seems to revive previous proposals such as Halfeld Ferrari et al. [26]. Indeed, when dealing with different rule levels and when proposing semantics tolerant with inconsistencies coming from the intentional level, the relation with the *exceptions* proposed in Halfeld Ferrari et al. [26] is undeniable. The problem of updating an incomplete database also concerns *when* and *how* null values should be replaced by the new data coming with the updates. Part of the problem is connected to the core computation. In Halfeld Ferrari and Laurent [28], we dealt with a non-null database, situation which corresponds to the case $\delta_{max} = 0$. The differences between update and revision are another important aspect to be considered. (We refer to Hansson [29] for an overview.) These differences are the consequence of different views of the problem and influence the semantic of changes of each particular proposal.

Tuple-generating constraints (TGD) generalize the commonly used foreign key constraints, allowing to express restrictions found in web semantic or graph database domains. But considering TGD increases expressiveness at the cost of some difficulties. The first one concerns the computation of the semantics which has been addressed through a chase procedure (see [45] for a survey). Work, as in Fagin et al. [16], Gottlob [20], Fagin et al. [15] and Pichler and Skritek [47], considers the use of the chase procedure in data exchange. Furthermore, in an update context, the chase procedure is associated with the generation of side effects—imposing extra insertions or deletions (with respect to those required by the user) to preserve consistency. Clearly, constraints are expected not only to be inherently consistent (e.g. a set of constraints generating contradictory side effects for the same update u is not acceptable) but also to avoid contradicting the original intention of the user's update. (For example, if the user asks for the deletion of f, the insertion of f is not an acceptable side effect.) The existence of non-repairable transitions (i.e. with side effects that invalidate the required update) is the focus of Schewe and Thalheim [50] which establishes sufficient and necessary conditions of a set of constraints to work correctly. In Halfeld Ferrari et al. [26], we can find results on the detection of inconsistency in a set of more simple constraints, such as those used in Halfeld Ferrari and Laurent [28]. In both cases, the authors use paths on a graph (defined from constraints) to detect inconsistency. The theory of consistency enforcement in databases has been the subject of Link [39], Link and Schewe [40] and Schewe and Thalheim [51] as well. Another difficulty is that more expressive constraints represent a barrier to the update determinism. In Halfeld Ferrari and Laurent [28] and Halfeld Ferrari et al. [26], determinism is guaranteed because constraints have only one atom in the body and in the head, while in Flouris et al. [19], determinism is possible thanks to a total ordering, which imposes arbitrary choices.

In the current approach, the determinism of deletions is ensured by distinguishing *one* atom in the body of each constraint. Notice, however, that this could be generalized so as to allow the deletion of more that one atom. Our solution is a *practical* answer to an update scenario where users want to juggle with different possibilities, namely: (a) expressing constraints richer than those usually allowed by keys and foreign keys; (b) ensuring an automatic maintenance of

the database consistency; and (c) not wasting time in considering different possible new consistent database instances.

In fact, our solution corresponds to a very popular way of specifying updates in relational databases, known as 'active rules'. Indeed, a constraint such as $p^-, q \Rightarrow r$ can be seen as the active rule: `when` delete r, `if` p and q are in the database, `then` delete p. Active rules have been widely studied in the literature (e.g. in Schewe and Thalheim [50]), and it is well known that they do not satisfy some suitable properties such as convergence, independence from the application order (which is related to determinism), minimal change, monotonicity, etc. Clearly, there is a price to pay for having deterministic deletion as a practical solution for users.

Minimal Data Instance When applying the chase over TGD to generate new data, one should naturally consider *universal solutions* (those having homomorphisms into every possible solution). They are identified in Fagin et al. [15], while Fagin et al. [16] and Gottlob [20] consider the problem of finding the smallest one (the core). Updates are not considered, except in Pichler and Skritek [47], which studies complexity of the TGD checking problem, including its update variant.

Positioning our work Our approach adopts the FOL formalism introduced in Reiter [49] (which, in fact, is a different way of modelling naive tables) to deal with updates. We extend our previous work in Halfeld Ferrari and Laurent [28] and Halfeld Ferrari et al. [27] by dealing both with an incomplete database and with linear tuple-generating dependencies (TGD) without restrictions. Determinism of deletion is ensured by a pre-established choice giving priority to one side effect (marked in the constraint's body). Although, in our approach, the behaviour of a deletion depends on the non-existence of side effects invalidating it, the problem of determining such kind of behaviour in a set of constraints is out of the scope of this paper. We conjecture that the approaches proposed in Schewe and Thalheim [50] and Schewe and Thalheim [51] can be adapted to our context—together with their results concerning the conditions and the complexity of performing such tests. As updates yield their side effects which may involve nulls, we propose two ways of avoiding instances with too many nulls generated from other nulls, namely: the possibility of fixing a pre-defined depth for such null generation and the strategy of keeping the instance as small as possible by applying the core computation. In this context, our updates are performed by taking into account marked null values (which can be stored). Notice that our results can be related to work on repair checking (such as Afrati and Kolaitis [2]), because the updated database can be seen as a repair of an inconsistent database obtained by performing only updates in dRequest and iRequest. However, our minimal change conditions are not necessarily considered in those papers.

