A

SQL's Three-Valued Logic and Certain Answers

LEONID LIBKIN, School of Informatics, University of Edinburgh

The goal of the paper is to bridge the difference between theoretical and practical approaches to answering queries over databases with nulls. Theoretical research has long ago identified the notion of correctness of query answering over incomplete data: one needs to find certain answers, which are true regardless of how incomplete information is interpreted. This serves as the notion of correctness of query answering, but carries a huge complexity tag. In practice, on the other hand, query answering must be very efficient, and to achieve this, SQL uses three-valued logic for evaluating queries on databases with nulls. Due to the complexity mismatch, the two approaches cannot coincide, but perhaps they are related in some way? For instance, does SQL always produce answers we can be certain about?

This is not so: SQL's and certain answers semantics could be totally unrelated. We show, however, that a slight modification of the three-valued semantics for relational calculus queries can provide the required certainty guarantees. The key point of the new scheme is to fully utilize the three-valued semantics, and classify answers not into certain or non-certain, as was done before, but rather into certainly true, certainly false, or unknown. This yields relatively small changes to the evaluation procedure, which we consider at the level of both declarative (relational calculus) and procedural (relational algebra) queries. These new evaluation procedures give us certainty guarantees even for queries returning tuples with null values.

Categories and Subject Descriptors: H.2.1 [**Database Management**]: Logical Design—*Data Models*; H.2.1 [**Database Management**]: Languages—*Query Languages*; H.2.4 [**Database Management**]: Systems—*Query Processing*

Additional Key Words and Phrases: Null values, incomplete information, query evaluation, three-valued logic, certain answers

1. INTRODUCTION

SQL's query evaluation uses three-valued logic when it comes to handling incomplete information: comparisons involving null values have the truth value *unknown* [Date and Darwen 1996]. The design of these features of SQL has been heavily criticized [Date 2005; Date and Darwen 1996] and in fact leads to a number of well known paradoxes. Consider, for instance, a relation S with a single numerical attribute A, and assume that S contains a single row with a null value in it. Then

```
select S.A from S where S.A = 0 or S.A <> 0        (1)
```

returns no tuple. However, if we view nulls as missing values (and this is the interpretation we shall concentrate on), the condition in the `where` clause is a tautology, and the answer looks intuitively wrong. The reason this happens is that both `null = 0` and `null <> 0` evaluate to *unknown* and so does their disjunction. If we have another relation R with the same numerical attribute A, then for the same reason the query computing $R - S$:

```
select R.A from R where R.A not in (select S.A from S)        (2)
```

Author's address: School of Informatics, University of Edinburgh, Informatics Forum, 10 Crichton Street, Edinburgh EH8 9AB, United Kingdom.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© YYYY ACM 0362-5915/YYYY/01-ARTA $10.00
DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

returns nothing if S contains a single null, no matter what R is, telling us that we might well have $|R| > |S|$ and $R - S = \emptyset$ at the same time.

These do seem rather counterintuitive, and a number of attempts were made to overcome the problem, from fixing the underlying many-valued logic [Gessert 1990; Yue 1991] to getting rid of nulls altogether [Darwen and Date 1995]. However, to understand why, and whether, these query answers are wrong, and to state correctness guarantees, we must have some yardstick to measure query evaluation procedures against. Such a yardstick was produced by the theory community a long time ago. Since the foundational papers [Lipski 1979; Imielinski and Lipski 1984] it has been accepted that the right way of evaluating queries on incomplete databases is by computing *certain answers* to them. Each incomplete database D has an associated *semantics* $[\![D]\!]$ that we can think of as the set of possible complete databases that D can represent, e.g., all databases obtained by substituting values for nulls, if those are interpreted as missing information. Then certain answers contain tuples that will be in the answer to Q over all possible complete databases represented by D:

$$\text{cert}(Q, D) \;=\; \bigcap \{Q(D') \mid D' \in [\![D]\!]\} \tag{3}$$

In other words, $\text{cert}(Q, D)$ contains tuples that will be in the answer regardless of the interpretation of the missing information in D.

Taking certain answers as the *correct* way of answering queries, there could be two ways in which a specific evaluation procedure (e.g., the one implemented in SQL) differs from it. It can produce one of the following.

False negatives. In this case the evaluation procedure fails to produce some of the certain answers (i.e., it says that a tuple is *not* the answer when it should be).

False positives. In this case the evaluation procedure produces an answer that is not certain.

We can think of false negatives as hiding some of the truth, while false positives are an outright lie.

Can SQL query evaluation avoid those? The first observation, based on what we know about incomplete databases, is this:

SQL query evaluation must produce either false negatives or false positives.

The reason is that certain answers, while providing us with the notion of correctness, are not easy to find: in fact for full relational calculus they are coNP-hard for most reasonable semantics [Abiteboul et al. 1991]. And yet SQL evaluation is in DLOGSPACE (in fact, even in the low parallel complexity class AC^0, which is properly contained in coNP). This complexity mismatch tells us that SQL cannot compute certain answers precisely.

Example (1) is an example of false negatives: we miss a certain answer but at least we do not produce false ones. Example (2) actually computes the certain answer, but could easily be modified to generate false negatives. So it seems from those typical cases used to criticize SQL's handling of nulls that at least we do not produce false positives.

This, however, is not so. Consider the query:

```
select R.A from R
where R.A not in (select R1.A from R R1
                  where R1.A not in (select * from S))
```
(4)

expressing $R - (R - S)$. Now look at a database in which $R = \{1\}$ and $S = \{\texttt{null}\}$. SQL's evaluation results in $\{1\}$. At the same time the certain answer is empty: if null

is interpreted as any value other than 1, the result of $R - (R - S)$ is $\emptyset$. Even avoiding double negation, the query

```
(select * from R) except (select * from S)
```

outputs $\{1\}$ while the certain answer is empty.

Thus, SQL's evaluation rules may result in both false positives and false negatives, and we cannot eliminate both. So the best we can hope for is an evaluation scheme that eliminates one kind of error. We take the view here that the outright lie (false positives) is worse than hiding some of the truth (false negatives).

This leads to the main question of the paper: *Can we devise an evaluation strategy of SQL's queries that completely eliminates false positives*? The idea of such a sound evaluation is not new. It was considered as early as [Reiter 1986], even before complexity bounds for certain answers were known. And yet despite some partial attempts (e.g., for fixing the null semantics of SQL queries with `any` or `all` subquery comparisons [Klein 1994]) there is currently no answer to this question.

Our goal is to provide this answer, and to present an evaluation strategy that:

— finds query answers fast, without a significant modification of the existing evaluation techniques, and at the same time
— guarantees that no false positives occur, i.e., every returned tuple is a certain answer.

We achieve this by providing a small modification to the three-valued logic approach of SQL that restores correctness guarantees: query evaluation no longer produces false positives, and all returned results are guaranteed to be certain answers.

To understand the idea of the modification, notice that SQL's query evaluation actually *mixes* three- and two-valued logic. Three-valued logic is used to evaluate conditions, but then query results return only those tuples for which conditions evaluate to *true*, effectively collapsing *unknown* and *false*. This works fine for positive queries, but once negation, especially negation in subqueries (e.g., `not in` or `not exists`) enters the picture, we have a problem, as it flips truth values. Now *true* flips to *false*, but both *unknown* and *false* (which were collapsed to one value when a subquery was evaluated) flip to *true*! This is exactly how false positives occur in query answers.

So to get correctness guarantees, we just need to be faithful to the three-valued approach. This means that there will be three possible outcomes for each candidate answer tuple: it can be either

— certainly in the answer (truth value *true*); or
— certainly *not* in the answer (truth value *false*); or
— possibly in the answer, or possibly not (truth value *unknown*).

We provide procedures with correctness guarantees for the evaluation of relational calculus and relational algebra queries. In terms of the SQL fragment, this covers the usual `select-from-where` queries, with subqueries, but without grouping and aggregation. Note that we do not work directly on SQL queries, rather on those they get translated into (in particular, the procedural relational algebra queries, although for understanding what exactly needs to be repaired, it is best to do it first with the declarative calculus queries).

As for the model of incompleteness, we choose here to work with *marked*, or *naïve* nulls [Abiteboul et al. 1995; Imielinski and Lipski 1984; Lipski 1984] in tables. They are more general than SQL's nulls: they can appear multiple times in tables. Marked nulls are often required by applications such as data integration and exchange [Arenas et al. 2014; Lenzerini 2002]. In fact they have already been implemented in connection with such applications [Haas et al. 2005; Marnette et al. 2011]. SQL's assumption (for

instance, for attribute comparisons) is that different appearances of nulls are distinct. Thus, SQL's nulls can be modeled with naïve nulls, simply by forbidding repetition.

The reason we need marked nulls is twofold. Firstly, we want to produce more general results. Secondly, we need to overcome an additional (and quite unreasonable) deficiency of SQL's handling of nulls: even checking whether a null value equals *itself* produces truth value *unknown*. Indeed, consider a table `T(A,B)` with a single tuple `(1,null)` and a query `select T.A from T where T.B=T.B`, i.e., $\pi_A(\sigma_{B=B}(T))$. Instead of the expected 1, it gives the empty result, as comparing a value with itself does not evaluate to true.

The plan of the paper is as follows. In Section 2 we present basic definitions. Section 3 describes the evaluation procedure for relational calculus and SQL's three-valued approach in the presence of nulls. Section 4 presents the modified evaluation procedure and states its correctness. In Section 5 we prove a generalization of that result, relying on a new notion of certain answers with nulls. This generalization properly accounts for all three possible outcomes of query evaluation (certainly true, certainly false, unknown). This notion is also useful in that it allows us to have tuples with nulls in query answers, something that the standard certain answers do not allow. In Section 6 we look at certainty guarantees at the procedural language, that is, for relational algebra queries. Section 7 delves deeper into some bizarre behavior of SQL for nested queries on databases with nulls, and shows that our new evaluation techniques overcome those instances of unexpected behavior. Concluding remarks are in Section 8.

In the paper, we assume the textbook version of relational algebra and calculus, that is, the version based on the set semantics. SQL, of course, uses the multiset semantics. Understanding how to extend results to the multiset semantics is a project for future work. We do not, at this time, have a proper analog of certain answers and relational theory of incompleteness for databases with multisets, and thus do not have the same yardstick as the definition (3) to base the notion of correctness on. This needs to be done first, before our results can be extended to SQL's features going beyond relational calculus/algebra.

2. PRELIMINARIES

Incomplete databases. We begin with some standard definitions [Abiteboul et al. 1995; Imielinski and Lipski 1984]. Incomplete databases are populated by *constants* and *nulls*. The sets of constants and nulls are countably infinite sets denoted by Const and Null respectively. Nulls are denoted by $\bot$, sometimes with sub- or superscripts.

A relational schema (vocabulary) is a set of relation names with associated arities. An incomplete relational instance D assigns to each k-ary relation symbol S from the vocabulary a k-ary relation S^D over $\mathsf{Const} \cup \mathsf{Null}$, i.e., a finite subset of $(\mathsf{Const} \cup \mathsf{Null})^k$. When the instance is clear from the context we shall write S, rather than S^D, for the relation itself.

The sets of constants and nulls that occur in D are denoted by $\mathsf{Const}(D)$ and $\mathsf{Null}(D)$. If $\mathsf{Null}(D)$ is empty, we refer to D as *complete*. That is, complete databases are those without nulls. The *active domain* of D is $\mathrm{adom}(D) = \mathsf{Const}(D) \cup \mathsf{Null}(D)$.

Homomorphisms, valuations, and semantics. Given two relational structures D and D', a homomorphism $h : D \to D'$ is a map from the active domain of D to the active domain of D' such that:

(1) for every relation symbol S, if a tuple $\bar{u}$ is in relation S in D, then the tuple $h(\bar{u})$ is in the relation S in D'; and
(2) $h(c) = c$ for every $c \in \mathsf{Const}(D)$.

By $h(D)$ we denote the image of D, i.e., the set of all tuples $S(h(\bar{u}))$ where $S(\bar{u})$ is in D. If $h : D \to D'$ is a homomorphism, then $h(D)$ is a subinstance of D'.

A homomorphism $h : D \to D'$ is called a *valuation* if $h(x)$ is a constant for every $x \in \text{adom}(D)$; in other words, it provides a valuation of nulls as constant values. If h is a valuation, then $h(D)$ is complete. We now define the semantics of incomplete databases by means of valuations:

$$[\![D]\!] \;=\; \{h(D) \mid h \text{ is a valuation}\}.$$

This is often referred to as the closed-world assumption, or CWA semantics of incompleteness [Imielinski and Lipski 1984; Reiter 1977]. Another common semantics uses the open-world assumption, or OWA, and allows adding complete tuples to $h(D)$. In the study of incompleteness, the closed-world semantics is a bit more common [Abiteboul et al. 1995; Abiteboul et al. 1991; Imielinski and Lipski 1984] since it is better behaved. We shall also consider evaluation under the OWA semantics in Section 5.2.

Query languages. As our basic query languages we consider relational calculus and its fragments. Relational calculus has exactly the power of *first-order logic*, or FO. Its formulae are built from relational atoms $R(\bar{x})$ and equality atoms $x = y$, by closing them under conjunction $\wedge$, disjunction $\vee$, negation $\neg$, existential $\exists$ quantifiers and universal $\forall$ quantifiers. If $\bar{x}$ is the list of free variables of a formula φ, we write $\varphi(\bar{x})$ to indicate this. We write $|\bar{x}|$ for the length of $\bar{x}$.

Conjunctive queries (CQs, also known as select-project-join queries) are defined as queries expressed in the $\exists, \wedge$-fragment of FO. The class UCQ of *unions of conjunctive queries* is the class of formulae of the form $\varphi_1 \vee \ldots \vee \varphi_m$, where each φ_i is a conjunctive query. In terms of its expressive power, this is the existential-positive fragment of FO, i.e., the $\exists, \vee, \wedge$-fragment.

We shall use relational algebra, the procedural language equivalent to FO, that has the operations of selection σ, projection π, cartesian product $\times$, union $\cup$, and difference $-$. We use the unnamed perspective of relational algebra which does not require the renaming operator [Abiteboul et al. 1995] (more on this in Section 6, where we shall add explicit intersection to relational algebra). The fragment without the difference operator is referred to as positive relational algebra; it has the same expressiveness as existential positive formulae (and thus unions of conjunctive queries).

3. EVALUATION PROCEDURES FOR FO QUERIES

We shall look at different query evaluation procedures. An evaluation procedure has as its inputs a database D, a query $\varphi(\bar{x})$, and a tuple $\bar{a}$ of free variables of the query; it has to say whether $\varphi(\bar{a})$ is true in D.

Thus, formally an evaluation procedure Eval will take a query (an FO formula) $\varphi(\bar{x})$, a database D, and an assignment ν of values to the free variables $\bar{x}$. The output $\mathsf{Eval}(\varphi, D, \nu)$ is a *truth value*. For the standard Boolean logic, the domain of truth values is $\{0, 1\}$, with 0 meaning *false* and 1 meaning *true*. For the three-valued logic, used by SQL, the domain is $\{0, \frac{1}{2}, 1\}$, with $\frac{1}{2}$ interpreted as *unknown*.

An assignment ν maps each free variable to an element of $\text{adom}(D)$. Note that such an element could be a constant or a null; hence assignments are *not* valuations. We write $\nu[a/x]$ for the assignment that changes ν by mapping x to a. Also, given a tuple $\bar{x} = (x_1, \ldots, x_n)$ of free variables, and a tuple $\bar{a} = (a_1, \ldots, a_n)$, we write simply $\mathsf{Eval}(\varphi, D, \bar{a})$ if the assignment ν is such that $\nu(x_i) = a_i$ for all $i \leq n$.

