Consistent Query Answering without Repairs in Tables with Nulls and Functional Dependencies

Dominique Laurent, Nicolas Spyratos

Consistent Query Answering without Repairs in Tables with Nulls and Functional Dependencies

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

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

February 15, 2023

Abstract

In this paper, we study consistent query answering in tables with nulls and functional dependencies. Given such a table T, we consider the set $\mathcal{T}$ of all tuples that can be built up from constants appearing in T; and we use set theoretic semantics for tuples and functional dependencies to characterize the tuples of $\mathcal{T}$ in two orthogonal ways: first as true or false tuples; and then as consistent or inconsistent tuples. Queries are issued against T and evaluated in $\mathcal{T}$.

In this setting, we consider a query Q: *select X from T where Condition* over T and define its consistent answer to be the set of tuples x in $\mathcal{T}$ such that: (a) x is a true and consistent tuple with schema X and (b) there exists a true super-tuple t of x in $\mathcal{T}$ satisfying the condition. We show that, depending on the 'status' that the super-tuple t has in $\mathcal{T}$, there are different types of consistent answer to Q.

The main contributions of the paper are: (a) a novel approach to consistent query answering *not* using table repairs; (b) polynomial algorithms for computing the sets of true/false tuples and the sets of consistent/inconsistent tuples of $\mathcal{T}$; (c) polynomial algorithms in the size of T for computing different types of consistent answer for both conjunctive and disjunctive queries; and (d) a detailed discussion of the differences between our approach and the approaches using table repairs.

Keywords: database semantics, inconsistent database, functional dependency, null value, consistent query answering

1 Introduction

In a relational database, each table is seen as a set of tuples that must satisfy a set of functional dependencies. Moreover, each table is assumed to be consistent with the dependencies and its tuples are assumed to be true when users query the table (these are basic assumptions in relational databases). Consistency is verified during updates in the following sense: if the update to be performed results in an inconsistent table then the update is rejected, otherwise it is accepted.

However, the consistency of a table in general is not always easy to verify, for example if the table is the result of integrating data coming from independent sources (e.g., web sources). In such cases the table may contain inconsistent tuples, and the problem is: how to answer user queries so that the answers contain only tuples that are true and consistent. This kind of query answering process is usually referred to as consistent query answering [1, 3].

T	Emp	$Dept$	$Addr$
1	e	d	a
2	e	d	a'
3	e'	d'	a

repairs ⟹

R_1	Emp	$Dept$	$Addr$
1	e	d	a
3	e'	d'	a

R_2	Emp	$Dept$	$Addr$
2	e	d	a'
3	e'	d'	a

Figure 1: An inconsistent table and its two repairs

To see an example, consider the table T of Figure 1, with dependencies $Emp \rightarrow Dept$ and $Dept \rightarrow Addr$. The table T is defined over attributes Emp, $Dept$ and $Addr$, standing for 'employee identifier', 'department identifier' and 'department address', respectively.

This table is inconsistent as the pair of tuples 1 and 2 violates the dependency $Dept \rightarrow Addr$. However, if we ask the SQL query Q : `select` $Emp, Dept$ `from` T, it makes sense to return the set of tuples $\{ed, e'd'\}$ as the answer. Indeed, there is no reason to reject this answer as it is a consistent answer, in the sense that it satisfies the dependency $Emp \rightarrow Dept$. In other words, an inconsistent table might contain some consistent parts (i.e., some useful information) which can be extracted through queries.

The traditional approach to alleviate the impact of inconsistent data on query answers is to introduce the notion of repair: a repair is a maximal consistent sub-set of the table, and a tuple t is in the consistent answer if t is present in the answer from every repair [1, 19]. To illustrate this approach, consider again the table T in Figure 1 with its two repairs, namely $R_1 = \{eda, e'd'a\}$ and $R_2 = \{eda', e'd'a\}$. Both these repairs are consistent with the given dependencies, and maximal with respect to set-theoretic inclusion. The answer to Q from R_1 is $\{ed, e'd'\}$ and so is the answer from R_2. Therefore the consistent answer to Q is $\{ed, e'd'\}$. We note here that the complexity of the query evaluation algorithm in the repairs approach is APX-complete[1] [14].

Now, in several applications today we have to deal with tables containing nulls (i.e., missing values) and having to satisfy a set of functional dependencies. The presence of nulls in a table is due to various reasons such as: 'value does not exist' (e.g., the maiden name of a male employee); 'value exists but is currently unknown' (e.g., the department of a newly hired employee currently following a training course before being assigned to a specific department); and so on. A survey of the different kinds of missing values considered in the literature can be found in [13]. In our approach we assume that missing values are of the kind 'value exists but is currently unknown'.

In the context of the previous example, let $T = \{ed, da, ea'\}$ in which the $Addr$-value in ed, the Emp-value in da and the $Dept$-value in ea' are nulls that is they exist but are currently unknown. Considering the query Q : `select` $Emp, Addr$ `from` T, if T is seen as a regular table, the consistent answer to Q is $\{ea'\}$, as the tuples in T 'seem' to satisfy the functional dependencies $Emp \rightarrow Dept$ and $Dept \rightarrow Addr$.

However, these functional dependencies allow to infer the missing values: from tuples ed and da, it can be inferred that eda is true, and from tuples ed, ea' it can be inferred that eda' is true. This shows that $Dept \rightarrow Addr$ is not satisfied, and thus $\{ea'\}$ should *not* be returned as a

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

consistent answer. The process of inferring missing values in a table is known in the literature as the chase procedure [10, 17, 18], and it is well-known that this procedure *fails* (i.e., stops and returns no table) when encountering an inconsistency. Therefore, the repair-based approaches do not work in the presence of nulls.

In the light of the previous example, we propose a novel approach to consistent query answering that *does not rely on repairs*, but that works in the case where the given set of tuples contains missing values.

In our approach one starts again with a set of tuples T that are not required to be all of them defined over the same set of attributes. Therefore T is represented as a table with nulls. However, nulls simply act as place holders that may receive values implied by functional dependencies, as shown in the previous example. What is significant in our approach is the set $\mathcal{T}$, called the *universe of discourse* of T, containing all tuples that can be built up from values appearing in T. In our previous example, $\mathcal{T}$ consists of the tuples eda and eda' together with all their sub-tuples.

Queries are addressed to T and consistent answers are obtained from $\mathcal{T}$. To achieve this we associate every tuple t in $\mathcal{T}$ with a *set* of identifiers, referred to as the interpretation of t. In doing so, we define set-theoretic semantics for tuples and for functional dependencies which allows us to characterize the tuples of $\mathcal{T}$ along two orthogonal dimensions: a tuple of $\mathcal{T}$ can be *true* or *false* on one hand and *consistent* or *inconsistent* on the other hand.

In this setting, we consider a query Q : `select` X `from` T `where` $Condition$ over T and we define its consistent answer as the set of tuples x in $\mathcal{T}$ such that: (a) x is a true and consistent tuple with schema X and (b) there exists a true super-tuple t of x in $\mathcal{T}$ satisfying the condition. We show that, depending on the 'status' that the super-tuple t has in $\mathcal{T}$, there are different types of consistent answer to Q.

The important point to emphasize again is that, in our approach, queries are addressed to T and evaluated using the universe of discourse $\mathcal{T}$. As mentioned earlier, $\mathcal{T}$ is the set of all tuples that one can define using values appearing in T. Actually, the fundamental difference between our approach and all approaches based on repairs is that their universe of discourse is the table T itself. Therefore our approach is simply not comparable to approaches based on repairs. To see this, consider again our previous example but this time with $Emp \rightarrow Addr$ as the only functional dependency.

Now suppose that $T = \{eda, eda', e'da'\}$ and that we ask for the set of all addresses. Then the answer in the repairs approach is $\{a'\}$ because there are only two repairs, $R_1 = \{eda, e'da'\}$ and $R_2 = \{eda', e'da'\}$, and the answers from each repair are respectively $\{a, a'\}$ and $\{a'\}$; whereas in our approach, the answer is empty because a and a' are both inconsistent tuples of $\mathcal{T}$. On the other hand, if $T = \{eda, ed'a'\}$ and we ask for the set of all departments then the answer in the repairs approach is empty because there are only two repairs, $R_1 = \{eda\}$ and $R_2 = \{ed'a'\}$ and none of d, d' is in the answer from both repairs; whereas, in our approach, the answer is $\{d, d'\}$ because both d and d' are true and consistent tuples of $\mathcal{T}$.

Our work builds upon earlier work on partition semantics [8, 17] and on inconsistent tables with missing values [12]. The main contributions of this paper are as follows:

1. We propose a novel approach to consistent query answering in tables with nulls and functional dependencies, without using table repairs.

2. We provide polynomial algorithms for computing (a) the sets of true/false tuples and the sets of consistent/inconsistent tuples of $\mathcal{T}$ and (b) for computing the consistent answer to any conjunctive or disjunctive query over T.

3. We offer a detailed discussion of the differences between our approach and other approaches including the approaches by table repairs.

The remaining of the paper is organized as follows. In Section 2 we first introduce the basic notions and notation of our approach, and we present our set-theoretic semantics for tuples and functional dependencies; in Section 3 we address the issue of consistent query answering, introducing the notion of consistency with respect to a selection condition; in Section 4 we first define the m-Chase algorithm, a chase-like algorithm for computing the sets of true/false tuples and the sets of consistent/inconsistent tuples of $\mathcal{T}$, and then we propose a polynomial algorithm for computing consistent answers; in Section 5 we compare our approach to the repair approaches and discuss other related work; and finally, in Section 6 we offer concluding remarks and outline current work and future perspectives.

2 The Semantics of our Model

In this section we recall basic definitions from the relational model regarding tuples and tables, and we present the set-theoretic semantics that we use for tuples and functional dependencies. Our approach builds upon the so-called "partition model" introduced in [17].

2.1 Terminology and Notation

Following [17], 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 $dom(A_i)$. An element of $\bigcup_{A \in U} dom(A)$ is called a *constant*.

We call *relation schema* (or simply *schema*) any nonempty sub-set 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 schemes S_1 and S_2 is denoted as S_1S_2 instead of $S_1 \cup S_2$.

We define a *tuple* t over U 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* [18] in the sense that each tuple component is an atomic value from the corresponding 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 a tuple s as S.

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 $j = 1, \ldots, k$, $t(A_{i_j}) = a_{i_j}$, where a_{i_j} is in $dom(A_{i_j})$, and $sch(t) = A_{i_1} \ldots A_{i_k}$.

As in [17], we assume that for any two distinct attributes A and B, we have $dom(A) \cap dom(B) = \emptyset$. However, it might be relevant for attribute domains to share values. For instance, in our introductory example, in the presence of attribute Mgr standing for 'manager', the domains of Emp and Mgr are employee identifiers. In such cases, in order to avoid confusion, we denote attribute values as *pairs* of the form $\langle attribute\_name, value \rangle$ and comparisons are assessed with respect to their *value* component only. In order to keep the notation simple we shall omit attribute names as prefixes whenever no ambiguity is possible.

We define a *table* T over U to be any *finite* set of tuples over U (therefore duplicates are not allowed). As tuples over U are partial functions over U, it follows that T may contain nulls.

Given a table T over U, we denote by $\mathcal{T}$ the set of all tuples that can be built up from constants appearing in T; and we call $\mathcal{T}$ the *universe of discourse* of T as it contains all tuples of interest in query processing. Actually, as we shall see shortly, queries are issued against T and

consistent answers are obtained from $\mathcal{T}$. For every relation schema X, we denote by $\mathcal{T}(X)$ the set of all tuples in $\mathcal{T}$ whose schema is X. Formally, $\mathcal{T}(X) = \{t \in \mathcal{T} \mid sch(t) = X\}$.

For every A in U, the set of all values from $dom(A)$ occurring in T is referred to as the *active domain of* A, denoted by $adom(A)$. We denote by $\mathcal{AD}$ the set of all constants appearing in T that is: $\mathcal{AD} = \bigcup_{A \in U} adom(A)$. We emphasize that even when attribute domains are infinite, active domains are always finite and therefore the sets $\mathcal{AD}$ and $\mathcal{T}$ are finite as well.

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

Moreover, $\sqsubseteq$ denotes 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. It is thus important to keep in mind that whenever $t_1 \sqsubseteq t_2$ holds, it is understood that $sch(t_1) \subseteq sch(t_2)$ also holds. The relation $\sqsubseteq$ is clearly a partial order over $\mathcal{T}$. Given a table T, the set of all sub-tuples of the tuples in T is called the *lower closure* of T and it is defined by: $\mathsf{LoCl}(T) = \{q \in \mathcal{T} \mid (\exists t \in T)(q \sqsubseteq t)\}$.

2.2 Partition Semantics for Tuples

In this work, we consider tables T over a set U of attributes, possibly with nulls, and we assume that every tuple t in $\mathcal{T}$ is associated with a unique identifier, $id(t)$. We denote by TID the set of all identifiers of tuples in $\mathcal{T}$; that is: $TID = \{id(t) \mid t \in \mathcal{T}\}$. In our examples, for simplicity, we assume that TID is a set of positive integers. Denoting by $\mathcal{P}(TID)$ the power-set of TID, the following definition of interpretation is in the spirit of [17].

Definition 1 *Let T be a table over U. We call* interpretation *of T any function I from $\mathcal{T}$ to $\mathcal{P}(TID)$ such that:*

- *For every tuple t of T, $I(t) \neq \emptyset$*
- *For every tuple $t = a_1 a_2 \ldots a_n$ of $\mathcal{T}$, $I(t) = I(a_1) \cap I(a_2) \cap \ldots \cap I(a_n)$*

We say that a tuple t in $\mathcal{T}$ is true *in I if $I(t) \neq \emptyset$; otherwise we say that t is* false *in I. We denote by $True(I)$ the set of tuples t that are true in I and by $False(I)$ the set of tuples t that are false in I.* □

Example 1 Let $T = \{ab, a'c\}$ and let I be a function from $\mathcal{T}$ to $\mathcal{P}(TID)$ defined as follows: $I(a) = \{1, 2\}$, $I(a') = \{2\}$, $I(b) = \{1, 2\}$, $I(c) = \{2\}$, $I(ab) = \{1, 2\}$, $I(ac) = \{2\}$, $I(a'b) = \{2\}$, $I(a'c) = \{2\}$, $I(abc) = \{2\}$, $I(a'bc) = \{2\}$. Then I is an interpretation of T as we have:

- $I(ab) \neq \emptyset$ and $I(a'c) \neq \emptyset$ that is the two tuples of T are true in I
- For every tuple t in $\mathcal{T}$, $I(t)$ is indeed the intersection of the interpretations of its components. For example, $I(ac) = I(a) \cap I(c) = \{1, 2\} \cap \{2\} = \{2\}$ and one can verify easily for the remaining tuples of $\mathcal{T}$. □

It is important to note that the first item in Definition 1 expresses a fundamental assumption regarding a relational table T, namely that every tuple t of T is assumed to be true. Two immediate consequences are that (a) $I(a) \neq \emptyset$, for all a in $\mathcal{AD}$ (i.e., every a in $\mathcal{AD}$ is true in I) and (b) if a tuple t is true in I then so is every sub-tuple of t; and if t is false in I then so is every super-tuple of t. As a consequence the set $\mathcal{T}$ is partitioned into true and false tuples that is $True(I) \cup False(I) = \mathcal{T}$ and $True(I) \cap False(I) = \emptyset$.

The interpretations of T can be compared according to the following definition.

Definition 2 *Let T be a table and I, I' two interpretations of T. Then we say that I is less than or equal to I', denoted $I \preceq I'$ if for every t in $\mathcal{T}$, $I(t) \subseteq I'(t)$.* □

It is easy to see that the relation $\preceq$ is a partial ordering over the set of all interpretations of T. Note that if we view the constants of $\mathcal{AD}$ as unary tuples then we have that: if $I \preceq I'$ then for every a in $\mathcal{AD}$, $I(a) \subseteq I'(a)$ holds; and in the opposite direction, if $I(a) \subseteq I'(a)$ holds for every a in $\mathcal{AD}$ then for every t in $\mathcal{T}$, $I(t) \subseteq I'(t)$ also holds that is $I \preceq I'$ holds. Therefore we have that:

- $I \preceq I'$ holds if and only if for every a in $\mathcal{AD}$, $I(a) \subseteq I'(a)$.

Based on this result, in order to verify whether $I \preceq I'$, it is sufficient to do so for every constant in $\mathcal{AD}$ rather than for every tuple in $\mathcal{T}$.

Now, to see the intuition behind the above definitions, think of the identifiers of TID as being objects and of every a in $\mathcal{AD}$ as being an atomic property that each object may have. Then $I(a)$ is the set of objects having property a; and similarly, the intuitive idea behind the interpretation of a tuple is that a tuple, say ab, is the 'conjunction' of the atomic properties a and b. Therefore $I(ab)$ is the set of objects each having both properties a and b, hence $I(ab) = I(a) \cap I(b)$. Clearly, if $I(ab) = \emptyset$ then no object has both properties a and b so the tuple ab is false in I.

As for the ordering of interpretations what $I \preceq I'$ means is that, for every property t of $\mathcal{T}$, the set of objects having property t in I is included in the set of objects having property t in I'.

Another fundamental assumption made in the relational model is that the components of every tuple be atomic. Actually such a tuple is said to satisfy the First Normal Form [18]. To express this assumption using our definition of interpretation, suppose that there are a, a' in $adom(A)$ such that $a \neq a'$ and $I(a) \cap I(a') \neq \emptyset$. Then every tuple whose identifier is in $I(a) \cap I(a')$ violates the First Normal Form as its A-component is $\{a, a'\}$, therefore not atomic. In the light of this observation we introduce the following definition of inconsistent tuple.

Definition 3 *Let T be a table over U and I an interpretation of T. A tuple t in $\mathcal{T}$ is said to be* inconsistent *in I if there exist A in $sch(t)$ and a' in $adom(A)$ such that $t.A \neq a'$ and $I(t) \cap I(a') \neq \emptyset$.*

The set of all inconsistent tuples in I is denoted by $Inc(I)$. A tuple that is not inconsistent in I is said to be consistent *in I, and the set of all consistent tuples in I is denoted by $Cons(I)$. An interpretation I such that $Inc(I) = \emptyset$ is said to be in* First Normal Form. □

For instance, in Example 1, the tuple $t = abc$ is inconsistent in I as A is in $sch(t)$, and a' is in $adom(A)$ such that $t.A \neq a'$ and $I(t) \cap I(a') = \{2\} \neq \emptyset$. Therefore I is *not* in First Normal Form.

In view of the above definition, the set $\mathcal{T}$ is partitioned into inconsistent and consistent tuples that is $Inc(I) \cup Cons(I) = \mathcal{T}$ and $Inc(I) \cap Cons(I) = \emptyset$. Roughly speaking, inconsistent tuples are those that violate the First Normal Form. This relationship between inconsistency and First Normal Form is stated formally in the following lemma.

Lemma 1 *Let I be an interpretation of T. Then I is in First Normal Form if and only if I satisfies the following constraint:*

> ***Partition constraint* (PC):** *For all A in U and for all a, a' in $adom(A)$, if $a \neq a'$ then $I(a) \cap I(a') = \emptyset$.*

Proof. Assume that I satisfies PC and that $Inc(I) \neq \emptyset$. Given t in $Inc(I)$, using the notation of Definition 3 and denoting by a the A-value occurring in t, we have $I(a) \cap I(a') \neq \emptyset$, because $I(t) \subseteq I(a) \cap I(a')$. Hence, I does not satisfy PC, which is a contradiction.

Conversely, if I does not satisfy PC, then there exist A in U and a and a' such that $I(a) \cap I(a') \neq \emptyset$. Therefore, by Definition 3, a and a' are in $Inc(I)$, thus implying that I is not in First Normal Form. The proof is therefore complete. □

Note that, as mentioned in [12], satisfaction of the partition constraint implies that, for all A in U, the set $\{I(a) \mid a \in adom(A)\}$ is a partition of the set $adom(A)$ (whence the term "partition constraint").

An important question remains regarding the definition of an interpretation: is there a systematic way for defining an interpretation I of a given table T which is in First Normal Form? The answer is yes, there is such a "canonical" interpretation for every table T; it is called the *basic interpretation* of T, denoted by I^b and defined as follows:

> ***Basic Interpretation*** I^b**:** For every A in U and every a in $adom(A)$, $I^b(a) = \{id(t) \mid t \in T \text{ and } t.A = a\}$.

In the context of Example 1 where $T = \{ab, a'c\}$, if ab and $a'c$ are respectively associated with identifiers 1 and 2, the associated basic interpretation I^b is defined by $I^b(a) = I^b(b) = \{1\}$ and $I^b(a') = I^b(c) = \{2\}$. Considering $T_1 = \{ab, bc, ac\}$ with respective identifiers 1, 2 and 3, the basic interpretation I^b_1 is defined by $I^b_1(a) = \{1, 3\}$, $I^b_1(b) = \{1, 2\}$ and $I^b_1(c) = \{2, 3\}$.

It is interesting to note that the definition of basic interpretation parallels that of *inverted file* [22, 21]. Indeed, we first recall that an inverted file is an index data structure that maps content to its location within a database file, in a document or in a set of documents. Hence, if we assume that each tuple of T is implemented as a record (possibly with missing values) then we can view T as a file that is as a function I assigning to each value x appearing in the file record the set $I(x)$ of all identifiers of records in which x appears.

The following lemma summarizes important properties of the basic interpretation.

Lemma 2 *Let T be a table and I^b its basic interpretation as defined above. Then:*

1. *I^b is an interpretation satisfying the partition constraint.*
2. *$True(I^b) = \mathsf{LoCl}(T)$ and $True(I^b)$ is the set of all tuples t that belong to $True(I)$ for every interpretation I of T.*
3. *I^b is minimal with respect to $\preceq$ among all interpretations I of T such that for every t in T, $id(t)$ belongs to $I(t)$.*

Proof. **1.** Assume that there exist A in U and a and a' in $adom(A)$ such that $I^b(a) \cap I^b(a') \neq \emptyset$. If i is in $I^b(a) \cap I^b(a') \neq \emptyset$, let t_i be the tuple in T such that $id(t_i) = i$. This implies that the value of $t_i(A)$ is a *and* a', which is impossible because, as t_i is a function, $t_i(A)$ consists of at most one value in $adom(A)$. This part of the proof is thus complete.

2. Since for every t in T, $I^b(t)$ contains $id(t)$ and as for every t' such that $t' \sqsubseteq t$, we have $I^b(t) \subseteq I^b(t')$, it follows that $\mathsf{LoCl}(T) \subseteq True(I^b)$. Conversely, let t not in $\mathsf{LoCl}(T)$ and assume that $I^b(t) \neq \emptyset$. Since t is not in T, it is not associated with an identifier. Hence, $I^b(t)$ contains identifiers of other tuples, and if q is such a tuple, then by definition of I^b, this implies that $id(q)$ belongs to every component of t, that is that t is a sub-tuple of q. A contradiction showing that $True(I^b) \subseteq \mathsf{LoCl}(T)$. Hence we have $True(I^b) = \mathsf{LoCl}(T)$. Moreover, since for every interpretation I of T, $True(I)$ must at least contain all tuples in T along with their sub-tuples (as argued just above), we have $\mathsf{LoCl}(T) \subseteq True(I)$. Hence, $True(I^b) \subseteq True(I)$, which completes this part of the proof.

