Handling Inconsistencies in Tables with Nulls and Functional Dependencies

Dominique Laurent, Nicolas Spyratos

Noname manuscript No.
(will be inserted by the editor)

Handling Inconsistencies in Tables with Nulls and Functional Dependencies

Dominique Laurent · Nicolas Spyratos

Received: date / Accepted: date

Abstract In this paper we address the problem of handling inconsistencies in tables with missing values (or nulls) and functional dependencies. Although the traditional view is that table instances must respect all functional dependencies imposed on them, it is nevertheless relevant to develop theories about how to handle instances that violate some dependencies.

The usual approach to alleviate the impact of inconsistent data on the answers to a query is to introduce the notion of repair: a repair is a minimally different consistent instance and an answer is consistent if it is present in every repair.

Our approach is fundamentally different: we use set theoretic semantics for tuples and functional dependencies that allow us to associate each tuple with a truth value among the following: true, false, inconsistent or unknown. The users of the table can then query the set of true tuples as usual. Regarding missing values, we make no assumptions on their existence: a missing value exists only if it is inferred from the functional dependencies of the table.

The main contributions of the paper are the following: (a) we introduce a new approach to handle inconsistencies in a table with nulls and functional dependencies, (b) we give algorithms for computing all true, inconsistent and false tuples, (c) we discuss how our approach relates to Belnap's four valued logic, (d) we describe how our approach can be applied to the consolidation of two or more tables and (e) we discuss the relationship between our approach and that of table repairs.

Keywords Database semantics . Inconsistent database . Functional dependency . Data Integration . Null value . Four-valued logic

Dominique Laurent
ETIS Laboratory - ENSEA, CY Cergy Paris University, CNRS
F-95000 Cergy-Pontoise, France
dominique.laurent@u-cergy.fr

Nicolas Spyratos
LISN Laboratory - University Paris-Saclay, CNRS
F-91405 Orsay, France
nicolas.spyratos@lri.fr

Acknowledgment: Work conducted while the second author was visiting at FORTH Institute of Computer Science, Crete, Greece (https://www.ics.forth.gr/)

1 Introduction

In several applications today we are confronted with tables which contain missing values and which have to satisfy functional dependencies. Such tables are often the result of merging two or more 'source' tables. For example, think of two groups of researchers studying the objects found in an archaeological site. Each object has a numerical identifier and the researchers record data regarding the following attributes of each object:

– Identifier (an integer in our example)
– Kind (such as statue, weapon, ...)
– Material from which the object is made (such as iron, bronze, marble, ...)
– Century in which the object is believed to have been made.

At the end of their work each group submits their findings to the site coordinator in the form of a table in which each row (or tuple) contains the data recorded for a single object. For example, a tuple $(1, statue, marble, 1.AD)$ means that object 1 is a statue made of marble and believed to have been made in the first century before our era. Similarly a tuple $(2, statue, , 2.AD)$ means that object 2 is a statue of unknown material, believed to have been made during the second century before our era. Note that, in this tuple, there is no value for material meaning that the material from which object 2 is made has not been determined. In the relational model of databases such blanks are referred to as 'null values' or simply as 'nulls'.

D_1	Id	K	M	C
	1	k	m	c
	1		m'	
	2	k'	m'	c
	2	k'	m''	

D_2	Id	K	M	C
	1	k		c
	2	k'		c'
	2	k'	m''	

D	Id	K	M	C
	1	k	m	c
	1		m'	
	2	k'	m'	c
	2	k'		c'
	2	k'	m''	

Fig. 1 The tables prepared by the individual groups and the merged table

Figure 1 shows an example of two tables D_1 and D_2 containing the results from two different groups of researchers about two objects, 1 and 2, where k, k' represent values of Kind (K); m, m', m'' values of Material (M); and c, c' values of Century (C). Both tables also contain nulls.

Now, the data contained in the two tables can be merged into a single table D containing all tuples from the two tables, as shown in Figure 1. In doing this merging we may have discrepancies between tuples of D. For example, object 1 appears in D as being made from two different materials; and object 2 appears as made from two different materials and in two different centuries. This kind of discrepancies may lead to 'inconsistencies' that should be identified by the site coordinator and resolved in cooperation with the researchers of the two groups.

It should be obvious from this example that the merging of two or more tables into a single table more often than not results in inconsistencies ? even if the individual tables are each consistent. For example, tables D_1 and D_2 each satisfies the functional dependencies $Id \rightarrow K$ and $Id \rightarrow C$, whereas the merged table D does not satisfy $Id \rightarrow C$. A similar situation arises in data warehouses where one tries to merge views of the underlying sources into a single materialized view to be stored in the data warehouse.

As a last example, in a relational database, although each table satisfies its functional dependencies, the database as a whole may be inconsistent. To determine whether the database is consistent one merges all tables into a single table D (under certain assumptions discussed in [22]) and applies all functional dependencies on D through the well known chase algorithm [11, 21]. If the algorithm terminates successfully (*i.e.,* no inconsistency is detected) then the database is consistent; otherwise the algorithm stops and the database is inconsistent.

So in general the question is: what should we do when a table is inconsistent? There are roughly three approaches: (a) reject the table, (b) try to correct or 'repair' it so that to make it consistent (and therefore be able to work with the repaired table) and (c) keep the table as is but make sure you know which part is consistent and which is not.

The first approach is that followed by logicians who generally apply the principle of explosion (Ex Contradictione Non Sequitur Quodlibet, 'from contradiction anything [follows]' (see [9]); it is also followed by database theorists: when the chase algorithm reveals inconsistency then the database is declared inconsistent. This approach is clearly not acceptable in practice.

The second approach is always difficult to implement as, in principle, it requires the generation of all repairs. This approach, also referred to as 'consistent query answering' has motivated important research efforts during the past two decades and is still the subject of current research. The reader is referred to Section 5.4 for a brief overview as well as a comparison with our approach.

In our work we follow the third approach that is we keep inconsistencies in the table but we determine which part of the table is consistent and which is not. More specifically, we use set theoretic semantics for tuples and functional dependencies that allow us to associate each tuple of he table with one truth value among the following: true, false, inconsistent or unknown. Users can then query the set of true tuples of the table as usual. Regarding missing values, we make no assumptions on their existence: a null value exists only if it is inferred from the functional dependencies of the table.

The main contributions of the paper can be summarized as follows:

1. We introduce a new approach to handle inconsistencies in a table with nulls and functional dependencies; we do so by adapting the set theoretic semantics of [19] to our context and by extending the chase algorithm so that all inconsistencies are accounted for in the table.
2. We give algorithms for computing all true, inconsistent and false tuples in the table.
3. We discuss how our approach relates to Belnap?s four-valued logic [5].
4. We describe how our approach can be applied to the consolidation of two or more tables.
5. We discuss how our approach relates to consistent query answering.

The paper is organized as follows: In Section 2 we recall basic definitions and notations regarding tables and we introduce the set theoretic semantics that we use in our work. In Section 3 we give definitions and properties regarding the truth values that we associate with tuples. Section 4 is devoted to computational aspects, providing algorithms for computing the truth values of tuples. In Section 5 we first discuss how our approach relates to Belnap?s four-valued logic (Sub-sections 5.1 and 5.2); how it can be applied to the consolidation of two or more tables (Sub-section 5.3); and how it relates to consistent query answering (Sub-section 5.4). Section 6 contains concluding remarks and suggestions for future research.

2 The Model

In this section we present the basic definitions regarding tuples and tables as well as the set theoretic semantics that we use for tuples and functional dependencies. Our approach builds upon earlier work on the partition model [19].

2.1 The Partition Model Revisited

Following [19], we consider a universe $U = \{A_1, \ldots, A_n\}$ in which every attribute A_i is associated with a set of atomic values called the domain of A_i and denoted by $dom(A_i)$. We call relation schema (or simply schema) any nonempty subset of U and we denote it by the concatenation of its elements; for example $\{A_1, A_2\}$ is simply denoted by A_1A_2. Similarly, the union of schemas S_1 and S_2 is denoted as S_1S_2 instead of $S_1 \cup S_2$.

We define a *tuple* t to be a partial function from U to $\bigcup_{A \in U} dom(A)$ such that, for every A in U, if t is defined over A then $t(A)$ belongs to $dom(A)$. The domain of definition of t is called the *schema* of t, denoted by $sch(t)$. We note that tuples in our approach satisfy the *First Normal Form* [21] in the sense that each tuple component is an atomic value from an attribute domain.

Regarding notation, we follow the usual convention that, whenever possible, lower-case characters denote domain constants and upper-case characters denote the corresponding attributes. Following this convention the schema of a tuple $t = ab$ is AB and more generally, we denote the schema of t as T.

Assuming that the schema of a tuple t is understood, t is denoted by the concatenation of its values, that is: $t = a_{i_1} \ldots a_{i_k}$ means that for every $j = 1, \ldots, k$, $t(A_{i_j}) = a_{i_j}$, a_{i_j} is in $dom(A_{i_j})$, and $sch(t) = A_{i_1} \ldots A_{i_k}$.

We assume that for any distinct attributes A and B, we have either $dom(A) = dom(B)$ or $dom(A) \cap dom(B) = \emptyset$. However, this may lead to ambiguities when two attributes have the same domain. Ambiguity can be avoided by prefixing each value of an attribute domain with the attribute name. For example, if $dom(A) = dom(B)$ we can say 'an A-value a' to mean that a belongs to $dom(A)$, and 'a B-value a' to mean that a belongs to $dom(B)$. In order to keep the notation simple we shall omit prefixes whenever no ambiguity is possible.

Given a tuple t, for every A in $sch(t)$, $t(A)$ is also denoted by $t.A$ and more generally, for every subset S of $sch(t)$ the restriction of t to S, also called *sub-tuple* of t, is denoted by $t.S$. In other words, $t.S$ is the tuple such that $sch(t.S) = S$ and for every A in S, $(t.S).A = t.A$.

Denoting by $\mathcal{T}$ the set of all tuples that can be built up given a universe U, let $\sqsubseteq$ be the 'sub-tuple' relation, defined over $\mathcal{T}$ as follows: for any tuples t_1 and t_2, $t_1 \sqsubseteq t_2$ holds if t_1 is a sub-tuple of t_2.

The relation $\sqsubseteq$ is clearly a partial order over $\mathcal{T}$. Given a set of tuples D, the set of all sub-tuples of the tuples in D is called the *lower closure* of D and it is defined by: $\mathsf{LoCl}(D) = \{q \in \mathcal{T} \mid (\exists t \in D)(q \sqsubseteq t)\}$.

The notion of $\mathcal{T}$-*mapping*, as defined below, generalizes that of interpretation defined in [19].

Definition 1 Let U be a universe. A $\mathcal{T}$-*mapping* is a mapping μ defined from $\bigcup_{A \in U} dom(A)$ to $2^{\mathbb{N}}$. A $\mathcal{T}$-*mapping* μ can be extended to the set $\mathcal{T}$ as follows: for every $t = a_{i_1} \ldots a_{i_k}$ in $\mathcal{T}$, $\mu(t) = \mu(a_{i_1}) \cap \ldots \cap \mu(a_{i_k})$.

A $\mathcal{T}$-mapping μ is an *interpretation* if μ satisfies the *partition constraint* stating that for every A in U, and for distinct values a and a' in $dom(A)$, $\mu(a) \cap \mu(a') = \emptyset$.

We emphasize that in [19] interpretations provide the basic ingredient for defining true tuples: a tuple t is said to be true in an interpretation μ if $\mu(t)$ is nonempty.

To see the intuition behind this definition consider a relational table D over U and suppose that each tuple is associated with a unique identifier, say an integer. Now, for every A in U and every a in $dom(A)$, define $\mu(a)$ to be the set of all identifiers of the tuples in D containing a. Then μ is an interpretation as it satisfies the partition constraint. Indeed, due to the fact that, for every attribute A in U, a tuple t can not have more than one A-value, it is then impossible that $\mu(a) \cap \mu(a')$ be nonempty for any distinct values a, a' in $dom(A)$.

Incidentally, if for every A in U we denote by $dom^*(A)$ the set of all A-values such that $\mu(a) \neq \emptyset$, then the set $\{\mu(a) \mid a \in dom^*(A)\}$ is a *partition* of $\bigcup_{a \in dom^*(A)} \mu(a)$ (whence the name "partition model"). The following example illustrates this important feature.

Example 1 Considering $U = \{A, B, C\}$ and $D = \{ab, bc, ac, a'b', b'c', abc\}$, the tuples in D can be respectively assigned the identifiers 1, 2, 3, 4 5 and 6. In that case, we have $\mu(a) = \{1, 3, 6\}$, $\mu(a') = \{4\}$, $\mu(b) = \{1, 2, 6\}$, $\mu(b') = \{4, 5\}$, $\mu(c) = \{2, 3, 6\}$, $\mu(c') = \{5\}$, and $\mu(\alpha) = \emptyset$ for any attribute value α different than a, a', b, b', c and c'.

It is clear that the $\mathcal{T}$-mapping μ is an interpretation and, since $dom^*(A)$, $dom^*(B)$ and $dom^*(C)$ are respectively equal to $\{a, a'\}$, $\{b, b'\}$ and $\{c, c'\}$, it is easy to see that $\{\mu(\alpha) \mid \alpha \in dom^*(A)\}$ is a partition of $\{1, 3, 4, 6\}$, $\{\mu(\beta) \mid \beta \in dom^*(B)\}$ is a partition of $\{1, 2, 4, 5, 6\}$, and $\{\mu(\gamma) \mid \gamma \in dom^*(C)\}$ is a partition of $\{2, 3, 5, 6\}$.

Moreover, extending μ to non unary tuples yields the following regarding the tuples in D: $\mu(ab) = \{1, 6\}$, $\mu(bc) = \{2, 6\}$, $\mu(ac) = \{3, 6\}$, $\mu(a'b') = \{4\}$, $\mu(b'c') = \{5\}$, and $\mu(abc) = \{6\}$. □

Summarizing our discussion, when dealing with consistent tables in [19], only interpretations are relevant. In the present work, we follow the same idea, but we also extend the work of [19] so that we can deal with inconsistencies. As we shall see, non satisfaction of the partition constraint in Definition 1 is the key criterion to characterize inconsistent tuples.

2.2 Functional Dependencies

The notion of functional dependency in our approach is defined as in [19].

Definition 2 Let U be a universe. A *functional dependency* is an expression of the from $X \to Y$ where X and Y are nonempty sub-sets of U.

A $\mathcal{T}$-mapping μ *satisfies* $X \to Y$, denoted by $\mu \models X \to Y$, if for all tuples x and y, respectively over X and Y such that $\mu(x) \cap \mu(y) \neq \emptyset$, $\mu(x) \subseteq \mu(y)$ holds.

Based on Definition 2, for all X and Y such that $X \cap Y = \emptyset$, and for every $\mathcal{T}$-mapping μ, the following holds:

$$\mu \models X \to Y \text{ if and only if } \mu \models X \to A \text{ for every } A \text{ in } Y.$$

Indeed, since for all sets E, E_1 and E_2, $E \subseteq E_1 \cap E_2$ holds if and only if so does ($E \subseteq E_1$ and $E \subseteq E_2$), it holds that for every x and y such that $\mu(x) \cap \mu(y) \neq \emptyset$, $\mu(x) \subseteq \mu(y)$ holds if and only if for every constant a in y, $\mu(x) \subseteq \mu(a)$ holds.

