Theory Comput Syst (2011) 49:460–488
DOI 10.1007/s00224-011-9323-x

Reconcilable Differences

Todd J. Green · Zachary G. Ives · Val Tannen

Published online: 5 May 2011

Abstract In this paper we study a problem motivated by the management of *changes* in databases. It turns out that several such change scenarios, e.g., the separately studied problems of *view maintenance* (propagation of data changes) and *view adaptation* (propagation of view definition changes) can be unified as instances of *query reformulation using views* provided that support for the *relational difference operator* exists in the context of query reformulation. Exact query reformulation using views in positive relational languages is well understood, and has a variety of applications in query optimization and data sharing. Unfortunately, most questions about queries become undecidable in the presence of difference (or negation), whether we use the foundational set semantics or the more practical bag semantics.

We present a new way of managing this difficulty by defining a novel semantics, $\mathbb{Z}$*-relations*, where tuples are annotated with positive or negative integers. $\mathbb{Z}$-relations conveniently represent data, insertions, and deletions in a uniform way, and can apply deletions with the union operator (deletions are tuples with negative counts). We show

A preliminary version of this paper [16] appeared in the Proceedings of the 12th International Conference on Database Theory (ACM International Conference Proceeding Series, 361 ACM 2009, ISBN 978-1-60558-423-2).
Our work has been supported by the National Science Foundation under grants IIS-0629846, IIS-0803524, IIS-0477972, IIS-0513778, and IIS-1050448.
Work of T. Green performed while at the University of Pennsylvania.

T.J. Green (✉)
University of California, Davis, CA 95616, USA
e-mail: green@cs.ucdavis.edu

Z.G. Ives · V. Tannen
University of Pennsylvania, Philadelphia, PA 19104, USA

Z.G. Ives
e-mail: zives@cis.upenn.edu

V. Tannen
e-mail: val@cis.upenn.edu

that under $\mathbb{Z}$-semantics relational algebra ($\mathcal{RA}$) queries have a normal form consisting of a single difference of positive queries, and this leads to the decidability of their equivalence. We provide a sound and complete algorithm for reformulating $\mathcal{RA}$ queries, including queries with difference, over $\mathbb{Z}$-relations. Additionally, we show how to support standard view maintenance and view adaptation over *set or bag semantics*, through an excursion into the $\mathbb{Z}$-semantics setting. Our algorithm turns out to be sound and complete also for bag semantics, albeit necessarily only for a subclass of $\mathcal{RA}$. This subclass turns out to be quite large and covers generously the applications of interest to us. We also show a subclass of $\mathcal{RA}$ where reformulation and evaluation under $\mathbb{Z}$-semantics can be combined with duplicate elimination to obtain the answer under set semantics. We investigate related complexity questions, and we also extend our results to queries with built-in predicates.

Keywords View maintenance · View adaptation · Query optimization · Query reformulation

1 Introduction

Fundamentally, databases are *dynamic* entities: data gets updated, and schemas and view definitions get revised. In an increasing number of scenarios (e.g., data warehouses, data exchange, and collaborative data sharing systems), large numbers of materialized views are derived from base data: a change to the base data, the view definition, or the base schema might have a cascading effect on the views. A key operation becomes reasoning about the most efficient way to recompute a view instance (i.e., *reformulate a query*) given a set of materialized views, a set of changes, and a set of base relations. A challenge is that updates include deletions as well as insertions—and in general, deletions require a means of handling the *relational difference operator*.

In this paper we study the reformulation (rewriting) of relational queries that contain the *difference* operator. Our goal for query reformulation is to optimize by reusing existing information, such as materialized views. Since the objective is optimization, we focus on *exact* reformulation, which finds only equivalent rewritings of the query.[1]

Query reformulation using views is well understood for *positive* fragments of relational languages, such as conjunctive queries (CQs) or unions of CQs (UCQs), under both set and bag semantics (see, e.g., [4, 28]). As we shall discuss in more detail in the preamble to Sect. 4 in both cases (bag and set semantics), complete procedures for finding UCQ rewritings using UCQ views exist, using finite search spaces. Also, in both cases UCQ equivalence is decidable. In fact, in the same discussion we argue that whenever a (reasonable) finite search space procedure exists, query equivalence must also be decidable.

It follows that the initial outlook on doing reformulations involving the difference operator is glum because even without views the equivalence of relational algebra

[1] In data integration, one is also interested in *maximally contained rewritings*, see e.g., [22].

($\mathcal{RA}$) queries is undecidable, for both set and bag semantics.[2] Hence, we cannot hope for the approaches to UCQ reformulation under bag or set semantics to extend to the entire $\mathcal{RA}$.

However, being able to solve this problem even partially would have an impact in at least three major change propagation scenarios. With reformulation of queries that include difference:

- *Optimization using materialized views* could be done over a broader space of plans. Even if the original query and view were just CQs/UCQs it would be valuable if we could find rewritings that, e.g., *subtract* one view from a larger view in order to return a query answer. Sometimes only such rewritings using difference exist.
- *View adaptation* [19], the act of updating a materialized view instance when the view definition has changed, could be seen as a reformulation using views. Here, the updated view can be recomputed based on the old contents of the view, by adding and/or subtracting queries over the base data and possibly other views. This would be significant progress over existing approaches.
- *Incremental view maintenance* [20] could be seen as a reformulation using views, since insertions and deletions could be treated as unions and differences. Consequently, one can consider multiple rewritings as solutions, using a cost model to choose the best one.