Our update strategy, applied to the RDF world, is different from proposals such as Ahmeti et al. [3], Chirkova and Fletcher [8], Gutierrez et al. [24] and Lösch et al. [42]. Although some of them discuss non-determinism, all these approaches consider constraints as inference rules (as in Gottlob et al. [21], Lausen et al. [36], Motik et al. [43] and Patel-Schneider [46]), whereas we deal with constraints in a traditional database viewpoint, as in Flouris et al. [19]. (But the latter does not deal with null values.) Constraints are taken into account in the context of RDF technologies such as ShEx [52], SPIN [33] and SHACL [34]. However, their focus is on schema and ours is on integrity constraints. Stardog [53] deals with constraints which are closer to ours. Finally, it is worth noting that we consider updates as changes in the world and not as a revision in our knowledge of the world [4,29,32]. However, as our model does not include any explicit representation of time, it is possible to relate results of our method to the core principles of Belief Revision. Indeed, our delete and insertion operations are similar

to contraction and revision in Belief Revision and guided by precepts corresponding to the principles of closure, success, inclusion, consistency and vacuity (see [4]).

10 Concluding remarks

In this paper, we present a formal semantics for updates of incomplete databases. Although this problem has motivated many research efforts during the last decades, no consensus exists. In proposing a generic framework based on FOL, it is expected that our approach can deal with many different contexts, including RDF data where blank nodes are the cause of many issues.

Indeed, new applications on linked data (such as on urban transportation system [7,14,25]) need to ensure the trust and the quality of information they give. In such a context, updating and consistency maintenance of RDF data is an increasingly required task. The update strategies and properties settled up in this paper contribute to the development of tools capable to deal with data evolution in RDF databases where nulls (blank nodes) are marked as semantically connected.

In our approach, updates are performed in a deterministic way, without any arbitrary choice, while preserving consistency with respect to TGDs and while ensuring that useless incompleteness has been discarded. One should possibly point as a drawback of our approach the fact that the complexity of our update processing is high in theory, due to core computations. However, our paper outlines possible optimizations that would lead to tractable implementation. Indeed, a prototype of our method has been implemented and is publicly available [10]. This initial in-memory version allows testing and is a proof of concepts setting the basis for the next step where the prototype will be enhanced to be able to deal with standard databases. Our future work includes enhancing the expression power of our constraints towards a more general framework.

Acknowledgements We thank our colleagues Carmem Hara, Nádia P. Kozievitch and Flávio Uber who kindly shared the use of RDF-URBS data, and our student Julien Revaud for his work on the current implementation. We also wish to thank the reviewers for their remarks and suggestions that lead to important improvements of a preliminary version of our work.

A Proof of Proposition 6.1

Proposition 6.1 *Let $\Delta = (\mathfrak{D}, \mathbb{C})$ be a database and* iRequest *a finite set of facts. If Algorithm 1 returns $\Delta' = (\mathfrak{D}', \mathbb{C})$ where $\mathfrak{D}' \neq \mathfrak{D}$, then $\mathfrak{D}'$ satisfies the following statements*:

1. *Effectiveness: (a) for every φ in* iRequest, *$\mathfrak{D}'$ contains an instance of φ, (b) $R\_core(\mathfrak{D}') = \mathfrak{D}'$ and (c) $\mathfrak{D}' \models \mathbb{C}$.*
2. *Monotonicity: $\mathfrak{D} \sqsubseteq \mathfrak{D}'$.*
3. *Minimal change: For every φ in $\mathfrak{D}'$ not in $\mathfrak{D}$ and not isomorphic to an atom in* iRequest, *$\mathfrak{D}' \backslash \{\varphi\} \not\models \mathbb{C}$.*

Proof 1(a) Let φ in iRequest. Then, φ is in $T^*_{\mathsf{fw}}(\mathfrak{D} \cup \mathsf{iRequest}) = \mathfrak{D}_1$ (line 3 of Algorithm 1). Thus, if φ is not in $\mathfrak{D}'$, then φ has been instantiated due to the computation of the restricted core on line 4 of Algorithm 1. Therefore, in this case $R\_core(\mathfrak{D}_1) = \mathfrak{D}'$ contains an instance of φ.

