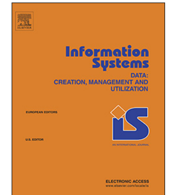
Information Systems 86 (2019) 46–60

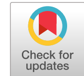
Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is



On the Codd semantics of SQL nulls



Paolo Guagliardo *, Leonid Libkin

School of Informatics, University of Edinburgh, United Kingdom of Great Britain and Northern Ireland

HIGHLIGHTS

- SQL nulls are commonly interpreted as non-repeating marked nulls, but even simple queries may produce answers that break this interpretation.
- The class of queries preserving the Codd interpretation of SQL nulls cannot be captured syntactically.
- Sufficient syntactic restrictions for preservation can be obtained by leveraging NOT NULL constraints on the database schema.

ARTICLE INFO

Article history:
Received 30 March 2018
Accepted 13 August 2018
Available online 16 August 2018

Keywords:
SQL
Null
Semantics
Relational database

ABSTRACT

Theoretical models used in database research often have subtle differences with those occurring in practice. One particular mismatch that is usually neglected concerns the use of *marked nulls* to represent missing values in theoretical models of incompleteness, while in an SQL database these are all denoted by the same syntactic **NULL** object. It is commonly argued that results obtained in the model with marked nulls carry over to SQL, because SQL nulls can be interpreted as *Codd nulls*, which are simply marked nulls that do not repeat. This argument, however, does not take into account that even simple queries may produce answers where distinct occurrences of **NULL** do in fact denote the same unknown value. For such queries, interpreting SQL nulls as Codd nulls would incorrectly change the semantics of query answers.

To use results about Codd nulls for real-life SQL queries, we need to understand which queries preserve the Codd interpretation of SQL nulls. We show, however, that the class of relational algebra queries preserving Codd interpretation is not recursively enumerable, which necessitates looking for sufficient conditions for such preservation. Those can be obtained by exploiting the information provided by **NOT NULL** constraints on the database schema. We devise mild syntactic restrictions on queries that guarantee preservation, do not limit the full expressiveness of queries on databases without nulls, and can be checked efficiently.

© 2018 Elsevier Ltd. All rights reserved.

1. Introduction

Query evaluation is a fundamental task in data management, and very often it must be performed on databases with incomplete information. This is especially true in applications such as data integration [1,2], data exchange [3] and ontology-based data access [4,5] that rely on the standard tools of existing relational database technology, in particular SQL, in order to take advantage of its efficiency. Much theoretical research on query answering and its applications uses well established models of incompleteness; however, there is an important mismatch between theoretical models and real-life SQL, which has an impact on the semantics of query answers.

Theoretical research traditionally adopts a model of incompleteness where missing values in a database are represented by *marked* (also called *naive* or *labeled*) *nulls*. In SQL, on the other hand, missing values are all represented by the same syntactic object: the ill-famed **NULL**. To reconcile the two approaches, there is a standard argument in the literature: SQL nulls are modeled by *Codd nulls*, i.e., non-repeating marked nulls. Then each occurrence of **NULL** is interpreted as a fresh marked null that does not appear anywhere else in the database, e.g., as shown in Fig. 1.

Do Codd nulls properly model SQL nulls? Sometimes (e.g., in [6]) it is argued that they are different, but this assumes a special type of query evaluation, called *naive* [7], under which marked nulls are simply viewed as new constants, e.g., $\bot_1 = \bot_1$ but $\bot_1 \neq \bot_2$. Under naive evaluation, on the database D' in Fig. 1, the relational algebra query $\sigma_{A=A}(R)$ will produce $\{\bot_1\}$. However, the same query in SQL, `SELECT * FROM R WHERE A=A`, returns the empty set on the database D in Fig. 1. The reason for this is that, in SQL, comparisons involving nulls evaluate to the truth value *unknown*, even if a null is compared with itself. This mismatch, however, is easily fixed by simply using the same comparison semantics for Codd nulls: every condition $x = y$ or $x \neq y$ evaluates to *unknown*

* Corresponding author.
E-mail addresses: pguaglia@inf.ed.ac.uk (P. Guagliardo), libkin@inf.ed.ac.uk (L. Libkin).

https://doi.org/10.1016/j.is.2018.08.001
0306-4379/© 2018 Elsevier Ltd. All rights reserved.

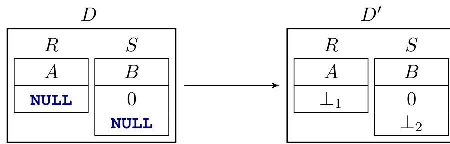


Fig. 1. SQL nulls are interpreted as Codd nulls.

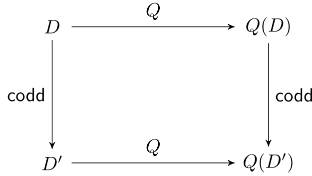


Fig. 2. Preservation of Codd semantics for SQL nulls.

if x or y is a null. In fact, we do not even need SQL's three-valued logic: the same evaluation strategy can be captured using only the usual Boolean logic with true and false [8].

So, is this sufficient to reconcile the mismatch between Codd nulls and SQL nulls? The answer is still negative, because we have not taken into account the role of queries. If SQL nulls are to be interpreted as Codd nulls, this interpretation should apply to input databases as well as query answers, which are incomplete databases themselves. To explain this point, let $\mathsf{codd}(D)$ be the result of replacing SQL nulls, in the database D, by distinct marked nulls, as shown in Fig. 1. Technically, $\mathsf{codd}(D)$ is the set of such databases, because we can choose distinct marked nulls arbitrarily (e.g., $\{\bot_3, \bot_4\}$ instead of $\{\bot_1, \bot_2\}$ for the database D' in Fig. 1), but these databases are all isomorphic. To ensure that Codd nulls faithfully represent SQL nulls for a query Q, the condition in Fig. 2 must be enforced. Intuitively, the diagram says the following: take an SQL database D, and compute the answer to Q on it, i.e., $Q(D)$. Now take some $D' \in \mathsf{codd}(D)$ and compute $Q(D')$; then $Q(D')$ must be in $\mathsf{codd}\big(Q(D)\big)$, i.e., there must be a way of assigning Codd nulls to SQL nulls in $Q(D)$ that will result in $Q(D')$.

Does this condition hold for queries in some sufficiently expressive language like relational algebra? Unfortunately, the answer is negative, even for very simple queries. Take for example the database D and its Codd interpretation D' shown in Fig. 1, and consider the query Q that computes the Cartesian product of R and S. Then, we have

$Q(D)$

A	B
NULL	0
NULL	NULL

$Q(D')$

A	B
$\bot_1$	0
$\bot_1$	$\bot_2$

and since $\bot_1$ is repeated in $Q(D)$, we *cannot* obtain $Q(D')$ from $Q(D)$ by replacing SQL nulls with Codd nulls.

The culprit in this example is that the query, evaluated on the Codd interpretation of the original SQL database, produces an answer that is not a Codd table, as it contains repeated nulls. While in this specific example it is easy to detect such an undesirable behavior, we show that in general the class of relational algebra queries that transform SQL databases into Codd databases is not recursively enumerable, and so it is impossible to capture it by a syntactic fragment of the language.

Due to the lack of an effective syntax for queries that preserve Codd semantics in answers, we can only hope for sufficient conditions, that is, finding reasonable syntactic fragments that guarantee this property. However, simply choosing a subset of relational algebra operations will not give us a useful restriction: as we saw, one of the problematic constructs is Cartesian product, and we obviously do not want a fragment that precludes joins. Thus, we must look for a more refined solution.

The idea here is to consider database schemas with **NOT NULL** constraints, which are very common in practice: base relations in real SQL databases will almost always have a **PRIMARY KEY** declared on them, and this implies **NOT NULL**. We then exploit these constraints to come up with mild restrictions on queries, which have the following properties:

- they guarantee the preservation of Codd semantics of SQL nulls in query answers on all databases;
- they can be checked efficiently; and
- they do not restrict the expressiveness of relational algebra queries on databases without nulls, where all attributes are declared as **NOT NULL**.

Organization. The paper is structured as follows:

- In Section 2 we introduce an underlying data model based on bags, and define the syntax and semantics of the query language we work with.
- In Section 3 we formally define the notion of preservation of Codd semantics in query answers and show that the class of queries having this property is not recursively enumerable.
- In Section 4 we devise syntactic restrictions on the syntax tree of queries to ensure preservation of Codd semantics, and we show that these are easy to check.
- In Section 5 we show that adding some operations to the query language may result in milder restrictions, which allow one to recognize more queries preserving Codd semantics.
- In Section 5 we study the preservation of Codd semantics for queries that are interpreted under set semantics, i.e., when duplicate rows in tables are not allowed.
- In Section 7 we conclude by summarizing the lessons learned and pointing out future research directions.

This paper is a revised and extended version of [9]. In addition to including the proofs, here we strengthen the main results of [9] by relaxing the conditions that ensure preservation of Codd semantics in query answers. Furthermore, we investigate the case of queries interpreted under *set semantics*, which is prominent in the literature. These new results are not a straightforward consequence of those in [9] and require some non-trivial additional work.

2. Preliminaries

A *bag* (or *multiset*) is an unordered collection of objects, where the same object – unlike in a set – can occur multiple times. The multiplicity (i.e., number of occurrences) of an element e in a bag B is denoted by $\#(e, B)$, and we write $e \in_k B$ for $\#(e, B) = k$. We also use the notation $e \in B$ to indicate that $e \in_k B$ for some $k > 0$, and we write $e \notin B$ for $e \in_0 B$ (i.e., e does not occur in B). For two bags B and B', we say that B is contained in B', written $B \subseteq B'$, if $\#(e, B) \le \#(e, B')$ for every $e \in B$. The bag operations of *union* $\cup$, *intersection* $\cap$ and *difference* $-$ are defined as follows:

$$B \cup B' \stackrel{\text{def}}{=\!=} \Big\{ \underbrace{e, \dots, e}_{m+n \text{ times}} \Big| e \in_m B,\ e \in_n B' \Big\};$$

$$B \cap B' \stackrel{\text{def}}{=\!=} \Big\{ \underbrace{e, \dots, e}_{\min(m,n) \text{ times}} \Big| e \in_m B,\ e \in_n B' \Big\};$$

$$B - B' \stackrel{\text{def}}{=\!=} \Big\{ \underbrace{e, \dots, e}_{m \dot{-} n \text{ times}} \Big| e \in_m B,\ e \in_n B' \Big\}.$$

where $\dot{-}$ is the *monus* operator: $m \dot{-} n = \max(0, m - n)$. Observe that union and intersection are commutative and associative, while difference is not.

For a bag B, *duplicate elimination* ε returns a new bag consisting of a single occurrence of each element in B; i.e., $\varepsilon(B)$ is the bag such that, for any element e,

$$\#\big(e, \varepsilon(B)\big) = \begin{cases} 1 & \text{if } e \in B\,, \\ 0 & \text{otherwise}\,. \end{cases}$$

2.1. Data model

We consider two countably infinite and disjoint sets of *names* and *values*, and refer to any finite (sub)set of names as a *signature*. A *record* is a function from some signature to values, and a *table* is a bag of records that are all of the same signature. The signature of a record r is denoted by $\mathsf{sig}(r)$, and likewise for tables.

The *projection* of a record r on a subset α of its signature is the restriction of r on α, denoted by $\pi_\alpha(r)$. For two records r and s of disjoint signatures, the *product* of r with s, denoted by $r \times s$, is the record over $\mathsf{sig}(r) \cup \mathsf{sig}(s)$ whose projections on $\mathsf{sig}(r)$ and $\mathsf{sig}(s)$ are r and s, respectively. Observe that product is commutative and associative. For a record r, given $N \in \mathsf{sig}(r)$ and $N' \notin \mathsf{sig}(r)$, we define the following *renaming* operation:

$$\rho_{N\to N'}(r) \stackrel{\text{def}}{=\!=} \pi_{\mathsf{sig}(r)-N}(r) \times \{N' \mapsto r(N)\}.$$

The operations on records described above extend naturally to tables:

$$\pi_\alpha(T) \stackrel{\text{def}}{=\!=} \Big\{ \underbrace{s, \ldots, s}_{k \text{ times}} \,\Big|\, k = \sum_{\substack{r \in T \\ \pi_\alpha(r) = s}} \#(r, T) \Big\}\,;$$

$$T \times T' \stackrel{\text{def}}{=\!=} \Big\{ \underbrace{r \times s, \ldots, r \times s}_{m \cdot n \text{ times}} \,\Big|\, r \in_m T,\ s \in_n T' \Big\}\,;$$

$$\rho_{N\to N'}(T) \stackrel{\text{def}}{=\!=} \Big\{ \underbrace{r', \ldots, r'}_{k \text{ times}} \mid r \in_k T,\ r' = \rho_{N\to N'}(r) \Big\}\,.$$

The bag operations $\cup$, $\cap$ and $-$ can be applied to tables of the same signature, which ensures the result is a table. Duplicate elimination ε applies without restrictions.

2.2. Schemas and incomplete databases

A (relational) schema consists of a signature of relation names and a function sig that maps each relation name R to a signature $\mathsf{sig}(R)$, whose elements are called the *attributes* of R. A *database* D associates each relation name R (of a given schema) with a table $[\![R]\!]_D$ over $\mathsf{sig}(R)$.

Databases are populated with two kinds of values: *constants* and *nulls*, which come from countably infinite and disjoint sets Const and Null, respectively. We assume that Null contains the special value $\mathbf{N}$ for SQL's **NULL**; all other elements of Null will be denoted by $\bot$, with sub- or superscripts. We denote the sets of constants and nulls that occur in a database D by $\mathsf{Const}(D)$ and $\mathsf{Null}(D)$, respectively, and we say that D is *complete* if $\mathsf{Null}(D) = \emptyset$.

A database D is called a *naive database* if $\mathbf{N} \notin \mathsf{Null}(D)$, and an *SQL database* if $\mathsf{Null}(D) \subseteq \{\mathbf{N}\}$. In other words, the special value $\mathbf{N}$ cannot appear in naive databases, while it is the only value allowed for denoting nulls in SQL databases. This terminology extends naturally to tables.

A *Codd database* (resp., *Codd table*) is a naive database (resp., naive table) in which nulls do not repeat, i.e., there can be at most one occurrence of each element from Null. Note that a Codd database is a set of Codd tables, but the converse need not hold.

2.3. Query language

We use relational algebra (RA) for bags, whose syntax is defined by the grammar in Fig. 3. This consists of two main syntactic constructs, *expressions* E and *conditions* θ, whose semantics is given in Fig. 4.