Since all these scenarios are highly relevant practical problems in databases, we are led to ask the following natural question: is there a slightly less expressive class of queries than $\mathcal{RA}$,—still including difference, and hence still providing the benefits cited above—for which reformulation can be handled effectively. In this paper we do this via an excursion through a non-standard semantics that is of interest in its own right: what we term $\mathbb{Z}$-*relations*. These are relations whose tuples are annotated with integers (positive or negative) and the positive $\mathcal{RA}$ operators are defined on them according to the semiring-annotated semantics used in our previous [17, 18]. In addition, difference has an obvious, natural definition on $\mathbb{Z}$-relations.

$\mathbb{Z}$-relations are a natural and uniform representation for *both data and updates*. For example, they can represent updates to source relations (collections of tuple insertions and deletions, a.k.a. *deltas*) which must be propagated in incremental view maintenance applications. Indeed, "application" of a delta to a relation corresponds to simply computing a union. We discuss this further in Sect. 2.

It turns out that reformulation of $\mathcal{RA}$ queries using $\mathcal{RA}$ views can be solved effectively with respect to the $\mathbb{Z}$-semantics since here equivalence of $\mathcal{RA}$ queries with respect to a set of $\mathcal{RA}$ views is *decidable*. We provide a *sound and complete* algorithm for finding rewritings of $\mathcal{RA}$ queries using $\mathcal{RA}$ views under $\mathbb{Z}$-semantics.

Moreover, we obtain practically useful results about the class of $\mathcal{RA}$ queries for which the reformulation with respect to $\mathbb{Z}$-semantics remains valid with respect to bag semantics. For example, the algorithm is complete for finding $\mathcal{RA}$-reformulations of UCQs using UCQ views, provided we are only looking for well-behaved reformulations that on $\mathbb{N}$-instances produce the same results under $\mathbb{Z}$-semantics as under

[2]The latter follows, e.g., from the undecidability of bag-containment of unions of conjunctive queries (UCQs) [24], since for UCQs Q, Q' we have Q is contained in Q' iff $Q - Q'$ is equivalent to the empty answer query.

$\mathbb{N}$-semantics. We show two examples that fall in this category in Sect. 2: an optimization using views example and a view adaptation example. Another case in which the algorithm is nicely complete, for both bag and set semantics, is the application to incremental view maintenance using the so-called *delta rules* [20], also discussed in Sect. 2.

The main contributions of the paper are:

- We show that under $\mathbb{Z}$-semantics every $\mathcal{RA}$ query is equivalent to the difference of two queries in $\mathcal{RA}^+$. The latter are selection/projection/join/union queries, forming the *positive* relational algebra, and equivalent in expressiveness to UCQs. Then the decidability of equivalence of $\mathcal{RA}$ queries under $\mathbb{Z}$-semantics is a corollary of the decidability of equivalence of UCQs.
- It follows that in reformulation using views under $\mathbb{Z}$-semantics we can work with differences of unions of conjunctive queries (DUCQs). We give a terminating, confluent, sound and complete rewrite system such that if two DUCQs are equivalent under a set of views then they can be rewritten to the same query (modulo isomorphism). This leads to our procedure for exploring the space of reformulations (using the opposites of the rewrite rules).
- In contrast to CQs/UCQs under set semantics, there is no inherent or natural, instance-independent notion of "minimality" for DUCQs under $\mathbb{Z}$-semantics that would yield a finite reformulation search space. We bound the search under a simple cost model, which is an abstraction of the one used in a query optimizer.
- We examine when we can use the $\mathbb{Z}$-semantics reformulation strategy to obtain results that work for the bag semantics and set semantics. We show that the reformulation procedure is complete for queries/views in a certain class of queries that on $\mathbb{N}$-instances produce the same results under $\mathbb{Z}$-semantics as under $\mathbb{N}$-semantics. Membership in this class is necessarily undecidable but we give simple and practical sufficient membership criteria. We further examine when we can also obtain results that are sound under set semantics provided we are allowed to add duplicate elimination to the reformulations.
- Finally, we also show how to extend our results to queries with *built-in predicates*, i.e., inequalities and non-equalities.

The paper is structured as follows. We discuss motivating applications in Sect. 2. We define the semantics of $\mathcal{RA}$ on $\mathbb{Z}$-relations, establish the decidability of $\mathbb{Z}$-equivalence of $\mathcal{RA}$ queries and introduce DUCQs in Sect. 3. We introduce the rewrite system for queries using views in Sect. 4. We present reformulation algorithms and strategies in Sect. 5. We discuss reformulation for bag semantics/set semantics via $\mathbb{Z}$-semantics in Sect. 6. We extend our $\mathbb{Z}$-equivalence results to $\mathcal{RA}$ with built-in predicates in Sect. 7. We conclude in Sect. 8 with a discussion of related work.

2 Applications of Differences

In this section, we illustrate the three motivating applications mentioned in the introduction, and show how these problems are closely related. Given a uniform way of representing *data* along with *changes to the data*—including deletions or difference

operations over data, as well as insertions—we can consider each of these problems to be a case of *query reformulation* or *query rewriting*. We shall propose $\mathbb{Z}$-relations as a unifying representation for this purpose, since they can capture base data, insertions, and deletions.