We thus assume without loss of generality that functional dependencies are of the form $X \to A$ where A is an attribute not in X. Under this assumption, we consider pairs $\Delta = (D, \mathcal{FD})$ where D is a table over U and $\mathcal{FD}$ a set of functional dependencies over U, and we say that a $\mathcal{T}$-mapping μ satisfies Δ, denoted by $\mu \models \Delta$, if (i) for every t in D, $\mu(t) \neq \emptyset$, and (ii) μ satisfies every $X \to A$ in $\mathcal{FD}$.

Relating our notion of functional dependency with the standard one in relational databases [21], we first recall that a relation r over universe U satisfies $X \to A$ if for all tuples t and t' in r such that $t.X = t'.X$, we have $t.A = t'.A$.

In our approach, let $\Delta = (D, \mathcal{FD})$ and consider two tuples t and t' in D such that $t.X = t'.X = x$. Then for every $\mathcal{T}$-mapping μ such that $\mu \models \Delta$, $\mu(t)$ and $\mu(t')$ are nonempty, implying that $\mu(x) \cap \mu(t.A)$ and $\mu(x) \cap \mu(t'.A)$ are also nonempty. By Definition 2, this implies that $\mu(x)$ is a sub-set of $\mu(t.A)$ and $\mu(t'.A)$. As a consequence, assuming that $t.A \neq t'.A$ (*i.e.,* that $X \to A$ is not satisfied in the sense of the relational model), means that $\mu(t.A) \cap \mu(t'.A)$ is nonempty, and therefore μ *can not be an interpretation.*

Therefore if we restrict $\mathcal{T}$-mappings to be interpretations then the notion of functional dependency satisfaction in our approach is the same as that of relational databases. As we shall see, this observation supports the notion of consistency for Δ, to be given later (in Definition 3).

Given $\Delta = (D, \mathcal{FD})$ and tuples t, t' and t'', the following notations are extensively used in the remainder of the paper.

- $\Delta \vdash t$, denotes that for every μ such that $\mu \models \Delta$, $\mu(t) \neq \emptyset$.
- $\Delta \vdash (t \sqcap t')$, denotes that for every μ such that $\mu \models \Delta$, $\mu(t) \cap \mu(t') \neq \emptyset$.
- $\Delta \vdash (t \preceq t')$ denotes that for every μ such that $\mu \models \Delta$, $\mu(t) \subseteq \mu(t')$.
- $\Delta \vdash (t \preceq t' \sqcap t'')$ denotes that for every μ such that $\mu \models \Delta$, $\mu(t) \subseteq \mu(t') \cap \mu(t'')$.

Given $\Delta = (D, \mathcal{FD})$, we now build a particular $\mathcal{T}$-mapping μ such that $\mu \models \Delta$ as follows: Let $(\mu_i)_{i \geq 0}$ be the sequence defined by the steps below:

1. Associate each tuple t with an identifier, $id(t)$, called the *tuple identifier* of t (this can be an integer that identifies t uniquely).
2. For every domain constant a let $\mu_0(a) = \{id(t) \mid t \in D \text{ and } a \sqsubseteq t\}$;
3. While there exists $X \to A$ in $\mathcal{FD}$, x over X and a in $dom(A)$ such that $\mu(xa) \neq \emptyset$ and $\mu(x) \not\subseteq \mu(a)$, define μ_{i+1} by: $\mu_{i+1}(a) = \mu_i(a) \cup \mu_i(x)$ and $\mu_{i+1}(\alpha) = \mu_i(\alpha)$ for any other symbol α.

Lemma 1 *For every $\Delta = (D, \mathcal{FD})$, the sequence $(\mu_i)_{i \geq 0}$ has a unique limit μ^* that satisfies $\mu^* \models \Delta$.*

Moreover, μ^* is such that for all constants α and β, $\Delta \vdash (\alpha \sqcap \beta)$ holds if and only if $\mu^*(\alpha) \cap \mu^*(\beta) \neq \emptyset$ holds.

Proof See Appendix A. □

Given $\Delta = (D, \mathcal{FD})$, Lemma 1 shows the following two basic points:

1. There always exists a $\mathcal{T}$-mapping μ such that $\mu \models \Delta$.
2. For every tuple t, $\Delta \vdash t$ if and only if $\mu^*(t) \neq \emptyset$.

We now characterize when $\Delta \vdash (t \preceq a)$ holds, inspired by the well known relational notion of closure of a relation scheme with respect to a set of functional dependencies [21]. To this end, given $\Delta = (D, \mathcal{FD})$ we define the *closure of a tuple t in Δ* (or *closure of t* for short, when Δ is understood), denoted by t^+, as the output of Algorithm 1.

Algorithm 1 Closure of t

Input: $\Delta = (D, \mathcal{FD})$ and a tuple t.
Output: A set t^+ of constants a

```
1: t+ := {a | a ⊑ t}
2: while t+ changes do
3:     for all X → A ∈ FD do
4:         for all x such that for every b in x, b ∈ t+ and Δ ⊢ xa do
5:             t+ := t+ ∪ {a}
6: return t+
```

The following lemma gives basic properties related to inclusions that hold between tuples in a given table.

Lemma 2 *Let $\Delta = (D, \mathcal{FD})$. Then $\Delta \vdash (t \preceq a)$ holds if and only if a is in t^+.*

Proof See Appendix B. □

The following example illustrates Lemma 1 and Lemma 2.

Example 2 Let $U = \{A, B, C\}$ and $\Delta = (D, \mathcal{FD})$ where $D = \{ab, bc, abc'\}$ and $\mathcal{FD} = \{B \rightarrow C\}$. Associating ab, bc and abc' respectively with 1, 2 and 3, μ^* is obtained as follows:
• First, we have $\mu_0(a) = \{1, 3\}$, $\mu_0(b) = \{1, 2, 3\}$, $\mu_0(c) = \{2\}$ and $\mu_0(c') = \{3\}$ and $\mu_0(\alpha) = \emptyset$ for any other domain constant α.
• Then, considering $B \rightarrow C$, we have $\mu_1(a) = \{1, 3\}$, $\mu_1(b) = \mu_1(c) = \mu_1(c') = \{1, 2, 3\}$ and $\mu^*(\alpha) = \emptyset$ for any other domain constant α.

Hence, $\mu^* = \mu_1$ and we remark that $\mu^*(c) \cap \mu^*(c') \neq \emptyset$, thus that μ^* is not an interpretation. Nevertheless, as stated by Lemma 1, it is easy to see that $\mu^* \models \Delta$. Computing $(ab)^+$ according to Algorithm 1 yields the following:
• $(ab)^+$ is first set to $\{a, b\}$
• Then, considering $B \rightarrow C$, since b is in $(ab)^+$, and since $\Delta \vdash bc$ and $\Delta \vdash bc'$, c and c' are inserted in $(ab)^+$.

As no further modifications are possible, we have $(ab)^+ = \{a, b, c, c'\}$, showing in particular that $\Delta \vdash (ab \preceq c)$ and $\Delta \vdash (ab \preceq c')$, that is $\Delta \vdash (ab \preceq c \sqcap c')$. □

3 Semantics

In this section we provide basic definitions and properties regarding the truth value associated with a tuple. It is important to keep in mind that, in doing so, we follow the intuition of Belnap's Four-valued logic of [5].

3.1 The Truth-Value of a Tuple

The following definition of consistency is borrowed from [19].

Definition 3 Δ is said to be *consistent* if there exists an *interpretation* μ such that $\mu \models \Delta$.

Since in our approach, inconsistent tables are *not* discarded, it is crucial to be able to provide semantics to any $\Delta = (D, \mathcal{FD})$, being it consistent or not. To this end, inspired by Belnap's Four-valued logic [5], we consider *four* possible truth values for a given tuple t in Δ. The notations of truth values for tuples in our approach and their intuitive meaning are as follows, for given a tuple t:

- Truth value `true`: t is true in Δ.
- Truth value `false`: t is false in Δ. This means that we do *not* follow the Closed World Assumption (CWA), according to which any non true tuple is false [18].
- Truth value `inc` (*i.e.,* inconsistent): t is true *and* false in Δ. This truth value is necessary for 'safely' dealing with inconsistent tuples.
- Truth value `unkn` (*i.e.,* unknown): t is not true, not false and not inconsistent in Δ. This truth value is necessary for dealing with tuples not falling in one of the above three categories.

We now emphasize the following intuitive remarks, that will be formalized in the forthcoming Definition 4:

- $\Delta \vdash t$ indicates that for every μ such that $\mu \models \Delta$, $\mu(t) \neq \emptyset$. In this case, t is said to be *potentially true* in Δ.
- $\Delta \vdash (t \preceq a \sqcap a')$ for some distinct a and a' in the same attribute domain, denoted by $\Delta \mid\!\sim t$, indicates that for every μ such that $\mu \models \Delta$, $\mu(t) \subseteq \mu(a) \cap \mu(a')$ holds. In this case, t is said to be *potentially false* to reflect that $\mu(a) \cap \mu(a')$ must be empty.

Consequently, when a tuple t is such that $\Delta \vdash t$ and $\Delta \mid\!\sim t$, for μ to be an interpretation, μ must associate t with a set expected to be empty and nonempty, which is of course a case of inconsistency! This explains why, in our approach, 'potentially true' and 'potentially false', should respectively be understood as 'true or inconsistent' and 'false or inconsistent'.

Based on this intuition, each tuple is assigned one of the four truth values according to the following definition.

Definition 4 Given $\Delta = (D, \mathcal{FD})$ and a tuple t, the truth value of t in Δ, denoted by $v_\Delta(t)$, is defined as follows:

- $v_\Delta(t) =$ `true` if $\Delta \vdash t$ and $\Delta \not\mid\!\sim t$; t is said to be *true in* Δ.
- $v_\Delta(t) =$ `false` if $\Delta \not\vdash t$ and $\Delta \mid\!\sim t$; t is said to be *false in* Δ.
- $v_\Delta(t) =$ `inc` if $\Delta \vdash t$ and $\Delta \mid\!\sim t$; t is said to be *inconsistent in* Δ.
- $v_\Delta(t) =$ `unkn` if $\Delta \not\vdash t$ and $\Delta \not\mid\!\sim t$; t is said to be *unknown in* Δ.

In Section 5, we investigate how our approach can be expressed in the context of the Four-valued logic [5] and we show that the semantics of [13] allows to compute the truth value of every tuple t as defined above. The following proposition shows that our notion of inconsistent tuple complies with Definition 3.

Proposition 1 *$\Delta = (D, \mathcal{FD})$ is consistent if and only if there exist no tuple t such that $v_\Delta(t) = \texttt{inc}$.*

Proof See Appendix C. □

Regarding potentially false tuples, the following proposition gives a necessary condition and several sufficient conditions for $\Delta \mid\!\sim t$ to hold.

Proposition 2 *Given $\Delta = (D, \mathcal{FD})$ and t over schema T, the following holds:*

1. *If $\Delta \mid\!\sim t$ then there exists $X \to A$ in $\mathcal{FD}$ such that $X \subseteq T$ and $\Delta \vdash t.X$.*
2. *For every t' such that $t \sqsubseteq t'$, if $\Delta \mid\!\sim t'$ then $\Delta \mid\!\sim t$.*
3. *For every $X \to A$ in $\mathcal{FD}$ such that $X \subseteq T$, denoting $t.X$ by x we have:*
   (a) if $\Delta \vdash xa$ and $\Delta \vdash xa'$ where a and a' are in $dom(A)$, then $\Delta \mid\!\sim t$;
   (b) if $A \in T$ and $\Delta \vdash xa$ for a in $dom(A)$ distinct from $t.A$, then $\Delta \mid\!\sim t$.

Proof 1. By Lemma 2, as $\Delta \mid\!\sim t$, there exist A in U and a and a' in $dom(A)$ such that $a \neq a'$ and a and a' are in t^+. Since a and a' cannot both occur in t as A-values, at least one of these values occurs in t^+ due to the loop of line 2 in Algorithm 1. The condition of the 'if' statement in this loop must thus succeed at least once, which is the condition stated in the proposition.

2. Since $\Delta \mid\!\sim t'$, then $\Delta \vdash t' \preceq a \sqcap a'$, and as $t \sqsubseteq t'$, then $\Delta \vdash t' \preceq t$. Thus $\Delta \vdash t \preceq a \sqcap a'$.

3(a). The hypotheses imply that $\Delta \vdash x \preceq a \sqcap a'$, thus that $\Delta \mid\!\sim x$. Since $x \sqsubseteq t$, the result follows from the previous item.
3(b). As above the hypotheses imply that $\Delta \vdash x \preceq a$. Thus a is in x^+, and as $x \sqsubseteq t$, a is in t^+. As $t.A$ is also in t^+, the result follows. □

Definition 4 and Proposition 2 allow for the following generic remarks concerning the truth value of a tuple:

- By Lemma 1 the set of potentially true tuples is determined by μ^*. Since D is finite, the number of potentially true tuples is finite even if domains are infinite. As a consequence the numbers of true tuples and of inconsistent tuples are finite as well, whatever the cardinalities of attribute domains.
- Proposition 2(1) shows that if $\mathcal{FD} = \emptyset$ then no tuple can be potentially false. As a consequence, in this case no tuple is inconsistent either, meaning that tuples are either true or unknown.
- Proposition 2(1) also shows that potentially false tuples must have a schema including the left-hand side X of a functional dependency of $\mathcal{FD}$.
- Proposition 2(2) shows that every super-tuple of a potentially false tuple is also potentially false. Therefore, the number of false tuples might be infinite.
- Proposition 2(3) implies that potentially true tuples and functional dependencies generate potentially false tuples, in a possibly infinite number.

We illustrate Definition 4 and Proposition 2 through the following example.

Example 3 As in Example 2, let $U = \{A, B, C\}$ and $\Delta = (D, \mathcal{FD})$ where $D = \{ab, bc, abc'\}$ and $\mathcal{FD} = \{B \rightarrow C\}$. It can be seen that for every μ such that $\mu \models D$ and $\mu \models B \rightarrow C$, the following holds:
– $\mu(abc) \neq \emptyset$ and $\mu(abc') \neq \emptyset$
– $\mu(c) \cap \mu(c') \neq \emptyset$, $\mu(b) \subseteq \mu(c)$, $\mu(b) \subseteq \mu(c')$.

It therefore follows that $\Delta \vdash abc$ and $\Delta \vdash abc'$, that Δ is inconsistent, and that $\Delta \not\sim b$. As $b \sqsubseteq abc$, $\Delta \vdash b$, and so, $v_\Delta(b) = \texttt{inc}$, meaning that b is inconsistent in Δ. By Proposition 2(2), it follows that, for example, abc, abc', bc and bc' are also inconsistent in Δ.

