Consistent Query Answering under Primary Keys: A Characterization of Tractable Queries

Jef Wijsen
University of Mons-Hainaut
Mons, Belgium
jef.wijsen@umh.ac.be

ABSTRACT

This article deals with consistent query answering to conjunctive queries under primary key constraints. The repairs of an inconsistent database **db** are obtained by selecting a maximum number of tuples from **db** without ever selecting two tuples that agree on their primary key. For a Boolean conjunctive query q, we are interested in the following question: does there exist a Boolean first-order query φ such that for every database **db**, φ evaluates to true on **db** if and only if q evaluates to true on every repair of **db**? We address this problem for acyclic conjunctive queries in which no relation name occurs more than once. Our results improve previous solutions that are based on Fuxman-Miller join graphs.

Categories and Subject Descriptors

H.2.3 [**Database Management**]: Languages—*query languages*; H.2.4 [**Database Management**]: Systems—*relational databases*

General Terms

Theory, Algorithms

Keywords

Conjunctive queries, consistent query answering, database repairing, primary key, query rewriting

1. INTRODUCTION

Database repairing and *consistent query answering* (CQA) have received much research attention ever since the seminal article by Arenas et al. in 1999 [2], as witnessed by invited talks at ICDT [6] and PODS [8]. The issue is defined relative to a relational database schema with a set of integrity constraints. A database **db** over this database schema may violate one or more of the integrity constraints. A *repair* **rep** of **db** is any database that satisfies the integrity constraints and that is maximally close to **db** according to some fixed distance criterion. Under most reasonable distance criteria, an inconsistent database can have multiple repairs. Then, for a fixed Boolean query q, $\mathsf{CQA}(q)$ is the following problem: given a database **db**, decide whether q evaluates to true on every repair of **db**.

In this article, we deal with primary key constraints, which are fundamental in the relational data model. Moreover, primary key violations naturally arise in modern data integration applications, when data from different databases are combined. Under primary key constraints, there is a single most natural way for defining the repairs of a (possibly inconsistent) database: every repair is obtained by selecting a maximal number of tuples from the database, without ever selecting two tuples that agree on their primary key.

It is known that consistent query answering under primary keys is generally intractable for conjunctive queries [7]. In particular, $\mathsf{CQA}(q_1)$ is **coNP**-complete for the Boolean conjunctive query q_1 given next:

$$q_1 \quad : \quad \exists x \exists y \exists z (R(\underline{x}, y) \wedge S(\underline{z}, y)) \ ,$$

in which primary key positions are underlined. Thus, it is understood that R (as well as S) is a relation name of arity 2 and that the first attribute is the primary key.

A significant research question is to characterize classes of queries for which consistent query answering is tractable. It is known, for example, that $\mathsf{CQA}(q_2)$ is tractable for the following query q_2 [11]:

$$q_2 \quad : \quad \exists x \exists y \exists z (R(\underline{x}, y) \wedge S(\underline{y}, z)) \ .$$

Notice the difference between both queries: the "join" variable y is a primary key in q_2, but not in q_1.

$\mathsf{CQA}(q_2)$ is not only tractable, but even first-order definable. Witness thereof is the following formula φ_2, which evaluates to true on any database **db** if and only if q_2 evaluates to true on every repair of **db**:

$$\varphi_2 \quad : \quad \exists x \exists y (R(\underline{x}, y) \wedge \forall y' (R(\underline{x}, y') \rightarrow \exists z (S(\underline{y'}, z)))) \ .$$

We call φ_2 a *consistent first-order rewriting* of q_2. Clearly, if q is a query with a consistent first-order rewriting φ, then $\mathsf{CQA}(q)$ is in **P** (because φ can be evaluated on **db** in polynomial time data complexity). Saying that q has a consistent first-order rewriting is tantamount to saying that $\mathsf{CQA}(q)$ is first-order definable.

First-order rewriting is an elegant approach to consistent query answering and is suited for implementation in practical systems [9]. Therefore, a significant task is: Characterize the queries that have a consistent first-order rewriting under primary key constraints. Fuxman and Miller commenced this task by defining the class $\mathcal{C}_{forest}$ [11] (the preceding conference article is [10]). Every query in $\mathcal{C}_{forest}$ is conjunctive and has a consistent first-order rewriting. There is an easy syntactic test to check whether a conjunctive query belongs to $\mathcal{C}_{forest}$.

Inspired by the work of Fuxman and Miller, we defined the class $\mathcal{C}_{rooted}$ [18], a subclass of conjunctive queries, together with a rewrite function that yields a consistent first-order rewriting for every query in $\mathcal{C}_{rooted}$. A query is called *rooted* if it belongs to $\mathcal{C}_{rooted}$. Thus, each rooted query has a consistent first-order rewriting. The class $\mathcal{C}_{rooted}$ (strictly) contains all Boolean queries in $\mathcal{C}_{forest}$. It is an open conjecture that every Boolean conjunctive query is either rooted or has no consistent first-order rewriting.

The definition of the class $\mathcal{C}_{rooted}$, however, is semantic and provides no practical means to check whether a conjunctive query belongs to $\mathcal{C}_{rooted}$. For practical purposes, a syntactic characterization of $\mathcal{C}_{rooted}$ is important. In this article, we show that membership of $\mathcal{C}_{rooted}$ can be decided for Boolean conjunctive queries q that simultaneously

- are acyclic (in the sense of [3]);
- contain no self joins (i.e. without repeated relation names); and
- satisfy some mild condition called key-awareness.

Key-awareness is a syntactic condition needed in the proof of Theorem 5; like acyclicity, it can only be falsified by queries with three or more atoms. The acyclicity condition and the absence of self joins simplify the theoretical development without rendering it practically irrelevant; this restriction is not unusual:

- The class $\mathcal{C}_{forest}$ also excludes self joins. It seems that determining the tractability and first-order definability of $\mathsf{CQA}(q)$ becomes more complicated if the same relation name can occur more than once in q.

  In previous work [18], we showed that for the Boolean query $q_0 = \exists x \exists y(R(\underline{x}, y) \wedge R(\underline{y}, c))$, where c is a constant, the set $\mathsf{CQA}(q_0)$ is in $\mathbf{P}$ but is not first-order definable. On the other hand, it is an open conjecture that the implication "$\mathsf{CQA}(q)$ not first-order definable $\Longrightarrow$ $\mathsf{CQA}(q)$ **coNP**-complete" is true for all Boolean conjunctive queries q *without* self joins.

- The class $\mathcal{C}_{forest}$ also imposes some acyclicity condition, which is however not defined in the same way as [3]. We show in this article (see Theorem 3) that the tree components that constitute a $\mathcal{C}_{forest}$ query are actually acyclic in the sense of [3].

Our membership test for (a syntactic restriction of) $\mathcal{C}_{rooted}$ pinpoints the reason why the class $\mathcal{C}_{forest}$ admits consistent first-order rewriting (see Theorem 3).

Finally, our membership test is used to show the following dichotomy result: a join of two distinct relations is either rooted or has no consistent first-order rewriting.

All results in this article are stated for Boolean queries. Nevertheless, as observed in [10] and as discussed in the upcoming journal version of [18], rewriting techniques for Boolean queries yield correct results for non-Boolean queries when free variables are treated as new constants.

The article thus extends the theory of CQA under primary keys by providing a new and more powerful syntactic characterization of conjunctive queries with a consistent first-order rewriting. It is organized as follows. Section 2 introduces notations and terminology, and Section 3 discusses related work. The class $\mathcal{C}_{rooted}$ is introduced in Section 4. The notion of *reifiable atom* is introduced and it is explained that it can be tested whether a query is rooted if it can be verified whether a query atom is reifiable. In Section 5, the syntactic construct of *reifiability-attack* is introduced. Section 6 shows that the absence of a reifiability-attack in a join tree is a sufficient condition for reifiability. Section 7 shows that for key-aware queries, the absence of a reifiability-attack is also a necessary condition for reifiability. Our results thus allow to check membership of $\mathcal{C}_{rooted}$ for key-aware acyclic conjunctive queries without self joins. Section 8 shows that if a join of two distinct relations is outside $\mathcal{C}_{rooted}$, then it has no consistent first-order rewriting. Section 9 concludes the article.

2. NOTATIONS AND TERMINOLOGY

A *symbol* is either a constant or a variable. If $\vec{x}$ is a sequence of symbols, then $\mathsf{vars}(\vec{x})$ is the set of variables that occur in $\vec{x}$.