Optimizing Queries Using Views [4, 28] Given a query Q and a set of materialized views $\mathcal{V}$, the goal is to speed up computation of Q by (possibly) rewriting Q using views in $\mathcal{V}$. Sometimes, a view may be "nearly" applicable for answering a query, but cannot be used unless difference is allowed in the rewriting. For example, consider a view V with paths of length 2 and 3 in R:

$$V(x, y) \text{ :- } R(x, z), R(z, y)$$
$$V(x, y) \text{ :- } R(x, u), R(u, v), R(v, y)$$

and a query Q for paths of length 3:

$$Q(x, y) \text{ :- } R(x, u), R(u, v), R(v, y)$$

One can obtain an answer to Q from an instance of V by *removing* all paths of length 2 from it: compute paths of length 2 (by joining R with itself), then compute the *difference* between V and those paths under bag semantics. If our end goal is set semantics, we would also add a duplicate elimination step at the end.

View Adaptation [19] Here, we have a set of base relations, an existing materialized view, and an updated view definition, and we want to refresh the materialized view instance to reflect the new definition. For example, the materialized view:

$$V(x, y, z) \text{ :- } R(x, y), R(x, z)$$
$$V(x, y, z) \text{ :- } R(x, y), R(y, z)$$
$$V(x, y, z) \text{ :- } R(x, y), R(y, z), y = z$$

might be redefined by deleting the second rule and projecting out the middle column:

$$V'(x, z) \text{ :- } R(x, y), R(x, z)$$
$$V'(x, z) \text{ :- } R(x, y), R(y, z), y = z$$

In this case, the computation of $V'(x, z)$ might be sped up under bag semantics by computing the second rule $V(x, y, z)$:- $R(x, y), R(y, z)$, *subtracting* the tuples of the result, and projecting only x and z. (Again, duplicate removal could be done at the end to get a set-semantics answer.)

Incremental View Maintenance [20] We are given a source database, a materialized view, and a set of changes to be applied to the source database (tuple insertions or deletions), and the goal is to compute the corresponding change to the materialized view. This can then be applied to the existing materialized view to obtain the new

version. For example, consider a source relation R and materialized view V, with definitions and instances:

$$R: \begin{array}{|cc|}\hline a & b \\ b & a \\ b & c \\ c & b \\ \hline\end{array}, \quad V(x,y) \text{ :- } R(x,z), R(z,y) \ : \ \begin{array}{|cc|}\hline a & a \\ a & c \\ b & b \\ c & a \\ c & c \\ \hline\end{array}$$

Suppose we update R by deleting (b,a) and inserting (c,d); to maintain V, we must insert a new tuple (b,d). Note that deleting (b,a) does *not* result in deleting (b,b) from V, because this tuple can still be derived by joining (b,c) with (c,b). Yet if we now delete (c,b) from R, then (b,b) and (c,c) must be deleted from V.

In order to solve the incremental view maintenance problem, Gupta et al. [20] proposed recording in V along with each tuple the *number of derivations* of that tuple, i.e., the *multiplicity* of the tuple under bag semantics. To represent changes to a (bag) relation, they introduced the concept of *delta relations*, essentially bag relations with associated *signs*: "+" indicates an insertion and "−" a deletion. Finally, in order to propagate updates, they proposed the device of *delta rules*. In the example above, a set of delta rules for V corresponds to the UCQ:

$$V^{\Delta}(x,y) \text{ :- } R(x,z), R^{\Delta}(z,y)$$
$$V^{\Delta}(x,y) \text{ :- } R^{\Delta}(x,z), R'(z,y)$$

Here R' denotes the updated version of R, obtained by *applying* the delta R^{Δ} to R, i.e., computing $R' \stackrel{\text{def}}{=} R^{\Delta} \cup R$ where union on delta relations sums the (signed) tuple multiplicities. By computing V^{Δ} and then applying it to V, we obtain the updated version of V, namely V'. Note that there may actually be more than one possible set of delta rules for V, e.g.:

$$V^{\Delta}(x,y) \text{ :- } R^{\Delta}(x,z), R(z,y)$$
$$V^{\Delta}(x,y) \text{ :- } R'(x,z), R^{\Delta}(z,y)$$

We would like to choose among the possible delta rules sets, as well as simply computing V' "from scratch" via the query:

$$V'(x,y) \text{ :- } R'(x,z), R'(z,y)$$

based on the expected costs of the various plans. We model the relation R^{Δ} with $\mathbb{Z}$-relations, presented in the next section. We consider this to again be a variant of optimizing queries using views: the goal is to compute the view with deltas applied, V', given not only the base data R and R^{Δ}, but also the existing materialized view V, and relation R' resulting from applying the updates in R^{Δ} to R. (We can compute V' either before or after updating R to R'.) Since every delta relation includes deletions, this version of the reformulation problem *also incorporates a form of difference*.

As we shall see, a unified treatment of these three applications is possible by using methods for representing data and changes via an excursion to an alternative

semantics ($\mathbb{Z}$-relations), performing query reformulation in this context, and proving sufficient conditions for cases in which the results agree with bag and set semantics.

3 Z-Relations

We use here the named perspective [1] of the relational model, in which tuples are functions $t : U \to \mathbb{D}$ with U a finite set of attributes and $\mathbb{D}$ a domain of values. We fix the domain $\mathbb{D}$ for the time being and we denote the set of all such U-tuples by U-Tup. (Usual) relations over U are subsets of U-Tup; we will also refer to these as $\mathbb{B}$-*relations*, since they can also be viewed as mappings from U-Tup to $\mathbb{B} = \{\mathsf{true}, \mathsf{false}\}$.

A *bag relation* over attributes U is a mapping $R : U\text{-Tup} \to \mathbb{N}$ from U-tuples to their associated *multiplicities*. Tuples with multiplicity 0 are those "not present" in R, and we require of a bag relation that its *support* defined by $\mathsf{supp}(R) \stackrel{\text{def}}{=} \{t \mid R(t) \neq 0\}$ is finite.