Given an evaluation procedure Eval, the outcome of query evaluation for $\varphi(\bar{x})$ with $|\bar{x}| = k$ is

$$\mathsf{Eval}(\varphi, D) \;=\; \{\bar{a} \in \text{adom}(D)^k \mid \mathsf{Eval}(\varphi, D, \bar{a}) = 1\}\,.$$

As we said earlier, the goal is to make modifications to the standard evaluation absolutely minimal and still achieve correctness. In particular, we do not want to modify the completely standard evaluation of the Boolean connectives and quantifiers:

$$\begin{array}{rcl}
\mathsf{Eval}(\varphi \vee \psi, D, \nu) & = & \max(\mathsf{Eval}(\varphi, D, \nu), \mathsf{Eval}(\psi, D, \nu)) \\
\mathsf{Eval}(\varphi \wedge \psi, D, \nu) & = & \min(\mathsf{Eval}(\varphi, D, \nu), \mathsf{Eval}(\psi, D, \nu)) \\
\mathsf{Eval}(\neg\varphi, D, \nu) & = & 1 - \mathsf{Eval}(\varphi, D, \nu) \\
\mathsf{Eval}(\exists x\varphi, D, \nu) & = & \max\{\mathsf{Eval}(\varphi, D, \nu[a/x]) \mid a \in \mathbf{adom}(D)\} \\
\mathsf{Eval}(\forall x\varphi, D, \nu) & = & \min\{\mathsf{Eval}(\varphi, D, \nu[a/x]) \mid a \in \mathbf{adom}(D)\}
\end{array} \tag{5}$$

Thus, from now we only explain the valuation of *atomic* formulae $R(\bar{x})$ and equalities $x = y$. The classical FO evaluation gives us the procedure $\mathsf{Eval}_{\mathsf{FO}}$ with the range $\{0, 1\}$ defined by (5) and:

$$\mathsf{Eval}_{\mathsf{FO}}(R(\bar{x}), D, \nu) \;=\; \begin{cases} 1 & \text{if } \nu(\bar{x}) \in R^D \\ 0 & \text{if } \nu(\bar{x}) \notin R^D \end{cases}$$

$$\mathsf{Eval}_{\mathsf{FO}}(x = y, D, \nu) \;=\; \begin{cases} 1 & \text{if } \nu(x) = \nu(y) \\ 0 & \text{if } \nu(x) \neq \nu(y) \end{cases}$$

SQL's evaluation has $\{0, \frac{1}{2}, 1\}$ as the range of values. Again it uses rules (5), and the rule for $\mathsf{Eval}_{\mathsf{SQL}}(R(\bar{x}), D, \nu)$ is exactly the same as for $\mathsf{Eval}_{\mathsf{FO}}$, but for equality atoms the rule differs:

$$\mathsf{Eval}_{\mathsf{SQL}}(x = y, D, \nu) \;=\; \begin{cases} 1 & \text{if } \nu(x) = \nu(y) \text{ and } \nu(x), \nu(y) \in \mathsf{Const} \\ 0 & \text{if } \nu(x) \neq \nu(y) \text{ and } \nu(x), \nu(y) \in \mathsf{Const} \\ \frac{1}{2} & \text{if } \nu(x) \in \mathsf{Null} \text{ or } \nu(y) \in \mathsf{Null} \end{cases}$$

Indeed, SQL's approach is to declare every comparison as *unknown* if a null is involved. Note that over complete databases, $\mathsf{Eval}_{\mathsf{FO}}$ and $\mathsf{Eval}_{\mathsf{SQL}}$ coincide. Over incomplete databases, $\mathsf{Eval}_{\mathsf{FO}}$ is usually referred to as naïve evaluation [Abiteboul et al. 1995; Imielinski and Lipski 1984].

How do these relate to certain answers? We now examine FO and SQL evaluation. But first note that the definition (3) ensures that only tuples of constants are present in certain answers. There is no such restriction on the standard evaluation procedures. So to do a fair comparison we only compare sets of constant tuples returned by evaluation procedures (this will be relaxed later in the paper, when we provide a new definition of certain answers with nulls).

Definition 3.1. *Given a class $\mathcal{Q}$ of queries, an evaluation procedure* Eval *has* certainty guarantees *for $\mathcal{Q}$ if for every query $\varphi(\bar{x}) \in \mathcal{Q}$, every database D, and every tuple $\bar{a}$ of constants with $|\bar{a}| = |\bar{x}|$, we have*

$$\bar{a} \in \mathsf{Eval}(\varphi, D) \;\;\Rightarrow\;\; \bar{a} \in \mathit{cert}(\varphi, D).$$

In other words,

$$\mathsf{Eval}(\varphi, D) \cap \mathsf{Const}^{|\bar{x}|} \;\subseteq\; \mathit{cert}(\varphi, D).$$

Certain answers and $\mathsf{Eval}_{\mathsf{FO}}$

The first observation is immediate:

$$\mathbf{cert}(\varphi, D) \;\subseteq\; \mathsf{Eval}_{\mathsf{FO}}(\varphi, D).$$

The converse in general is not true, we can have $\mathsf{Eval}_{\mathsf{FO}}(\varphi(\bar{x}), D) \cap \mathsf{Const}^{|\bar{x}|} \not\subseteq \mathsf{cert}(\varphi, D)$. Consider for instance $\varphi(x) = R(x) \wedge \neg S(x)$ expressing the difference of R and S. Let D contain $R^D = \{1\}$ and $S^D = \{\bot\}$; then $\mathsf{Eval}(\varphi, D) = \{1\}$ while $\mathsf{cert}(\varphi, D) = \emptyset$.

However, sometimes certainty guarantees can be established. It has long been known [Imielinski and Lipski 1984] that we get them by excluding universal quantification and negation from first-order logic: $\mathsf{Eval}_{\mathsf{FO}}$ has certainty guarantees for the class UCQ. This was recently extended in [Gheerbrant et al. 2014] which showed that the same is true for queries from a rather significant expansion of the class UCQ, by adding universal quantification and a limited form of implication. More precisely, we look at the class $\mathcal{Q}^{\mathsf{cert}}_{\mathsf{FO}}$ defined as follows:

— atomic formulae $R(\bar{x})$ and $x = y$ are in $\mathcal{Q}^{\mathsf{cert}}_{\mathsf{FO}}$;
— if $\varphi, \psi \in \mathcal{Q}^{\mathsf{cert}}_{\mathsf{FO}}$ then so are $\varphi \vee \psi$ and $\varphi \wedge \psi$;
— if $\varphi \in \mathcal{Q}^{\mathsf{cert}}_{\mathsf{FO}}$ then so are $\exists x \varphi$ and $\forall x \varphi$;
— if $\varphi(\bar{x}, \bar{y})$ is in $\mathcal{Q}^{\mathsf{cert}}_{\mathsf{FO}}$, then so is $\forall \bar{x}\ (R(\bar{x}) \rightarrow \varphi(\bar{x}, \bar{y}))$, where R is a relation symbol in the schema, and $\bar{x}$ does not have a repetition of variables.

This class extends unions of conjunctive queries with a rather common class of queries involving negation and/or universal conditions, such as 'find students that take all courses'. The $\mathsf{Eval}_{\mathsf{FO}}$ procedure has certainty guarantees for $\mathcal{Q}^{\mathsf{cert}}_{\mathsf{FO}}$ queries [Gheerbrant et al. 2014]. From the point of view of relational algebra, the class $\mathcal{Q}^{\mathsf{cert}}_{\mathsf{FO}}$ corresponds to operations $\sigma, \pi, \cup, \times$ and the division operation $Q \div Q'$, where Q' is written in the $\pi, \cup, \times$-fragment of relational algebra, see [Libkin 2014b].

Certain answers and $\mathsf{Eval}_{\mathsf{SQL}}$

How does SQL change things? Actually, it changes them for the *worse*: now there is no connection between $\mathsf{Eval}_{\mathsf{SQL}}(\varphi, D)$ and $\mathsf{cert}(\varphi, D)$ whatsoever. Indeed, we saw that for the query $\varphi(x) = R(x) \wedge \neg(R(x) \wedge \neg S(x))$ and database D with $R^D = \{1\}$ and $S^D = \{\bot\}$, the certain answer is empty while $\mathsf{Eval}_{\mathsf{SQL}}(\varphi, D) = \{1\}$, and for the relation $T = \{(1, \bot)\}$ and the query $\psi(x) = \exists y\ (T(x, y) \wedge (x = 1 \vee \neg(x = 1)))$, the certain answer is $\{1\}$, while $\mathsf{Eval}_{\mathsf{SQL}}$ produces the empty set.

In a restricted case we provide correctness guarantees:

PROPOSITION 3.2. $\mathsf{Eval}_{\mathsf{SQL}}$ *has certainty guarantees for unions of conjunctive queries.*

Proof. The result is an immediate consequence of the following observation. If $\varphi(\bar{x})$ is a union of conjunctive queries (i.e., an existential positive formula), then

$$\mathsf{Eval}_{\mathsf{SQL}}(\varphi, D, \nu) = 1 \ \Rightarrow \ \mathsf{Eval}_{\mathsf{FO}}(\varphi, D, \nu) = 1. \tag{6}$$

Indeed, this implies the result since $\mathsf{Eval}_{\mathsf{FO}}$ has certainty guarantees for unions of conjunctive queries [Imielinski and Lipski 1984]. To show (6), we note that it trivially holds for relational atoms (since the rules are the same) and for equality atoms, by definition. Then a straightforward induction shows that (6) propagates through the rules for $\vee, \wedge$, and $\exists$. □

4. EVALUATION PROCEDURES WITH CERTAINTY GUARANTEES

We now introduce an evaluation procedure that comes with certainty guarantees for *all* relational calculus queries. For that, we have to explain what is wrong with the FO and SQL evaluation procedures shown above, particularly for the evaluation of atomic formulae.

Problems with atomic relational formulae $R(\bar{x})$. For both SQL and FO, one simply checks, for a given assignment ν, whether $\nu(\bar{x})$ belongs to R. However, returning 0 if $\nu(\bar{x}) \notin R$ is too strong if we view 0 as saying that the tuple certainly *cannot* belong to relation R.

Indeed, consider $R = \{(\bot_1, 1), (2, \bot_2)\}$ and let ν be the identity (recall that the range of ν is the whole active domain). Consider a tuple $\bar{x} = (\bot_1, \bot_2)$. It is not in R, but can it be in R under some valuation h? Of course it can: if $h(\bot_1) = 2$ and $h(\bot_2) = 1$, then $h(\bar{x}) = (2, 1)$ and $h(R) = \{(2, 1)\}$, i.e., $h(\bar{x}) \in h(R)$. On the other hand, if $h'(\bot_1) = 1$ and $h'(\bot_2) = 2$, then $h'(\bar{x}) = (1, 2)$ and $h'(R) = \{(1, 1), (2, 2)\}$, so $h'(\bar{x}) \notin h'(R)$. Thus, the correct value for evaluating the membership of $\bar{x}$ in R seems to be $\frac{1}{2}$, not 0. Value 0 should be reserved for cases when no valuation h makes $h(\bar{x}) \in h(R)$ possible.

The $\mathsf{Eval}_{\mathsf{FO}}$ and $\mathsf{Eval}_{\mathsf{SQL}}$ procedures return 0 too eagerly, and this becomes a problem when negation is applied to a formula, as 0 becomes a 1, and suddenly we have a false positive answer that, in fact, is not certain at all. If the value is kept at $\frac{1}{2}$, applying negation still results in $1 - \frac{1}{2} = \frac{1}{2}$, and thus no false 'certain answers' appear.

Problems with equality formulae $x = y$. FO evaluation results in 0 if $\nu(x)$ and $\nu(y)$ are different nulls, but they could still be mapped to the same constant, so the right value should be $\frac{1}{2}$, not 0. On the other hand, SQL evaluation produces $\frac{1}{2}$ if one of $\nu(x)$ or $\nu(y)$ is a null. But if we know $\nu(x) = \nu(y)$, then for every valuation h we will have $h(\nu(x)) = h(\nu(y))$, so the evaluation procedure must return 1 and not $\frac{1}{2}$ in this case, or else it will miss some certain answers.

Now with this in mind, we introduce a proper *3-valued evaluation procedure* $\mathsf{Eval}_{3\mathsf{v}}$. For this, we need one additional concept. Given two tuples $\bar{t}$ and $\bar{s}$ of the same length over $\mathsf{Const} \cup \mathsf{Null}$, we say that they *unify* if there is a homomorphism h such that $h(\bar{t}) = h(\bar{s})$. We then write $\bar{t} \Uparrow \bar{s}$.

For instance, tuples $\bar{t} = (1, 2, \bot, 2)$ and $\bar{s} = (1, \bot', 3, \bot'')$ unify, since for the map h that sends $\bot$ to 3 and both $\bot'$ and $\bot''$ to 2 we have $h(\bar{t}) = h(\bar{s})$. On the other hand, $\bar{t}$ and $\bar{u} = (\bot', \bot', 3, \bot'')$ do not unify, since the first two null components of $\bar{u}$ are the same, but the first two constant components of $\bar{t}$ are different.

It is easy to see that we can define $\bar{t} \Uparrow \bar{s}$ by asking for a valuation h so that $h(\bar{t}) = h(\bar{s})$. By classical results on unification, it is known that $\bar{t} \Uparrow \bar{s}$ can be tested in linear time [Paterson and Wegman 1978].

The evaluation procedure is as follows. It uses rules (5) and the following rules for atomic formulae, that overcome the issues outlined in the beginning of the section:

$$\mathsf{Eval}_{3\mathsf{v}}(R(\bar{x}), D, \nu) \;=\; \begin{cases} 1 & \text{if } \nu(\bar{x}) \in R^D\,, \\ 0 & \text{if there is no } \bar{t} \in R^D \text{ such that } \nu(\bar{x}) \Uparrow \bar{t}\,, \\ \frac{1}{2} & \text{otherwise.} \end{cases}$$

$$\mathsf{Eval}_{3\mathsf{v}}(x = y, D, \nu) \;=\; \begin{cases} 1 & \text{if } \nu(x) = \nu(y)\,, \\ 0 & \text{if } \nu(x), \nu(y) \in \mathsf{Const} \text{ and } \nu(x) \neq \nu(y)\,, \\ \frac{1}{2} & \text{otherwise.} \end{cases}$$

Coming back to the example in the beginning of the section, if we have a database D with $R^D = \{(\bot_1, 1), (2, \bot_2)\}$ and $\nu : (x, y) \mapsto (\bot_1, \bot_2)$, then $\mathsf{Eval}_{3\mathsf{v}}(R(x, y), D, \nu) = \frac{1}{2}$. Indeed, even though $(\bot_1, \bot_2)$ is not in R^D, there are valuations h so that $h(\bot_1, \bot_2) \in h(R^D)$. On the other hand, no valuation h makes $(1, 2) \in h(R^D)$ possible, so for $\nu' : (x, y) \mapsto (1, 2)$ we have $\mathsf{Eval}_{3\mathsf{v}}(R(x, y), D, \nu') = 0$.

These modifications turn out to be sufficient to ensure certainty guarantees for all relational calculus queries.

THEOREM 4.1. Eval_{3v} *has certainty guarantees for all* FO *queries.*

As an example, consider again query (4), or $\varphi(x) = R(x) \wedge \neg(R(x) \wedge \neg S(x))$ over D with $R^D = \{1\}$ and $S^D = \{\bot\}$. It produced a false positive since $\mathsf{Eval}_{\mathsf{SQL}}(\varphi, D) = \{1\}$ but the certain answer is empty. But now we have $\mathsf{Eval}_{3v}(\varphi, D) = \emptyset$. Indeed, we had $\mathsf{Eval}_{\mathsf{SQL}}(R(x) \wedge \neg S(x), D, 1) = 0$, and thus $\mathsf{Eval}_{\mathsf{SQL}}(\varphi, D, 1) = 1$, but now $\mathsf{Eval}_{3v}(R(x) \wedge \neg S(x), D, 1) = \frac{1}{2}$ and hence $\mathsf{Eval}_{3v}(\varphi, D, 1) = \frac{1}{2}$.

As another remark, note that the result of Eval_{3v} need not be contained in the result of $\mathsf{Eval}_{\mathsf{SQL}}$, i.e., Eval_{3v} can produce results that SQL evaluation misses. For instance, given a database D with $R^D = \{(\bot, \bot)\}$ and a query $\psi = \exists x, y\ R(x, y) \wedge x = y$, one can easily check that $\mathsf{Eval}_{3v}(\psi, D, \nu) = 1$ (for the only possible valuation over a singleton active domain), while $\mathsf{Eval}_{\mathsf{SQL}}(\psi, D, \nu) = \frac{1}{2}$.

Theorem 4.1 will be a consequence of a more general result (Theorem 5.3), that does not restrict us to constant tuples. But for this we first need to define certain answers with nulls.

5. CERTAIN ANSWERS WITH NULLS

While the definition of certain answers (3) has been by far the most common in use, it has a number of deficiencies (see [Libkin 2014a; 2014b] for discussion). One of the problems is that it only returns tuples containing constants. Consider a database D with a relation $R^D = \{(1, 2), (3, \bot)\}$ and a query $\psi(x, y) = R(x, y)$. Then $\mathsf{cert}(\psi, D) = \{(1, 2)\}$ but intuitively we should return the entire relation R^D since we are certain its tuples are in the answer. The reason we are certain about it is that for every valuation h, the tuple $(3, h(\bot))$ is in $h(D)$.