Now, let c'' in $dom(C)$ distinct from c and c'. Computing the truth value of bc'' in Δ, we first notice that $\mu(bc'') \subseteq \mu(c'')$ holds for every μ. On the other hand it also holds that $\mu(bc'') \subseteq \mu(c)$ for every μ such that $\mu \models \Delta$. Indeed:

– If $\mu(bc'') = \emptyset$, the inclusion trivially holds.
– If $\mu(bc'') \neq \emptyset$, then $\mu(b) \subseteq \mu(c'')$ because $\mu \models B \rightarrow C$. Hence $\mu(bc'') = \mu(b)$, and as $\mu(b) \subseteq \mu(c)$, $\mu(bc'') \subseteq \mu(c)$ holds.

Therefore $\Delta \not\sim bc''$ and since $\Delta \not\vdash bc''$ (because μ^* as computed in Example 2 is such that $\mu^*(bc'') = \emptyset$ and $\mu^* \models \Delta$), it follows that $v_\Delta(bc'') = \texttt{false}$, and thus that bc'' and all its super-tuples are false in Δ.

As an example of unknown tuple in Δ, let a' be in $dom(A)$ such $a' \neq a$, and consider $a'c$. Since, as shown by Example 2, $\mu^*(a'c) = \emptyset$, $\Delta \not\vdash a'c$. On the other hand, since $(a'c)^+ = \{a', c\}$, $\Delta \not\vdash a'c$, which shows that $v_\Delta(a'c) = \texttt{unkn}$.

Now, for $t = a'bc$, Proposition 2 implies that $\Delta \not\sim t$ because $\Delta \not\sim bc$ and $bc \sqsubseteq t$. However, as $\Delta \not\vdash t$, although bc is inconsistent in Δ, it can *not* be inferred that t is inconsistent in Δ. It therefore turns out that t is false in Δ. □

The following more sophisticated examples show that computing *all* inconsistent tuples in Δ is not an easy task.

Example 4 Let $U = \{A, B, C\}$ and $\Delta = (D, \mathcal{FD})$ where $D = \{ab, bc, ab', b'c'\}$ and $\mathcal{FD} = \{B \rightarrow C, A \rightarrow C\}$. Although D contains no inconsistent tuples, it turns out that Δ is *not* consistent. To see this, let us compute μ^* as defined in Lemma 1:
• First, to define μ_0, we associate the tuples ab, bc, ab' and $b'c'$ with the integers 1, 2, 3 and 4, respectively. Thus, $\mu_0(a) = \{1, 3\}$, $\mu_0(b) = \{1, 2\}$, $\mu_0(b') = \{3, 4\}$, $\mu_0(c) = \{2\}$, $\mu_0(c') = \{4\}$ and $\mu_0(\alpha) = \emptyset$ for any other domain constant α.
• The next steps modify μ_0 so as to satisfy $B \rightarrow C$ and $A \rightarrow C$ as follows:

1. μ_1 is defined by: $\mu_1(a) = \{1, 3\}$, $\mu_1(b) = \{1, 2\}$, $\mu_1(b') = \{3, 4\}$, $\mu_1(c) = \{1, 2\}$ and $\mu_1(c') = \{2, 4\}$;
2. μ_2 is defined by: $\mu_2(a) = \{1, 3\}$, $\mu_2(b) = \{1, 2\}$, $\mu_2(b') = \{3, 4\}$, $\mu_2(c) = \{1, 2, 3\}$ and $\mu_2(c') = \{2, 3, 4\}$.
3. μ_3 is defined by: $\mu_3(a) = \{1, 3\}$, $\mu_3(b) = \{1, 2\}$, $\mu_3(b') = \{3, 4\}$, $\mu_3(c) = \{1, 2, 3, 4\}$ and $\mu_3(c') = \{1, 2, 3, 4\}$.

As $\mu_3 \models \mathcal{FD}$, we have $\mu^* = \mu_3$. Since $\mu^*(c) \cap \mu^*(c') \neq \emptyset$, μ^* is not an interpretation, meaning that Δ is not consistent.

Moreover, since $a^+ = \{a, c, c'\}$, $b^+ = \{b, c, c'\}$ and $(b')^+ = \{b', c, c'\}$, a, b and b' are inconsistent in Δ. This implies that, for example, ac, bc' and $ab'c'$ are also inconsistent in Δ, because each of these tuples t is such that $\Delta \vdash t$ (as $\mu^*(t) \neq \emptyset$) and $\Delta \not\sim t$ (by Proposition 2(2)). On the other hand it can be seen that c and c'

are not inconsistent because $c^+ = \{c\}$ and $(c')^+ = \{c'\}$. Since $\mu^*(c)$ and $\mu^*(c')$ are nonempty, these tuples are true in Δ. □

Example 5 Let $\Delta = (D, \mathcal{FD})$ be defined over $U = \{A, B, C\}$ by $D = \{abc, ac'\}$ and $\mathcal{FD} = \{A \rightarrow B, B \rightarrow C\}$. Here again, the tuples in D along with the functional dependencies in $\mathcal{FD}$ show no inconsistency. However computing μ^* yields the following:
• To define μ_0, we associate the tuples abc and ac' with the integers 1 and 2, respectively. It follows that $\mu_0(a) = \{1, 2\}$, $\mu_0(b) = \{1\}$, $\mu_0(c) = \{1\}$, $\mu_0(c') = \{2\}$ and $\mu_0(\alpha) = \emptyset$ for any other domain constant α.
• The next steps modify μ_0 so as to satisfy $A \rightarrow B$ and $B \rightarrow C$ as follows:

1. Due to $A \rightarrow B$, μ_1 is defined by: $\mu_1(a) = \{1, 2\}$, $\mu_1(b) = \{1, 2\}$, $\mu_1(c) = \{1\}$ and $\mu_1(c') = \{2\}$;
2. Due to $B \rightarrow C$, μ_2 is defined by: $\mu_2(a) = \{1, 2\}$, $\mu_2(b) = \{1, 2\}$, $\mu_2(c) = \{1, 2\}$ and $\mu_2(c') = \{1, 2\}$.

As $\mu_2 \models \mathcal{FD}$, $\mu^* = \mu_2$ and, since $\mu^*(c) \cap \mu^*(c')$ is nonempty, we obtain as in Example 2, that Δ is not consistent.

Moreover, we have $a^+ = \{a, b, c, c'\}$ and $b^+ = \{b, c, c'\}$ showing that, by Lemma 2, $\Delta \vdash a \preceq (c \sqcap c')$ and $\Delta \vdash (b \preceq c \sqcap c')$, thus that a and b are inconsistent in Δ. It can then be seen that, for example, abc, bc' and ac are also inconsistent in Δ.

Now, let $\Delta' = (D', \mathcal{FD})$ such that $D' = \{ac, ac'\}$. In this case, μ^* is defined by $\mu^*(a) = \{1, 2\}$, $\mu^*(c) = \{1\}$, $\mu^*(c') = \{2\}$ and $\mu^*(\alpha) = \emptyset$ for any other domain constant α. Therefore, $a^+ = \{a\}$, showing that a is *not* inconsistent in Δ'. As a consequence, ac, ac' along with all their sub-tuples are true in Δ' and all other tuples are unknown in Δ'. □

As shown by the previous examples, although it is possible to compute inconsistent and true tuples based on Lemma 1 and Lemma 2, a systematic and efficient computation is likely to be problematic; we address this issue next.

4 Computing the Semantics

In the context of standard two valued logic, computing the semantics of Δ means computing the set of all tuples true in Δ. In our approach, computing the semantics is more involved because we have to compute the following three sets: the set of all true tuples, the set of all inconsistent tuples and the set of all false tuples in Δ; and once this is done, the set of all unknown tuples is the complement of the union of these three sets with respect to $\mathcal{T}$.

However, as already seen, the set of potentially false tuples is infinite, thus making it impossible to compute the set of all false tuples. We cope with this difficulty as follows: we first give algorithms to compute all true tuples and all inconsistent tuples, and then, we provide an algorithm for computing the truth value of a given tuple, including when this tuple is false or unknown!

Algorithm 2 Chasing a table

Input: $\Delta = (D, \mathcal{FD})$
Output: The chased table $\Delta_{ch} = (D_{ch}, \mathcal{FD})$ and a set $inc(\mathcal{FD})$ containing sets of tuples associated with each $X \to A$ in $\mathcal{FD}$

$D_{ch} := D$
for all $X \to A$ in $\mathcal{FD}$ **do**
    $inc(X \to A) := \emptyset$
while D_{ch} changes **do**
    for all $X \to A \in \mathcal{FD}$ **do**
        for all t_1 in D_{ch} such that $XA \subseteq sch(t_1)$ **do**
            for all t_2 in D_{ch} such that $t_1.X = t_2.X$ **do**
                if $A \notin sch(t_2)$ **then**
                    $D_{ch} := D_{ch} \cup \{t_2 a_1\}$ where $a_1 = t_1.A$
                if $A \in sch(t_2)$ and $t_1.A \neq t_2.A$ **then**
                    Let $y_i = t_i.(sch(t_i) \setminus A)$ and $a_i = t_i.A$, for $i = 1, 2$
                    $D_{ch} := D_{ch} \cup \{y_1 a_2, y_2 a_1\}$
                    $inc(X \to A) := inc(X \to A) \cup \{x\}$ where $x = t_1.X = t_2.X$
// Normalization: keep in D_{ch} only maximal tuples
for all t_1 in D_{ch} **do**
    for all t_2 in D_{ch} **do**
        if $t_2 \sqsubseteq t_1$ and $t_1 \neq t_2$ **then**
            $D_{ch} := D_{ch} \setminus \{t_2\}$
$inc(\mathcal{FD}) := \{inc(X \to A) \mid inc(X \to A) \neq \emptyset\}$
return $\Delta_{ch} = (D_{ch}, \mathcal{FD})$ and $inc(\mathcal{FD})$

4.1 Computing True Tuples and Inconsistent Tuples

We first propose an effective algorithm for the computation of all potentially true tuples in a given Δ. This algorithm is in fact inspired by the standard chase algorithm [19,21], with the main difference that when a functional dependency cannot be satisfied, our algorithm does *not* stop. Instead, our chasing algorithm carries on the computation, returning a database $\Delta_{ch} = (D_{ch}, \mathcal{FD})$ and a set $inc(\mathcal{FD})$ based on which inconsistent and true tuples can be efficiently computed.

To prove this basic result, we first show the following lemma which states the strong relationship between tuples in D_{ch} and potentially true tuples.

Lemma 3 *Algorithm 2 applied to $\Delta = (D, \mathcal{FD})$ always terminates. Moreover, for every tuple t, $\mu^*(t) \neq \emptyset$ holds if and only if t is in $\mathsf{LoCl}(D_{ch})$.*

Proof See Appendix D. □

Comparing our chasing algorithm to the standard one [19,21], it is easily seen that, if $\Delta = (D, \mathcal{FD})$ is consistent, then D_{ch} coincides with the standard chase output. To compute the set of all inconsistent tuples in our approach, we introduce Algorithm 3, which is shown to be correct by the following lemma.

Lemma 4 *Given $\Delta = (D, \mathcal{FD})$, a tuple t is inconsistent in Δ if and only if $t \in \mathsf{Inc}(\Delta)$.*

Proof See Appendix E. □

The following proposition characterizes inconsistent and true tuples in Δ based on Algorithm 2 and Algorithm 3.

Algorithm 3 Inconsistent tuples in $\Delta = (D, \mathcal{FD})$

Input: The output of Algorithm 2, that is $\Delta_{ch} = (D_{ch}, \mathcal{FD})$ and $inc(\mathcal{FD})$.
Output: A set of tuples $\mathsf{Inc}(\Delta)$

```
1: Inc(Δ) := ∅
2: for all t in D_ch do
3:     for all X → A in FD such that XA ⊆ T do
4:         if x = t.X is in inc(X → A) then
5:             temp := {q | x ⊑ q ⊑ t}
6:             while temp changes do
7:                 for all q in temp do
8:                     for all Y → B in FD such that YB ⊆ Q \ A and B ∈ X do
9:                         temp := temp ∪ {q_B} where q_B = q.(Q \ B)
10:            Inc(Δ) := Inc(Δ) ∪ temp
11: return Inc(Δ)
```

Proposition 3 *Given $\Delta = (D, \mathcal{FD})$ and a tuple t:*
1. t is inconsistent in Δ if and only if $t \in \mathsf{Inc}(\Delta)$.
2. t is true in Δ if and only if $t \in \mathsf{LoCl}(D_{ch}) \setminus \mathsf{Inc}(\Delta)$.

Proof Immediate consequence of Definition 4, Lemma 3 and Lemma 4. □

The following examples illustrate Proposition 3.

Example 6 As in Example 2, let $U = \{A, B, C\}$ and $\Delta = (D, \mathcal{FD})$ where $D = \{ab, bc, abc'\}$ and $\mathcal{FD} = \{B \to C\}$. A tabular version of D is displayed on the left below, whereas D_{ch} is the table on the right.

D	A	B	C
	a	b	
		b	c
	a	b	c'

D_{ch}	A	B	C
	a	b	c
	a	b	c'

Algorithm 2 applies to Δ as follows. Because of $B \to C$, the statement line 9 first inserts abc in D_{ch}. Then, based on abc and abc' in D_{ch} and $B \to C$, due to line 13, b is added in $inc(B \to C)$ but no tuple is added in D_{ch}. The main loop terminates at this stage and, normalization is processed on line 14, removing ab and bc from D_{ch}. Hence, Algorithm 2 returns the table D_{ch} shown above and $inc(\mathcal{FD}) = \{inc(B \to C)\}$ where $inc(B \to C) = \{b\}$.

When running Algorithm 3 for abc in D_{ch}, the tuples b, ab, bc and abc are first inserted in *temp* on line 5, and on line 9 no tuple is inserted because no functional dependency of the form $Y \to B$ as in the algorithm can be found in $\mathcal{FD}$. A similar computation is preformed with abc' in D_{ch}, adding bc' and abc' in *temp*.

Therefore, Algorithm 3 returns $\mathsf{Inc}(\Delta) = \{abc, abc', ab, bc, bc', b\}$, which by Proposition 3(1) is the set of all inconsistent tuples in Δ. As a consequence, by Proposition 3(2), ac, ac', a, c and c' are the true tuples in Δ. □

More sophisticated examples regarding the consequences of functional dependencies when running Algorithm 2 follow.

Example 7 As in Example 5, let $\Delta = (D, \mathcal{FD})$ over $U = \{A, B, C\}$ where $D = \{abc, ac'\}$ and $\mathcal{FD} = \{A \to B, B \to C\}$. The tabular version of D is shown on the left below, whereas D_{ch} is shown on the right.

D	A	B	C
	a	b	c
	a		c'

D_{ch}	A	B	C
	a	b	c
	a	b	c'

Running Algorithm 2 first inserts abc' in D_{ch} by the statement line 9 due to $A \to B$, and the tuples abc and ac'. Then, b is inserted in $inc(B \to C)$ by the statement line 13, due to $B \to C$ and the tuples abc and abc'. The table D_{ch} and the set $inc(\mathcal{FD})$ output by Algorithm 2 are as in the previous example.