A $\mathbb{Z}$-*relation* over attributes U is a mapping $R : U\text{-Tup} \to \mathbb{Z}$ of finite support. In other words, it is a bag relation where multiplicities may be positive or negative.

A *bag instance* ($\mathbb{Z}$-instance) is a mapping from predicate symbols to bag relations ($\mathbb{Z}$-relations). A *set instance* I (or $\mathbb{B}$-*instance*) is a mapping from predicate symbols to $\mathbb{B}$-relations. If I is a K-instance (for $K \in \{\mathbb{Z}, \mathbb{B}, \mathbb{N}\}$), then R^I denotes the value of the K-relation for R in I.

We define the semantics of the relational algebra on $\mathbb{Z}$-instances according to the *semiring-annotated* relational semantics used in our previous papers [17, 18]. We begin with the operations of the *positive algebra* ($\mathcal{RA}^+$). If I is a $\mathbb{Z}$-instance and Q is a positive algebra query, then the *result of evaluating* Q *on* I is the $\mathbb{Z}$-relation $[\![Q]\!]^I$ defined inductively as follows:

empty relation For any set of attributes U, there is $\emptyset : U\text{-Tup} \to \mathbb{Z}$ such that

$$[\![\emptyset]\!]^I(t) = 0$$

identity If R is a predicate symbol with attributes U then $[\![R]\!]^I : U\text{-Tup} \to \mathbb{Z}$ is defined by

$$[\![R]\!]^I(t) \stackrel{\text{def}}{=} R^I(t)$$

union If $[\![R_1]\!]^I, [\![R_2]\!]^I : U\text{-Tup} \to \mathbb{Z}$ then $[\![R_1 \cup R_2]\!]^I : U\text{-Tup} \to \mathbb{Z}$ is defined by

$$[\![R_1 \cup R_2]\!]^I(t) \stackrel{\text{def}}{=} [\![R_1]\!]^I(t) + [\![R_2]\!]^I(t)$$

projection If $[\![R]\!]^I : U\text{-Tup} \to \mathbb{Z}$ and $V \subseteq U$ then $[\![\pi_V R]\!]^I : V\text{-Tup} \to \mathbb{Z}$ is defined by

$$[\![\pi_V R]\!]^I(t) \stackrel{\text{def}}{=} \sum_{t = t' \text{ on } V \text{ and } [\![R]\!]^I(t') \neq 0} [\![R]\!]^I(t')$$

(here $t = t'$ on V means t' is a U-tuple whose restriction to V is the same as the V-tuple t; note also that the sum is finite assuming $[\![R]\!]^I$ has finite support).

selection If $[\![R]\!]^I : U\text{-Tup} \to \mathbb{Z}$ and the selection predicate $\mathbf{P}$ maps each U-tuple to either 0 or 1 then $[\![\sigma_{\mathbf{P}} R]\!]^I : U\text{-Tup} \to \mathbb{Z}$ is defined by

$$[\![\sigma_{\mathbf{P}} R]\!]^I(t) \stackrel{\text{def}}{=} [\![R]\!]^I(t) \cdot \mathbf{P}(t)$$

Which $\{0, 1\}$-valued functions are used as selection predicates is left unspecified, except that we assume that false—the constantly 0 predicate, and true—the constantly 1 predicate, are always available.

natural join If $[\![R_i]\!]^I : U_i\text{-Tup} \to \mathbb{Z}$ $i = 1, 2$ then $[\![R_1 \ltimes R_2]\!]^I$ is the $\mathbb{Z}$-relation over $U_1 \cup U_2$ defined by

$$[\![R_1 \ltimes R_2]\!]^I(t) \stackrel{\text{def}}{=} [\![R_1]\!]^I(t_1) \cdot [\![R_2]\!]^I(t_2)$$

where $t_1 = t$ on U_1 and $t_2 = t$ on U_2 (recall that t is a $U_1 \cup U_2$-tuple).

renaming If $[\![R]\!]^I : U\text{-Tup} \to \mathbb{Z}$ and $\beta : U \to U'$ is a bijection then $[\![\rho_\beta R]\!]^I$ is a $\mathbb{Z}$-relation over U' defined by

$$[\![\rho_\beta R]\!]^I(t) \stackrel{\text{def}}{=} [\![R]\!]^I(t \circ \beta)$$

For the time being we assume selection predicates correspond to equalities $A = B$ of attributes or equalities $A = c$ of attributes with domain values. (We extend this to include inequality predicates in Sect. 7.)

Observe that if we start with $\mathbb{Z}$-relations with just positive multiplicities, i.e. $\mathbb{N}$-relations, the results of the operations defined above are also $\mathbb{N}$-relations (leading to Lemma 3.1 below) and the resulting semantics is in fact the usual bag semantics. $\mathbb{B}$-relations correspond to reading the "$+$" as disjunction and the "$\cdot$" as conjunction, essentially "eliminating duplicates," hence we get the usual set semantics.

We extend the above definition to the full relational algebra ($\mathcal{RA}$) on $\mathbb{Z}$-instances by defining the difference operator in the obvious way:

difference If $[\![R_1]\!]^I : U\text{-Tup} \to \mathbb{Z}$ and $[\![R_2]\!]^I : U\text{-Tup} \to \mathbb{Z}$ then $[\![R_1 - R_2]\!]^I : U\text{-Tup} \to \mathbb{Z}$ is defined by

$$[\![R_1 - R_2]\!]^I(t) \stackrel{\text{def}}{=} [\![R_1]\!]^I(t) - [\![R_2]\!]^I(t)$$

For bag semantics, the subtraction in the definition above is replaced by *proper subtraction* (negative numbers are truncated to 0). For $\mathbb{B}$-relations this becomes the usual set difference in set semantics.