These considerations led to a slight extension of the notion of certain answers [Lipski 1984] that, despite being very natural, is not as commonly used as the definition (3). Note that certain answers (3) can be defined as the set of constant tuples $\bar{u}$ (i.e., tuples without nulls) such that for all valuations h, the tuple $\bar{u} = h(\bar{u})$ is in $Q(h(D))$. The extended definition of [Lipski 1984] simply removes the condition that query answers should not contain nulls.

Definition 5.1. Given an incomplete database D and a k-ary query Q defined over complete databases, *certain answers with nulls* $\mathsf{cert}_\bot(Q, D)$ is defined as the set of all tuples $\bar{u} \in \mathsf{adom}(D)^k$ such that $h(\bar{u}) \in Q(h(D))$ for all valuations h.

For instance, if a query is given by an FO formula with k free variables, then

$$\mathsf{cert}_\bot(\varphi, D) \;=\; \{\bar{u} \in \mathsf{adom}(D)^k \;\mid\; \mathsf{Eval}_{\mathsf{FO}}(\varphi, h(D), h(\bar{u})) = 1 \text{ for every valuation } h\}.$$

Returning to the above example, we have $\mathsf{cert}_\bot(\psi, D) = \{(1, 2), (3, \bot)\}$, so the tuple $(3, \bot)$ is no longer omitted. We remark that [Lipski 1984] did not actually provide a name for this concept, and it appeared under other names, e.g., sure uni-answer in [Klein 1999]. We prefer the term 'certain answers with nulls' as it conveys the nature of the concept.

We now summarize properties of certain answers with nulls. The usual certain answers can be obtained from certain answers with nulls by dropping tuples containing nulls, and certain answers with nulls are always contained in the result of the simple FO evaluation of formulae. Sometimes, but not always, they may coincide with the result of such an evaluation.

Formally, we have the following.

PROPOSITION 5.2. The following hold:

— $\mathsf{cert}(\varphi(\bar{x}), D) = \mathsf{cert}_\bot(\varphi, D) \cap \mathsf{Const}^{|\bar{x}|}$.

— $\mathrm{cert}_\perp(\varphi, D) \subseteq \mathsf{Eval}_{\mathsf{FO}}(\varphi, D)$ for every FO query φ.
— If $\varphi \in \mathcal{Q}^{\mathrm{cert}}_{\mathsf{FO}}$, then $\mathrm{cert}_\perp(\varphi, D) = \mathsf{Eval}_{\mathsf{FO}}(\varphi, D)$.
— There exist FO queries φ so that $\mathrm{cert}_\perp(\varphi, D) \neq \mathsf{Eval}_{\mathsf{FO}}(\varphi, D)$.

Proof. We first show a useful auxiliary result that $\bar{u} \in \mathrm{cert}_\perp(\varphi, D)$ iff $\mathsf{Eval}_{\mathsf{FO}}(\varphi, h(D), h(\bar{u})) = 1$ for every homomorphism h (rather than every evaluation h). To see this, we show that the following conditions are equivalent: $\mathsf{Eval}_{\mathsf{FO}}(\varphi, h(D), h(\bar{u})) = 1$ for each homomorphism h, and $\mathsf{Eval}_{\mathsf{FO}}(\varphi, g(D), g(\bar{u})) = 1$ for each valuation g are equivalent.

Of course if the statement $\mathsf{Eval}_{\mathsf{FO}}(\varphi, h(D), h(\bar{u})) = 1$ is true for every homomorphism, then it also true for every valuation. Conversely, assume that $\mathsf{Eval}_{\mathsf{FO}}(\varphi, g(D), g(\bar{u})) = 1$ holds for every valuation g. Let h be a homomorphism defined on $\mathsf{Null}(D)$. Let N_0 be the set of nulls in the image of h, and let $N = N_0 \cup \mathsf{Null}(D)$. Consider a set C of constants that do not occur in $\mathrm{adom}(D)$, nor in the image of h, so that $|C| = |N|$, and let $f : N \to C$ be a bijection. Define a valuation g as $f \circ h$. By the assumption, we have $\mathsf{Eval}_{\mathsf{FO}}(\varphi, g(D), g(\bar{u})) = 1$, and by the genericity of FO queries we have $\mathsf{Eval}_{\mathsf{FO}}(\varphi, f^{-1}(g(D)), f^{-1}(g(\bar{u}))) = 1$. Since $g \circ f^{-1} = h$, this means $\mathsf{Eval}_{\mathsf{FO}}(\varphi, h(D), h(\bar{u})) = 1$ as required. This proves our claim.

We now prove all the items of the proposition. Given $\varphi(\bar{x})$, if $\bar{a} \in \mathrm{cert}(\varphi, D)$, then $\bar{a}$ is a tuple of constants and $\bar{a} \in \mathrm{cert}_\perp(\varphi, D)$ and thus $\mathrm{cert}(\varphi, D) \subseteq \mathrm{cert}_\perp(\varphi, D) \cap \mathsf{Const}^{|\bar{x}|}$. Conversely, if $\bar{a}$ is a tuple of constants in $\mathrm{cert}_\perp(\varphi, D)$, then for every valuation h we have $h(\bar{a}) = \bar{a}$ and thus $\mathsf{Eval}_{\mathsf{FO}}(\varphi, h(D), \bar{a}) = 1$, i.e., $\bar{a} \in \varphi(h(D))$, implying $\bar{a} \in \mathrm{cert}(\varphi, D)$. Hence, $\mathrm{cert}_\perp(\varphi, D) \cap \mathsf{Const}^{|\bar{x}|} \subseteq \mathrm{cert}(\varphi, D)$.

For the second item, let $\bar{u} \in \mathrm{cert}_\perp(\varphi, D)$. By the observation made earlier, for every homomorphism h we have $\mathsf{Eval}_{\mathsf{FO}}(\varphi, h(\bar{u}), h(D)) = 1$. Applying this to the identity homomorphism h we have $\bar{u} \in \mathsf{Eval}_{\mathsf{FO}}(\varphi, D)$.

For the third item, let φ belong to $\mathcal{Q}^{\mathrm{cert}}_{\mathsf{FO}}$. It is known that in this case φ is preserved under strong onto homomorphisms [Compton 1983; Gheerbrant et al. 2014]. That is, if $\mathsf{Eval}_{\mathsf{FO}}(\varphi, D, \bar{u}) = 1$ and h is a homomorphism, then $\mathsf{Eval}_{\mathsf{FO}}(\varphi, h(D), h(\bar{u})) = 1$. We already know that $\mathrm{cert}_\perp(\varphi, D) \subseteq \mathsf{Eval}_{\mathsf{FO}}(\varphi, D)$. So now conversely assume $\bar{u} \in \mathsf{Eval}_{\mathsf{FO}}(\varphi, D)$. Let h be an arbitrary homomorphism. By preservation, we have $\mathsf{Eval}_{\mathsf{FO}}(\varphi, h(D), h(\bar{u})) = 1$ which implies $\bar{u} \in \mathrm{cert}_\perp(\varphi, D)$.

Finally, consider a schema with two unary relations R and S, a database D with $R^D = \{1\}$ and $S^D = \{\perp\}$, and $\varphi(x) = R(x) \wedge \neg S(x)$. Then $\mathsf{Eval}_{\mathsf{FO}}(\varphi, D) = \{1\}$ but $1 \notin \mathrm{cert}_\perp(\varphi, D)$ as witnessed by the valuation $\perp \mapsto 1$. □

We can now state a more general description of the evaluation procedure $\mathsf{Eval}_{3\mathsf{v}}$: the output value 1 guarantees that a tuple belongs to certain answers with nulls for query φ, the output value 0 guarantees that it belongs to certain answers with nulls for the negation $\neg\varphi$, and output value $\frac{1}{2}$ comes with no guarantees.

THEOREM 5.3. *For every* FO *query* $\varphi(\bar{x})$ *and every database* D*,*

$$\mathsf{Eval}_{3\mathsf{v}}(\varphi, D) \subseteq \mathit{cert}_\perp(\varphi, D).$$

Moreover, if $\bar{a} \in \mathit{adom}(D)^{|\bar{x}|}$ *and* $\mathsf{Eval}_{3\mathsf{v}}(\varphi, D, \bar{a}) = 0$*, then* $\bar{a} \in \mathit{cert}_\perp(\neg\varphi, D)$*.*

Theorem 4.1 is now an immediate corollary: if $\bar{a}$ is a tuple of constants and $\mathsf{Eval}_{3\mathsf{v}}(\varphi, D, \bar{a}) = 1$, then by Theorem 5.3, $\bar{a} \in \mathrm{cert}_\perp(\varphi, D)$, and by Proposition 5.2, $\bar{a} \in \mathrm{cert}(\varphi, D)$.

Proof. Recall that we use the convention that instead of $\mathsf{Eval}(\varphi(\bar{x}), D, \nu)$ we may write $\mathsf{Eval}(\varphi, D, \bar{u})$ where $\bar{u} = \nu(\bar{x})$. To prove the theorem, in view of the observation made

at the beginning of the proof of Proposition 5.2 (which showed that we can use homomorphisms and valuations interchangeably), we need to show the following for each formula $\varphi(\bar{x})$ with $|\bar{x}| = k$ and each assignment ν:

$$\mathsf{Eval}_{3\mathsf{v}}(\varphi, D, \nu) = 1 \;\Rightarrow\; \forall \text{ homomorphism } h:\; \mathsf{Eval}_{\mathsf{FO}}(\varphi, h(D), h(\nu(\bar{x}))) = 1 \qquad (*)$$

$$\mathsf{Eval}_{3\mathsf{v}}(\varphi, D, \nu) = 0 \;\Rightarrow\; \forall \text{ homomorphism } h:\; \mathsf{Eval}_{\mathsf{FO}}(\varphi, h(D), h(\nu(\bar{x}))) = 0 \qquad (**)$$

We now do this by induction of FO formulae $\varphi(\bar{x})$.

Case 1: $\varphi(\bar{x})$ is a relational atom $R(\bar{x})$.

(*) If $\mathsf{Eval}_{3\mathsf{v}}(\varphi, D, \nu) = 1$ then $\nu(\bar{x}) \in R^D$; in particular, $h(\nu(\bar{x})) \in h(R^D)$ for every homomorphism h, showing $h(\nu(\bar{x})) \in \mathsf{Eval}_{\mathsf{FO}}(\varphi, h(D))$.

(**) If $\mathsf{Eval}_{3\mathsf{v}}(\varphi, D, \nu) = 0$ then for each tuple $\bar{t} \in R^D$ we have that $\nu(\bar{x}) \Uparrow \bar{t}$ does not hold. Thus for each homomorphism h, and each tuple $\bar{t} \in R^D$, we have $h(\nu(\bar{x})) \neq h(\bar{t})$. This means that $h(\bar{x}) \notin h(R^D)$, and thus for each homomorphism h we have $\mathsf{Eval}_{\mathsf{FO}}(\varphi, h(D), h(\nu(\bar{x}))) = 0$.

Case 2: $\varphi(x, y)$ is an equality atom $x = y$.

(*) If $\mathsf{Eval}_{3\mathsf{v}}(x = y, D, \nu) = 1$ then $\nu(x) = \nu(y)$, and thus for for every homomorphism h, we have $h(\nu(x)) = h(\nu(y))$; in particular, $\mathsf{Eval}_{\mathsf{FO}}(x = y, h(D), h \circ \nu) = 1$.

(**) If $\mathsf{Eval}_{3\mathsf{v}}(x = y, D, \nu) = 0$, then both $\nu(x)$ and $\nu(y)$ are constants and $\nu(x) \neq \nu(y)$. Since they are constants, every homomorphism leaves them intact, and thus $\mathsf{Eval}_{\mathsf{FO}}(x = y, h(D), h \circ \nu) = 0$.

Case 3: $\varphi(\bar{x})$ is $\varphi_1(\bar{x}_1) \wedge \varphi_2(\bar{x}_2)$ (where $\bar{x}$ contains all the variables in $\bar{x}_1$ and $\bar{x}_2$).

(*) If $\mathsf{Eval}_{3\mathsf{v}}(\varphi(\bar{x}), D, \nu) = 1$, then $\mathsf{Eval}_{3\mathsf{v}}(\varphi_i(\bar{x}_i), D, \nu) = 1$ for $i = 1, 2$, which, by the induction hypothesis, implies that for each homomorphism h, we have that $\mathsf{Eval}_{\mathsf{FO}}(\varphi_i(\bar{x}_i), h(D), h(\nu(\bar{x}))) = 1$ for $i = 1, 2$. This in turn implies that for every homomorphism h, the value of $\mathsf{Eval}_{\mathsf{FO}}(\varphi(\bar{x}), h(D), h(\nu(\bar{x})))$ is 1.

(**) If $\mathsf{Eval}_{3\mathsf{v}}(\varphi(\bar{x}), D, \nu) = 0$, then $\mathsf{Eval}_{3\mathsf{v}}(\varphi_i(\bar{x}_i), D, \nu) = 0$ for at least one of $i = 1, 2$; assume that $\mathsf{Eval}_{3\mathsf{v}}(\varphi_1(\bar{x}_1), D, \nu) = 0$ (the other case is, of course, analogous). By the induction hypothesis, this implies that for each homomorphism h, we have $\mathsf{Eval}_{\mathsf{FO}}(\varphi_1(\bar{x}_1), h(D), h(\nu(\bar{x}_1))) = 0$ and thus $\mathsf{Eval}_{\mathsf{FO}}(\varphi(\bar{x}), h(D), h(\nu(\bar{x}))) = 0$, as required.

Case 4: $\varphi(\bar{x}) = \varphi_1(\bar{x}_1) \vee \varphi_2(\bar{x}_2)$ is completely analogous.

Case 5: $\varphi(\bar{x})$ is $\neg\psi(\bar{x})$.

(*) If $\mathsf{Eval}_{3\mathsf{v}}(\varphi(\bar{x}), D, \nu) = 1$, then $\mathsf{Eval}_{3\mathsf{v}}(\psi(\bar{x}), D, \nu) = 0$, and by the induction hypothesis we have $\mathsf{Eval}_{\mathsf{FO}}(\psi, h(D), h(\nu(\bar{x}))) = 0$ for every homomorphism h. This, by FO rules, implies $\mathsf{Eval}_{\mathsf{FO}}(\varphi, h(D), h(\nu(\bar{x}))) = 1$ for every homomorphism h.

(**) is analogous to the proof for (*).

Case 6: $\varphi(\bar{x})$ is $\exists y\ \psi(\bar{x}, y)$.

(*) If $\mathsf{Eval}_{3\mathsf{v}}(\varphi(\bar{x}), D, \nu) = 1$, then we can find $u_0 \in \mathrm{adom}(D)$ such that the result of $\mathsf{Eval}_{3\mathsf{v}}(\psi(\bar{x}, y), D, \nu[u_0/y])$ is 1. Now take an arbitrary homomorphism h defined on $\mathsf{Null}(D)$. We know, by the induction hypothesis, that

$$\mathsf{Eval}_{\mathsf{FO}}(\psi(\bar{x}, y),\ h(D),\ \big(h(\nu(\bar{x})), h(u_0)\big)) \;=\; 1$$

and hence $\mathsf{Eval}_{\mathsf{FO}}(\varphi, h(D), h(\nu(\bar{x}))) = 1$.

(**) If $\mathsf{Eval}_{3\mathsf{v}}(\varphi(\bar{x}), D, \nu) = 0$, then for each $u_0 \in \mathrm{adom}(D)$ we know that the result of $\mathsf{Eval}_{3\mathsf{v}}(\psi(\bar{x}, y), D, \nu[u_0/y])$ is 0. Thus, by the induction hypothesis, for an arbitrary homomorphism h defined on $\mathsf{Null}(D)$ we have

$$\mathsf{Eval}_{\mathsf{FO}}(\psi(\bar{x}, y),\ h(D),\ \big(h(\nu(\bar{x})), h(u_0)\big)) \ = \ 0$$

and hence, since u_0 was chosen arbitrarily, we see that $\mathsf{Eval}_{\mathsf{FO}}(\varphi, h(D), h(\nu(\bar{x}))) = 0$ for each such homomorphism.

Case 7: $\varphi(\bar{x})$ is $\forall y\ \psi(\bar{x}, y)$ is analogous to Case 6 (or can be obtained by combining Cases 5 and 6).

This concludes the proof. □

So far we dealt only with under-approximations of certain answers, i.e., evaluation procedures without false positives. One can adopt an alternative point of view and try to eliminate false negatives. In that case, one would look for *over-approximations*, i.e., procedures that return all certain answers, and perhaps something else. It turns out that $\mathsf{Eval}_{3\mathsf{v}}$ can be used to give over-approximations, simply by taking the complement of the result on the negation of the query.