1(b) The fact that $R\_core(\mathfrak{D}') = \mathfrak{D}'$ is an obvious consequence of Algorithm 1.

1(c) If $\mathfrak{D}_1$ contains no null N such that $\delta(N) \geq k$, then $\mathfrak{D}' \models \mathbb{C}$ holds because, by Proposition 4.1, $\mathfrak{D}_1$ satisfies $\mathbb{C}$ and thus, by Lemma 3.1, so does its core which is equal to its restricted core due to Lemma 4.2(1).

Now, assume that $\mathfrak{D}_1$ contains at least one null N with $\delta(N) \geq \delta_{max}$ and that $\mathfrak{D}' \not\models \mathbb{C}$. This means that there exists $c : B(\mathbf{X}, \mathbf{Y}) \Rightarrow L(\mathbf{X}, \mathbf{Z})$ in $\mathbb{C}$ such that $\mathfrak{D}' \not\models c$. Thus, there exists a homomorphism h such that $h(body(c)) \subseteq \mathfrak{D}'$ and, for any extension h' of h to the variables occurring in $\mathbf{Z}$, $h'(head(c))$ is not in $\mathfrak{D}'$. Denoting $h(body(c))$ by $B(\alpha, \beta)$, this means that $B(\alpha, \beta) \subseteq \mathfrak{D}'$ and that $\mathfrak{D}'$ contains no atom of the form $L(\alpha, \_)$. Since $\mathfrak{D}' \subseteq \mathfrak{D}_1$, we have $h(body(c)) \subseteq \mathfrak{D}_1$. As for every null N in $h(body(c))$, $\delta(N) < \delta_{max}$, $\mathfrak{D}_1$ contains an atom of the form $L(\alpha, \nu)$ where ν is made of nulls whose degree is δ_{max}. Therefore, $L(\alpha, \nu)$ has been removed during the core processing, meaning that $\mathfrak{D}_1$ also contains an instance $L(\alpha', \gamma)$ of $L(\alpha, \nu)$. Denoting by h the homomorphism computing the restricted core $\mathfrak{D}'$, $\mathfrak{D}_1$ contains $h(L(\alpha, \mathbf{N})) = L(\alpha', \gamma)$ along with all atoms in $B(\alpha, \beta)$ or in $h(B(\alpha, \beta))$. The case $h(\alpha) \neq \alpha$ is not possible because this implies that $B(\alpha, \beta) \subseteq \mathfrak{D}'$ does not hold. Hence, $h(\alpha) = \alpha$, and so, $h(L(\alpha, \nu)) = L(\alpha, \gamma)$ is in $\mathfrak{D}'$, a contradiction to the fact that $\mathfrak{D}'$ contains no atom of the form $L(\alpha, \_)$. Therefore, this part of the proof is complete.

2. In Algorithm 1, on line 3, we have $\mathfrak{D} \subseteq T^*_{\mathsf{fw}}(\mathfrak{D} \cup \mathsf{iRequest})$, thus $\mathfrak{D} \sqsubseteq T^*_{\mathsf{fw}}(\mathfrak{D} \cup \mathsf{iRequest})$ holds. As earlier noticed, $T^*_{\mathsf{fw}}(\mathfrak{D} \cup \mathsf{iRequest}) \sqsubseteq R\_core(T^*_{\mathsf{fw}}(\mathfrak{D} \cup \mathsf{iRequest}))$, which implies by transitivity that on line 4, $\mathfrak{D} \sqsubseteq \mathfrak{D}_1$ holds. Therefore, the proof of this part is complete.

3. Let φ in $\mathfrak{D}'$, not in $\mathfrak{D}$ and not isomorphic to an atom in $\mathsf{iRequest}$, and assume that $\mathfrak{D}' \backslash \{\varphi\} \models \mathbb{C}$. Then, φ has been generated during the computation of $\mathfrak{D}_1 = T^*_{\mathsf{fw}}(\mathfrak{D} \cup \mathsf{iRequest})$. Hence, there exists $c : B(\mathbf{X}, \mathbf{Y}) \Rightarrow L(\mathbf{X}, \mathbf{Z})$ in $\mathbb{C}$ and a homomorphism h such that $h(c) \subseteq \mathfrak{D}_1$ and $h(head(c)) = \varphi$.