Every bag instance is also a $\mathbb{Z}$-instance. Relational queries on set or bag instances can be evaluated under bag semantics or under $\mathbb{Z}$-semantics. To disambiguate we use the notation $[\![Q]\!]^I_K$ to mean the evaluation of Q on bag instance I under K-semantics, for $K \in \{\mathbb{N}, \mathbb{Z}\}$.

For $Q, Q' \in \mathcal{RA}$ and $K \in \{\mathbb{B}, \mathbb{N}, \mathbb{Z}\}$ we say that Q and Q' are *K-equivalent* (denoted $Q \equiv_K Q'$) if for every K-instance I, $[\![Q]\!]^I = [\![Q']\!]^I$. The following simple but useful observation relates $\mathbb{N}$-equivalence and $\mathbb{Z}$-equivalence of positive queries:

Lemma 3.1 *If $Q, Q' \in \mathcal{RA}^+$ then $Q \equiv_{\mathbb{Z}} Q'$ iff $Q \equiv_{\mathbb{N}} Q'$.*

Proof "⇒" follows from the fact that every bag instance is also a $\mathbb{Z}$-instance and the two semantics agree for positive queries on bag instances. "⇐" follows from the fact that bag equivalent positive queries, when transformed into UCQs, are isomorphic [9]. □

3.1 Normal Form and Decidability

Equivalence of relational queries under set semantics has long been known to be undecidable [34]. We have also seen (cf. footnote in Sect. 1) that bag equivalence of relational queries is undecidable, via an easy reduction from containment of UCQs (which makes essential use of proper subtraction). In contrast, the form of subtraction used in $\mathbb{Z}$-relations turns out to be surprisingly well-behaved: we will show in this section that $\mathbb{Z}$-equivalence of relational queries is actually *decidable*. The key idea is that in contrast to bag and set semantics, under $\mathbb{Z}$-semantics, every relational query is equivalent to a single difference of positive queries.

Definition 3.2 For any $Q \in \mathcal{RA}$ we define a *difference normal form* denoted by $\mathsf{DiffNF}(Q)$ by structural recursion on Q as follows:

- If R is a predicate symbol then $\mathsf{DiffNF}(R) = R - \emptyset$.
- If $\mathsf{DiffNF}(Q) = A - B$ then

$$\mathsf{DiffNF}(\pi_X(Q)) = \pi_X(A) - \pi_X(B), \quad \text{and}$$
$$\mathsf{DiffNF}(\sigma_P(Q)) = \sigma_P(A) - \sigma_P(B).$$

- If $\mathsf{DiffNF}(Q_1) = A_1 - B_1$ and $\mathsf{DiffNF}(Q_2) = A_2 - B_2$ then

$$\mathsf{DiffNF}(Q_1 \bowtie Q_2) = (A_1 \bowtie A_2 \ \cup \ B_1 \bowtie B_2) - (A_1 \bowtie B_2 \ \cup \ B_1 \bowtie A_2),$$
$$\mathsf{DiffNF}(Q_1 \cup Q_2) = (A_1 \cup A_2) - (B_1 \cup B_2), \quad \text{and}$$
$$\mathsf{DiffNF}(Q_1 - Q_2) = (A_1 \cup B_2) - (A_2 \cup B_1).$$

Moreover, the above rules clearly also define an effective procedure for computing $\mathsf{DiffNF}(Q)$.

Theorem 3.3 (Normalization) *For any $Q \in \mathcal{RA}$ we can effectively find $A, B \in \mathcal{RA}^+$ such that $Q \equiv_{\mathbb{Z}} A - B$.*

Proof Clearly $\mathsf{DiffNF}(Q)$ has the form $A - B$ with $A, B \in \mathcal{RA}^+$. It is straightforward to show by induction on Q that $Q \equiv_{\mathbb{Z}} \mathsf{DiffNF}(Q)$ using the algebraic $\mathbb{Z}$-semantics identities in Fig. 1. Note that in general $\mathsf{DiffNF}(Q)$ may be of size exponential in the size of Q.

The given definition of $\mathsf{DiffNF}(Q)$ proliferates redundant occurrences of $\emptyset$. We will assume that simplifications are made based on identities such as $A \cup \emptyset \equiv A$ or $A \bowtie \emptyset \equiv \emptyset$ (true for $\mathbb{Z}$-semantics, bag semantics, set semantics, and in fact all semiring-annotated semantics). As a result, it is easy to check that for $\mathcal{RA}^+$ queries Q we have $\mathsf{DiffNF}(Q) = Q - \emptyset$. □

$$
\begin{array}{ll}
(A-B)-C \equiv_{\mathbb{Z}} A-(B\cup C) & \sigma_P(A-B) \equiv_{\mathbb{Z}} \sigma_P(A)-\sigma_P(B) \\
A \ltimes (B-C) \equiv_{\mathbb{Z}} (A \ltimes B)-(A \ltimes C) & (!)\ (A-B)\cup C \equiv_{\mathbb{Z}} (A\cup C)-B \\
(!)\ A-(B-C) \equiv_{\mathbb{Z}} (A\cup C)-B & (!)\ \pi_X(A-B) \equiv_{\mathbb{Z}} \pi_X(A)-\pi_X(B) \\
(A-B) \ltimes C \equiv_{\mathbb{Z}} (A \ltimes C)-(B \ltimes C) & \rho_\beta(A-B) \equiv_{\mathbb{Z}} \rho_\beta(A)-\rho_\beta(B) \\
(!)\ A\cup(B-C) \equiv_{\mathbb{Z}} (A\cup B)-C &
\end{array}
$$