COROLLARY 5.4. *For every* FO *query* $\varphi(\bar{x})$ *we have*

$$\mathit{cert}_{\perp}(\varphi, D) \ \subseteq \ \mathit{adom}(D)^{|\bar{x}|} - \mathsf{Eval}_{3\mathsf{v}}(\neg\varphi, D).$$

The procedure $\mathsf{Eval}_{3\mathsf{v}}$ is faithful to the usual FO evaluation on complete databases: in this case, the values of $\mathsf{Eval}_{3\mathsf{v}}$ are only 0 and 1, and the value 1 is achieved exactly for tuples in query results. That is, we have the following.

PROPOSITION 5.5. *If D is a complete database, then the values of* $\mathsf{Eval}_{3\mathsf{v}}(\varphi, D, \nu)$ *are only* 0 *and* 1*, and* $\mathsf{Eval}_{3\mathsf{v}}(\varphi, D) = \mathsf{Eval}_{\mathsf{FO}}(\varphi, D)$ *for every* FO *query* φ.

Proof. The first statement is immediate by the inspection of the rules: only the presence of nulls introduces the truth value $\frac{1}{2}$. Since for databases D and assignments ν over Const the rules of $\mathsf{Eval}_{3\mathsf{v}}$ for atomic formulae $R(\bar{x})$ and $x = y$ become identical to those of $\mathsf{Eval}_{\mathsf{FO}}$ (again, as seen by a simple inspection of them), the result follows. □

As for the complexity of the procedure, one can easily show the following.

PROPOSITION 5.6. *For each relational vocabulary σ and $\alpha \in \{0, \frac{1}{2}, 1\}$, from every* FO *query $\varphi(\bar{x})$ one can compute* FO *queries $\varphi^{\alpha}(\bar{x})$ in the vocabulary that extends σ with a unary predicate* $\mathsf{const}(\cdot)$ *interpreted as the set of constants, such that, for every database D,*

$$\{\bar{a} \in \mathit{adom}(D)^{|\bar{x}|} \mid \mathsf{Eval}_{3\mathsf{v}}(\varphi, D, \bar{a}) = \alpha\} \ \ = \ \ \mathsf{Eval}_{\mathsf{FO}}(\varphi^{\alpha}, D).$$

Consequently, data complexity of computing $\mathsf{Eval}_{3\mathsf{v}}(\varphi, D)$ *is in* AC^0.

Proof. The proof is straightforward in all the cases except one as, by simple inspection of the rules for $\mathsf{Eval}_{3\mathsf{v}}$, one sees that each case is definable in FO. The only case requiring proof is that of $\mathsf{Eval}_{3\mathsf{v}}$ returning 0 for relational atoms. We must show that, for a relation symbol R of arity k, there is a formula $\varphi(\bar{x})$ with $|\bar{x}| = k$ so that $\varphi(\bar{a})$ is true in R^D iff $\bar{a} \Uparrow \bar{t}$ does not hold for every tuple $\bar{t}$ in R^D. This in turn follows since one can express, in the vocabulary expanded with const, a formula $\psi_{\Uparrow}(\bar{x}, \bar{y})$ so that $\psi_{\Uparrow}(\bar{a}, \bar{b})$ is true iff $\bar{a} \Uparrow \bar{b}$; then $\varphi(\bar{x})$ is simply $\forall \bar{y}\ R(\bar{y}) \rightarrow \neg\psi_{\Uparrow}(\bar{x}, \bar{y})$.

To show that $\psi_{\Uparrow}$ is expressible, we define patterns of pairs of k-tuples as pairs $\pi = (\bar{u}, E)$ where $\bar{u}$ is a Boolean tuple of length $2k$, and E is an equivalence relation on the set $\{1, \ldots, 2k\}$. Given two tuples $\bar{a} = (a_1, \ldots, a_k)$ and $\bar{b} = (b_1, \ldots, b_k)$ over $\mathsf{Const} \cup \mathsf{Null}$, they conform to π if in the $2k$-tuple $(c_1, \ldots, c_{2k})$ obtained by concatenating $\bar{a}$ and $\bar{b}$, the following is true:

— the ith position in $\bar{u}$ is 1 iff c_i is a constant; and
— $c_i = c_j$ iff $(i, j) \in E$.

Note that for each pattern π, either all pairs conforming to it unify, or none unifies. Hence, we can express $\psi_{\Uparrow}$ as the disjunction over all patterns π guaranteeing unification of formulae stating that $\bar{x}, \bar{y}$ conform to π. The latter can be straightforwardly expressed in FO. Since the number of patterns depends only on k, the formula $\psi_{\Uparrow}$ is expressible in FO. □

This gives us a complexity argument showing that there are cases when Eval_{3v} fails to produce *all* certain answers. A concrete example of strict containment of Eval_{3v} in $\mathrm{cert}_\perp$ will be shown below in Section 5.1.

5.1. CQs and UCQs with inequalities

A common extension of conjunctive queries and their unions is by adding inequalities [Abiteboul et al. 1995]. This is a very mild form of negation; essentially, we only allow negation to be applied to equality atoms. Instead of writing them as $\neg(x = y)$, it is common to use $x \neq y$ in formulae, and refer to them as inequality atoms. Then the $\exists, \wedge$-closure of relational, equality and inequality atoms is referred to as CQs with inequalities, and the $\exists, \wedge, \vee$-closure as *UCQs with inequalities*. This class of queries is denoted by $\mathsf{UCQ}^{\neq}$.

We now present a particularly easy evaluation procedure that correctly accounts for Eval_{3v} producing value 1 for UCQs with inequalities, and thus gives us correctness guarantees for those queries. This procedure uses two-valued, rather than three-valued, logic and only one rule that separates it from $\mathsf{Eval}_{\mathsf{FO}}$. To understand it, note for an inequality atom $x \neq y$, FO evaluation returns true if x and y are assigned different values – even if they are different nulls. But actually the evaluation of conditions such as $\perp_1 \neq \perp_2$ must be false, since $\perp_1$ and $\perp_2$ can be mapped, by a valuation, to the same element. For $\mathsf{UCQ}^{\neq}$, there is no risk with assigning *false* rather than *unknown*, since negation will never be applied further on. This lets us define the evaluation procedure for $\mathsf{UCQ}^{\neq}$ by adding the following explicit rule for $\neq$ formulae to the $\mathsf{Eval}_{\mathsf{FO}}$ rules:

$$\mathsf{Eval}_{\mathsf{UCQ}^{\neq}}(x \neq y, D, \nu) \;=\; \begin{cases} 1 & \text{if } \nu(x), \nu(y) \in \mathsf{Const} \text{ and } \nu(x) \neq \nu(y) \\ 0 & \text{otherwise} \end{cases}$$

This evaluation is particularly easy to implement in SQL with the usual `is not null` conditions in the `where` clause. And it has the desired correctness guarantees.

THEOREM 5.7. *For every* $\mathsf{UCQ}^{\neq}$ *query* φ*, we have*

$$\mathsf{Eval}_{\mathsf{UCQ}^{\neq}}(\varphi, D) \;=\; \mathsf{Eval}_{3v}(\varphi, D) \;\subseteq\; \mathit{cert}_\perp(\varphi, D).$$

In particular, $\mathsf{Eval}_{\mathsf{UCQ}^{\neq}}$ *has certainty guarantees for* $\mathsf{UCQ}^{\neq}$ *queries.*

Proof. We will show that for a $\mathsf{UCQ}^{\neq}$ query φ, we have

$$\mathsf{Eval}_{\mathsf{UCQ}^{\neq}}(\varphi, D, \nu) = 1 \;\;\Leftrightarrow\;\; \mathsf{Eval}_{3v}(\varphi, D, \nu) = 1 \tag{7}$$

for every database D and assignment ν.

($\Rightarrow$) Since a $\mathsf{UCQ}^{\neq}$ query φ is a disjunction of CQs with inequalities $\varphi_1 \vee \ldots \vee \varphi_m$, it suffices to prove (7) for CQs: if $\mathsf{Eval}_{\mathsf{UCQ}^{\neq}}(\varphi, D, \nu) = 1$, then $\mathsf{Eval}_{\mathsf{UCQ}^{\neq}}(\varphi_i, D, \nu) = 1$ for some $i \leq m$, and thus $\mathsf{Eval}_{3v}(\varphi_i, D, \nu) = 1$ would imply $\mathsf{Eval}_{3v}(\varphi, D, \nu) = 1$. We show this by induction on the formula.

— If φ is an atomic formula $R(\bar{x})$, then $\mathsf{Eval}_{\mathsf{UCQ}^{\neq}}(R(\bar{x}), D, \nu) = 1$ means that $\nu(\bar{x}) \in R^D$ and hence $\mathsf{Eval}_{\mathsf{3v}}(R(\bar{x}), D, \nu) = 1$.
— If φ is an equality atom $x = y$, then $\mathsf{Eval}_{\mathsf{UCQ}^{\neq}}(x = y, D, \nu) = 1$ means that $\nu(x) = \nu(y)$ and hence $\mathsf{Eval}_{\mathsf{3v}}(x = y, D, \nu) = 1$.
— If φ is $x \neq y$, then $\mathsf{Eval}_{\mathsf{UCQ}^{\neq}}(x = y, D, \nu) = 1$ means that $\nu(x) \neq \nu(y)$ and both $\nu(x)$ and $\nu(y)$ are constants. Thus $\mathsf{Eval}_{\mathsf{3v}}(x = y, D, \nu) = 0$ and $\mathsf{Eval}_{\mathsf{3v}}(\neg(x = y), D, \varphi) = 1$.
— If $\varphi = \varphi_1 \wedge \varphi_2$ and $\mathsf{Eval}_{\mathsf{UCQ}^{\neq}}(\varphi, D, \nu) = 1$, then $\mathsf{Eval}_{\mathsf{UCQ}^{\neq}}(\varphi_i, D, \nu) = 1$ for $i = 1, 2$, and the induction hypothesis implies $\mathsf{Eval}_{\mathsf{3v}}(\varphi, D, \nu) = 1$.
— The case of $\varphi = \exists x \psi$ similarly immediately follows by induction.

($\Leftarrow$) As for the ($\Rightarrow$) case, it suffices to provide the proof for CQs with inequalities. Assume that $\varphi(\bar{x}) = \exists \bar{y}\ \psi(\bar{x}, \bar{y})$, where

$$\psi(\bar{x}, \bar{y}) \ = \ \bigwedge_i R_i(\bar{u}_i) \ \wedge \ \bigwedge_j (z_j = z'_j) \ \wedge \ \bigwedge_k (w_k \neq w'_k), \tag{8}$$

and the variables in the $\bar{u}_i$s, as well as the z_j, z'_j, w_k, w'_ks are all among $\bar{x}, \bar{y}$ and all the R_is are relation symbols in the schema. If $\mathsf{Eval}_{\mathsf{3v}}(\varphi, D, \nu) = 1$, then for some assignment ν' extending ν to variables $\bar{y}$ we have $\mathsf{Eval}_{\mathsf{3v}}(\psi, D, \nu') = 1$, i.e., each conjunct in (8) evaluates to 1 under $\mathsf{Eval}_{\mathsf{3v}}$. We now look at those conjuncts.

— If $\mathsf{Eval}_{\mathsf{3v}}(R_i(\bar{u}_i), D, \nu') = 1$, then $\nu'(\bar{u}_i) \in R_i^D$ and thus $\mathsf{Eval}_{\mathsf{UCQ}^{\neq}}(R_i(\bar{u}_i), D, \nu') = 1$.
— If $\mathsf{Eval}_{\mathsf{3v}}(z_j = z'_j, D, \nu') = 1$, then $\nu'(z_j) = \nu'(z'_j)$ and thus $\mathsf{Eval}_{\mathsf{UCQ}^{\neq}}(z_j = z'_j, D, \nu') = 1$.
— If $\mathsf{Eval}_{\mathsf{3v}}(w_k \neq w'_k, D, \nu') = 1$, then $\mathsf{Eval}_{\mathsf{3v}}(w_k = w'_k, D, \nu') = 0$ and thus $\nu'(w_k) \neq \nu'(w'_k)$ and both $\nu'(w_k)$ and $\nu'(w'_k)$ are constants. This implies $\mathsf{Eval}_{\mathsf{UCQ}^{\neq}}(w_k \neq w'_k, D, \nu') = 1$.

Hence, each conjunct evaluates to 1 under $\mathsf{Eval}_{\mathsf{UCQ}^{\neq}}$, and therefore $\mathsf{Eval}_{\mathsf{UCQ}^{\neq}}(\psi, D, \nu') = 1$. Hence $\mathsf{Eval}_{\mathsf{3v}}(\varphi, D, \nu) = 1$ by using the rule for existential quantification. This completes the proof of ($\Leftarrow$). □

One cannot capture $\mathrm{cert}_{\perp}(\varphi, D)$ precisely with the $\mathsf{UCQ}^{\neq}$ evaluation procedure. Indeed, consider the query $\psi = \exists x \exists y\ R(x, y) \wedge x \neq y$ and a database D with $R^D = \{(\perp, 1), (\perp, 2)\}$. One easily checks $\mathrm{cert}_{\perp}(\psi, D) = \mathrm{cert}(\psi, D) = \text{true}$ but at the same time $\mathsf{Eval}_{\mathsf{UCQ}^{\neq}}(\psi, D) = 0$. By Theorem 5.7, this also means that $\mathsf{Eval}_{\mathsf{3v}}(\psi, D)$ fails to capture $\mathrm{cert}_{\perp}(\varphi, D)$; this is the example promised at the end of the last section.

In fact there could be no polynomial-time evaluation procedure for finding certain answers for $\mathsf{UCQ}^{\neq}$ queries since they have coNP-complete data complexity, even without free variables. Indeed, suppose we have a graph $G = \langle V, E \rangle$ where the set of vertices is $\{a_1, \ldots, a_n\}$. Create a binary relation D_G with $\mathrm{adom}(D_G) = \{\perp_1, \ldots, \perp_n\}$ and pairs $(\perp_i, \perp_j)$ for every edge $(a_i, a_j) \in E$. Let $\varphi \in \mathsf{UCQ}^{\neq}$ be given by $\exists x\ D_G(x, x) \vee \exists x, y, z, u\ (x \neq y \wedge x \neq z \wedge x \neq u \wedge y \neq z \wedge y \neq u \wedge z \neq u)$. Then $\mathrm{cert}(\varphi, D_G)$ is true iff G is not 3-colorable.

5.2. Open world semantics

Another semantics of incompleteness is based on the *open-world assumption*, or OWA [Abiteboul et al. 1995; Imielinski and Lipski 1984; Reiter 1977]. Under this assumption, an incomplete database can be extended with extra tuples. This semantics is prominent in several actively studied applications of incompleteness. It is the basic semantics in ontology based query answering, when a database comes with an ontology, and query answering is defined as certain answers with respect to all extensions of the database that satisfy the ontology constraints [Calvanese et al. 2007; Calì et al. 2012]. In other applications, such as data integration and exchange, both OWA and CWA semantics are used [Abiteboul and Duschka 1998; Arenas et al. 2014; Lenzerini 2002].

We now explain how our approach can be extended to OWA. Note that due to the higher complexity of query answering under OWA (to be explained shortly), it is harder to get good approximations.

The OWA semantics state that after applying a valuation h to a database, finitely many complete tuples can be added to it. That is,

$$[\![D]\!]_{\text{OWA}} \;=\; \{h(D) \cup D' \mid h \text{ is a valuation and } D' \text{ is complete}\}.$$

Certain answers under OWA are defined as $\text{cert}_{\text{OWA}}(Q, D) = \bigcap\{Q(D') \mid D' \in [\![D]\!]_{\text{OWA}}\}$. The evaluation procedure $\mathsf{Eval}_{3\text{v}}$ no longer has certainty guarantees under OWA. To see this, consider D with relations $R^D = \{(1, 2)\}$ and $S^D = \{(\bot_1, 1), (2, \bot_2)\}$. Let $\varphi(x, y) = R(x, y) \wedge \neg S(x, y)$. Since the tuple $(1, 2)$ does not unify with either tuple in S^D, we have $(1, 2) \in \mathsf{Eval}_{3\text{v}}(\varphi, D)$. However, under OWA, it is not a certain answer: for instance, the database D' with $R^{D'} = \{(1, 2)\}$ and $S^{D'} = \{(1, 1), (2, 2), (1, 2)\}$ is in $[\![D]\!]_{\text{OWA}}$, and $\mathsf{Eval}_{\mathsf{FO}}(\varphi, D')$ is empty.