3. Let I be an interpretation such that $I \preceq I^b$ and $I \neq I^b$. Thus, there exists a in $\mathcal{AD}$ such that $I(a) \subset I^b(a)$. However, since I is an interpretation such that for every t in T, $id(t)$ belongs

to $I(t)$, if a occurs in t then $id(t)$ belongs to $I(a)$. In other words, $I^b(a) \subseteq I(a)$ holds. A contradiction showing that I^b is minimal with respect to $\preceq$ among all interpretations I of T such that for every t in T, $id(t)$ belongs to $I(t)$. The proof is therefore complete. □

By Definition 3 and Lemma 1, it turns out that, since I^b satisfies the partition constraint PC, I^b is in First Normal Form, therefore $Inc(I^b) = \emptyset$.

From now on, in all our examples we shall use the following convention: the tuples of T will receive successive integer identifiers in the order of their appearance in T, starting with number 1. For example, if $T = \{ab, a'c\}$ as in Example 1, then it is understood that $id(ab) = 1$ and $id(a'c) = 2$. Then, the basic interpretation of T is defined as earlier stated, that is: $I^b(ab) = \{1\}$, $I^b(a'c) = \{2\}$, $I^b(a) = \{1\}$, $I^b(a') = \{2\}$, $I^b(b) = \{1\}$, $I^b(c) = \{2\}$. Thus, $I^b(ac) = \emptyset$, $I^b(abc) = \emptyset$, $I^b(a'bc) = \emptyset$. Therefore we have: $True(I^b) = \{a, a', b, c, ab, a'c\}$ and $False(I^b) = \{ac, abc, a'bc\}$. Note that I^b is an interpretation that satisfies the partition constraint.

2.3 Partition Semantics for Functional Dependencies

Let T be a table over a set U of attributes together with a set FD of functional dependencies. As usual in relational databases [18], a *functional dependency* is an expression of the form $X \to Y$ where X and Y are relation schemes. The notion of functional dependency satisfaction in the context of tables with nulls is stated in the literature [9, 10] in the following terms. A table T satisfies $X \to Y$ if for all tuples t and t' in T the following holds: if XY is a sub-set of $sch(t)$ and of $sch(t')$ then $t.X = t'.X$ implies $t.Y = t'.Y$.

Moreover, it is well-known that a relation satisfies $X \to Y$ if and only if it satisfies the functional dependencies $X \to A$, for every A in Y. This justifies that functional dependencies are usually assumed to be of the form $X \to A$, where X is a relation schema and A is an attribute not in X. In what follows, we make this assumption.

Now, the question that we answer in this section is the following: how can an interpretation I of T express the satisfaction of a functional dependency by T? The answer is provided by the following lemma.

Lemma 3 *Let I be an interpretation of table T in First Normal Form, and let $X \to A$ be a functional dependency of FD. Then T satisfies $X \to A$ if I satisfies the following constraint:*

> ***Inclusion constraint* (IC):** *For all t in $\mathcal{T}$ such that $XA \subseteq sch(t)$, $I(t.X) \cap I(t.A) \neq \emptyset$ implies $I(t.X) \subseteq I(t.A)$.*

Proof. Suppose that I satisfies the inclusion constraint and consider the relation $f : adom(X) \to adom(A)$ defined by: for every true tuple t in T, $f(t.X) = t.A$. We show that f is actually a function. Indeed, suppose there is a true tuple t' in T such that $t'.X = t.X$ and $t'.A \neq t.A$. As I is in First Normal Form, by Lemma 1, I satisfies the partition constraint, and thus, $I(t.A) \cap I(t'.A) = \emptyset$. On the other hand, as I satisfies the inclusion constraint IC for $X \to A$, we have $I(t.X) \subseteq I(t.A)$ and $I(t'.X) \subseteq I(t'.A)$. As $t.X = t'.X$, we have $I(t.X) \subseteq I(t.A) \cap I(t'.A)$, thus that $I(t.X) \subseteq \emptyset$. It follows that $I(t.X) = \emptyset$, which is a contradiction to the fact that $t.X$ is true, being a sub-tuple of t. The proof is therefore complete. □

Example 2 Let $T = \{ab, bc\}$ and $FD = \{A \to B\}$. Then the basic interpretation I^b of T is as follows: $I^b(a) = \{1\}$, $I^b(b) = \{1, 2\}$, $I^b(c) = \{2\}$ and we have that $I^b(a) \cap I^b(b) = \{1\} \neq \emptyset$ and $I^b(a) \subseteq I^b(b)$. As the only tuple of T to which the lemma applies is ab we conclude that T satisfies $A \to B$. In this case I^b satisfies the partition constraint PC *and* the inclusion constraint IC.

As another example, let $T = \{ab, ac\}$ and $FD = \{A \to B\}$. Then the interpretation I^b of T is as follows: $I^b(a) = \{1, 2\}$, $I^b(b) = \{1\}$, $I^b(c) = \{2\}$ and we have $I^b(a) \cap I^b(b) = \{1\} \neq \emptyset$ but

$I^b(a) \not\subseteq I^b(b)$, showing that I^b does not satisfy the inclusion constraint. Therefore, I^b satisfies the partition constraint PC but *not* the inclusion constraint IC. □

In view of Lemma 3, I^b satisfies the partition constraint, the set of true tuples of I^b is equal to the set of true tuples of every interpretation I, and I^b satisfies a minimality property with respect to $\preceq$ among interpretations of T. Therefore the question that arises here is whether we can "expand" I^b to an interpretation I' that satisfies the inclusion constraint as well, so that I' satisfies the same properties as I^b, along with the inclusion constraint IC.

In Example 2, the answer is yes. Indeed, if we add $I^b(a)$ to $I^b(b)$ then $I^b(b)$ becomes $I^b(a) \cup I^b(b)$ and the resulting interpretation is: $I'(a) = \{1, 2\}$, $I'(b) = \{1, 2\}$, $I'(c) = \{2\}$; and I' satisfies the inclusion constraint for $A \rightarrow B$. However, the following example shows that expanding I^b may lead to non satisfaction of the partition constraint PC.

Example 3 Let $T = \{ab, bc, ac'\}$ with $FD = \{A \rightarrow C, B \rightarrow C\}$. Here, I^b is defined by $I^b(a) = \{1, 3\}$, $I^b(b) = \{1, 2\}$, $I^b(c) = \{2\}$ and $I^b(c') = \{3\}$. Thus, I^b does not satisfy the constraint IC because $I^b(a) \cap I^b(c) = \{2\} \neq \emptyset$ but $I^b(a) \not\subseteq I^b(c)$.

Notice that this example shows that the converse of Lemma 3 does not hold. It is so because T satisfies $A \rightarrow C$ and $B \rightarrow C$, although I^b is an interpretation of T in First Normal Form that does not satisfy the inclusion constraint IC.

Expanding I^b as in Example 2 yields the interpretation I' such that $I'(c) = I'(c') = \{1, 2, 3\}$, $I'(a) = I^b(a)$ and $I'(b) = I^b(b)$. It should be clear that I' satisfies the constraint IC, whereas I' does *not* satisfy the constraint PC since $I'(c) \cap I'(c') \neq \emptyset$. As a consequence, by Lemma 1, $Inc(I') \neq \emptyset$. More precisely, as $True(I')$ is the set containing abc and abc' along with their sub-tuples, we have $Inc(I') = \{abc, abc', bc, bc', ac, ac', c, c'\}$. □

In view of our discussion above, the following definition states how we can expand an interpretation I so that it satisfies a given functional dependency $X \rightarrow A$.

Definition 4 *The* expansion of I *with respect to a functional dependency* $X \rightarrow A$ *and a tuple* xa *in* $\mathcal{T}(XA)$ *is the interpretation* $Exp(I, xa)$ *defined as follows:*

- *If* $I(x) \cap I(a) \neq \emptyset$ *and* $I(x) \not\subseteq I(a)$ *then:* $Exp(I, xa)(a) = I(a) \cup I(x)$*, and* $Exp(I, xa)(\alpha) = I(\alpha)$ *for* α *in* $\mathcal{AD}$ *different than* a*.*
- *Otherwise,* $Exp(I, xa) = I$*.* □

Of course, the expansion processing should be iterated when several tuples satisfy the above condition and when several functional dependencies are considered. As will be seen next, starting with the basic interpretation I^b and applying expansion repeatedly until a fixed point is reached, we obtain an interpretation I^* such that: (a) I^* satisfies the inclusion constraint IC (but not necessarily the partition constraint PC) (b) a tuple t of $\mathcal{T}$ is true in I^* if and only if t is true in every interpretation I of T satisfying IC and (c) a tuple t of $\mathcal{T}$ is inconsistent in I^* if and only if t is inconsistent in every interpretation I of T satisfying IC.

2.4 The True and the Inconsistent Tuples

To define the limit interpretation I^* of T mentioned above, for a given a table T over universe U with a set FD of functional dependencies, we consider the sequence of interpretations $(I^j)_{j \geq 0}$ of T defined by:

- For $j = 0$, we set I^0 to be the basic interpretation I^b as defined earlier.
- For $j \geq 0$, let I^{j+1} be defined as follows:

– If there exist $X \to A$, x and a such that $Exp(I^j, xa) \neq I^j$, then $I^{j+1} = Exp(I^j, xa)$

– Else $I^{j+1} = I^j$.

The theorem below states that this sequence has a limit with important properties. In this theorem, given a nonempty sub-set S of $\mathcal{AD}$ and an interpretation I, we denote by $I(S)$ the set $\bigcap_{a \in S} I(a)$.

Theorem 1 *The sequence $(I^j)_{j \geq 0}$ is increasing with respect to $\preceq$ and bounded above, therefore it has a limit that we denote by I^*. This limit has the following properties:*

1. *I^* is unique in the sense that it is independent of the order in which expansions are applied (i.e., the construction of I^* has the Church-Rosser property).*
2. *I^* satisfies the inclusion constraint IC.*
3. *For every nonempty sub-set S of $\mathcal{AD}$, $I^*(S) \neq \emptyset$ holds if and only if $I(S) \neq \emptyset$ holds for every interpretation I of T satisfying IC.*

Proof. First, it is clear that for $j \geq 0$, $I^j \preceq I^{j+1}$ holds by definition of the sequence. Now, as for every a in $\mathcal{AD}$ we have that: $I^j(a) \subseteq TID$ for every $j \geq 0$, the sequence is bounded by the interpretation I^{TID} defined by: $I^{TID}(a) = TID$ for every a in $\mathcal{AD}$. Hence, the sequence has a limit, denoted by I^*. The proof of the items in the theorem are as follows:

1. Uniqueness of the limit I^* is shown in Appendix A.

2. Suppose that I^* does not satisfy the inclusion constraint IC. It follows that there exist $X \to A$ in FD and t in $\mathcal{T}$ such that $XA \subseteq sch(t)$, $I^*(t.X) \cap I^*(t.A) \neq \emptyset$ and $I^*(t.X) \not\subseteq I^*(a)$. Thus, assuming j is such that $I^* = I^j$, we have $I^{j+1} \neq I^j$, which implies that I^* is *not* the limit of the sequence, a contradiction.

3. Let S be a nonempty sub-set of $\mathcal{AD}$ such that $I(S) \neq \emptyset$ for every I satisfying IC. Since I^* is an interpretation that satisfies IC, we also have $I^*(S) \neq \emptyset$.

Conversely, if $I^*(S) \neq \emptyset$, we prove by induction that $I(S) \neq \emptyset$ holds for every interpretation I satisfying IC.

• If $I^0(S) \neq \emptyset$, then it follows from Lemma 2 that S does not contain two elements from the same active domain (because I^b satisfies the partition constraint PC). Thus, the elements of S form a tuple t that belongs to $True(I^0)$. It follows that t is a sub-tuple of a tuple q in T. Hence, t belongs to $\mathsf{True}(I)$ for every interpretation I of T, thus for every interpretation of T satisfying the inclusion constraint IC.

• Assume that for every nonempty sub-set Σ of $\mathcal{AD}$, if $I^j(\Sigma) \neq \emptyset$ then $I(\Sigma) \neq \emptyset$ for every I satisfying IC, and let S be such that $I^{j+1}(S) \neq \emptyset$. If $I^j(S) \neq \emptyset$, then $I^{j+1}(S) \neq \emptyset$ holds because $I^j \preceq I^{J+1}$. Considering the case where $I^j(S) = \emptyset$, as $I^{j+1}(S) \neq I^j(S)$, S contains a constant a such that $I^j(a) \neq I^{j+1}(a)$. Denoting $S \setminus \{a\}$ by S_a, we have $I^j(S_a) = I^{j+1}(S_a)$ (because expansion changes the interpretation of only one constant, namely a in this case), and there exist $X \to A$ in FD and x in $\mathcal{T}(X)$ such that $I^j(x) \cap I^j(a) \neq \emptyset$ and $I^j(x) \not\subseteq I^j(a)$. In this case, we have $I^{j+1}(a) = I^j(a) \cup I^j(x)$, and thus

$$\begin{aligned} I^{j+1}(S) &= I^{j+1}(a) \cap I^{j+1}(S_a) \\ &= (I^j(a) \cup I^j(x)) \cap I^j(S_a) \\ &= (I^j(a) \cap I^j(S_a)) \cup (I^j(x) \cap I^j(S_a)) \end{aligned}$$

Since $I^j(S) = I^j(a) \cap I^j(S_a)$ is empty, it follows that $I^j(x) \cap I^j(S_a) \neq \emptyset$ and so, for every I satisfying IC, $I(x) \cap I(a) \neq \emptyset$ and $I(x) \cap I(S_a) \neq \emptyset$. Hence, $I(x) \subseteq I(a)$, and so $I(x) \cap I(S_a) \subseteq I(a) \cap I(S_a)$. As this implies that $I(a) \cap I(S_a) \neq \emptyset$, we obtain that $I(S) \neq \emptyset$, and the proof is complete. □

We note that similar results have been obtained with a slightly different formalism in [12]. Moreover, as a consequence of Theorem 1, for every tuple t in $\mathcal{T}$, we have:

- t belongs to $True(I^*)$ if and only if t belong to $True(I)$ for every interpretation I of T satisfying IC. We denote the set $True(I^*)$ by $\mathsf{True}(\mathcal{T})$ and its tuples are said to be *true in* $\mathcal{T}$. The set $False(I^*)$ is denoted by $\mathsf{False}(\mathcal{T})$ and its tuples are said to be *false in* $\mathcal{T}$.
- t belongs to $Inc(I^*)$ if and only if t belongs to $Inc(I)$ for every interpretation I of T satisfying IC. We denote the set $Inc(I^*)$ by $\mathsf{Inc}(\mathcal{T})$ and its tuples are said to be *inconsistent in* $\mathcal{T}$. The set $Cons(I^*)$ is denoted by $\mathsf{Cons}(\mathcal{T})$ and its tuples are said to be *consistent in* $\mathcal{T}$.

The following proposition lists basic properties of true, false, consistent and inconsistent tuples in $\mathcal{T}$.

Proposition 1 *For every table T over universe U, the following holds:*

1. *If t is in $\mathsf{True}(\mathcal{T})$ then every t' such that $t' \sqsubseteq t$ belongs to $\mathsf{True}(\mathcal{T})$.*
2. $\mathsf{Inc}(\mathcal{T}) \subseteq \mathsf{True}(\mathcal{T})$. *However, the inclusions $\mathsf{Cons}(\mathcal{T}) \subseteq \mathsf{True}(\mathcal{T})$ and $\mathsf{Cons}(\mathcal{T}) \subseteq \mathsf{False}(\mathcal{T})$ do* not *hold.*
3. *Let t be in $\mathsf{Inc}(\mathcal{T})$ and A in $sch(t)$. Let a' be in $adom(A)$ such that $a' \neq t.A$ and $I^*(t) \cap I^*(a') \neq \emptyset$. Then every tuple t' such that $t' \sqsubseteq t$ and A belongs to $sch(t')$ is in $\mathsf{Inc}(\mathcal{T})$.*
4. *It does* not *hold that for t in $\mathsf{Inc}(\mathcal{T})$, every true super-tuple of t is in $\mathsf{Inc}(\mathcal{T})$.*

Proof. **1.** Since t is in $True(I^*)$, $I^*(t) \neq \emptyset$. Thus, as $I^*(t) \subseteq I^*(t')$ (because $t' \sqsubseteq t$), we have $I^*(t) \neq \emptyset$, showing that t' is in $\mathsf{True}(\mathcal{T})$.

2. By Definition 3, if t is in $\mathsf{Inc}(\mathcal{T})$ then there is A in $sch(t)$ and $a' \neq t.A$ in $adom(A)$ such that $I^*(t) \cap I^*(a') \neq \emptyset$. Thus $I^*(t) \neq \emptyset$, which implies that t belong to $\mathsf{True}(\mathcal{T})$.

Regarding the two other inclusions in the item, for $T = \{ab, a'b'\}$ and $FD = \emptyset$, we have $\mathsf{True}(\mathcal{T}) = \mathsf{LoCl}(T)$ and $\mathsf{Inc}(\mathcal{T}) = \emptyset$. Thus, $\mathsf{False}(\mathcal{T}) = \{ab', a'b\}$ and $\mathsf{Cons}(\mathcal{T}) = \mathcal{T}$, showing that $\mathsf{Cons}(\mathcal{T}) \not\subseteq \mathsf{True}(\mathcal{T})$ and $\mathsf{Cons}(\mathcal{T}) \not\subseteq \mathsf{False}(\mathcal{T})$.

3. In this case, we have $I^*(t) \subseteq I^*(t')$ (because $t \sqsubseteq t'$) and as $I^*(t) \cap I^*(a') \neq \emptyset$, we obtain that $I^*(t') \cap I^*(a') \neq \emptyset$. Since $t'.A = t.A$ (because $A \in sch(t')$ and $t \sqsubseteq t'$), we have $a' \neq t'.A$, which implies that t' is in $\mathsf{Inc}(\mathcal{T})$.

4. Let $T = \{ab, ab', bc\}$ with $A \to B$. In this case I^* is defined by $I^*(a) = I^*(b') = \{1, 2\}$, $I^*(b) = \{1, 2, 3\}$ and $I^*(c) = \{3\}$. Thus b is inconsistent, because $I^*(b) \cap I^*(b') \neq \emptyset$, and we argue that bc is *not* inconsistent although bc is a true super-tuple of b. This is so because $I^*(bc) = \{3\}$ but $I^*(b') \cap I^*(bc) = \emptyset$. □

We illustrate now the construction of I^* through the following example.

Example 4 Referring to Example 3, we recall that $T = \{ab, bc, ac'\}$ with $FD = \{A \to C, B \to C\}$, and that I^b was found to be defined by $I^b(a) = \{1, 3\}$, $I^b(b) = \{1, 2\}$, $I^b(c) = \{2\}$ and $I^b(c') = \{3\}$. The construction of I^* starts with $I^0 = I^b$, and the following steps are performed:
(1) Considering $A \to C$, we have $I^0(a) \cap I^0(c) \neq \emptyset$ but $I^0(a) \not\subseteq I^0(c)$. Hence $I^1(c) = I^0(c) \cup I^0(a) = \{1, 2, 3\}$, $I^1(a) = \{1, 3\}$, $I^1(b) = \{1, 2\}$ and $I^1(c') = \{3\}$.
(2) Considering again $A \to C$, we have $I^1(a) \cap I^1(c') \neq \emptyset$ but $I^1(a) \not\subseteq I^1(c')$. Hence $I^2(c')$ is set to $I^1(c') \cup I^1(a) = \{1, 3\}$, and $I^2(a) = \{1, 3\}$, $I^2(b) = \{1, 2\}$ and $I^2(c) = \{1, 2, 3\}$.
(3) For $B \to C$, we have $I^2(b) \cap I^2(c') \neq \emptyset$ but $I^2(b) \not\subseteq I^2(c')$. Hence $I^3(c')$ is set to $I^2(c') \cup I^2(b) = \{1, 2, 3\}$, and $I^3(a) = \{1, 3\}$, $I^3(b) = \{1, 2\}$ and $I^3(c) = \{1, 2, 3\}$.

Since I^3 satisfies the inclusion constraint, we obtain $I^* = I^3$. We notice that I^* is equal to the expected interpretation I' defined in Example 3. Moreover, it can be seen that in this example, $\mathsf{True}(\mathcal{T})$ contains the tuples abc and abc' along with all their sub-tuples, meaning that $\mathsf{True}(\mathcal{T}) = \mathcal{T}$, and thus that $\mathsf{False}(\mathcal{T}) = \emptyset$.

Regarding inconsistent tuples, as we have $I^*(c) \cap I^*(c') \neq \emptyset$, it turns out that $\mathsf{Inc}(\mathcal{T}) = \{abc, abc', ac, ac', bc, bc', c, c'\}$, and thus that $\mathsf{Cons}(\mathcal{T}) = \{ab, a, b\}$. □

To further illustrate the impact of functional dependencies over true tuples and over inconsistent tuples, we mention the following two properties: given a table T over U, $X \to A$ in FD and a true tuple xa in $\mathcal{T}(XA)$, then:

1. For every tuple t in $\mathsf{True}(\mathcal{T})$ such that $X \subseteq sch(t)$ and $A \notin sch(t)$, if $t.X = x$ then the tuple ta is in $\mathsf{True}(\mathcal{T})$. In other words, writing t as qx this result expresses the well-known property of relational lossless join: if qx and xa are true, the functional dependency $X \to A$ implies that qxa is true as well.
   This is so because $I^*(x) \subseteq I^*(a)$, $I^*(t) \subseteq I^*(x)$ (because $x \sqsubseteq t$) and $I^*(ta) = I^*(t) \cap I^*(a)$. Hence, $I^*(t) \subseteq I^*(a)$, implying that $I^*(ta) = I^*(t)$. Therefore, $I^*(ta) \neq \emptyset$, because $I^*(t) \neq \emptyset$.

2. For every true super-tuple t of xa (i.e., $t \in \mathsf{True}(\mathcal{T})$, $XA \subseteq sch(t)$ and $t.XA = xa$) and for every a' in $adom(A)$ different than a such that xa' is true (i.e., $xa' \in \mathsf{True}(\mathcal{T})$), if we write t as qxa then $t' = qxa'$ also belongs to $\mathsf{True}(\mathcal{T})$, and t and t' both belong to $\mathsf{Inc}(\mathcal{T})$.
   Indeed, since $I^*(xa)$ and $I^*(xa')$ are nonempty, and I^* satisfies the inclusion constraint, we have $I^*(x) \subseteq I^*(a) \cap I^*(a')$ and $I^*(x) \neq \emptyset$. It follows that $I^*(a) \cap I^*(a') \neq \emptyset$. Moreover, since $t = qxa$ and $I^*(t) \neq \emptyset$, we have $I^*(qx) \neq \emptyset$ and as $I^*(qx) \subseteq I^*(x)$, it follows that $I^*(qx) \subseteq I^*(a) \cap I^*(a')$, that is, $I^*(qxa') = I^*(qx)$. Consequently, $I^*(qxa') \neq \emptyset$, entailing that $t' = qxa'$ is in $\mathsf{True}(\mathcal{T})$. Since $I^*(t) \neq \emptyset$, $I^*(a') \cap I^*(xa) \neq \emptyset$ and $I^*(t) \subseteq I^*(xa)$, t is in $\mathsf{Inc}(\mathcal{T})$ and a similar reasoning shows that t' is also in $\mathsf{Inc}(\mathcal{T})$.