Let V be a set of variables. A *valuation over* V is a total mapping θ from symbols to symbols such that for every variable $v \in V$, $\theta(v)$ is a constant; if s is a variable not in V or if s is a constant, then $\theta(s) = s$. If θ is a valuation over V and $Z \subseteq V$, then $\theta|_Z$ denotes the restriction of θ to Z; thus $\theta|_Z(x) = \theta(x)$ if $x \in Z$ and $\theta|_Z(x) = x$ if $x \notin Z$.

Key-equal atoms. A *database schema* is a finite set of *relation names*. Every relation name R has a unique *signature*, which is a pair $[n, k]$ with $n \geq k \geq 1$: n is the *arity* of the relation name and the coordinates $1, 2, \ldots, k$ make up the *primary key*. If R is a relation name with signature $[n, k]$, then $R(s_1, \ldots, s_n)$ is an *R-atom* (or simply atom), where each s_i is a constant or a variable ($1 \leq i \leq n$). Such an atom is commonly written as $R(\underline{\vec{x}}, \vec{y})$ where $\vec{x} = s_1, \ldots, s_k$ and $\vec{y} = s_{k+1}, \ldots, s_n$. An atom is *ground* if it contains no variables. All constructs that follow are defined relative to a fixed database schema.

A *database* (over a database schema) is a finite set $\mathbf{db}$ of ground atoms using only the relation names of the schema. Two ground atoms $R_1(\underline{\vec{a}_1}, \vec{b}_1), R_2(\underline{\vec{a}_2}, \vec{b}_2) \in \mathbf{db}$ are *key-equal* if $R_1 = R_2$ and $\vec{a}_1 = \vec{a}_2$. We write $[\![R_1(\underline{\vec{a}_1}, \vec{b}_1)]\!]_{\mathbf{db}}$ for the set containing each atom of $\mathbf{db}$ that is key-equal to $R_1(\underline{\vec{a}_1}, \vec{b}_1)$.

Repair. A database $\mathbf{db}$ is *consistent* if it does not contain two distinct atoms that are key-equal. Thus, $\mathbf{db}$ is consistent

$$
\begin{aligned}
q_1 &= R(\underline{x}, y), S(\underline{z}, y)\\
q_2 &= R(\underline{x}, y), S(\underline{y}, z)\\
q_3 &= R_0(\underline{z}), R_1(\underline{x}, y, z), R_2(\underline{x, y}, z, u), R_3(\underline{x}, y), R_4(\underline{x, y}, u)\\
q_4 &= R(\underline{x}, y), S(\underline{x}, y)\\
q_5 &= R(\underline{x, y}), S(\underline{x}, y) \qquad (R \text{ is "all-key"})\\
q_6 &= R_0(\underline{x}, y), R_1(\underline{u}, x, y), R_2(\underline{u}, x, y)\\
q_7 &= R_0(\underline{x}, y), R_1(\underline{u}, x, y), R_2(\underline{u}, y)
\end{aligned}
$$

Figure 1: List of example rules used in this article.

if for every atom $A \in \mathbf{db}$, $[\![A]\!]_{\mathbf{db}} = \{A\}$. A *repair* of a database $\mathbf{db}$ is a maximal (under set inclusion) consistent subset of $\mathbf{db}$.

Rules. As in [1, p. 41], the term *rule* will be used as a shorthand for *rule-based conjunctive query*. Moreover, all rules are understood to be Boolean. Thus we have the following definitions.

A *rule* is a finite set $q = \{R_1(\underline{\vec{x}_1}, \vec{y}_1), \dots, R_n(\underline{\vec{x}_n}, \vec{y}_n)\}$ of atoms. This rule is *satisfied* by a database $\mathbf{db}$, denoted $\mathbf{db} \models q$, if there exists a valuation θ over $\mathsf{vars}(\vec{x}_1\vec{y}_1 \dots \vec{x}_n\vec{y}_n)$ such that for each $i \in \{1, \dots, n\}$, $R_i(\underline{\theta(\vec{x}_i)}, \theta(\vec{y}_i)) \in \mathbf{db}$. Rules do not contain built-in predicates. We say that a rule q has a *self join* if some relation name occurs more than once in q.

For easy reference, Fig. 1 lists the rules that will serve as examples throughout this article.

Acyclic rules. A rule q is *acyclic* if it has a join tree [3]. A *join tree* for a rule q is a tree whose vertices are the atoms of q such that:

> *Connectedness Condition:* whenever the same variable x occurs in two atoms $R_i(\underline{\vec{x}_i}, \vec{y}_i)$ and $R_j(\underline{\vec{x}_j}, \vec{y}_j)$, then x occurs in each atom on the unique path linking $R_i(\underline{\vec{x}_i}, \vec{y}_i)$ and $R_j(\underline{\vec{x}_j}, \vec{y}_j)$.

The term *Connectedness Condition* appears in [12] and refers to the fact that the set of vertices in which x occurs induces a connected subtree. Notice that a join tree is an undirected graph. In certain proofs, some vertex of the join tree is chosen as the root, yielding a directed rooted join tree. It is common to label the edges of a join tree as follows: if E is an edge between $R_i(\underline{\vec{x}_i}, \vec{y}_i)$ and $R_j(\underline{\vec{x}_j}, \vec{y}_j)$, then E is labeled by the set of variables that occur in both $R_i(\underline{\vec{x}_i}, \vec{y}_i)$ and $R_j(\underline{\vec{x}_j}, \vec{y}_j)$. Given a join tree, we will write $R_i(\underline{\vec{x}_i}, \vec{y}_i) \overset{L}{\frown} R_j(\underline{\vec{x}_j}, \vec{y}_j)$ to indicate that there is an edge labeled L between vertices $R_i(\underline{\vec{x}_i}, \vec{y}_i)$ and $R_j(\underline{\vec{x}_j}, \vec{y}_j)$. This notation extends to paths.

Example 1. A join tree is shown in Fig. 2 (left). The path from the top to the bottom right vertex can be denoted $R_0(\underline{z}) \overset{\{z\}}{\frown} R_1(\underline{x}, y, z) \overset{\{x,y\}}{\frown} R_3(\underline{x}, y)$.

The class $\mathcal{C}_{forest}$. The original definition of $\mathcal{C}_{forest}$ covers both Boolean and non-Boolean queries [11]. The definitions hereafter are for Boolean queries only.

The *Fuxman-Miller join graph* of a rule q is a directed graph whose vertices are the atoms of q such that there is a directed edge from an atom $R_i(\underline{\vec{x}_i}, \vec{y}_i)$ to a different atom $R_j(\underline{\vec{x}_j}, \vec{y}_j)$ if some variable that occurs in $\vec{y}_i$ also occurs in the atom $R_j(\underline{\vec{x}_j}, \vec{y}_j)$.

The class $\mathcal{C}_{forest}$ is the class of rules q satisfying the following properties:

1. no relation name occurs more than once in q;
2. the Fuxman-Miller join graph of q is a directed forest; and
3. *Full-join Condition:* whenever the Fuxman-Miller join forest of q contains a directed edge from $R_i(\underline{\vec{x}_i}, \vec{y}_i)$ to $R_j(\underline{\vec{x}_j}, \vec{y}_j)$, then every variable that occurs in $\vec{x}_j$ also occurs in $\vec{y}_i$.

It is important not to confuse classical join trees [3] and Fuxman-Miller join graphs. Fig. 2 illustrates the difference for the rule q_3. Since the Fuxman-Miller join graph is not a directed forest, it follows $q_3 \notin \mathcal{C}_{forest}$. On the other hand, in Example 12, we will show $q_3 \in \mathcal{C}_{rooted}$.

In this article, we denote by $\mathcal{C}_{tree}$ the class containing every query of $\mathcal{C}_{forest}$ whose Fuxman-Miller join graph is a (connected) directed tree.[1]

Consistently true. A rule q is *consistently true* in database $\mathbf{db}$, denoted $\mathbf{db} \models_{\mathsf{sure}} q$, if for every repair $\mathbf{rep}$ of $\mathbf{db}$, $\mathbf{rep} \models q$. The problem $\mathsf{CQA}_\Sigma(q)$, where Σ is a database schema and q is a rule, is the complexity of (testing membership of) the set:

$$\mathsf{CQA}_\Sigma(q) = \{\mathbf{db} \mid \mathbf{db} \text{ is a database over } \Sigma \text{ and } \mathbf{db} \models_{\mathsf{sure}} q\} \; .$$

Throughout this article, the schema Σ will be implicitly understood and therefore omitted.