Since $\mathfrak{D}' \backslash \{\varphi\} \models c$, c cannot be full, meaning that c has existentially quantified variables and that $\mathfrak{D}' \backslash \{\varphi\}$ contains an atom of the form $L(\alpha, \gamma)$ where $\alpha = h(\mathbf{X})$ and $\gamma \neq h(\mathbf{Z})$. Let p be the least integer such that $h(body(c)) \subseteq T^p_{\mathsf{fw}}(\mathfrak{D} \cup \mathsf{iRequest})$ and $\varphi \notin T^p_{\mathsf{fw}}(\mathfrak{D} \cup \mathsf{iRequest})$. Since φ is in $\mathfrak{D}'$, for every null N in φ, $\delta(N) < \delta_{\max}$. Thus, when computing $T^{p+1}_{\mathsf{fw}}(\mathfrak{D} \cup \mathsf{iRequest})$, all nulls occurring in the instantiated body of the corresponding constraint have a degree strictly less than $\delta_{\max} - 1$, meaning that φ is generated as the atom $L(\alpha, \mathbf{N})$ where $\mathbf{N}$ is a vector of fresh nulls whose degrees are strictly less than δ_{max}. As a consequence, by definition of T_{fw}, $L(\alpha, \gamma)$ is not in $T^p_{\mathsf{fw}}(\mathfrak{D} \cup \mathsf{iRequest})$ (otherwise $L(\alpha, \mathbf{N})$ would not be generated), meaning that $L(\alpha, \gamma)$ is generated at a subsequent computation step q (i.e. $q > p + 1$).

Now, let h_0 be the homomorphism defined by $h_0(\mathbf{N}) = \gamma$ and $h_0(N) = N$ for every N not occurring in $\mathbf{N}$. We first notice that h_0 is restricted because, since $L(\alpha, \gamma)$ is in $\mathfrak{D}'$ (this so because $L(\alpha, \gamma) \in \mathfrak{D}' \backslash \{\varphi\}$), all nulls in γ have a degree strictly less than $\delta_{\max}$. Denoting by $Atoms(\mathbf{N})$ the set of atoms in $\mathfrak{D}_1$ in which at least one null in $\mathbf{N}$ occurs, we show that for every A in $Atoms(\mathbf{N})$, $h_0(A)$ is also in $\mathfrak{D}_1$. Consider now the step m where the first atom A in $Atoms(\mathbf{N})$ different from φ appears in $T^m_{\mathsf{fw}}(\mathfrak{D} \cup \mathsf{iRequest})$. During the computation, a constraint c_A is instantiated into $h_A(c_A)$ so as to contain φ in its body (the only atom so far in which $\mathbf{N}$ occurs), in order to generate $A = h_A(head(c_A))$. Hence, as soon as $h_0(\varphi)$ appears in $T^q_{\mathsf{fw}}(\mathfrak{D} \cup \mathsf{iRequest})$, $h_0(h_A(c_A))$ applies and generates $h_0(A)$, showing that $h_0(A)$ is in $\mathfrak{D}_1$. Since this reasoning applies step by step for every A in $Atoms(\mathbf{N})$, this implies that for every A in $Atoms(\mathbf{N})$, $h_0(A)$ is in $\mathfrak{D}_1$.

Hence, in the computation of $R\_core(\mathfrak{D}_1)$, $\mathbf{N}$ can be instantiated into γ using h_0 (or one of its extensions), which implies that φ is not in $\mathfrak{D}'$, a contradiction to our hypothesis that φ is in $\mathfrak{D}'$. Therefore, the proof is complete. □

B Proof of Proposition 6.2

Proposition 6.2 *Let $\Delta = (\mathfrak{D}, \mathbb{C})$ be a database and* dRequest *a finite set of instantiated atoms where every null occurs at most once. Algorithm 2 returns $\Delta' = (\mathfrak{D}', \mathbb{C})$ where $\mathfrak{D}'$ satisfies the following statements*:

1. *Effectiveness (a) for every φ in* dRequest*, $\mathfrak{D}'$ contains no atom isomorphic to φ, (b) $R\_core(\mathfrak{D}') = \mathfrak{D}'$, and (c) $\mathfrak{D}' \models \mathbb{C}$.*
2. *Monotonicity $\mathfrak{D}' \sqsubseteq \mathfrak{D}$.*

Proof 1(a) Let φ in dRequest and suppose an atom φ_1 isomorphic to φ that belongs to $\mathfrak{D}'$, when running Algorithm 2. If φ_1 is in $ToDel$ on line 1, then for φ_1 to belong to $\mathfrak{D}'$, it must be that on line 5, φ_1 is in $\mathfrak{D}_1$ and thus that, on line 7, φ_1 is in $Back$. If φ_1 is not in $ToDel$, then φ_1 is not in $\mathfrak{D}$ showing that this atom appears in $\mathfrak{D}_1$ on line 4, the only step in Algorithm 2 where atoms are added to the instance. In this case, we again have that φ_1 is in $Back$. But then, on line 11, φ_1 belongs to $T^*_{\mathsf{bw}}(Disallowed \cup Back, \mathfrak{D}_1)$ showing that φ_1 cannot belong to $\mathfrak{D}_2$. Since $\mathfrak{D}'$ is a subset of $\mathfrak{D}_2$, it is not possible that φ_1 belongs to $\mathfrak{D}'$, a contradiction that completes this part of the proof.