3 Consistent Query Answering

In this section, we first define the syntax of queries that we consider and then we define and discuss four kinds of consistent answer to a query, based on the semantics presented in the previous section.

3.1 Syntax of Queries

We use SQL as the query language and, as we query a single table T, the queries Q that we consider have one of the following two forms:

$$Q : \texttt{select } X \texttt{ from } T \qquad \text{or} \qquad Q : \texttt{select } X \texttt{ from } T \texttt{ where } \Gamma$$

In either of these forms, X is an attribute list seen as a relation schema, and in the second form the `where` clause specifies a selection condition Γ. As in SQL the where clause in a query is optional, the generic form of a query Q is denoted by

$$Q : \texttt{select } X \texttt{ from } T \texttt{ [where } \Gamma\texttt{]}$$

The set of all attributes occurring in Γ is called the *schema of* Γ, denoted by $sch(\Gamma)$; and the attribute set $X \cup sch(\Gamma)$ is called the *schema of* Q, denoted by $sch(Q)$.

A selection condition Γ is a well-formed formula involving the usual connectors $\neg$, $\vee$ and $\wedge$ and built up from atomic Boolean comparisons of one of the following forms: $A\,\theta\,a$ or $A\,\theta\,A'$,

where θ is a comparison predicate, A and A' are attributes in U whose domain elements are comparable through θ, and a is in $dom(A)$.

Given a selection condition Γ, we denote by $Sat(\Gamma)$ the set of all tuples in $\mathcal{T}(sch(\Gamma))$ satisfying Γ, as defined below:

- if Γ is of the the form $A\,\theta\,a$, $Sat(\Gamma) = \{t \in \mathcal{T}(sch(\Gamma)) \mid t.A\,\theta\,a\}$,
- if Γ is of the the form $A\,\theta\,B$, $Sat(\Gamma) = \{t \in \mathcal{T}(sch(\Gamma)) \mid t.A\,\theta\,t.B\}$,
- if Γ is of the form $\Gamma_1 \vee \Gamma_2$, $Sat(\Gamma) = Sat(\Gamma_1) \cup Sat(\Gamma_2)$,
- if Γ is of the form $\Gamma_1 \wedge \Gamma_2$, $Sat(\Gamma) = Sat(\Gamma_1) \cap Sat(\Gamma_2)$,
- if Γ is of the form $\neg\Gamma_1$, $Sat(\Gamma) = \mathcal{T}(sch(\Gamma)) \setminus Sat(\Gamma_1)$.

Moreover, the set of tuples that do *not* satisfy Γ is defined by:

- $Sat^-(\Gamma) = \mathcal{T}(sch(\Gamma)) \setminus Sat(\Gamma)$.

For example, if Sal is an attribute whose active domain is $\{s, s'\}$, we have:

- for $\Gamma_4 = (Sal = s')$, $Sat(\Gamma_4) = \{s'\}$ and $Sat^-(\Gamma_4) = \{s\}$,
- for $\Gamma_5 = (Sal = s \vee Sal = s')$, $Sat(\Gamma_5) = \{s, s'\}$ and $Sat^-(\Gamma_5) = \emptyset$,
- for $\Gamma_6 = (Sal > 10K)$, if $s = 5K$ and $s' = 20K$, $Sat(\Gamma_6) = \{s'\}$ and $Sat^-(\Gamma_6) = \{s\}$, and if $s = 15K$ and $s' = 20K$, $Sat(\Gamma_6) = \{s, s'\}$ and $Sat^-(\Gamma_6) = \emptyset$.

Next, we explain how, in our context, the above syntactic definition of satisfaction of a selection condition can be 'coupled' with semantic considerations, when it comes to defining the notion of consistent answer to a query Q.

Intuitively, given Q : `select` X `from` T `[where` Γ`]`, the least requirements for a tuple x to belong to a consistent answer to Q are the following:

R1 x is in $\mathcal{T}(X)$, i.e., the schema of x is X,

R2 x is in $\mathsf{True}(\mathcal{T})$, i.e., x is a true tuple,

R3 x is in $\mathsf{Cons}(\mathcal{T})$, i.e., x is consistent, and

R4 if Q involves a selection condition Γ then there exists t in $\mathsf{True}(\mathcal{T})$ such that $sch(Q) \subseteq sch(t)$, $x \sqsubseteq t$, and $t.sch(\Gamma)$ is in $Sat(\Gamma)$ (that is, t is a true super-tuple of x whose restriction to attributes in $sch(\Gamma)$ satisfies Γ).

It is important to note that when all the above requirements are satisfied, then the consistent answer coincides with the standard notion of answer to projection-selection queries against a consistent table. Indeed, if T is a relational consistent table whose schema contains all attributes in $sch(Q)$, then the answer to a query Q : `select` X `from` T `[where` Γ`]` is the set of all tuples x such that:

- the schema of x is X (see requirement R1), and
- T contains a tuple t such that t is a super-tuple of x and the restriction of t to all attributes occurring in Γ satisfies Γ (see requirement R4, knowing that t is true since all tuples in T are implicitly assumed to be true).

Moreover, in this case, requirement R2 is also satisfied because all tuples in a consistent relational table are implicitly assumed to be *true*, and requirement R3 is trivial because in a consistent table, every tuple is consistent.

On the other hand, in the presence of inconsistencies, it should be noticed that, based on our semantics as earlier defined, the status of any tuple x in the consistent answer to Q is clearly defined because x is then a *true and consistent tuple of* $\mathcal{T}$ *whose schema is* X. However, the situation is less clear regarding the status of the super-tuple t in requirement R4. More precisely, the question here is whether t should be consistent or not and whether t should satisfy other criteria as well (e.g., is t required to be consistent or is t required to be maximal with respect to $\sqsubseteq$?). The following example illustrates this discussion, showing in particular that the status of t has a significant impact on the consistent answer.

Example 5 Consider the universe $U = \{Emp, Sal, Dept\}$ with the functional dependency $Emp \rightarrow Sal$ and the table $T = \{es, es'd, e's'\}$ over U whose tuples state that employee e has s and s' as salaries, employee e works in department d, and that employee e' has salary s'. The content of T clearly does not satisfy the functional dependency $Emp \rightarrow Sal$ as employee e is assigned *two* distinct salaries s and s'. The limit interpretation I^* of T is defined by $I^*(e) = \{1, 2\}$, $I^*(e') = \{3\}$, $I^*(s) = \{1, 2\}$, $I^*(s') = \{1, 2, 3\}$ and $I^*(d) = \{2\}$. Thus $\mathsf{True}(\mathcal{T})$ consists of the sub-tuples of esd, $es'd$ and of $e's'$, and $\mathsf{Inc}(\mathcal{T}) = \{esd, es'd, es, es', sd, s'd, s, s'\}$. Let us now consider the following queries Q_1, Q_2, Q_3, Q_4, Q_5 and Q_6:

Q_1 : `select` $Emp, Dept$ `from` T
Q_2 : `select` Emp, Sal `from` T
Q_3 : `select` Sal `from` T
Q_4 : `select` Emp `from` T `where` $Sal = s'$
Q_5 : `select` Emp `from` T `where` $Sal = s \vee Sal = s'$
Q_6 : `select` Emp `from` T `where` $Sal > 10K$

Regarding Q_1, since ed is the only true and consistent tuple in $\mathcal{T}(Emp, Dept)$, ed is the only tuple satisfying the requirements R1, R2 and R3 above. Requirement R4 is irrelevant because Q_1 involves no selection condition. Hence ed is a candidate tuple to belong to the consistent answer to Q_1. Notice however that one could object that ed is *not* a good candidate since all its maximal (with respect to $\sqsubseteq$) true super-tuples in $\mathcal{T}$ (namely esd and $es'd$) are inconsistent.

Regarding Q_2, es, es' and $e's'$ are the only tuples satisfying requirements R1 (they all belong to $\mathcal{T}(Emp, Sal)$) and R2 (they all belong to $\mathsf{True}(\mathcal{T})$). However, since es and es' are in $\mathsf{Inc}(\mathcal{T})$, they do not satisfy requirement R3, whereas $e's'$ does since this tuple is in $\mathsf{Cons}(\mathcal{T})$. We moreover notice that, contrary to Q_1 above, $e's'$ has no true strict super-tuple. Hence the consistent answer to Q_2 should be $\{e's'\}$.

The case of Q_3 is clearer because s and s' are the only tuples satisfying requirements R1 and R2, and since these two tuples are in $\mathsf{Inc}(\mathcal{T})$, none of them can satisfy requirement R3. Therefore, the consistent answer to Q_3 should be $\emptyset$.

Let us now turn to queries involving a selection condition. Regarding Q_4, the two tuples satisfying requirements R1 and R2 are e and e', and since e and e' are in $\mathsf{Cons}(\mathcal{T})$, they also satisfy requirement R3. Moreover, we notice that es' is a true super-tuple of e satisfying $\Gamma_4 = (Sal = s')$, showing that e satisfies requirement R4. However, it should be noticed that es' is inconsistent and that es is another true super-tuple of e *not* satisfying Γ_4. Hence, it can thought that e should not belong to the consistent answer to Q_4. On the other hand, e' has only one true super-tuple, namely $e's'$, and since this tuple is in $Sat(\Gamma_4)$, e' satisfies requirement R4 and no further information allows us to think that the salary of e' could be different than s'. As a consequence, the consistent answer to Q_4 is expected to contain e' and possibly e.

Regarding Q_5, for the same reasons as for Q_4, e and e' are the two possible tuples satisfying requirements R1, R2 and R3. Now, and contrary to the case of Q_4, the two true super-tuples of e, namely es and es', belong to $Sat(\Gamma_5)$, thus ensuring that whatever e's salary it satisfies the selection condition. This remark supports the fact that e is a candidate to belong to the consistent answer to Q_5. Of course, as for query Q_1, one could object that since es and es' are inconsistent, e has *not* to belong to the consistent answer to Q_5. It should be noticed here that the case of e' is treated as for Q_4 above, because $e's'$ is a true super-tuple of e' such that s' is in $Sat(\Gamma_5)$. Hence, the consistent answer to Q_5 is expected to be $\{e, e'\}$ or $\{e'\}$, following the choice made regarding inconsistency of the tuple t in requirement R4.

A reasoning similar to those for Q_4 or Q_5 applies to Q_6, depending on the actual values of s and s'. Assuming first that $s = 5K$ and $s' = 20K$, as for Q_4, the expected consistent answer is $\{e'\}$ or possibly $\{e, e'\}$, knowing that e's unique salary may not satisfy the condition. However, if $s = 15K$ and $s' = 20K$, the consistent answer to Q_6 is expected to be as for Q_5, that is, $\{e, e'\}$ or $\{e'\}$, following the choice made regarding inconsistency of the tuple t in requirement R4. □

3.2 Consistent Answers

To account for some of the remarks regarding Q_5 and Q_6 in Example 5, we introduce the notion of *consistency with respect to a selection condition*. To this end, for every relation schema X, we denote by $\mathcal{T}(X^{\uparrow})$ the set of all tuples t in $\mathcal{T}$ such that $X \subseteq sch(t)$. In other words, $\mathcal{T}(X^{\uparrow})$ is the set of all super-tuples of tuples over X, and it follows that $\mathcal{T}(X)$ is a sub-set of $\mathcal{T}(X^{\uparrow})$.

Definition 5 *Given a table T over U, and Γ a selection condition:*

A tuple t in $\mathcal{T}(sch(\Gamma)^{\uparrow})$ is said to be inconsistent with respect to $Sat(\Gamma)$ *if $t.sch(\Gamma)$ is in $Sat(\Gamma)$ and there exists s in $Sat^{-}(\Gamma)$ such that $I^*(t) \cap I^*(s) \neq \emptyset$. We denote by $\mathsf{Inc}(\Gamma, \mathcal{T})$ the set of all tuples inconsistent with respect to $Sat(\Gamma)$.*

A tuple t in $\mathcal{T}(sch(\Gamma)^{\uparrow})$ is said to be consistent with respect to $Sat(\Gamma)$ *if $t.sch(\Gamma)$ is in $Sat(\Gamma)$ and t is not in $\mathsf{Inc}(\Gamma, \mathcal{T})$, that is if for every s in $Sat^{-}(\Gamma)$, $I^*(t) \cap I^*(s) = \emptyset$. We denote by $\mathsf{Cons}(\Gamma, \mathcal{T})$ the set of all tuples consistent with respect to $Sat(\Gamma)$.* □

The following proposition states basic properties implied by Definition 5.

Proposition 2 *Given a table T over U and a selection condition Γ, the following holds:*

1. *$\mathsf{Inc}(\Gamma, \mathcal{T})$ is a sub-set of $\mathsf{Inc}(\mathcal{T})$, and $\mathsf{Cons}(\mathcal{T}) \cap \mathcal{T}(sch(\Gamma)^{\uparrow})$ is a sub-set of $\mathsf{Cons}(\Gamma, \mathcal{T})$. But $\mathsf{Cons}(\Gamma, \mathcal{T}) \subseteq \mathsf{Cons}(\mathcal{T})$ does* not *always hold.*
2. *For every t in $\mathsf{Cons}(\Gamma, \mathcal{T})$, every super-tuple t' of t is also in $\mathsf{Cons}(\Gamma, \mathcal{T})$.*
3. *If I^* is in First Normal Form, then $\mathsf{Inc}(\Gamma, \mathcal{T}) = \emptyset$ and $\mathsf{Cons}(\Gamma, \mathcal{T})$ is the set of all super-tuples of tuples in $Sat(\Gamma)$.*
4. *For a given tuple t, if $Sat(\Gamma) = \{t\}$, then t is in $\mathsf{Inc}(\mathcal{T})$ if and only if t is in $\mathsf{Inc}(\Gamma, \mathcal{T})$, and t is in $\mathsf{Cons}(\mathcal{T})$ if and only if t is in $\mathsf{Cons}(\Gamma, \mathcal{T})$.*
5. *If $Sat(\Gamma) = \emptyset$ then $\mathsf{Cons}(\Gamma, \mathcal{T}) = \mathsf{Inc}(\Gamma, \mathcal{T}) = \emptyset$. If $Sat(\Gamma) = \mathcal{T}(X)$, then $\mathsf{Cons}(\Gamma, \mathcal{T})$ is the set of all super-tuples of tuples in $Sat(\Gamma)$, and $\mathsf{Inc}(\Gamma, \mathcal{T})$ is empty.*

PROOF. **1.** If t is in $\mathsf{Inc}(\Gamma, \mathcal{T})$, there exists s in $Sat^{-}(\Gamma)$ such that $I^*(t) \cap I^*(s) \neq \emptyset$. Since $t.sch(\Gamma)$ is in $Sat(\Gamma)$, $t.sch(\Gamma)$ and s are distinct tuples in $\mathcal{T}(sch(\Gamma))$. Hence, there exists A in $sch(\Gamma)$ such that $t.A \neq s.A$. As $I^*(s) \subseteq I^*(s.A)$, $I^*(t) \cap I^*(s) \neq \emptyset$ implies that $I^*(t) \cap I^*(s.A) \neq \emptyset$. Therefore, by Definition 3, t is in $\mathsf{Inc}(\mathcal{T})$.

As for the inclusion $\mathsf{Cons}(\mathcal{T}) \cap \mathcal{T}(sch(\Gamma)) \subseteq \mathsf{Cons}(\Gamma, \mathcal{T})$, given t in $\mathcal{T}(sch(\Gamma))$, if t is in $\mathsf{Cons}(\mathcal{T})$ then t is not in $\mathsf{Inc}(\mathcal{T})$. Thus, t is not in $\mathsf{Inc}(\Gamma, \mathcal{T})$, which implies by Definition 5 that t is in $\mathsf{Cons}(\Gamma, \mathcal{T})$.

To see that $\mathsf{Cons}(\Gamma, \mathcal{T}) \subseteq \mathsf{Cons}(\mathcal{T})$ does not always hold, consider the earlier example where $T = \{abc, ab'c, a'b''c'\}$ over $U = \{A, B, C\}$ and $C \to B$. In this case, b and b' are clearly in $\mathsf{Inc}(\mathcal{T})$, thus, not in $\mathsf{Cons}(\mathcal{T})$. For $\Gamma = (B = b \vee B = b')$, we have $Sat(\Gamma) = \{b, b'\}$ and $Sat^-(\Gamma) = \{b''\}$. Since I^* is defined by $I^*(a) = I^*(b) = I^*(b') = I^*(c) = \{1, 2\}$, $I^*(a') = I^*(b'') = I^*(c') = \{3\}$, Definition 5 shows that b and b' are in $\mathsf{Cons}(\Gamma, \mathcal{T})$ (because $I^*(b) \cap I^*(b'') = I^*(b') \cap I^*(b'') = \emptyset$).

2. If t' is a super-tuple of t, t' is in $\mathcal{T}(sch(\Gamma)^\uparrow)$ because so is t. Moreover, as t is in $\mathsf{Cons}(\Gamma, \mathcal{T})$, $I^*(t) \cap I^*(s) = \emptyset$ holds for every s in $Sat^-(\Gamma)$. As $I^*(t') \subseteq I^*(t)$, it follows that $I^*(t') \cap I^*(s) = \emptyset$ holds as well, showing that t' is in $\mathsf{Cons}(\Gamma, \mathcal{T})$.

3. Assuming that I^* is in First Normal Form means that $\mathsf{Inc}(\mathcal{T}) = \emptyset$. By (1) above, this implies that $\mathsf{Inc}(\Gamma, \mathcal{T}) = \emptyset$ and thus, by Definition 5, $\mathsf{Cons}(\Gamma, \mathcal{T})$ is the set of all super-tuples of tuples in $Sat(\Gamma)$.

4. By (1) above, we have $\mathsf{Inc}(\Gamma, \mathcal{T}) \subseteq \mathsf{Inc}(\mathcal{T})$ always holds. Conversely, if t is in $\mathsf{Inc}(\mathcal{T})$ then, by Definition 3, there exist A in $sch(t)$ and a' in $adom(A)$ such that $a' \neq t.A$ and $I^*(t) \cap I^*(a') \neq \emptyset$. Denoting $t.A$ by a and writing t as qa, the tuple qa' is such that $sch(qa') = sch(t)$ and $qa' \in Sat^-(\Gamma)$. Hence, $I^*(t) \cap I^*(qa') \neq \emptyset$, showing that t is in $\mathsf{Inc}(\Gamma, \mathcal{T})$.

If t is in $\mathsf{Cons}(\Gamma, \mathcal{T})$, then $Sat(\Gamma) = \mathcal{T}(X) \setminus \{t\}$, and for every s in $Sat^-(\Gamma)$, $I^*(t) \cap I^*(s) = \emptyset$, implying that for every A in $sch(\Gamma)$ and every a' in $adom(A)$ such that $a' \neq t.A$, $I^*(t) \cap I^*(a') = \emptyset$. By Definition 3, this implies that t is in $\mathsf{Cons}(\mathcal{T})$. Conversely, if t is in $\mathsf{Cons}(\mathcal{T})$, by (1) above, as t belongs to $\mathcal{T}(sch(\Gamma))$, t is in $\mathsf{Cons}(\Gamma, \mathcal{T})$.

5. The results being immediate consequences of Definition 5, their proofs are omitted. The proof of the proposition is therefore complete. □

In what follows, in order to account for the remarks in Example 5, we propose four ways of defining the consistent answer to a query Q. These definitions are then illustrated by examples and compared to each other.

Definition 6 *Let T be a table over universe U, FD the set of associated functional dependencies. Given a query Q :* `select` X `from` T `[where` Γ`]`*, we define the following:*

1. *The* weakly consistent answer to Q, *denoted* $\mathsf{WC\_ans}(Q)$, *is the set of all tuples x in $\mathcal{T}(X)$ such that*
   (a) *x is in $\mathsf{True}(\mathcal{T}) \cap \mathsf{Cons}(\mathcal{T})$,*
   (b) *there exists t is in $\mathsf{True}(\mathcal{T})$ such that t is in $\mathcal{T}(sch(Q)^\uparrow)$, $t.X = x$, $t.sch(\Gamma)$ is in $Sat(\Gamma)$.*
2. *The* consistent answer to Q, *denoted* $\mathsf{C\_ans}(Q)$, *is the set of all tuples x in $\mathcal{T}(X)$ such that*
   (a) *x is in $\mathsf{True}(\mathcal{T}) \cap \mathsf{Cons}(\mathcal{T})$,*
   (b) *there exists t is in $\mathsf{True}(\mathcal{T})$ such that t is in $\mathcal{T}(sch(Q)^\uparrow)$, $t.X = x$, t is in $\mathsf{Cons}(\Gamma, \mathcal{T})$.*
3. *The* strongly consistent answer to Q, *denoted* $\mathsf{SC\_ans}(Q)$, *is the set of all tuples x in $\mathcal{T}(X)$ such that*
   (a) *x is in $\mathsf{True}(\mathcal{T}) \cap \mathsf{Cons}(\mathcal{T})$,*
   (b) *there exists t is in $\mathsf{True}(\mathcal{T}) \cap \mathsf{Cons}(\mathcal{T})$ such that t is in $\mathcal{T}(sch(Q)^\uparrow)$, $t.X = x$, and $t.sch(\Gamma)$ is in $Sat(\Gamma)$.*

4. *The* max-strongly consistent answer to Q, *denoted* $\mathsf{MSC\_ans}(Q)$, *is the set of all tuples* x *in* $\mathcal{T}(X)$ *such that*

   (a) x *is in* $\mathsf{True}(\mathcal{T}) \cap \mathsf{Cons}(\mathcal{T})$,

   (b) *there exists* t *is in* $\mathsf{True}(\mathcal{T}) \cap \mathsf{Cons}(\mathcal{T})$ *such that* t *is maximal with respect to* $\sqsubseteq$ *in* $\mathsf{True}(\mathcal{T})$, t *is in* $\mathcal{T}(sch(Q)^{\uparrow})$, $t.X = x$, *and* $t.sch(\Gamma)$ *is in* $Sat(\Gamma)$. □

The four kinds of consistent answer in Definition 6 relate to requirements R1, R2, R3 and R4 in the following respects:

- Statements (1.a), (2.a), (3.a) and (4.a) clearly show that any tuple x in any of the four kinds of consistent answer satisfies requirements R1 ($x \in \mathcal{T}(X)$), R2 ($x \in \mathsf{True}(\mathcal{T})$) and R3 ($x \in \mathsf{Cons}(\mathcal{T})$).
- Statements (1.b), (2.b), (3.b) and (4.b) show that any tuple x in any of the four kinds of consistent answer satisfies the requirement R4 by ensuring the existence of a true super-tuple t of x satisfying Γ.

Moreover, as announced in our earlier discussion, the status of the tuple t is clarified in Definition 6 as follows:

- in the case of the weakly consistent answer, t is only requested to be true (as stated in requirement R4),
- in the case of the consistent answer, t is requested to be true and consistent with respect to Γ (which does not disallow t to be inconsistent),
- in the case of the strongly consistent answer, t is requested to be true and consistent (which of course disallows t to be inconsistent), and
- in the case of the max-strongly consistent answer, t is requested to be true, consistent *and* maximal.