A *term* t is either a name or a value, and its semantics $[\![t]\!]_r$ is given w.r.t. a record r: if t is a name in $\mathsf{sig}(r)$, then $[\![t]\!]_r = r(t)$; else, if t is a value, $[\![t]\!]_r = t$; otherwise, $[\![t]\!]_r$ is undefined.

Atomic conditions are equality/inequality comparisons between terms, and tests that determine whether a term is null or constant; complex conditions are constructed from atomic ones by means of conjunction and disjunction. We do not use explicit negation, as this can be propagated all the way down to atoms.

The signature of a condition θ, denoted by $\mathsf{sig}(\theta)$, is the set of names appearing in it. Its semantics $[\![\theta]\!]_r$ is defined w.r.t. a record r such that $\mathsf{sig}(\theta) \subseteq \mathsf{sig}(r)$: it can be either $\mathbf{t}$ (true) or $\mathbf{f}$ (false), as determined by the rules in Fig. 4(b).

For a table T and a condition θ s.t. $\mathsf{sig}(\theta) \subseteq \mathsf{sig}(T)$, we can then define the following *selection* operation:

$$\sigma_\theta(T) \stackrel{\text{def}}{=\!=} \Big\{ \underbrace{r, \ldots, r}_{k \text{ times}} \mid r \in_k T,\ [\![\theta]\!]_r = \mathbf{t} \Big\}$$

Atomic RA expressions are simply names (of base relations in the schema), and complex ones are constructed by means of the usual operations of projection π, selection σ, Cartesian product $\times$, union $\cup$, difference $-$, intersection $\cap$, renaming ρ, and duplicate elimination ε. The signature of an expression is defined inductively as follows:

$$\begin{aligned} &\mathsf{sig}(R) \text{ is given for any relation name } R \\ \mathsf{sig}(E_1 \times E_2) &= \mathsf{sig}(E_1) \cup \mathsf{sig}(E_2) \\ \mathsf{sig}(E_1 \text{ op } E_2) &= \mathsf{sig}(E_1) \text{ for op } \in \{\cup, \cap, -\} \\ \mathsf{sig}\big(\pi_\alpha(E)\big) &= \alpha \\ \mathsf{sig}\big(\sigma_\theta(E)\big) &= \mathsf{sig}\big(\varepsilon(E)\big) = \mathsf{sig}(E) \\ \mathsf{sig}\big(\rho_{A\to B}(E)\big) &= \big(\mathsf{sig}(E) - \{A\}\big) \cup \{B\} \end{aligned}$$

Whether an expression is *well-defined* w.r.t. a schema is inductively determined as follows:

- an atomic expression R is well-defined if R is a relation name in the schema;
- $E_1 \times E_2$ is well-defined if E_1 and E_2 are well-defined and $\mathsf{sig}(E_1)$ is disjoint with $\mathsf{sig}(E_2)$;
- E_1 op E_2, for op $\in \{\cap, \cup, -\}$, is well-defined if E_1 and E_2 are well-defined and $\mathsf{sig}(E_1) = \mathsf{sig}(E_2)$;
- $\pi_\alpha(E)$ is well-defined if E is well-defined and $\mathsf{sig}(\alpha) \subseteq \mathsf{sig}(E)$;
- $\sigma_\theta(E)$ is well-defined if E is well-defined and $\mathsf{sig}(\theta) \subseteq \mathsf{sig}(E)$;
- $\varepsilon(E)$ is well-defined if E is well-defined;
- $\rho_{A\to B}(E)$ is well-defined if E is well-defined and $A \in \mathsf{sig}(E)$ and $B \notin \mathsf{sig}(E) - \{A\}$;

The RA *queries* over a given schema are all of the RA expressions that are well-defined w.r.t. that schema. Their semantics is given with respect to a database D (over the same schema) as shown in Fig. 4(a).

This language captures the basic fragment of SQL [8]: **SELECT** [**DISTINCT**]-**FROM**-**WHERE** queries, with (correlated) subqueries preceded by possibly negated **IN** and **EXISTS**, combined by **UNION**, **INTERSECT** and **EXCEPT**, with or without the **ALL** modifier.

3. Codd interpretation of SQL Nulls

Differently from naive databases, where nulls are elements of Null, missing values in SQL are all denoted by the special symbol

(a) Expressions		(b) Conditions	
$E := N$	(relation name)	$\theta := t = t$	(equality)
$\mid E \times E$	(product)	$\mid t \neq t$	(inequality)
$\mid E \cup E$	(union)	$\mid \mathsf{null}(t)$	(null test)
$\mid E \cap E$	(intersection)	$\mid \mathsf{const}(t)$	(constant test)
$\mid E - E$	(difference)	$\mid \theta \wedge \theta$	(conjunction)
$\mid \pi_\alpha(E)$	(projection)	$\mid \theta \vee \theta$	(disjunction)
$\mid \sigma_\theta(E)$	(selection)		
$\mid \varepsilon(E)$	(duplicate elimination)	$N :=$ name ; $\alpha :=$ signature	
$\mid \rho_{N\to N}(E)$	(renaming)	$t :=$ name | value	

Fig. 3. Syntax of relational algebra.

(a) Expressions		(b) Conditions
$[\![R]\!]_D$	is given for any base relation name R	$[\![t_1 = t_2]\!]_r = \mathbf{t} \iff [\![t_1]\!]_r = [\![t_2]\!]_r \in \mathsf{Const}$
$[\![E_1 \mathbin{\mathsf{op}} E_2]\!]_D$	$\stackrel{\text{def}}{=} [\![E_1]\!]_D \mathbin{\mathsf{op}} [\![E_2]\!]_D$ for $\mathsf{op} \in \{\times, \cup, \cap, -\}$	$[\![t_1 \neq t_2]\!]_r = \mathbf{t} \iff [\![t_1 = t_2]\!]_r \neq \mathbf{t}$
$[\![\pi_\alpha(E)]\!]_D$	$\stackrel{\text{def}}{=} \pi_\alpha\big([\![E]\!]_D\big)$	$[\![\mathsf{null}(t)]\!]_r = \mathbf{t} \iff [\![t]\!]_r \in \mathsf{Null}$
$[\![\sigma_\theta(E)]\!]_D$	$\stackrel{\text{def}}{=} \sigma_\theta\big([\![E]\!]_D\big)$	$[\![\mathsf{const}(t)]\!]_r = \mathbf{t} \iff [\![t]\!]_r \in \mathsf{Const}$
$[\![\varepsilon(E)]\!]_D$	$\stackrel{\text{def}}{=} \varepsilon\big([\![E]\!]_D\big)$	$[\![\theta_1 \wedge \theta_2]\!]_r = \mathbf{t} \iff [\![\theta_1]\!]_r = [\![\theta_2]\!]_r = \mathbf{t}$
$[\![\rho_{N\to N'}(E)]\!]_D$	$\stackrel{\text{def}}{=} \rho_{N\to N'}\big([\![E]\!]_D\big)$	$[\![\theta_1 \vee \theta_2]\!]_r = \mathbf{t} \iff [\![\theta_1]\!]_r = \mathbf{t}$ or $[\![\theta_2]\!]_r = \mathbf{t}$

Fig. 4. Semantics of relational algebra.

$\mathbf{N}$. To reconcile this mismatch, the occurrences of $\mathbf{N}$ in an SQL database are typically interpreted as *non-repeating* elements of Null. That is, an SQL database is seen as a Codd database where each occurrence of $\mathbf{N}$ is replaced by a fresh distinct marked null. Obviously, these nulls can be chosen arbitrarily as long as they do not repeat, so an SQL database may admit infinitely many Codd interpretations in general.

To make this notion more precise, given a record r, we denote by $\mathsf{sql}(r)$ the record r' over $\mathsf{sig}(r)$ such that

$$r'(A) = \begin{cases} r(A) & \text{if } r(A) \in \mathsf{Const}\,, \\ \mathbf{N} & \text{otherwise}\,, \end{cases}$$

for every $A \in \mathsf{sig}(r)$. We also denote by $\mathsf{sql}^{-1}(r)$ the preimage of r under sql, that is:

$$\mathsf{sql}^{-1}(r) = \big\{\, s \mid s \text{ is a record such that } \mathsf{sql}(s) = r \,\big\}\,.$$

The properties below follow easily from the definitions:

Lemma 1. *Let r be a record. Then,*

(a) *for every $\alpha \subseteq \mathsf{sig}(r)$,*

$$\mathsf{sql}\big(\pi_\alpha(r)\big) = \pi_\alpha\big(\mathsf{sql}(r)\big)\,;$$

(b) *for every $A \in \mathsf{sig}(r)$ and $B \notin \big(\mathsf{sig}(r) - \{A\}\big)$,*

$$\mathsf{sql}\big(\rho_{A\to B}(r)\big) = \rho_{A\to B}\big(\mathsf{sql}(r)\big)\,;$$

(c) *for every record s such that $\mathsf{sig}(s) \cap \mathsf{sig}(r) = \emptyset$,*

$$\mathsf{sql}(r \times s) = \mathsf{sql}(r) \times \mathsf{sql}(s)\,;$$

(d) *for every selection condition θ s.t. $\mathsf{sig}(\theta) \subseteq \mathsf{sig}(r)$,*

$$[\![\theta]\!]_r = [\![\theta]\!]_{\mathsf{sql}(r)}\,.$$

These notions are extended to tables and databases as follows. For a table T, $\mathsf{sql}(T)$ is the table over $\mathsf{sig}(T)$ such that, for every record r,

$$\#\big(r, \mathsf{sql}(T)\big) = \sum_{s \in \mathsf{sql}^{-1}(r)} \#(s, T)\,;$$

$\mathsf{sql}^{-1}(T)$ is the set of all tables T' such that $\mathsf{sql}(T') = T$.

For a database D, $\mathsf{sql}(D)$ is the SQL database of the same schema as D such that $[\![R]\!]_{\mathsf{sql}(D)} = \mathsf{sql}\big([\![R]\!]_D\big)$ for every relation name R. As with records and tables, $\mathsf{sql}^{-1}(D)$ denotes the set of all databases D' such that $\mathsf{sql}(D') = D$. Then, for an SQL database D, we define $\mathsf{codd}(D)$ as the set of all *Codd databases* in $\mathsf{sql}^{-1}(D)$. Note that, even though this set may be infinite, all of its elements are isomorphic. So, with some abuse of terminology, we speak of *the* Codd interpretation of an SQL database, which is unique up to renaming of nulls.

If SQL nulls are interpreted as Codd nulls, this should apply to input databases as well as query answers. Given a query Q, for every SQL database D there should always be a way of assigning Codd nulls to SQL nulls in $[\![Q]\!]_D$ so as to obtain $[\![Q]\!]_{D'}$, where D' is the Codd interpretation of D. In other words, the Codd interpretation of $[\![Q]\!]_D$ must be isomorphic to $[\![Q]\!]_{D'}$.

This requirement was shown in the diagram in the introduction – where $Q(D)$ denotes $[\![Q]\!]_D$ – and it is formally defined below.

Definition 1. A query Q *preserves Codd semantics* if for every SQL database D it holds that

$$[\![Q]\!]_{D'} \in \mathsf{codd}\big([\![Q]\!]_D\big)\,, \tag{1}$$

where D' is the Codd interpretation of D.

The above definition can be equivalently formulated as follows: a query Q preserves Codd semantics if, for every Codd database D, it holds that

$$\mathsf{sql}\big([\![Q]\!]_D\big) = [\![Q]\!]_{\mathsf{sql}(D)} \tag{1a}$$

and

$$[\![Q]\!]_D \text{ is a Codd table.} \tag{1b}$$

These two requirements are in fact independent of one another, as the following example shows.

Example 1. Consider a schema with two unary relation names R and S over attribute A, and the Codd database D where

$[\![R]\!]_D =$

A
$\bot_1$
0

; $[\![S]\!]_D =$

A
$\bot_2$

Then, $\mathsf{sql}(D)$ is the SQL database D' such that

$[\![R]\!]_{D'} =$

A
N
0

; $[\![S]\!]_{D'} =$

A
N

Now, take the queries $Q_1 = R \times \rho_{A\to B}(S)$ and $Q_2 = R - S$, whose answers on D and D' are as follows:

$[\![Q_1]\!]_D =$

A	B
$\bot_1$	$\bot_2$
0	$\bot_2$

; $[\![Q_1]\!]_{D'} =$

A	B
N	**N**
0	**N**

$[\![Q_2]\!]_D =$

A
$\bot_1$
0

; $[\![Q_2]\!]_{D'} =$

A
0

Thus, w.r.t. D, Q_1 satisfies (1a) but not (1b), since $[\![Q_1]\!]_D$ contains multiple occurrences of the same null, namely $\bot_2$. On the other hand, Q_2 satisfies (1b) but not (1a), because $[\![Q_2]\!]_{D'} \neq \mathsf{sql}\big([\![Q_2]\!]_D\big)$.

A natural question at this point is whether we can syntactically capture the class of RA queries that satisfy both (1a) and (1b) for every Codd database. Unfortunately, the answer is no.

Proposition 1. *For every schema with at least one binary relation symbol, the set of RA queries that preserve Codd semantics is not recursively enumerable.*

Proof. To prove this, we can show that the set of preserving queries is not recursive, by reduction from the undecidability of emptiness of relational algebra expressions; since the complement of the set is clearly r.e. (by simultaneous enumeration of queries and databases), the result follows.

For now assume that the schema contains three binary relations, R, S, and E. We know that Cartesian product $R \times S$ does not preserve Codd semantics. Now, given an arbitrary relational algebra query Q over E, consider its extension $Q_{R,S}$ over all three relations defined as follows: if Q produces nonempty result on E, output $R \times S$, otherwise output the empty table. This can be expressed as

$$\pi_{R,S}\big(\pi_\emptyset(Q) \times (R \times S)\big),$$

where $\pi_{R,S}$ means projecting on all attributes of R and S. Due to the above observation, $Q_{R,S}$ preserves Codd semantics if and only if Q always produces the empty set. Since the latter is well known to be undecidable [10,11], it implies that, for the schema with relations R, S, E, preserving Codd semantics is undecidable.

We now explain how to encode R, S, E in one binary relation. First we encode them in a ternary relation T. For this, assume three new constants r, s, e and add to T tuples (r, a, b) for every $(a, b) \in R$ and likewise (s, c, d) and (e, u, v) for $(c, d) \in S$ and $(u, v) \in E$. In addition, add a tuple (r, s, e). This tuple can be recognized as the only one whose components are precisely the elements of the first column: this is a condition that one easily checks in first-order logic and hence in relational algebra. Once we have a first-order query selecting this tuple, we can extract R, S, E from T. Since there is a first-order query that checks whether T properly encodes three relations (i.e., there exists a tuple whose elements are precisely the elements that occur in the first column and nowhere else), then the previous proof applies.