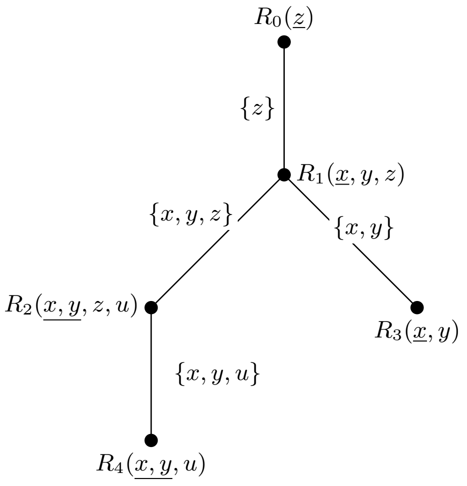
Consistent first-order rewriting. We say that a Boolean first-order query ψ is a *consistent first-order rewriting* of a rule q if for every database $\mathbf{db}$, $\mathbf{db} \models_{\mathsf{sure}} q$ if and only if $\mathbf{db} \models \psi$. Thus, a rule q has a consistent first-order rewriting if and only if $\mathsf{CQA}(q)$ is first-order definable.

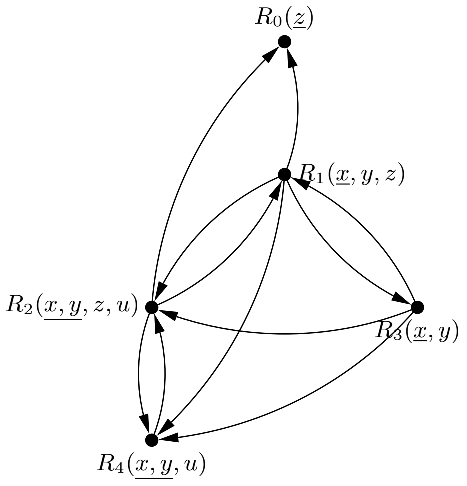
3. RELATED WORK

Under primary key constraints, each repair is a maximal consistent subset of the original database. In the case of primary keys, it makes no difference whether maximality is expressed relative to set inclusion (as in [2]) or cardinality (as in [16]). Inserting new tuples is useless for restoring primary key violations. Tuple modifications, as proposed in [17], are not considered in this article.

The idea of consistent query rewriting first appeared in [2]. Fuxman and Miller [11] have realized a number of breakthroughs in the consistent first-order rewriting of queries

[1]Caveat: This class is not the same as the class $\mathcal{C}_{tree}$ in Fuxman and Miller's original conference article [10]. The definition of $\mathcal{C}_{tree}$ in [10] does not require that the Fuxman-Miller join graph be connected.





$R_0(\underline{z})$
$R_1(\underline{x}, y, z)$
$R_2(\underline{x, y}, z, u)$
$R_3(\underline{x}, y)$
$R_4(\underline{x, y}, u)$

Figure 2: Join tree (left) and Fuxman-Miller join graph (right) for the same rule q_3.

under primary key constraints, which motivated the ConQuer system [9]. They introduced the class $\mathcal{C}_{forest}$ [11] and showed that every query in $\mathcal{C}_{forest}$ has a consistent first-order rewriting. Fuxman and Miller also proved that relaxations of the conditions imposed by $\mathcal{C}_{forest}$ easily lead to intractability of consistent query answering. In particular, Lemma 6 in [11] exhibits a rule q_0 that contains no self join and whose Fuxman-Miller join graph is a directed forest, such that $\mathsf{CQA}(q_0)$ is intractable. That rule q_0 violates the *Full-join Condition*.

The results of Fuxman and Miller have been generalized and extended to exclusion dependencies by Grieco et al. [13] and to unions of conjunctive queries by Lembo et al. [14].

In [18], we defined the class of rooted queries (denoted $\mathcal{C}_{rooted}$ in the current article) and we showed that every query in that class has a consistent first-order rewriting. The class $\mathcal{C}_{rooted}$ strictly contains $\mathcal{C}_{forest}$. A difficulty with the class $\mathcal{C}_{rooted}$ is that its definition is semantic and provides no syntactic test to check whether a conjunctive query is rooted. In [18], we developed a number of ad hoc syntactic conditions that are sufficient for membership of $\mathcal{C}_{rooted}$.

Complexity results on consistent query answering for larger classes of constraints appear in [4, 7]. Calì et al. [5] study query rewriting under key and inclusion dependencies in a larger context of data integration.

4. ROOTED RULES

Definition 3 recalls the class of *rooted rules* [18], which is based on the notion of *reifiable atom*. Fuxman and Miller [11] were the first ones to observe that reifiable atoms are significant for consistent first-order rewriting (see their Example 7 and the paragraph preceding that example), though they did not use the term "reifiable." The term was coined in [18]. Roughly, reifiable atoms admit existential quantifiers in consistent first-order rewritings.

Definition 1. Let $q = \{R_1(\underline{\vec{x}_1}, \vec{y}_1), \dots, R_n(\underline{\vec{x}_n}, \vec{y}_n)\}$ be a rule and $1 \leq i \leq n$. We say that the atom $R_i(\underline{\vec{x}_i}, \vec{y}_i)$ *is reifiable in* q if for every database $\mathbf{db}$, if $\mathbf{db} \models_{\mathsf{sure}} q$, then there exists a valuation θ over $\mathsf{vars}(\vec{x}_i)$ such that $\mathbf{db} \models_{\mathsf{sure}} \theta(q)$.

Example 2. Take $q_1 = \{R(\underline{x}, y), S(\underline{z}, y)\}$. Neither $R(\underline{x}, y)$ nor $S(\underline{z}, y)$ is reifiable in q_1. Witness thereof is the database $\mathbf{db} = \{R(\underline{a}, 1), R(\underline{a}, 2), R(\underline{d}, 3), S(\underline{e}, 1), S(\underline{b}, 2), S(\underline{b}, 3)\}$, with four repairs:

$$\begin{aligned}
\mathbf{rep}_1 &= \{R(\underline{a}, 1), R(\underline{d}, 3), S(\underline{e}, 1), S(\underline{b}, 2)\}\\
\mathbf{rep}_2 &= \{R(\underline{a}, 1), R(\underline{d}, 3), S(\underline{e}, 1), S(\underline{b}, 3)\}\\
\mathbf{rep}_3 &= \{R(\underline{a}, 2), R(\underline{d}, 3), S(\underline{e}, 1), S(\underline{b}, 2)\}\\
\mathbf{rep}_4 &= \{R(\underline{a}, 2), R(\underline{d}, 3), S(\underline{e}, 1), S(\underline{b}, 3)\}
\end{aligned}$$

Since each repair satisfies q_1, we have $\mathbf{db} \models_{\mathsf{sure}} q_1$. The atom $R(\underline{x}, y)$ is not reifiable in q_1: for every valuation θ over $\{x\}$, there is some repair that does not satisfy $\theta(q_1)$. In particular, for $\theta_a = \{x \mapsto a\}$, we have $\mathbf{rep}_4 \not\models \theta_a(q_1)$; and for $\theta_d = \{x \mapsto d\}$, we have $\mathbf{rep}_1 \not\models \theta_d(q_1)$. By a similar reasoning, $S(\underline{z}, y)$ is not reifiable in q_1.

The definition of $\mathcal{C}_{rooted}$ relies on some linear order on the atoms in a rule.

Definition 2. An *ordered rule* is a finite sequence $q = \langle R_1(\underline{\vec{x}_1}, \vec{y}_1), \dots, R_n(\underline{\vec{x}_n}, \vec{y}_n)\rangle$ of atoms. Every construct that is defined for (non-ordered) rules straightforwardly carries over to ordered rules (through omission of the order).

Definition 3. *Rooted* ordered rules are recursively defined as follows:

1. The empty rule is rooted.
2. A nonempty ordered rule

$$q = \langle R_1(\underline{\vec{x}_1}, \vec{y}_1), R_2(\underline{\vec{x}_2}, \vec{y}_2), \dots, R_n(\underline{\vec{x}_n}, \vec{y}_n)\rangle$$

($n \geq 1$) is rooted if $R_1(\underline{\vec{x}_1}, \vec{y}_1)$ is reifiable in q and for every valuation θ over $\mathsf{vars}(\vec{x}_1\vec{y}_1)$, the shorter rule

$$\langle \theta(R_2(\underline{\vec{x}_2}, \vec{y}_2)), \dots, \theta(R_n(\underline{\vec{x}_n}, \vec{y}_n))\rangle$$

is rooted.