Thus, our question is whether the approach of $\mathsf{Eval}_{3\text{v}}$, guaranteeing correctness for all FO queries under CWA, can be extended to OWA. Of course there is always a trivial positive answer: the evaluation procedure that always returns 0 vacuously has correctness guarantees. Since $[\![D]\!]_{\text{CWA}} \subseteq [\![D]\!]_{\text{OWA}}$, certain answers under OWA will be included in certain answers under CWA, so the question really is how much we eliminate from the latter so that the result is still meaningful, and provides certainty guarantees under OWA. Note also that finding certain answers under OWA is undecidable [Abiteboul et al. 1991] (even for data complexity [Gheerbrant et al. 2012]) which ties our hands even more in terms of finding suitable approximations.

To understand the changes that need to be made under OWA, consider again relational atoms. For them, there is no way to assert with certainty that a tuple does not belong to a relation, since each relation can be expanded under OWA. Hence, the case when evaluation produces 0 must go.

Next, look at existential formulae. Again we cannot state with certainty that the result of evaluation of those is 0, as perhaps in some extension of the database there is a witness for the existential formula, so the lowest value for evaluating such a formula is $\frac{1}{2}$, not 0. Likewise, for universal formulae, one cannot state with certainty that the result of evaluation is 1, as it requires checking the universal conditions in all extensions of the database, which is an undecidable problem. Hence, the highest value in this case is $\frac{1}{2}$ and not 1.

This explains the three changes that we make for the evaluation procedure. In particular, to achieve correctness, evaluation for quantifiers cannot follow the standard rules (5). Indeed, this simply reflects the fact that under OWA, quantification is *not* equivalent to conjunction or disjunction over all elements of the active domain.

The procedure $\mathsf{Eval}_{3\text{v}}^{\text{OWA}}$ has the range $\{0, \frac{1}{2}, 1\}$ and differs from $\mathsf{Eval}_{3\text{v}}$ in three rules:

$$\mathsf{Eval}_{3\text{v}}^{\text{OWA}}(R(\bar{x}), D, \nu) \;=\; \begin{cases} 1 & \text{if } \nu(\bar{x}) \in R^D \\ \frac{1}{2} & \text{otherwise} \end{cases}$$

$$\mathsf{Eval}_{3\text{v}}^{\text{OWA}}(\exists x\varphi, D, \nu) \;=\; \max\big\{\tfrac{1}{2},\ \max\{\mathsf{Eval}_{3\text{v}}^{\text{OWA}}(\varphi, D, \nu[a/x]) \mid a \in \text{adom}(D)\}\big\}$$

$$\mathsf{Eval}_{3\text{v}}^{\text{OWA}}(\forall x\varphi, D, \nu) \;=\; \min\big\{\tfrac{1}{2},\ \min\{\mathsf{Eval}_{3\text{v}}^{\text{OWA}}(\varphi, D, \nu[a/x]) \mid a \in \text{adom}(D)\}\big\}$$

These modifications are sufficient for correctness under OWA.

PROPOSITION 5.8. *The evaluation algorithm* $\mathsf{Eval}_{3\text{v}}^{\text{OWA}}$ *has correctness guarantees under* OWA.

Proof. We follow the proof of Theorem 5.3; the difference is that now we need to show the following to establish connection with OWA certain answers:

$$\mathsf{Eval}_{3v}^{\text{OWA}}(\varphi, D, \nu) = 1 \;\Rightarrow\; \forall \text{ homomorphism } h : D \to D' \;\; \mathsf{Eval}_{\mathsf{FO}}(\varphi, D', h(\nu(\bar{x}))) = 1 \quad (*')$$

$$\mathsf{Eval}_{3v}^{\text{OWA}}(\varphi, D, \nu) = 0 \;\Rightarrow\; \forall \text{ homomorphism } h : D \to D' \;\; \mathsf{Eval}_{\mathsf{FO}}(\varphi, D', h(\nu(\bar{x}))) = 0 \quad (**')$$

The proof proceeds exactly as for Theorem 5.3: the cases of equality atoms and Boolean connectives are identical. For relational atoms, the case for $(*')$ is the same as before, and the case for $(**')$ disappears, since $\mathsf{Eval}_{3v}^{\text{OWA}}$ never returns 0 for relational atoms. For existential quantification, the case for $(*')$ is again the same as before (since the witness is found within the active domain), and the case for $(**')$ similarly disappears, since the minimum possible return value is $\frac{1}{2}$. Likewise, for universal quantification, the case for $(*')$ disappears since the maximum return value is $\frac{1}{2}$, and the case for $(**')$ is the same as before. □

Returning to the example from the beginning of the subsection, note that the value of $\mathsf{Eval}_{3v}^{\text{OWA}}(S(x,y), D, (1,2))$ is $\frac{1}{2}$ (for Eval_{3v} it would have been 0), and thus the result $\mathsf{Eval}_{3v}^{\text{OWA}}(R(x,y) \wedge \neg S(x,y), D, (1,2))$ is $\frac{1}{2}$ as well; in particular, $(1,2) \notin \mathsf{Eval}_{3v}^{\text{OWA}}(\varphi, D)$ while we had $(1,2) \in \mathsf{Eval}_{3v}(\varphi, D)$.

6. EVALUATION PROCEDURE FOR RELATIONAL ALGEBRA

Queries that get executed in a DBMS are procedural queries, in particular, in the relational case, they are written in relational algebra, or some of its extensions. We now present an algorithm that provides an evaluation with correctness guarantees for relational algebra expressions. Even though from the point of view of expressiveness, relational algebra is equivalent to FO, the equivalence itself, established under the standard two-valued semantics, is not yet a guarantee that it will provide us with a desired evaluation procedure in the three-valued world.

To expand on this, note that by Proposition 5.6, for every FO query $\varphi(\bar{x})$ we have a relational algebra expression e_φ which has access to the extra predicate $\mathsf{const}(\cdot)$ so that e_φ faithfully implements $\mathsf{Eval}_{3v}(\varphi, \cdot)$. So it seems that starting with a relational algebra query Q, we could find an equivalent FO query φ_Q and then consider e_{φ_Q} to evaluate Q.

Reasoning of this sort, however, mixes the equivalence of FO and relational algebra (that is true with respect to the usual two-valued FO evaluation) with the three-valued evaluation. However, equivalences under the two-valued FO evaluation need not be true under Eval_{3v}. Consider, for instance, $\varphi(x) = \exists y\big(R(x,y) \wedge (y = 1 \vee y \neq 1)\big)$. Under the two-valued semantics, it is equivalent to $\varphi'(x) = \exists y R(x,y)$, but with respect to three values, it is not. Indeed, if $R^D = \{(1, \bot)\}$ and $\nu : x \mapsto 1$, then $\mathsf{Eval}_{\mathsf{FO}}(\varphi, D, \nu) = \mathsf{Eval}_{\mathsf{FO}}(\varphi', D, \nu) = 1 = \mathsf{Eval}_{3v}(\varphi', D, \nu)$, but $\mathsf{Eval}_{3v}(\varphi, D, \nu) = \frac{1}{2}$.

Still, from the equivalence of $\mathsf{Eval}_{\mathsf{FO}}(\varphi_Q, \cdot)$ and Q one can easily derive $e_{\varphi_Q}(D) = \mathsf{Eval}_{3v}(\varphi_Q, D) \subseteq \mathsf{cert}_\bot(Q, D)$, so we do in fact get correctness guarantees with this approach. Nonetheless, it is not satisfactory for two reasons. First, the detour via translation into FO and back to algebra may produce unnecessarily complicated expressions. Second, this approach assumes a particular translation between relational algebra and FO (which of course is not unique), and the quality of the resulting query depends on that translation. For instance, we view expressions R and $\sigma_{A=1}(R) \cup \sigma_{A\neq 1}(R)$ as equivalent, but using the latter in e_{φ_Q} can miss some answers with certainty guarantees due to the presence of nulls.

The bottom line is that it is better to have a *direct* evaluation procedure for relational algebra that gives us correctness guarantees without going through both algebra-to-FO and FO-to-algebra translations.

In the two-valued world sound translations for relational algebra have been considered in the past [Reiter 1986]. Our goal is a bit different though as we have to provide specific correctness guarantees, and relate them to SQL's way of evaluating queries; in fact we shall produce approximations for sets of tuples on which Eval_{3v} returns 1 and 0.

We now explain the procedure for correct evaluation of relational algebra queries. First, recall the operations of the relational algebra. These are selection σ, projection π, cartesian product $\times$, union $\cup$, intersection $\cap$, and difference $-$. To avoid the clutter, and in particular to avoid renaming, we use the *unnamed* perspective for presenting relational algebra [Abiteboul et al. 1995], that is, for each expression returning an m-attribute relation, we simply assume that the names of those attributes are $\sharp 1, \ldots, \sharp m$. As conditions θ in selections, we use positive Boolean combinations of equalities and inequalities between attribute values and constants. For instance, $(\sharp 1 \neq \sharp 2) \vee (\sharp 3 = 1)$ is a condition that can be used in selections. Note that such conditions are closed under negation, simply by propagating it all the way to (in)equalities, so we shall also refer sometimes to conditions $\neg\theta$, meaning the result of such a propagation. We refer to this standard relational algebra as RA.

We also consider an extension called $\mathsf{RA}_{\mathsf{null}}$. In this extension, conditions θ are positive Boolean combinations of

— equalities and inequalities between attributes, and
— conditions $\mathsf{const}(\sharp n)$ and $\mathsf{null}(\sharp n)$ stating that the value of attribute $\sharp n$ is a constant or a null, respectively.

Our goal is to provide a translation $\mathsf{RA} \to \mathsf{RA}_{\mathsf{null}}$ that associates with each query Q of RA a query Q^+ of $\mathsf{RA}_{\mathsf{null}}$ such that $Q^+(D) \subseteq \mathsf{cert}_\perp(Q, D)$. Recall that $\mathsf{cert}_\perp(Q, D)$ refers to certain answers with nulls, see Definition 5.1.

As noticed already, due to coNP-data complexity of $\mathsf{cert}_\perp(Q, D)$, we cannot hope for equality, so this correctness guarantee is the best we can count on.

We shall actually produce more. Let $\bar{Q}$ be the query that computes the complement of Q, i.e., for an n-ary Q, the result of $\bar{Q}(D)$ is $\mathsf{adom}(D)^n - Q(D)$. Then we actually provide a translation

$$Q \;\mapsto\; (Q^+, Q^-)$$

of RA queries into a pair of $\mathsf{RA}_{\mathsf{null}}$ queries such that

$$Q^+(D) \;\subseteq\; \mathsf{cert}_\perp(Q, D) \quad \text{and} \quad Q^-(D) \subseteq \mathsf{cert}_\perp(\bar{Q}, D).$$

When this happens, we say that the translation $Q \;\mapsto\; (Q^+, Q^-)$ *provides correctness guarantees*.

Since $\mathsf{cert}_\perp(Q, D) \cap \mathsf{cert}_\perp(\bar{Q}, D) = \emptyset$, this also means that $Q^+(D) \cap Q^-(D) = \emptyset$. One can think of Q^+ and Q^- as analogs of finding tuples for which Eval_{3v} produces 0 or 1. Everything that does not fall into the results of these two, is essentially 'unknowns'.

We now provide the translations. We need three auxiliary elements: a translation $\theta \mapsto \theta^*$ from RA conditions to $\mathsf{RA}_{\mathsf{null}}$ conditions, one RA query, and one $\mathsf{RA}_{\mathsf{null}}$ query. These are given as follows.

The translation $\theta \mapsto \theta^*$. It is defined inductively. We assume that in conditions $\sharp n = \sharp m$ or $\sharp n \neq \sharp m$, attributes $\sharp n$ and $\sharp m$ are different (otherwise they are easily eliminated).

— If θ is $(\sharp n = \sharp m)$ or $(\sharp n = c)$, where c is a constant, then $\theta^* = \theta$.

$$
\begin{array}{rclrcl}
R^+ & = & R & R^- & = & R^{\ominus} \\
(Q_1 \cup Q_2)^+ & = & Q_1^+ \cup Q_2^+ & (Q_1 \cup Q_2)^- & = & Q_1^- \cap Q_2^- \\
(Q_1 \cap Q_2)^+ & = & Q_1^+ \cap Q_2^+ & (Q_1 \cap Q_2)^- & = & Q_1^- \cup Q_2^- \\
(Q_1 - Q_2)^+ & = & Q_1^+ \cap Q_2^- & (Q_1 - Q_2)^- & = & Q_1^- \cup Q_2^+ \\
(\sigma_\theta(Q))^+ & = & \sigma_{\theta^*}(Q^+) & (\sigma_\theta(Q))^- & = & Q^- \cup \sigma_{(\neg\theta)^*}(\mathrm{adom}^{\mathrm{ar}(Q)}) \\
(Q_1 \times Q_2)^+ & = & Q_1^+ \times Q_2^+ & (Q_1 \times Q_2)^- & = & Q_1^- \times \mathrm{adom}^{\mathrm{ar}(Q_2)} \cup \mathrm{adom}^{\mathrm{ar}(Q_1)} \times Q_2^- \\
(\pi_{\boldsymbol{\alpha}}(Q))^+ & = & \pi_{\boldsymbol{\alpha}}(Q^+) & (\pi_{\boldsymbol{\alpha}}(Q))^- & = & \pi_{\boldsymbol{\alpha}}(Q^-) - \pi_{\boldsymbol{\alpha}}(\mathrm{adom}^{\mathrm{ar}(Q)} - Q^-)
\end{array}
$$

Fig. 1. Relational algebra translations

— $(\sharp n \neq \sharp m)^* = (\sharp n \neq \sharp m) \wedge \mathsf{const}(\sharp n) \wedge \mathsf{const}(\sharp m)$.
— $(\sharp n \neq c)^* = (\sharp n \neq c) \wedge \mathsf{const}(\sharp n)$.
— $(\theta_1 \vee \theta_2)^* = \theta_1^* \vee \theta_2^*$.
— $(\theta_1 \wedge \theta_2)^* = \theta_1^* \wedge \theta_2^*$.

The active domain query. We use adom as an RA query that returns the active domain of a database; clearly it can be written as a $\pi, \cup$-query, that takes the union of all projections of all relations in the database.

The relative complement query. The *relative complement* of a k-ary relation R in database D is

$$R^{\ominus} \;=\; \{\bar{u} \in \mathrm{adom}(D)^k \mid \neg\exists \bar{t} \in R :\; \bar{u} \Uparrow \bar{t}\,\}.$$

It is not hard to see that $R^{\ominus}$ is expressible in $\mathsf{RA}_{\mathsf{null}}$. We show this formally in the proof of Theorem 6.1. In fact this is the only expression where conditions $\mathsf{null}(\sharp n)$ are used in selections.

With these, translations of relational algebra are given by inductive rules presented in Figure 1. We use abbreviation $\mathrm{ar}(Q)$ for the arity of Q, and $\boldsymbol{\alpha}$ refers to a list of attributes.

THEOREM 6.1. *The translation $Q \;\mapsto\; (Q^+, Q^-)$ in Figure 1 provides correctness guarantees.*

Proof. As the first step, we need to show that the relative complement query $R^{\ominus}$ is expressible in $\mathsf{RA}_{\mathsf{null}}$. Note that in the proof of Proposition 5.6 we showed the existence of an FO formula $\psi_{\Uparrow}(\bar{x}, \bar{y})$ which holds iff $\bar{x}$ and $\bar{y}$ unify. Such a formula uses the extra unary predicate const checking for constants. Then $R^{\ominus}$ can be expressed by $\forall \bar{y}\; R(\bar{y}) \rightarrow \neg\psi_{\Uparrow}(\bar{x}, \bar{y})$. Translating this into relational algebra using the standard translation [Abiteboul et al. 1995], we get an expression that can use arbitrary Boolean combinations of equalities and $\mathsf{const}(x)$ in selections. Then, propagating negations in such conditions, we end up with positive Boolean combinations of equalities, inequalities, $\mathsf{const}(x)$, and $\mathsf{null}(x)$, i.e., an $\mathsf{RA}_{\mathsf{null}}$ expression.

Now we show correctness guarantees. To do so, we shall need to show that, for every database D, a tuple $\bar{u} \in \mathrm{adom}(D)^k$, and a k-ary relational algebra query Q,

$$\bar{u} \in Q^+(D) \;\Rightarrow\; \forall\, \text{homomorphism } h :\; h(\bar{u}) \in Q(h(D)) \qquad (\checkmark)$$

$$\bar{u} \in Q^-(D) \;\Rightarrow\; \forall\, \text{homomorphism } h :\; h(\bar{u}) \notin Q(h(D)) \qquad (\checkmark\checkmark)$$

We now prove these by induction on the structure of relational algebra queries Q. We shall adopt the convention that a k-tuple $\bar{u}$ is $(u_1, \ldots, u_k)$; in particular, the value of attribute $\sharp n$ in $\bar{u}$ is u_n.

Case 1: Q is R, where R is a base relation.

(✓) Since Q^+ is R itself, this is immediate.