However, as in the two examples, the sets of functional dependencies are not the same, the sets of inconsistent tuples are not equal. Indeed, when running Algorithm 3 for abc in D_{ch}, *temp* first contains b, ab, bc and abc, due to line 5. On line 9, $A \to B$ satisfies the condition for $q = ab$ because $AB \subseteq sch(ab)$ and B is the left-hand side of $B \to C$. Thus, a and ac are inserted in *temp*. A similar computation is preformed with abc' in D_{ch}, adding bc', abc' and ac' in *temp*.

Hence, Algorithm 3 returns $\mathsf{Inc}(\Delta) = \{abc, abc', ab, ac, ac', bc, bc', a, b\}$, which by Proposition 3(1) is the set of all inconsistent tuples in Δ. As a consequence, by Proposition 3(2), c and c' are the only true tuples in Δ.

Now, as in Example 5, referring to $\Delta' = (D', \mathcal{FD})$ with $D' = \{ac, ac'\}$, it is easy to see that $D'_{ch} = D'$. This implies that Δ' is consistent, and that ac, ac', a, c and c' are true in Δ'. □

Example 8 Let $U = \{A, B, C, D\}$ and $\Delta = (D, \mathcal{FD})$ where D is the table on the left below, and $\mathcal{FD} = \{A \to B, A \to C, BC \to D\}$.

D	A	B	C	D
	a	b		
	a		c	d
		b	c	d'

D_{ch}	A	B	C	D
	a	b	c	d
	a	b	c	d'

Running Algorithm 2, by the statement line 9, the following tuples are inserted in D_{ch}: $abcd$, due to $A \to B$, then abc, due to $A \to C$, and then again $abcd$, due to $A \to B$. Consequently, bcd and bcd' are inserted in $inc(BC \to D)$ by the statement line 13, due to $abcd$ and bcd'.

Since no other tuple is added in D_{ch}, the main loop stops, and after normalization in line 14, Algorithm 2 returns D_{ch} shown above on the right and $inc(\mathcal{FD}) = \{inc(BC \to D)\}$ where $inc(BC \to D) = \{bc\}$. Running Algorithm 3 first inserts $abcd$, abc, bcd, $abcd'$, bcd' and bc in *temp* on line 5, and the loop line 6 inserts successively the following tuples in *temp* through the statement line 9: first acd, abd, ac, ab, acd', abd', and then ad, a, ad'. Therefore:

$\mathsf{Inc}(\Delta) = \{abcd, abcd', abc, abd, abd', acd, acd', bcd, bcd', ac, ab, ad, ad', bc, a\}$

and thus, bd, bd', cd, cd', b, c, d and d' are the true tuples in Δ. □

4.2 The Case of False Tuples

By Proposition 2, computing all tuples false in a given $\Delta = (D, \mathcal{FD})$ is not feasible in case of infinite attribute domains. We also emphasize in this respect that, even in case of *finite* domains, the computation is likely to be non tractable in practice.

To cope with this problem, instead of computing the set of all tuples false in Δ, we propose an algorithm to compute the truth value of any *given* tuple t. In this way, instead of being systematically identified, false tuples are identified on

Algorithm 4 Tuple truth value in $\Delta = (D, \mathcal{FD})$

Input: A tuple t, $\Delta_{ch} = (D_{ch}, \mathcal{FD})$ and $\mathsf{Inc}(\Delta)$
Output: The truth value of t as one of the truth values `true`, `false`, `inc` or `unkn`

```
1: v := unkn
2: for all q in D_ch do
3:     if t ⊑ q then
4:         v := true
5:         if t ∈ Inc(Δ) then
6:             v := inc
7: if v = unkn then
8:     closure := {a | a occurs in t}
9:     while closure changes do
10:        for all q in D_ch do
11:            for all X → A in FD such that XA ⊆ Q do
12:                if every α in q.X is in closure then
13:                    closure := closure ∪ {q.A}
14:    if closure contains a and a' in the same attribute domain then
15:        v := false
16: return v
```

demand. Assuming that Δ_{ch} and $\mathsf{Inc}(\Delta)$ have been computed beforehand, given a tuple t, Algorithm 4 returns a truth value that is shown to be equal to $v_\Delta(t)$ by the following proposition.

Proposition 4 *Given $\Delta = (D, \mathcal{FD})$, and a tuple t and assuming that $\Delta_{ch} = (D_{ch}, \mathcal{FD})$ and $\mathsf{Inc}(\Delta)$ have been computed, the truth value returned by Algorithm 4 is equal to $v_\Delta(t)$.*

Proof The proposition is an immediate consequence of Proposition 3 when the returned value is `true` or `inc`. Let us assume that Algorithm 4 returns `false`, then since the loop line 9 reproduces the main loop of Algorithm 1, the value of the set *closure* is equal to t^+, and the result follows from Lemma 2. □

The following example illustrates the algorithm.

Example 9 We recall that in our introductory example of an archaeological site we defined $\Delta = (D, \mathcal{FD})$ as the result of the integration of two data sources Δ_1 and Δ_2. In this setting, $\mathcal{FD} = \{Id \to A, Id \to C\}$ and the table D of Figure 1 is shown in Figure 2.

D	Id	K	M	C
	1	k	m	c
	1		m'	
	2	k'	m'	c
	2	k'		c'
	2	k'	m''	

D_{ch}	Id	K	M	C
	1	k	m	c
	1	k	m'	c
	2	k'	m'	c
	2	k'	m'	c'
	2	k'	m''	c
	2	k'	m''	c'

Fig. 2 The integrated table from the introductory example and its chased version

Applying Algorithm 2 to D produces D_{ch} as shown in Figure 2, and returns $inc(\Delta) = \{inc(Id \to C)\}$ where $inc(Id \to C) = \{2\}$. Then, by Algorithm 3, the set $\mathsf{Inc}(\Delta)$ is defined by:

$$\mathsf{Inc}(\Delta) = \begin{array}{l} \{t \mid 2 \sqsubseteq t \sqsubseteq (2, a', m', c)\} \cup \{t \mid 2 \sqsubseteq t \sqsubseteq (2, a', m', c')\} \\ \{t \mid 2 \sqsubseteq t \sqsubseteq (2, a', m'', c)\} \cup \{t \mid 2 \sqsubseteq t \sqsubseteq (2, a', m'', c')\} \end{array}$$

Hence, applying Algorithm 4, we have the following:

- $v_\Delta(1, a, m, c) = v_\Delta(1, a, m', c) = \texttt{true}$, because as these tuples are in D_{ch} but not in $\mathsf{Inc}(\Delta)$, line 4 changes the value of v and line 6 does not.
- $v_\Delta(2) = v_\Delta(2, c) = v_\Delta(2, c') = \texttt{inc}$, because these tuples are in $\mathsf{Inc}(\Delta)$ and so, line 4 and line 6 successively change the value of v, producing the value `inc`.
- $v_\Delta(1, a') = v_\Delta(1, c') = \texttt{false}$. Indeed, when running Algorithm 4 with $(1, a')$ as input, lines 4 and 6 do not change the value of v (as $(1, a')$ does not occur in D_{ch}). Hence, the loop line 9 is run producing $closure = \{1, a', a\}$, and thus the value of v is set to `false` on line 15. The case of $(1, c')$ is similar and thus not explained here.
- $v_\Delta(a', m) = \texttt{unkn}$. Indeed, as above, when running Algorithm 4 with (a', m) as input, lines 4 and 6 do not change the value of v (as (a', m) does not occur in D_{ch}). Moreover, when running the loop line 9, $closure = \{a', m\}$ is produced, thus the value of v is not changed, and `unkn` is returned. □

4.3 Complexity Issues

We first argue that the computation of inconsistent and true tuples in $\Delta = (D, \mathcal{FD})$ is polynomial in the size of the table D and of the 'number of inconsistencies' (to be defined in this section). To see this, denoting by $|E|$ the cardinality of a set E, we investigate the complexities of Algorithm 2 and of Algorithm 3.

Regarding Algorithm 2, we first notice that, contrary to the standard chase algorithm [21], rows are added in the table during the computation, and some others are then removed by the normalization statement of line 14. To assess the size of the table D_{ch} during the processing, we consider the following facts:

1. The table D_{ch} is first set equal to D, and as long as no inconsistency is detected, one tuple is added in D_{ch} as the 'join' of two tuples in D. Therefore, the cardinality of D_{ch} remains in the same order as that of D. Notice in this respect that, upon normalization, *one* 'join' tuple replaces *two* joined tuples in D, which reduces the size of the table D_{ch} output by the algorithm.
2. However, when inconsistent tuples occur, the statement line 13 shows that a cross-product is performed, whose size has to be taken into account.

We now assess the maximal size of the cross-products mentioned above. For every x in $inc(\mathcal{FD})$, let $X \to A$ be the dependency in $\mathcal{FD}$ such that x belongs to $inc(X \to A)$ during the computation. Denoting by $N(x)$ the number of different A-values such that xa is true in Δ, we define $N(\Delta)$ as the maximal value of all $N(x)$ for all x in $inc(\mathcal{FD})$; in other words $N(\Delta) = \max(\{N(x) \mid x \in inc(\mathcal{FD})\})$.

Thus, given x in $inc(\mathcal{FD})$, during the computation, considering that D_{ch} has at most $N(\Delta)$ tuples of the form $y_i x a_i$ where $i = 1, \ldots, N(\Delta)$, the statement line 13 generates $N(\Delta)^2$ tuples of the form $y_i x a_j$ for $i, j = 1, \ldots, N(\Delta)$. We therefore consider that the size of the table D_{ch} when running Algorithm 2 is in $\mathcal{O}\left(|D| + |inc(\mathcal{FD}))|.N(\Delta)^2\right)$, more simply written as $\mathcal{O}(\widetilde{D})$.

Assessing now the number λ of while-loops run by Algorithm 2, as in the traditional chase algorithm, λ is bounded by the number of missing values in D_{ch}

as it is at its first assignment on line 1. Therefore λ is in $\mathcal{O}\left(|U|.\widetilde{D}\right)$. Since each while-loop is clearly quadratic in the size of D_{ch}, we obtain that the computational complexity is in $\mathcal{O}\left(|U|.\widetilde{D}^3\right)$.

The last point to be mentioned regarding the complexity of Algorithm 2 is that the normalization processing on line 14 is performed through a scan D_{ch} whereby for every t in D_{ch} every sub-tuple of t is removed. Such a processing is clearly in $\mathcal{O}\left(\widetilde{D}^2\right)$. Consequently, considering that $|U| \ll |D_{ch}|$, the overall computational complexity of Algorithm 2, is in $\mathcal{O}\left(\widetilde{D}^3\right)$, that is in

$$\mathcal{O}\left(|D|^3 + |inc(\mathcal{FD})|^3.N(\Delta)^6\right) \tag{1}$$

As the computational complexity of Algorithm 3 is clearly linear with respect to the size of D_{ch}, the global complexity of the computation of inconsistent and true tuples in Δ is as stated just above.

Regarding the complexity of Algorithm 4, the only case to consider is when the loop on line 9 is run. We notice in this respect that this loop is run at most as many times as D contains distinct constants. Since D contains at most $|U|.|D|$ distinct constants and since each run of the loop is in $\mathcal{O}(|D_{ch}|)$, we obtain a complexity in $\mathcal{O}(|D|.|D_{ch}|)$, which does not increase the overall complexity given by (1).

As expected, these results show that the efficiency of computing the semantics is highly decreased when the number of inconsistent tuples increases. We mention in this respect that in the the case of consolidating or merging of two or more tables (see Sub-section 5.3), if we assume that the tables are all consistent, then $N(\Delta)$ is bounded by the number of tables being consolidated.

5 Discussion and Related Work

In this section, we successively discuss how our approach relates to Belnap?s Four-valued logic (Sub-sections 5.1 and 5.2); how it can be applied to the consolidation of two or more tables (Sub-section 5.3); and how it relates to consistent query answering (Sub-section 5.4).

5.1 Basics of Four-Valued Logic

Four-valued logic was introduced by Belnap in [5], who argued that his formalism is of interest when integrating data from various data sources. To this end, he introduced four truth values denoted by `t`, `b`, `n` and `f` and read as *true*, *both true and false*, *neither true nor false* and *false*, respectively. An important feature of this Four-valued logic is that its truth values can be compared according to two partial orderings, known as *truth ordering* and *knowledge ordering*, respectively denoted by $\preceq_t$ and $\preceq_k$ and defined as follows:

$$\texttt{n} \preceq_k \texttt{t} \preceq_k \texttt{b} \ ; \ \texttt{n} \preceq_k \texttt{f} \preceq_k \texttt{b} \qquad \text{and} \qquad \texttt{f} \preceq_t \texttt{n} \preceq_t \texttt{t} \ ; \ \texttt{f} \preceq_t \texttt{b} \preceq_t \texttt{t}.$$

As a consequence, two new connectors were introduced, denoted by $\oplus$ and $\otimes$, in addition to the standard connectors $\vee$ (disjunction) and $\wedge$ (conjunction). The corresponding truth tables, along with that for negation, are displayed in Figure 3 and show that $\vee$ and $\oplus$ correspond to the least upper bound (lub) with respect to

φ	$\neg\varphi$
t	f
b	b
n	n
f	t

$\vee$	t	b	n	f
t	t	t	t	t
b	t	b	t	b
n	t	t	n	n
f	t	b	n	f

$\wedge$	t	b	n	f
t	t	b	n	f
b	b	b	f	f
n	n	f	n	f
f	f	f	f	f

$\oplus$	t	b	n	f
t	t	b	t	b
b	b	b	b	b
n	t	b	n	f
f	b	b	f	f

$\otimes$	t	b	n	f
t	t	t	n	n
b	t	b	n	f
n	n	n	n	n
f	n	f	n	f

Fig. 3 Truth tables of basic connectors

$\preceq_t$ and $\preceq_k$, respectively; whereas $\wedge$ and $\otimes$, correspond to the geatest lower bound (glb) with respect to $\preceq_t$ and $\preceq_k$, respectively . It is also shown in [5,12] that the set $\{\mathtt{t}, \mathtt{b}, \mathtt{n}, \mathtt{f}\}$ equipped with the two orderings $\preceq_t$ and $\preceq_k$ has a distributive bi-lattice structure.

Not surprisingly, some basic properties holding in standard logic do not hold in this setting. For example, Figure 3 shows that formulas of the form $\Phi \vee \neg\Phi$ are not always true, independently of the truth value of Φ. The reader is referred to the literature [3,5,12,13,20] for more on the properties of Four-valued logic.

Based on Four-valued logic, semantics to rule based languages have been proposed in the literature [12,13]. In this work, we consider as in [13], conjunctive rules of the form:

$$(\forall \xi_1, \ldots, \xi_n)(head(\xi i_1, \ldots, \xi_{i_p}) \leftarrow body(\xi_1, \ldots, \xi_n))$$

where $\xi_1, \ldots \xi_k$ are variables to be instantiated by integers, $head(\xi i_1, \ldots \xi_{i_p})$ is a literal and $body(\xi_1, \ldots \xi_k)$ is a conjunction of literals. To simplify the notation, universal quantifiers are omitted in the rules.