A (non-ordered) rule is called *rooted* if it is rooted for some linear ordering of its atoms. The class of rooted rules is denoted by $\mathcal{C}_{rooted}$.

Rooted rules may be cyclic and contain self joins. The second item in Def. 3 refers to reifiability, which is a purely semantic construct.

Example 3. The ordered rule $q_2 = \langle R(\underline{x}, y), S(\underline{y}, z)\rangle$ is rooted, because $R(\underline{x}, y)$ is reifiable in q_2 (this follows from later results in this article) and for all constants a and b, if $\theta = \{x \mapsto a, y \mapsto b\}$, then $\theta(S(\underline{y}, z)) = S(\underline{b}, z)$ is rooted, because an atom without variables in its primary key is obviously reifiable.

The rule q_1 of Example 2 is not rooted because neither of its atoms is reifiable.

THEOREM 1 ([18]). *If $q \in \mathcal{C}_{rooted}$, then $\mathsf{CQA}(q)$ is first-order definable. In other words, every rooted rule has a consistent first-order rewriting.*

Theorem 1 is an interesting and significant result. There remains a practical difficulty, however: Definition 3 of rooted rules provides no syntactic test to check whether a rule is rooted, because the second item in that definition is semantic in nature. Decidability of $\mathcal{C}_{rooted}$ has not been addressed in previous work.

As announced in Section 1, the remainder of of this article will deal with rules in which no relation name occurs more than once. The following theorem states that for rules without self joins, the condition in the second item of Def. 3 can be tested by inspecting only a single arbitrary valuation over $\mathsf{vars}(\vec{x}_1\vec{y}_1)$.

THEOREM 2. *Let q be an ordered rule in which no relation name occurs more than once. Let V be the set of variables that occur in q, and $X \subseteq V$. If $\theta(q)$ is rooted for some valuation θ over X, then $\mu(q)$ is rooted for every valuation μ over X.*

Consequently, let q be a rule without self join and let θ be a valuation over the variables in q. Then, q is rooted if and only if there exists an ordering $\langle R_1(\underline{\vec{x}_1}, \vec{y}_1), \dots, R_n(\underline{\vec{x}_n}, \vec{y}_n)\rangle$ of q's atoms such that for every $i \in \{1, \dots, n\}$, $\theta_i(R_i(\underline{\vec{x}_i}, \vec{y}_i))$ is reifiable in $\langle \theta_i(R_i(\underline{\vec{x}_i}, \vec{y}_i)), \dots, \theta_i(R_n(\underline{\vec{x}_n}, \vec{y}_n))\rangle$, where θ_i is θ restricted to $\mathsf{vars}(\vec{x}_1\vec{y}_1 \dots \vec{x}_{i-1}\vec{y}_{i-1})$. It follows that for rules in which no relation name occurs more than once, membership of $\mathcal{C}_{rooted}$ can be decided if the following problem can be decided:

Reifiability Problem: given a rule q and an atom $R_i(\underline{\vec{x}_i}, \vec{y}_i) \in q$, decide whether $R_i(\underline{\vec{x}_i}, \vec{y}_i)$ is reifiable in q.

Solutions to the *Reifiability Problem* will be developed in the remainder of this article.

5. REIFIABILITY-ATTACK

We introduce the notion of *reifiability-attack*. Roughly, if q is an acyclic rule with join tree τ, then a reifiability-attack against atom $R_i(\underline{\vec{x}_i}, \vec{y}_i)$ is a special path in τ that starts at $R_i(\underline{\vec{x}_i}, \vec{y}_i)$. It can be easily checked whether a given join tree contains a reifiability-attack against any given atom. Reifiability-attacks will give us a handle on the *Reifiability Problem*: if there is no reifiability-attack against an atom, then the atom is reifiable; while a reifiability-attack against an atom usually implies that the atom is not reifiable.

5.1 Key-closure

The presence in a rule of two distinct atoms with the same primary key variables needs special care. An example is the query q_4 shown next:

$$q_4 \quad : \quad \exists x \exists y (R(\underline{x}, y) \land S(\underline{x}, y)) \ .$$

The query q_4 is not in $\mathcal{C}_{forest}$ [11, page 628], yet is rooted and hence has a consistent first-order rewriting, which is quite straightforward:

$$\begin{aligned} \varphi_4 \quad : \quad & \exists x \exists y \Big(R(\underline{x}, y) \land S(\underline{x}, y) \\ & \land \forall y_1 (R(\underline{x}, y_1) \rightarrow y_1 = y) \\ & \land \forall y_2 (S(\underline{x}, y_2) \rightarrow y_2 = y) \Big) \end{aligned}$$

The subformulas $\forall y_1 (R(\underline{x}, y_1) \rightarrow y_1 = y)$ and $\forall y_2 (S(\underline{x}, y_2) \rightarrow y_2 = y)$ express that y is uniquely determined given x. We will say that y is in the *key-closure* of x, as defined next.

Definition 4. Let q be a rule in which no relation name occurs more than once. Let Z be a subset of the variables occurring in q. A *key-closure computation* of Z (relative to q) is a maximal sequence

$$\begin{array}{cl} Z = Z_0 & \\ \downarrow & R_1(\underline{\vec{x}_1}, \vec{y}_1), S_1(\underline{\vec{u}_1}, \vec{w}_1) \\ Z_1 & \\ \downarrow & R_2(\underline{\vec{x}_2}, \vec{y}_2), S_2(\underline{\vec{u}_2}, \vec{w}_2) \\ Z_2 & \\ \vdots & \\ \downarrow & R_n(\underline{\vec{x}_n}, \vec{y}_n), S_n(\underline{\vec{u}_n}, \vec{w}_n) \\ Z_n & \end{array}$$

where:

1. $Z_0 \subsetneq Z_1 \subsetneq Z_2 \dots \subsetneq Z_n$;
2. for each $i \in \{1, \dots, n\}$,
   (a) $R_i(\underline{\vec{x}_i}, \vec{y}_i), S_i(\underline{\vec{u}_i}, \vec{w}_i)$ are distinct atoms of q satisfying $\mathsf{vars}(\vec{x}_i) = \mathsf{vars}(\vec{u}_i) \subseteq Z_{i-1}$;
   (b) $Z_i = Z_{i-1} \cup (\mathsf{vars}(\vec{y}_i) \cap \mathsf{vars}(\vec{w}_i))$.

The last element Z_n is called the *result* of the key-closure computation. It is straightforward to show that two key-closure computations of Z must necessarily yield the same result, which will be denoted $[Z]^+$ and called the *key-closure* of Z. We will write $\mathsf{vars}^+(\vec{x})$ as shorthand for $[\mathsf{vars}(\vec{x})]^+$.

Example 4. Here is a key-closure computation of $\{x\}$ relative to the rule q_3 shown in Fig. 2.

$$\begin{tikzcd}
\{x\} \arrow[d, "{R_1(\underline{x}, y, z), R_3(\underline{x}, y)}"] \\
\{x, y\} \arrow[d, "{R_2(\underline{x, y}, z, u), R_4(\underline{x, y}, u)}"] \\
\{x, y, u\}
\end{tikzcd}$$

Thus, $[\{x\}]^+ = \{x, y, u\}$.

Remark 1. There is an apparent resemblance between key-closures and *fd closures* of sets of attributes [1, p. 165]. For a given rule q, let F be the set of functional dependencies constructed as follows. Whenever $R(\underline{\vec{x}}, \vec{y})$ and $S(\underline{\vec{u}}, \vec{w})$ are distinct atoms of q such that $\mathsf{vars}(\vec{x}) = \mathsf{vars}(\vec{u})$, then F contains the functional dependency $X \rightarrow Y$ with $X = \mathsf{vars}(\vec{x})$ and $Y = \mathsf{vars}(\vec{y}) \cap \mathsf{vars}(\vec{w})$. The key-closure of Z then coincides with the fd closure of Z relative to the set F of functional dependencies.

5.2 Reifiability-attack

Definition 5 refines reifiability (see Def. 1) from primary keys to arbitrary subsets of (key-closures of) primary keys. The motivation can be understood from the following example.