Finally we show how to encode T in a binary relation B. Then for each tuple (a, b, c) in T, create a new constant $\imath$ for its id and add tuples $(\imath, b), (c, \imath), (\imath, a), (a, \imath)$, and $(\imath, \imath)$ to B. Then one can check, using a first-order query, if B is a code for T. Indeed, one needs to look for all elements $\imath$ for which $(\imath, \imath) \in B$, and see if the only remaining tuples that use $\imath$ are of the form $(\imath, b), (c, \imath), (\imath, a), (a, \imath)$. Since in tuples $(a, b, c) \in T$ we have $a \neq b$ and $a \neq c$, there is a simple way to decode the triple (a, b, c) from them, again using a first-order query, as all of a, b, c can be unambiguously identified. Thus, the construction that uses the schema with relations R, S, E can be done using a single binary relation B, which proves the proposition. □

In light of the negative result above, we should look for syntactic restrictions that provide sufficient conditions for the preservation of Codd semantics. This is what we do in the next section.

4. Queries preserving Codd semantics

Since the set of RA queries that preserve Codd semantics cannot be captured syntactically, we can only hope for syntactic restrictions that are sufficient to guarantee this property. However, simply choosing a subset of relational algebra operations would be too restrictive to be useful: as we have seen in the examples, Cartesian product is one of the operations causing problems with the preservation of Codd semantics, and we certainly do not want a fragment that forbids joins altogether.

This suggests that, in order to come up with useful restrictions, we need some additional information beyond the query itself. Real databases typically must satisfy some integrity constraints, and when it comes to incomplete SQL databases the most basic form of constraints is to declare some of the attributes in the schema as `NOT NULL`. The use of this kind of constraints is extremely common in practice, as each base table in a real-life SQL database almost always has a subset of its attributes declared as `PRIMARY KEY`, which implies `NOT NULL`.

We model `NOT NULL` constraints as follows: the signature $\mathsf{sig}(R)$ of each base relation R is partitioned into two sets $\mathsf{n\text{-}sig}(R)$ and $\mathsf{c\text{-}sig}(R)$ of *nullable* and *non-nullable* attributes, so that $\pi_{\mathsf{c\text{-}sig}(R)}([\![R]\!]_D)$ contains only elements of Const, for every database D. That is, nulls are not allowed as values of the attributes in $\mathsf{c\text{-}sig}(R)$, while there is no restriction on the values of the attributes in $\mathsf{n\text{-}sig}(R)$. Below, we extend these notions to queries.

Definition 2. The set $\mathsf{n\text{-}sig}(Q)$ of *nullable attributes* of an RA query Q is inductively defined as follows:

$$\mathsf{n\text{-}sig}(R) \text{ is given for every relation name } R$$

$$\mathsf{n\text{-}sig}\big(Q_1 \text{ op } Q_2\big) \stackrel{\text{def}}{=\!=} \mathsf{n\text{-}sig}(Q_1) \cup \mathsf{n\text{-}sig}(Q_2) \text{ for op } \in \{\times, \cup\}$$

$$\mathsf{n\text{-}sig}\big(Q_1 \cap Q_2\big) \stackrel{\text{def}}{=\!=} \mathsf{n\text{-}sig}(Q_1) \cap \mathsf{n\text{-}sig}(Q_2)$$

$$\mathsf{n\text{-}sig}\big(Q_1 - Q_2\big) \stackrel{\text{def}}{=\!=} \mathsf{n\text{-}sig}(Q_1)$$

$$\mathsf{n\text{-}sig}\big(\pi_\alpha(Q)\big) \stackrel{\text{def}}{=\!=} \mathsf{n\text{-}sig}(Q) \cap \alpha$$

$$\mathsf{n\text{-}sig}\big(\sigma_\theta(Q)\big) \stackrel{\text{def}}{=\!=} \mathsf{n\text{-}sig}\big(\varepsilon(Q)\big) \stackrel{\text{def}}{=\!=} \mathsf{n\text{-}sig}(Q)$$

$$\mathsf{n\text{-}sig}\big(\rho_{A\to B}(Q)\big) \stackrel{\text{def}}{=\!=} \mathsf{n\text{-}sig}(Q)[A/B]$$

where $\mathsf{n\text{-}sig}(Q)[A/B]$ is obtained by replacing A with B in $\mathsf{n\text{-}sig}(Q)$. We also define $\mathsf{c\text{-}sig}(Q) = \mathsf{sig}(Q) - \mathsf{n\text{-}sig}(Q)$, and we say that Q is *non-nullable* if $\mathsf{n\text{-}sig}(Q) = \emptyset$.

From the above definition we immediately obtain:

Proposition 2. *For every RA query Q and every database D, $\pi_{\mathsf{c\text{-}sig}(Q)}\big([\![Q]\!]_D\big)$ consists only of elements of* Const. □

That is, independently of the data, the answer to a query Q can only have null values for the attributes in $\mathsf{n\text{-}sig}(Q)$.

To explain the restrictions that ensure preservation of Codd semantics, we need two additional definitions.

Definition 3. The *syntax tree* of an RA query Q is a binary (ordered) tree constructed as follows:

- Each relation symbol R is a single node labeled R.
- For each unary operation symbol op_1, the syntax tree of $\mathsf{op}_1(Q)$ has root labeled op_1 and the syntax tree of Q rooted at its single child.
- For each binary operation symbol op_2, the syntax tree of $Q\ \mathsf{op}_2\ Q'$ has root labeled op_2 and the syntax trees of Q and Q' rooted at its left child and right child, respectively.

Note that each node in the syntax tree of Q defines a subquery of Q, so we can associate properties of such queries with properties of syntax tree nodes.

The *base* of a query Q, denoted by $\mathsf{base}(Q)$, is the set of relation names that appear in it (i.e., the set of its atomic subexpressions).

Definition 4. A node in the syntax tree of a query satisfies:

NNC *(non-nullable child)*
 if one of its children is non-nullable;

NNA *(non-nullable ancestor or self)*
 if either itself or one of its ancestors is non-nullable;

DJN *(disjoint nullable attributes)*
 if its children have no common nullable attributes;

DJB *(disjoint bases)*
 if its children have bases with no relation names in common.

We now state the main result.

Theorem 1. *Let Q be an RA query whose syntax tree is such that:*

(a) *each ε node satisfies* NNC*;*
(b) *each $\cap$ and $-$ node satisfies* DJN*;*
(c) *each $\times$ node satisfies* NNA*;*
(d) *each $\cup$ node satisfies* NNC *or* DJB *or* NNA*.*

Then, Q preserves Codd semantics.

These conditions do not restrict the full expressiveness of relational algebra on databases without nulls: when all attributes in the schema are non-nullable, every query is such, hence the restrictions are trivially satisfied.

Corollary 1. *On database schemas without nullable attributes, every relational algebra query satisfies the conditions of Theorem 1.* □

Also, the restrictions are easy to check:

Proposition 3. *Deciding whether an RA query satisfies the conditions of Theorem 1 can be done in linear time w.r.t. the number of nodes in its syntax tree.*

Proof. Let n be the number of nodes in the syntax tree of the query, and proceed as follows:

1. Compute the base and the nullable attributes of each node. This step requires visiting all nodes in the tree and so its cost is n.
2. Mark all ε and $\cup$ nodes satisfying NNC, all $\cap$ and $-$ nodes satisfying DJN, all $\cup$ nodes satisfying DJB, all σ, π, ρ nodes, and all leaves. Since for each node we visit up to two child nodes, this step costs $2n$.
3. Mark all $\cup$ and $\times$ nodes in each subtree with a non-nullable root. This step can be carried out by visiting all nodes once, independently of how many of them are non-nullable, so its cost is n.
4. The query satisfies the conditions of Theorem 1 iff all nodes in the resulting tree are marked. This can be checked in one more pass, so the cost of this step is n.

The time required by the above algorithm is $O(n)$. □

Below we give an example query that satisfies the conditions of Theorem 1.

Example 2. Consider the following RA query

$$\Big(\rho_{B\to C}(R\cup S) - \rho_{D\to A}\big(\varepsilon(T)\big)\Big) \cup \pi_{A,C}\big(\sigma_{A=1\vee B=C}(R\times T)\big)$$

whose syntax tree is shown in Fig. 5. To the left of each node we indicate the corresponding signature, where non-nullable attributes are underlined. For the leaves, this information is provided by the schema.

On the left side of the tree, the innermost $\cup$ node satisfies DJB (but not NNC), the ε node satisfies NNC and so it is non-nullable, and the $-$ node satisfies DJN as its children have no nullable attributes in common.

On the right side of the tree, the $\times$ node satisfies NNA because its ancestor π is non-nullable, which also ensures that the root node $\cup$ satisfies NNC (but not DJB). Thus, the query preserves Codd semantics.

In what follows, we will outline the main intuition and ideas behind the conditions of Theorem 1, and develop the technical results needed to provide its formal proof.

If a query is non-nullable, then it trivially satisfies (1b), because the answer to it will not contain nulls for any database. However, there are non-nullable queries for which (1a) does not hold: for instance, $\pi_A\big(\varepsilon(R)\big)$ when R has attributes A and B, of which only A is non-nullable. Hence, non-nullability must be used more carefully, on the subqueries of Q.

Duplicate elimination, intersection and difference may cause problems with (1a). What these operations have in common is that they match nulls *syntactically*, i.e., as if they were constants. Now, while SQL nulls are all syntactically the same, this is not the case for Codd nulls, so for these operations it makes a difference which model of nulls we use. Indeed, replacing nulls with **N** before or after each of these operations is applied may give different results, as shown in Fig. 6.

On the other hand, the remaining operations are not affected: projection, renaming, union and Cartesian product do not rely on any kind of value matching, and nulls in selection conditions are not compared syntactically. Indeed, the sql transformation commutes with projection, selection and renaming, and it distributes over union and Cartesian product, on all input tables (not necessarily Codd or even naive) for which each of these operations is well defined, but without further restrictions.

Lemma 2. *Let T be a table. Then,*

(a) $\mathsf{sql}\big(\pi_\alpha(T)\big) = \pi_\alpha\big(\mathsf{sql}(T)\big)$ *for every $\alpha \subseteq \mathsf{sig}(T)$;*
(b) $\mathsf{sql}\big(\sigma_\theta(T)\big) = \sigma_\theta\big(\mathsf{sql}(T)\big)$ *for every selection condition θ such that $\mathsf{sig}(\theta) \subseteq \mathsf{sig}(T)$;*
(c) $\mathsf{sql}\big(\rho_{A\to B}(T)\big) = \rho_{A\to B}\big(\mathsf{sql}(T)\big)$ *for every $A \in \mathsf{sig}(T)$ and $B \notin \big(\mathsf{sig}(T) - \{A\}\big)$.*

Lemma 3. *Let T_1 and T_2 be tables. Then,*

(a) $\mathsf{sql}(T_1 \cup T_2) = \mathsf{sql}(T_1) \cup \mathsf{sql}(T_2)$ *whenever T_1 and T_2 have the same signature;*
(b) $\mathsf{sql}(T_1 \times T_2) = \mathsf{sql}(T_1) \times \mathsf{sql}(T_2)$ *whenever T_1 and T_2 have disjoint signatures.*

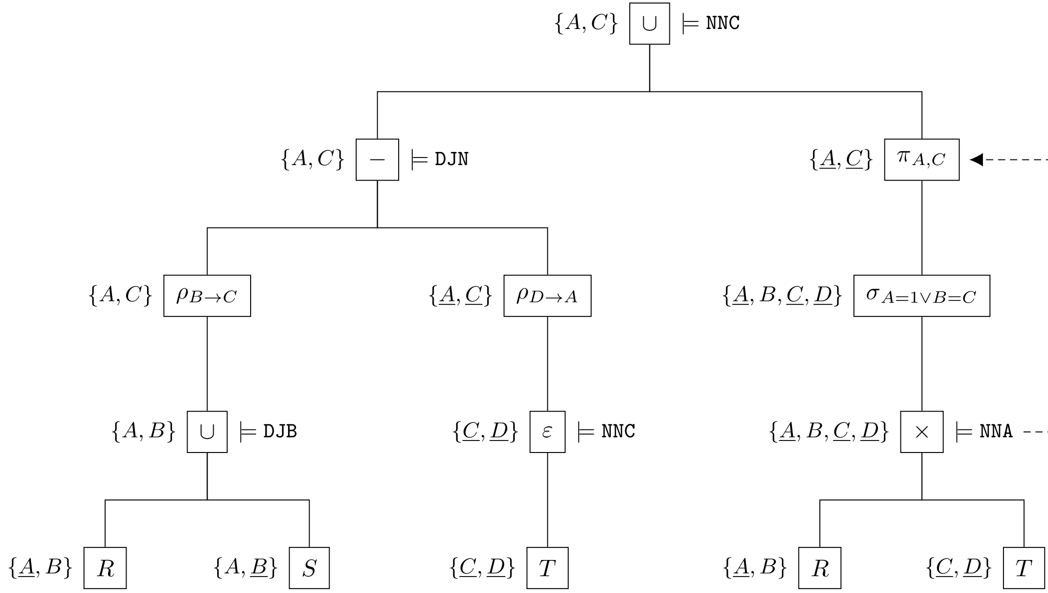


Fig. 5. Annotated syntax tree of the query of Example 2.

$$\mathsf{sql}\left(\varepsilon\left(\begin{array}{|c|}\hline A\\\hline \bot_1\\ \bot_2\\\hline\end{array}\right)\right) = \mathsf{sql}\left(\begin{array}{|c|}\hline A\\\hline \bot_1\\ \bot_2\\\hline\end{array}\right) = \begin{array}{|c|}\hline A\\\hline \mathbf{N}\\ \mathbf{N}\\\hline\end{array} \neq \begin{array}{|c|}\hline A\\\hline \mathbf{N}\\\hline\end{array} = \varepsilon\left(\begin{array}{|c|}\hline A\\\hline \mathbf{N}\\ \mathbf{N}\\\hline\end{array}\right) = \varepsilon\left(\mathsf{sql}\left(\begin{array}{|c|}\hline A\\\hline \bot_1\\ \bot_2\\\hline\end{array}\right)\right)$$

$$\mathsf{sql}\left(\begin{array}{|c|}\hline A\\\hline \bot_1\\\hline\end{array} \cap \begin{array}{|c|}\hline A\\\hline \bot_2\\\hline\end{array}\right) = \mathsf{sql}(\varnothing) = \varnothing \neq \begin{array}{|c|}\hline A\\\hline \mathbf{N}\\\hline\end{array} = \begin{array}{|c|}\hline A\\\hline \mathbf{N}\\\hline\end{array} \cap \begin{array}{|c|}\hline A\\\hline \mathbf{N}\\\hline\end{array} = \mathsf{sql}\left(\begin{array}{|c|}\hline A\\\hline \bot_1\\\hline\end{array}\right) \cap \mathsf{sql}\left(\begin{array}{|c|}\hline A\\\hline \bot_2\\\hline\end{array}\right)$$

$$\mathsf{sql}\left(\begin{array}{|c|}\hline A\\\hline \bot_1\\\hline\end{array} - \begin{array}{|c|}\hline A\\\hline \bot_2\\\hline\end{array}\right) = \mathsf{sql}\left(\begin{array}{|c|}\hline A\\\hline \bot_1\\\hline\end{array}\right) = \begin{array}{|c|}\hline A\\\hline \mathbf{N}\\\hline\end{array} \neq \varnothing = \begin{array}{|c|}\hline A\\\hline \mathbf{N}\\\hline\end{array} - \begin{array}{|c|}\hline A\\\hline \mathbf{N}\\\hline\end{array} = \mathsf{sql}\left(\begin{array}{|c|}\hline A\\\hline \bot_1\\\hline\end{array}\right) - \mathsf{sql}\left(\begin{array}{|c|}\hline A\\\hline \bot_2\\\hline\end{array}\right)$$

Fig. 6. In general, sql does not commute with ε and it does not distribute over $\cap$ and $-$.