We also emphasize that when the query Q in Definition 6 involves no selection condition, then statements (1.b), (2.b) and (3.b) are reduced to the existence of t such that t is in $\mathsf{True}(\mathcal{T})$, t is in $\mathcal{T}(X^{\uparrow})$ and $t.X = x$. As statements (1.a), (2.a) and (3.a), ensure the existence of such a tuple t, statements (1.b), (2.b) and (3.b) are redundant. However, such is not the case for statement (4.b), because the true and consistent super-tuple of x needed to satisfy (4.a) might *not* be maximal, and thus might not satisfy (4.b). We refer to query Q_1 in the forthcoming Example 6 for an illustration of this case.

Moreover, Definition 6 and the results in Proposition 2 lead to the following observations when the query Q involves a selection condition Γ:

1. For every t in $\mathsf{Inc}(\Gamma, \mathcal{T})$, the tuple $t.sch(\Gamma)$ is such that its syntax satisfies the condition Γ whereas its semantics meets that of a tuple whose syntax does *not* satisfy the condition Γ. Since $\mathsf{Inc}(\Gamma, \mathcal{T})$ is a sub-set of $\mathsf{Inc}(\mathcal{T})$, this shows that the notion of satisfaction of a selection condition might concern inconsistent tuples. It should be noticed that any super-tuple of such tuples is discarded from max-strongly consistent answers and from strongly consistent answers, whereas they may occur in consistent answers or in weakly consistent answers.

2. Since every super-tuple of a tuple in $\mathsf{Cons}(\Gamma, \mathcal{T})$ is also in $\mathsf{Cons}(\Gamma, \mathcal{T})$, $\mathsf{C\_ans}(Q)$ can be computed by scanning maximal true tuples in $\mathcal{T}(sch(Q)^{\uparrow})$. Obviously, $\mathsf{MSC\_ans}(Q)$ should be computed by scanning only maximal true tuples in $\mathcal{T}(sch(Q)^{\uparrow})$, whereas the scan should

concern true tuples in $\mathcal{T}(sch(Q))$, when computing $\mathsf{SC\_ans}(Q)$ or $\mathsf{WC\_ans}(Q)$. It will however be seen in the next section, that all consistent answers are computed based on the same scan.

3. If I^* is in First Normal Form, i.e., if $\mathsf{Inc}(\mathcal{T}) = \emptyset$, then $\mathsf{Cons}(\Gamma, \mathcal{T})$ is the set of all super-tuples of tuples in $Sat(\Gamma)$. As moreover, $t.X$ is in $\mathsf{Cons}(\mathcal{T})$, $t.X$ is in $\mathsf{C\_ans}(Q)$. This remark fits the intuition that if T is consistent then the consistent answer to Q is equal to the standard answer to Q, that is the set of projections over X of true tuples in $\mathcal{T}(sch(Q)^{\uparrow})$ that satisfy Γ. This comes in line with our earlier remark stating that requirements R1, R2, R3 and R4 are satisfied by the tuples in the answer to Q when T is consistent.

4. If $Sat(\Gamma)$ is reduced to one tuple s, then for every t in $\mathcal{T}(sch(Q)^{\uparrow}) \cap \mathsf{True}(\mathcal{T})$ such that $t.X$ is in $\mathsf{Cons}(\mathcal{T}) \cap \mathsf{True}(\mathcal{T})$ and $t.sch(\Gamma) = s$, $t.X$ is in $\mathsf{WC\_ans}(Q)$, and moreover, $t.X$ is in $\mathsf{SC\_ans}(Q)$ and in $\mathsf{C\_ans}(Q)$ if and only if s is in $\mathsf{Cons}(\mathcal{T})$. The case of $\mathsf{MSC\_ans}(Q)$ is more involved because maximal true tuples must be considered. For example, it can happen that, for Q : `select` A `from` T `where` $B = b$, a is in $\mathsf{SC\_ans}(Q)$ and in $\mathsf{C\_ans}(Q)$, but not in $\mathsf{MSC\_ans}(Q)$ because ab is true and consistent, whereas abc is the only true super-tuple of ab and abc is inconsistent.

5. If $Sat(\Gamma) = \emptyset$ (i.e., if Γ is not satisfiable in $\mathcal{T}$), then it is obvious that $\mathsf{MSC\_ans}(Q)$, $\mathsf{SC\_ans}(Q)$ and $\mathsf{WC\_ans}(Q)$ are empty. As in this case, $\mathsf{Cons}(\Gamma, \mathcal{T}) = \emptyset$, $\mathsf{C\_ans}(Q)$ is empty as well.
   On the other hand, if $Sat(\Gamma) = \mathcal{T}(sch(\Gamma))$, it is easy to see that $\mathsf{SC\_ans}(Q)$ and $\mathsf{WC\_ans}(Q)$ are equal to the set of all tuples in $\mathcal{T}(X)$ that belong to $\mathsf{Cons}(\mathcal{T}) \cap \mathsf{True}(\mathcal{T})$. Moreover, since $\mathsf{Cons}(\Gamma, \mathcal{T}) = \mathcal{T}(sch(\Gamma)^{\uparrow})$, $\mathsf{C\_ans}(Q)$ is also the set of all tuples in $\mathcal{T}(X)$ that belong to $\mathsf{Cons}(\mathcal{T}) \cap \mathsf{True}(\mathcal{T})$. However, as in the previous item, the case of $\mathsf{MSC\_ans}(Q)$ is more involved because maximal true tuples must be considered.

The following proposition states an important relationship among the four kinds of consistent answers, namely that they form an increasing chain of inclusions.

Proposition 3 *Let T be a table over universe U, FD the set of associated functional dependencies. Given a query Q :* `select` X `from` T `[where` Γ`]`*, the following holds:* $\mathsf{MSC\_ans}(Q) \subseteq \mathsf{SC\_ans}(Q) \subseteq \mathsf{C\_ans}(Q) \subseteq \mathsf{WC\_ans}(Q)$.

Proof. To show that $\mathsf{MSC\_ans}(Q) \subseteq \mathsf{SC\_ans}(Q)$ holds, let x in $\mathsf{MSC\_ans}(Q)$. Then statements (1.a) and (1.b) in Definition 6 hold, and thus, so does statement (2.a). The result comes form the fact that it is easily seen that statement (1.b) implies statement (2.b).

Regarding the inclusion $\mathsf{SC\_ans}(Q) \subseteq \mathsf{C\_ans}(Q)$, we notice that for every x in $\mathsf{SC\_ans}(Q)$, statements (2.a) and (2.b) in Definition 6 hold, and thus, that statement (3.a) holds as well. Moreover, since statement (2.b) holds, there exists t in $\mathcal{T}(sch(Q)^{\uparrow})$, such that $t.X = x$ and $t.sch(\Gamma)$ is in $Sat(\Gamma)$. Thus t is in $\mathsf{Cons}(\mathcal{T}) \cap \mathcal{T}(sch(\Gamma)^{\uparrow})$, which by Proposition 2(1) implies that t is in $\mathsf{Cons}(\Gamma, \mathcal{T})$. Hence statement (3.b) is satisfied, showing that x is in $\mathsf{C\_ans}(Q)$.

Regarding the last inclusion, for every x in $\mathsf{C\_ans}(Q)$, statement (4.a) clearly holds and moreover, as the tuple t in $\mathsf{Cons}(\Gamma, \mathcal{T}) \cap \mathsf{True}(\mathcal{T})$ is obviously in $\mathsf{True}(\mathcal{T})$ and in $Sat(\Gamma)$, the proof is complete. □

We illustrate Definition 6 and Proposition 3 in the context of Example 5. Moreover, through this example, we show that the inclusions in Proposition 3 might be strict.

Example 6 We recall that in Example 5, $U = \{Emp, Sal, Dept\}$, $FD = \{Emp \rightarrow Sal\}$ and $T = \{es, es'd, e's'\}$, which implies that $\mathsf{True}(\mathcal{T})$ consists of the sub-tuples of esd, $es'd$ and of $e's'$

and that $\mathsf{Inc}(\mathcal{T}) = \{esd, es'd, es, es', sd, s'd, s, s'\}$. The queries we are interested in are Q_1, Q_2, Q_3, Q_4, Q_5 and Q_6 as shown below:

Q_1 : `select` $Emp, Dept$ `from` T
Q_2 : `select` Emp, Sal `from` T
Q_3 : `select` Sal `from` T
Q_4 : `select` Emp `from` T `where` $Sal = s'$
Q_5 : `select` Emp `from` T `where` $Sal = s \vee Sal = s'$
Q_6 : `select` Emp `from` T `where` $Sal > 10K$

In the case of Q_1, the only possible true tuple in any answer is ed. As all maximal true super-tuples of ed are esd and $es'd$, none of these tuples is consistent. Thus, we have $\mathsf{MSC\_ans}(Q_1) = \emptyset$. However, $\mathsf{SC\_ans}(Q_1) = \mathsf{C\_ans}(Q_1) = \mathsf{WC\_ans}(Q_1) = \{ed\}$, because this tuple is true and consistent and because no selection condition is involved. Regarding the inclusions in Proposition 3, this case shows that the first inclusion might be strict.

Considering now Q_2, we have $\mathsf{MSC\_ans}(Q_2) = \{e's'\}$, because this tuple is true and consistent, and has no strict true super-tuple. For a similar reason, we also have $\mathsf{SC\_ans}(Q_2) = \mathsf{C\_ans}(Q_2) = \mathsf{WC\_ans}(Q_2) = \{e's'\}$ because no selection condition is involved in Q_2. In this case the inclusions in Proposition 3 hold because $\mathsf{MSC\_ans}(Q_2) = \mathsf{SC\_ans}(Q_2) = \mathsf{C\_ans}(Q_2) = \mathsf{WC\_ans}(Q_2)$ hold.

The case of Q_3 is easy because as earlier mentioned, no Sal-value is consistent, implying that statement (a) of the four consistent answers in Definition 6 can not be satisfied. As Q_3 involves no selection condition, we have $\mathsf{MSC\_ans}(Q_3) = \mathsf{SC\_ans}(Q_3) = \mathsf{C\_ans}(Q_3) = \mathsf{WC\_ans}(Q_3) = \emptyset$, showing again that the inclusions in Proposition 3 hold.

Regarding Q_4 involving the selection condition $\Gamma_4 = (Sal = s')$, since e and e' are both true and consistent tuples over Emp, they satisfy statements (1.a), (2.a), (3.a) and (4.a) in Definition 6. Since $Sat(\Gamma_3) = \{s'\}$, es' and $e's'$ are the only tuples of interest regarding statements (1.b), (2.b), (3.b) and (4.b) in Definition 6. Thus $\mathsf{WC\_ans}(Q_4) = \{e, e'\}$. As $es'd$ is maximal and inconsistent, statement (1b) is not satisfied and since es' is also inconsistent, statement (2.b) is not satisfied either. Now, regarding statement (3.b), we have s in $Sat^{-}(\Gamma_4)$ and $I^*(es') \cap I^*(s) = \{1, 2\}$ (see Example 5). Thus, by Definition 5, es' is in $\mathsf{Inc}(\Gamma_4, \mathcal{T})$, and so, es' does not satisfy statement (3.b). As a consequence, e is not in $\mathsf{MSC\_ans}(Q_4)$, $\mathsf{SC\_ans}(Q_4)$ nor in $\mathsf{C\_ans}(Q_4)$.

On the other hand, $e's'$ does satisfy statements (1.b), (2.b) and (3.b) in Definition 6, because $e's'$ is a maximal true and consistent tuple such that s' is in $Sat(\Gamma_4)$, and regarding (3.b), $e's'$ is in $\mathsf{Cons}(\Gamma_4, \mathcal{T})$ (because $I^*(e's') \cap I^*(s) = \emptyset$). Therefore, $\mathsf{MSC\_ans}(Q_4) = \mathsf{SC\_ans}(Q_4) = \mathsf{C\_ans}(Q_4) = \{e'\}$ and $\mathsf{WC\_ans}(Q_4) = \{e, e'\}$, showing that the inclusions in Proposition 3 hold, and that the last one might be strict.

As for Q_5 involving the selection condition Γ_5 defined by $(Sal = s \vee Sal = s')$, for the same reasons as for Q_4, the two possible tuples in the answer are e and e' and $\mathsf{WC\_ans}(Q_5) = \{e, e'\}$. Moreover, since $Sat(\Gamma_5) = \{s, s'\}$, e' is in $\mathsf{MSC\_ans}(Q_5)$, $\mathsf{SC\_ans}(Q_5)$ and $\mathsf{C\_ans}(Q_5)$. Regarding now e, the same arguments as for Q_4 apply regarding the statements (1.b) and (2.b) in Definition 6, showing that e is neither in $\mathsf{MSC\_ans}(Q_5)$ nor in $\mathsf{SC\_ans}(Q_5)$. However, contrary to the case of Q_4, es is in $\mathsf{Cons}(\Gamma_5, \mathcal{T})$, and thus, statement (3.b) of Definition 6 is satisfied. We therefore obtain that $\mathsf{MSC\_ans}(Q_5) = \mathsf{SC\_ans}(Q_5) = \{e'\}$ and $\mathsf{C\_ans}(Q_5) = \mathsf{WC\_ans}(Q_5) = \{e, e'\}$, which also shows that the second inclusion in Proposition 3 might be strict.

Consider now Q_6 involving the selection condition $\Gamma_6 = (Sal > 10K)$ and suppose that $s = 5K$ and $s' = 20K$. Then, we have $Sat(\Gamma_6) = \{s'\}$ and $Sat^{-}(\Gamma_6) = \{s\}$, and thus, as in the case of Q_4, $\mathsf{MSC\_ans}(Q_4) = \mathsf{SC\_ans}(Q_4) = \mathsf{C\_ans}(Q_4) = \{e'\}$, and $\mathsf{WC\_ans}(Q_6) = \{e, e'\}$. On the other hand, if $s = 15K$ and $s' = 20K$, then $Sat(\Gamma_6) = \{s, s'\}$ and $Sat^{-}(\Gamma_3) = \emptyset$, and thus, as for

Q_5 above, we have $\mathsf{MSC\_ans}(Q_5) = \mathsf{SC\_ans}(Q_5) = \{e'\}$ and $\mathsf{C\_ans}(Q_5) = \mathsf{WC\_ans}(Q_6) = \{e, e'\}$. □

To end this section, we would like to stress that proposing distinct consistent answers as done in Definition 6 raises the question of *which one should be chosen.* Although we think that this choice should be made by the final user, the comparisons stated in Proposition 3 should help in making this choice. Moreover, assuming the choice of $\mathsf{C\_ans}(Q)$ is made, the knowledge of the answers $\mathsf{MSC\_ans}(Q)$, $\mathsf{SC\_ans}(Q)$ and $\mathsf{WC\_ans}(Q)$ are likely to be useful to the user, regarding the *quality* of the provided consistent answer.

For instance, in Example 6 the fact that e is in $\mathsf{C\_ans}(Q_5)$ but not in $\mathsf{SC\_ans}(Q_5)$ implies that the database contains an inconsistency regarding e's salary. This information shows that the presence of e in $\mathsf{C\_ans}(Q_5)$ might be considered *less reliable* than that of e' for which such inconsistency does not exist as e' is in $\mathsf{SC\_ans}(Q_5)$ and in $\mathsf{SC\_ans}(Q_5)$. On the other hand, a tuple in $\mathsf{WC\_ans}(Q)$ but not in $\mathsf{C\_ans}(Q)$ shows that although this tuple occurs with values satisfying the selection condition, it also systematically occurs in an inconsistency involving values *not* satisfying the selection condition. This happens for e when answering Q_4: although e occurs associated with s', it also occurs associated with s.

This example also shows that comparing the consistent answers $\mathsf{MSC\_ans}(Q)$, $\mathsf{SC\_ans}(Q)$ and $\mathsf{WC\_ans}(Q)$ with $\mathsf{C\_ans}(Q)$ has an impact not only on the *quality* of the answers, but also allows for *explaining* the presence or the absence of some tuples in a consistent answer. These issues related to data quality and to answer explanation are among the subjects of our future work.

4 Computational Issues

In this section, we present algorithms first for computing efficiently the sets $\mathsf{Cons}(\mathcal{T})$ and $\mathsf{Inc}(\mathcal{T})$ (see Section 4.1) and then the four consistent answers to a given query (see Section 4.2).

4.1 The Tabular Representation of I^*

In this section, we show how to compute $\mathsf{True}(\mathcal{T})$ and $\mathsf{Inc}(\mathcal{T})$, using a modified version of the well-known chase algorithm [10, 18] that we *call m-Chase.* Our algorithm is based on the notion of *multi-valued tuple*, or m-tuple for short, which generalizes that of usual tuple in the following sense: instead of associating every attribute A with *at most one value* in $adom(A)$, a multi-valued tuple associates every attribute A with a *sub-set* of $adom(A)$, possibly empty. Multi-valued tuples are defined as follows.

Definition 7 *Given a universe $U = \{A_1, \ldots, A_n\}$ and active domains $adom(A_i)$, $i = 1, \ldots, n$, a* multi-valued tuple *σ over U, or m-tuple for short, is a total function from U to the cross product $\mathsf{X}_{A \in U} \mathcal{P}(adom(A))$, where $\mathcal{P}(adom(A))$ denotes the power set of the active domain of attribute A. The 'empty' m-tuple, denoted by ω, is defined by $\omega(A) = \emptyset$ for every A in U.*

The set of all attributes A such that $\sigma(A) \neq \emptyset$, is called the schema of *σ, denoted by $sch(\sigma)$, and the schema of ω is $\emptyset$. Given $\sigma \neq \omega$ and a sub-set X of $sch(\sigma)$, the restriction of σ to X, denoted $\sigma(X)$, is the m-tuple defined by $(\sigma(X))(A) = \sigma(A)$ for every A in X and $(\sigma(X))(A) = \emptyset$ for any A not in X.*

Given an m-tuple σ, the set $\mathsf{tuples}(\sigma)$ denotes the set of all tuples t such that $sch(t) = sch(\sigma)$ and for every A in $sch(t)$, $t(A)$ is a value of $adom(A)$ that belongs to $\sigma(A)$. As for the empty m-tuple ω, we have $\mathsf{tuples}(\omega) = \emptyset$.

The set of all m-tuples over U is denoted by $\mathcal{MT}$, and a finite set of m-tuples is called an m-table. □

It is easy to see that for every t in $\mathcal{T}$, it holds that $sch(\sigma_t) = sch(t)$ and $\mathsf{tuples}(\sigma_t) = \{t\}$. In what follows, we use t and σ_t interchangeably, whenever no confusion is possible. To simplify notation, given an m-tuple σ, the set $\sigma(A)$ is denoted by the concatenation of its elements between parentheses, and σ itself is denoted by the concatenation of all $\sigma(A)$ such that $\sigma(A) \neq \emptyset$.

Based on Definition 7, a tuple t in $\mathcal{T}$ yields an m-tuple σ_t in $\mathcal{MT}$ such that for every A in $sch(t)$, $\sigma_t(A) = \{t.A\}$ (or $\sigma_t(A) = (t.A)$ according to the notation just introduced), and $\sigma_t(B) = \emptyset$ for every B not in $sch(t)$.

For example if $U = \{A, B, C, D\}$, the m-tuple σ such that $\sigma(A) = \{a, a'\}$, $\sigma(B) = \{b\}$, $\sigma(C) = \emptyset$ and $\sigma(D) = \{d_1, d_2, d_3\}$ is denoted by $(a, a')(b)(d_1d_2d_3)$ and we have $sch(\sigma) = \{A, B, D\}$ (or simply $sch(\sigma) = ABD$). Now, to find the set $\mathsf{tuples}(\sigma)$ we make all combinations of values, one from each of $\sigma(A)$, $\sigma(B)$ and $\sigma(D)$. We find $\mathsf{tuples}(\sigma) = \{abd_1, abd_2, abd_3, a'bd_1, a'bd_2, a'bd_3\}$. Moreover, we have $\sigma(AB) = (aa')(b)$. On the other hand, for $t = abd_1$, we have $\sigma_t = (a)(b)(d_1)$, and thus $\mathsf{tuples}(\sigma_t) = \{abd_1\}$.

We draw attention to the fact that, although m-tables can be seen as a particular case of *embedded relations* [18], we do not intend to elaborate on such relations in our approach. We rather use m-tuples as a tool to explicitly account for situations where the First Normal Form (or equivalently the partition constraint) is not satisfied. We define the following operations for m-tuples:

Comparison: $\sigma_1 \sqsubseteq \sigma_2$ holds if for every A in U, $\sigma_1(A) \subseteq \sigma_2(A)$[2]. Thus if $\sigma_1 \sqsubseteq \sigma_2$ then $sch(\sigma_1) \subseteq sch(\sigma_2)$. Moreover, if $sch(\sigma_1) = sch(\sigma_2)$ then $\mathsf{tuples}(\sigma_1) \subseteq \mathsf{tuples}(\sigma_2)$. It also holds that $\omega \sqsubseteq \sigma$, for every m-tuple σ.

Union: $\sigma_1 \sqcup \sigma_2$ is the m-tuple σ such that for every A in U, $\sigma(A) = \sigma_1(A) \cup \sigma_2(A)$. Thus, $sch(\sigma_1 \sqcup \sigma_2) = sch(\sigma_1) \cup sch(\sigma_2)$.

Intersection: $\sigma_1 \sqcap \sigma_2$ is the m-tuple σ such that for every A in U, $\sigma(A) = \sigma_1(A) \cap \sigma_2(A)$. Thus, $sch(\sigma_1 \sqcap \sigma_2) \subseteq sch(\sigma_i)$ for $i = 1, 2$.

We now introduce a new chase algorithm, called *m-Chase* whose definition is shown as Algorithm 1. The m-Chase Algorithm works on sets of m-tuples to provide an m-table from which true and inconsistent tuples can easily be retrieved. Roughly speaking, Algorithm 1 follows the same idea as the standard chase, but when a functional dependency cannot be satisfied, our algorithm does not stop, but rather collects all values inferred from the functional dependencies. Consequently, it may happen that for a given row and a given column (i.e., for a given attribute value) we may have to store more than one constant, which is achieved by means of m-tuples. It is important to note that the output of Algorithm 1 (denoted by Σ^*) allows to explicitly 'show' all violations of the partition constraint PC. We illustrate how Algorithm 1 works using an example.

Example 7 Let $T = \{ab, ab', a'b'\}$ over universe $U = \{A, B\}$ with functional dependencies $A \rightarrow B$ and $B \rightarrow A$. The computation of Σ^* based on Algorithm 1 works as follows:

1. Considering tuples in T as m-tuples yields the following m-tuples: $(a)(b)$, $(a)(b')$ and $(a')(b')$. Thus, Σ^* is set to $\{(a)(b), (a)(b'), (a')(b')\}$ on line 1.

2. Considering $A \rightarrow B$, with $\sigma = (a)(b)$ and $\sigma' = (a)(b')$, the m-tuples $\sigma_1 = (a)(bb')$ and $\sigma'_1 = (a)(b'b)$ are generated. The variable *change* is set to *true* and since $\sigma_1 = \sigma'_1$, Σ^* is set to $\{(a)(bb'), (a')(b')\}$. Moreover, $fail$ is set to true line 11 because we have $(b) \neq (b')$.

[2]With a slight abuse of notation, we use here the same symbol $\sqsubseteq$ introduced earlier for the sub-tuple relation between usual tuples.