1(b) $R\_core(\mathfrak{D}') = \mathfrak{D}'$ is an obvious consequence of the statements on lines 5 and 12 of Algorithm 2.

1(c) If $\mathfrak{D}'$ is output by Algorithm 2 on line 9, then $Disallowed$ is empty, meaning that every null N in $\mathfrak{D}_1$ is such that $\delta(N) < \delta_{max}$. Thus, by Proposition 4.1, we have $T^*_{\mathsf{fw}}(\mathfrak{D} \backslash ToDel) \models \mathbb{C}$, implying that, by Lemma 3.1, $\mathfrak{D}' \models \mathbb{C}$. Otherwise, $\mathfrak{D}'$ is output on line 13, in which case Proposition 4.2 shows that $\mathfrak{D}_2 \models \mathbb{C}$, and thus, by Lemma 3.1, we have $\mathfrak{D}' \models \mathbb{C}$, which completes this part of the proof.

2. To prove that $\mathfrak{D}' \sqsubseteq \mathfrak{D}$, we first show that for $\mathfrak{D}_1 = R\_core(T^*_{\mathsf{fw}}(\mathfrak{D} \backslash ToDel))$ as computed on line 5 of Algorithm 2, we have $\mathfrak{D}_1 \sqsubseteq \mathfrak{D}$. To this end, let us first prove by induction that $T^*_{\mathsf{fw}}(\mathfrak{D} \backslash ToDel) \sqsubseteq \mathfrak{D}$. The base case of the induction, that is, $\mathfrak{D} \backslash ToDel \sqsubseteq \mathfrak{D}$, holds because $\mathfrak{D} \backslash ToDel \subseteq \mathfrak{D}$. Then, for a given integer p, assuming that $T^k_{\mathsf{fw}}(\mathfrak{D} \backslash ToDel) \sqsubseteq \mathfrak{D}$ for every $0 \leq k \leq p$, we show that $T^{p+1}_{\mathsf{fw}}(\mathfrak{D} \backslash ToDel) \sqsubseteq \mathfrak{D}$. To see this, let h^p be the homomorphism such that $h^p(T^p_{\mathsf{fw}}(\mathfrak{D} \backslash ToDel)) \subseteq \mathfrak{D}$ and consider the two cases as follows:

- Assume first that $T^{p+1}_{\mathsf{fw}}(\mathfrak{D} \backslash ToDel) = T^p_{\mathsf{fw}}(\mathfrak{D} \backslash ToDel)$. This means that the least fixed point has been reached, in which case h^{k+1} is equal to h^k.
- When $T^{p+1}_{\mathsf{fw}}(\mathfrak{D} \backslash ToDel) \neq T^p_{\mathsf{fw}}(\mathfrak{D} \backslash ToDel)$, for every null N in $T^p_{\mathsf{fw}}(\mathfrak{D} \backslash ToDel)$ let $h^{p+1}(N) = h^p(N)$. Then, considering a null N in $T^{p+1}_{\mathsf{fw}}(\mathfrak{D} \backslash ToDel)$ but not in $T^p_{\mathsf{fw}}(\mathfrak{D} \backslash ToDel)$ implies that $\delta(N) \leq \delta_{\max}$ and that there exists c in $\mathbb{C}$ and a homomorphism h such that $h(body(c)) \subseteq T^p_{\mathsf{fw}}(\mathfrak{D} \backslash ToDel)$ and N occurs in $\varphi = h(head(c))$. In this case, we have $h^p(h(body(c)) \subseteq \mathfrak{D}$ because, by our induction hypothesis, $h^p(T^p_{\mathsf{fw}}(\mathfrak{D} \backslash ToDel)) \subseteq \mathfrak{D}$. Since $\mathfrak{D} \models \mathbb{C}$, c applies in $\mathfrak{D}$, meaning that $\mathfrak{D}$ contains an atom $A = v(\varphi)$ where v is defined over the nulls ν in φ by $v(\nu) = t$ where t is the corresponding term in A. Notice that v is well defined because any null ν in φ either occurs in $T^p_{\mathsf{fw}}(\mathfrak{D} \backslash ToDel)$, in which case $v(\nu) = \nu$, or ν is a fresh null such N, in which case $v(\nu)$ is uniquely defined. In this case, let $h^{p+1}(N)$ be $v(N)$. We thus obtain a homomorphism h^{p+1} such that $h^{p+1}(T^{p+1}_{\mathsf{fw}}(\mathfrak{D} \backslash ToDel)) \subseteq \mathfrak{D}$, showing that $T^{p+1}_{\mathsf{fw}}(\mathfrak{D} \backslash ToDel) \sqsubseteq \mathfrak{D}$.