Example 5. The "all-key" atom $R(\underline{x, y})$ is not reifiable in the rule $q_5 = \{R(\underline{x, y}), S(\underline{x}, y)\}$, as witnessed by the database $\mathbf{db} = \{R(\underline{a, 1}), R(\underline{a, 2}), S(\underline{a}, 1), S(\underline{a}, 2)\}$ with two repairs:

$$\mathbf{rep}_1 = \{R(\underline{a, 1}), R(\underline{a, 2}), S(\underline{a}, 1)\}$$
$$\mathbf{rep}_2 = \{R(\underline{a, 1}), R(\underline{a, 2}), S(\underline{a}, 2)\}$$

Since both repairs satisfy q_5, we have $\mathbf{db} \models_{\mathsf{sure}} q_5$. The atom $R(\underline{x, y})$ is not reifiable in q_5, because there is no valuation θ over the entire primary key $\{x, y\}$ such that $\mathbf{db} \models_{\mathsf{sure}} \theta(q_5)$. On the other hand, the subset $\{x\}$ is reifiable in the sense that for each database $\mathbf{db}$, if $\mathbf{db} \models_{\mathsf{sure}} q_5$, then there exists a valuation μ over $\{x\}$ such that $\mathbf{db} \models_{\mathsf{sure}} \mu(q_5)$. For the example database $\mathbf{db}$, take $\mu = \{x \mapsto a\}$.

Definition 5. Let q be a rule containing the atom $R(\underline{\vec{x}}, \vec{y})$. Let $Z \subseteq \mathsf{vars}^+(\vec{x}) \cap \mathsf{vars}(\vec{x}\vec{y})$. We say that $[Z \mid R(\underline{\vec{x}}, \vec{y})]$ is *reifiable* in q if for every database $\mathbf{db}$, if $\mathbf{db} \models_{\mathsf{sure}} q$, then there exists a valuation θ over Z such that $\mathbf{db} \models_{\mathsf{sure}} \theta(q)$.

Conform with Def. 1, saying that an atom $R(\underline{\vec{x}}, \vec{y})$ is reifiable in q is shorthand for saying that $[X \mid R(\underline{\vec{x}}, \vec{y})]$ is reifiable in q with $X = \mathsf{vars}(\vec{x})$.

Example 6 illustrates that the set Z in Def. 5 can contain variables not contained in the primary key $\vec{x}$.

Example 6. From the beginning of Section 5.1, it is correct to conclude that in the rule $q_4 = \{R(\underline{x}, y), S(\underline{x}, y)\}$, not only $[\{x\} \mid R(\underline{x}, y)]$ but even $[\{x, y\} \mid R(\underline{x}, y)]$ is reifiable. Thus, for every database $\mathbf{db}$, if $\mathbf{db} \models_{\mathsf{sure}} q_4$, then there exist two constants a, b such that $\mathbf{db} \models_{\mathsf{sure}} R(\underline{a}, b), S(\underline{a}, b)$.

We can now define the syntactic construct of reifiability-attack, which will be the key to the *Reifiability Problem*.

Definition 6. Let q be an acyclic rule in which no relation name occurs more than once. Let $R(\underline{\vec{x}}, \vec{y})$ be an atom of q and $Z \subseteq \mathsf{vars}^+(\vec{x}) \cap \mathsf{vars}(\vec{x}\vec{y})$. Let τ be a join tree for q. A *reifiability-attack* against $[Z \mid R(\underline{\vec{x}}, \vec{y})]$ is a path in τ of the form

$$R_0(\underline{\vec{x}_0}, \vec{y}_0) \overset{L_1}{\frown} R_1(\underline{\vec{x}_1}, \vec{y}_1) \overset{L_2}{\frown} R_2(\underline{\vec{x}_2}, \vec{y}_2) \ldots \overset{L_n}{\frown} R_n(\underline{\vec{x}_n}, \vec{y}_n) \ ,$$

with $R(\underline{\vec{x}}, \vec{y}) = R_0(\underline{\vec{x}_0}, \vec{y}_0)$ such that:

1. $Z \not\subseteq \mathsf{vars}^+(\vec{x}_n)$; and
2. for each $i \in \{1, \ldots, n\}$, $L_i \not\subseteq \mathsf{vars}^+(\vec{x}_n)$.

Thus, Z and the labels on the path are not contained in the key-closure of the primary key of the last atom on the path. Recall from Section 2 that each edge label L_i is the set of variables that occur in both $R_{i-1}(\underline{\vec{x}_{i-1}}, \vec{y}_{i-1})$ and $R_i(\underline{\vec{x}_i}, \vec{y}_i)$.

A reifiability-attack against $[X \mid R_i(\underline{\vec{x}_i}, \vec{y}_i)]$ where X is the set of variables in the primary key $\vec{x}_i$ (i.e. $X = \mathsf{vars}(\vec{x}_i)$) is shortly called a reifiability-attack against $R_i(\underline{\vec{x}_i}, \vec{y}_i)$.

Example 7. For the rule q_3 in Fig. 2, we have $[\{x, y\}]^+ = \{x, y, u\}$. The path

$$R_0(\underline{z}) \overset{\{z\}}{\frown} R_1(\underline{x}, y, z) \overset{\{x,y,z\}}{\frown} R_2(\underline{x, y}, z, u)$$

in the join tree is a reifiability-attack against $[\{z\} \mid R_0(\underline{z})]$, because the primary key $\{z\}$ and the labels $\{z\}$ and $\{x, y, z\}$ are not fully contained in $[\{x, y\}]^+$. The path in the opposite direction, i.e. $R_2(\underline{x, y}, z, u) \overset{\{x,y,z\}}{\frown} R_1(\underline{x}, y, z) \overset{\{z\}}{\frown} R_0(\underline{z})$, cannot be a reifiability-attack because the label $\{z\}$ is included in the primary key of the R_0-atom.

Example 8. For the rule $q_5 = \{R(\underline{x, y}), S(\underline{x}, y)\}$, we have $[\{x\}]^+ = \{x\}$. The path $R(\underline{x, y}) \overset{\{x,y\}}{\frown} S(\underline{x}, y)$ is a reifiability-attack against $[\{y\} \mid R(\underline{x, y})]$, because $\{y\}$ and the label $\{x, y\}$ are not contained in $[\{x\}]^+$. The path is not a reifiability-attack against $[\{x\} \mid R(\underline{x, y})]$.

We now state a property concerning the absence of reifiability-attacks in rules of the class $\mathcal{C}_{tree}$.

THEOREM 3. *Let q be a rule in $\mathcal{C}_{tree}$. Let τ be the Fuxman-Miller join tree of q. Then, τ is a directed rooted join tree (join tree in the sense of [3]) such that for every atom $A \in q$, there exists no directed path π from A to one of its descendants such that π is a reifiability-attack against A.*

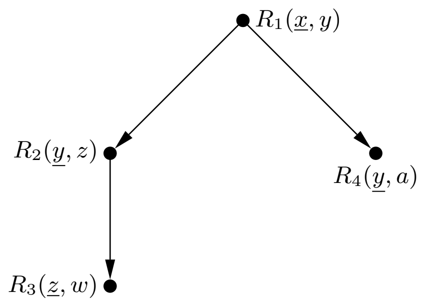


Figure 3: Fuxman-Miller join tree.

Example 9. This example illustrates Theorem 3. Fig. 3 shows the Fuxman-Miller join graph of a rule in $\mathcal{C}_{tree}$ (it is the rule called q_4 in [11]). The tree is also a directed rooted join tree (in the sense of [3]) because it satisfies the *Connectedness Condition* (edge labels have been omitted). No directed "downward" path is a reifiability-attack against the first atom on the path.

Theorem 3 together with the following Theorem 4 immediately leads to the inclusion $\mathcal{C}_{tree} \subseteq \mathcal{C}_{rooted}$, hence every rule in $\mathcal{C}_{tree}$ has a consistent first-order rewriting by Theorem 1. The inclusion $\mathcal{C}_{forest} \subseteq \mathcal{C}_{rooted}$ can be easily obtained using the observation that whenever two $\mathcal{C}_{tree}$ components of a $\mathcal{C}_{forest}$ query share a variable x, then x can only occur in the primary key of the root atoms of the component queries (see Lemma 2 in [11]).