Fig. 1 Algebraic identities for the difference operator under $\mathbb{Z}$-semantics

Note that the two identities in Fig. 1 that are flagged by (!) *fail* in fact for both set and bag semantics. For instance, consider binary relations R and S. For the first identity marked (!), we have $(R-(R\cup R))\cup R \equiv_{\mathbb{Z}} (R\cup R)-(R\cup R)$, but the two queries are inequivalent under set or bag semantics; while for the second identity marked (!), we have $\pi_1(R-S) \equiv_{\mathbb{Z}} \pi_1(R)-\pi_1(S)$, but again the two queries are inequivalent under either set or bag semantics. Indeed, Theorem 3.3 fails for set or bag semantics.

Corollary 3.4 *$\mathbb{Z}$-equivalence of $\mathcal{RA}$ queries is decidable.*

Proof Let $Q, Q' \in \mathcal{RA}$. By Theorem 3.3, $Q \equiv_{\mathbb{Z}} Q'$ iff $\mathsf{DiffNF}(Q) \equiv_{\mathbb{Z}} \mathsf{DiffNF}(Q')$. Let $A, B, C, D \in \mathcal{RA}^+$ such that $\mathsf{DiffNF}(Q) = A - B$ and $\mathsf{DiffNF}(Q') = C - D$. But $A - B \equiv_{\mathbb{Z}} C - D$ iff $A \cup D \equiv_{\mathbb{Z}} B \cup C$ iff $A \cup D \equiv_{\mathbb{N}} B \cup C$. But $A \cup D$ and $B \cup C$ are in $\mathcal{RA}^+$ and hence equivalent to UCQs so the result follows from the decidability of UCQ bag equivalence [9]. □

Corollary 3.4 can be used to show a *PSPACE* upper bound on the complexity of checking $\mathbb{Z}$-equivalence of $\mathcal{RA}$ queries; however, the exact complexity remains open. A lower bound is given by Proposition 3.8 which shows that the problem is at least *GI*-hard.[3]

Remark 3.5 Converting $\mathcal{RA}^+$ queries to UCQs may increase their size exponentially, and it is well-known that for set semantics, there is a corresponding jump in the complexity of checking containment/equivalence: for UCQs, checking containment/equivalence is *NP*-complete but for $\mathcal{RA}^+$ queries it is Π_2^p-complete [34]. However, for bag-equivalence/$\mathbb{Z}$-equivalence of $\mathcal{RA}^+$ queries, the question seems to be open and may be difficult to resolve. There are intriguing connections with the *polynomial identity testing* problem (PIT), an important open problem in theoretical computer science. PIT is known to be solvable in probabilistic polynomial time and conjectured to be perhaps solvable in *PTIME* (via a derandomization of the probabilistic procedure). In fact, we can show that for the cross product-union fragment of $\mathcal{RA}^+$, at least, bag-equivalence/$\mathbb{Z}$-equivalence is interreducible with the *non-commutative* variant of PIT, which *is* known to lie in *PTIME* [33].

Another useful consequence of Theorem 3.3 is the following:

[3] *GI* is the class of problems polynomial time reducible to graph isomorphism. Graph isomorphism is known to be in *NP*, but is not known or believed to be either *NP*-complete or in *PTIME*.

Lemma 3.6 *For any $Q_1, Q_2 \in \mathcal{RA}$, we have $Q_1 \equiv_{\mathbb{Z}} Q_2$ iff $[\![Q_1]\!]^I_{\mathbb{Z}} = [\![Q_2]\!]^I_{\mathbb{Z}}$ for every $\mathbb{N}$-instance I.*

Proof "$\Rightarrow$" is immediate. For "$\Leftarrow$", suppose $Q_1 \not\equiv_{\mathbb{Z}} Q_2$. By Theorem 3.3, we have $Q_1 \equiv_{\mathbb{Z}} A - B$ and $Q_2 \equiv_{\mathbb{Z}} C - D$ with $A, B, C, D \in \mathcal{RA}^+$. Since $A - B \not\equiv_{\mathbb{Z}} C - D$, we have $A \cup D \not\equiv_{\mathbb{Z}} B \cup C$. By Lemma 3.1, this implies $A \cup D \not\equiv_{\mathbb{N}} B \cup C$. Therefore for some $\mathbb{N}$-instance I, we have $[\![A \cup D]\!]^I_{\mathbb{N}} \neq [\![B \cup C]\!]^I_{\mathbb{N}}$, hence $[\![A \cup D]\!]^I_{\mathbb{Z}} \neq [\![B \cup C]\!]^I_{\mathbb{Z}}$. Now consider some output tuple t such that $[\![A \cup D]\!]^I_{\mathbb{Z}}(t) \neq [\![B \cup C]\!]^I_{\mathbb{Z}}(t)$. It follows that $[\![A]\!]^I_{\mathbb{Z}}(t) + [\![D]\!]^I_{\mathbb{Z}}(t) \neq [\![B]\!]^I_{\mathbb{Z}}(t) + [\![C]\!]^I_{\mathbb{Z}}(t)$. But then $[\![A]\!]^I_{\mathbb{Z}}(t) - [\![B]\!]^I_{\mathbb{Z}}(t) \neq [\![C]\!]^I_{\mathbb{Z}}(t) - [\![D]\!]^I_{\mathbb{Z}}(t)$. It follows that $[\![A - B]\!]^I_{\mathbb{Z}}(t) \neq [\![C - D]\!]^I_{\mathbb{Z}}(t)$. Hence $[\![Q_1]\!]^I_{\mathbb{Z}} \neq [\![Q_2]\!]^I_{\mathbb{Z}}$. □

We shall see an application of Lemma 3.6 in Sect. 6.