In this setting, the semantics is a set of pairs $\langle \varphi, \mathtt{v} \rangle$ where φ is a ground atom and $\mathtt{v}$ a truth value different than $\mathtt{n}$. The intuition here is that, in such set, called a *v-set*, the truth value of φ is $\mathtt{v}$.

A database Π is a pair $\Pi = (E, R)$ where E is a v-set and R a set of rules. The *semantic consequence operator* associated with Π, denoted by Σ_Π, is defined for every v-set S by the following steps:

(1) Define first $\Gamma_\Pi(S)$ as follows:

$$\begin{aligned}
\Gamma_\Pi(S) = E \cup \quad & \{\langle h, \mathtt{t}\rangle \mid (\exists \rho \in inst(E,R))(h = head(\rho) \wedge v_S(body(\rho)) = \mathtt{t})\} \\
\cup \quad & \{\langle h, \mathtt{b}\rangle \mid (\exists \rho \in inst(E,R))(h = head(\rho) \wedge v_S(body(\rho)) = \mathtt{b})\} \\
\cup \quad & \{\langle h, \mathtt{f}\rangle \mid (\exists \rho \in inst(E,R))(\neg h = head(\rho) \wedge v_S(body(\rho)) = \mathtt{t})\} \\
\cup \quad & \{\langle h, \mathtt{b}\rangle \mid (\exists \rho \in inst(E,R))(\neg h = head(\rho) \wedge v_S(body(\rho)) = \mathtt{b})\}
\end{aligned}$$

(2) Then, $\Sigma_\Pi(S)$ is defined by: $\Sigma_\Pi(S) = \{\langle \varphi, \mathtt{v}_\oplus(\varphi)\rangle \mid \varphi \text{ occurs in } \Gamma^E_\Delta(S)\}$ where $\mathtt{v}_\oplus(\varphi) = \bigoplus\{\mathtt{v} \mid \langle \varphi, \mathtt{v}\rangle \in \Gamma^E_\Delta(S)\}$.

It has been shown in [13] that the sequence defined by $\Sigma^0 = E$ and $\Sigma^n = \Sigma_\Pi(\Sigma^{n-1})$, for $n \geq 1$, is monotonic with respect to $\preceq_k$ and thus has a limit. This limit, denoted by $\Sigma(\Pi)$ is then referred to as the *semantics* of Π.

5.2 Partition Semantics and Four-valued Logic

To relate the set theoretic semantics as presented so far with Four-valued logic, we make the assumption that *every attribute domain contains a finite number of constants*. As will be seen next, this assumption is fundamental, so as to ensure that we consider *finite* sets of rules.

Under this restriction, we express our model as a set of rules *a la* Datalog [10], and we show (see Proposition 5) that the semantics of this set of rules, as defined in [13], is 'equivalent' to tuple truth-value as defined in Definition 4. It is important to note that, in doing so, our purpose is *not* to obtain a tractable set of rules. Instead, our goal is to show that our partition semantics can be somehow 'grounded' in an appropriate logic.

To build the rules, we associate every tuple t in $\mathcal{T}$ with three unary predicates denoted by $\varphi_t^+(.)$, $\varphi_t^-(.)$ and $\varphi_t(.)$. The intended meaning is that, for $n \in \mathbb{N}$, the truth value of $\varphi_t^+(n)$ is $\mathtt{t}$ when $\Delta \vdash t$ holds, the truth value of $\varphi_t^-(n)$ is $\mathtt{t}$ when $\Delta \nvdash t$ holds; moreover the truth value of $\varphi_t(n)$ is meant to provide the truth value of t in Δ, based on the previous two ones.

Moreover, for every tuple t and every attribute value a, we consider a unary predicate, denoted by $\psi_\alpha^t(.)$, meant to be true when α belongs to t^+. In this way, based on Lemma 2 and on Proposition 2, we state rules to express that $\Delta \nvdash t$ holds. Note that what our assumption implies is that the number of these predicates is *finite*, although potentially huge.

Given $\Delta = (D, \mathcal{FD})$, we build a database $\Pi(\Delta) = (\Phi, \mathcal{R})$ as described next. The set Φ is the union of two sets of pairs Φ^D and Φ^{cl} defined as follows:

1. As done when defining μ^*, every tuple t in D is associated with a unique identifier, say id_t. Then Φ^D contains all pairs $\langle \varphi_t^+(id_t), \mathtt{t} \rangle$ for every t in D. In doing so, we mean that all tuples in D are meant to be true.
2. Second, Φ^{cl} is the set of all pairs $\langle \psi_a^t(0), \mathtt{t} \rangle$, for every tuple t and every constant a occurring in t. In this way, we 'initiate' the construction of t^+ as done by statement line 1 of Algorithm 1, that is by stating that every a in t belongs to t^+. Notice here that the constant 0 in $\psi_a^t(0)$ is arbitrarily chosen.

The set of rules $\mathcal{R}$ contains four sets of rules, denoted $\mathcal{R}^+$, $\mathcal{R}^{cl}$, $\mathcal{R}^-$ and $\mathcal{R}^\pm$, defined below. The first set, $\mathcal{R}^+$, contains the following rules, for every tuple t and every $X \to A$ in $\mathcal{FD}$

$$\begin{aligned}\mathcal{R}^+ : \quad & \varphi_t^+(\xi) \leftarrow \varphi_{a_{i_1}}^+(\xi), \ldots, \varphi_{a_{i_k}}^+(\xi) \text{ for } t = a_{i_1} \ldots a_{i_k} \\ & \varphi_{a_{i_j}}^+(\xi) \leftarrow \varphi_t^+(\xi), \text{ for every } a_{i_j} \text{ occurring in } t \\ & \varphi_a^+(\xi) \leftarrow \varphi_{xa}^+(\zeta), \varphi_x^+(\xi), \text{ for every } X \to A \text{ in } \mathcal{FD}, \text{ every } x \text{ over } X \text{ and} \\ & \text{every } a \text{ in } dom(A)\end{aligned}$$

The set $\mathcal{R}^{cl}$, meant to compute the closure of a tuple t, contains the following rules for every tuple t, every $X \to A$ in $\mathcal{FD}$, every $x = a_{i_1} \ldots a_{i_p}$ over X and every a in $dom(A)$:

$$\mathcal{R}^{cl} : \quad \psi_a^t(0) \leftarrow \varphi_{xa}^+(\xi), \psi_{a_{i_1}}^t(0), \ldots, \psi_{a_{i_p}}^t(0)$$

For every tuple t, every $Y \to B$ in $\mathcal{FD}$ such that $Y \subseteq sch(t)$, denoting $t.Y$ by y, the set $\mathcal{R}^-$ contains the following rules where a and a' are in the same attribute domain $dom(A)$:

$$\mathcal{R}^- : \quad \varphi_t^-(\xi) \leftarrow \varphi_y^+(\xi), \psi_a^t(0), \psi_{a'}^t(0)$$

The following two rules, which constitute the set $\mathcal{R}^{\pm}$, combine for every t the truth values of $\varphi_t^+(i)$ and $\varphi_t^-(j)$ so as to compute a global truth value of t:

$$\mathcal{R}^{\pm}: \quad \varphi_t(0) \leftarrow \varphi_t^+(\xi)$$
$$\neg\varphi_t(0) \leftarrow \varphi_t^-(\xi)$$

Denoting by $\Pi(\Delta)$ the pair $(\Phi, \mathcal{R})$ the semantics of $\Pi(\Delta)$ in the Four-valued logic as defined in [13], denoted by $\Sigma(\Delta)$, satisfies the following property.

Proposition 5 *Given $\Delta = (D, \mathcal{FD})$ and $\Pi(\Delta) = (\Phi, \mathcal{R})$, for every t in $\mathcal{T}$ the following holds:*

- $v_\Delta(t) = \texttt{true}$ *if and only if* $\langle \varphi_t(0), \texttt{t} \rangle \in \Sigma(\Delta)$
- $v_\Delta(t) = \texttt{false}$ *if and only if* $\langle \varphi_t(0), \texttt{f} \rangle \in \Sigma(\Delta)$
- $v_\Delta(t) = \texttt{inc}$ *if and only if* $\langle \varphi_t(0), \texttt{b} \rangle \in \Sigma(\Delta)$
- $v_\Delta(t) = \texttt{unkn}$ *if and only if* $\Sigma(\Delta)$ *contains no pair involving* $\varphi_t(0)$

Proof See Appendix F. □

5.3 Consolidation of two or more Tables

Data consolidation consists in collecting data from multiple, possibly heterogeneous sources and putting them in a single destination. The data from each source usually comes in the form of a CSV file, along with some hints on the data, referred to as metadata [15, 17]. During this process, different data sources are put together, or consolidated, into a single data store. Data consolidation is also related to data merging and to data integration.

When data comes from a broad range of sources, consolidation allows organizations to more easily present data, while also facilitating effective data analysis. Data consolidation techniques reduce inefficiencies, like data duplication, costs related to reliance on multiple databases and multiple data management points.

In this section, we consider a simplified, relational scenario of n sources $\Delta_1 = (D_1, \mathcal{FD}_1), \ldots, \Delta_n = (D_n, \mathcal{FD}_n)$, where each source $\Delta_i = (D_i, \mathcal{FD}_i)$ consists of a table D_i possibly with nulls and functional dependencies $\mathcal{FD}_i$. We then explain how to consolidate these sources in our approach under the following assumptions:

1. All source tables are over the *same* universe U.
2. Consolidation is done in the simplest possible way, namely (a) the consolidated table is the union of the source tables and (b) the set of functional dependencies of the consolidated table is the union of the sets of functional dependencies of the source tables. That is, the sources are consolidated through the pair: $\Delta = (D, \mathcal{FD})$, where $D = \bigcup_{i=1}^{i=n} D_i$ and $\mathcal{FD} = \bigcup_{i=1}^{i=n} \mathcal{FD}_i$.

Relying on the close relationship between our approach and Belnap's Four-valued logic (as seen in the previous section), we investigate the relationship between the truth values a tuple has in the source tables and the truth value the tuple has in the consolidated table.

First, notice that Proposition 5 allows for a 'natural' one-to-one mapping h from our set $\mathsf{Four} = \{\texttt{true}, \texttt{inc}, \texttt{unkn}, \texttt{false}\}$ to Belnap's set $\mathcal{FOUR} = \{\texttt{t}, \texttt{b}, \texttt{n}, \texttt{f}\}$, where h is defined by: $h(\texttt{true}) = \texttt{t}$, $h(\texttt{inc}) = \texttt{b}$, $h(\texttt{unkn}) = \texttt{n}$ and $h(\texttt{false}) = \texttt{f}$. Then, the connector $\oplus$ defined on $\mathcal{FOUR}$ induces a connector $\overline{\oplus}$ over Four defined by: $\texttt{v}_1 \overline{\oplus} \texttt{v}_2 = h^{-1}(h(\texttt{v}_1) \oplus h(\texttt{v}_2))$ for all $\texttt{v}_1$ and $\texttt{v}_2$.

Moreover, we can define a partial ordering on Four isomorphic to the knowledge ordering of $\mathcal{FOUR}$ that allows us to compare truth values in Four. Denoting this partial ordering by $\lhd$, we have:

$$\texttt{unkn} \lhd \texttt{false} \lhd \texttt{inc} \quad \text{and} \quad \texttt{unkn} \lhd \texttt{true} \lhd \texttt{inc}$$

The following proposition shows that the truth value of a tuple t in the consolidated table is always greater (with respect to $\lhd$) than any of the truth values that t has in the source tables in which it appears. In other words, when consolidating tables, the knowledge about tuples always increases, compared to the knowledge we have about tuples in the source tables.

Proposition 6 *Let $\Delta_i = (D_i, \mathcal{FD}_i)$ $(i = 1, \ldots, n)$ be n data sources over the same universe, and let $\Delta = (D, \mathcal{FD})$ be defined by $D = \bigcup_{i=1}^{i=n} D_i$ and $\mathcal{FD} = \bigcup_{i=1}^{i=n} \mathcal{FD}_i$. For every tuple t the following holds:*

$$\overline{\bigoplus}_{i=1}^{i=n} v_{\Delta_i}(t) \ \lhd \ v_{\Delta}(t).$$

Proof For every $i = 1, \ldots, n$, let $\Delta_i' = (D_i, \mathcal{FD})$. We first prove that for every tuple t, $v_{\Delta_i}(t) \ \lhd \ v_{\Delta_i'}(t)$ holds. Indeed, for every $i = 1, \ldots, n$, let $(D_i)_{ch}$, respectively $(D_i)'_{ch}$, the chased table of D_i with respect to $\mathcal{FD}_i$, respectively $\mathcal{FD}$. Since $\mathcal{FD}_i \subseteq \mathcal{FD}$ holds, it is easy to see that for every q_i in $(D_i)'_{ch}$ there exists q in $(D_i)_{ch}$ such that $q_i \sqsubseteq q$. Consequently, for every q in $\mathcal{T}$, $[q^+]_i \subseteq [q^+]'_i$, where $[q^+]_i$, respectively $[q^+]'_i$, denotes the closure of q in Δ_i, respectively Δ_i'. Therefore, if $\Delta_i \vdash t$, respectively $\Delta_i \mid\!\sim t$, then $\Delta_i' \vdash t$, respectively $\Delta_i' \mid\!\sim t$, and so, for every $i = 1, \ldots, n$, $v_{\Delta_i}(t) \lhd v_{\Delta_i'}(t)$.

Considering Δ_i' $(i = 1, \ldots, n)$ and Δ, it can be seen that for every $i = 1, \ldots, n$ and every q_i in $(D_i)'_{ch}$ there exists q in D_{ch} such that $q_i \sqsubseteq q$. Consequently, for every $i = 1, \ldots, n$, and every q in $\mathcal{T}$, $[q^+]'_i \subseteq q^+$, where q^+ denotes the closure of q in Δ. Therefore, if for some i, $\Delta_i' \vdash t$, respectively $\Delta_i' \mid\!\sim t$, then $\Delta \vdash t$, respectively $\Delta \mid\!\sim t$, and so, for every $i = 1, \ldots, n$, $v_{\Delta_i'}(t) \lhd v_{\Delta}(t)$. The proposition follows from the transitivity of $\lhd$ and from the fact that $\overline{\oplus}$ defines the least upper bound (lub) with respect to $\lhd$, in the same way as $\oplus$ defines the lub with respect to $\preceq_k$. □

In what follows, we identify cases where the equality $\overline{\bigoplus}_{i=1}^{i=n} v_{\Delta_i}(t) = v_{\Delta}(t)$ holds and cases where it does not. To simplify, we assume that $n = 2$, and that the two sources have the same functional dependencies.