(✓✓) In this case $Q^- = R^\ominus$. Assume $\bar{u} \in R^\ominus$ and let h be a homomorphism. By definition, $\bar{u}$ does not unify with any of $\bar{t} \in R$, in particular, $h(\bar{u})$ cannot equal $h(\bar{t})$, thus implying $h(\bar{u}) \notin h(R)$.

Case 2: $Q = \sigma_\theta(Q_1)$.

(✓) In this case $Q^+ = \sigma_{\theta^*}(Q_1^+)$. We prove (✓) by (sub)induction on the selection condition.

— Let $\theta = (\sharp n = \sharp m)$. Suppose $\bar{u} \in \sigma_{\theta^*}(Q_1(D))$ (in this case, $\theta^* = \theta$), and let h be a homomorphism. We have $\bar{u} \in Q_1^+(D)$ which, by the induction hypothesis, means $h(\bar{u}) \in Q_1(h(D))$. Furthermore, $u_n = u_m$, and thus $h(u_n) = h(u_m)$, proving $h(\bar{u}) \in \sigma_\theta(Q_1(h(D)))$, which is the condition (✓) in this case.

— If $\theta = (\sharp n = c)$ for a constant c, the proof is the same.

— Let $\theta = (\sharp n \neq \sharp m)$. Suppose $\bar{u} \in \sigma_{\theta^*}(Q_1(D))$, and let h be a homomorphism. As above, we see, by the hypothesis, that $h(\bar{u}) \in Q_1^+(h(D))$. Furthermore, since θ^* holds, we know $u_n \neq u_m$ and both u_n and u_m are constants. In particular, this means $h(u_n) \neq h(u_m)$, proving $h(\bar{u}) \in \sigma_\theta(Q_1(h(D)))$.

— The case $\sharp n \neq c$ for a constant c is completely analogous.

— Let $\theta = \theta_1 \wedge \theta_2$. Then $(\sigma_\theta(Q_1))^+ = \sigma_{\theta^*}(Q_1^+) = \sigma_{\theta_1^* \wedge \theta_2^*}(Q_1^+)$. Let $\bar{u} \in (\sigma_\theta(Q_1))^+$, and let h be a homomorphism. We have $\bar{u} \in \sigma_{\theta_i^*}(Q_1^+) = (\sigma_{\theta_i}(Q_1))^+$ for each $i = 1, 2$, and thus by the hypothesis (of the induction on the selection condition) we have $h(\bar{u}) \in \sigma_{\theta_i}(Q_1(h(D)))$ for $i = 1, 2$, and hence $h(\bar{u}) \in \sigma_{\theta_1 \wedge \theta_2}(Q_1(h(D))) = \sigma_\theta(Q_1(h(D)))$, as required.

— Let $\theta = \theta_1 \vee \theta_2$. Then $(\sigma_\theta(Q_1))^+ = \sigma_{\theta^*}(Q_1^+) = \sigma_{\theta_1^*}(Q_1^+) \cup \sigma_{\theta_2^*}(Q_1^+)$. Let $\bar{u} \in (\sigma_\theta(Q_1))^+$, and let h be a homomorphism. We have $\bar{u} \in \sigma_{\theta_i^*}(Q_1^+) = (\sigma_{\theta_i}(Q_1))^+$ for one of $i = 1, 2$, and thus by the induction hypothesis on the selection condition, we have $h(\bar{u}) \in \sigma_{\theta_i}(Q_1(h(D)))$ for $i = 1$ or $i = 2$, implying $h(\bar{u}) \in \sigma_\theta(Q_1(h(D)))$.

(✓✓) In this case $Q^- = Q_1^- \cup \sigma_{(\neg\theta)^*}(\mathrm{adom}^k)$ where k is the arity of Q. First assume that a tuple $\bar{u}$ is in $Q_1^-(D)$. Then, by the induction hypothesis, $h(\bar{u}) \notin Q_1(h(D))$ for every homomorphism h, and thus $h(\bar{u}) \notin \sigma_\theta(Q_1(h(D)))$.

Thus, we only need to consider the case when $\bar{u} \in \sigma_{(\neg\theta)^*}(\mathrm{adom}^k)$. Recall that by $\neg\theta$ we mean the negation of θ in which negation has been propagated to the atomic conditions, i.e., it is still a positive Boolean combination of equalities, inequalities, and statements $\mathsf{const}(x)$ and $\mathsf{null}(x)$. We prove this case by a (sub)induction on the selection conditions.

— Let $\theta = (\sharp n = \sharp m)$. Then $\neg\theta$ is $\sharp n \neq \sharp m$ and thus $(\neg\theta)^*$ is $(\sharp m \neq \sharp m) \wedge \mathsf{const}(\sharp n) \wedge \mathsf{const}(\sharp m)$. Hence if $\bar{u}$ satisfies $(\neg\theta)^*$, then u_n and u_m are distinct constants, and the same is true for $h(u_n)$ and $h(u_m)$ for every homomorphism h; in particular, $h(\bar{u})$ cannot satisfy θ and thus $h(\bar{u}) \notin \sigma_\theta(Q_1(h(D)))$.

— The case when $\theta = (\sharp n = c)$ is analogous.

— Let $\theta = (\sharp n \neq \sharp m)$. Then $\neg\theta$ is $\sharp n = \sharp m$ and thus $(\neg\theta)^*$ is the same condition. So if $\bar{u}$ satisfies $(\neg\theta)^*$, then $u_n = u_m$ and thus $h(u_n) = h(u_m)$ for every homomorphism. In particular $h(\bar{u})$ cannot satisfy θ and thus $h(\bar{u}) \notin \sigma_\theta(Q_1(h(D)))$. Again, the case when one attribute is replaced by a constant value is completely analogous.

— Let $\theta = \theta_1 \wedge \theta_2$. Then $\neg\theta$ is $\neg\theta_1 \vee \neg\theta_2$ and thus $(\neg\theta)^*$ is $(\neg\theta_1)^* \vee (\neg\theta_2)^*$. Assume $\bar{u}$ satisfies $(\neg\theta)^*$; then it satisfies one of these conditions, say $(\neg\theta_1)^*$. By the induction hypothesis (on the selection condition), we then have

$h(\bar{u}) \notin \sigma_{\theta_1}(Q_1(h(D)))$ for every homomorphism h, which implies $h(\bar{u}) \notin \sigma_{\theta_1 \wedge \theta_2}(Q_1(h(D))) = \sigma_\theta(Q_1(h(D)))$.

— Let $\theta = \theta_1 \vee \theta_2$. Then $\neg\theta$ is $\neg\theta_1 \wedge \neg\theta_2$ and thus $(\neg\theta)^*$ is $(\neg\theta_1)^* \wedge (\neg\theta_2)^*$. Assume $\bar{u}$ satisfies $(\neg\theta)^*$; then it satisfies $(\neg\theta_1)^*$ and $(\neg\theta_2)^*$. By the induction hypothesis on the selection condition, we then have $h(\bar{u}) \notin \sigma_{\theta_i}(Q_1(h(D)))$ for every homomorphism h and for $i = 1, 2$, which implies $h(\bar{u}) \notin \sigma_{\theta_1}(Q_1(h(D))) \cup \sigma_{\theta_2}(Q_1(h(D))) = \sigma_\theta(Q_1(h(D)))$.

This completes the proof of the sub-induction on the selection condition and thus the proof of (✓✓) for selection.

Case 3: $Q = \pi_{\boldsymbol{\alpha}}(Q_1)$.

(✓) In this case $Q^+ = \pi_{\boldsymbol{\alpha}}(Q_1^+)$. Let $\bar{u} \in \pi_{\boldsymbol{\alpha}}(Q_1^+)$ and let h be a homomorphism. There is a tuple $\bar{w} \in Q_1^+$ so that $\bar{u} = \pi_{\boldsymbol{\alpha}}(\bar{w})$. By the induction hypothesis, $h(\bar{w}) \in Q_1(h(D))$ and thus $h(\bar{u}) = \pi_{\boldsymbol{\alpha}}(h(\bar{w})) \in \pi_{\boldsymbol{\alpha}}(Q_1(h(D)))$.

(✓✓) In this case $Q^- = \pi_{\boldsymbol{\alpha}}(Q_1^-) - \pi_{\boldsymbol{\alpha}}(\text{adom}^{\text{ar}(Q_1)} - Q_1^-)$. Suppose $\bar{u} \in Q^-(D)$ and let h be a homomorphism. Since $\bar{u} \in \pi_{\boldsymbol{\alpha}}(Q_1^-)$, there is a tuple $\bar{w}$, corresponding to attributes not in $\boldsymbol{\alpha}$, such that $(\bar{u}, \bar{w}) \in Q_1^-(D)$. This implies $(h(\bar{u}), h(\bar{w})) \notin Q_1(h(D))$. So the only way $h(\bar{u})$ could be in $Q(h(D))$ is if for some tuple $\bar{w}'$, we have $(h(\bar{u}), h(\bar{w}')) \in Q_1(h(D))$. Suppose this is so. Then clearly $(\bar{u}, \bar{w}') \notin Q_1^-$ for otherwise we would have $(h(\bar{u}), h(\bar{w}')) \notin Q_1(h(D))$ by the induction hypothesis. Therefore, $(\bar{u}, \bar{w}') \in \text{adom}^{\text{ar}(Q_1)} - Q_1(D)$, but this implies $\bar{u} \in \pi_{\boldsymbol{\alpha}}(\text{adom}^{\text{ar}(Q_1)} - Q_1)(D)$ which contradicts the assumption $\bar{u} \in Q^-(D)$. Thus, no such $\bar{w}'$ can exist, and this implies $h(\bar{u}) \notin Q(h(D))$.

Case 4: $Q = Q_1 \cup Q_2$.

(✓) In this case $Q^+ = Q_1^+ \cup Q_2^+$. Assume $\bar{u} \in Q^+(D)$, then $\bar{u} \in Q_i^+(D)$ for at least one of $i = 1, 2$ and by the hypothesis $h(\bar{u}) \in Q_i(h(D))$ for each homomorphism h; hence $h(\bar{u}) \in Q(h(D))$.

(✓✓) In this case $Q^- = Q_1^- \cap Q_2^-$. Let $\bar{u} \in Q(D)$ and let h be a homomorphism. By the induction hypothesis we know $h(\bar{u}) \notin Q_i(h(D))$ for both $i = 1, 2$, and hence $h(\bar{u}) \notin Q(h(D))$.

Case 5: $Q = Q_1 \cap Q_2$.

(✓) In this case $Q^+ = Q_1^+ \cap Q_2^+$. Assume $\bar{u} \in Q^+(D)$, then $\bar{u} \in Q_i^+(D)$ for $i = 1, 2$ and thus $h(\bar{u}) \in Q_i(h(D))$ for each homomorphism h; hence $h(\bar{u}) \in Q(h(D))$.

(✓✓) In this case $Q^- = Q_1^- \cup Q_2^-$. Let $\bar{u} \in Q(D)$ and let h be a homomorphism. By the induction hypothesis we know that for one of $i = 1, 2$ we have $h(\bar{u}) \notin Q_i(h(D))$, and hence $h(\bar{u}) \notin Q(h(D))$.

Case 6: $Q = Q_1 - Q_2$.

(✓) In this case $Q^+ = Q_1^+ \cap Q_2^-$. Assume $\bar{u} \in Q^+(D)$ and let h be a homomorphism. By the induction hypothesis we have $h(\bar{u}) \in Q_1(h(D))$ and $h(\bar{u}) \notin Q_2(h(D))$ and thus $h(\bar{u}) \in Q(h(D))$ as required.

(✓✓) In this case $Q^- = Q_1^- \cup Q_2^+$. Assume $\bar{u} \in Q^+(D)$ and let h be a homomorphism. If $\bar{u} \in Q_1^-(D)$, then $h(\bar{u}) \notin Q_1(h(D))$ and hence $h(\bar{u}) \notin Q(h(D))$. If $\bar{u} \in Q_2^+(D)$, then $h(\bar{u}) \in Q_2(h(D))$ and again $h(\bar{u}) \notin Q(h(D))$.

Case 6: $Q = Q_1 \times Q_2$.

(✓) In this case $Q^+ = Q_1^+ \times Q_2^+$. Assume $\bar{u} \in Q^+(D)$; then $\bar{u}$ is the concatenation of tuples $\bar{u}_1$ and $\bar{u}_2$ in $Q_1^+(D)$ and $Q_2^+(D)$. Hence, for each homomorphism h we have $h(\bar{u}_i) \in Q_i(h(D))$ and thus $h(\bar{u}) \in Q(h(D))$.

(✓✓) In this case $Q^- = Q_1^- \times \text{adom}^{\text{ar}(Q_2)} \cup \text{adom}^{\text{ar}(Q_1)} \times Q_2^-$. Let $\bar{u} \in Q(D)$. The tuple $\bar{u}$ is the concatenation of tuples $\bar{u}_1$ and $\bar{u}_2$ of lengths $\text{ar}(Q_1)$ and $\text{ar}(Q_2)$ respectively. Consider the case when $\bar{u}_1 \in Q_1^-$ and $\bar{u}_2$ is an arbitrary tuple over

the active domain (the other case is analogous). Then, by the hypothesis, $h(\bar{u}_1) \notin Q_1(h(D))$, and thus $(h(\bar{u}_1), h(\bar{u}_2)) \notin Q_1(h(D)) \times Q_2(h(D))$, i.e., $h(\bar{u}) \notin Q(h(D))$.

This concludes the proof. □

Like in the case of $\mathsf{Eval}_{3\mathrm{v}}$, these translations are faithful to the usual interpretation of relational algebra on complete databases. For a k-ary relational algebra query Q, by $\overline{Q}$ we mean its complement, i.e., the query that for a database D returns $\overline{Q}(D) = \mathrm{adom}(D)^k - Q(D)$.

PROPOSITION 6.2. *If D is a complete database, and Q is a relational algebra query, then*

$$Q^+(D) = Q(D) \;\; \textit{and} \;\; Q^-(D) = \overline{Q}(D).$$

Proof. First, observe that if no nulls occur in the database, the translated condition θ^* is equivalent to θ, and thus the rules for selection are $(\sigma_\theta(Q))^+ = \sigma_\theta(Q^+)$ and $(\sigma_\theta(Q))^- = Q^- \cup \sigma_{\neg\theta}(\mathrm{adom}^{\mathrm{ar}(Q)})$.

With these, we proceed by induction. It is immediate from the definition that $R^\ominus = \overline{R}$ for complete relations R. The rules for the Boolean connectives are immediate from De Morgan laws. For selection, we have $(\sigma_\theta(Q))^+(D) = \sigma_\theta(Q^+(D)) = \sigma_\theta(Q(D))$ and $(\sigma_\theta(Q))^-(D) = Q^-(D) \cup \sigma_{\neg\theta}(\mathrm{adom}^{\mathrm{ar}(Q)}) = \overline{Q}(D) \cup \sigma_{\neg\theta}(\mathrm{adom}^{\mathrm{ar}(Q)}) = \overline{\sigma_\theta(Q)}(D)$.

For Cartesian product, the positive case is immediate from the induction hypothesis, and for the negative case we have, again by the hypothesis, $(Q_1 \times Q_2)^-(D) = \overline{Q_1}(D) \times \mathrm{adom}^{\mathrm{ar}(Q_2)} \cup \mathrm{adom}^{\mathrm{ar}(Q_1)} \times \overline{Q_2}(D) = \overline{Q_1 \times Q_2}(D)$.