The proofs of these lemmas are given in the Appendix.

In light of the above, w.r.t. (1a), we only need to take care of ε, $-$ and $\cap$. It is obvious from the definitions that ε commutes with sql on complete tables (i.e., tables without nulls). In particular, we have that

$$\mathsf{sql}(\varepsilon(T)) = \varepsilon(\mathsf{sql}(T)) = \varepsilon(T)$$

for every table T such that $\mathsf{Null}(T) = \emptyset$. For queries, the NNC condition then ensures that the input table of each ε operation is indeed complete.

Below, we provide a sufficient condition on tables that guarantees that sql distributes over intersection and difference.

Lemma 4. *Let T_1 and T_2 be tables with the same signature. Assume that, for every attribute A, there do not exist records r_1 and r_2 such that $r_i \in T_i$ and $r_i(A) \in \mathsf{Null}$, for $i = 1, 2$. Then, all of the following hold:*

(a) $\mathsf{sql}(T_1 - T_2) = \mathsf{sql}(T_1) - \mathsf{sql}(T_2)$;
(b) $\mathsf{sql}(T_1 \cap T_2) = \mathsf{sql}(T_1) \cap \mathsf{sql}(T_2) = T_1 \cap T_2$.

The proof of the above lemma is given in the Appendix.

Requiring NNC for each ε, $\cap$ and $-$ subquery, as we did in [9], is enough to ensure that the overall query satisfies (1a). Intuitively, if at least one of the input subqueries to each of these operations is non-nullable, then no syntactic matching of nulls can occur, because one of the operands (the only one for ε) will not have nulls at all. As a matter of fact, for $\cap$ and $-$, NNC is a special case of DJN, which only requires that the input subqueries do not share nullable attributes, and it suffices for the tables produced by such subqueries to satisfy the assumptions of Lemma 4.

Proposition 4. *Every RA query whose syntax tree satisfies conditions (a) and (b) of Theorem 1 satisfies (1a) on all databases (not only Codd ones).*

Proof. Let Q be an RA query whose syntax tree is such that every $\cap$ and $-$ node satisfies DJN, and every ε node satisfies NNC. Let D be a database and let $D' = \mathsf{sql}(D)$. By induction on the structure of Q, we will show that $[\![Q]\!]_{D'} = \mathsf{sql}([\![Q]\!]_D)$.

Base: Q is a relation name R. Then, obviously, $[\![R]\!]_{D'} = \mathsf{sql}([\![R]\!]_D)$ since $D' = \mathsf{sql}(D)$.

Induction:

- Q is $\varepsilon(Q_1)$

  Since Q satisfies NNC, Q_1 is non-nullable and so $[\![Q_1]\!]_D$ is complete. By the induction hypothesis, $\mathsf{sql}([\![Q_1]\!]_D) = [\![Q_1]\!]_{D'}$ and, as $[\![Q_1]\!]_D$ is complete, this implies $[\![Q_1]\!]_D = [\![Q_1]\!]_{D'}$. In turn, we have $[\![\varepsilon(Q_1)]\!]_{D'} = [\![\varepsilon(Q_1)]\!]_D$ and, as $[\![\varepsilon(Q_1)]\!]_D$ is complete, $[\![\varepsilon(Q_1)]\!]_{D'} = \mathsf{sql}\big([\![\varepsilon(Q_1)]\!]_D\big)$.

- Q is $\mathsf{op}_1(Q_1)$ for $\mathsf{op}_1 \in \{\pi_\alpha, \sigma_\theta, \rho_{A\to B}\}$

  By applying (†) the semantics of op_1 in queries, (††) the induction hypothesis, and (‡) Lemma 2, we obtain:

$$[\![\mathsf{op}_1(Q_1)]\!]_{D'} \overset{(\dagger)}{=} \mathsf{op}_1([\![Q_1]\!]_{D'}) \quad \overset{(\dagger\dagger)}{=} \mathsf{op}_1\left(\mathsf{sql}([\![Q_1]\!]_D)\right)$$

$$\stackrel{(\ddagger)}{=} \mathsf{sql}\big(\mathsf{op}_1([\![Q_1]\!]_D)\big) \stackrel{(\dagger)}{=} \mathsf{sql}\big(([\![\mathsf{op}_1(Q_1)]\!]_D)\big)\ .$$

- $\boxed{Q \text{ is } Q_1\ \mathsf{op}_2\ Q_2 \text{ for } \mathsf{op}_2 \in \{\times, \cup, \cap, -\}}$

  By applying (†) the semantics of op_2 in queries, (‡) the induction hypothesis, (††) Lemma 3 (for Cartesian product and union) and Lemma 4 (for intersection and difference), we obtain:

$$\begin{aligned}[\![Q_1\ \mathsf{op}_2\ Q_2]\!]_{D'} &\stackrel{(\dagger)}{=} [\![Q_1]\!]_{D'}\ \mathsf{op}_2\ [\![Q_2]\!]_{D'}\\ &\stackrel{(\ddagger)}{=} \mathsf{sql}\big([\![Q_1]\!]_D\big)\ \mathsf{op}_2\ \mathsf{sql}\big([\![Q_2]\!]_D\big)\\ &\stackrel{(\dagger\dagger)}{=} \mathsf{sql}\big([\![Q_1]\!]_D\ \mathsf{op}_2\ [\![Q_2]\!]_D\big)\\ &\stackrel{(\dagger)}{=} \mathsf{sql}\big([\![Q_1\ \mathsf{op}_2\ Q_2]\!]_D\big)\ .\end{aligned}$$

  The only observation is that, since $\cap$ and $-$ are required to satisfy DJN, in these cases $\mathsf{n\text{-}sig}(Q_1)$ and $\mathsf{n\text{-}sig}(Q_2)$ are disjoint, ensuring that the assumptions of Lemma 4 hold for the tables $[\![Q_1]\!]_D$ and $[\![Q_2]\!]_D$. □

At this point, if in addition to the conditions of Proposition 4 we also required the query itself to be non-nullable, then preservation of Codd semantics would be guaranteed.

Corollary 2. *Every non-nullable RA query that satisfies the conditions of Proposition 4 preserves Codd semantics.*

Proof. Let Q be a non-nullable RA query, and let D be a Codd database. If Q satisfies the conditions of Proposition 4, then it satisfies (1a) on all databases, in particular D. Since Q is non-nullable, $[\![Q]\!]_D$ is complete by Proposition 2, and so it is trivially a Codd table; therefore, Q also satisfies (1b) on D. □

However, this is too restrictive, as it would forbid even the simple retrieval of a base relation whenever some of its attributes are nullable. For example, if R has a nullable attribute A, the query $Q = R$ would not be allowed because it is nullable, even though Q satisfies (1a) by Proposition 4 and its output is trivially guaranteed to be a Codd table on every Codd database. Thus, we need more refined ways of ensuring (1b) by restricting only the problematic operations.

Duplicate elimination, difference, selection and intersection always produce a table that is contained in at least one of the input tables, so their output may have repeated nulls only if these repetitions were already in the input. The same holds for projection, for a similar reason. The problematic operations w.r.t. (1b) are $\cup$ and $\times$. Both can create repetitions of nulls across records, and the latter can also create repetitions of nulls within a record. To restrict these operations appropriately, we use the NNA condition. Requiring this condition for each $\times$ and $\cup$ node is enough to ensure that the query satisfies (1b). Intuitively, repetitions of nulls that may be created by $\cup$ and $\times$ are allowed in the intermediate results of a query, but only as long as they will be eventually discarded, at the latest when the final output is produced.

In fact, the NNA condition for union can be relaxed. We want to restrict this operation when it may create "new" repetitions of nulls, but we need not do so when the only repetitions it may produce are those inherited from the input, which were introduced by the application of previous operations. There are two cases in which this happens: when one of the operands is non-nullable, or when the input tables are built from disjoint sets of base relations. The first is captured by the NNC condition, and the second by the DJB condition. We can then allow for these additional possibilities by requiring each $\cup$ node in the syntax tree of a query to satisfy NNA *or* NNC *or* DJB.

Now that we explained why and how the conditions of Definition 4 are needed, we can finally prove Theorem 1.

Proof of Theorem 1. Let Q be an RA query whose syntax tree satisfies the conditions of Theorem 1. Clearly, Q satisfies (1a) on all databases by Proposition 4, thus we only need to show that $[\![Q]\!]_D$ is a Codd table for every Codd database D. To this end, we proceed by induction on the structure of Q.

Base:

- Q is non-nullable. Then, $[\![Q]\!]_D$ is complete and so it is trivially a Codd table.
- Q is a relation name R. Then, $[\![Q]\!]_D = [\![R]\!]_D$, which is a Codd table since D is a Codd database by assumption.

Induction: Observe that the case when Q is $Q_1 \times Q_2$ is already covered in the base case, since the NNA condition implies that Q is non-nullable. For the same reason, we need not consider the case when Q is $Q_1 \cup Q_2$ and satisfies NNA, as this again implies non-nullability. Similarly, the cases when Q is $\varepsilon(Q_1)$ or $Q_1 \cap Q_2$ are also covered in the base case, because of the NNC and DJN conditions.

- $\boxed{Q \text{ is } \pi_\alpha(Q_1)}$

  Towards a contradiction, suppose that $[\![\pi_\alpha(Q_1)]\!]_D$ is not a Codd table. There are three possibilities:

  (1) There is a record $r \in [\![\pi_\alpha(Q_1)]\!]_D$ such that $r(A) = r(B) \in \mathsf{Null}$ for two distinct attributes $A, B \in \alpha$. As $r \in [\![\pi_\alpha(Q_1)]\!]_D$, there must be $r' \in [\![Q_1]\!]_D$ such that $r = \pi_\alpha(r')$. But then $r'(A) = r(A) = r(B) = r'(B) \in \mathsf{Null}$, which is impossible because $[\![Q_1]\!]_D$ is a Codd table by the induction hypothesis.
  (2) There exist distinct records $r, s \in [\![\pi_\alpha(Q_1)]\!]_D$ such that $r(A) = s(B) \in \mathsf{Null}$ for two (not necessarily distinct) attributes $A, B \in \alpha$. Since $r, s \in [\![\pi_\alpha(Q_1)]\!]_D$, there must be distinct $r', s' \in [\![Q_1]\!]_D$ s.t. $r = \pi_\alpha(r')$ and $s = \pi_\alpha(s')$. But then $r'(A) = r(A) = s(B) = s'(B) \in \mathsf{Null}$, which is impossible because $[\![Q_1]\!]_D$ is a Codd table by the induction hypothesis.
  (3) There is a record $r \in_k [\![\pi_\alpha(Q_1)]\!]_D$ with $k > 1$ such that $r(A) \in \mathsf{Null}$ for some attribute $A \in \alpha$. Since $r \in_k [\![\pi_\alpha(Q_1)]\!]_D$, there must be records $r_1, \dots, r_n \in [\![Q_1]\!]_D$ such that $k = \sum_{i=1}^{n} \#\big(r_i, [\![Q_1]\!]_D\big)$ and $r_i = \pi_\alpha(r)$ for $i = 1, \dots, n$. But then $r(A) = r_1(A) = \cdots = r_n(A) \in \mathsf{Null}$, which is impossible since $[\![Q_1]\!]_D$ is a Codd table by the induction hypothesis.

- $\boxed{Q \text{ is } \sigma_\theta(Q_1) \text{ or } Q_1 - Q_2}$

  The claim follows from the fact that $[\![Q]\!]_D \subseteq [\![Q_1]\!]_D$ and, by the induction hypothesis, $[\![Q_1]\!]_D$ is a Codd table.

- $\boxed{Q \text{ is } \rho_{A\to B}(Q_1)}$

  The claim trivially follows from the fact that $\rho_{A\to B}$ only changes the name of attribute A in each record in $[\![Q_1]\!]_D$, which is a Codd table by the induction hypothesis, but not the value this attribute is mapped to.

- $\boxed{Q \text{ is } Q_1 \cup Q_2 \text{ and satisfies NNC}}$

  We have $[\![Q]\!]_D = [\![Q_1]\!]_D \cup [\![Q_2]\!]_D$. As union is commutative, we assume w.l.o.g. that Q_1 is non-nullable; then $[\![Q_1]\!]_D$ is complete and thus $\mathsf{Null}([\![Q]\!]_D) = \mathsf{Null}([\![Q_2]\!]_D)$. Since $[\![Q_2]\!]_D$ is a Codd table by the induction hypothesis, this implies that $[\![Q]\!]_D$ is a Codd table as well.

- $\boxed{Q \text{ is } Q_1 \cup Q_2 \text{ and satisfies DJB}}$

  By the induction hypothesis $[\![Q_1]\!]_D$ and $[\![Q_2]\!]_D$ are Codd tables; if we show that they have no nulls in common, we can conclude

that $[\![Q]\!]_D = [\![Q_1]\!]_D \cup [\![Q_2]\!]_D$ is a Codd table as well. For $i = 1, 2$ we have

$$\mathsf{Null}([\![Q_i]\!]_D) \subseteq \bigcup_{R \in \mathsf{base}(Q_i)} \mathsf{Null}([\![R]\!]_D) .$$

As D is a Codd database, $\mathsf{Null}([\![R]\!]_D) \cap \mathsf{Null}([\![S]\!]_D) = \emptyset$ for any two distinct relation names in the schema. Then, as $\mathsf{base}(Q_1) \cap \mathsf{base}(Q_2) = \emptyset$, we obtain that $\mathsf{Null}([\![Q_1]\!]_D)$ and $\mathsf{Null}([\![Q_2]\!]_D)$ are disjoint as desired. □

5. Derived operations and further refinements

In our query language we explicitly included intersection, even though this operation is expressible in terms of difference. While it may seem redundant to have ∩ in the syntax of queries, it is not so w.r.t. our restrictions for the preservation of Codd semantics. To see why, suppose we want the intersection of base relations R and S (with the same signature), but we do not have ∩ available as a primitive operation in our query language. Of course, we can always write $R - (R - S)$ or $S - (S - R)$; the problem is that, even though these queries are equivalent, the former satisfies the conditions of Theorem 1 iff R is non-nullable, while the latter satisfies them iff S is non-nullable. In either case, each requirement (and even their disjunction) is stronger than the one for $R \cap S$ (i.e., DJN), which only requires R and S not to have nullable attributes in common.