5.3 Stringent Repairs

Let q be a rule containing a reifiable atom $R(\underline{\vec{x}}, \vec{y})$. Let **db** be a database such that $\mathbf{db} \models_{\mathsf{sure}} q$. Then by definition, there exists at least one valuation θ over $\mathsf{vars}(\vec{x})$ such that $\theta(q)$ is satisfied by every repair of **db**. In general, there exist other valuations μ over $\mathsf{vars}(\vec{x})$ such that $\mu(q)$ is satisfied by some, but not all, repairs. The following question arises: is there a repair **rep** such that for every valuation ω over $\mathsf{vars}(\vec{x})$, $\omega(q)$ is satisfied by **rep** only if $\omega(q)$ is satisfied by every repair of **db**? Such a repair **rep**, if it exists, will be called *stringent*.

Definition 7. Let q be a rule. Let $R(\underline{\vec{x}}, \vec{y})$ be an atom of q and $Z \subseteq \mathsf{vars}^+(\vec{x}) \cap \mathsf{vars}(\vec{x}\vec{y})$. Let **db** be a database. A repair **rep** of **db** is called *stringent for* $[Z \mid R(\underline{\vec{x}}, \vec{y})]$ *in* q if for every valuation θ over Z, if $\mathbf{rep} \models \theta(q)$, then $\mathbf{db} \models_{\mathsf{sure}} \theta(q)$.

Example 10. Let $q_0 = \{R(\underline{x}, x)\}$ and let $\mathbf{db} = \{R(\underline{a}, a), R(\underline{b}, b), R(\underline{b}, c)\}$. The two repairs are:

$$\mathbf{rep}_1 = \{R(\underline{a}, a), R(\underline{b}, b)\}$$
$$\mathbf{rep}_2 = \{R(\underline{a}, a), R(\underline{b}, c)\}$$

Let $\theta_a = \{x \mapsto a\}$ and $\theta_b = \{x \mapsto b\}$. Clearly, $\theta_a(q_0)$ evaluates to true on both repairs. On the other hand, $\theta_b(q_0)$ evaluates to true on $\mathbf{rep}_1$, but not on $\mathbf{rep}_2$. It is correct to conclude that $\mathbf{rep}_2$ is stringent, but $\mathbf{rep}_1$ is not.

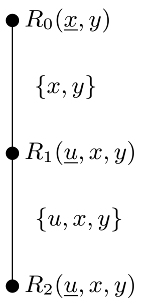


Figure 4: Join tree for the rule q_6 which has a consistent first-order rewriting.

6. ABSENCE OF REIFIABILITY-ATTACK

It remains to be shown that reifiability-attacks provide a sound and complete syntactic characterization of the semantic notion of reifiability, and hence allow to solve the *Reifiability Problem*:

1. If a join tree contains no reifiability-attack against $[Z \mid R(\underline{\vec{x}}, \vec{y})]$, then $[Z \mid R(\underline{\vec{x}}, \vec{y})]$ is reifiable.
2. If a join tree contains a reifiability-attack against $[Z \mid R(\underline{\vec{x}}, \vec{y})]$, then $[Z \mid R(\underline{\vec{x}}, \vec{y})]$ is not reifiable (we will only show this for key-aware rules).

This section shows the first item. The second item is handled in Section 7. The following lemma generalizes a result by Fuxman and Miller [11, p. 622].

Lemma 1. *Let q be an acyclic rule in which no relation name occurs more than once. Let $R(\underline{\vec{x}}, \vec{y})$ be an atom of q and $Z \subseteq \mathsf{vars}^+(\vec{x}) \cap \mathsf{vars}(\vec{x}\vec{y})$. Let τ be a join tree for q. If τ contains no reifiability-attack against $[Z \mid R(\underline{\vec{x}}, \vec{y})]$, then for every database* **db***, there exists a repair* **rep** *of* **db** *that is stringent for $[Z \mid R(\underline{\vec{x}}, \vec{y})]$ in q.*

Theorem 4. *Under the same assumptions of Lemma 1, if τ contains no reifiability-attack against $[Z \mid R(\underline{\vec{x}}, \vec{y})]$, then $[Z \mid R(\underline{\vec{x}}, \vec{y})]$ is reifiable in q.*

Proof. Assume τ contains no reifiability-attack against $[Z \mid R(\underline{\vec{x}}, \vec{y})]$. Assume $\mathbf{db} \models_{\mathsf{sure}} q$. By Lemma 1, we can construct a repair **rep** of **db** such that for every valuation θ over Z, $\mathbf{rep} \models \theta(q)$ implies $\mathbf{db} \models_{\mathsf{sure}} \theta(q)$. Since $\mathbf{db} \models_{\mathsf{sure}} q$ and since **rep** is a repair of **db**, $\mathbf{rep} \models q$. Hence, we can assume the existence of a valuation μ over Z such that $\mathbf{rep} \models \mu(q)$, and hence $\mathbf{db} \models_{\mathsf{sure}} \mu(q)$. ☐

Example 11. For the rule q_6 of Fig. 4, we have $[\{u\}]^+ = \{u, x, y\}$. The join tree contains no reifiability-attack against $R_0(\underline{x}, y)$ (recall that this is shorthand for: reifiability-attack against $[\{x\} \mid R_0(\underline{x}, y)]$). From Theorem 4, it follows that $R_0(\underline{x}, y)$ is reifiable.

Using Theorem 2, it is easy to see that the rule belongs to $\mathcal{C}_{rooted}$: if we replace x and y by arbitrary constants (say a_x and b_y) and drop the R_0-atom, we obtain the rule $\{R_1(\underline{u}, a_x, b_y), R_2(\underline{u}, a_x, b_y)\}$, which contains no reifiability-attacks against its atoms.

Since the rule q_6 belongs to $\mathcal{C}_{rooted}$, it has a consistent-first order rewriting. Note that rule q_6 is not in $\mathcal{C}_{forest}$ because its Fuxman-Miller join graph is cyclic.

Example 12. We show that the rule q_3 is rooted, using the join tree of Fig. 2. We have $[\{z\}]^+ = \{z\}$ and $[\{x\}]^+ = [\{x, y\}]^+ = \{x, y, u\}$. We show that the join tree contains no reifiability-attack against $R_2(\underline{x, y}, z, u)$. Consider each path that starts from $R_2(\underline{x, y}, z, u)$:

- $R_2(\underline{x, y}, z, u) \overset{\{x,y,u\}}{\frown} R_4(\underline{x, y}, u)$ is not a reifiability-attack (against $R_2(\underline{x, y}, z, u)$), because the label $\{x, y, u\}$ is contained in $[\{x, y\}]^+$.
- $R_2(\underline{x, y}, z, u) \overset{\{x,y,z\}}{\frown} R_1(\underline{x}, y, z)$ is not a reifiability-attack, because the primary key $\{x, y\}$ is contained in $[\{x\}]^+$.
- $R_2(\underline{x, y}, z, u) \overset{\{x,y,z\}}{\frown} R_1(\underline{x}, y, z) \overset{\{z\}}{\frown} R_0(\underline{z})$ is not a reifiability-attack, because the label $\{z\}$ is contained in $[\{z\}]^+$.
- $R_2(\underline{x, y}, z, u) \overset{\{x,y,z\}}{\frown} R_1(\underline{x}, y, z) \overset{\{x,y\}}{\frown} R_3(\underline{x}, y)$ is not a reifiability-attack, because the label $\{x, y\}$ is contained in $[\{x\}]^+$.

By Theorem 4, $R_2(\underline{x, y}, z, u)$ is reifiable in q_3. Let θ be any valuation over $\{x, y, z, u\}$. Let $q_3' = q_3 \setminus \{R_2(\underline{x, y}, z, u)\}$. Since $\theta(q_3')$ is variable-free, $\theta(q_3')$ is obviously rooted for any ordering of its atoms. By Theorem 2, it follows that q_3 is rooted.

7. PRESENCE OF REIFIABILITY-ATTACK

Theorem 4 expresses that an atom is reifiable if there is no reifiability-attack against it. We now show that under some mild condition, the inverse is also true, i.e. if an atom is reifiable, then there is no reifiability-attack against it. This mild condition is defined next.

Definition 8. Let q be an acyclic rule in which no relation name occurs more than once. We say that a join tree τ for q is *key-aware* if for all distinct atoms $R_i(\underline{\vec{x}_i}, \vec{y}_i), R_j(\underline{\vec{x}_j}, \vec{y}_j) \in q$ such that $\mathsf{vars}^+(\vec{x}_i) = \mathsf{vars}^+(\vec{x}_j)$,

1. *Intersection Condition:* if the same variable z occurs in both $R_i(\underline{\vec{x}_i}, \vec{y}_i)$ and $R_j(\underline{\vec{x}_j}, \vec{y}_j)$, then z belongs to $\mathsf{vars}^+(\vec{x}_i)$; and

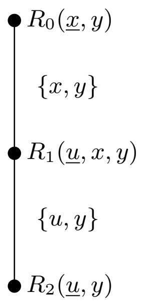


Figure 5: Key-aware join-tree for q_7 with a reifiability-attack against $R_0(\underline{x}, y)$.