First, if for $i = 1$ or $i = 2$, $v_{\Delta_i}(t) = \texttt{inc}$, then the proposition implies that $v_{\Delta}(t) = \texttt{inc}$, because inc is maximal with respect to $\lhd$. In this case, the equality always holds. Another case where the equality holds is if $v_{\Delta_1}(t) = \texttt{true}$ and $v_{\Delta_2}(t) = \texttt{false}$. Indeed, in this case we have $\Delta \vdash t$ and $\Delta \mid\!\sim t$, showing that $v_{\Delta}(t) = \texttt{inc}$. Therefore, $v_{\Delta}(t) = v_{\Delta_1}(t) \,\overline{\oplus}\, v_{\Delta_2}(t)$.

To see cases where the equality $v_{\Delta_1}(t) \,\overline{\oplus}\, v_{\Delta_2}(t) = v_{\Delta}(t)$ does not hold, let $U = \{A, B, C\}$, $\mathcal{FD} = \{B \rightarrow C\}$, $\Delta_1 = (\{abc\}, \mathcal{FD})$ and $\Delta_2 = (\{bc'\}, \mathcal{FD})$. In this case, $\Delta = (D, \mathcal{FD})$ where $D = \{abc, bc'\}$, and so $D_{ch} = \{abc, abc'\}$. Thus:
– $v_{\Delta_1}(b) = v_{\Delta_2}(b) = \texttt{true}$, but $v_{\Delta}(b) = \texttt{inc}$.
– $v_{\Delta_1}(ac') = v_{\Delta_2}(ac') = \texttt{unkn}$, but $v_{\Delta}(ac') = \texttt{inc}$.
We further illustrate Proposition 6 in the following example.

Example 10 We recall that in our introductory example, we have two data sources $\Delta_1 = (D_1, \mathcal{FD})$ and $\Delta_2 = (D_2, \mathcal{FD})$, where $\mathcal{FD} = \{ID \rightarrow A, ID \rightarrow C\}$ and D_1 and D_2 are as shown in Figure 1. Applying Algorithm 2 to D_1 and D_2 produces $(D_1)_{ch}$ and $(D_2)_{ch}$ as shown in Figure 4, and returns $\mathsf{Inc}(\Delta_1) = \mathsf{Inc}(\Delta_2) = \emptyset$.

$(D_1)_{ch}$	Id	K	M	C
	1	k	m	c
	1	k	m'	c
	2	k'	m'	c
	2	k'	m''	c

$(D_2)_{ch}$	Id	K	M	C
	1	k		c
	2	k'	m''	c'

Fig. 4 The chased source tables of our introductory example

Hence, as already mentioned, Δ_1 and Δ_2 are consistent. Referring to Example 9 and Figure 2, applying Proposition 6 entails the following:

- $v_{\Delta_1}(1, k, m, c) = \texttt{true}$, $v_{\Delta_2}(1, k, m, c) = \texttt{unkn}$ and $v_{\Delta}(1, k, m, c) = \texttt{true}$.
  $v_{\Delta_1}(1, k, m', c) = \texttt{true}$, $v_{\Delta_2}(1, k, m', c) = \texttt{unkn}$ and $v_{\Delta}(1, k, m', c) = \texttt{true}$.
  These are cases of equality because $\texttt{true} \,\overline{\oplus}\, \texttt{unkn} = \texttt{true}$.
- $v_{\Delta_1}(2, c) = \texttt{true}$, $v_{\Delta_2}(2, c) = \texttt{false}$ and $v_{\Delta}(2, c) = \texttt{inc}$.
  This is another case of equality because $\texttt{true} \,\overline{\oplus}\, \texttt{false} = \texttt{inc}$.
- $v_{\Delta_1}(2) = \texttt{true}$, $v_{\Delta_2}(2) = \texttt{true}$ and $v_{\Delta}(2) = \texttt{inc}$.
  This is a case where equality does not hold because $\texttt{true} \,\overline{\oplus}\, \texttt{true} \neq \texttt{inc}$. Notice however that $\texttt{true} \lhd \texttt{inc}$ holds. □

5.4 Query Answering

The problem of query answering in presence of inconsistencies has motivated important research efforts during the past two decades and is still the subject of current research. As mentioned in the introductory section, the most popular approaches in the literature are based on the notion of 'repair', a repair of $\mathcal{D}$ being intuitively *a consistent database $\mathcal{R}$ 'as close as possible' to $\mathcal{D}$*.

However, it has been recognized that generating *all* repairs is difficult to implement - if not unfeasible. This is a well known problem in practice which explains, for instance, why data cleansing is a very important but tedious task in the management of databases and data warehouses [16]. This issue has been thoroughly investigated in [14], where it has been shown that computing repairs of a given relational table in the presence of functional dependencies is either polynomial or APX-complete[1], depending on the form of the functional dependencies. The reader is referred to [1] for theoretical results on the complexity of testing whether $\mathcal{R}$ is a repair of $\mathcal{D}$, when considering a more generic context than we do in this work (more than one table and constraints other than functional dependencies). A Prolog based approach for the generation of repairs can be found in [4].

Dealing with repairs without generating them is thus an important issue, also known as *Consistent Query Answering in Inconsistent Databases*. One of the first works in this area is [7] and the problem has since been addressed in the context of various database models (mainly the relational model or deductive database models) and under various types of constraints (first order constraints, key constraints, key foreign-key constraints). Seminal papers in this area are [2] and [24], while an overview of works in this area can be found in [6].

[1] Roughly, APX is the set of NP optimization problems that allow polynomial-time approximation algorithms (source: Wikipedia).

The problem considered in all these works can be stated as follows: Given a database $\mathcal{D}$ with integrity constraints $\mathcal{IC}$, assume that $\mathcal{D}$ is inconsistent with respect to $\mathcal{IC}$. Under this assumption, given a query Q against $\mathcal{D}$, what is the *consistent answer* to Q? The usual approach to alleviate the impact of inconsistent data on the answers to a query is to consider that an answer to Q is consistent if it is present in *every repair* $\mathcal{R}$ of $\mathcal{D}$.

Complexity results regarding the computation of the consistent answer have been widely studied in [8]. For example one important case is when $\mathcal{IC}$ consists in having one key constraint per database relation and Q is a conjunctive query containing no self-join (*i.e.,* no join of a relation with itself). In this case computing the consistent answer is polynomial whereas if self-joins occur then the problem is co-NP-complete.

Another important problem in considering repairs is that there are many ways of defining the notion of repair. This is so because there are many ways of defining a distance between two database instances, and there is no consensus as to the 'best' definition of distance. Although the distance based on symmetric difference seems to be the most popular, other distances exist as well based for example on sub-sets, on cardinality, on updates or on homomorphism [23]. Notice in this respect that the results in [14] are set for two distances: one based on sub-sets and one based on updates.

In our work we do *not* use any notion of repair, thus avoiding the above problem of choosing among all possible ways of defining repairs. Instead, we use set theoretic semantics for tuples and functional dependencies that allow us to associate each tuple with one truth value among true, false, inconsistent or unknown. Users can then query the table as usual by querying the set of its true tuples.

In what follows, we outline the issue of query answering in our approach, and then compare it to the approaches based on repairs. Recalling that consistent query answering refers to tuples in the answers to the query in every repair, we transpose this intuition to our approach by considering that such tuples are those that have truth value `true` in our model.

As usual when dealing with one table with nulls, a query Q is an SQL-like expression of the following form:

$$Q : \mathsf{SELECT}\ X\ [\mathsf{WHERE}\ Condition]$$

where X is an attribute list and the optional WHERE clause specifies a selection condition. Given $\Delta = (D, \mathcal{FD})$, the answer to Q in Δ is the set of the restrictions to X of all tuples for which $Condition$, when present in Q, evaluates to *true*.

In this context, we call *true answer to Q in Δ* the set of all tuples t in the answer to Q such that $v_\Delta(t) = \mathtt{true}$.

To compute such answers, assuming that D_{ch} and $\mathsf{Inc}(\Delta)$ have already been computed, we first build the table D^+ from D_{ch} through the following two steps:

1. Replace each tuple t in D_{ch} by the set of all its maximal sub-tuples (with respect to $\sqsubseteq$) that do not belong to $\mathsf{Inc}(\Delta)$. Let D_{true} be the resulting table.
2. Normalize D_{true} by removing every t such that D_{true} contains a super-tuple of t. Let D^+ be the resulting table.

Denoting by Δ^+ the pair $(D^+, \mathcal{FD})$, the following proposition shows that Δ^+ is consistent and that all true tuples in Δ can be computed based on Δ^+.

Proposition 7 *Given $\Delta = (D, \mathcal{FD})$, let $\Delta^+ = (D^+, \mathcal{FD})$. The following holds:*

1. The time complexity of the construction of D^+ is quadratic in the size of D_{ch}.
2. $D^+_{ch} = D^+$ and Δ^+ is consistent.
3. For every tuple t in $\mathcal{T}$, $v_\Delta(t) = \texttt{true}$ if and only if $v_{\Delta^+}(t) = \texttt{true}$.

Proof See Appendix G. □

Proposition 7 shows that, if D_{ch} and $\mathsf{Inc}(\Delta)$ have been computed beforehand, the cost of computing true answers involves no significant additional cost, as compared with the cost of computing the answer to Q, assuming that D is consistent. We notice moreover in this respect that (i) we have shown that these computations are polynomial, thus expected to be tractable, and (ii) that these computations, including that of D^+, should be seen as *pre*-computations performed once for all, provided that D has not been modified.

This important feature of our approach represents a major advantage over standard approaches to consistent query answering, in which query answering complexity is the major issue. The following example illustrates how queries are answered in our approach and in approaches based on repairs.

Example 11 We recall that in our introductory example of an archaeological site, we defined $\Delta = (D, \mathcal{FD})$ as the result of the integration of two source tables Δ_1 and Δ_2. In this setting, $\mathcal{FD} = \{Id \to A, Id \to C\}$ and the consolidated table D is displayed in Figure 1, whereas its chased version, shown in Figure 2, is recalled in Figure 5 for the sake of readability.

D_{ch}	Id	K	M	C
	1	k	m	c
	1	k	m'	c
	2	k'	m'	c
	2	k'	m'	c'
	2	k'	m''	c
	2	k'	m''	c'

D^+	Id	K	M	C
	1	k	m	c
	1	k	m'	c
		k'	m'	c
		k'	m'	c'
		k'	m''	c
		k'	m''	c'

Fig. 5 The chased consolidated table from the introductory example and its 'true tuple' version

As seen in Example 10, $\mathsf{Inc}(\Delta)$ is the set of all tuples t such that $2 \sqsubseteq t \sqsubseteq q$, where q is one of the last four tuples in D_{ch}. Therefore, computing D^+ in this case amounts to inserting the first two tuples of D_{ch} and replacing each of the last four tuples q with its restriction to the schema $K\,M\,C$, omitting the Id-value.

Consider now the following queries:

Q_1 : SELECT Id, K, C $\quad$ Q_2 : SELECT Id
Q_3 : SELECT K, C $\quad$ Q_4 : SELECT M, C WHERE $K = k'$

Based on the table D^+ shown in Figure 5, it is easy to see that the true answers to queries $Q_1, \ldots, Q_4$ are respectively:

$Q_1 : \{(1, k, c)\}$; $Q_2 : \{1\}$; $Q_3 : \{(k, c), (k', c), (k', c')\}$;
$Q_4 : \{(m, c), (m, c'), (m'', c); (m'', c')\}$.

Considering now the approaches based on repairs, given D_{ch} we have two repairs, R_1 and R_2, as shown in Figure 6.

R_1	Id	K	M	C
	1	k	m	c
	1	k	m'	c
	2	k'	m'	c
	2	k'	m''	c

R_2	Id	K	M	C
	1	k	m	c
	1	k	m'	c
	2	k'	m'	c'
	2	k'	m''	c'

Fig. 6 The two repairs of D_{ch}

Thus answering to queries $Q_1, \ldots, Q_4$ in the repair based approaches yields the following sets of tuples:

$$Q_1 : \{(1, k, c)\} \ ; \ Q_2 : \{1, 2\} \ ; \ Q_3 : \{(k, c)\} \ ; \ Q_4 : \emptyset.$$

This example shows clearly that the two approaches are hardly comparable. Indeed, the answers to Q_1 are equal in the two approaches, the consistent answers to Q_2 and Q_4 in the repair based approaches are larger than the true answers in our approach, and the inverse is true for Q_3.

However, one main difference between the two approaches is the following: as $Id \rightarrow C$ is violated by object 2, this object is inconsistent in our approach, and this implies that no tuple involving identifier 2 can occur in true answers.

On the other hand, in repair based approaches, object 2 occurs in every repair, associated with only one of the C-values responsible for the functional dependency violation, namely c and c'. Consequently, 2 may occur in the consistent answers to queries, as is the case for Q_2.

This is an important difference between the two semantics. We believe that our semantics is justified by the fact that object 2 is in an *inconsistent state* because it is concerned by a dependency violation. Therefore, intuitively, it is 'fair' to consider that 2 cannot occur in any consistent answer! □

6 Concluding Remarks

In this paper we have introduced a novel approach to handle inconsistencies in a table with nulls and functional dependencies. Our approach uses set theoretic semantics and relies on an extended version of the well known chase procedure to associate every possible tuple with one of the four truth values true, false, inconsistent and unknown. Moreover, we have seen that the truth values of tuples can be computed in time polynomial in the size of the input table.

We have also seen that our approach is tightly connected to Belnap's four valued logic and we have seen how it can be applied to the consolidation of two or more tables. Finally, we have discussed the relationship of our approach to consistent query answering based on repairs.

Building upon these results, we currently pursue four lines of research: (1) extending the query language to be able to query the table based on tuple truth values; (2) applying our approach to the particular but important case of key-foreign key constraints in the context of a star schema or a snow-flake schema; (3) designing incremental algorithms to improve performance in case of updates, and (4) extending our approach to account for the presence of tuples declared as *false*.

Declarations

Author contributions: The two authors contributed to the study, conception and design. Both read and approved the submitted manuscript.

Funding: No funds, grants, or other support was received for conducting this study.

Financial interests: The authors declare they have no financial interests.

Non-financial interests: The second author is a member of the editorial board of the journal.

Data availability: Data sharing is not applicable to this article as no datasets were generated or analysed during the current study.