This suggests that adding some derived operations explicitly to the syntax of queries may lead to milder restrictions, allowing us to recognize more queries that preserve Codd semantics. We will show two such refinements: constant selections and anti/semijoins.

Selections. Looking at σ may seem counterintuitive as this operation did not need to be restricted in any way. However, consider the query $\sigma_{\mathsf{const}(A)}(R) \cap S$, for unary relation symbols R and S over a nullable attribute A. This query clearly does not satisfy the conditions of Theorem 1, as A is nullable in both $\sigma_{\mathsf{const}(A)}(R)$ and S. But even though A is nullable in $\sigma_{\mathsf{const}(A)}(R)$, we can rest assured that, for every database D, no nulls will in fact appear in $[\![\sigma_{\mathsf{const}(A)}(R)]\!]_D$, because only records where the value of A is a constant will be selected from $[\![R]\!]_D$.

The idea here is to extend the query language with an explicit operation of *constant selection* $\sigma_{\mathsf{const}(\alpha)}$, by adding the production rule $\sigma_{\mathsf{const}(\alpha)}(E)$ to the grammar of expressions in Fig. 3(a). The signature of $\sigma_{\mathsf{const}(\alpha)}(E)$ is $\mathsf{sig}(E)$, as for regular selections, and the expression is well-defined if E is well-defined and $\alpha \subseteq \mathsf{sig}(E)$. Its semantics is given as follows:

$$[\![\sigma_{\mathsf{const}(\alpha)}(E)]\!]_D \stackrel{\text{def}}{=\!=} \sigma_\theta\big([\![Q]\!]_D\big)\,, \quad \text{where } \theta = \bigwedge_{A \in \alpha} \mathsf{const}(A)$$

Finally – and this is the crucial bit – its nullable attributes are given by

$$\text{n-sig}\big(\sigma_{\mathsf{const}(\alpha)}(Q)\big) = \text{n-sig}(Q) - \alpha$$

Semijoins and antijoins. The *theta-join* $\bowtie_\theta$ of two tables T_1 and T_2 with disjoint sets of attributes is defined as

$$T_1 \bowtie_\theta T_2 \stackrel{\text{def}}{=\!=} \sigma_\theta\big(T_1 \times T_2\big)$$

where θ is a condition such that $\mathsf{sig}(\theta) \subseteq \mathsf{sig}(T_1) \cup \mathsf{sig}(T_1)$. Adding $\bowtie_\theta$ to the language does not change anything: we impose no restriction on selections, and $\bowtie_\theta$ would have to be restricted in the same way × is. But two operations derived from it can be added: *semijoin* $\ltimes_\theta$ and *antijoin* $\overline{\ltimes}_\theta$, which essentially correspond to **EXISTS** and **NOT EXISTS** in SQL. For tables T_1 and T_2 with disjoint signatures and a condition θ such that $\mathsf{sig}(\theta) \subseteq \mathsf{sig}(T_1) \cup \mathsf{sig}(T_2)$, these are defined as:

$$T_1 \ltimes_\theta T_2 \stackrel{\text{def}}{=\!=} T_1 \cap \pi_{\mathsf{sig}(T_1)}(T_1 \bowtie_\theta T_2)$$

$$T_1 \overline{\ltimes}_\theta T_2 \stackrel{\text{def}}{=\!=} T_1 - T_1 \ltimes_\theta T_2$$

That is, $T_1 \ltimes_\theta T_2$ returns all occurrences of each record r in T_1 for which there exists a record s in T_2 such that θ is true on $r \times s$. Similarly, $T_1 \overline{\ltimes}_\theta T_2$ returns all occurrences of each record r in T_1 for which there is no record s in T_2 such that $[\![\theta]\!]_{r\times s} = \mathbf{t}$.

To include these operations in the query language, we add the production rules $E \ltimes_\theta E$ and $E \overline{\ltimes}_\theta E$ to the grammar of expressions in Fig. 3(a). The signature of $E_1 \ltimes_\theta E_2$ and $E_1 \overline{\ltimes}_\theta E_2$ is $\mathsf{sig}(E_1)$. The expressions are well-defined if E_1 and E_2 are well-defined and $\mathsf{sig}(\theta) \subseteq \mathsf{sig}(E_1) \cup \mathsf{sig}(E_2)$. Their semantics is given by

$$[\![E_1 \ltimes_\theta E_2]\!]_D \stackrel{\text{def}}{=\!=} [\![E_1]\!]_D \ltimes_\theta [\![E_2]\!]_D$$

$$[\![E_1 \overline{\ltimes}_\theta E_2]\!]_D \stackrel{\text{def}}{=\!=} [\![E_1]\!]_D \overline{\ltimes}_\theta [\![E_2]\!]_D$$

and, finally, we define

$$\text{n-sig}(Q_1 \ltimes_\theta Q_2) = \text{n-sig}(Q_1 \overline{\ltimes}_\theta Q_2) = \text{n-sig}(Q_1)$$

How do we restrict these operations to ensure (1a) and (1b)? Observe that $\ltimes$ is an intersection and $\overline{\ltimes}$ is a difference, for which syntactic matching of nulls must in general be prevented. Here, however, one of the operands to this intersection or difference is always contained in the other, regardless of whether SQL nulls or Codd nulls are used, so syntactic matching of nulls is not a problem in this case. In turn, we need no restriction on $\ltimes$ and $\overline{\ltimes}$ to ensure (1a).

As for (1b), observe that the output of both $\ltimes$ and $\overline{\ltimes}$ is always contained in their left input, so no new repetitions of nulls can be created and, in turn, we do not need any restriction to guarantee (1b) either.

Since repetitions of nulls that might have been created in the right operand of $\ltimes$ or $\overline{\ltimes}$ are discarded, we can use an alternative to NNA, called RDS: a node in the syntax tree of a query satisfies this condition if it is the right-descendant of a node labeled $\ltimes$ or $\overline{\ltimes}$.

We denote by $\mathrm{RA}^{\mathrm{ext}}$ the language of Fig. 3 extended with the operations of constant selection, semijoin and antijoin described above. Then, we have:

Theorem 2. *Let Q be an RA^{ext} query whose syntax tree is such that:*

(a) *each ε node satisfies* NNC*;*
(b) *each ∩ and − node satisfies* DJN*;*
(c) *each × node satisfies* NNA *or* RDS*;*
(d) *each ∪ node satisfies* NNC *or* DJB *or* NNA *or* RDS*.*

Then, Q preserves Codd semantics.

An analog of Corollary 1 follows, and simple modifications to the algorithm of Proposition 3 show that these conditions are still linear-time testable.

To prove the result, we first show that the sql operation on tables distributes over semijoin and antijoin.

Lemma 5. *Let T_1 and T_2 be tables with disjoint signatures and θ be a selection condition over $\mathsf{sig}(T_1) \cup \mathsf{sig}(T_2)$. Then,*

(a) $\mathsf{sql}(T_1 \ltimes_\theta T_2) = \mathsf{sql}(T_1) \ltimes_\theta \mathsf{sql}(T_2)$*;*
(b) $\mathsf{sql}(T_1 \overline{\ltimes}_\theta T_2) = \mathsf{sql}(T_1) \overline{\ltimes}_\theta \mathsf{sql}(T_2)$*.*

Proof. We only prove (a); the proof of (b) is analogous. To this end, let r be a record over $\mathsf{sig}(T_1)$. Then, we have:

$$m = \#\big(r, \mathsf{sql}(T_1) \ltimes_\theta \mathsf{sql}(T_2)\big)$$

$$= \begin{cases} \#\big(r, \mathsf{sql}(T_1)\big) & \text{if } \exists\, r' \in \mathsf{sql}(T_2)\colon [\![\theta]\!]_{r\times r'} = \mathbf{t} \\ 0 & \text{otherwise} \end{cases}$$

$$= \begin{cases} \displaystyle\sum_{s\in \mathsf{sql}^{-1}(r)} \#(s, T_1) & \text{if } \exists\, r' \in \mathsf{sql}(T_2)\colon [\![\theta]\!]_{r\times r'} = \mathbf{t} \\ 0 & \text{otherwise} \end{cases}$$

and

$$n = \#\big(r, \mathsf{sql}(T_1 \ltimes_\theta T_2)\big) = \sum_{s\in \mathsf{sql}^{-1}(r)} \#(s, T_1 \ltimes_\theta T_2)$$

$$= \begin{cases} \displaystyle\sum_{s\in \mathsf{sql}^{-1}(r)} \#(s, T_1) & \text{if } \exists\, s' \in T_2\colon [\![\theta]\!]_{s\times s'} = \mathbf{t} \\ 0 & \text{otherwise} \end{cases}$$

Obviously, for every $r' \in \mathsf{sql}(T_2)$ there is $s' \in T_2$ such that $r' = \mathsf{sql}(s')$, and for every $s' \in T_2$ there exists $r' \in \mathsf{sql}(T_2)$ such that $r' = \mathsf{sql}(s')$. Moreover, for records $s \in \mathsf{sql}^{-1}(r)$ and $r' = \mathsf{sql}(s')$, by Lemma 1 we have that

$$[\![\theta]\!]_{r\times r'} = [\![\theta]\!]_{\mathsf{sql}(r\times r')} = [\![\theta]\!]_{\mathsf{sql}(r)\times \mathsf{sql}(r')} = [\![\theta]\!]_{s\times s'}$$

and therefore $m = n$. □

Next, we can show that:

Proposition 5. *Every RA^{ext} query whose syntax tree satisfies conditions (a) and (b) of Theorem 2 satisfies (1a) on all databases (not only Codd ones).*

Proof. This is the same as the proof of Proposition 4, with the only difference that, in the induction, we additionally need to address the cases for the operations of RA^{ext} that are not in RA. Constant selection is simply a special case of selection and so there is nothing new to show. The cases for semijoin and antijoin follow directly from the inductive hypothesis and Lemma 5 above. □

Finally, we can prove Theorem 2 by adapting the proof of Theorem 1.

Proof of Theorem 2. Let Q be an RA^{ext} query whose syntax tree satisfies the conditions of Theorem 2. By Proposition 5, Q satisfies (1a) on all databases, so we only need to show that $[\![Q]\!]_D$ is a Codd table for every Codd database D. To this end, we proceed by induction on the structure of Q as in the proof of Theorem 1, with the only difference that we additionally need to show also the following cases for the inductive step:

- Q is $\sigma_{\mathsf{const}(\alpha)}(Q_1)$.

  There is nothing new to show here, since this is a special case of $\sigma_\theta(Q_1)$ where $\theta = \bigwedge_{A\in\alpha} \mathsf{const}(A)$.

- Q is Q_1 op Q_2 for op $\in \{\ltimes_\theta, \overline{\ltimes}_\theta\}$

  The claim follows from the fact that $[\![Q]\!]_D \subseteq [\![Q_1]\!]_D$ and, by the induction hypothesis, $[\![Q_1]\!]_D$ is a Codd table. □

6. Set semantics

Without changing our underlying data model based on bags, we now discuss the case of queries interpreted under set semantics. To this end, we let RA^{set} be the fragment of RA without duplicate elimination, and we call *set database* one where no record occurs more than once within a table. The *set semantics* of an RA^{set} query Q on a set database D is denoted by $[\![Q]\!]^{set}_D$ and is defined inductively as follows:

$$[\![R]\!]^{set}_D \stackrel{\text{def}}{=} [\![R]\!]_D$$

$$[\![Q_1 \text{ op } Q_2]\!]^{set}_D \stackrel{\text{def}}{=} [\![Q_1]\!]^{set}_D \text{ op } [\![Q_2]\!]^{set}_D \text{ for op } \in \{\cap, -, \times\}$$

$$[\![Q_1 \cup Q_2]\!]^{set}_D \stackrel{\text{def}}{=} \varepsilon\big([\![Q_1]\!]^{set}_D \cup [\![Q_2]\!]^{set}_D\big)$$

$$[\![\pi_\alpha(Q')]\!]^{set}_D \stackrel{\text{def}}{=} \varepsilon\big(\pi_\alpha([\![Q']\!]^{set}_D)\big)$$

$$[\![\rho_{A\to B}(Q')]\!]^{set}_D \stackrel{\text{def}}{=} \rho_{A\to B}([\![Q']\!]^{set}_D)$$

$$[\![\sigma_\theta(Q')]\!]^{set}_D \stackrel{\text{def}}{=} \sigma_\theta([\![Q']\!]^{set}_D)$$

The above is well-defined because D is a set database and, even though the semantics of each subexpression returns a table that is formally a bag, each record in it is guaranteed to occur exactly once.

The notion of preservation of Codd semantics for RA^{set} queries in the context of sets is modified as follows:

Definition 5. An RA^{set} query Q *preserves Codd semantics over sets* if, for every SQL *set database* D and for every $D' \in \mathsf{codd}(D)$, it holds that $[\![Q]\!]^{set}_{D'} \in \mathsf{codd}\big([\![Q]\!]^{set}_D\big)$.

From the definition of $[\![\cdot]\!]^{set}$, it is immediate to see that every RA^{set} query Q can always be rewritten into an RA query Q' with duplicate elimination, so that, on every set database D, evaluating Q under set semantics is equivalent to evaluating Q' under bag semantics, i.e., $[\![Q]\!]^{set}_D = [\![Q']\!]_D$. This translation – shown in Fig. 7 for completeness – is straightforward: each projection and union operation must be immediately followed by duplicate elimination. Now, if Q' satisfies the condition of Theorem 1, then Q' preserves Codd semantics on all databases, in particular set ones; in turn, Q preserves Codd semantics over sets.