Finally, for the projection case, $(\pi_{\boldsymbol{\alpha}}(Q))^+(D) = \pi_{\boldsymbol{\alpha}}(Q^+(D)) = \pi_{\boldsymbol{\alpha}}(Q(D))$ by the induction hypothesis. Next, $(\pi_{\boldsymbol{\alpha}}(Q))^-(D) = \pi_{\boldsymbol{\alpha}}(\overline{Q}(D)) - \pi_{\boldsymbol{\alpha}}(\mathrm{adom}^{\mathrm{ar}(Q)} - \overline{Q}(D))$. Since $\overline{Q}(D) = \mathrm{adom}^{\mathrm{ar}(Q)} - Q(D)$, we have

$$\begin{aligned}
(\pi_{\boldsymbol{\alpha}}(Q))^-(D) &= \pi_{\boldsymbol{\alpha}}(\overline{Q}(D)) - \pi_{\boldsymbol{\alpha}}(Q(D)) \\
&= \pi_{\boldsymbol{\alpha}}(\mathrm{adom}^{\mathrm{ar}(Q)} - Q(D)) - \pi_{\boldsymbol{\alpha}}(Q(D)) \\
&= \pi_{\boldsymbol{\alpha}}(\mathrm{adom}^{\mathrm{ar}(Q)}) - \pi_{\boldsymbol{\alpha}}(Q(D)) \\
&= \mathrm{adom}^{\mathrm{ar}(\pi_{\boldsymbol{\alpha}}(Q))} - \pi_{\boldsymbol{\alpha}}(Q(D)) \qquad = \overline{\pi_{\boldsymbol{\alpha}}(Q)}(D)
\end{aligned}$$

by the induction hypothesis and the usual relational algebra equivalences. Specifically, to go from line 2 to line 3, we use the following, which can be easily verified: if $R \subseteq S$, then $\pi_{\boldsymbol{\alpha}}(S - R) - \pi_{\boldsymbol{\alpha}}(R) = \pi_{\boldsymbol{\alpha}}(S) - \pi_{\boldsymbol{\alpha}}(R)$. This completes the proof. □

6.1. The flexibility of the translation

The translation in Figure 1 provides correctness guarantees, but some of the queries look rather inefficient. Consider, for instance, the rule for $(Q_1 \times Q_2)^-$ which gives us $Q_1^- \times \mathrm{adom}^{\mathrm{ar}(Q_2)} \cup \mathrm{adom}^{\mathrm{ar}(Q_1)} \times Q_2^-$. This is a rather inefficient query, involving an expensive cartesian product. Can we replace it by a more efficient query, say, $Q_1^- \times Q_2^-$, which immediately comes to mind?

As another example, consider the rule for $(\sigma_\theta(Q))^-$ giving us $Q^- \cup \sigma_{(\neg\theta)^*}(\mathrm{adom}^{\mathrm{ar}(Q)})$. Again, the query involves an expensive cartesian product, and one possible way to deal with it is simply to replace this query by Q^-.

Can we always do this? And are there other possible queries we can use? The answer to these questions is positive. Note that in both cases we replaced a query from the translation in Figure 1 by a query contained in it. It turns out that any such replacement preserves correctness guarantees.

To be more precise, the translation in Figure 1 can be viewed as not just one translation but a *family* of translations, due to the following observation. A translation can

be seen as a mapping $\mathcal{F}$ that assigns to each relational algebra operation ω (including nullary operations for base relations) two queries F^+_ω and F^-_ω. These queries are simply the queries that appear on the right in the translation; for instance, for the translation scheme we used, $F^+_\cap$ is the intersection (since the result of $(Q_1 \cap Q_2)^+$ is the intersection of Q^+_1 and Q^+_2) and $F^-_\cap$ is the union (since the result of $(Q_1 \cap Q_2)^-$ is the union of Q^-_1 and Q^-_2).

Such a mapping $\mathcal{F}$ results in a translation $Q \mapsto \mathcal{F}^+_Q, \mathcal{F}^-_Q$, where $\mathcal{F}^+_Q$ and $\mathcal{F}^-_Q$ are queries of the same type as Q (i.e., they operate on databases of the same schema and have the same arity). Intuitively, these are analogs of Q^+ and Q^- that we had for the translation in Figure 1.

Formally, they are defined as follows.

— If ω is a base relation R, then F^+_R and F^-_R take no arguments and $\mathcal{F}^+_R = F^+_R$ and $\mathcal{F}^-_R = F^-_R$.

  That is, F^+_R and F^-_R are queries that give us certainly positive and certainly negative information about R.

— If ω is a unary operation (σ or π), then F^+_ω and F^-_ω take two arguments and $\mathcal{F}^+_{\omega(Q)} = F^+_\omega(\mathcal{F}^+_Q, \mathcal{F}^-_Q)$ and $\mathcal{F}^-_{\omega(Q)} = F^-_\omega(\mathcal{F}^+_Q, \mathcal{F}^-_Q)$.

  That is, if we already have queries $\mathcal{F}^+_Q$ and $\mathcal{F}^-_Q$ describing certainly positive and certainly negative answers for Q, the queries describing such answers for $\omega(Q)$ are obtained by applying F^+_ω and F^-_ω to those.

— If ω is a binary operation ($\cup, \cap, -, \times$), then F^+_ω and F^-_ω take four arguments and $\mathcal{F}^+_{\omega(Q_1,Q_2)} = F^+_\omega(\mathcal{F}^+_{Q_1}, \mathcal{F}^+_{Q_2}, \mathcal{F}^-_{Q_1}, \mathcal{F}^-_{Q_2})$ and $\mathcal{F}^-_{\omega(Q_1,Q_2)} = F^-_\omega(\mathcal{F}^+_{Q_1}, \mathcal{F}^+_{Q_2}, \mathcal{F}^-_{Q_1}, \mathcal{F}^-_{Q_2})$.

  That is, if we already have queries $\mathcal{F}^+_{Q_i}$ and $\mathcal{F}^-_{Q_i}$ describing certainly positive and certainly negative answers for Q_i, with $i = 1, 2$, the queries describing such answers for $\omega(Q_1, Q_2)$ are again obtained by applying F^+_ω and F^-_ω to those.

Given a translation $\mathcal{F}$ and another translation $\mathcal{G}$ that assigns to each operation ω queries G^+_ω and G^-_ω, we say that $\mathcal{F}$ is *contained* in $\mathcal{G}$ if $F^+_\omega \subseteq G^+_\omega$ and $F^-_\omega \subseteq G^-_\omega$, where $\subseteq$ refers to the usual query containment.

PROPOSITION 6.3. *Every translation that is contained in the translation of Figure 1 provides correctness guarantees.*

Proof. It is the same as the proof of Theorem 6.1: one just follows the proof, and notices that as long as we take a query contained in the query used on the right hand side of the translation, the correctness guarantees of the induction hypothesis continue to hold. □

This lets us adjust translations for the sake of efficiency without having to worry about correctness guarantees. Indeed, in the rule for $(Q_1 \times Q_2)^-$ we can use $Q^-_1 \times Q^-_2$, and in the rule for $(\sigma_\theta(Q))^-$ we can just use Q^-, as suggested earlier. These result in more efficient queries still giving us correctness guarantees. There is a price to pay for the efficiency though: we can get fewer answers in the result. Hence one should decide how to resolve the efficiency versus the quality of approximation trade-off.

Another corollary concerns positive relational algebra, even extended with inequalities, and it just follows from examining the basic translation of Figure 1. Define $\mathsf{PosRA}^{\neq}$ as the positive fragment of RA (i.e., $\sigma, \pi, \times, \cup$) where conditions in selections are allowed to use inequalities. In terms of its expressiveness, this fragment corresponds to $\mathsf{UCQ}^{\neq}$.

COROLLARY 6.4. *Let Q be a* PosRA$^{\neq}$ *query, and let Q^* be obtained from it by changing each selection condition θ to θ^*. Then, for every database D, we have $Q^*(D) \subseteq cert_\perp(Q, D)$.*

7. NESTED QUERIES AND EXTENDED ALGEBRA OPERATIONS

We have seen that nested SQL queries tend to cause problems when they are evaluated on databases with nulls. In fact, even some equivalences one would expect to hold are broken, and this is in fact written into the specification of the SQL semantics. The goal of this section is to illustrate that some of this unpleasant and unwanted behavior of SQL can be eliminated by using adjusted three-valued evaluation techniques presented in the earlier sections. We recall one more time that in this paper we use set semantics, i.e., in all the examples and results below we assume that relations contain no duplicates.

To illustrate additional issues one can have with nested queries, consider again the difference query (2) from the Introduction, i.e.,

```
select R.A from R where R.A not in (select S.A from S),        (2)
```

where R and S have one attribute A each. We normally expect this query to correspond to relational algebra difference operation, i.e., $R - S$. The difference of two relations can be expressed in another way in SQL, namely

```
(select * from R) except (select * from S).        (9)
```

In fact one can often see statements that these are two equivalent ways of expressing the difference query, but in fact they are not, if nulls are present. To see this, assume that $R = \{1, 2\}$ and $S = \{\perp\}$. Then query (2) returns the empty set, but query (9) returns $\{1, 2\}$. Thus, the queries are not the same after all in the presence of nulls[1].

To explain what is going on here, we need to delve a bit deeper into the meaning of such queries and their implementation. We shall use $R-S$ for the usual set difference, implemented by (9), and $R \setminus S$ for the result of the nested query (2). Of course if no nulls are present we have $R - S = R \setminus S$.

7.1. Relational calculus

Let us now look at how these queries are defined in relational calculus. If R and S have the same attributes, then $R - S$ is defined by the query $\varphi(\bar{x}) = R(\bar{x}) \wedge \neg S(\bar{x})$. On the other hand, $R \setminus S$ checks for each element of R whether it belongs to S, so the formula defining it is

$$\psi(\bar{x}) \;=\; R(\bar{x}) \wedge \neg \exists \bar{y}\, (S(\bar{y}) \wedge \bar{y} = \bar{x})\,, \tag{10}$$

where for tuples $\bar{x} = (x_1, \ldots, x_n)$ and $\bar{y} = (y_1, \ldots, y_n)$, we use the abbreviation $\bar{y} = \bar{x}$ for $(x_1 = y_1) \wedge \ldots \wedge (x_n = y_n)$. At first one may think that $\exists \bar{y}\, (S(\bar{y}) \wedge \bar{x} = \bar{y})$ is equivalent to $S(\bar{x})$ but this is *not* the case for the three-valued semantics. The table below shows what the truth values are for the database D with $R^D = \{1\}$ and $S^D = \{\perp\}$ (in the previous example we used $R = \{1, 2\}$ but the situation for values 1 and 2 is completely symmetric). Note that the certain answer in this case is empty.

[1]The behavior of query (9) is for the majority of DBMSs that have the `except` operator, or `minus` for Oracle. For those without, e.g., MySQL, the same result is achieved with the outerjoin query `select R.A from R left outer join S on (S.A is null)`.

	$\varphi(x) \;=\; R(x) \wedge \neg S(x)$	$\psi(x) \;=\; R(x) \wedge \neg\exists y\,(S(y) \wedge \bar{y} = x)$
$\mathsf{Eval}_{\mathsf{FO}}(\cdot, D, 1)$	1	1
$\mathsf{Eval}_{\mathsf{FO}}(\cdot, D, \bot)$	0	0
$\mathsf{Eval}_{\mathsf{SQL}}(\cdot, D, 1)$	1	1/2
$\mathsf{Eval}_{\mathsf{SQL}}(\cdot, D, \bot)$	0	0
$\mathsf{Eval}_{\mathsf{3v}}(\cdot, D, 1)$	1/2	1/2
$\mathsf{Eval}_{\mathsf{3v}}(\cdot, D, \bot)$	0	0

Thus, both FO and SQL evaluation produce a false positive for the difference query $\varphi(x)$. Furthermore, under the FO evaluation, formulae φ and ψ are equivalent, so FO produces a false positive for ψ as well. On the other hand, SQL evaluation returns the correct value $\frac{1}{2}$ for ψ. Then $\mathsf{Eval}_{\mathsf{3v}}$ fixes these issues, returning the correct – and the same – value $\frac{1}{2}$ for both formulae.

The equivalence of $R(\bar{x}) \wedge \neg S(\bar{x})$ and (10) boils down to the equivalence of $S(\bar{x})$ and of tuple-by-tuple definition of S. That is, for a database D, we define

$$S^D(\bar{x}) \;=\; \bigvee\{\bar{x} = \bar{t} \mid \bar{t} \in S^D\}.$$

Then we expect, in the case of the usual two-valued logic, that $S(\bar{x}) \leftrightarrow S^D(\bar{x})$ in D. In general, if this equivalence holds, then $R(\bar{x}) \wedge \neg S(\bar{x})$ and (10) are equivalent.

However, in the case of SQL, the formulae $S(\bar{x})$ and $S^D(\bar{x})$ can be completely unrelated.

PROPOSITION 7.1. *There is a database D with a unary relation S in it and two assignments ν, ν' such that*

$$\begin{array}{rcl} \mathsf{Eval}_{\mathsf{SQL}}(S(x), D, \nu) & < & \mathsf{Eval}_{\mathsf{SQL}}(S^D(x), D, \nu) \\ \mathsf{Eval}_{\mathsf{SQL}}(S(x), D, \nu') & > & \mathsf{Eval}_{\mathsf{SQL}}(S^D(x), D, \nu')\,. \end{array}$$

Proof. Consider a database D with two relations $R^D = \{\bot_2\}$ and $S^D = \{\bot_1\}$. Let $\nu : x \mapsto \bot_2$ and $\nu' : x \mapsto \bot_1$. Then

— $\mathsf{Eval}_{\mathsf{SQL}}(S(x), D, \nu) = 0$;
— $\mathsf{Eval}_{\mathsf{SQL}}(S^D(x), D, \nu) = \frac{1}{2}$;
— $\mathsf{Eval}_{\mathsf{SQL}}(S(x), D, \nu') = 1$;
— $\mathsf{Eval}_{\mathsf{SQL}}(S^D(x), D, \nu') = \frac{1}{2}$. □

For $\mathsf{Eval}_{\mathsf{3v}}$, on the other hand, we can establish a clear connection between the two definitions of a relation by formulae. In fact, for SQL's model of nulls (i.e., non-repeating nulls), the two are equivalent under $\mathsf{Eval}_{\mathsf{3v}}$, i.e., $\mathsf{Eval}_{\mathsf{3v}}$ restores the equivalence $S(\bar{x}) \leftrightarrow S^D(\bar{x})$.

PROPOSITION 7.2. *Given a database D with a relation S in it, for every assignment ν we have*

$$\mathsf{Eval}_{\mathsf{3v}}(S(\bar{x}), D, \nu) \;\leq\; \mathsf{Eval}_{\mathsf{3v}}(S^D(\bar{x}), D, \nu)\,.$$

Moreover, the equality is achieved if nulls in S^D do not repeat.

Proof. Consider a database D with a relation S in it and an assignment $\nu : \bar{x} \mapsto \bar{a}$.

First assume $\mathsf{Eval}_{\mathsf{3v}}(S(\bar{x}), D, \nu) = 1$. Then $\bar{a} \in S^D$, i.e., $\bar{a} = \bar{t}$ for some $\bar{t} \in S^D$. Hence $\mathsf{Eval}_{\mathsf{3v}}(\bar{x} = \bar{t}, D, \nu) = 1$ and thus $\mathsf{Eval}_{\mathsf{3v}}(S^D(\bar{x}), D, \nu) = 1$.

Next assume $\mathsf{Eval}_{\mathsf{3v}}(S(\bar{x}), D, \nu) = \frac{1}{2}$. Then $\bar{a}$ is not a tuple in S^D but there is a tuple $\bar{t} \in S^D$ such that $\bar{a} \Uparrow \bar{t}$. Let $\bar{a} = (a_1, \ldots, a_m)$ and $\bar{t} = (t_1, \ldots, t_m)$. Since $\bar{a}$ and $\bar{t}$ unify, there is no position i such that a_i and t_i are different constants. Hence $\mathsf{Eval}_{\mathsf{3v}}(x_i = t_i, D, \nu) \neq$

0 for every $i \leq m$. Thus, $\mathsf{Eval}_{3v}(\bar{x} = \bar{t}, D, \nu) \geq \frac{1}{2}$ and hence $\mathsf{Eval}_{3v}(S^D(\bar{x}), D, \nu) \geq \frac{1}{2}$. If, on the other hand, $\mathsf{Eval}_{3v}(S^D(\bar{x}), D, \nu) = 1$, then we have a tuple $\bar{t}$ such that $\mathsf{Eval}_{3v}(\bar{x} = \bar{t}, D, \nu) = 1$, and hence $\mathsf{Eval}_{3v}(x_i = t_i, D, \nu) = 1$ for each $i \leq m$. But this means $\nu(x_i) = a_i = t_i$ for all $i \leq m$, and thus $\bar{a} = \bar{t} \in S^D$, contradicting $\mathsf{Eval}_{3v}(S(\bar{x}), D, \nu) = \frac{1}{2}$. This implies $\mathsf{Eval}_{3v}(S^D(\bar{x}), D, \nu) = \frac{1}{2}$.