Algorithm 1 The m-Chase Algorithm

Input: A table T over U and a set FD of functional dependencies over U.
Output: An m-table denoted by Σ^* and the truth value of $fail$.

```
1: Σ* := {σ_t | t ∈ T}
2: change := true
3: fail := false
4: while change = true do
5:     change := false
6:     for all σ in Σ* do
7:         for all X → A in FD such that XA ⊆ sch(σ) do
8:             for all σ' in Σ* such that X ⊆ sch(σ') do
9:                 if for every B in X, σ(B) ∩ σ'(B) ≠ ∅ then
10:                    if A ∈ sch(σ) and A ∈ sch(σ') and σ(A) ≠ σ'(A) and fail = false then
11:                        fail := true
12:                    σ1 := σ ⊔ σ'(A) ; σ'1 := σ' ⊔ σ(A)
13:                    if σ ≠ σ1 or σ' ≠ σ'1 then
14:                        change := true
15:                        if σ1 ⊑ σ'1 then
16:                            Σ* := (Σ* \ {σ, σ'}) ∪ {σ'1}
17:                        else if σ'1 ⊑ σ1 then
18:                            Σ* := (Σ* \ {σ, σ'}) ∪ {σ1}
19:                        else
20:                            Σ* := (Σ* \ {σ, σ'}) ∪ {σ1, σ'1}
21: return Σ* and the truth value of fail
```

This means intuitively that when running the standard chase algorithm, a failure would be reached at this step.
Considering $B \to A$, since the B-value b' occurs in $(a)(bb')$ and $(a')(b')$, these m-tuples generate respectively $(aa')(bb')$ and $(aa')(b')$. Moreover, since $(aa')(b') \sqsubseteq (aa')(bb')$, Σ^* is set to $\{(aa')(bb')\}$.

3. As the second execution of the while-loop line 4 does not change Σ^*, the variable *change* has value $false$. Hence the processing stops, returning $\Sigma^* = \{(aa')(bb')\}$ along with the value *true* assigned to $fail$. □

We now state a basic lemma based on which it will be shown how to compute $\mathsf{True}(\mathcal{T})$ and $\mathsf{Inc}(\mathcal{T})$. In what follows, the notation LoCl is extended to Σ^* in a straightforward way: $\mathsf{LoCl}(\Sigma^*) = \{t \in \mathcal{T} \mid (\exists \sigma \in \Sigma^*)(t \sqsubseteq \sigma)\}$. Moreover, for every interpretation I and every m-tuple σ, $I(\sigma)$ is defined by $I(\sigma) = \bigcap_{a \sqsubseteq \sigma} I(a)$. It should be noticed that, as for tuples in $\mathcal{T}$, if $\sigma \sqsubseteq \sigma'$, then $I(\sigma') \subseteq I(\sigma)$ holds.

Lemma 4 *Let T be a table over universe U. Then the following hold:*

1. *Algorithm 1 applied to T always terminates.*
2. *For every m-tuple s in $\mathcal{MT}$, $I^*(s) \neq \emptyset$ holds if and only if there exists σ in Σ^* such that $s \sqsubseteq \sigma$.*
3. *$\mathsf{Inc}(\mathcal{T}) = \emptyset$ if and only if $fail = false$.*
4. *For every σ in Σ^* and all t and t' in $\mathsf{tuples}(\sigma)$, $I^*(t) = I^*(t')$.*

Proof. See Appendix B. □

Note that Algorithm 1 is an extension of the standard chase algorithm. Indeed, the standard chase algorithm works on tuples (and not on m-tuples) which is the case for T. Moreover, as long

as σ and σ' are usual tuples, our processing is similar to that of standard chase. In particular, for every $X \to A$ in FD if XA is a sub-set of $sch(\sigma)$ and $sch(\sigma')$ and if $\sigma(X)$ and $\sigma'(X)$ are equal tuples, while $\sigma(A)$ and $\sigma'(A)$ are distinct values, then, in Algorithm 1, $fail$ is set to $true$, which is precisely a case of failure for the standard chase algorithm. Thus, based on Lemma 4, the returned value for $fail$ is $true$ if and only if the standard chase algorithm fails.

Moreover, the statements lines 16, 18 and 20 in Algorithm 1 imply that when σ and σ' are in Σ^*, for every $X \to A$ in FD such that XA is a sub-set of $sch(\sigma)$ and $sch(\sigma')$, if $\mathsf{tuples}(\sigma(X)) \cap \mathsf{tuples}(\sigma'(X)) \neq \emptyset$, then $\sigma(A) = \sigma'(A)$. This remark can be seen as an extension of the well-known property of the output of the standard chase, by which all rows equal over X must have the same A-value. As shown in the following proposition, the m-table Σ^* allows to compute the sets $\mathsf{True}(\mathcal{T})$ and $\mathsf{Inc}(\mathcal{T})$.

Proposition 4 *Let T be a table over U, FD the associated set of functional dependencies, and Σ^* the output of Algorithm 1 run with T and FD as input. Then:*

1. $\mathsf{True}(\mathcal{T}) = \mathsf{LoCl}(\Sigma^*)$.
2. $\mathsf{Inc}(\mathcal{T}) = \{t \in \mathcal{T} \mid (\exists \sigma \in \Sigma^*)(\exists A \in sch(t))(\exists a, a' \in adom(A))(a \sqsubseteq t \wedge t \sqsubseteq \sigma \wedge (aa') \sqsubseteq \sigma)\}$.
3. $\mathsf{Cons}(\mathcal{T}) \cap \mathsf{True}(\mathcal{T}) = \{t \in \mathsf{True}(\mathcal{T}) \mid (\forall \sigma \in \Sigma^*)(t \sqsubseteq \sigma \Rightarrow \mathsf{tuples}(\sigma(sch(t))) = \{t\})\}$.

Proof. **1.** The result is an immediate consequence of Lemma 4(2).
2. The result follows from Lemma 4(2) and Definition 3.
3. The result follows from Definition 3 and the previous two items. □

In order to assess the complexity of Algorithm 1, for every $X \to A$ in FD and every tuple x in $\mathcal{T}(X)$, we let $N(x)$ be the number of different A-values a such that $I^*(xa) \neq \emptyset$ and a is inconsistent, and we denote by δ the maximal value of all $N(x)$ for all $X \to A$ in FD and all x in $\mathsf{True}(\mathcal{T})$; in other words $\delta = \max(\{N(x) \mid X \to A \in FD \wedge x \in \mathsf{True}(\mathcal{T})\})$.

Using this notation, the size of the m-tuples in Σ^* is bounded by $|U|.\delta$, and as Σ^* contains at most $|T|$ m-tuples, a run of the core of loop line 4 is in $\mathcal{O}(|T|^2.(|U|.\delta))$. Since the number of runs of the loop line 4 is bounded by the changes in the sets in Σ^*, this number of runs is in $\mathcal{O}(|T|.|FD|.\delta)$ (that is the maximum number of constants that can be added in a set of Σ^* times the size of Σ^*). Therefore, the computation of Σ^* is in $\mathcal{O}((|T|.|FD|.\delta).(|T|^2.(|U|.\delta))$, that is in $\mathcal{O}(|T|^3.|FD|.|U|.\delta^2)$ or even in $\mathcal{O}(|T|^3.\delta^2)$, since $|U|$ and $|FD|$ are independent from $|T|$.

We draw attention to the following important points regarding this result:

1. When the database is consistent, δ is equal to 1, thus yielding a complexity in $\mathcal{O}(|T|^3)$. This result can be shown independently from the above computations as follows: In the case of traditional chase the maximum of nulls in T being bounded by $|U|.|T|$, the number of iterations when running the algorithm is also bounded by $|U|.|T|$. Since the run of one iteration is in $|T|^2$, the overall complexity is in $\mathcal{O}(|U|.|T|^3)$, or in $\mathcal{O}(|T|^3)$, as $|U|$ is independent from $|T|$.

2. The above complexity study can be seen as a *data* complexity study as it relies on the size of T. Thus, this study should be further investigated in order to provide more accurate results regarding the estimation of the number of actual tests necessary to the computation of Σ^*. The results in [8] are likely to be useful for such a more thorough study of this complexity.

4.2 Computing Consistent Answers

In this sub-section, we show how the consistent answers to a query can be computed based on the m-table Σ^*. We stress that to state one of the results, it is assumed that Σ^* is *not redundant*,

that is for all σ_1 and σ_2 in Σ^*, neither $\sigma_1 \sqsubseteq \sigma_2$ nor $\sigma_2 \sqsubseteq \sigma_1$ holds. Notice that, *if* Σ^* *is not redundant then for all* σ_1 *and* σ_2 *in* Σ^*, $I^*(\sigma_1) \cap I^*(\sigma_2) = \emptyset$.

Indeed, if $I^*(\sigma_1) \cap I^*(\sigma_2) \neq \emptyset$, Lemma 4(2) implies that Σ^* contains σ such that $\sigma_i \sqsubseteq \sigma$ for $i = 1, 2$. As this entails that Σ^* contains redundancies, we obtain a contradiction. We also notice in this respect that redundancies are eliminated at each step of the computation of Σ^* in Algorithm 1 (see the statements lines 16 and 18). Thus, it turns out that if the table T contains no redundancy, so does Σ^*.

Moreover, to state our results, we introduce the following additional notation. Let T be a non-redundant table T over universe U, and Σ^* its associated non-redundant m-table, as computed by Algorithm 1. Given a query Q : `select` X `from` T `[where` Γ`]`, and a tuple x in $\mathcal{T}(X)$, we respectively denote by $\Sigma(x)$ and $\Sigma_Q(x)$, the sets defined by $\Sigma(x) = \{\sigma \in \Sigma^* \mid (x \sqsubseteq \sigma)\}$ and $\Sigma_Q(x) = \{\sigma \in \Sigma(x) \mid (\exists s \in Sat(\Gamma))(s \sqsubseteq \sigma)\}$.

Considering that $\Sigma_Q(x) = \Sigma(x)$ if Q involves no selection condition, the following proposition characterizes the tuples in $\mathsf{MSC\_ans}(Q)$, $\mathsf{SC\_ans}(Q)$, $\mathsf{C\_ans}(Q)$ and $\mathsf{WC\_ans}(Q)$ in terms of m-tuples as follows.

Proposition 5 *Let* T *be a non-redundant table over universe* U, FD *the set of associated functional dependencies and* Σ^* *the non-redundant m-table as computed by Algorithm 1. Given a query* Q : `select` X `from` T `[where` Γ`]`, *and a tuple* x *in* $\mathcal{T}(X)$*:*

1. x *is in* $\mathsf{MSC\_ans}(Q)$ *if and only if*
   (a) *for every* σ *in* $\Sigma(x)$, $\mathsf{tuples}(\sigma(X)) = \{x\}$,
   (b) *there exists* σ *in* $\Sigma_Q(x)$ *such that* $|\mathsf{tuples}(\sigma)| = 1$.
2. x *is in* $\mathsf{SC\_ans}(Q)$ *if and only if*
   (a) *for every* σ *in* $\Sigma(x)$, $\mathsf{tuples}(\sigma(X)) = \{x\}$,
   (b) *there exists* s *in* $Sat(\Gamma) \cap \mathsf{True}(\mathcal{T})$ *such that for every* σ *in* $\Sigma_Q(x)$, *if* s *is in* $\sigma(sch(\Gamma))$ *then* $\mathsf{tuples}(\sigma(sch(\Gamma)))) = \{s\}$.
3. x *is in* $\mathsf{C\_ans}(Q)$ *if and only if*
   (a) *for every* σ *in* $\Sigma(x)$, $\mathsf{tuples}(\sigma(X)) = \{x\}$,
   (b) *there exists* σ *in* $\Sigma_Q(x)$ *such that* $\mathsf{tuples}(\sigma(sch(\Gamma))) \subseteq Sat(\Gamma)$.
4. x *is in* $\mathsf{WC\_ans}(Q)$ *if and only if*
   (a) *for every* σ *in* $\Sigma(x)$, $\mathsf{tuples}(\sigma(X)) = \{x\}$,
   (b) $\Sigma_Q(x)$ *is not empty.*

Proof. See Appendix C. □

Example 8 We illustrate Proposition 5 in the context of Example 5 where $U = \{Emp, Sal, Dept\}$, $FD = \{Emp \rightarrow Sal\}$ and $T = \{es, es'd, e's\}$. It is easy to see that Algorithm 1 returns truth value *true* for $fail$ and $\Sigma^* = \{\sigma_1, \sigma_2\}$ where $\sigma_1 = (e)(ss')(d)$ and $\sigma_2 = (e')(s')$. We apply Proposition 5 to the queries Q_1, Q_2, Q_3, Q_4, Q_5 and Q_6 shown below:

Q_1 : `select` $Emp, Dept$ `from` T
Q_2 : `select` Emp, Sal `from` T
Q_3 : `select` Sal `from` T
Q_4 : `select` Emp `from` T `where` $Sal = s'$

Q_5 : `select` Emp `from` T `where` $Sal = s \vee Sal = s'$
Q_6 : `select` Emp `from` T `where` $Sal > 10K$

Since Q_1 involves no selection condition, and since σ_1 is the only m-tuple in Σ^* whose schema contains Emp and $Dept$, the tests concern only this m-tuple. As $\mathsf{tuples}(\sigma_1(Emp, Dept)) = \{ed\}$, we have $\mathsf{SC\_ans}(Q_1) = \mathsf{C\_ans}(Q_1) = \{ed\}$. Moreover, as $\Sigma_{Q_1}(ed) = \{ed\}$, $\mathsf{WC\_ans}(Q_1) = \{ed\}$ and $\mathsf{MSC\_ans}(Q_1) = \emptyset$, because $|\mathsf{tuples}(\sigma_1)| = 2$.

For Q_2, as $\mathsf{tuples}(\sigma_1(Emp, Sal)) = \{es, es'\}$ and $\mathsf{tuples}(\sigma_2(Emp, Sal)) = \{e's'\}$, it is immediate to see that $\mathsf{SC\_ans}(Q_2) = \mathsf{C\_ans}(Q_2) = \mathsf{WC\_ans}(Q_2) = \{e's'\}$. Moreover, as $e's'$ is a maximal consistent and true tuple, we also have $\mathsf{MSC\_ans}(Q_2) = \{e's'\}$. In the case of Q_3, it is easily seen that the four answers are empty because $\mathsf{tuples}(\sigma_1(Sal)) = \{s, s'\}$.

Considering Q_4, as $\mathsf{tuples}(\sigma_1(Emp)) = \{e\}$ and $\mathsf{tuples}(\sigma_2(Emp)) = \{e'\}$, and thus, these two values satisfy statement (a) in the four items in Proposition 5. As $Sat(\Gamma_4) = \{s'\}$, it is easy to see that $\Sigma_{Q_4}(e) = \{\sigma_1\}$ and $\Sigma_{Q_4}(e') = \{\sigma_2\}$. Thus, $\mathsf{WC\_ans}(Q_4) = \{e, e'\}$. Then, as $|\mathsf{tuples}(\sigma_1(Emp, Sal))| = 2$, e does not belong to $\mathsf{MSC\_ans}(Q_4)$ nor to $\mathsf{SC\_ans}(Q_4)$. Since $\mathsf{tuples}(\sigma_1(Sal)) = \{s, s'\}$, e is not in $\mathsf{C\_ans}(Q_4)$ either, because statement (3.b) is not satisfied (as s is not in $Sat(\Gamma_4)$). On the other hand is is easy to see that σ_2 satisfies statements (b), entailing that we have $\mathsf{MSC\_ans}(Q_4) = \mathsf{SC\_ans}(Q_4) = \mathsf{C\_ans}(Q_4) = \{e'\}$.

Concerning Q_5, the two candidate Emp-values are again e and e'. As $Sat(\Gamma_5) = \{s, s'\}$, $\Sigma_{Q_5}(e) = \{\sigma_1\}$ and $\Sigma_{Q_5}(e') = \{\sigma_2\}$, thus $\mathsf{WC\_ans}(Q_5) = \{e, e'\}$. As for Q_4, e does not belong to $\mathsf{MSC\_ans}(Q_5)$ and $\mathsf{SC\_ans}(Q_5)$ because we have $|\mathsf{tuples}(\sigma_1(Emp, Sal))| = 2$. However, σ_1 satisfies item (3.b) in Proposition 5 because $\mathsf{tuples}(\sigma_1(Sal))$ is indeed a sub-set of $Sat(\Gamma_5)$. Thus, e belongs to $\mathsf{C\_ans}(Q_5)$. The case of e' and σ_2 being similar to the previous query, we obtain $\mathsf{MSC\_ans}(Q_5) = \mathsf{SC\_ans}(Q_5) = \{e'\}$ and $\mathsf{C\_ans}(Q_5) = \{e, e'\}$.

Regarding Q_6, if $s = 15K$ and $s' = 20K$, $\mathsf{MSC\_ans}(Q_6) = \mathsf{SC\_ans}(Q_6) = \{e'\}$ and $\mathsf{C\_ans}(Q_6) = \mathsf{WC\_ans}(Q_6) = \{e, e'\}$ due to arguments similar to the previous case, and if $s = 5K$ and $s' = 20K$, we have $\mathsf{MSC\_ans}(Q_6) = \mathsf{SC\_ans}(Q_6) = \mathsf{C\_ans}(Q_6) = \{e'\}$ and $\mathsf{WC\_ans}(Q_6) = \{e, e'\}$, due to arguments similar to the case of Q_4 above. □

Based on Proposition 5, it can be seen that, given a query Q, Algorithm 2 computes the consistent answer to Q. Indeed, the test line 4 discards all m-tuples of Σ^* whose schema does not contain X, because such m-tuples have no chance to provide an X-value. Then, the items in Proposition 5 are checked as follows:

- The test line 5 allows to store in the set $ToRemove\_X$ all X-values of m-tuples σ such that $\mathsf{tuples}(\sigma(X))$ contains more than one tuple. All these tuples being inconsistent, are then removed from the set of retrieved X-values in $\mathsf{MSC\_ans}(Q)$ (line 24), in $\mathsf{SC\_ans}(Q)$ (line 26), in $\mathsf{C\_ans}(Q)$ (line 27) and in $\mathsf{WC\_ans}(Q)$ (line 28). Therefore, the statements (1.a), (2.a), (3.a) and (4.a) in Proposition 5 are satisfied by the returned sets. In what follows, we consider that the test line 5 fails, that is that $\mathsf{tuples}(\sigma(X)) = \{x\}$.

- Statement (4.b) is concerned when the tests lines 9 and 10 succeed. Indeed, in this case, there exists a tuple s in $\mathsf{tuples}(\sigma(sch(\Gamma)))$ such that s is in $Sat(\Gamma)$, meaning that σ is in $\Sigma_Q(x)$. Therefore, in this case, x is inserted in $\mathsf{WC\_ans}(Q)$ (line 11).

- Regarding the statement (1.b) in Proposition 5, on line 9 it is checked that the schema of Γ is a sub-set of that of σ. If yes, when the tests lines 10 and 12 succeed, σ is in fact a tuple t satisfying Γ. As moreover, Σ^* is not redundant, t occurs nowhere else in Σ^*, meaning by Proposition 4(3) that t is in $\mathsf{Cons}(\mathcal{T})$. Hence, x is inserted in $\mathsf{MSC\_ans}(Q)$ (line 13). Notice that in this case, the inclusions stated in Proposition 3 justify that x be also inserted in $\mathsf{SC\_ans}(Q)$ (line 14) and in $\mathsf{C\_ans}(Q)$ (line 15).

Algorithm 2 Consistent answers

Input: A query Q : `select` X `from` T `[where` Γ`]` and Σ^*
Output: The sets $\mathsf{MSC\_ans}(Q)$, $\mathsf{SC\_ans}(Q)$, $\mathsf{C\_ans}(Q)$ and $\mathsf{WC\_ans}(Q)$

1: $\mathsf{MSC\_ans}(Q) := \emptyset$; $\mathsf{SC\_ans}(Q) := \emptyset$; $\mathsf{C\_ans}(Q) := \emptyset$; $\mathsf{WC\_ans}(Q) := \emptyset$
2: $ToRemove\_X := \emptyset$; $Candidate\_SC := \emptyset$; $ToRemove\_SC := \emptyset$
3: **for all** σ in Σ^* **do**
4: **if** $X \subseteq sch(\sigma)$ **then**
5: **if** $|tuples(\sigma(X))| > 1$ **then**
6: $ToRemove\_X := ToRemove\_X \cup \mathsf{tuples}(\sigma(X))$
7: **else**
8: Let x denote the unique tuple in $\sigma(X)$
9: **if** $sch(\Gamma) \subseteq sch(\sigma)$ **then**
10: **if** $\mathsf{tuples}(\sigma(sch(\Gamma))) \cap Sat(\Gamma) \neq \emptyset$ **then**
11: $\mathsf{WC\_ans}(Q) := \mathsf{WC\_ans}(Q) \cup \{x\}$
12: **if** $|\mathsf{tuples}(\sigma)| = 1$ **then**
13: $\mathsf{MSC\_ans}(Q) := \mathsf{MSC\_ans}(Q) \cup \{x\}$
14: $\mathsf{SC\_ans}(Q) := \mathsf{SC\_ans}(Q) \cup \{x\}$
15: $\mathsf{C\_ans}(Q) := \mathsf{C\_ans}(Q) \cup \{x\}$
16: **else**
17: **if** $|\mathsf{tuples}(\sigma(sch(\Gamma)))| = 1$ **then**
18: $Candidate\_SC := Candidate\_SC \cup \mathsf{tuples}(\sigma(sch(Q)))$
19: $\mathsf{C\_ans}(Q) := \mathsf{C\_ans}(Q) \cup \{x\}$
20: **else**
21: $ToRemove\_SC := ToRemove\_SC \cup \mathsf{tuples}(\sigma(sch(Q)))$
22: **if** $\mathsf{tuples}(\sigma(sch(\Gamma))) \subseteq Sat(\Gamma)$ **then**
23: $\mathsf{C\_ans}(Q) := \mathsf{C\_ans}(Q) \cup \{x\}$
24: $\mathsf{MSC\_ans}(Q) := \mathsf{MSC\_ans}(Q) \setminus ToRemove\_X$
25: $\mathsf{SC\_ans}(Q) := \mathsf{SC\_ans}(Q) \cup \{x \mid (\exists q \in Candidate\_SC \setminus ToRemove\_SC)(q.X = x)\}$
26: $\mathsf{SC\_ans}(Q) := \mathsf{SC\_ans}(Q) \setminus ToRemove\_X$
27: $\mathsf{C\_ans}(Q) := \mathsf{C\_ans}(Q) \setminus ToRemove\_X$
28: $\mathsf{WC\_ans}(Q) := \mathsf{WC\_ans}(Q) \setminus ToRemove\_X$
29: **return** $\mathsf{MSC\_ans}(Q)$, $\mathsf{SC\_ans}(Q)$, $\mathsf{C\_ans}(Q)$, $\mathsf{WC\_ans}(Q)$

- As for statement (2.b), assuming that the test line 9 succeeds, the case when the test line 12 succeeds is as above. Now, if this test fails, then, if the test line 17 succeeds, then $\sigma(sch(Q))$ might be a consistent true tuple t satisfying Γ, unless t occurs in some other $\mathsf{tuples}(\sigma'(sch(Q)))$ such that $|\mathsf{tuples}(\sigma'(sch(Q)))| > 1$, meaning that t is inconsistent. This is why t is inserted in the set $Candidate\_SC$ when the test line 17 succeeds, and when this test fails, the tuples in the set $\mathsf{tuples}(\sigma(sch(Q)))$ are stored in $ToRemove\_SC$. Proposition 4(3) then ensures that the tuples in $Candidate\_SC \setminus ToRemove\_SC$ are indeed consistent true tuples satisfying Γ (other than maximal ones earlier detected). The statement line 25 adds the corresponding X-values in the set $\mathsf{SC\_ans}(Q)$.