$R \mapsto R$	for any relation name R
$\pi_\alpha(Q) \mapsto \varepsilon\big(\pi_\alpha(Q')\big)$	if $Q \mapsto Q'$
$\sigma_\theta(Q) \mapsto \sigma_\theta(Q')$	if $Q \mapsto Q'$
$\rho_{A\to B}(Q) \mapsto \rho_{A\to B}(Q')$	if $Q \mapsto Q'$
Q_1 op $Q_2 \mapsto Q'_1$ op Q'_2	if $Q_1 \mapsto Q'_1$ and $Q_2 \mapsto Q'_2$ for op $\in \{\times, \cap, -\}$
$Q_1 \cup Q_2 \mapsto \varepsilon\big(Q'_1 \cup Q'_2\big)$	if $Q_1 \mapsto Q'_1$ and $Q_2 \mapsto Q'_2$

Fig. 7. Translation from RA^{set} to RA.

Proposition 6. *Let Q be an RA^{set} query, and let Q' be obtained from Q by applying the translation in Fig. 7. If Q' satisfies the conditions of Theorem 1, then Q preserves Codd semantics over sets.* □

However, this is too restrictive for even simple queries, as the following example shows.

Example 3. Consider a schema with R and S over a single attribute A that is nullable in R but not in S, and take the RA^{set} query $Q = R \cup S$. The translation of Fig. 7 yields $Q' = \varepsilon(R \cup S)$, which does not satisfy the conditions of Theorem 1, since $R \cup S$ is nullable and so $\varepsilon(R \cup S)$ does not satisfy `NNC` as required. Thus, we cannot use Proposition 6 to conclude that Q preserves Codd semantics over sets, although this is easily seen – in this particular example – because, on every SQL *set* database D, $[\![R]\!]_D$ will not have duplicates and $[\![S]\!]_D$ will not contain nulls.

This tells us that specific restrictions should be devised to deal with set semantics. To this end, we observe that the fundamental difference with the interpretation under bag semantics is that projection and union operations are immediately followed by duplicate elimination. As we know, this is problematic because records that differ only on nullable attributes will become the

same when the nulls are all replaced by **N**. Under set semantics, this can be allowed as long as nullable attributes are eventually discarded, which is enforced by requiring π_α and $\cup$ to satisfy the NNA condition. For $\cup$ we can further allow NNC as before, and this is indeed the reason why the query Q in Example 3 preserves Codd semantics over sets.

Theorem 3. *Let Q be an $\mathrm{RA}^{\mathrm{set}}$ query whose syntax tree is such that:*

(a) *each $\cap$ and $-$ node satisfies* DJN*;*
(b) *each $\times$ and π_α node satisfies* NNA*;*
(c) *each $\cup$ node satisfies* NNC *or* NNA*.*

Then, Q preserves Codd semantics over sets.

An analog of Corollary 1 follows, and straightforward modifications to the algorithm of Proposition 3 show that these conditions are still linear-time testable.

Note that, under set semantics, the DJB condition for $\cup$ does not work any longer, as the following example shows.

Example 4. Consider a schema with R and S over a single nullable attribute A. The $\mathrm{RA}^{\mathrm{set}}$ query $Q = R \cup S$ trivially satisfies DJB, but it does not preserve Codd semantics over sets. To see this, take the SQL set database D where

$$[\![R]\!]_D = \begin{array}{|c|}\hline A \\ \hline \mathbf{N} \\ \hline\end{array} \quad ; \quad [\![S]\!]_D = \begin{array}{|c|}\hline A \\ \hline \mathbf{N} \\ \hline\end{array}$$

and the Codd set database D' where

$$[\![R]\!]_{D'} = \begin{array}{|c|}\hline A \\ \hline \bot_1 \\ \hline\end{array} \quad ; \quad [\![S]\!]_{D'} = \begin{array}{|c|}\hline A \\ \hline \bot_2 \\ \hline\end{array}$$

Clearly, $D' \in \mathsf{codd}(D)$. However, we have:

$$[\![R \cup S]\!]^{\mathrm{set}}_D = \begin{array}{|c|}\hline A \\ \hline \mathbf{N} \\ \hline\end{array} \quad ; \quad [\![R \cup S]\!]^{\mathrm{set}}_{D'} = \begin{array}{|c|}\hline A \\ \hline \bot_1 \\ \bot_2 \\ \hline\end{array}$$

Therefore $[\![Q]\!]^{\mathrm{set}}_{D'} \notin \mathsf{codd}([\![Q]\!]^{\mathrm{set}}_D)$.

The restrictions of Theorem 3 for an $\mathrm{RA}^{\mathrm{set}}$ query Q are weaker than those of Theorem 1 one would have to impose on the RA translation of Q.

Proposition 7. *Let Q be an $\mathrm{RA}^{\mathrm{set}}$ query, and let Q' be obtained from Q by applying the translation in Fig. 7. If Q' satisfies the conditions of Theorem 1, then Q satisfies the conditions of Theorem 3.*

Proof. The only difference between Q and Q' is that every π_α and $\cup$ node in (the syntax tree of) Q will have as parent an ε node in (the syntax tree of) Q', and the parent of such ε node in Q' is in turn the parent of the corresponding π_α or $\cup$ node in Q.

Now, if Q' satisfies the conditions of Theorem 1, then each ε node in its syntax tree must satisfy NNA, and thus its only child is non-nullable. In turn, every π_α and $\cup$ node in Q will trivially satisfy NNA. □

As witnessed by the query in Example 3, the converse of Proposition 7 is obviously not true. This means that the conditions of Theorem 3 allow us to identify strictly more $\mathrm{RA}^{\mathrm{set}}$ queries that preserve Codd semantics over sets than Proposition 6, and without any increase in complexity.

We conclude this section by providing the formal proof of Theorem 3. Towards this goal, we first need a few lemmas about the interaction between duplicate elimination and other operations on tables. These follow immediately from the definitions.

Lemma 6. *Let T be a table. Then,*

(a) $\varepsilon\big(\sigma_\theta(T)\big) = \sigma_\theta\big(\varepsilon(T)\big)$ *for every condition θ such that* $\mathsf{sig}(\theta) \subseteq \mathsf{sig}(T)$*;*
(b) $\varepsilon\big(\rho_{A\to B}(T)\big) = \rho_{A\to B}\big(\varepsilon(T)\big)$ *for every A and B such that* $A \in \mathsf{sig}(T)$ *and* $B \notin \mathsf{sig}(T) - \{A\}$*;*
(c) $\varepsilon\big(\pi_\alpha(\varepsilon(T))\big) = \varepsilon\big(\pi_\alpha(T)\big)$ *for every* $\alpha \subseteq \mathsf{sig}(T)$*;*
(d) $\varepsilon\big(\mathsf{sql}(T)\big) = \varepsilon\big(\mathsf{sql}(\varepsilon(T))\big)$.

Lemma 7.

(a) $\varepsilon(T_1 \times T_2) = \varepsilon(T_1) \times \varepsilon(T_2)$ *for all tables T_1, T_2 with disjoint signatures.*
(b) $\varepsilon(T_1 \cap T_2) = \varepsilon(T_1) \cap \varepsilon(T_2)$ *for all tables T_1, T_2 of the same signature.*
(c) $\varepsilon(T_1 \cup T_2) = \varepsilon\big(\varepsilon(T_1) \cup \varepsilon(T_2)\big)$ *for all tables T_1, T_2 of the same signature.*
(d) $\varepsilon(T_1) - T_2 = \varepsilon(T_1) - \varepsilon(T_2)$ *for all tables T_1, T_2 of the same signature.*

Lemma 8. *Let T_1', T_1'' and T_2 be tables of the same signature and such that T_1'' is disjoint with T_1' and T_2. Then,*

$$\varepsilon\big((T_1' \cup T_1'') - T_2\big) = \varepsilon(T_1' \cup T_1'') - T_2$$

Next, we show a technical result that will be crucial in the proof of Theorem 3.

Lemma 9. *Let Q be an $\mathrm{RA}^{\mathrm{set}}$ query whose syntax tree satisfies condition (a) of Theorem 3. Then, for all set databases D and D' such that $D' = \mathsf{sql}(D)$, it holds that*

$$\varepsilon\big(\mathsf{sql}([\![Q]\!]^{\mathrm{set}}_D)\big) = [\![Q]\!]^{\mathrm{set}}_{D'}$$

Proof. We proceed by induction on the structure of Q.

Base: Q is a relation name R. Then, obviously, $[\![R]\!]^{\mathrm{set}}_{D'} = [\![R]\!]_{D'} = \mathsf{sql}([\![R]\!]_D) = \mathsf{sql}([\![R]\!]^{\mathrm{set}}_D) = \varepsilon\big(\mathsf{sql}([\![R]\!]^{\mathrm{set}}_D)\big)$ because D and D' are set databases such that $D' = \mathsf{sql}(D)$.

Induction:

- $\boxed{Q \text{ is } \mathsf{op}_1(Q_1) \text{ for } \mathsf{op}_1 \in \{\sigma_\theta, \rho_{A\to B}\}}$

$$\begin{aligned}
[\![\mathsf{op}_1(Q_1)]\!]^{\mathrm{set}}_{D'} &= \mathsf{op}_1([\![Q_1]\!]^{\mathrm{set}}_{D'}) && \text{(by def. of } [\![]\!]^{\mathrm{set}}\text{)}\\
&= \mathsf{op}_1\ \big(\varepsilon(\mathsf{sql}([\![Q_1]\!]^{\mathrm{set}}_D))\big) && \text{(by the I.H.)}\\
&= \varepsilon\big(\mathsf{op}_1\ \big(\mathsf{sql}([\![Q_1]\!]^{\mathrm{set}}_D)\big)\big) && \text{(by Lemma 6)}\\
&= \varepsilon\big(\mathsf{sql}\big(\mathsf{op}_1([\![Q_1]\!]^{\mathrm{set}}_D)\big)\big) && \text{(by Lemma 2)}\\
&= \varepsilon\big(\mathsf{sql}\big([\![\mathsf{op}_1(Q_1)]\!]^{\mathrm{set}}_D\big)\big) && \text{(by def. of } [\![]\!]^{\mathrm{set}}\text{)}
\end{aligned}$$

- $\boxed{Q \text{ is } \pi_\alpha(Q_1)}$

$$\begin{aligned}
[\![\pi_\alpha(Q_1)]\!]^{\mathrm{set}}_{D'} &= \varepsilon\big(\pi_\alpha([\![Q_1]\!]^{\mathrm{set}}_{D'})\big) && \text{(by def. of } [\![]\!]^{\mathrm{set}}\text{)}\\
&= \varepsilon\big(\pi_\alpha\big(\varepsilon(\mathsf{sql}([\![Q_1]\!]^{\mathrm{set}}_D))\big)\big) && \text{(by the I.H.)}\\
&= \varepsilon\big(\pi_\alpha\big(\mathsf{sql}([\![Q_1]\!]^{\mathrm{set}}_D)\big)\big) && \text{(by Lemma 6)}\\
&= \varepsilon\big(\mathsf{sql}\big(\pi_\alpha([\![Q_1]\!]^{\mathrm{set}}_D)\big)\big) && \text{(by Lemma 2)}\\
&= \varepsilon\big(\mathsf{sql}\big(\varepsilon(\pi_\alpha([\![Q_1]\!]^{\mathrm{set}}_D))\big)\big) && \text{(by Lemma 6)}\\
&= \varepsilon\big(\mathsf{sql}\big([\![\pi_\alpha(Q_1)]\!]^{\mathrm{set}}_D\big)\big) && \text{(by def. of } [\![]\!]^{\mathrm{set}}\text{)}
\end{aligned}$$

- $\boxed{Q \text{ is } Q_1 \times Q_2}$

$$\begin{aligned}
[\![Q_1 \times Q_2]\!]^{\mathrm{set}}_{D'} &= [\![Q_1]\!]^{\mathrm{set}}_{D'} \times [\![Q_2]\!]^{\mathrm{set}}_{D'} && \text{(by def. of } [\![]\!]^{\mathrm{set}}\text{)}\\
&= \varepsilon\big(\mathsf{sql}([\![Q_1]\!]^{\mathrm{set}}_D)\big) \times \varepsilon\big(\mathsf{sql}([\![Q_2]\!]^{\mathrm{set}}_D)\big) && \text{(by the I.H.)}\\
&= \varepsilon\big(\mathsf{sql}([\![Q_1]\!]^{\mathrm{set}}_D) \times \mathsf{sql}([\![Q_2]\!]^{\mathrm{set}}_D)\big) && \text{(by Lemma 7)}\\
&= \varepsilon\big(\mathsf{sql}([\![Q_1]\!]^{\mathrm{set}}_D \times [\![Q_2]\!]^{\mathrm{set}}_D)\big) && \text{(by Lemma 3)}\\
&= \varepsilon\big(\mathsf{sql}([\![Q_1 \times Q_2]\!]^{\mathrm{set}}_D)\big) && \text{(by def. of } [\![]\!]^{\mathrm{set}}\text{)}
\end{aligned}$$

- $\boxed{Q \text{ is } Q_1 \cup Q_2}$

$$\begin{aligned}
[\![Q_1 \cup Q_2]\!]^{\text{set}}_{D'} &= \varepsilon\big([\![Q_1]\!]^{\text{set}}_{D'} \cup [\![Q_2]\!]^{\text{set}}_{D'}\big) && \text{(by def. of } [\![]\!]^{\text{set}}) \\
&= \varepsilon\big(\varepsilon(\mathsf{sql}([\![Q_1]\!]^{\text{set}}_D)) \cup \varepsilon(\mathsf{sql}([\![Q_2]\!]^{\text{set}}_D))\big) && \text{(by the I.H.)} \\
&= \varepsilon\big(\mathsf{sql}([\![Q_1]\!]^{\text{set}}_D) \cup \mathsf{sql}([\![Q_2]\!]^{\text{set}}_D)\big) && \text{(by Lemma 7)} \\
&= \varepsilon\big(\mathsf{sql}([\![Q_1]\!]^{\text{set}}_D \cup [\![Q_2]\!]^{\text{set}}_D)\big) && \text{(by Lemma 3)} \\
&= \varepsilon\big(\mathsf{sql}\big(\varepsilon([\![Q_1]\!]^{\text{set}}_D \cup [\![Q_2]\!]^{\text{set}}_D)\big)\big) && \text{(by Lemma 6)} \\
&= \varepsilon\big(\mathsf{sql}([\![Q_1 \cup Q_2]\!]^{\text{set}}_D)\big) && \text{(by def. of } [\![]\!]^{\text{set}})
\end{aligned}$$

- $\boxed{Q \text{ is } Q_1 \cap Q_2}$

$$\begin{aligned}
[\![Q_1 \cap Q_2]\!]^{\text{set}}_{D'} &= [\![Q_1]\!]^{\text{set}}_{D'} \cap [\![Q_2]\!]^{\text{set}}_{D'} && \text{(by def. of } [\![]\!]^{\text{set}}) \\
&= \varepsilon\big(\mathsf{sql}([\![Q_1]\!]^{\text{set}}_D)\big) \cap \varepsilon\big(\mathsf{sql}([\![Q_2]\!]^{\text{set}}_D)\big) && \text{(by the I.H.)} \\
&= \varepsilon\big(\mathsf{sql}([\![Q_1]\!]^{\text{set}}_D) \cap \mathsf{sql}([\![Q_2]\!]^{\text{set}}_D)\big) && \text{(by Lemma 7)}
\end{aligned}$$