References

1. Foto N. Afrati and Phokion G. Kolaitis. Repair checking in inconsistent databases: algorithms and complexity. In Ronald Fagin, editor, *Database Theory - ICDT 2009, 12th International Conference,Proceedings*, volume 361 of *ACM International Conference Proceeding Series*, pages 31–41. ACM, 2009.
2. Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In Victor Vianu and Christos H. Papadimitriou, editors, *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Pennsylvania, USA*, pages 68–79. ACM Press, 1999.
3. Ofer Arieli and Arnon Avron. The value of the four values. *Artif. Intell.*, 102(1):97–141, 1998.
4. Ofer Arieli, Marc Denecker, Bert Van Nuffelen, and Maurice Bruynooghe. Computational methods for database repair by signed formulae. *Ann. Math. Artif. Intell.*, 46(1-2):4–37, 2006.
5. Nuel D. Belnap. A useful four-valued logic. In J. Michael Dunn and George Epstein, editors, *Modern Uses of Multiple-Valued Logic*, pages 5–37", isbn="978–94–010–1161–7, Dordrecht, 1977. Springer Netherlands.
6. Leopoldo E. Bertossi. *Database Repairing and Consistent Query Answering*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.
7. François Bry. Query answering in information systems with integrity constraints. In Sushil Jajodia, William List, Graeme W. McGregor, and Leon Strous, editors, *Integrity and Internal Control in Information Systems*, volume 109 of *IFIP Conference Proceedings*, pages 113–130. Chapman Hall, 1997.
8. Andrea Calì, Domenico Lembo, and Riccardo Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In Frank Neven, Catriel Beeri, and Tova Milo, editors, *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 9-12, 2003, San Diego, CA, USA*, pages 260–271. ACM, 2003.
9. Walter Alexandre Carnielli and João Marcos. Ex contradictione non sequitur quodlibet. *Bulletin of Advanced Reasoning and Knowledge*, 1:89–109, 2001.
10. S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases*. Surveys in Computer Science, Springer Verlag, 1990.
11. Ronald Fagin, Alberto O. Mendelzon, and Jeffrey D. Ullman. A simplified universal relation assumption and its properties. *ACM Trans. Database Syst.*, 7(3):343–360, 1982.
12. Melvin Fitting. Bilattices and the semantics of logic programming. *J. Log. Program.*, 11(1&2):91–116, 1991.
13. Dominique Laurent. 4-valued semantics under the OWA: A deductive database approach. In Giorgos Flouris, Dominique Laurent, Dimitris Plexousakis, Nicolas Spyratos, and Yuzuru Tanaka, editors, *Information Search, Integration, and Personalization - 13th International Workshop, ISIP, Revised Selected Papers*, volume 1197 of *Communications in Computer and Information Science*, pages 101–116. Springer, 2019.
14. Ester Livshits, Benny Kimelfeld, and Sudeepa Roy. Computing optimal repairs for functional dependencies. *ACM Trans. Database Syst.*, 45(1):4:1–4:46, 2020.

15. Cedrine Madera and Anne Laurent. The next information architecture evolution: The data lake wave. In *Proceedings of the 8th International Conference on Management of Digital EcoSystems*, MEDES, pages 174–180, New York, NY, USA, 2016. ACM.
16. Erhard Rahm and Hong Hai Do. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.
17. Franck Ravat and Yan Zhao. Data lakes: Trends and perspectives. In Sven Hartmann, Josef Küng, Sharma Chakravarthy, Gabriele Anderst-Kotsis, A Min Tjoa, and Ismail Khalil, editors, *Database and Expert Systems Applications - 30th International Conference, DEXA, Proceedings, Part I*, volume 11706 of *Lecture Notes in Computer Science*, pages 304–313. Springer, 2019.
18. Raymond Reiter. On closed world data bases. In Hervé Gallaire and Jack Minker, editors, *Logic and Data Bases, Symposium on Logic and Data Bases, Centre d'études et de recherches de Toulouse, France, 1977*, Advances in Data Base Theory, pages 55–76, New York, 1977. Plemum Press.
19. Nicolas Spyratos. The partition model: A deductive database model. *ACM Trans. Database Syst.*, 12(1):1–37, 1987.
20. Alexis Tsoukiàs. A first order, four-valued, weakly paraconsistent logic and its relation with rough sets semantics. *Foundations of Computing and Decision Sciences*, 27(2):77–96, 2002.
21. Jeffrey D. Ullman. *Principles of Databases and Knowledge-Base Systems*, volume 1-2. Computer Science Press, 1988.
22. Moshe Y. Vardi. The universal-relation data model for logical independence. *IEEE Softw.*, 5(2):80–85, 1988.
23. Jef Wijsen. Database repairing using updates. *ACM Trans. Database Syst.*, 30(3):722–768, 2005.
24. Jef Wijsen. On the consistent rewriting of conjunctive queries under primary key constraints. *Inf. Syst.*, 34(7):578–601, 2009.

A Proof of Lemma 1

Lemma 1. *For every $\Delta = (D, \mathcal{FD})$, the sequence $(\mu_i)_{i\geq 0}$ has a unique limit μ^* that satisfies that $\mu^* \models \Delta$.*

Moreover, μ^* is such that for all constants α and β, $\Delta \vdash (\alpha \sqcap \beta)$ holds if and only if $\mu^*(\alpha) \cap \mu^*(\beta) \neq \emptyset$ holds.

Proof We recall that the sequence $(\mu_i)_{i\geq 0}$ is defined by the following steps:

1. For every t in D, assign a 'fresh' integer $id(t)$ to t;
2. For every domain constant a let $\mu_0(a) = \{id(t) \mid t \in S \text{ and } a \sqsubseteq t\}$;
3. While there exists $X \to A$ in $\mathcal{FD}$, x over X and a in $dom(A)$ such that $\mu(xa) \neq \emptyset$ and $\mu(x) \not\subseteq \mu(a)$, define μ_{i+1} by: $\mu_{i+1}(a) = \mu_i(a) \cup \mu_i(x)$ and $\mu_{i+1}(\alpha) = \mu_i(\alpha)$ for any other symbol α.

The sequence $(\mu_i)_{i\geq 0}$ is increasing in the sense that for every α, $\mu_i(\alpha) \subseteq \mu_{i+1}(\alpha)$, and bounded in the sense that for every α, $\mu_i(\alpha) \subseteq \{id(t) \mid t \in \Delta)\}$. Hence the sequence has a unique limit. Moreover, for every t in D, $\mu^*(t) \neq \emptyset$ holds because $id(t)$ always belongs to $\mu^*(t)$, and $\mu^* \models \mathcal{FD}$, because otherwise μ^* would not be the limit of the sequence. Therefore $\mu^* \models \Delta$, which shows the first part of the proposition.

Regarding the second part of the lemma, since $\mu^* \models \Delta$, $\Delta \vdash (\alpha \sqcap \beta)$ obviously implies that $\mu^*(\alpha) \cap \mu^*(\beta) \neq \emptyset$. Conversely, assuming that $\mu^*(\alpha) \cap \mu^*(\beta) \neq \emptyset$, we show that $\Delta \vdash (\alpha \sqcap \beta)$, that is, for every μ such that $\mu \models \Delta$, $\mu(\alpha) \cap \mu(\beta) \neq \emptyset$. The proof is by induction on the steps of the construction of μ^*. The result first holds for $i = 0$. Indeed, if $\mu_0(\alpha) \cap \mu_0(\beta) \neq \emptyset$ then there exists u in D such that $\alpha \sqsubseteq u$ and $\beta \sqsubseteq u$. Hence for every μ such that $\mu \models \Delta$, we have $\mu(u) \neq \emptyset$ and $\mu(u) \subseteq \mu(\alpha) \cap \mu(\beta)$, implying that $\mu(\alpha) \cap \mu(\beta) \neq \emptyset$ holds.

Now, let i_0 such that $\mu_{i_0}(\alpha) \cap \mu_{i_0}(\beta) = \emptyset$ and $\mu_{i_0+1}(\alpha) \cap \mu_{i_0+1}(\beta)(t) \neq \emptyset$, and assume that for all ζ and η such that $\mu_{i_0}(\zeta) \cap \mu_{i_0}(\eta) \neq \emptyset$, we have that $\mu(\zeta) \cap \mu(\eta) \neq \emptyset$ for every μ such that $\mu \models \Delta$. In this case, by definition of the sequence, there exists $X \to A$ in $\mathcal{FD}$, x over X and a in $dom(A)$ such that $\mu_{i_0}(x) \cap \mu_{i_0}(a) \neq \emptyset$ but $\mu_{i_0}(x) \not\subseteq \mu_{i_0}(a)$. As μ_{i_0+1} changes $\mu_{i_0}(a)$ into $\mu_{i_0}(a) \cup \mu_{i_0}(x)$, either α or β is equal to a, and if for instance $\alpha = a$, $\mu_{i_0}(\beta) \cap \mu_{i_0}(x) \neq \emptyset$. Therefore, by our induction hypothesis, for μ such that $\mu \models \Delta$, $\mu(x) \cap \mu(\alpha)$ and $\mu(\beta) \cap \mu(x)$ are nonempty. Since $\mu(x) \subseteq \mu(a)$ holds, $\mu(\alpha) \cap \mu(\beta) \neq \emptyset$. Therefore, the proof is complete. □

B Proof of Lemma 2

Lemma 2. *Let $\Delta = (D, \mathcal{FD})$. Then $\Delta \vdash (t \preceq a)$ holds if and only if a is in t^+.*

Proof We first prove that if $a \notin t^+$ then there exists μ' such that $\mu' \models \Delta$ and $\mu'(t) \not\subseteq \mu'(a)$, showing that $\Delta \vdash (t \preceq a)$ does not hold.

Let μ^* be as earlier defined and let k be an integer not occurring in any $\mu^*(\alpha)$ for any domain constant α. We define μ' for every α as follows: If α is in t^+, then $\mu'(\alpha) = \mu^*(\alpha) \cup \{k\}$; otherwise, $\mu'(\alpha) = \mu^*(\alpha)$.

By Algorithm 1, every α in t is in t^+, and so, k belongs to $\mu'(t)$. On the other hand, since $a \notin t^+$, $k \notin \mu'(a)$, showing that $\mu'(t) \subseteq \mu'(a)$ does not hold.

To prove that $\mu' \models \Delta$, we first note that for every tuple q in D, $\mu'(q) \neq \emptyset$, because $\mu^*(q) \neq \emptyset$ and $\mu^*(q) \subseteq \mu'(q)$ hold. Regarding functional dependencies, let $Y \rightarrow B$ in $\mathcal{FD}$ such that $\mu' \not\models Y \rightarrow B$. Then there exist y over Y and b in $dom(B)$ such that $\mu'(y) \cap \mu'(b) \neq \emptyset$ and $\mu'(y) \not\subseteq \mu'(b)$. As $\mu^* \models \Delta$, either (i) $\mu^*(y) \cap \mu^*(b) = \emptyset$ or (ii) $\mu^*(y) \cap \mu^*(b) \neq \emptyset$ and $\mu^*(y) \subseteq \mu^*(b)$.

(i) If $\mu^*(y) \cap \mu^*(b) = \emptyset$, then $\mu'(y) \cap \mu'(b) = \{k\}$, and thus, every α in y and b is in t^+. By Algorithm 1, the fact that b is in t^+ implies that all symbols in y are in t^+ and that $\Delta \vdash yb$. This is not possible because $\Delta \vdash yb$ entails that $\mu^*(y) \cap \mu^*(b) \neq \emptyset$.

(ii) If $\mu^*(y) \cap \mu^*(b) \neq \emptyset$, then we have $\mu^*(y) \subseteq \mu^*(b)$ and $\mu'(y) \not\subseteq \mu'(b)$. Thus k is in $\mu'(y)$ but not in $\mu'(b)$ and so, every α in y is in t^+ and b is not in t^+. By Lemma 1, $\mu^*(y) \cap \mu^*(b) \neq \emptyset$ implies that $\Delta \vdash yb$ holds, and so, the condition in the loop line 4 in Algorithm 1 is satisfied, showing that b is in t^+. We thus obtain a contradiction showing that $\mu' \models Y \rightarrow B$, and thus that if $a \notin t^+$ then $\Delta \not\vdash (t \preceq a)$.

Conversely, by Algorithm 1, it is easy to see that at each iteration step, we always have $\mu(t) \subseteq \mu(a)$ for every a added in t^+. Indeed, it holds that for every a in t, every μ satisfies $\mu(t) \subseteq \mu(a)$. Now, assuming that up to step j in the main loop, $\mu(t) \subseteq \mu(b)$ holds for every b in the current value of t^+, let a be coming in t^+ at step $j+1$. In this case, according to the condition in line 4 of Algorithm 1, $\mathcal{FD}$ contains $X \rightarrow A$ such that there exist x and a such that $\Delta \vdash xa$ and for every b in x, $b \in t^+$. Thus for every μ such that $\mu \models \Delta$, we have $\mu(x) \cap \mu(a) \neq \emptyset$ and $\mu(t) \subseteq \mu(b)$ for every b in x. Hence $\mu(t) \subseteq \mu(x)$ and $\mu(x) \subseteq \mu(a)$ hold, thus implying that $\mu(t) \subseteq \mu(a)$. As a consequence $\Delta \vdash (t \preceq a)$ and the proof is complete. □

C Proof of Proposition 1

Proposition 1. *$\Delta = (D, \mathcal{FD})$ is consistent if and only if there exist no tuple t such that $v_\Delta(t) = \texttt{inc}$.*

Proof We first note that if there exists a tuple t such that $v_\Delta(t) = \texttt{inc}$, then $\Delta \vdash t$ and $\Delta \vdash\!\!\sim t$. Hence there exist a and a' in the same attribute domain $dom(A)$ such that $\Delta \vdash t \preceq a \sqcap a'$. Thus every $\mathcal{T}$-mapping μ such that $\mu \models \Delta$ is not an interpretation. Consequently, if Δ is not consistent.

Conversely, assuming that there is no tuple t such that such that $\Delta \vdash t$ and $\Delta \vdash\!\!\sim t$, we prove that there exists an interpretation μ such that $\mu \models \Delta$. Indeed, given μ such that $\mu \models \Delta$, if a and a' are two constants in the same attribute domain A such that $\mu(a) \cap \mu(a') \neq \emptyset$, then we define μ' so as $\mu'(a) \cap \mu'(a') = \emptyset$ and $\mu' \models \Delta$.

To this end, we define $\mu'(a)$ and $\mu'(a')$ by $\mu'(a) = \mu(a) \setminus \mu(a')$ and $\mu'(a') = \mu(a') \setminus \mu(a)$, and we set $\mu'(\alpha) = \mu(\alpha)$ for every α different from a and a'.

• We first prove that for every t in D, $\mu'(t) \neq \emptyset$. Indeed, as $\mu(t) \neq \emptyset$, $\mu'(t) = \emptyset$ implies that $\mu(t) \subseteq \mu(a) \cap \mu(a')$. However, since Δ has no inconsistent tuple, this cannot hold for every μ such that $\mu \models \Delta$. We can thus assume that μ is such that $\mu(t) \not\subseteq \mu(a) \cap \mu(a')$, which implies that $\mu'(t) \neq \emptyset$.

• To show that $\mu' \models \mathcal{FD}$, let $Y \rightarrow B$ in $\mathcal{FD}$ such that $\mu' \not\models Y \rightarrow B$. There exist y over Y and b in $dom(B)$ such that $\mu'(y) \cap \mu'(b) \neq \emptyset$ but $\mu'(y) \not\subseteq \mu'(b)$. Then, since $\mu'(y) \subseteq \mu(y)$ and $\mu'(b) \subseteq \mu(b)$, we have that $\mu(y) \cap \mu(b) \neq \emptyset$, implying that $\mu(y) \subseteq \mu(b)$ holds. As $\mu'(y) \not\subseteq \mu'(b)$, we have $\mu(y) \neq \mu'(y)$ and $\mu(b) = \mu'(b)$. In this case, $\mu(a) \cap \mu(a') \cap \mu(y) \neq \emptyset$ and $\mu(a) \cap \mu(a') \cap \mu(b) = \emptyset$ must hold. This is however not possible because $\mu(y) \subseteq \mu(b)$, and we obtain that $\mu' \models Y \rightarrow B$, thus that $\mu' \models \Delta$, which completes the proof. □

D Proof of Lemma 3

Lemma 3. *Algorithm 2 applied to $\Delta = (D, \mathcal{FD})$ always terminates. Moreover, for every tuple t, $\mu^*(t) \neq \emptyset$ holds if and only if t is in $\mathsf{LoCl}(D_{ch})$.*

Proof Algorithm 2 terminates because this algorithm only inserts tuples into D_{ch} and the inserted tuples are in finite number as they are built up using only constants occurring in Δ.