- Considering now statement (3.b) when the test line 9 succeeds, we first notice that, thanks to Proposition 3, the insertions in $\mathsf{C\_ans}(Q)$ in statements lines 15 and 19 are correct. Moreover, when the test line 22 succeeds, the condition in the statement (3.b) is satisfied, and so, x is inserted in $\mathsf{C\_ans}(Q)$ as stated line 23.

We illustrate Algorithm 2 using the queries Q_3 and Q_4 of Example 8, respectively defined by Q_3 : `select` Sal `from` T and Q_5 : `select` Emp `from` T `where` $Sal = s \vee Sal = s'$.

Regarding Q_3, when considering σ_1, $ToRemove\_X$ is set to $\{s, s'\}$ because the test line 5 succeeds. Then, with σ_2, the test line 5 fails and the test line 10 succeeds, entailing that s' is inserted in $\mathsf{WC\_ans}(Q_3)$. Then, as the test line 12 succeeds, s' is inserted in $\mathsf{MSC\_ans}(Q)$ (line 13), $\mathsf{SC\_ans}(Q)$ (line 14) and in $\mathsf{C\_ans}(Q)$ (line 15). When processing the statements lines 24, 26 and 27, s' is removed, thus producing empty answers, as expected.

As for Q_5, when processing σ_1, the test line 5 fails for σ_1 and σ_2 because $|\mathsf{tuples}(\sigma_i(Emp))| = 1$ $(i = 1, 2)$. Thus $ToRemove\_X$ remains empty in this case. When processing σ_1, the tests line 12 and line 17 fail (because $|\mathsf{tuples}(\sigma_1)| = 2$ and $|\mathsf{tuples}(\sigma_1(Sal))| = 2$) and the test line 22 succeeds (because $\mathsf{tuples}(\sigma_1(Sal)) = Sat(\Gamma_5) = \{s, s'\}$). Thus, $ToRemove\_SC$ is set to $\{es, es'\}$ line 21 and $\mathsf{C\_ans}(Q)$ is set to $\{e\}$ line 23. Then, when processing σ_2, all tests succeed (because $|\mathsf{tuples}(\sigma_2)| = |\mathsf{tuples}(\sigma_2(Sal))| = 1$ and $\mathsf{tuples}(\sigma_2(Sal)) = \{s'\}$). Thus, e' is inserted in $\mathsf{WC\_ans}(Q_5)$, $\mathsf{MSC\_ans}(Q_5)$, $\mathsf{SC\_ans}(Q_5)$ and $\mathsf{C\_ans}(Q_5)$. The loop line 3 returns $ToRemove\_X = \emptyset$, $Candidate\_SC = \emptyset$, $ToRemove\_SC = \emptyset$, along with $\mathsf{WC\_ans}(Q_5) = \mathsf{MSC\_ans}(Q_5) = \mathsf{SC\_ans}(Q_5) = \mathsf{C\_ans}(Q_5) = \{e, e'\}$, which is not changed by the statements lines 24-28. We thus obtain all four consistent answers equal to $\{e, e'\}$, as expected.

In the next example, we further illustrate the role of the sets $Candidate\_SC$ and $ToRemove\_SC$ in Algorithm 2.

Example 9 Let $U = \{A, B, C, D\}$ over which the functional dependencies $C \rightarrow B$ and $C \rightarrow D$ are assumed. For $T = \{abcd, abcd', abc', ab'c'\}$, Algorithm 1 returns $\Sigma^* = \{(a)(b)(c)(dd'), (a)(bb')(c')\}$, and the processing of $\mathsf{SC\_ans}(Q)$ for Q : `select` A `from` T `where` $B = b$, is as follows:

- For $\sigma = (a)(b)(c)(dd')$, the tests line 4 and line 12 fail (as $|\mathsf{tuples}(\sigma(A))| = 1$ and $|\mathsf{tuples}(\sigma)| = 2$), whereas the test line 10 succeeds (because $Sat(\Gamma) = \{b\}$). Thus $ToRemove\_X$ remains empty and as the test line 17 succeeds (because $\mathsf{tuples}(\sigma(B)) = \{b\}$), ab is inserted in $Candidate\_SC$ on line 18, and $\mathsf{SC\_ans}(Q)$ remains empty.

- For $\sigma = (a)(bb')(c')$, as the test line 4 fails, $ToRemove\_X$ remains empty, and as the tests line 12 and line 17 also fail, ab and ab' are inserted in $ToRemove\_SC$ due to the statement line 21.

Thus, the loop line 3 generates $ToRemove\_X = \emptyset$, $Candidate\_SC = \{ab\}$ and $ToRemove\_SC = \{ab, ab'\}$. Therefore, when processing the statement line 25, $\mathsf{SC\_ans}(Q)$ remains empty (because $Candidate\_SC \setminus ToRemove\_SC = \emptyset$), as expected based on Definition 6(2), because ab is inconsistent.

Considering now $T_1 = \{abcd, abcd'\}$, Algorithm 1 returns $\Sigma_1^* = \{(a)(b)(c)(dd')\}$ and the computations are as in the first item above, implying that the loop line 3 generates $ToRemove\_X = \emptyset$, $Candidate\_SC = \{ab\}$, $ToRemove\_SC = \emptyset$ and $\mathsf{SC\_ans}(Q) = \emptyset$. When processing the statement line 25, $\mathsf{SC\_ans}(Q)$ is set to $\{a\}$ (because $Candidate\_SC \setminus ToRemove\_SC = \{ab\}$), as expected based on Definition 6(2), because ab is a consistent true tuple satisfying Γ. □

To assess the complexity of Algorithm 2, we first notice that this algorithm is clearly linear in the size of Σ^*. Moreover, using the same notation as when assessing the complexity of Algorithm 1, for σ in Σ^*, the size of $\mathsf{tuples}(\sigma)$ is in $\mathcal{O}(|FD|.\delta)$, that is in $\mathcal{O}(\delta)$, because $|FD|$ is independent from T. Hence, the complexity of computing the consistent answer to a query in our approach is in $\mathcal{O}(|\Sigma^*|.\delta)$. Since it has been shown that $|\Sigma^*| \leq |T|$, we state that the sizes of Σ^* and of T are of the same order. Therefore, the complexity of computing all four consistent answers to a query in our approach is in $\mathcal{O}(|T|.\delta)$, that is *linear in the size of the table* T.

It is however important to stress that Algorithm 2 requires significant extra storage due to the sets $ToRemove\_X$, $Candidate\_SC$ and $ToRemove\_SC$. It should be noticed that the storage of $ToRemove\_X$ and $ToRemove\_SC$ can be avoided by implementing a second scan of Σ^* whose role would be to detect the X-values to be removed (when $|\mathsf{tuples}(\sigma(X))| > 1$) and the tuples in $Candidate\_SC$ to be removed (when $|\mathsf{tuples}(\sigma(sch(Q)))| > 1$). As usual, this option saves storage at the cost of further processing, namely an additional scan of Σ^*.

5 Related Work

The problem of consistent query answering has attracted considerable attention in the last two decades [1] and continues to be an active area of research today [4]. The approach by repairs is one that has attracted most attention in recent years and thus has been the subject of many papers, whose overview lies outside the scope of this paper. We refer to [5, 20] for detailed surveys covering the topic. The fact that "deciding whether a candidate answer is a consistent answer is most commonly intractable in data complexity" [7] generated several research efforts recently focusing on approximate consistent answers [6, 11], and on issues related to the counting of repairs [7, 15].

Our approach to consistent query answering is radically different as it does not rely on repairs but rather on set theoretic semantics for tuples and functional dependencies. Consequently, to compare our approach with repair-based approaches, we focus only on the basic differences between the two approaches. Our goal is to show that although the problem tackled is the same, our approach is hardly comparable with those from the literature (see Section 5.1). We then introduce in Section 5.2 the notion of *strong repairs* according to which some comparison is possible, and in Section 5.3 we compare the approach presented in this paper and the approach in our earlier work [12].

5.1 Comparison with Repair-Based Approaches

We first point out that repair-based approaches assume a table T with no nulls. Therefore to compare our approach to repair-based approaches we need to restrict our attention to tables with functional dependencies but *without* nulls.

In this context, we identify and discuss three fundamental differences between our approach and the approach by repairs, namely : (D1) the two approaches use different universes of discourse, (D2) the two approaches use different assumptions when evaluating queries, and (D3) the two approaches use functional dependencies for different purposes.

(D1) ***different universes of discourse.***
We recall from Section 2 that the universe of discourse of a table T is the space in which queries are evaluated. The universe of discourse in our approach, denoted by $\mathcal{T}$, is the set of all tuples one can define using constants appearing in T. On the other hand, the universe of discourse in the repairs approach, that we shall denote by T_r, is the set of all tuples of T together with all their sub-tuples. For example, if $T = \{ab, a'b'\}$ then $\mathcal{T} = \{ab, a'b', a'b, ab', a, b, a', b'\}$ whereas $T_r = \{ab, a'b', a, b, a', b'\}$.

(D2) ***different assumptions when evaluating queries.***
The repairs approach uses the Closed World Assumption [16] that is:

> the only true tuples are those in T_r (implying that tuples not in T_r are false)

whereas our approach uses the Open World Assumption that is:

> the only true tuples are those in T together with all tuples derived using the functional dependencies (implying that all other tuples in $\mathcal{T}$ are false).

It follows that if a tuple t of T_r is true then t is true in $\mathcal{T}$, that is $True(T_r) \subseteq \mathsf{True}(\mathcal{T})$, while the opposite does not hold.

For example, consider the table $T = \{abc, ab'c'\}$ with functional dependency $A \rightarrow B$. In this case, ac is in $\mathcal{T}$ and in $\mathsf{True}(\mathcal{T})$, and ac is in $True(T_r)$ as well. On the other hand, since $\Sigma^* = \{(a)(bb')(c), (a)(bb')(c')\}$, the tuple abc' is in $\mathcal{T}$ but *not* in T_r. Thus abc' is not in $True(T_r)$, whereas abc' is in $\mathsf{True}(\mathcal{T})$.

Regarding inconsistent tuples, the comparison not as easy since, to the best of our knowledge, the notion of inconsistent tuple is not explicitly defined in repair-based approaches. However, it is clear that a table T is inconsistent if and only if there exists a functional dependency $X \rightarrow A$ such that the two tuples xa and xa' in $\mathcal{T}(XA)$ occur in T. The pair (xa, xa') is usually called a *conflicting pair*, and we define inconsistent tuples in this context as follows:

t is inconsistent if there exists a conflicting pair (xa, xa') such that t is a super-tuple of xa.

Denoting the set of these tuples by $Conf\_Inc(T_r)$, we argue that $Conf\_Inc(T_r) \subseteq \mathsf{Inc}(\mathcal{T})$ whereas the inverse inclusion does not always hold. Indeed, based on Definition 3, it is easy to see that, for every conflicting pair (xa, xa'), the tuples xa, xa' and all their true super-tuples are in $\mathsf{Inc}(\mathcal{T})$, thus showing the inclusion $Conf\_Inc(T_r) \subseteq \mathsf{Inc}(\mathcal{T})$. The previous example with $T = \{abc, ab'c'\}$ and $A \rightarrow B$, shows that the inverse inclusion does not always hold, because b is clearly not in $Conf\_Inc(T_r)$, whereas Definition 3 implies that b is in $\mathsf{Inc}(\mathcal{T})$.

(D3) ***different purposes in the use of functional dependencies.***
The repair-based approaches use functional dependencies as a means for checking the consistency of T (i.e., as a means of inference of inconsistent tuples). In our approach, we also use the functional dependencies as a means of inference of inconsistent tuples. However, additionally, we use the functional dependencies as a means of inference of *true* tuples. For example, if $T = \{abc, ab'c'\}$ and $FD = \{A \rightarrow B\}$, our approach allows us to infer that abc is true and inconsistent, as repair-based approaches do. However, additionally, our approach allows to infer that the tuple $ab'c$ is also true and inconsistent, which repair-based approaches fail to do since $ab'c$ is not even in T_r.

Now that we have stated the differences in the ways the two approaches operate, consider a query Q : `select` X `from` T `where` Γ, and recall that in repair-based approaches, the consistent answer to Q is defined as the set of true tuples x in T_r such that, every repair contains a tuple satisfying the condition Γ and whose restriction over X is x. Denoting this set by $Rep\_ans(Q)$, we recall that, by Definition 5, all consistent answers to Q in our approach are sets of true and consistent tuples in $\mathcal{T}(X)$. It is thus not surprising that the two approaches to consistent query answering are not comparable.

Summarising our discussion so far we cannot claim that either of the two approaches is better or preferable than the other. They are simply different as they have different universes of discourse, they operate under different assumptions and they use the functional dependencies in different ways.

What we can reasonably claim in favor of our approach is that: (a) it provides a formal framework based on which different kinds of consistent answers can be defined, (b) it offers a polynomial algorithm for evaluating the consistent answer to a query, whether conjunctive or disjunctive, and (c) by allowing nulls in the table, our approach broadens the angle of its application to other important areas of research as explained in the concluding Section 6.

5.2 Strong Repairs

As highlighted in the previous section, repair approaches do not compare to our approach. Nevertheless, in this section, we propose a modified version of the notion of repair according to which some comparison is possible.

First we recall that given a table T without nulls and a set FD of functional dependencies over T, a repair of T is defined to be a maximal consistent sub-set of T, where maximality is understood with respect to set-theoretic inclusion. For example given $T = \{eda, eda', e'd'a\}$ and $FD = \{Emp \rightarrow Dept, Dept \rightarrow Addr\}$ as in our introductory example, we obtain two repairs

$R_1 = \{eda, e'd'a\}$ and $R_2 = \{eda', e'd'a\}$ and so the consistent answer to Q : select Emp from T where $Addr = a$ is $\{e, e'\} \cap \{e'\}$, that is $\{e'\}$. We refer to [3] for a theoretical account on how to compute such consistent query answers when T is a table *without* nulls.

However, when the table T contains nulls then the question is: which table should be repaired? In other words: what is the definition of a repair in this case? We illustrate this point in the following example.

Example 10 Referring to Examples 3 and 4, let $T = \{ab, bc, ac'\}$ and $FD = \{A \to C, B \to C\}$. Considering that T shows no explicit violation of functional dependencies, repairing T is trivial, and thus, the consistent answer to Q : select A, C from T would be $\{ac'\}$. However, according to approaches related to universal relation [17, 18], the table T has to be chased before computing the answer to Q. In this case, the traditional chase algorithm fails because of $A \to C$ and the presence of abc and ac' in the chased table. Consequently, the query cannot be answered.

We cope with this difficulty in our approach by computing $\Sigma^* = \{(a)(b)(cc')\}$, and thus, showing the inconsistency. In this context, it is possible to define $R_1 = \{abc\}$ and $R_2 = \{abc'\}$ as the two repairs of T and then, to state that the consistent answer to Q is empty. The remainder of the section elaborates on this idea and compares it to our approach explained earlier. □

As suggested by Example 10, in the presence of nulls, the table to be repaired is not T itself, but the table denoted by T^* that contains all tuples in $\mathsf{tuples}(\sigma)$ for every σ in Σ^*. Formally, $T^* = \bigcup_{\sigma \in \Sigma^*} \mathsf{tuples}(\sigma)$.

Going one step further, as mentioned in the previous section, consistent answers to queries in our approach and in the repair-based approach using T^* are *not comparable*. To formalize this statement, we denote by $\mathsf{R\_cons}(T)$ the set of all tuples occurring in every repair of T^*, and we give examples showing that $\mathsf{R\_cons}(T)$ and $\mathsf{Cons}(\mathcal{T}) \cap \mathsf{True}(\mathcal{T})$ *are not comparable with respect to set-theoretic inclusion.*

First, for $T = \{ab, ab', bc\}$ over $U = \{A, B, C\}$ and $FD = \{A \to B\}$, we have $\Sigma^* = \{(a)(bb'), (b)(c)\}$. We thus have two repairs R_1 and R_2 where $R_1 = \{ab, bc\}$ and $R_2 = \{ab', bc\}$. It follows that b is in $\mathsf{R\_cons}(T)$ whereas it is obvious that b is in $\mathsf{Inc}(\mathcal{T})$, thus not in $\mathsf{Cons}(\mathcal{T}) \cap \mathsf{True}(\mathcal{T})$. Therefore, $\mathsf{R\_cons}(T) \subseteq \mathsf{Cons}(\mathcal{T}) \cap \mathsf{True}(\mathcal{T})$ does not hold in general.

Now, let $U = \{A, B, C, D\}$, $FD = \{A \to C, B \to C\}$ and $T = \{acd,\ ac'd,\ bcd',\ bc'd',\ abc,\ abc'\}$, in which case $\Sigma^* = \{(a)(c, c')(d),\ (b)(c, c')(d'),\ (a)(b)(c, c')\}$. Thus, we obtain the four repairs $R_1 = \{acd,\ bcd',\ abc\}$, $R_2 = \{ac'd,\ bc'd',\ abc'\}$, $R_3 = \{acd, bc'd'\}$ and $R_4 = \{ac'd, bcd'\}$. In this case, ab is clearly *not* in $\mathsf{R\_cons}(T)$, whereas, by Proposition 4(3), ab belongs to $\mathsf{Cons}(\mathcal{T}) \cap \mathsf{True}(\mathcal{T})$. It therefore turns out that $\mathsf{Cons}(\mathcal{T}) \cap \mathsf{True}(\mathcal{T}) \subseteq \mathsf{R\_cons}(T)$ does not hold in general.

The last counter-example suggests that, in repair-based approaches, an m-tuple in Σ^* might have no 'representatives' in some repairs (e.g., no tuple of $\mathsf{tuples}((a)(b)(cc'))$ is present in R_3 or in R_4). This means that the repairs R_3 or R_4 do *not* convey the information that, in any repair abc or abc' must be true, as stated by Σ^*. In order to account for this remark, we restrict repairs so as, for every σ in Σ^*, every repair contains at least one tuple of $\mathsf{tuples}(\sigma)$.

Definition 8 *Given a table T over U, FD the associated set of functional dependencies, and Σ^* the corresponding m-table, a table R over U is a* strong repair *of T if R is a repair of T^*, and if for every σ in Σ^*, $R \cap \mathsf{tuples}(\sigma)$ is nonempty. The set of all strong repairs of T is denoted by $\mathsf{S\_rep}(T)$.* □

In the case of the last counter-example above, R_1 and R_2 are in $\mathsf{S\_rep}(T)$ whereas R_3 and R_4 are not. On the other hand, since in this case $\mathsf{Cons}(\mathcal{T}) \cap \mathsf{True}(\mathcal{T}) = \{ab,\ ad,\ bd',\ a,\ b,\ d,\ d'\}$, it should be observed that all tuples in $\mathsf{Cons}(\mathcal{T}) \cap \mathsf{True}(\mathcal{T})$ occur in R_1 and in R_2. To show that this property always holds, we introduce the following terminology:

- The *strong-repair-based consistent answer* to Q, denoted by $\mathsf{SRep\_ans}(Q)$, is defined as the intersection of all answers to Q in every strong repair of T. Formally, if $\mathsf{Ans}(Q_{[R]})$ denotes the answer to Q in R, $\mathsf{SRep\_ans}(Q) = \bigcap_{R \in \mathsf{S\_rep}(T)} \mathsf{Ans}(Q_{[R]})$.

- The set of all *strong-repair-based consistent tuples*, denoted by $\mathsf{S\_cons}(T)$, is defined as the set of all tuples occurring in every strong repair.

Before showing the main result of this sub-section to compare $\mathsf{Cons}(\mathcal{T}) \cap \mathsf{True}(\mathcal{T})$ and $\mathsf{S\_cons}(T)$, we show the following basic lemma.

Lemma 5 *Given a table T over U and the associated set of functional dependencies FD, for every t in T^*, there exists R in $\mathsf{S\_rep}(T)$ such that t is in R.*

Proof. The repair R considered here is built up as follows, based on the m-tuples of Σ^*. First let σ_t be an m-tuple of Σ^* such that $t \in \mathsf{tuples}(\sigma_t)$. For every A in $sch(t)$ and every m-tuple σ such that A is in $sch(\sigma)$ and $t.A$ in $\mathsf{tuples}(\sigma(A))$, mark $t.A$ in $\sigma(A)$. Notice that at this stage, all components of σ_t have a marked value. In this case, the tuple consisting of these marked values is inserted in R and the process is iterated until there remains no unmarked components in any m-tuple of Σ^*. That is, at each step, a 'non-fully marked' m-tuple in Σ^* is chosen and processed similarly to t. When the process terminates, every component of every σ has one value marked, which defines a tuple of $\mathsf{tuples}(\sigma)$ to be inserted in R.

We now prove that R satisfies all functional dependencies in FD. If not, let t and t' in R and $X \to A$ in FD such that $t.X = t'X$ and $t.A \neq t'.A$. It follows that there exist σ and σ' in Σ^* such that $t \in \mathsf{tuples}(\sigma)$ and $t' \in \mathsf{tuples}(\sigma')$, $t.X \in \mathsf{tuples}(\sigma(X)) \cap \mathsf{tuples}(\sigma'(X))$. Hence, by construction of Σ^*, we have $\sigma(A) = \sigma'(A)$, and thus, by construction of R, it must be that $t.A = t'.A$, which is a contradiction. Thus R satisfies FD, and to end the proof, we notice that, if R is not maximal with respect to set-theoretic inclusion, tuples from T^* respecting the functional dependencies are added, thus leading to a strong repair containing t. The proof is therefore complete. □

As a case of non-maximality of R in the proof above, consider $\Sigma^* = \{(aa')(bb')\}$, $FD = \{A \to B, B \to A\}$ and $t = ab$. Then a and b are marked and the process stops producing $\{ab\}$ which is not maximal since $\{ab, a'b'\}$ is a repair of T^*.

It is important to notice that, as a consequence of Lemma 5, the set $\mathsf{S\_rep}(T)$ can not be empty, when T is nonempty. Moreover, given a selection condition Γ, Lemma 5 implies that if Σ^* contains an m-tuple σ such that every t in $\mathsf{tuples}(\sigma)$ satisfies Γ, then every strong repair of T contains a tuple satisfying Γ.

The following proposition shows that consistent true tuples are consistent with respect to strong repairs, that is every consistent true tuple occurs in every strong repair.

Proposition 6 *Let T be a table over U and FD its associated set functional dependencies. Then $\mathsf{Cons}(\mathcal{T}) \cap \mathsf{True}(\mathcal{T}) \subseteq \mathsf{S\_cons}(T)$ holds.*

Moreover, every t in $\mathsf{S\_cons}(T)$ but not in $\mathsf{Cons}(\mathcal{T}) \cap \mathsf{True}(\mathcal{T})$ is in $\mathsf{Inc}(\mathcal{T})$.

Proof. Let t be in $\mathsf{Cons}(\mathcal{T}) \cap \mathsf{True}(\mathcal{T})$. By Proposition 4(3), for every σ in Σ^* such that $\mathsf{tuples}(\sigma(sch(t)))$ contains t, we have $\mathsf{tuples}(\sigma(sch(t))) = \{t\}$. Thus, for every q in $\mathsf{tuples}(\sigma)$, $q.sch(t) = t$, and so, as every R in $\mathsf{S\_rep}(T)$ contains at least one tuple of $\mathsf{tuples}(\sigma)$, t occurs in R, meaning that t belongs to $\mathsf{S\_cons}(T)$.