Therefore, $T^*_{\mathsf{fw}}(\mathfrak{D} \backslash ToDel) \sqsubseteq \mathfrak{D}$ holds. Moreover, since $\mathfrak{D}_1 = R\_core(T^*_{\mathsf{fw}}(\mathfrak{D} \backslash ToDel))$, $\mathfrak{D}_1 \sqsubseteq T^*_{\mathsf{fw}}(\mathfrak{D} \backslash ToDel)$ holds, implying $\mathfrak{D}_1 \sqsubseteq \mathfrak{D}$. Thus, if the test on line 9 succeeds, we trivially have $\mathfrak{D}' \sqsubseteq \mathfrak{D}$. Otherwise, since $\mathfrak{D}' \subseteq \mathfrak{D}_2 \subseteq \mathfrak{D}_1$, $\mathfrak{D}' \sqsubseteq \mathfrak{D}$ holds as well, and the proof is complete. □

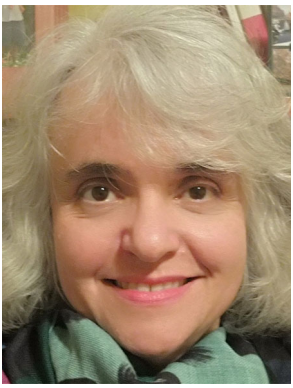
References

1. Abiteboul S, Hull R, Vianu V (1995) Foundations of databases, vol 8. Addison-Wesley, Reading
2. Afrati FN, Kolaitis PG (2009) Repair checking in inconsistent databases: algorithms and complexity. In: Proceedings of the 12th international conference on database theory—ICDT, Russia, March 23–25, 2009, pp 31–41
3. Ahmeti A, Calvanese D, Polleres A (2014) Updating RDFS ABoxes and TBoxes in SPARQL. CoRR arXiv:1403.7248
4. Alchourrón CE, Gärdenfors P, Makinson D (1985) On the logic of theory change: partial meet contraction and revision functions. J Symb Log 50(2):510–530. https://doi.org/10.2307/2274239
5. Arenas M, Pérez J (2011) Querying semantic web data with SPARQL. In: Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems, PODS, Athens, Greece, pp 305–316
6. Benedikt M, Konstantinidis G et al (n.d.) Benchmarking the chase. In: Principles of database systems (PODS 2017) **(to appear)**
7. Chabin J, Gomes Jr L, Halfeld Ferrari M (2018) A context-driven querying system for urban graph analysis. In: IDEAS. ACM, pp 297–301
8. Chirkova R, Fletcher GHL (2009) Towards well-behaved schema evolution. In: 12th international workshop on the web and databases, WebDB, USA
9. Codd EF (1975) Understanding relations (installment #6). FDT Bull ACM SIGMOD 7(1):1–4
10. DBOrleans-Team (2018) A prototype—updating with marked nulls—version 2018. http://www.univ-orleans.fr/lifo/Members/Mirian.Halfeld/mi2-software.html. Accessed 23 Oct 2018
11. De Giacomo G, Lembo D, Oriol X, Savo DF, Teniente E (2017) Practical update management in ontology-based data access. In: Proceedings of the semantic web—ISWC–16th international semantic web conference, Vienna, Austria, Part I, pp 225–242
12. De Giacomo G, Lenzerini M, Poggi A, Rosati R (2009) Dealing with inconsistencies and incompleteness in database update (position paper)
13. Deutsch A, Nash A, Remmel JB (2008) The chase revisited. In: Proceedings of the twenty-seventh ACMSIGMOD-SIGACT-SIGART symposium on principles of database systems, PODS 2008, June 9–11, 2008, Vancouver, BC, Canada, pp 149–158
14. D'Orazio L, Halfeld-Ferrari M, Hara CS, Kozievitch NP, Musicante MA (2017) Graph constraints in urban computing: dealing with conditions in processing urban data. In: DARLI-AP, international workshop on data analytics solutions for Real-LIfe APplications in conjunction with the 3rd IEEE international conference on smart data, England, UK
15. Fagin R, Kolaitis PG, Miller RJ, Popa L (2003) Data exchange: semantics and query answering. In: Proceedings of the database theory—ICDT, 9th international conference, Italy, pp 207–224
16. Fagin R, Kolaitis PG, Popa L (2005) Data exchange: getting to the core. ACM Trans Database Syst 30(1):174–210
17. Fagin R, Kuper GM, Ullman JD, Vardi MY (1986) Updating logical databases. Adv Comput Res 3:1–18
18. Fagin R, Ullman JD, Vardi MY (1983) On the semantics of updates in databases. In: Proceedings of the second ACM SIGACT-SIGMOD symposium on principles of database systems, Colony Square Hotel, Atlanta, Georgia, USA, pp 352–365
19. Flouris G, Konstantinidis G, Antoniou G, Christophides V (2013) Formal foundations for RDF/S KB evolution. Knowl Inf Syst 35(1):153–191
20. Gottlob G (2005) Computing cores for data exchange: new algorithms and practical solutions. In: Proceedings of the twenty-fourth ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems, Baltimore, Maryland, USA, pp 148–159
21. Gottlob G, Orsi G, Pieris A (2011) Ontological queries: rewriting and optimization. In: Proceedings of the 27th international conference on data engineering, ICDE, Germany, pp 2–13
22. Grahne G (1991) The problem of incomplete information in relational databases. Lecture notes in computer science, vol 554. Springer, New York
23. Grahne G, Onet A (2011) On conditional chase termination. In: Proceedings of the 5th Alberto Mendelzon international workshop on foundations of data management, Santiago, Chile, May 9–12, 2011
24. Gutierrez C, Hurtado CA, Vaisman AA (2011) RDFS update: from theory to practice. In: Proceedings of the semanic web: research and applications—8th extended semantic web conference, ESWC, Greece, Part II, pp 93–107
25. Halfeld Ferrari Alves M, Hara CS, Kozievitch NP, Uber FR (2018) Urban data consistency in RDF: a case study of Curitiba transportation system. In: 'LADaS@VLDB', volume 2170 of CEUR workshop proceedings, CEUR-WS.org, pp 33–40