The above proves $\mathsf{Eval}_{3v}(S(\bar{x}), D, \nu) \leq \mathsf{Eval}_{3v}(S^D(\bar{x}), D, \nu)$. To show equality for databases without repeating nulls, it suffices to show that for them, $\mathsf{Eval}_{3v}(S(\bar{x}), D, \nu) = 0$ implies $\mathsf{Eval}_{3v}(S^D(\bar{x}), D, \nu) = 0$. Consider an arbitrary tuple $\bar{t} \in S^D$. We know that it does not unify with $\bar{a}$, since $\mathsf{Eval}_{3v}(S(\bar{x}), D, \nu) = 0$. Since in $\bar{t}$ nulls do not repeat, the only way for it to not unify with $\bar{a}$ is if a_i and t_i are distinct constants for some position i (otherwise we just define a homomorphism that sends each t_i that is a null to a_i, which is well-defined due to non-repetition of nulls, and the tuples unify). But in this case $\mathsf{Eval}_{3v}(x_i = t_i, D, \nu) = 0$ and thus $\mathsf{Eval}_{3v}(\bar{x} = \bar{t}, D, \nu) = 0$. Since this is true for every tuple in S^D, we have $\mathsf{Eval}_{3v}(S^D(\bar{x}), D, \nu) = 0$, as required. This completes the proof. □

Note that the condition that nulls do not repeat is necessary for equality in Proposition 7.2. Indeed, consider a database D with a binary relation $S^D = \{(1, 2)\}$ and assume $\mathrm{adom}(D)$ contains a null $\bot$. Then, for the assignment $\nu : (x, y) \mapsto (\bot, \bot)$, we have $\mathsf{Eval}_{3v}(S(\bar{x}), D, \nu) = 0$ while $\mathsf{Eval}_{3v}(S^D(\bar{x}), D, \nu) = \frac{1}{2}$.

To sum up, Eval_{3v} behaves in a much more reasonable way compared with $\mathsf{Eval}_{\mathsf{SQL}}$ and, in addition to providing correctness guarantees, eliminates some of the bizarre types of behavior of SQL when it comes to processing nested queries.

7.2. Relational algebra

We now look at the procedural implementation of queries that cause issues with databases with nulls. The `except` query, or $R(\bar{x}) \wedge \neg S(\bar{x})$ is of course the relational algebra difference, i.e., $R - S$. But what about nested queries?

Those using `in` and `not in` conditions are implemented with *semi-joins* and *anti-joins* operators. Given a relation R and a list $\boldsymbol{\alpha}$ of its attributes, and a relation S and a list $\boldsymbol{\beta}$ of its attributes so that $|\boldsymbol{\alpha}| = |\boldsymbol{\beta}|$, the semi-join $R \ltimes_{\boldsymbol{\alpha},\boldsymbol{\beta}} S$ is defined as the set of tuples $\bar{t}$ in R that match a tuple $\bar{s}$ in S, i.e., $\pi_{\boldsymbol{\alpha}}(\bar{t}) = \pi_{\boldsymbol{\beta}}(\bar{s})$. The anti-join is simply the complement $R \overline{\ltimes}_{\boldsymbol{\alpha},\boldsymbol{\beta}} S = R - (R \ltimes_{\boldsymbol{\alpha},\boldsymbol{\beta}} S)$. It consists of all the tuples in R that do not match any tuple in S.

When $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are the sets of all attributes of R and S, we drop the subscript $\boldsymbol{\alpha}, \boldsymbol{\beta}$. Note that for complete relations we have $R \ltimes S = R \cap S$ and $R \overline{\ltimes} S = R - S$. The connection with nested queries is as follows. The query

$$\texttt{select * from } R \texttt{ where } R.\boldsymbol{\alpha} \texttt{ in (select } S.\boldsymbol{\beta} \texttt{ from } S\texttt{)}$$

is $R \ltimes_{\boldsymbol{\alpha},\boldsymbol{\beta}} S$ while the query

$$\texttt{select * from } R \texttt{ where } R.\boldsymbol{\alpha} \texttt{ not in (select } S.\boldsymbol{\beta} \texttt{ from } S\texttt{)}$$

is $R \overline{\ltimes}_{\boldsymbol{\alpha},\boldsymbol{\beta}} S$.

We now extend the translation of Theorem 6.1 to handle semi- and anti-joins. We then see that queries Q^+ are identical for the difference query implemented as $R - S$ or as $R \overline{\ltimes} S$ over databases with nulls, thus providing a procedural fix to the problem outlined earlier in the section.

We need one bit of notation. Recall that we use the unnamed perspective of relational algebra, i.e., n attributes of a relation are $\sharp 1, \ldots, \sharp n$. Now let R be a relation of arity $m < n$, and S a relation of arity $n - m$, and let $\boldsymbol{\beta}$ be a list of m attributes among $\sharp 1, \ldots, \sharp n$. Then $R \times_{\boldsymbol{\beta}} S$ returns tuples in the cartesian product $R \times S$ rearranged so

that the components of tuples from R occupy the positions given by $\boldsymbol{\beta}$. That is,

$$R \times_{\boldsymbol{\beta}} S \;=\; \{(a_1, \ldots, a_n) \mid (a_i)_{i \in \boldsymbol{\beta}} \in R \text{ and } (a_i)_{i \notin \boldsymbol{\beta}} \in S\}\,.$$

Here $(a_i)_{i \in \boldsymbol{\beta}}$ is the subtuple $(a_{i_1}, \ldots, a_{i_k})$ of $(a_1, \ldots, a_n)$, where $\boldsymbol{\beta}$ consists of $\sharp i_1, \ldots, \sharp i_k$ and $i_1 < \ldots < i_k$ (and likewise for $(a_i)_{i \notin \boldsymbol{\beta}}$). For instance, if $R = \{(1,2)\}$, $S = \{(3)\}$, and $\beta = \{\sharp 1, \sharp 3\}$, then $R \times_{\boldsymbol{\beta}} S = \{(1,3,2)\}$.

Now we are ready to define translations for semi-joins and anti-joins.

$$\begin{array}{rcl}
(Q_1 \ltimes_{\boldsymbol{\alpha},\boldsymbol{\beta}} Q_2)^+ & = & Q_1^+ \ltimes_{\boldsymbol{\alpha},\boldsymbol{\beta}} Q_2^+ \\
(Q_1 \ltimes_{\boldsymbol{\alpha},\boldsymbol{\beta}} Q_2)^- & = & Q_1^- \;\cup\; \big((\pi_{\boldsymbol{\beta}}(Q_2))^- \times_{\boldsymbol{\beta}} \mathrm{adom}^{\mathrm{ar}(Q_2)-|\boldsymbol{\beta}|}\big) \\
(Q_1 \,\overline{\ltimes}_{\boldsymbol{\alpha},\boldsymbol{\beta}} Q_2)^+ & = & Q_1^+ \;\cap\; \big((\pi_{\boldsymbol{\beta}}(Q_2))^- \times_{\boldsymbol{\beta}} \mathrm{adom}^{\mathrm{ar}(Q_2)-|\boldsymbol{\beta}|}\big) \\
(Q_1 \,\overline{\ltimes}_{\boldsymbol{\alpha},\boldsymbol{\beta}} Q_2)^- & = & Q_1^- \;\cup\; Q_1^+ \ltimes_{\boldsymbol{\alpha},\boldsymbol{\beta}} Q_2^+
\end{array} \tag{11}$$

THEOREM 7.3. *Extending translations of Theorem 6.1 with rules (11) for semi-joins and anti-joins preserves correctness guarantees.*

Proof. The equations for anti-joins follow from those for semi-joins and the equations for difference from the table in Figure 1, so we only need to prove correctness guarantees for $(Q_1 \ltimes_{\boldsymbol{\alpha},\boldsymbol{\beta}} Q_2)^+$ and $(Q_1 \ltimes_{\boldsymbol{\alpha},\boldsymbol{\beta}} Q_2)^-$. For this, we simply extend the inductive proof of Theorem 6.1 with these rules. The positive case is immediate (as in fact semi-join is a positive query and for positive queries the $+$ operation propagates).

For the negative case, we need to show that for every database D and every $\bar{u} \in (Q_1 \ltimes_{\boldsymbol{\alpha},\boldsymbol{\beta}} Q_2)^-(D)$ we have $h(\bar{u}) \notin Q_1(h(D)) \ltimes_{\boldsymbol{\alpha},\boldsymbol{\beta}} Q_2(h(D))$ for every homomorphism h. If $\bar{u} \in Q_1^-(D)$ then, by the induction hypothesis, $h(\bar{u}) \notin Q_1(h(D))$ for every homomorphism h, and thus it cannot be in the result of the semi-join. If $\bar{u}$ is a tuple whose projection on the $\boldsymbol{\beta}$-attributes is in $(\pi_{\boldsymbol{\beta}}(Q))^-(D)$, then we know that for every homomorphism h, the tuple $\pi_{\boldsymbol{\beta}}(\bar{u})$ is not in $\pi_{\boldsymbol{\beta}}(Q(h(D)))$, by the hypothesis. Thus, $h(\bar{u})$ cannot be in $Q_1(h(D)) \ltimes Q_2(h(D))$, as the join condition cannot be satisfied. This concludes the proof. □

When $\boldsymbol{\alpha}$ is the set of all attributes of Q_1 and $\boldsymbol{\beta}$ is the set of all attributes of Q_2, (11) gives us:

$$\begin{array}{rcl}
(Q_1 \ltimes Q_2)^+ & = & Q_1^+ \ltimes Q_2^+ \\
(Q_1 \ltimes Q_2)^- & = & Q_1^- \;\cup\; Q_2^- \\
(Q_1 \,\overline{\ltimes}\, Q_2)^+ & = & Q_1^+ \;\cap\; Q_2^- \\
(Q_1 \,\overline{\ltimes}\, Q_2)^- & = & Q_1^- \;\cup\; Q_1^+ \ltimes Q_2^+
\end{array} \tag{12}$$

In particular, we have the following.

COROLLARY 7.4. $(Q_1 \,\overline{\ltimes}\, Q_2)^+ \;=\; (Q_1 - Q_2)^+$.

Thus, if we adopt the view that in the case of databases with nulls, the correct implementation of a relational algebra query Q is Q^+, then the anti-join and set difference become the same, unlike in SQL's implementation. In particular, there is no longer any difference between queries (2) and (9). Thus, our approach fixes this inconsistency in SQL at both the declarative level (with the help of Eval_{3v}) and the procedural level (with the help of queries Q^+).

8. CONCLUSIONS AND PRACTICAL CONSIDERATIONS

We have shown that small changes to the 3-valued query evaluation used in SQL produce sound query answers, i.e., answers without false positives. We have presented such evaluation procedures at the levels of both relational calculus and algebra.

Our goal was to address the often criticized aspects of SQL's handling of nulls. Clearly the next step after showing that correctness can in principle be restored, is to show how to do this in practice. We now outline the next steps that need to be done in this direction.

How bad the problem is. Do we often have the problem of false positives in SQL query answers? We know there are some classes of queries (e.g., positive queries extended with the division by a base relation) where the problem does not occur. On the other hand, having the `except` operator is an easy way to get false positives, as is double negation, which is common in nested queries expressing universal conditions. We need an experimental study showing how common the problem actually is. A starting point could be a set of benchmark queries on randomly generated data; the main challenge is comparing with real certain answers which are computationally very costly.

Efficiency. The theoretical complexity of the procedures we proposed is very low, in fact it is as low as evaluating relational calculus and algebra themselves, in terms of data complexity. The next obvious step is to implement these algorithms to study their real-life applicability. It is clear that some adjustments will need to be made. First of all, we assumed a single domain for all attributes (which is a standard theoretical simplification) while in reality all columns have their own types. This is easy to deal with; a more challenging problem is handling fairly large cartesian products that the translation produces. While not increasing theoretical complexity (it is still AC^0), they definitely increase practical complexity of query answering. It is worth remembering though that our translations at the procedural level are really families of algorithms. This gives plenty to play with, to find good quality approximations of certain answers that are real-world efficient.

Duplicates and multiset semantics. Another natural question is to consider the multiset (bag) semantics of SQL. As explained in the introduction, everything we have done here assumes the textbook set semantics, but SQL evaluation, by default, does not eliminate duplicates. We need to understand the right notion of correctness, similar to certain answers, in the case of databases with bags. Such a notion can then be used a basis for obtaining correctness guarantees of evaluation procedures in the presence of duplicates.

Quality of approximations. How good are our approximations of certain answers? Propositions 5.5 and 6.2 say that they do not needlessly miss answers. but there is much more to be done, both in terms of their experimental evaluation, and theoretical analysis (as we are producing polynomial time approximations of a coNP-complete problem).

Query optimization. With the change to the evaluation procedure, we must also consider the usual optimization rules: do they continue to be valid? On the positive side, we know that queries Q^+ do provide correctness guarantees, and thus standard optimizations can be applied to them, but we also would like to know whether optimizations can be applied to Q itself first before producing the Q^+ translation.

Acknowledgment I thank Chris Date, Hugh Darwen, Ron Fagin, and Cristina Sirangelo for discussions, comments and suggestions. I am also very grateful to the referees of both this paper and its earlier conference version for their careful reading of the paper and numerous helpful suggestions. Work partially supported by EPSRC grants J015377 and M025268.

REFERENCES

ABITEBOUL, S. AND DUSCHKA, O. 1998. Complexity of answering queries using materialized views. In *ACM Symposium on Principles of Database Systems (PODS)*. 254–263.

ABITEBOUL, S., HULL, R., AND VIANU, V. 1995. *Foundations of Databases*. Addison-Wesley.

ABITEBOUL, S., KANELLAKIS, P., AND GRAHNE, G. 1991. On the representation and querying of sets of possible worlds. *Theoretical Computer Science 78,* 1, 158–187.

ARENAS, M., BARCELÓ, P., LIBKIN, L., AND MURLAK, F. 2014. *Foundations of Data Exchange*. Cambridge University Press.

CALÌ, A., GOTTLOB, G., AND LUKASIEWICZ, T. 2012. A general datalog-based framework for tractable query answering over ontologies. *J. Web Sem. 14*, 57–83.

CALVANESE, D., DE GIACOMO, G., LEMBO, D., LENZERINI, M., AND ROSATI, R. 2007. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. Autom. Reasoning 39,* 3, 385–429.

COMPTON, K. 1983. Some useful preservation theorems. *Journal of Symbolic Logic 48,* 2, 427–440.

DARWEN, H. AND DATE, C. J. 1995. The third manifesto. *SIGMOD Record 24,* 1, 39–49.

DATE, C. J. 2005. *Database in Depth - Relational Theory for Practitioners*. O'Reilly.

DATE, C. J. AND DARWEN, H. 1996. *A Guide to the SQL Standard*. Addison-Wesley.

GESSERT, G. H. 1990. Four valued logic for relational database systems. *SIGMOD Record 19,* 1, 29–35.

GHEERBRANT, A., LIBKIN, L., AND SIRANGELO, C. 2014. Naïve evaluation of queries over incomplete databases. *ACM Trans. Database Syst. 39,* 4, 31:1–31:42.

GHEERBRANT, A., LIBKIN, L., AND TAN, T. 2012. On the complexity of query answering over incomplete XML documents. In *International Conference on Database Theory (ICDT)*. 169–181.

HAAS, L. M., HERNÁNDEZ, M. A., HO, H., POPA, L., AND ROTH, M. 2005. Clio grows up: from research prototype to industrial tool. In *SIGMOD*. 805–810.

IMIELINSKI, T. AND LIPSKI, W. 1984. Incomplete information in relational databases. *Journal of the ACM 31,* 4, 761–791.

KLEIN, H. 1994. How to modify SQL queries in order to guarantee sure answers. *SIGMOD Record 23,* 3, 14–20.

KLEIN, H. 1999. On the use of marked nulls for the evaluation of queries against incomplete relational databases. In *Fundamentals of Information Systems*, T. Polle, T. Ripke, and K. Schewe, Eds. Kluwer, 81–98.

LENZERINI, M. 2002. Data integration: a theoretical perspective. In *ACM Symposium on Principles of Database Systems (PODS)*. 233–246.

LIBKIN, L. 2014a. Certain answers as objects and knowledge. In *Principles of Knowledge Representation and Reasoning (KR)*. 328–337.

LIBKIN, L. 2014b. Incomplete information: what went wrong and how to fix it. In *ACM Symposium on Principles of Database Systems (PODS)*. 1–13.

LIPSKI, W. 1979. On semantic issues connected with incomplete information databases. *ACM Transactions on Database Systems 4,* 3, 262–296.

LIPSKI, W. 1984. On relational algebra with marked nulls. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*. 201–203.

MARNETTE, B., MECCA, G., PAPOTTI, P., RAUNICH, S., AND SANTORO, D. 2011. ++Spicy: an opensource tool for second-generation schema mapping and data exchange. *PVLDB 4,* 12, 1438–1441.

PATERSON, M. AND WEGMAN, M. N. 1978. Linear unification. *J. Comput. Syst. Sci. 16,* 2, 158–167.

REITER, R. 1977. On closed world data bases. In *Logic and Data Bases*. 55–76.

REITER, R. 1986. A sound and sometimes complete query evaluation algorithm for relational databases with null values. *Journal of the ACM 33,* 2, 349–347.

YUE, K. 1991. A more general model for handling missing information in relational databases using a 3-valued logic. *SIGMOD Record 20,* 3, 43–49.