Since $Q_1 \cap Q_2$ satisfies DJN by assumption, n-sig(Q_1) and n-sig(Q_2) are disjoint, so $[\![Q_1]\!]^{\text{set}}_D$ and $[\![Q_2]\!]^{\text{set}}_D$ satisfy the assumptions of Lemma 4. Clearly, this remains true for $\mathsf{sql}([\![Q_1]\!]^{\text{set}}_D)$ and $\mathsf{sql}([\![Q_2]\!]^{\text{set}}_D)$, and we obtain:

$$\begin{aligned}
&\varepsilon\big(\mathsf{sql}([\![Q_1]\!]^{\text{set}}_D) \cap \mathsf{sql}([\![Q_2]\!]^{\text{set}}_D)\big) \\
&= \varepsilon\big(\mathsf{sql}([\![Q_1]\!]^{\text{set}}_D \cap [\![Q_2]\!]^{\text{set}}_D)\big) && \text{(by Lemma 4)} \\
&= \varepsilon\big(\mathsf{sql}([\![Q_1 \cap Q_2]\!]^{\text{set}}_D)\big) && \text{(by def. of } [\![]\!]^{\text{set}})
\end{aligned}$$

- $\boxed{Q \text{ is } Q_1 - Q_2}$

$$\begin{aligned}
[\![Q_1 - Q_2]\!]^{\text{set}}_{D'} &= [\![Q_1]\!]^{\text{set}}_{D'} - [\![Q_2]\!]^{\text{set}}_{D'} && \text{(by def. of } [\![]\!]^{\text{set}}) \\
&= \varepsilon\big(\mathsf{sql}([\![Q_1]\!]^{\text{set}}_D)\big) - \varepsilon\big(\mathsf{sql}([\![Q_2]\!]^{\text{set}}_D)\big) && \text{(by the I.H.)} \\
&= \varepsilon\big(\underbrace{\mathsf{sql}([\![Q_1]\!]^{\text{set}}_D)}_{T_1}\big) - \underbrace{\mathsf{sql}([\![Q_2]\!]^{\text{set}}_D)}_{T_2} && \text{(by Lemma 7)}
\end{aligned}$$

Now, we let $T_1 = \mathsf{sql}([\![Q_1]\!]^{\text{set}}_D)$ and $T_2 = \mathsf{sql}([\![Q_2]\!]^{\text{set}}_D)$, and we partition T_1 into two tables T_1' and T_1'' such that T_1' consists of every occurrence of all and only the constant records in T_1. Since $Q_1 - Q_2$ satisfies DJN by assumption, n-sig(Q_1) and n-sig(Q_2) are disjoint. In turn, $[\![Q_1]\!]^{\text{set}}_D$ and $[\![Q_2]\!]^{\text{set}}_D$, and thus also T_1 and T_2, satisfy the assumptions of Lemma 4. Moreover, T_2 is disjoint with T_1'', and T_1'' is trivially disjoint with T_1', so these three tables satisfy the assumptions of Lemma 8. Then, we have:

$$\begin{aligned}
&\varepsilon\big(\overbrace{\mathsf{sql}([\![Q_1]\!]^{\text{set}}_D)}^{T_1' \cup T_1''}\big) - \overbrace{\mathsf{sql}([\![Q_2]\!]^{\text{set}}_D)}^{T_2} \\
&= \varepsilon\big(\mathsf{sql}([\![Q_1]\!]^{\text{set}}_D) - \mathsf{sql}([\![Q_2]\!]^{\text{set}}_D)\big) && \text{(by Lemma 8)} \\
&= \varepsilon\big(\mathsf{sql}([\![Q_1]\!]^{\text{set}}_D - [\![Q_2]\!]^{\text{set}}_D)\big) && \text{(by Lemma 4)} \\
&= \varepsilon\big(\mathsf{sql}([\![Q_1 - Q_2]\!]^{\text{set}}_D)\big) && \text{(by def. of } [\![]\!]^{\text{set}})
\end{aligned}$$

This concludes the proof of the lemma. □

Now, observe that Definition 5 can be equivalently reformulated as follows: an RA^{set} query Q preserves Codd semantics over sets if, for all set databases D and D' such that $D' = \mathsf{sql}(D)$ and D is a Codd database, it holds that

$$\mathsf{sql}\big([\![Q]\!]^{\text{set}}_D\big) = [\![Q]\!]^{\text{set}}_{D'} \tag{2a}$$

and

$$[\![Q]\!]^{\text{set}}_D \text{ is a Codd table.} \tag{2b}$$

With all of this in place, we can finally proceed to prove Theorem 3.

Proof of Theorem 3. Let Q be an RA^{set} query whose syntax tree is s.t. every $\cap$ and $-$ node satisfies DJN, every $\times$ and π_α node satisfies NNA, and every $\cup$ node satisfies NNC or NNA. Let D and D' be set databases such that $D' = \mathsf{sql}(D)$ and D is a Codd database. We need to show (2a) and (2b). By Lemma 9, $\mathsf{sql}\big(\varepsilon([\![Q]\!]^{\text{set}}_D)\big) = [\![Q]\!]^{\text{set}}_{D'}$, hence showing (2a) amounts to proving that $\mathsf{sql}([\![Q]\!]^{\text{set}}_D)$ is a set. We show this, along with (2b), by induction on the structure of Q.

Base:

- Q is non-nullable. Then, $[\![Q]\!]^{\text{set}}_D$ is a complete set, so it is a Codd table and $\mathsf{sql}([\![Q]\!]^{\text{set}}_D) = [\![Q]\!]^{\text{set}}_D$ is a set.
- Q is a relation name R. Then, $[\![Q]\!]^{\text{set}}_D = [\![R]\!]_D$, which is a Codd set table because D is a Codd set database by assumption. Moreover, as $D' = \mathsf{sql}(D)$ is a set database by assumption, we have that $\mathsf{sql}([\![R]\!]_D) = [\![R]\!]_{D'}$ is a set.

Induction: The cases when Q is $\pi_\alpha(Q_1)$ or $Q_1 \times Q_2$ are already covered in the base case, because the NNA condition implies that such queries are non-nullable. For the same reason, we need not consider the case when Q is $Q_1 \cup Q_2$ and satisfies NNA. Similarly, the case when Q is $Q_1 \cap Q_2$ is also covered in the base case, as the DJN condition for this query again implies non-nullability.

From the definition of $\mathsf{sql}(\cdot)$ on tables, we also have that $\mathsf{sql}(T) \subseteq \mathsf{sql}(T')$ for all tables T, T' of the same signature such that $T \subseteq T'$.

- $\boxed{Q \text{ is } \sigma_\theta(Q_1) \text{ or } Q_1 - Q_2}$

  Clearly $[\![Q]\!]^{\text{set}}_D \subseteq [\![Q_1]\!]^{\text{set}}_D$ and, in turn, we also have that $\mathsf{sql}([\![Q]\!]^{\text{set}}_D) \subseteq \mathsf{sql}([\![Q_1]\!]^{\text{set}}_D)$. By the induction hypotheses, $[\![Q_1]\!]_D$ is a Codd relation and $\mathsf{sql}([\![Q_1]\!]^{\text{set}}_D)$ is a set table, hence the claims follow.

- $\boxed{Q \text{ is } \rho_{A\to B}(Q_1)}$

  The claims trivially follow from the induction hypotheses, because $\rho_{A\to B}$ only changes the name of attribute A in each record in $[\![Q_1]\!]^{\text{set}}_D$ but not the value this attribute is mapped to.

- $\boxed{Q \text{ is } Q_1 \cup Q_2 \text{ and satisfies NNC}}$

  We have $[\![Q]\!]^{\text{set}}_D = \varepsilon([\![Q_1]\!]^{\text{set}}_D \cup [\![Q_2]\!]^{\text{set}}_D)$. As union is commutative, we assume w.l.o.g. that Q_1 is non-nullable, so $[\![Q_1]\!]^{\text{set}}_D$ is complete. We then partition $[\![Q_2]\!]^{\text{set}}_D$ into two tables T_2', T_2'' such that T_2' contains every occurrence of all and only the constant records in $[\![Q_2]\!]^{\text{set}}_D$. As D is a set database, $[\![Q_2]\!]^{\text{set}}_D$ is a set and, by the I.H., it is also a Codd table. Therefore, T_2'' is a Codd set table because $T_2'' \subseteq [\![Q_2]\!]^{\text{set}}_D$. Now, as T_2'' is a set disjoint with $[\![Q_1]\!]^{\text{set}}_D$ and T_1', we have

  $$[\![Q]\!]^{\text{set}}_D = \varepsilon([\![Q_1]\!]^{\text{set}}_D \cup T_2') \cup T_2''$$

  Let $T_1 = \varepsilon([\![Q_1]\!]^{\text{set}}_D \cup T_2')$; since T_1 is complete and T_2'' is a Codd table, $[\![Q]\!]^{\text{set}}_D$ is a Codd table too. Moreover, by Lemma 3 we get

  $$\mathsf{sql}([\![Q]\!]^{\text{set}}_D) = \mathsf{sql}(T_1) \cup \mathsf{sql}(T_2'')$$

  where $\mathsf{sql}(T_1) = T_1$ and $\mathsf{sql}(T_2'') \subseteq \mathsf{sql}([\![Q]\!]^{\text{set}}_2)$. By the induction hypotheses, $\mathsf{sql}([\![Q]\!]^{\text{set}}_2)$ is a set, hence $\mathsf{sql}(T_2'')$ is a set as well. Therefore, since T_1 and $\mathsf{sql}(T_2'')$ are disjoint sets, $\mathsf{sql}([\![Q]\!]^{\text{set}}_D)$ is also a set. □

7. Conclusions and future work

The main lesson to be learned from this work is that, contrary to what is commonly believed, Codd nulls do not properly model SQL nulls, as even simple queries may produce answers that break such interpretation. In turn, this also means that theoretical results devised within the general model of incompleteness with marked nulls are not directly applicable in practical scenarios where SQL is used, unless queries are appropriately restricted.

Unfortunately, the class of relational algebra queries for which the Codd interpretation of SQL nulls is preserved in answers cannot be captured by a syntactic fragment of the language. Thus, we can only hope for reasonable syntactic restrictions that are *sufficient* to guarantee preservation. In this paper we presented mild restrictions on the syntax tree of queries, which ensure preservation and are easy to check. One obvious direction for future research is to look for more refined restrictions that allow one to recognize an even larger set of queries preserving Codd semantics.

As we have shown for intersection, semijoins and antijoins, including derived operations to the query language, while redundant from the expressivity point of view, may allow for restrictions that are milder than the general ones. Thus, the query language could be tailored to specific applications, e.g., by adding operations such as division ($\div$), or by abstracting other commonly used query patterns.

Another possibility is to propagate nullable attributes in selections in a more refined way, along the lines of what we have done for constant selection. For example, we can be sure that the values of attributes A and B in the answer to $\sigma_{A=1\wedge B=2}(Q)$ will always be constants, independently of whether they are non-nullable in Q. On the other hand, we cannot say the same for $\sigma_{A=1\vee B=2}(Q)$, so this requires a careful handling of selection conditions.

Yet another opportunity for additional refinement may be offered by intersection and union, which are commutative and associative operations. While commutativity does not affect whether a query satisfies our restrictions or not, associativity does have an impact. For example, consider a schema with relation names R and S over attributes A and B, where A is nullable only in R and B is nullable only in S; then the query $(R\cup S)\cup(R\cap S)$ satisfies the conditions of Theorem 1, while $R\cup\big(S\cup(R\cap S)\big)$ does not. Moreover, there exist also queries that preserve Codd semantics but for which no permutation of the operands results in an expression that satisfies our restrictions; e.g., $R_A\cap R_B\cap R_C$ on a schema with relation names R_A, R_B and R_C over attributes A, B, C such that the only non-nullable attribute of each R_i is i, for $i\in\{A,B,C\}$. To overcome these limitations, the idea would be to view $\cap$ and $\cup$ in the syntax tree of queries as *variadic*, rather than binary. This could allow for appropriate restrictions to be imposed on a single node with ≥ 2 children, instead of multiple nodes with exactly two children.

Future work is not limited to further refinement of the restrictions that ensure preservation of Codd semantics. The notion of Codd database we adopted in this paper only allows for constant tuples to occur multiple times, as nulls cannot repeat. Relaxing this requirement to also allow for duplicates of non-constant tuples raises several interesting questions: How do we interpret multiple occurrences of a tuple with nulls in an SQL database? What is preservation of Codd semantics in this context? Do our restrictions still guarantee it? One possible approach would be to explicitly represent a table as a pair consisting of a set S of records and an associated multiplicity function that indicates the number of occurrences of each record of S within the table. Then, the notion of Codd preservation, and the conditions that enforce it, would probably be similar to those we used for the case of queries interpreted under set semantics.

There are then questions related to more practical aspects. For instance, how can we check that a query written in (an appropriate fragment of) SQL satisfies our restrictions? We could translate the query to RA (e.g., by using the translation in [8]) and then check whether the resulting expression satisfies the restrictions, but this of course depends on the particular translation one chooses. A better solution would be to devise explicit restrictions on the actual SQL syntax.

Finally, another interesting direction for future work is to implement marked nulls in SQL, instead of limiting the expressivity of queries in an attempt to fit a flawed model. It remains to be seen whether such an implementation is feasible using only Standard features of the SQL language and whether it requires custom extensions. In any case, an implementation of marked nulls does not detract from the work in this paper, which would still be relevant for closed or legacy systems that cannot be extended, or applications where the Standard must be strictly followed.

Acknowledgments

This work was partially supported by EPSRC [grant numbers EP/N023056/1, EP/M025268/1].

Appendix. Additional proofs

Proof of Lemma 2.