26. Halfeld Ferrari Alves M, Laurent D, Spyratos N (1998) Update rules in datalog programs. J Log Comput 8(6):745–775
27. Halfeld Ferrari M, Hara CS, Uber FR (2017) RDF updates with constraints. In: Proceedings of the knowledge engineering and semantic web—8th international conference, KESW, Szczecin, Poland, pp 229–245
28. Halfeld Ferrari M, Laurent D (2017) Updating RDF/S databases under constraints. In: Proceedings of the advances in databases and information systems—21st European conference, ADBIS, Nicosia, Cyprus, pp 357–371
29. Hansson SO (2016) Logic of belief revision. In: Zalta EN (ed) The Stanford encyclopedia of philosophy, winter 2016 edition, 2016th edn. Metaphysics Research Lab, Stanford University, Stanford
30. Hell P, Nesetril J (1992) The core of a graph. Discrete Math 109(1–3):117–126. https://doi.org/10.1016/0012-365X(92)90282-K
31. Imielinski T, Lipski W Jr (1984) Incomplete information in relational databases. J ACM 31(4):761–791
32. Katsuno H, Mendelzon AO (1991) On the difference between updating a knowledge base and revising it. In: Proceedings of the 2nd international conference on principles of knowledge representation and reasoning (KR'91). Cambridge, MA, USA, April 22–25, pp 387–394
33. Knublauch H, Hendler JA, Idehen K (2011) SPIN—overview and motivation. W3C member submission. http://www.w3.org/Submission/2011/SUBM-spin-overview-20110222. Accessed 3 Nov 2017
34. Knublauch H, Ryman A (2017) Shapes constraint language (SHACL). W3C first public working draft, w3c. http://www.w3.org/TR/2015/WD-shacl-20151008/. Accessed 3 Nov 2017
35. Kozievitch NP, Gadda TMC, Fonseca KVO, Rosa MO, Gomes Jr LC, Akbar M (2016) Exploratory analysis of public transportation data in Curitiba. In: 'XXXVI CSBC'. Sociedade Brasileira de Computação, pp 1656–1666
36. Lausen G, Meier M, Schmidt M (2008) Sparqling constraints for RDF. In: Proceedings of the EDBT, 11th international conference on extending database technology, France, pp 499–509
37. Libkin L (2006) Data exchange and incomplete information. In: Proceedings of the twenty-fifth ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems, June 26–28, 2006, Chicago, Illinois, USA, pp 60–69
38. Libkin L (2015) Sql's three-valued logic and certain answers. In: 18th international conference on database theory, ICDT, Brussels, Belgium, pp 94–109
39. Link S (2002) Towards a tailored theory of consistency enforcement in databases. In: Proceedings of the foundations of information and knowledge systems, second international symposium, FoIKS, Germany, pp 160–177
40. Link S, Schewe K (2002) An arithmetic theory of consistency enforcement. Acta Cybern 15(3):379–416
41. Lipski Jr W (1984) On relational algebra with marked nulls. In: Proceedings of the third ACM SIGACT-SIGMOD symposium on principles of database systems, Waterloo, Ontario, Canada, pp. 201–203
42. Lösch U, Rudolph S, Vrandecic D, Studer R (2009) Tempus fugit. In: Proceedings of the semantic web: research and applications, 6th European semantic web conference, ESWC, Crete, Greece, , pp 278–292
43. Motik B, Horrocks I, Sattler U (2007) Bridging the gap between OWL and relational databases. In: Proceedings of the 16th international conference on world wide web, WWW, Canada, pp 807–816
44. Nikolaou C, Koubarakis M (2016) Querying incomplete information in RDF with SPARQL. Artif Intell 237:138–171
45. Onet A (2013) The chase procedure and its applications in data exchange. In: Data exchange, integration, and streams, pp 1–37
46. Patel-Schneider PF (2015) Using description logics for RDF constraint checking and closed-world recognition. In: Proceedings of the twenty-ninth AAAI conference on artificial intelligence, USA, pp 247–253
47. Pichler R, Skritek S (2011) The complexity of evaluating tuple generating dependencies. In: Proceedings of the database theory—ICDT, 14th international conference, Sweden, pp 244–255
48. Rabinovitch J, Leitman J (1996) Urban planning in Curitiba. Sci Am 274(3):46–53
49. Reiter R (1986) A sound and sometimes complete query evaluation algorithm for relational databases with null values. J ACM 33(2):349–370
50. Schewe K, Thalheim B (1998) Limitations of rule triggering systems for integrity maintenance in the context of transition specifications. Acta Cybern 13(3):277–304
51. Schewe K, Thalheim B (1999) Towards a theory of consistency enforcement. Acta Inf 36(2):97–141. https://doi.org/10.1007/s002360050155
52. Solbrig H, hommeaux EP (2014) Shape expressions 1.0 definition. W3C member submission. http://www.w3.org/Submission/2014/SUBM-shex-defn-20140602. Accessed 3 Nov 2017
53. Stardog5 (2017) Enterprise knowledge graph. http://www.stardog.com/docs/
54. W3C-Working-Group (n.d.) Linked data patch format—pathological graph. https://www.w3.org/TR/ldpatch/#pathological-graph. Accessed 3 Nov 2017