3.2 DUCQs

As we shall see below, our reformulation algorithm uses $\mathcal{RA}$ queries in difference normal form. In, fact as with CQs and UCQs it is notationally convenient to use a Datalog-style syntax for the queries on which reformulation operates directly. We define *differences of unions of conjunctive queries* (DUCQs) for this purpose, for example:

$$\begin{aligned} Q(x,z) &:\text{-}\ R(x,y), S(y,z) \\ Q(x,y) &:\text{-}\ R(x,u), R(u,v), R(v,y) \\ -Q(x,y) &:\text{-}\ R(x,y), T(y,y) \end{aligned}$$

The "$-$" marks a "negated" CQ. If A, B, C are the $\mathcal{RA}^+$ queries encoding the first, second, and third rules above, respectively, the equivalent $\mathcal{RA}$ query is

$$Q = A \cup B - C.$$

DUCQs are related to the *elementary differences* of [34]. Under $\mathbb{Z}$-semantics, any $\mathcal{RA}$ query can be written equivalently as a DUCQ; this follows from Theorem 3.3 and the fact that any $\mathcal{RA}^+$ query can be rewritten as a UCQ. The semantics of DUCQs on bag relations/$\mathbb{Z}$-relations/set relations can be given formally by translation to $\mathcal{RA}$.

Definition 3.7 If $Q = Q_1 - Q_2$ is a DUCQ which is the difference of UCQs Q_1 and Q_2, and I is a K-instance (for $K \in \{\mathbb{Z}, \mathbb{N}, \mathbb{B}\}$), then the *result of evaluating Q on I* is the K-relation

$$[\![Q]\!]^I_K(t) \stackrel{\text{def}}{=} [\![\mathsf{trans}(Q_1)]\!]^I_K(t) - [\![\mathsf{trans}(Q_2)]\!]^I_K(t)$$

where $\mathsf{trans} : \text{UCQ} \to \mathcal{RA}^+$ is the standard translation of UCQs to positive relational queries.

Although we do not have a precise complexity for the $\mathbb{Z}$-equivalence of $\mathcal{RA}$ queries, we can fully characterize the complexity of checking $\mathbb{Z}$-equivalence of DUCQs. For this and further developments we recall the standard notion of isomorphism of UCQs. CQs A, B are said to be *isomorphic*, denoted $A \cong B$, if there is a bijective mapping h of variables of A to variables of B (extended to the identity on constants) that induces a bijection of atoms from the body of A to the body of B. UCQs $A = A_1 \cup \cdots \cup A_m$ and $B = B_1 \cup \cdots \cup B_n$ are said to be *isomorphic*, again denoted $A \cong B$, if there is a bijection $g : \{1, \ldots, m\} \to \{1, \ldots, n\}$ (hence $m = n$) such that $A_i \cong B_{g(i)}$ for $1 \leq i \leq n$. Finally, we say that DUCQs $A - B$ and $C - D$ are *isomorphic*, denoted $A - B \cong C - D$, if $A \cong C$ and $B \cong D$.

Proposition 3.8 *For* DUCQ*s* $Q = A - B$, $Q' = C - D$ *we have* $Q \equiv_{\mathbb{Z}} Q'$ *iff for UCQs* $A \cup D$ *and* $B \cup C$, $A \cup D \cong B \cup C$. *As a consequence, checking* $Q \equiv_{\mathbb{Z}} Q'$ *is GI-complete.*

Proof Following similar reasoning as in the proof of Corollary 3.4, we have $A - B \equiv_{\mathbb{Z}} C - D$ iff $A \cup D \equiv_{\mathbb{Z}} B \cup C$ iff $A \cup D \equiv_{\mathbb{N}} B \cup C$. But by the results of [9], this holds iff $A \cup D \cong B \cup C$. Thus $\mathbb{Z}$-equivalence of DUCQs is polynomial time many-one interreducible with the *GI*-complete problem of checking isomorphism of UCQs. □

4 Reformulation Using Views

Let Σ be a relational schema and $\mathcal{V}$ a finite set of views over Σ. Let $\Sigma_{\mathcal{V}}$ be the schema consisting of the view names (disjoint from Σ). A $\Sigma \cup \Sigma_{\mathcal{V}}$ instance is $\mathcal{V}$-*compatible* if it consists of a Σ-instance I and a $\Sigma_{\mathcal{V}}$-instance J such that $J = [\![\mathcal{V}]\!]^I$. Orthogonally, these instances may consist of $\mathbb{Z}$-relations, $\mathbb{N}$-relations or $\mathbb{B}$-relations. For $K \in \{\mathbb{B}, \mathbb{N}, \mathbb{Z}\}$ we say that two relational queries Q, Q' over $\Sigma \cup \Sigma_{\mathcal{V}}$ are K-*equivalent under* $\mathcal{V}$, denoted $Q \equiv_K^{\mathcal{V}} Q'$, if Q and Q' agree on all $\mathcal{V}$-*compatible* K-instances.