(a) Let T be a table and let $\alpha\subseteq\mathsf{sig}(T)$. Obviously, the tables $T_1=\mathsf{sql}\big(\pi_\alpha(T)\big)$ and $T_2=\pi_\alpha\big(\mathsf{sql}(T)\big)$ are well defined and $\mathsf{sig}(T_1)=\mathsf{sig}(T_2)=\mathsf{sig}(T)$.
Let r be a record over $\mathsf{sig}(T)$; we need to show that $\#(r,T_1)=\#(r,T_2)$. On the one hand, we have

$$\#\big(r,T_1\big) \;=\; \sum_{s\in\mathsf{sql}^{-1}(r)}\#\big(s,\pi_\alpha(T)\big) \;=\; \sum_{s':\mathsf{sql}(\pi_\alpha(s'))=r}\#\big(s',T\big)$$

because

$$\#\big(s,\pi_\alpha(T)\big) \;=\; \sum_{s':\pi_\alpha(s')=s}\#\big(s',T\big)$$

and $\mathsf{sql}(s)=r$ since $s\in\mathsf{sql}^{-1}(r)$. On the other hand,

$$\#\big(r,T_2\big) \;=\; \sum_{r':\pi_\alpha(r')=r}\#\big(r',\mathsf{sql}(T)\big) \;=\; \sum_{s':\pi_\alpha(\mathsf{sql}(s'))=r}\#\big(s',T\big)$$

because

$$\#\big(r',\mathsf{sql}(T)\big) \;=\; \sum_{s'\in\mathsf{sql}^{-1}(r')}\#\big(s',T\big) \;=\; \sum_{s':\mathsf{sql}(s')=r'}\#\big(s',T\big)$$

and r' is such that $\pi_\alpha(r')=r$.
By Lemma 1, we have that $\pi_\alpha\big(\mathsf{sql}(s')\big)=\mathsf{sql}\big(\pi_\alpha(s')\big)$, and therefore the claim follows.

(b) Let T be a table, and let θ be a selection condition such that $\mathsf{sig}(\theta)\subseteq\mathsf{sig}(T)$. Obviously, the tables $T_1=\mathsf{sql}\big(\sigma_\theta(T)\big)$ and $T_2=\sigma_\theta\big(\mathsf{sql}(T)\big)$ are well defined and $\mathsf{sig}(T_1)=\mathsf{sig}(T_2)=\mathsf{sig}(T)$.
Let r be a record over $\mathsf{sig}(T)$; we need to show that $\#(r,T_1)=\#(r,T_2)$. We have the following:

$$\begin{aligned}\#\big(r,T_1\big) &= \sum_{s\in\mathsf{sql}^{-1}(r)}\#\big(s,\sigma_\theta(T)\big)\\ &\overset{(\dagger)}{=} \begin{cases}\displaystyle\sum_{s\in\mathsf{sql}^{-1}(r)}\#\big(s,T\big) & \text{if } [\![\theta]\!]_r=\mathbf{t}\\ 0 & \text{if } [\![\theta]\!]_r=\mathbf{f}\end{cases}\\ &= \begin{cases}\#\big(r,\mathsf{sql}(T)\big) & \text{if } [\![\theta]\!]_r=\mathbf{t}\\ 0 & \text{if } [\![\theta]\!]_r=\mathbf{f}\end{cases}\\ &= \#\big(r,T_2\big)\end{aligned}$$

where $(\dagger)$ is because, for every record $s\in\mathsf{sql}^{-1}(r)$, it holds that $\mathsf{sql}(s)=r$ and so $[\![\theta]\!]_r=[\![\theta]\!]_s$ by Lemma 1, and

$$\#\big(s,\sigma_\theta(T)\big) \;=\; \begin{cases}\#\big(s,T\big) & \text{if } [\![\theta]\!]_s=\mathbf{t}\\ 0 & \text{if } [\![\theta]\!]_s=\mathbf{f}\end{cases}$$

by definition of the selection operation on tables.

(c) Let T be a table, let $A \in \mathsf{sig}(T)$ and let $B \notin \big(\mathsf{sig}(T) - \{A\}\big)$. Obviously, the tables $T_1 = \mathsf{sql}\big(\rho_{A\to B}(T)\big)$ and $T_2 = \rho_{A\to B}\big(\mathsf{sql}(T)\big)$ are well defined, and both have signature $\big(\mathsf{sig}(T) - \{A\}\big) \cup \{B\}$.
Let r be a record over $\mathsf{sig}(T_1)$; we need to show that $\#(r, T_1) = \#(r, T_2)$. On the one hand, we have

$$\#\big(r, T_1\big) = \sum_{s \in \mathsf{sql}^{-1}(r)} \#\big(s, \rho_{A\to B}(T)\big) = \sum_{s' : \mathsf{sql}(\rho_{A\to B}(s'))=r} \#\big(s', T\big)$$

as $\#\big(s, \rho_{A\to B}(T)\big) = \#(s', T)$, where $\rho_{A\to B}(s') = s$, and $\mathsf{sql}(s) = r$ since $s \in \mathsf{sql}^{-1}(r)$.
On the other hand, $\#(r, T_2) = \#\big(s, \mathsf{sql}(T)\big)$, where $\rho_{A\to B}(s) = r$. Thus,

$$\#\big(r, T_2\big) = \sum_{\substack{s' \in \mathsf{sql}^{-1}(s) \\ \rho_{A\to B}(s)=r}} \#\big(s', T\big) = \sum_{s' : \rho_{A\to B}(\mathsf{sql}(s'))=r} \#\big(s', T\big)$$

Then, since $\mathsf{sql}\big(\rho_{A\to B}(s')\big) = \rho_{A\to B}\big(\mathsf{sql}(s')\big)$ by Lemma 1, the claim follows. □

Proof of Lemma 3.

(a) Let T_1 and T_2 be tables of the same signature. Obviously, the tables $\mathsf{sql}(T_1 \cup T_2)$ and $\mathsf{sql}(T_1) \cup \mathsf{sql}(T_2)$ are well defined and they are both over $\mathsf{sig}(T_1) = \mathsf{sig}(T_2)$. Then, for every record r over $\mathsf{sig}(T_1)$, we have:

$$\begin{aligned}
\#\big(r, \mathsf{sql}(T_1 \cup T_2)\big) &\overset{\dagger}{=} \sum_{s \in \mathsf{sql}^{-1}(r)} \#(s, T_1 \cup T_2) \\
&\overset{\dagger\dagger}{=} \sum_{s \in \mathsf{sql}^{-1}(r)} \big(\#(s, T_1) + \#(s, T_2)\big) \\
&= \sum_{s \in \mathsf{sql}^{-1}(r)} \#(s, T_1) + \sum_{s \in \mathsf{sql}^{-1}(r)} \#(s, T_2) \\
&\overset{\dagger}{=} \#\big(r, \mathsf{sql}(T_1)\big) + \#\big(r, \mathsf{sql}(T_2)\big) \\
&\overset{\dagger\dagger}{=} \#\big(r, \mathsf{sql}(T_1) \cup \mathsf{sql}(T_2)\big)
\end{aligned}$$

where † are by definition of sql on tables, and †† are by definition of the union operation on tables.

(b) Let T_1 and T_2 be tables with disjoint signatures, and let $\alpha = \mathsf{sig}(T_1)$ and $\beta = \mathsf{sig}(T_2)$. Obviously, the tables $\mathsf{sql}(T_1 \times T_2)$ and $\mathsf{sql}(T_1) \times \mathsf{sql}(T_2)$ are well defined and both are over $\alpha \cup \beta$. Since α and β are disjoint, every record r over $\alpha \cup \beta$ union is the product of the projections of r on α and β, denoted for simplicity by r_α and r_β, respectively. By Lemma 1, $\mathsf{sql}(s_\alpha \times s_\beta) = \mathsf{sql}(s_\alpha) \times \mathsf{sql}(s_\beta)$; therefore, $s_\alpha \times s_\beta \in \mathsf{sql}^{-1}(r_\alpha \times r_\beta)$ if and only if $s_\alpha \in \mathsf{sql}^{-1}(r_\alpha)$ and $s_\beta \in \mathsf{sql}^{-1}(r_\beta)$. Then, for every record r over $\alpha \cup \beta$, we have:

$$\begin{aligned}
&\#\big(r_\alpha \times r_\beta,\ \mathsf{sql}(T_1 \times T_2)\big) \\
&\quad = \sum_{s \in \mathsf{sql}^{-1}(r_\alpha \times r_\beta)} \#(s, T_1 \times T_2) \\
&\quad = \sum_{s \in \mathsf{sql}^{-1}(r_\alpha \times r_\beta)} \big(\#(s_\alpha, T_1) \cdot \#(s_\beta, T_2)\big) \\
&\quad = \sum_{s_1 \in \mathsf{sql}^{-1}(r_\alpha)} \Big(\#(s_1, T_1) \cdot \sum_{s_2 \in \mathsf{sql}^{-1}(r_\beta)} \#(s_2, T_2)\Big) \\
&\quad = \#\big(r_\alpha, \mathsf{sql}(T_1)\big) \cdot \#\big(r_\beta, \mathsf{sql}(T_2)\big) \\
&\quad = \#\big(r_\alpha \times r_\beta,\ \mathsf{sql}(T_1) \times \mathsf{sql}(T_2)\big) \quad \square
\end{aligned}$$

Proof of Lemma 4. Let us denote by α the signature of T_1 and T_2.

(a) Clearly, the tables $\mathsf{sql}(T_1) - \mathsf{sql}(T_2)$ and $\mathsf{sql}(T_1 - T_2)$ are well defined and both have signature α. Let r be a record over α; to prove the claim we will show the following:

$$\begin{aligned}
\#\big(r, \mathsf{sql}(T_1 - T_2)\big) &= \sum_{s \in \mathsf{sql}^{-1}(r)} \#\big(s, T_1 - T_2\big) \\
&= \sum_{s \in \mathsf{sql}^{-1}(r)} \big(\#(s, T_1) \dot{-} \#(s, T_2)\big) \\
&\overset{(\dagger)}{=} \sum_{s \in \mathsf{sql}^{-1}(r)} \#(s, T_1) \dot{-} \sum_{s \in \mathsf{sql}^{-1}(r)} \#(s, T_2) \\
&= \#\big(r, \mathsf{sql}(T_1)\big) \dot{-} \#\big(r, \mathsf{sql}(T_2)\big) \\
&= \#\big(r, \mathsf{sql}(T_1) - \mathsf{sql}(T_2)\big)\ .
\end{aligned}$$

In particular, we only need to show (†), as the other equalities hold by definition of $-$ and sql on tables.
If r maps all attributes to constants, then $\mathsf{sql}^{-1}(r) = \{r\}$ and (†) holds, as both its l.h.s. and r.h.s. are trivially equal to $\#(r, T_1) \dot{-} \#(r, T_2)$ in this case. Thus, assume $r(A) \in \mathsf{Null}$ for some attribute A. By definition, every $s \in \mathsf{sql}^{-1}(r)$ is such that $\mathsf{sql}(s) = r$ and so $s(A) \in \mathsf{Null}$. In turn, by the assumption on T_1 and T_2, there cannot exist records s_1 and s_2 in $\mathsf{sql}^{-1}(r)$ such that $s_1 \in T_1$ and $s_2 \in T_2$. Therefore, the l.h.s. and r.h.s. of (†) will both be $\sum_{s \in \mathsf{sql}^{-1}(r)} \#(s, T_1)$ if some record in $\mathsf{sql}^{-1}(r)$ occurs in T_1, and they will both be 0 otherwise.

(b) Obviously, all of the tables $T_1 \cap T_2$, $\mathsf{sql}(T_1) \cap \mathsf{sql}(T_2)$ and $\mathsf{sql}(T_1 \cap T_2)$ are well defined and have signature α.
By assumption, there is no record that assigns a null to some attribute and occurs in both T_1 and T_2. Thus, $T_1 \cap T_2$ is complete and $T_1 \cap T_2 = \mathsf{sql}(T_1 \cap T_2)$.
Now, let r be a record over α; to prove that $T_1 \cap T_2 = \mathsf{sql}(T_1) \cap \mathsf{sql}(T_2)$, we will show the following:

$$\begin{aligned}
\#\big(r, T_1 \cap T_2\big) &= \min\Big\{\#\big(r, T_1\big), \#\big(r, T_2\big)\Big\} \\
&\overset{(*)}{=} \min\Big\{\overbrace{\sum_{s \in \mathsf{sql}^{-1}(r)} \#\big(s, T_1\big)}^{m}, \overbrace{\sum_{s \in \mathsf{sql}^{-1}(r)} \#\big(s, T_2\big)}^{n}\Big\} \\
&= \min\Big\{\#\big(r, \mathsf{sql}(T_1)\big), \#\big(r, \mathsf{sql}(T_2)\big)\Big\} \\
&= \#\big(r, \mathsf{sql}(T_1) \cap \mathsf{sql}(T_2)\big)\ .
\end{aligned}$$

In particular, we only need to show (∗), as the other equalities hold by definition of ∩ and sql on tables.
If r maps all attributes to constants, then $\mathsf{sql}^{-1}(r) = \{r\}$ and (∗) trivially holds. Thus, assume $r(A) \in \mathsf{Null}$ for some attribute A. Then, as $T_1 \cap T_2$ is complete, $r \notin T_1 \cap T_2$, i.e., the l.h.s. of (∗) is 0. By definition, every record $s \in \mathsf{sql}^{-1}(r)$ is such that $\mathsf{sql}(s) = r$ and so $s(A) \in \mathsf{Null}$. In turn, by the assumption on T_1 and T_2, there cannot exist records s_1 and s_2 in $\mathsf{sql}^{-1}(r)$ such that $s_1 \in T_1$ and $s_2 \in T_2$. Therefore, the r.h.s. of (∗) is also 0, as $m = 0$ whenever $n > 0$, and $n = 0$ whenever $m > 0$. □

References

[1] M. Lenzerini, Data integration: a theoretical perspective, in: ACM Symposium on Principles of Database Systems, PODS, 2002, pp. 233–246.
[2] A. Halevy, A. Rajaraman, J. Ordille, Data integration: The teenage years, in: VLDB, 2006, pp. 9–16.
[3] M. Arenas, P. Barceló, L. Libkin, F. Murlak, Foundations of Data Exchange, Cambridge University Press, 2014.
[4] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati, Tractable reasoning and efficient query answering in description logics: The *DL-lite* family, J. Autom. Reason. 39 (3) (2007) 385–429.

[5] R. Kontchakov, C. Lutz, D. Toman, F. Wolter, M. Zakharyaschev, The combined approach to ontology-based data access, in: IJCAI, 2011, pp. 2656–2661.
[6] L. Libkin, SQL's three-valued logic and certain answers, ACM Trans. Database Syst. 41 (1) (2016) 1:1–1:28.
[7] T. Imielinski, W. Lipski, Incomplete information in relational databases, J. ACM 31 (4) (1984) 761–791.
[8] P. Guagliardo, L. Libkin, A formal semantics of SQL queries, its validation, and applications, PVLDB 11 (1) (2017) 27–39.
[9] P. Guagliardo, L. Libkin, On the codd semantics of SQL nulls, in: AMW, in: CEUR Workshop Proceedings, 1912, CEUR-WS.org, 2017.
[10] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley, 1995.
[11] L. Libkin, Elements of Finite Model Theory, Springer, 2004.