55. Winslett M (1990) Updating logical databases. Cambridge University Press, New York
56. Zaniolo C (1984) Database relations with null values. J Comput Syst Sci 28(1):142–166

Jacques Chabin is currently Associate Professor in Computer Sciences, teaching at the University Institute of Technology of Orléans. Since 1994, he is a member of PAMDA team in the *Laboratoire d'Informatique Fondamentale d'Orléans* (LIFO) of the *Université d'Orléans* and the INSA *Centre Val de Loire*. He is awarded a Bachelor of Science in Mathematics of *Université d'Orléans*, and he earned master and PhD degrees in Computer Science from the same University. His research interests include term rewriting, proof and database theory.



Mirian Halfeld-Ferrari is currently a Full Professor at *Université d'Orléans* (France), member of LIFO research laboratory and co-director of the ICVL research Federation, comprising two academic research laboratories of the *Région Centre Val de Loire* in France. She earned a PhD degree from *Université Paris Sud* (XI), LRI (*Laboratoire de Recherche en Informatique*), in 1996 and an HDR (*Habilitation à Diriger des Recherches*) from *Université de Tours* in 2007. She has been an Associate Professor at UFPR (*Universidade Federal do Paraná*), in Brazil (1996–1998), and at *Université de Tours*, in France, from 1998 to 2009. Professor Halfeld-Ferrari has been involved in different international collaboration work. Her research interests include theoretical aspects of databases, query languages, graph databases and dynamic aspects of databases.

Dominique Laurent is currently Emeritus Professor in the University of Paris-Seine (France) and a member of the research laboratory ETIS (UMR CNRS). He graduated in 1978 in Mathematics, received his doctoral degree in Computer Science in 1987 and then his Habilitation in 1994, all from the *Université d'Orléans* (France). From 1987 to 1996, he was Associate Professor in this same university, and in 1996, he joined the *Université de Tours* (France) as a Full Professor. In 2003, he was appointed as a Full Professor at *Université de Cergy-Pontoise* (France), where he led first the Computer Science Department and then the Graduate School *Science et Ingénierie*. His research interests include database theory, data mining and data warehousing. Professor Laurent has published numerous papers in these areas and has been involved in many international programme committees and editorial boards.