2. *Key-connectedness Condition:* if some atom $R_k(\underline{\vec{x}_k}, \vec{y}_k)$ is on the unique path linking $R_i(\underline{\vec{x}_i}, \vec{y}_i)$ and $R_j(\underline{\vec{x}_j}, \vec{y}_j)$, then $\mathsf{vars}^+(\vec{x}_k) = \mathsf{vars}^+(\vec{x}_i)$. In other words, the set of atoms with the same key-closure of their primary key induces a (connected) subtree.

We say that q is *key-aware* if it has a key-aware join tree.

It is easy to deduce that a rule is key-aware if no two distinct atoms contain exactly the same primary key variables. If two atoms agree on their primary key variables, then they should not be separated in the join tree by an atom with a different primary key (unless that primary key has the same key-closure). Among the rules $q_1, \ldots, q_7$ in Fig. 1, only q_3 is not key-aware, as illustrated next.

Example 13. For the rule q_3 in Fig. 2, we have $[\{x\}]^+ = [\{x, y\}]^+ = \{x, y, u\}$ (see Example 4). The *Key-connectedness Condition* is satisfied by the join tree shown in Fig. 2. The *Intersection Condition* is not satisfied, because the variable z occurs in both $R_1(\underline{x}, y, z)$ and $R_2(\underline{x, y}, z, u)$, but $z \notin \{x, y, u\}$.

THEOREM 5. *Let q be a key-aware acyclic rule in which no relation name occurs more than once. Let $R(\underline{\vec{x}}, \vec{y})$ be an atom of q. Let $Z \subseteq \mathsf{vars}^+(\vec{x}) \cap \mathsf{vars}(\vec{x}\vec{y})$. Let τ be a key-aware join tree for q. If τ contains a reifiability-attack against $[Z \mid R(\underline{\vec{x}}, \vec{y})]$, then $[Z \mid R(\underline{\vec{x}}, \vec{y})]$ is not reifiable in q.*

Example 14. The rule q_7 shown in Fig. 5 is key-aware. We have $[\{u\}]^+ = \{u, y\}$. The path $R_0(\underline{x}, y) \overset{\{x,y\}}{\frown} R_1(\underline{u}, x, y)$ is a reifiability-attack against $R_0(\underline{x}, y)$, because the primary key $\{x\}$ and the label $\{x, y\}$ are not contained in $[\{u\}]^+$. It follows that $R_0(\underline{x}, y)$ is not reifiable. Compare with the lookalike rule q_6 of Fig. 4, in which $R_0(\underline{x}, y)$ is reifiable; the (absence of) variable x in the R_2-atom makes the difference.

THEOREM 6. *Given a key-aware acyclic rule q in which no relation name occurs more than once, it can be decided whether or not q belongs to $\mathcal{C}_{rooted}$.*

PROOF. Consequence of Theorems 2, 4, and 5. $\square$

8. QUERIES WITH ONE JOIN

The construct of reifiability-attack is useful to the membership problem for $\mathcal{C}_{rooted}$. A remaining task is to show the conjecture that rules outside $\mathcal{C}_{rooted}$ have no consistent first-order rewriting. In this section, we show this conjecture for rules with two atoms having distinct relation names. Since such rules are obviously acyclic and key-aware, Theorems 4 and 5 can be used to obtain the following syntactic characterization.

LEMMA 2. *Let $q = \{R(\underline{\vec{x}}, \vec{y}), S(\underline{\vec{u}}, \vec{w})\}$ with $R \neq S$. Then, $q \in \mathcal{C}_{rooted}$ if and only if one of the following four conditions is satisfied, where L is the set of variables that occur in both $R(\underline{\vec{x}}, \vec{y})$ and $S(\underline{\vec{u}}, \vec{w})$: $\mathsf{vars}(\vec{x}) \subseteq \mathsf{vars}(\vec{u})$, $L \subseteq \mathsf{vars}(\vec{u})$, $\mathsf{vars}(\vec{u}) \subseteq \mathsf{vars}(\vec{x})$, or $L \subseteq \mathsf{vars}(\vec{x})$.*

PROOF. (*Crux.*) The first two conditions express the absence of a reifiability-attack against $R(\underline{\vec{x}}, \vec{y})$; the last two conditions express the absence of a reifiability-attack against $S(\underline{\vec{u}}, \vec{w})$. $\square$

Rules q for which $\mathsf{CQA}(q)$ is **coNP**-complete appear in [7, 11]. Unless $\mathbf{P} = \mathbf{NP}$, a rule q can obviously have no consistent first-order rewriting if $\mathsf{CQA}(q)$ is **coNP**-complete. The following theorem shows that if a rule q, of size 2 and with distinct relation names, does not belong to $\mathcal{C}_{rooted}$, then it cannot have a consistent first-order rewriting, regardless of the complexity of $\mathsf{CQA}(q)$. The proof uses the notion of Hanf-locality [15].

THEOREM 7. *Let $q = \{R(\underline{\vec{x}}, \vec{y}), S(\underline{\vec{u}}, \vec{w})\}$ with $R \neq S$. Then, q has a consistent first-order rewriting if and only if $q \in \mathcal{C}_{rooted}$.*

We explain the general idea behind the only-if part of the proof (the if-part follows from Theorem 1). The explanation relies on the following graphical representation defined relative to a database $\mathbf{db}$ and a rule q without self join.

Definition 9. We define two hypergraphs whose vertices are the atoms of $\mathbf{db}$:

- the hyperedges of the *key-conflict hypergraph* are maximal sets of key-equivalent atoms; and
- the hyperedges of the *query-answer hypergraph* are the elements of the set:

$$\{\theta(q) \mid \theta \text{ valuation such that } \theta(q) \subseteq \mathbf{db}\} \ .$$

In a graphical representation, hyperedges of the key-conflict hypergraph are indicated by full contours, and hyperedges of the query-answer hypergraph by dotted contours.

Clearly, $\mathbf{db} \notin \mathsf{CQA}(q)$ if and only if there exists a set $\mathbf{rep} \subseteq \mathbf{db}$ such that for each hyperedge E:

1. if E is an hyperedge in the key-conflict hypergraph for $\mathbf{db}$, then $\mathbf{rep}$ contains exactly one atom of E; and
2. if E is an hyperedge in the query-answer hypergraph for $\mathbf{db}$, then $E \nsubseteq \mathbf{rep}$.

Now we return to the only-if part of Theorem 7. Assume a rule $q = \{R(\underline{\vec{x}}, \vec{y}), S(\underline{\vec{u}}, \vec{w})\}$ with $R \neq S$ such that $q \notin \mathcal{C}_{rooted}$. Using Lemma 2, one can show that for every integer $k \geq 0$, it is possible to construct two databases, $\mathbf{db}_{yes}$ and $\mathbf{db}_{no}$, that are indistinguishable by first-order sentences of quantifier rank k, such that $\mathbf{db}_{yes} \models_{\mathsf{sure}} q$ and $\mathbf{db}_{no} \not\models_{\mathsf{sure}} q$. Indistinguishability of $\mathbf{db}_{yes}$ and $\mathbf{db}_{no}$ follows, using Hanf's theorem, from the fact that $\mathbf{db}_{yes}$ and $\mathbf{db}_{no}$ locally look alike. The graphical representations of $\mathbf{db}_{yes}$ and $\mathbf{db}_{no}$ are shown in Fig. 6. Both databases contain the same number of atoms (represented by vertices) and the same number of constants (not shown); within each database, R-atoms are positioned at the left of S-atoms. The quintessence is that in constructing the atoms of $\mathbf{db}_{yes}$ and $\mathbf{db}_{no}$, we use new fresh constants wherever possible; a constant occurs more than once only if this is necessary for realizing the hyperedges shown in Fig. 6. The figure suggests that the local neighborhoods of (constants in) the atoms denoted by α in $\mathbf{db}_{yes}$ and $\mathbf{db}_{no}$, are isomorphic (similar for β, γ, δ). A repair of $\mathbf{db}_{no}$ is shown at the right: we have selected one vertex from each hyperedge of the key-conflict hypergraph of $\mathbf{db}_{no}$, without selecting two vertices that together satisfy q. It can be easily verified that no such vertex selection can be made in $\mathbf{db}_{yes}$.