The proof that for every t in $\mathsf{LoCl}(D_{ch})$, $\mu^*(t) \neq \emptyset$ holds is by induction on the steps of Algorithm 2. If $(D_k)_{k\geq 0}$ denotes the sequence of the states of D_{ch} during the execution, we first note that since $D_0 = D$, for every t in $\mathsf{LoCl}(D_0)$, $\mu_0(t) \neq \emptyset$, thus implying that $\mu^*(t) \neq \emptyset$.

Assuming now that for $i > 0$, for every t in $\mathsf{LoCl}(D_i)$, $\mu^*(t) \neq \emptyset$, we prove the result for every t in $\mathsf{LoCl}(D_{i+1})$. Indeed, let t' in D_{i+1} such that $t \sqsubseteq t'$. If t' is in D_i, the proof is immediate; we thus now assume that t' is not in D_i, that is that t' occurs in D_{i+1} when running Algorithm 2, that is, according to line 9 of Algorithm 2, there exists $X \to A$ in $\mathcal{FD}$, t_1 and t_2 in D_i such that $t_1.X = t_2.X$ and $t' = t_2 a$ where $a = t_1.A$. Thus t_1, t_2 and t' can be respectively written as $t'_1 xa$, $t'_2 x$ and $t'_2 xa$, and by induction, $\mu^*(t_1)$ and $\mu^*(t_2)$ are nonempty. Thus $\mu^*(x) \cap \mu^*(a) \neq \emptyset$, which implies that $\mu^*(x) \subseteq \mu^*(a)$, because $\mu^* \models \mathcal{FD}$. Since $\mu^*(t') = \mu^*(t'_2) \cap \mu^*(x) \cap \mu^*(a)$, it follows that $\mu^*(t') = \mu^*(t'_2) \cap \mu^*(x)$, thus that $\mu^*(t') = \mu^*(t_2)$. Hence $\mu^*(t') \neq \emptyset$. A similar reasoning holds in the case of statement line 12.

Conversely, we show that for every t, if $\mu^*(t) \neq \emptyset$ then t is in $\mathsf{LoCl}(D_{ch})$. The proof is done by induction on the construction of μ^*. By definition of μ_0, it is clear that if $\mu_0(t) \neq \emptyset$ then t is in $\mathsf{LoCl}(D)$ and thus in $\mathsf{LoCl}(D_{ch})$. Now, if we assume that for every $i > 0$ and every t, if $\mu_i(t) \neq \emptyset$ then t belongs to $\mathsf{LoCl}(D_{ch})$, we prove that this result holds for μ_{i+1}.

Indeed, let t such that $\mu_i(t) = \emptyset$ and $\mu_{i+1}(t) \neq \emptyset$. By definition of μ_{i+1}, there exists $X \to A$ in $\mathcal{FD}$, and x and a respectively over X and A such that $\mu_i(x) \cap \mu_i(a) \neq \emptyset$, $\mu_i(x) \not\subseteq \mu_i(a)$ and $\mu_{i+1}(x) \subseteq \mu_{i+1}(a)$. Thus, the only possibility for $\mu_{i+1}(t)$ to be nonempty is that $\mu_i(t) \cap \mu_i(a) = \emptyset$ and $\mu_i(t) \cap \mu_i(x) \neq \emptyset$. Consequently, there exist t_1 and t_2 in $\mathsf{LoCl}(D^+_{ch})$ such that $xa \sqsubseteq t_1$ and $tx \sqsubseteq t_2$. Then, by one of the statements line 9 or line 12 of Algorithm 2, the tuple txa is added in D_{ch}, showing that t is in $\mathsf{LoCl}(D_{ch})$. The proof is therefore complete. □

E Proof of Lemma 4

Lemma 4. *Given $\Delta = (D, \mathcal{FD})$, a tuple t is inconsistent in Δ if and only if $t \in \mathsf{Inc}(\Delta)$.*

Proof We first prove that if t belongs to $\mathsf{Inc}(\Delta)$ then t is inconsistent in Δ. We note in this respect that for every x in $inc(X \to A)$ we have k ($k \geq 2$) constants in $dom(A)$ $a_1, \ldots, a_k$ such that for every $i = 1, \ldots, k$, $xa_i \in \mathsf{LoCl}(D_{ch})$, thus such that $\Delta \vdash xa_i$. Therefore, for every $i = 1, \ldots, k$, a_i belongs to x^+, meaning that, by Lemma 2, $\Delta \vdash (x \preceq a_1 \sqcap \ldots \sqcap a_k)$, thus that x is inconsistent in Δ.

Then, as already noticed, for every t such that $\Delta \vdash t$ and $x \sqsubseteq t$, t is inconsistent in Δ. Therefore every tuple in D that belongs to *temp* due to the statement line 5 in Algorithm 3 is inconsistent in Δ. Moreover, the tuples q inserted in *temp* during the execution of the loop line 6 in Algorithm 3 are all such that there exists q' in *temp* such that $q' = qb$ and $\Delta \vdash (q \preceq q')$. Thus $(q')^+ \subseteq q^+$ showing that, since q' is assumed to be inconsistent in Δ, then so is q. Therefore every tuple in $\mathsf{Inc}(\Delta)$ is inconsistent in Δ.

Conversely, we first show an intermediate result whose statement requires the following additional notation: for every t, T^+_t denotes the set of all attributes B such that there exists b in $dom(B)$ and b in t^+. Then we show the following:

For every t such that $\Delta \vdash t$, there exists q in D_{ch} such that $t \sqsubseteq q$ and $T^+_t \subseteq sch(q)$.

The proof is by induction on the construction of t^+ according to Algorithm 1.
• At the first step, all symbols in D are inserted in t^+, and so, $T^0_t = T$. As $\Delta \vdash t$, D_{ch} contains q such that $q \sqsubseteq t$, and $T^0_t \subseteq sch(q)$.
• At step i of the computation of t^+, we denote by t^i the current value of t^+ and by T^i_t the set of all attributes B such that there exists b in $dom(B)$ such that b is in t^i. We assume that D_{ch} contains q^i such that $t \sqsubseteq q^i$ and $T^i_t \subseteq sch(q^i)$.

Let a in t^{i+1} and not in t^i. By Algorithm 1 there exists $X \to A$ in $\mathcal{FD}$, x over X and a in $dom(A)$ such that $\Delta \vdash xa$ and every b in x is in t^i. By our induction hypothesis, there exists q^i in D_{ch} such that $t \sqsubseteq q^i$, $X \subseteq T^i_t \subseteq sch(q^i)$. Thus, we have $x \sqsubseteq q^i$, and so, by Algorithm 2,

D_{ch} contains q^{i+1} such that $q^i \sqsubseteq q^{i+1}$, $A \in sch(q^{i+1})$ and $q^{i+1}.A = a$. Thus, $t \sqsubseteq q^{i+1}$ and $T_t^{i+1} \subseteq sch(q^{i+1})$. This part of the proof is therefore complete.

Now, assuming that t is inconsistent in Δ implies by Definition 4, that $\Delta \vdash t$ and there exist A in U and a and a' in $dom(A)$ such that $\Delta \vdash (t \preceq a \sqcap a')$.

By the result shown just above, there exist q and q' in D_{ch} such that $t \sqsubseteq q$, $t \sqsubseteq q'$, $T_t^+ \subseteq sch(q)$ and $T_t^+ \subseteq sch(q')$. Moreover as a and a' are in t^+ we have that A is in T_t^+. According to Algorithm 1, this implies that $\mathcal{FD}$ contains $X \to A$ such that $\Delta \vdash xa$, $\Delta \vdash xa'$ and every constant α in x is in t^+. As a consequence, q and q' can be written respectively as yxa and $y'xa'$, and by the statement on line 13 in Algorithm 2, the tuples yxa' and $y'xa$ are in D_{ch} and x belongs to $inc(X \to A)$.

Hence, by the statement on line 5 in Algorithm 3, tx belongs to *temp*. Thus if $x \sqsubseteq t$, t belongs to *temp*, thus to $\mathsf{Inc}(\Delta)$. Otherwise, if $x \sqsubseteq t$ does not hold, we know that every symbol b in x not in t is in t^+, and so, by the statement on line 9 in Algorithm 3, all these symbols are removed one by one from tx to generate tuples that belong to *temp*. Thus t is inserted in *temp* when the tuple processed in the loop line 6 in Algorithm 3 is equal to t, and so, t belongs to $\mathsf{Inc}(\Delta)$, and the proof is complete. □

F Proof of Proposition 5

Proposition 5. *Given $\Delta = (D, \mathcal{FD})$ and $\Pi(\Delta) = (\Phi, \mathcal{R})$, for every t in $\mathcal{T}$ the following holds:*

- $v_\Delta(t) = \texttt{true}$ *if and only if* $\langle \varphi_t(0), \texttt{t} \rangle \in \Sigma(\Delta)$
- $v_\Delta(t) = \texttt{false}$ *if and only if* $\langle \varphi_t(0), \texttt{f} \rangle \in \Sigma(\Delta)$
- $v_\Delta(t) = \texttt{inc}$ *if and only if* $\langle \varphi_t(0), \texttt{b} \rangle \in \Sigma(\Delta)$
- $v_\Delta(t) = \texttt{unkn}$ *if and only if* $\Sigma(\Delta)$ *contains no pair involving* $\varphi_t(0)$

Proof It should first be noticed that $\mathcal{R}$ can be stratified into the following four strata: $\mathcal{R}^+$, $\mathcal{R}^{cl}$, $\mathcal{R}^-$, and $\mathcal{R}^\pm$ among which the first three ones define positive programs. Their semantics is then equal to standard logic program semantics. Moreover, in the framework of the present approach, these strata allow the following computations:
(1) The pairs in Φ^D and the rule in $\mathcal{R}^+$ allow to compute all pairs of the form $\langle \varphi_t^+(n), \texttt{t} \rangle$ that identify all tuples t such that $\Delta \vdash t$ holds. This is so because the rules in $\mathcal{R}^+$ express the construction of μ^*, in the sense that $\langle \varphi_t^+(n), \texttt{t} \rangle$ is in the output if and only if $\mu^*(t) \neq \emptyset$. Then the result is a consequence of Lemma 1.
(2) Based on this result we now argue that the pairs in Φ^{cl} along with rules in $\mathcal{R}^{cl}$ allow for the computation of the closure t^+ of every tuple t. Indeed, by the structure of Algorithm 1, it is clear that for every t, Φ^{cl} corresponds to the result of statement line 1, and then, the rules in $\mathcal{R}^{cl}$ are the transcription of the statements in the loop line 2, in particular of the test in line 4, given that all tuples t such that $\Delta \vdash t$ are known from the previous step. Therefore $\langle \psi_a^t(0), \texttt{t} \rangle$ is output at this stage if and only if a is in t^+.
(3) In this stratum, the rules in $\mathcal{R}^-$ are the transcription of Proposition 2(1), and thus, by Lemma 2, given that the previous two strata have been shown correct, all pairs $\langle \varphi_t^-(n), \texttt{t} \rangle$ output at this stage are exactly those such that $\Delta \nvdash t$ holds.
(4) The output of this stratum combines the truth values of $\varphi_t^+(i)$ and of $\varphi_t^-(j)$ so as to produce the truth value of $\varphi_t(0)$ as the least upper bound of according to the connector $\oplus$, which reflects the four cases in Definition 4. Therefore the proof is complete. □

G Proof of Proposition 7

Proposition 7. *Given $\Delta = (D, \mathcal{FD})$, let $\Delta^+ = (D^+, \mathcal{FD})$. The following holds:*

1. The time complexity of the construction of D^+ is quadratic in the size of D_{ch}.
2. $D_{ch}^+ = D^+$ and Δ^+ is consistent.
3. For every tuple t in $\mathcal{T}$, $v_\Delta(t) = \texttt{true}$ if and only if $v_{\Delta^+}(t) = \texttt{true}$.

Proof 1. Step 1 of the construction of D^+ is clearly in $\mathcal{O}(|D_{ch}| \times |\mathsf{Inc}(\Delta)|)$, that is in $\mathcal{O}(|D_{ch}|^2)$. The result follows from the fact that, as already noticed regarding Algorithm 2, the normalization step 2 of the construction of D^+ is also in $\mathcal{O}(|D_{ch}|^2)$.

2. Assuming that applying Algorithm 2 to D^+ generates changes, implies that D^+ contains two rows for which the conditions in lines 9 or 12 apply. Thus $\mathcal{FD}$ contains $X \rightarrow A$ and D^+ contains two rows of the form y_1x and y_2xa in the case of condition line 9, or of the form y_1xa_1 and y_2xa_2 in the case of condition line 12. The latter case is not possible because it would imply that rows y_1xa_1 and y_2xa_2 are in D_{ch} and thus that these tuples are in $\mathsf{Inc}(\Delta)$. Hence, the former case holds, and by construction of D^+, this implies that D_{ch} contains two rows, each being a super-tuple of y_1x and y_2xa, respectively. Therefore, the rows in D_{ch} are of the form y_1z_1xa' and y_2z_2xa, where a' occurs due to Algorithm 2.

If $a \neq a'$ then x is in $\mathsf{Inc}(\Delta)$, and thus, x cannot occur in D^+, which is a contradiction. We therefore have that $a = a'$ and that y_1xa and y_2xa are true in Δ, thus implying that y_1xa appears in D^+, contrary to our hypothesis. We thus obtain that $D^+_{ch} = D^+$.

Assume that Δ^+ is not consistent, by Proposition 1, there exists a tuple t which is inconsistent in Δ^+. In this case t occurs in a row of D^+ and there exists A in U and a and a' in $dom(A)$ such that $\Delta^+ \vdash (t \preceq a \sqcap a')$. As, by construction of D^+, all potentially true tuples in Δ^+ are also potentially true in Δ, $\Delta \vdash (t \preceq a \sqcap a')$ holds. As t is in $\mathsf{LoCl}(D_{ch})$ we obtain that t is $\mathsf{Inc}(\Delta)$, which is a contradiction with the fact that t occurs in a row of D^+.

3. By Proposition 4, $v_\Delta(t) = \texttt{true}$ if and only if t belongs to $\mathsf{LoCl}(D_{ch}) \backslash \mathsf{Inc}(\Delta)$. By construction of D^+, it follows that t belongs to $\mathsf{LoCl}(D^+)$, and since Δ^+ is consistent, $\mathsf{Inc}(\Delta^+) = \emptyset$, which implies that $v_{\Delta^+}(t) = \texttt{true}$. The proof is therefore complete. □