Assume now that t is in $\mathsf{S\_cons}(T)$ but not in $\mathsf{Cons}(\mathcal{T}) \cap \mathsf{True}(\mathcal{T})$. Since t is in $\mathsf{S\_cons}(T)$, then there exists σ in Σ^* such that t is in $\mathsf{tuples}(\sigma(sch(t)))$. Thus, by Proposition 4(1), t is in $\mathsf{True}(\mathcal{T})$. Since on the other hand, t is not in $\mathsf{Cons}(\mathcal{T}) \cap \mathsf{True}(\mathcal{T})$, t is in $\mathsf{Inc}(\mathcal{T})$, and the proof is complete. □

Corollary 1 *Let T be a table over U and FD its associated set of functional dependencies. For every query Q,* $\mathsf{C\_ans}(Q) \subseteq \mathsf{SRep\_ans}(Q)$ *holds.*

Proof. Given x in $\mathsf{C\_ans}(Q)$, by Proposition 5, there exists σ in Σ^* such that $\mathsf{tuples}(\sigma(X)) = \{x\}$ and $\mathsf{tuples}(\sigma(sch(\Gamma))) \subseteq Sat(\Gamma)$. As every strong repair R of T^* contains a tuple in $\mathsf{tuples}(\sigma)$, every strong repair R of T^* contains a tuple q such that $q.X = x$ and $q.sch(\Gamma) \in Sat(\Gamma)$. Therefore, for every strong repair R of T^*, x is in the answer to Q in R, meaning that x is in $\mathsf{SRep\_ans}(Q)$, and the proof is complete. □

As a consequence of Corollary 1 and Proposition 3, the following inclusions hold for every Q:

$\mathsf{MSC\_ans}(Q) \subseteq \mathsf{SC\_ans}(Q) \subseteq \mathsf{C\_ans}(Q) \subseteq \mathsf{SRep\_ans}(Q)$.

5.3 Comparison with the Approach in [12]

The approach presented in this paper and the approach in [12], both rely on partition semantics [17]. However, the two approaches have important differences. First, although the limit interpretation I^* as defined in Section 2 is the same as the interpretation μ^* in [12], we have proved here the uniqueness of I^* which is not done for μ^*.

Second, the notions of true/false tuples and of inconsistent/consistent tuples differ between the two approaches. More precisely, since in [12], truth values are defined inspired by the Four-valued logic of [2], it is not possible to have tuples that are, for instance, true *and* consistent, as in the present work.

Third, the universe of discourse in [12], is potentially infinite as it consists of all tuples that can be built up using constants from the entire attribute domains; and this is an issue when it comes to computing the set of all inconsistent tuples. In contrast, the universe of discourse in the present work is finite as it consists of all tuples that can be built up using constants *only* from the active attribute domains.

Fourth, the notion of inconsistent tuple is not defined in the same way in the two approaches. Indeed, in [12], a tuple is inconsistent if its interpretation is a sub-set of the interpretations of two constants from the same domain. For example, consider the (true) tuples xa and xa' in the presence of $X \rightarrow A$. These tuples are inconsistent in the present approach (see Definition 3), as well as in the approach of [12]. However, the tuple x is also an inconsistent tuple according to [12] (because its interpretation is a sub-set of those of a and a'), but is *not* inconsistent in the present approach (because A is not in $sch(x)$). As a consequence, the consistent answer to a query in [12] differs from that in the present work.

Finally, the issue of consistent query answering has not been addressed in full details in [12]: it is restricted to the case where the table T contains no nulls. We also note in this respect that, based on the concept of m-table as introduced in Section 4, the present approach allows to consider also *disjunctive* queries, an issue not addressed in [12], nor in repair-based approaches, such as [5, 19].

6 Concluding Remarks and Future Work

In this paper, we have presented a novel approach to consistent query answering in tables with nulls and functional dependencies. Given such a table T, we defined $\mathcal{T}$, the universe of discourse of T, and using set-theoretic semantics for tuples and functional dependencies we partitioned $\mathcal{T}$ in two orthogonal ways: into true and false tuples, on the one hand and into consistent and inconsistent tuples on the other hand. Queries are addressed to T and evaluated in $\mathcal{T}$ in order to obtain consistent answers. The consistent answer to a query Q : `select` X `from` T `where` *Condition* over T was defined to be the set of tuples x in $\mathcal{T}$ such that: (a) x is a true and

consistent tuple with schema X and (b) there exists a true super-tuple t of x in $\mathcal{T}$ satisfying the condition. We have seen that, depending on the 'status' of the super-tuple t in $\mathcal{T}$, there are four different types of consistent answer to Q and we have presented polynomial time algorithms for computing these answers. We have also seen that our approach is not comparable to the approaches based on table repairs or the approach of [12], as they have different universes of discourse and operate under different assumptions.

By allowing the presence of nulls in the table, our approach opens the way for a number of important applications. We envisage three such applications of our approach. The first is related to universal relation interfaces [10, 17, 18]. Given a relational database with n tables $T_1, T_2, \ldots, T_n$, over schemes $S_1, S_2, \ldots, S_n$ and sets of functional dependencies $FD_1, FD_2, \ldots, FD_n$, respectively, the universal relation is defined to be a table T defined as follows:

- The schema of T is $S = S_1 \cup S_2 \cup \ldots \cup S_n$ and the associated set of functional dependencies is $FD = FD_1 \cup FD_2 \cup \ldots \cup FD_n$.
- For each tuple t in T_i, t is placed in a new line of T with a new tuple identifier, for all $i = 1, 2, \ldots, n$.

For example, if $n = 2$, $T_1 = \{ab\}$ and $T_2 = \{bc\}$ then the universal relation operates from a table T with schema ABC and current instance $T = \{ab, bc\}$ with two nulls, one under C and one under A.

A prerequisite for defining the table T is "name consolidation" that is (a) if an attribute name appears in two different database tables then it has the same meaning in the two tables and (b) if two different attribute names in the database have the same meaning then one is renamed to the other (using the renaming operation of the relational model).

Once the table T is put in place querying the interface is straightforward: the system computes a consistent answer using our polynomial Algorithm 2 and returns the result. However, updating through such an interface poses several hard problems that we are currently investigating.

The second application that we envisage is query result explanation. As already mentioned, the m-table Σ^* offers a way to 'display' inconsistencies. We therefore plan to investigate how m-tuples in Σ^* can be used to 'explain' to the user the presence or the absence of a given tuple in a consistent answer. This issue seems particularly relevant in the context of data integration and data exchange, where it is crucial to 'explain' consistent answers in reference to the sources the data come from [4].

The third application that we envisage is related to data quality. Based on the two orthogonal ways of characterizing tuples, namely consistent/inconsistent and true/false, we consider that the inconsistent tuples are as good a source of information as they are the consistent tuples; and similarly, we consider that the false tuples are as good a source of information as they are the true tuples. In doing so we can develop a meaningful theory for the *quality of data* in a data-set (whether the data-set is a table or a query result). To this end, we plan to consider various quality measures such as percentages of consistent/inconsistent tuples in the data-set, percentages of true or false tuples, or combinations thereof.

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

Declarations
The two authors contributed to the study, conception and design. Both read and approved the manuscript.
No funds, grants, or other support was received for conducting this study.

References

[1] 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.

[2] 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.

[3] Leopoldo E. Bertossi. *Database Repairing and Consistent Query Answering.* Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

[4] Leopoldo E. Bertossi. Specifying and computing causes for query answers in databases via database repairs and repair-programs. *Knowl. Inf. Syst.*, 63(1):199–231, 2021.

[5] Leopoldo E. Bertossi and Jan Chomicki. Query answering in inconsistent databases. In Jan Chomicki, Ron van der Meyden, and Gunter Saake, editors, *Logics for Emerging Applications of Databases [outcome of a Dagstuhl seminar]*, pages 43–83. Springer, 2003.

[6] Marco Calautti, Sergio Greco, Cristian Molinaro, and Irina Trubitsyna. Query answering over inconsistent knowledge bases: A probabilistic approach. *Theor. Comput. Sci.*, 935:144–173, 2022.

[7] Marco Calautti, Ester Livshits, Andreas Pieris, and Markus Schneider. Uniform operational consistent query answering. In Leonid Libkin and Pablo Barceló, editors, *PODS '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, pages 393–402. ACM, 2022.

[8] Stavros S. Cosmadakis, Paris C. Kanellakis, and Nicolas Spyratos. Partition semantics for relations. *J. Comput. Syst. Sci.*, 33(2):203–233, 1986.

[9] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

[10] 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.

[11] Floris Geerts, Fabian Pijcke, and Jef Wijsen. First-order under-approximations of consistent query answers. In Christoph Beierle and Alex Dekhtyar, editors, *Scalable Uncertainty Management - 9th International Conference, SUM 2015, Québec City, QC, Canada, September 16-18, 2015. Proceedings*, volume 9310 of *Lecture Notes in Computer Science*, pages 354–367. Springer, 2015.

[12] Dominique Laurent and Nicolas Spyratos. Handling inconsistencies in tables with nulls and functional dependencies. *J. Intell. Inf. Syst.*, 59(2):285–317, 2022.

[13] Mark Levene and George Loizou. *A guided tour of relational databases and beyond.* Springer, 1999.

[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] Ester Livshits, Benny Kimelfeld, and Jef Wijsen. Counting subset repairs with functional dependencies. *J. Comput. Syst. Sci.*, 117:154–164, 2021.

[16] R. Reiter. *On closed World DataBases.* Gallaire et Minker, Plenium Press, New York, 1978.

[17] Nicolas Spyratos. The partition model: A deductive database model. *ACM Trans. Database Syst.*, 12(1):1–37, 1987.

[18] Jeffrey D. Ullman. *Principles of Databases and Knowledge-Base Systems*, volume 1-2. Computer Science Press, 1988.

[19] Jef Wijsen. Database repairing using updates. *ACM Trans. Database Syst.*, 30(3):722–768, 2005.

[20] Jef Wijsen. Foundations of query answering on inconsistent databases. *SIGMOD Rec.*, 48(3):6–16, 2019.

[21] Justin Zobel and Alistair Moffat. Inverted files for text search engines. *ACM Comput. Surv.*, 38(2):6, 2006.

[22] Justin Zobel, Alistair Moffat, and Kotagiri Ramamohanarao. Inverted files versus signature files for text indexing. *ACM Trans. Database Syst.*, 23(4):453–490, 1998.

A On the Church-Rosser Property of Expansion

We show that expansion has the Church-Rosser property, that is when iterating the process, the order in which the steps are preformed is irrelevant. We first recall the definition of expansion.

Definition 4 The expansion of an interpretation I with respect to a functional dependency $X \rightarrow A$ and the tuple xa in $\mathcal{T}(XA)$ is the interpretation $Exp(I, xa)$ defined as follows:

- If $I(x) \cap I(a) \neq \emptyset$ and $I(x) \not\subseteq I(a)$ then:
  $Exp(I, xa)(a) = I(a) \cup I(x)$, and $Exp(I, xa)(\alpha) = I(\alpha)$ for α in $\mathcal{AD}$ different than a.
- Otherwise, $Exp(I, xa) = I$.

We consider two expansions one with respect to the dependency $X \rightarrow A$ and the tuple xa and the other one with respect to the dependency $Y \rightarrow B$ and the tuple yb. Starting with an interpretation I_1, let $I_2 = Exp(I_1, xa)$ and $I_3 = Exp(I_1, yb)$. Referring to Figure 2(a), the problem is to find sequences of expansions called E_2 and E_3 in the figure, such that $I_{24} = E_2(I_2)$ and $I_{34} = E_3(I_3)$ are equal.

It should be clear that if $I_2 = I_1$ or $I_3 = I_1$, then the order in which the expansions are processed is irrelevant. For example, if $I_2 = I_1$ and $I_3 \neq I_1$, we obtain $I_{24} = I_{34}$ by setting $E_3 = \epsilon$ (where ϵ denotes the empty sequence) and $E_2 = Exp(I_1, yb)$.

From now on, we assume that $I_2 \neq I_1$ and $I_3 \neq I_1$. Based on Definition 4, we have:

- $I_2(a) = I_1(a) \cup I_1(x)$ and for every $\alpha \neq a$, $I_2(\alpha) = I_1(\alpha)$
- $I_3(b) = I_1(b) \cup I_1(y)$ and for every $\beta \neq b$, $I_2(\beta) = I_1(\beta)$.

As a particular case, if it is assumed that $a = b$, we have:

- $I_2(a) = I_1(a) \cup I_1(x)$ and for every $\alpha \neq a$, $I_2(\alpha) = I_1(\alpha)$

- $I_3(a) = I_1(a) \cup I_1(y)$ and for every $\alpha \neq a$, $I_3(\alpha) = I_1(\alpha)$.

Thus, as shown in Figure 2(b), for $E_2 = Exp(I_2, ya)$ and $E_3 = Exp(I_3, xa)$, I_{24} and I_{34} are such that $I_{24}(a) = I_{34}(a) = I_1(a) \cup I_1(x) \cup I_1(y)$ and for every $\alpha \neq a$, $I_{24}(\alpha) = I_{34}(\alpha) = I_1(\alpha)$. We therefore have, in this case, $I_{24} = I_{34}$.

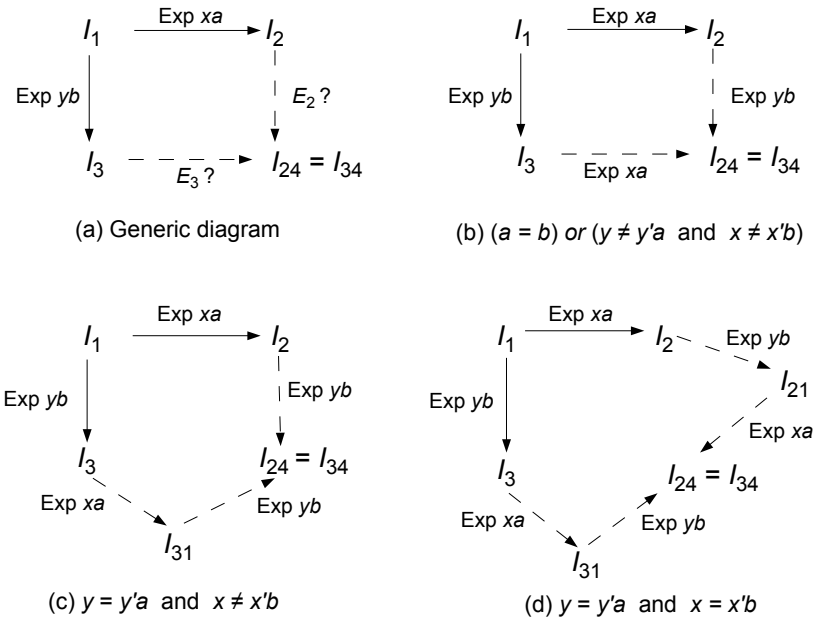

Figure 2: Expansion Diagrams

Now assuming that $I_2 \neq I_1$ and $I_3 \neq I_1$ and $a \neq b$, if a does not occur in y and b does not occur in x, the two expansions are independent from each other. In this case, as illustrated in Figure 2(b), the following lemma holds.

Lemma 6 *If $a \neq b$, if a does not occur in y and b does not occur in x, for $E_2 = Exp(I_2, yb)$ and $E_3 = Exp(I_3, xa)$, denoting respectively by I_{24} and I_{34} the interpretations $Exp(I_2, yb)$ and $Exp(I_3, xa)$, we have $I_{24} = I_{34}$, that is:*

$Exp(Exp(I_1, xa), yb) = Exp(Exp(I_1, yb), xa)$.

Proof. Since a does not occur in y, we have $I_2(y) = I_1(y)$ and similarly, since b does not occur in x, we have $I_3(x) = I_1(x)$. Therefore, for a and b, $I_{24} = Exp(Exp(I_1, xa), yb)$ and $I_{34} = Exp(Exp(I_1, yb), xa)$ are defined by:
– $I_{24}(a) = Exp(I_2, yb)(a) = I_2(a) = I_1(a) \cup I_1(x)$ (because $a \neq b$)
– $I_{24}(b) = Exp(I_2, yb)(b) = I_2(b) \cup I_2(y) = I_1(b) \cup I_1(y)$
– $I_{34}(a) = Exp(I_3, xa)(a) = I_3(a) \cup I_3(x) = I_1(a) \cup I_1(x)$
– $I_{34}(b) = Exp(I_3, xa)(b) = I_3(b) = I_1(b) \cup I_1(y)$ (because $a \neq b$).

Hence, we have $I_{24}(a) = I_{34}(a)$ and $I_{24}(b) = I_{34}(b)$. Since no other symbol is changed by expansions, we have shown the commutativity property in this case, namely $I_{24} = I_{34}$. □

We now switch to the most general case, namely when each expansion has an impact on the other. This happens when a occurs in y and when b occurs in x. Indeed, as shown in Figure 2(d), in this case, writing x and y respectively as bx' and ay', we notice the following: $I_2(y) \neq I_1(y)$ because

$I_2(y) = I_2(a) \cap I_2(y')$ and $I_2(a) \neq I_1(a)$, and similarly, $I_3(x) \neq I_1(x)$ because $I_3(x) = I_3(b) \cap I_3(x')$ and $I_3(b) \neq I_1(b)$.

An important remark regarding x' and y' is that neither a nor b occurs is these tuples, thus implying that, during the computations of the various expansions considered here, their interpretation will not change, thus remaining equal to $I_1(x')$ and $I_1(y')$, respectively. For example, $I_2(y) = I_2(a) \cap I_2(y')$ and $I_3(x) = I_3(b) \cap I_3(x')$ can be respectively written as $I_2(y) = I_2(a) \cap I_1(y')$ and $I_3(x) = I_3(b) \cap I_1(x')$. The following lemma holds in this case.

Lemma 7 *If $a \neq b$, and if $y = ay'$ and $x = bx'$, for $E_2 = Exp(Exp(I_2, yb), xa)$ and $E_3 = Exp(Exp(I_3, xa), yb)$, denoting respectively by I_{24} and I_{34} the corresponding interpretations, we have $I_{24} = I_{34}$, that is:*

$Exp(Exp(Exp(I_1, xa), yb), xa) = Exp(Exp(Exp(I_1, yb), xa), yb)$.

Proof. Referring to Figure 2(d), we first give the computations regarding I_2, I_{21} and I_{24}. In each case the computations of the interpretations of a, x, b and y are shown.

• *Interpretation $I_2 = Exp(I_1, xa)$.*
First, we have $I_2(a) = I_1(a) \cup I_1(x)$, due to expansion. Moreover, $I_2(b) = I_1(b)$ and as $I_2(y) = I_2(a) \cap I_2(y')$, we obtain $I_2(y) = (I_1(a) \cup I_1(x)) \cap I_1(y')$. Since $I_1(y) = I_1(a) \cap I_1(y')$, we have $I_2(y) = I_1(y) \cup (I_1(x) \cap I_1(y'))$. Moreover, $I_2(x) = I_2(b) \cap I_2(x')$, and thus, $I_2(x) = I_1(b) \cap I_1(x') = I_1(x)$. Therefore I_2 is defined as shown below:

$I_2(a) = I_1(a) \cup I_1(x)$ $\quad$ $I_2(x) = I_1(x)$
$I_2(b) = I_1(b)$ $\quad$ $I_2(y) = I_1(y) \cup (I_1(x) \cap I_1(y'))$

• *Interpretation $I_{21} = Exp(I_2, yb)$.*
Now, we have $I_{21}(b) = I_2(b) \cup I_2(y)$, due to expansion. It follows that $I_{21}(b) = I_1(b) \cup I_1(y) \cup (I_1(x) \cap I_1(y'))$. Moreover, $I_{21}(a) = I_2(a)$, that is $I_{21}(a) = I_1(a) \cup I_1(x)$. As $I_{21}(x) = I_{21}(b) \cap I_{21}(x')$ and $I_{21}(x') = I_1(x')$, we obtain

$$\begin{aligned} I_{21}(x) &= (I_1(b) \cup I_1(y) \cup (I_1(x) \cap I_1(y'))) \cap I_1(x') \\ &= (I_1(b) \cap I_1(x')) \cup (I_1(y) \cap I_1(x')) \cup (I_1(x) \cap I_1(y') \cap I_1(x')) \end{aligned}$$

Considering that $I_1(b) \cap I_1(x') = I_1(x)$ and that $I_1(x) \subseteq I_1(x')$ (because $x' \sqsubseteq x$), we obtain: $I_{21}(x) = I_1(x) \cup (I_1(x') \cap I_1(y)) \cup (I_1(x) \cap I_1(y'))$. Since $(I_1(x) \cap I_1(y')) \subseteq I_1(x)$ always holds, we have: $I_{21}(x) = I_1(x) \cup (I_1(x') \cap I_1(y))$.

On the other hand, $I_{21}(y) = I_{21}(a) \cap I_{21}(y')$, and thus, $I_{21}(y) = I_2(a) \cap I_1(y')$. Therefore, $I_{21}(y) = (I_1(a) \cup I_1(x)) \cap I_1(y')$, which can be written as $I_{21}(y) = I_1(y) \cup (I_1(x) \cap I_1(y'))$. Hence I_{21} is defined as shown below:

$I_{21}(a) = I_1(a) \cup I_1(x)$
$I_{21}(x) = I_1(x) \cup (I_1(x') \cap I_1(y))$
$I_{21}(b) = I_1(b) \cup I_1(y) \cup (I_1(x) \cap I_1(y'))$
$I_{21}(y) = I_1(y) \cup (I_1(x) \cap I_1(y'))$

• *Interpretation $I_{24} = Exp(I_{21}, xa)$.*
Here, we have $I_{24}(a) = I_{21}(a) \cup I_{21}(x)$, due to expansion. It follows that:

$$\begin{aligned} I_{24}(a) &= (I_1(a) \cup I_1(x)) \cup (I_1(x) \cup (I_1(x') \cap I_1(y))) \\ &= I_1(a) \cup I_1(x) \cup (I_1(x') \cap I_1(y)) \end{aligned}$$

Moreover, $I_{24}(b) = I_{21}(b)$, that is $I_{24}(b) = I_1(b) \cup I_1(y) \cup (I_1(x) \cap I_1(y'))$. As $I_{24}(x) = I_{24}(b) \cap I_{24}(x')$ and $I_{24}(x') = I_1(x')$, we have:

$$\begin{aligned} I_{24}(x) &= (I_1(b) \cup I_1(y) \cup (I_1(x) \cap I_1(y'))) \cap I_1(x') \\ &= (I_1(b) \cap I_1(x')) \cup (I_1(y) \cap I_1(x')) \cup (I_1(x) \cap I_1(y') \cap I_1(x')) \\ &= I_1(x) \cup (I_1(x') \cap I_1(y)) \cup (I_1(x) \cap I_1(y')) \\ &\quad \text{(because } I_1(x) \subseteq I_1(x') \text{ as } x' \sqsubseteq x) \\ &= I_1(x) \cup (I_1(x') \cap I_1(y)) \text{ (because } (I_1(x) \cap I_1(y')) \subseteq I_1(x)) \end{aligned}$$

As $I_{24}(y) = I_{24}(a) \cap I_{24}(y')$, we obtain

$$\begin{array}{rcl} I_{24}(y) & = & (I_1(a) \cup I_1(x) \cup (I_1(x') \cap I_1(y))) \cap I_1(y') \\ & = & (I_1(a) \cap I_1(y')) \cup (I_1(x) \cap I_1(y')) \cup (I_1(x') \cap I_1(y) \cap I_1(y')) \\ & = & I_1(y) \cup (I_1(x) \cap I_1(y')) \cup (I_1(x') \cap I_1(y)) \\ & & \text{(because } I_1(y) \subseteq I_1(y') \text{ as } y' \sqsubseteq y) \\ & = & I_1(y) \cup (I_1(x) \cap I_1(y')) \text{ (because } (I_1(x') \cap I_1(y)) \subseteq I_1(y)) \end{array}$$

Hence I_{24} is defined as shown below:

$I_{24}(a) = I_1(a) \cup I_1(x) \cup (I_1(x') \cap I_1(y))$
$I_{24}(x) = I_1(x) \cup (I_1(x') \cap I_1(y))$
$I_{24}(b) = I_1(b) \cup I_1(y) \cup (I_1(x) \cap I_1(y'))$
$I_{24}(y) = I_1(y) \cup (I_1(x) \cap I_1(y'))$

Referring again to Figure 2(d), we now turn to the computations regarding I_3, I_{31} and I_{34}. As the computations are similar to those above, some details are skipped.