9. CONCLUSION

Consistent query answering under primary key constraints is an important issue because, first, key constraints appear in all database applications and, second, the problem of duplicate key values arises in modern data integration applications. An elegant and practical approach to the problem is the technique known as (consistent) first-order query rewriting.

The semantic notion of *reifiable atom* seems fundamental in the consistent first-order rewriting of conjunctive queries under primary keys. The notion is implied in the class $\mathcal{C}_{forest}$ and its rewriting algorithm [11], and is explicit in the class $\mathcal{C}_{rooted}$ and its associated rewrite function [18]. All queries in $\mathcal{C}_{rooted}$ have a consistent first-order rewriting. The class $\mathcal{C}_{rooted}$ is defined in a semantic way and, up to now, decidability of $\mathcal{C}_{rooted}$ remained largely unexplored.

The new syntactic construct of *reifiability-attack* in join trees allows to decide reifiability of atoms in most acyclic conjunctive queries without repeated relation names:

1. the absence of a reifiability-attack against an atom always implies that the atom is reifiable; and
2. a reifiability-attack against an atom implies that the atom is not reifiable, unless the query is not key-aware.

The first result can be used to prove that a query belongs to $\mathcal{C}_{rooted}$. In particular, it allows to ascertain $\mathcal{C}_{forest} \subseteq \mathcal{C}_{rooted}$ (using Theorem 3), and to identify queries outside $\mathcal{C}_{forest}$

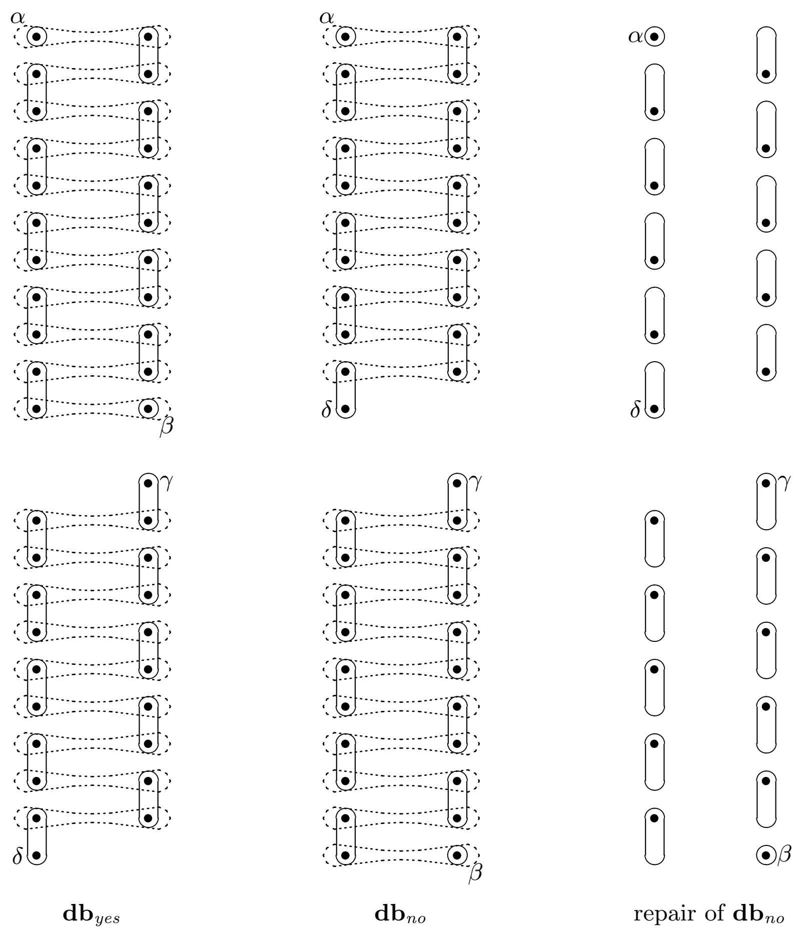


Figure 6: Key-conflict hypergraphs (full contours) and query-answer hypergraphs (dotted contours) of $\mathbf{db}_{yes} \in \mathsf{CQA}(q)$ and $\mathbf{db}_{no} \notin \mathsf{CQA}(q)$, used in the proof of Theorem 7.

with a consistent first-order rewriting (examples are q_3, q_4, and q_6). The notion of reifiability-attack is thus more powerful than the construct of Fuxman-Miller join graph as a tool for characterizing conjunctive queries that have a consistent first-order rewriting under primary keys. Furthermore, unlike $\mathcal{C}_{forest}$, our characterization does not require that nonkey-to-key joins be full.

The second result can be used to prove that a query does not belong to $\mathcal{C}_{rooted}$. In particular, a query q is not in $\mathcal{C}_{rooted}$ if it has a key-aware join tree that contains a reifiability-attack against each of its atoms (examples are q_1 and q_7).

These results imply that the following problem is decidable: given a key-aware acyclic conjunctive query q without repeated relation names, does q belong to $\mathcal{C}_{rooted}$? Key-awareness seems to be a mild condition satisfied by most queries.

A syntactic characterization for $\mathcal{C}_{rooted}$ is helpful for proving the conjecture that every Boolean conjunctive query is either rooted or has no consistent first-order rewriting. Using the notion of Hanf-locality, we were able to prove this conjecture for queries that are joins of two distinct relations. We are currently investigating how to generalize that proof to larger queries.

10. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[2] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79. ACM Press, 1999.

[3] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30(3):479–513, 1983.

[4] A. Calì, D. Lembo, and R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *PODS*, pages 260–271. ACM, 2003.

[5] A. Calì, D. Lembo, and R. Rosati. Query rewriting and answering under constraints in data integration systems. In G. Gottlob and T. Walsh, editors, *IJCAI*, pages 16–21. Morgan Kaufmann, 2003.

[6] J. Chomicki. Consistent query answering: Five easy pieces. In T. Schwentick and D. Suciu, editors, *ICDT*, volume 4353 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2007.

[7] J. Chomicki and J. Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Inf. Comput.*, 197(1-2):90–121, 2005.

[8] W. Fan. Dependencies revisited for improving data quality. In M. Lenzerini and D. Lembo, editors, *PODS*, pages 159–170. ACM, 2008.

[9] A. Fuxman, E. Fazli, and R. J. Miller. Conquer: Efficient management of inconsistent databases. In F. Özcan, editor, *SIGMOD Conference*, pages 155–166. ACM, 2005.

[10] A. Fuxman and R. J. Miller. First-order query rewriting for inconsistent databases. In T. Eiter and L. Libkin, editors, *ICDT*, volume 3363 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 2005.

[11] A. Fuxman and R. J. Miller. First-order query rewriting for inconsistent databases. *J. Comput. Syst. Sci.*, 73(4):610–635, 2007.

[12] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries. *J. Comput. Syst. Sci.*, 64(3):579–627, 2002.

[13] L. Grieco, D. Lembo, R. Rosati, and M. Ruzzi. Consistent query answering under key and exclusion dependencies: algorithms and experiments. In O. Herzog, H.-J. Schek, N. Fuhr, A. Chowdhury, and W. Teiken, editors, *CIKM*, pages 792–799. ACM, 2005.

[14] D. Lembo, R. Rosati, and M. Ruzzi. On the first-order reducibility of unions of conjunctive queries over inconsistent databases. In T. Grust, H. Höpfner, A. Illarramendi, S. Jablonski, M. Mesiti, S. Müller, P.-L. Patranjan, K.-U. Sattler, M. Spiliopoulou, and J. Wijsen, editors, *EDBT Workshops*, volume 4254 of *Lecture Notes in Computer Science*, pages 358–374. Springer, 2006.

[15] L. Libkin. *Elements of Finite Model Theory.* Springer, 2004.

[16] J. Lin and A. O. Mendelzon. Merging databases under constraints. *Int. J. Cooperative Inf. Syst.*, 7(1):55–76, 1998.

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

[18] J. Wijsen. On the consistent rewriting of conjunctive queries under primary key constraints. In M. Arenas and M. I. Schwartzbach, editors, *DBPL*, volume 4797 of *Lecture Notes in Computer Science*, pages 112–126. Springer, 2007. An extended version will appear in *Information Systems.*