• *Interpretation* $I_3 = Exp(I_1, yb)$.
Here, we have $I_3(b) = I_1(b) \cup I_1(y)$, due to expansion. Computations similar to those in the case of I_2 show that I_3 is defined as shown below:

$I_3(a) = I_1(a)$ $\quad$ $I_3(x) = I_1(x) \cup (I_1(x') \cap I_1(y))$
$I_3(b) = I_1(b) \cup I_1(y)$ $\quad$ $I_3(y) = I_1(y)$

• *Interpretation* $I_{31} = Exp(I_3, xa)$.
Here, we have $I_{31}(a) = I_3(a) \cup I_3(x)$, due to expansion. Computations similar to those for I_{24} show that I_{31} is defined as shown below:

$I_{31}(a) = I_1(a) \cup I_1(x) \cup (I_1(x') \cap I_1(y))$
$I_{31}(x) = I_1(x) \cup (I_1(x') \cap I_1(y))$
$I_{31}(b) = I_1(b) \cup I_1(y)$
$I_{31}(y) = I_1(y) \cup (I_1(x) \cap I_1(y'))$

• *Interpretation* $I_{34} = Exp(I_{31}, yb)$.
Here, we have $I_{34}(b) = I_{31}(b) \cup I_{31}(y)$, due to expansion. It follows that computations similar to those for I_{24} show that I_{34} is defined as shown below:

$I_{34}(a) = I_1(a) \cup I_1(x) \cup (I_1(x') \cap I_1(y))$
$I_{34}(x) = I_1(x) \cup (I_1(x') \cap I_1(y))$
$I_{34}(b) = I_1(b) \cup I_1(y) \cup (I_1(x) \cap I_1(y'))$
$I_{34}(y) = I_1(y) \cup (I_1(x) \cap I_1(y'))$

It thus turns out that we indeed have $I_{24} = I_{34}$, and the proof is complete. □

The last case to be addressed is when $y = ay'$ and $x \neq bx'$ or when $x = bx'$ and $y \neq ay'$. We only consider the former case since the latter can be dealt with in a similar way. Moreover, as shown in Figure 2(c), this case is in fact a simplification of the case of Lemma 7 where the terms involving x' are omitted. We thus omit the proof and just state that is this case again $I_{24} = I_{34}$. It can be checked that in case of Figure 2(c) (that is $y = ay'$ and $x \neq bx'$), we obtain the following interpretations:

$I_{24}(a) = I_{34}(a) = I_1(a) \cup I_1(x)$
$I_{24}(x) = I_{34}(x) = I_1(x)$
$I_{24}(b) = I_{34}(b) = I_1(b) \cup I_1(y) \cup (I_1(x) \cap I_1(y'))$
$I_{24}(y) = I_{34}(y) = I_1(y) \cup (I_1(x) \cap I_1(y'))$

We therefore state the following basic result regarding expansion:

> *The process of expanding an interpretation I with respect to a set FD of functional dependencies satisfies the Church-Rosser property.*

Hence, given T and FD, the process of expanding I^b until a fixed point is reached does *not* depend on the order the expansions are processed. Thus, the limit is indeed *unique*.

B Proof of Lemma 4

Lemma 4 *Let T be a table over universe U. Then the following holds:*

1. *Algorithm 1 applied to T always terminates.*
2. *For every m-tuple s in $\mathcal{MT}$, $I^*(s) \neq \emptyset$ holds if and only if there exists σ in Σ^* such that $s \sqsubseteq \sigma$.*
3. *$\mathsf{Inc}(\mathcal{T}) = \emptyset$ if and only if $fail = false$.*
4. *For every σ in Σ^* and all t and t' in $\mathsf{tuples}(\sigma)$, $I^*(t) = I^*(t')$.*

Proof. **1.** The computation of Σ^* terminates because (i) the sets under construction for a given attribute A are all bounded by the *finite* set $adom(A)$, and (ii) the construction is monotonous, in the sense that if Σ^k and Σ^{k+1} are two consecutive states, for every σ in Σ^{k+1}, there exists σ' in Σ^k such that $\sigma' \sqsubseteq \sigma$.

2. Given s in $\mathcal{MT}$, we first show that $I^*(s)$ is nonempty if there exists σ in Σ^* such that $s \sqsubseteq \sigma$. The proof is conducted by induction on the steps in the runs of the loop line 4. We show that if $(\Sigma^k)_{k\geq 0}$ denotes the sequence of the states of Σ^* during the computation, for every σ in Σ^k, $I^*(\sigma) \neq \emptyset$. We first note in this respect that since $\Sigma^0 = T$, for every σ in Σ^0, we have $I^*(\sigma) \neq \emptyset$.

Assuming now that for $k \geq 0$, for every $\sigma \in \Sigma^k$, $I^*(\sigma) \neq \emptyset$, we prove the result for every σ in Σ^{k+1}. Let σ in Σ^{k+1} but not in Σ^k. Then σ occurs in Σ^{k+1} because there exist $X \to A$ in FD, σ_1 and σ_2 in Σ^k such that for every B in X, $\sigma_1.B \cap \sigma_2.B \neq \emptyset$. Let $\sigma_1' = \sigma_1 \sqcup \sigma_2(A)$ and $\sigma_2' = \sigma_2 \sqcup \sigma_1(A)$. Then, either $\sigma = \sigma_1'$ and $\sigma_1' \neq \sigma_1$ or $\sigma = \sigma_2'$ and $\sigma_2' \neq \sigma_2$. The two cases being similar, we just consider the first one, that is $\sigma = \sigma_1'$ and $\sigma_1' \neq \sigma_1$, which implies that $\sigma_2(A) \neq \emptyset$ and $\sigma_2(A) \not\sqsubseteq \sigma_1$. By our induction hypothesis, we have $I^*(\sigma_2) \neq \emptyset$ and thus denoting by x an X-tuple occurring in $\mathsf{tuples}(\sigma_1 \sqcap \sigma_2)$, we have $I^*(x) \cap I^*(\sigma_2(A)) \neq \emptyset$. Since I^* satisfies $X \to A$, $I^*(x) \subseteq I^*(\sigma_2(A))$. As $x \sqsubseteq \sigma_1$, $I^*(\sigma_1) \subseteq I^*(x)$ and thus, $I^*(\sigma_1) \subseteq I^*(\sigma_2(A))$. Therefore, $I^*(\sigma) = I^*(\sigma_1) \cap I^*(\sigma_2(A)) = I^*(\sigma_1)$. As by our induction hypothesis, $I^*(\sigma_1) \neq \emptyset$, we obtain that $I^*(\sigma) \neq \emptyset$. Therefore, it holds that for every $k \geq 0$, $I^*(\sigma) \neq \emptyset$ for every σ in Σ^k, and so, it holds that $I^*(\sigma) \neq \emptyset$ for every σ in Σ^*. Thus, if s satisfies that there exists σ in Σ^* such that $s \sqsubseteq \sigma$, we have $I^*(\sigma) \subseteq I^*(s)$. Hence $I^*(s) \neq \emptyset$, and this part of the proof is complete.

Conversely, we show that for every s, if $I^*(s) \neq \emptyset$ then there exists σ in Σ^* such that $s \sqsubseteq \sigma$. The proof is done by induction on the construction of I^*.

By definition of $I^0 = I^b$, if $I^0(s) \neq \emptyset$ then s is a sub-tuple of a tuple t in T, i.e., $s \sqsubseteq t$ holds. Since $\Sigma^0 = T$, there exists σ in Σ^0 such that $t \sqsubseteq \sigma$. By monotonicity of the construction of Σ^* (i.e., for every σ^{k+1} in Σ^{k+1} there exists σ^k in Σ^k such that $\sigma^k \sqsubseteq \sigma^{k+1}$), there exists σ^* in Σ^* such that $\sigma \sqsubseteq \sigma^*$, which shows that $t \sqsubseteq \sigma^*$, thus that $s \sqsubseteq \sigma^*$.

Now, assuming that for every $k \geq 0$ and every s, if $I^k(s) \neq \emptyset$ then there exists σ in Σ^* such that $s \sqsubseteq \sigma$, we prove that this result holds for I^{k+1}.

Let s be such that $I^k(s) = \emptyset$ and $I^{k+1}(s) \neq \emptyset$. For every a in $\mathcal{AD}$ we have $I^{k+1}(a) = I^k(a) \cup I^k(x)$ if x and a are the tuples involved in $Exp(I^k, xa)$ and $I^{k+1}(a) = I^k(a)$ otherwise. Since $I^{k+1}(s) \neq I^k(s)$ and since a is the only symbol for which I^k has changed, it must be that $a \sqsubseteq s$. Therefore, writing s as $s' \sqcup a$, we have $I^{k+1}(s) = I^{k+1}(s') \cap I^{k+1}(a)$ and $I^{k+1}(s') = I^k(s')$. It follows that $I^{k+1}(s) = I^k(s') \cap (I^k(a) \cup I^k(x))$, thus that $I^{k+1}(s) = (I^k(s') \cap (I^k(a)) \cup (I^k(s') \cap I^k(x))$, that is $I^{k+1}(s) = I^k(s) \cup (I^k(s') \cap I^k(x))$. As $I^{k+1}(s) \neq \emptyset$ and $I^k(s) = \emptyset$, we have

$I^k(s') \cap I^k(x) \neq \emptyset$, in which case our induction hypothesis entails that there exists σ in Σ^* such that $(s' \sqcup x) \sqsubseteq \sigma$. Since by definition of expansion, we also have $I^k(x) \cap I^k(a) \neq \emptyset$, Σ^* contains an m-tuple σ' such that $xa \sqsubseteq \sigma'$. In this case, Algorithm 1 shows that there exists σ^* in Σ^* such that $(s' \sqcup a) \sqsubseteq \sigma^*$, that is $s \sqsubseteq \sigma^*$. This part of the proof is therefore complete.

3. If the returned value for $fail$ is false, the previous item shows that there exist $X \to A$ in FD, x in $\mathcal{T}(X)$ and a and a' in $adom(A)$ such that $I^*(a) \cap I^*(a') \neq \emptyset$. Therefore in this case, $\mathsf{Inc}(\mathcal{T})$ is not empty. Conversely, if $\mathsf{Inc}(\mathcal{T}) \neq \emptyset$, for every t in $\mathsf{Inc}(\mathcal{T})$, there exist A in U and a and a' in $adom(A)$ such that $I^*(a) \cap I^*(a') \neq \emptyset$, and the previous item shows that there exists σ in Σ^* such that $(aa') \sqsubseteq \sigma$. Hence, during the execution of Algorithm 1, the value of $fail$ must have been turned to *true* line 11.

4. The proof is by induction on the construction of Σ^*. As all m-tuples in Σ^0 can be seen as tuples, for every σ in Σ^0, $\mathsf{tuples}(\sigma)$ consists of one tuple. The result thus trivially holds in this case. Now assume that the result holds for Σ^k, and let σ be in Σ^{k+1} obtained by steps lines 16-20 of Algorithm 1. Thus there exist σ' in Σ^*, $X \to A$ in FD, x_0 in $\mathcal{T}(X)$ and a_0 in $adom(A)$ such that in Σ^k, $x_0 \in \mathsf{tuples}(\sigma(X)) \cap \mathsf{tuples}(\sigma'(X))$, a_0 is in $\mathsf{tuples}(\sigma'(A)) \setminus \mathsf{tuples}(\sigma(A))$, and in Σ^{k+1}, $\sigma(A)$ becomes $\sigma_1(A)$ such that $a_0 \in \mathsf{tuples}(\sigma_1(A))$.

Let t_1 be in $\mathsf{tuples}(\sigma_1)$ such that $t_1.A = a_0$, and let x denote $t_1.X$. Then, t_1 is not in $\mathsf{tuples}(\sigma)$ (because $a_0 \notin \mathsf{tuples}(\sigma(A))$), but $\mathsf{tuples}(\sigma)$ contains a tuple t such that $t.X = x$ and a tuple t' such that $t'.X = x_0$. Thus $\mathsf{tuples}(\sigma_1)$ contains t_1 such that $t_1.XA = xa_0$ and a tuple t'_1 such that $t'_1.XA = x_0a_0$.

On the other hand, the fact that x_0a_0 occurs in $\mathsf{tuples}(\sigma')$ implies that $I^*(x_0a_0) \neq \emptyset$, thus that $I^*(x_0) \subseteq I^*(a_0)$. Hence, $I^*(x_0a_0) = I^*(x_0)$. We now argue that $I^*(t'_1) = I^*(t')$. Indeed, if $\sigma(A) = \emptyset$, then t'_1 can be written as $t'a_0$ in which case $I^*(t'_1) = I^*(t') \cap I^*(a_0)$ holds. Since it also holds that $I^*(t') \subseteq I^*(x_0) \subseteq I^*(a_0)$, we have $I^*(t') \cap I^*(a_0) = I^*(t')$. Otherwise, if $\sigma(A) \neq \emptyset$ then t' is written as $q'a'$ and thus t'_1 is written as $q'a_0$ (a' has been replaced by a_0 to obtain t'_1 from t'). In this case, $I^*(t'_1) = I^*(q') = I^*(t')$.

As t and t' are both in $\mathsf{tuples}(\sigma)$, by our induction hypothesis, in Σ^i we have $I^*(t) = I^*(t')$, which implies that $I^*(t) = I^*(t'_1) = I(t')$. We therefore obtain that for every t_1 in $\mathsf{tuples}(\sigma_1)$, there exists t in $\mathsf{tuples}(\sigma)$ such that $I^*(t_1) = I^*(t)$. As all tuples in t $\mathsf{tuples}(\sigma)$ have the same $I^*(t)$, all tuples t_1 in $\mathsf{tuples}(\sigma_1)$ have the same $I^*(t_1)$. The proof is therefore complete. □

C Proof of Proposition 5

Proposition 5 *Let T be a non-redundant table over universe U, FD the set of associated functional dependencies and Σ^* the non-redundant m-table as computed by Algorithm 1. Given a query Q : `select` X `from` T `[where` Γ`]`, and a tuple x in $\mathcal{T}(X)$:*

1. *x is in $\mathsf{MSC\_ans}(Q)$ if and only if*
   (a) *for every σ in $\Sigma(x)$, $\mathsf{tuples}(\sigma(X)) = \{x\}$,*
   (b) *there exists σ in $\Sigma_Q(x)$ such that $|\mathsf{tuples}(\sigma)| = 1$.*
2. *x is in $\mathsf{SC\_ans}(Q)$ if and only if*
   (a) *for every σ in $\Sigma(x)$, $\mathsf{tuples}(\sigma(X)) = \{x\}$,*
   (b) *there exists s in $Sat(\Gamma) \cap \mathsf{True}(\mathcal{T})$ such that for every σ in $\Sigma_Q(x)$, if s is in $\sigma(sch(\Gamma))$ then $\mathsf{tuples}(\sigma(sch(\Gamma)))) = \{s\}$.*
3. *x is in $\mathsf{C\_ans}(Q)$ if and only if*

(a) for every σ *in* $\Sigma(x)$, $\mathsf{tuples}(\sigma(X)) = \{x\}$,

(b) there exists σ *in* $\Sigma_Q(x)$ *such that* $\mathsf{tuples}(\sigma(sch(\Gamma))) \subseteq Sat(\Gamma)$.

4. x *is in* $\mathsf{WC\_ans}(Q)$ *if and only if*

(a) for every σ *in* $\Sigma(x)$, $\mathsf{tuples}(\sigma(X)) = \{x\}$,

(b) $\Sigma_Q(x)$ *is not empty.*

PROOF. In this proof, Q is assumed to involve a selection condition Γ, because the case where no selection condition is involved is a simplification, and thus, is omitted.

1. By Proposition 4(1) and (2), statement (a) in the proposition is equivalent to statement (1.a) in Definition 6. Moreover, as Σ^* is assumed to be non-redundant, for every σ in Σ^* and every t in $\mathsf{tuples}(\sigma)$, there exists no tuple t' in $\mathsf{True}(\mathcal{T})$ such that $t \sqsubset t'$. Thus, by Proposition 4(2), statement (b) in the proposition yields a tuple having the same properties as the tuple t in statement (1.b) of Definition 6 (i.e., t is a maximal true and consistent tuple whose restriction over $sch(\Gamma)$ satisfies Γ). Therefore this part of the proof is complete.

2. As above, by Proposition 4(1) and (2), statement (a) in the proposition is equivalent to statement (2.a) in Definition 6. Moreover, statement (b) in the proposition yields a tuple t such that $t.X = x$, $t.sch(\Gamma) = s$, which is in $Sat(\Gamma)$. Moreover, Proposition 4(3) implies that this tuple t is true and consistent, thus satisfying the statement (2.b) of Definition 6 (i.e., t is a true and consistent tuple whose restriction over $sch(\Gamma)$ satisfies Γ). Therefore this part of the proof is complete.

3. Let us first assume that x is in $\mathsf{C\_ans}(Q)$. As x is in $\mathsf{Cons}(\mathcal{T}) \cap \mathsf{True}(\mathcal{T})$, by Proposition 4, Σ^* must contain at least one σ such that $x \sqsubseteq \sigma$ and every such m-tuple must be such that $\sigma(X) = \{x\}$. Hence $\Sigma(x)$ is nonempty and the first item in the proposition holds. Moreover, by Definition 6, one of these m-tuples σ is also such that $sch(\Gamma) \subseteq sch(\sigma)$ and $\mathsf{tuples}(\sigma)$ contains a tuple t of $\mathsf{Cons}(\Gamma, \mathcal{T})$. By Definition 5, this implies that the following statement (*) holds: $t.sch(\Gamma)$ is in $Sat(\Gamma)$ and for every s in $Sat^-(\Gamma)$, $I^*(t) \cap I^*(s) = \emptyset$. This in particular implies that σ is in $\Sigma_Q(x)$, because $t.(sch(\Gamma)) \sqsubseteq t \sqsubseteq \sigma$ holds.

As $\Sigma_Q(x) \subseteq \Sigma(x)$, every σ in $\Sigma_Q(x)$ is such that $\sigma(X) = \{x\}$. Assume now that every σ in $\Sigma_Q(x)$ is such that $\mathsf{tuples}(\sigma(sch(\Gamma)))$ contains q_0 not in $Sat(\Gamma)$. Then q_0 is in $Sat^-(\Gamma)$ and $\mathsf{tuples}(\sigma)$ contains a tuple t_0 such that $t_0.sch(\Gamma) = q_0$. Thus we have two tuples t and t_0 in $\mathsf{tuples}(\sigma)$ such that $t.X = t_0.X = x$, $t.sch(\Gamma) \in Sat(\Gamma)$ and $t_0.sch(\Gamma) \in Sat^-(\Gamma)$. Lemma 4(4) implies that $I^*(t) = I^*(t_0)$ (because t and t_0 are in $\mathsf{tuples}(\sigma)$), and as $I^*(t_0) \subseteq I^*(q_0)$ (because $t_0.sch(\Gamma) = q_0$), we have $I^*(t_0) \cap I^*(q_0) = I^*(t_0)$. Thus $I^*(t_0) \cap I^*(q_0) \neq \emptyset$, implying that $I^*(t) \cap I^*(q_0) \neq \emptyset$, which is a contradiction with statement (*) above. We therefore have shown that if x is in $\mathsf{C\_ans}(Q)$ then the items in the proposition hold.

Conversely, if the items in the proposition are satisfied, Proposition 4 implies that x is in $\mathsf{Cons}(\mathcal{T}) \cap \mathsf{True}(\mathcal{T})$. Thus by Definition 5, we have to prove the existence of a tuple t in $\mathsf{Cons}(\Gamma, \mathcal{T}) \cap \mathsf{True}(\mathcal{T})$ such that $x \sqsubseteq t$. Let σ be in $\Sigma_Q(x)$ as stated in item (3.b) of the proposition, that is such that $\mathsf{tuples}(\sigma(sch(\Gamma))) \subseteq Sat(\Gamma)$, and let t_0 be in $\mathsf{tuples}(\sigma)$. As t_0 is in $\mathsf{True}(\mathcal{T})$, we have to show that t_0 is in $\mathsf{Cons}(\Gamma, \mathcal{T})$, that is that there exists q in $Sat(\Gamma)$ such that (i) $q \sqsubseteq t_0$, and (ii) for every q' in $Sat^-(\Gamma)$, $I^*(t_0) \cap I^*(q') = \emptyset$. Since we know that $t_0.sch(\Gamma)$ is in $Sat(\Gamma)$, (i) holds for $q = t_0.sch(\Gamma)$. Regarding (ii), let q'_0 be in $Sat^-(\Gamma)$ such that $I^*(t_0) \cap I^*(q'_0) \neq \emptyset$. As $I^*(\sigma) = \bigcap_{u \in \mathsf{tuples}(\sigma)} I^*(u)$, Lemma 4(4) implies that $I^*(\sigma) = I^*(t_0)$, and thus that $I^*(\sigma) \cap I^*(q'_0) \neq \emptyset$. Applying now Lemma 4(2), we obtain that Σ^* contains an m-tuple σ' involving all attribute values in σ or in q'_0. Since q'_0 is not in $\mathsf{tuples}(\sigma)$, we have $\sigma' \neq \sigma$, and thus $\sigma \sqsubset \sigma'$ holds. This is a contradiction with our hypothesis that Σ^* contains no redundancy, showing that t_0 is in $\mathsf{Cons}(\Gamma, \mathcal{T})$. This part of the proof is therefore complete.

4. As in cases (1) and (2) above, by Proposition 4(1) and (2), statement (4.a) in the proposition is equivalent to statement (2.a) in Definition 6. Moreover, $\Sigma_Q(x) \neq \emptyset$ is equivalent to the existence of σ in Σ^* such that $\sigma(X) = \{x\}$ and $\sigma(sch(\Gamma))$ contains a tuple s of $Sat(\Gamma)$. Hence, $\Sigma_Q(x) \neq \emptyset$ is equivalent to the existence of a tuple t in $\mathsf{True}(\mathcal{T})$ (due to Proposition 4(1)) such that $t.X = x$ and $t.sch(\Gamma)$ is in $Sat(\Gamma)$. The proof is therefore complete. □