Given a Σ-query Q, a *reformulation of* Q *using* $\mathcal{V}$ is a $\Sigma \cup \Sigma_{\mathcal{V}}$-query Q' such that $Q \equiv_K^{\mathcal{V}} Q'$. We also call Q' an *equivalent rewriting* using $\mathcal{V}$.

Query reformulation algorithms typically work by *effectively enumerating* certain queries, call this a *search space*, filtering the queries that are not equivalent rewritings, and finding a minimum-cost query (according to some cost model). There are three requirements for designing such algorithms; (1) the search space must be *finite* so the algorithm terminates, (2) the returned rewriting should be actually equivalent so the algorithm is *sound*, and (3) if equivalent rewritings exist then at least one of them should be found so the algorithm is *complete*. A more subtle requirement is that the returned rewriting has smaller cost than *any* rewriting, even when there are infinitely many ones.

In the case of set semantics one can construct infinitely many rewritings of CQs even without views, simply by adding to their bodies atoms that can be homomorphically mapped to existing ones [1]. This can be dealt with by considering only queries that have no non-trivial endomorphism, call them *locally minimal*. It was shown in [28] that the space of locally minimal CQ reformulations of a CQ query

Q using CQ views $\mathcal{V}$ is finitely bounded. Further work focused on pruning and efficiently exploring this search space (looking for not just equivalent rewritings but also maximally contained rewritings) [13, 29, 31, 32]. Even if certain classes of integrity constraints (capturing views and more) are considered, a corresponding notion of minimality can be used to define a finite search space for rewritings, using the *chase and backchase* technique [12]. For CQs and bag semantics a finite search space exploration is described in [4]. Interestingly, the UCQ and/or bag-semantics analogs of the complexity results in [28] do not seem to appear anywhere (to the best of our knowledge). For bag semantics, we attempt to remedy this omission in Sect. 4.1 below.

In the reformulation procedures we have mentioned, the search spaces are constructed and enumerated combinatorially, thus including non-equivalent rewritings. One takes advantage of the decidability of $\equiv_K^{\mathcal{V}}$ to filter them out. But such decidability is not just sufficient, it is also *necessary*. Indeed, all such reasonable approaches describe a total recursive function that associates to each Σ-query Q a finite set $searchSpace(Q)$ of $\Sigma \cup \Sigma_{\mathcal{V}}$-queries. Moreover, procedures like local minimization can be abstracted by another total recursive function that associates to each Σ-query Q and each $\Sigma \cup \Sigma_{\mathcal{V}}$-query Q' another $\Sigma \cup \Sigma_{\mathcal{V}}$-query $\mu(Q, Q')$ such that $Q' \equiv_K \mu(Q, Q')$. And finally, one shows that $Q' \equiv_K^{\mathcal{V}} Q$ if and only if $\mu(Q, Q') \in searchSpace(Q)$. Since the latter is decidable, so is $\equiv_K^{\mathcal{V}}$.

In particular, K-equivalence of Σ-queries must also be decidable which is why under bag or set semantics we cannot hope to extend the ideas that have worked for positive queries to rewritings of relational queries with difference.

We have seen that shifting to $\mathbb{Z}$-semantics however leads to the decidability of $\mathbb{Z}$ equivalence of $\mathcal{RA}$ queries (hence DUCQs). According to the discussion above, next we need to explore $\mathbb{Z}$-equivalence under a set of views and we do so in Sect. 4.2 using a uniquely terminating *term rewrite system*.

In the same subsection we discuss adding the reverse of the term rewrite rules as the basis for a reformulation algorithm. In the last two subsections we consider limiting our search to *diff-irredundant rewritings*, and we develop a *cost model* and a procedure for limiting the search space further.

4.1 Complexity of Bag Reformulation

In the case of bag semantics, UCQ queries and UCQ views there are only finitely many rewritings (modulo symbol renaming). This is essentially because under bag semantics, positive query equivalence is the same as isomorphism (see [5] for CQs and [9] for UCQs). For CQs, that is, select-project-join queries, an algorithm for exploring the resulting finite search space in conjunction with System-R style query optimization was given in [4].

We consider here, for UCQs and bag semantics, the analogs of the complexity results shown in [28] for the existence of rewritings of CQs using CQ views under set semantics.

Theorem 4.1 *Given a query $Q \in$ UCQ and a set of views $\mathcal{V} \subseteq$ UCQ, it is NP-complete to determine whether*:

(i) *There exists a* UCQ *reformulation under bag semantics of* Q *using at least one predicate in* $\mathcal{V}$ (*the problem is NP-hard even for* CQ*s*).
(ii) *There exists a* UCQ *reformulation under bag semantics of* Q *that is* complete, *i.e., using only predicates in* $\mathcal{V}$ (*the problem is NP-hard even for* CQ*s*).

Proof To show membership in *NP* for both cases, we observe that in each case we can guess the rewriting along with an isomorphism between Q and the rewriting with unfolded view definitions needed to demonstrate bag-equivalence, and these are easily verifiable in polynomial time. It remains to show *NP*-hardness in each case.

(i) *NP*-hardness when Q and $\mathcal{V}$ are all CQs (and hence also for the case where Q and $\mathcal{V}$ may be UCQs) is established by a straightforward reduction from the *subgraph isomorphism* problem: given directed graphs G_1, G_2 is there a subgraph of G_1 which is isomorphic to G_2? (In contrast to graph isomorphism, the subgraph isomorphism problem is known to be *NP*-complete.) Indeed, given directed graphs G_1, G_2, we construct CQs Q_1, Q_2 whose bodies encode the graphs in the standard fashion using a single binary predicate E and a variable x_i for each graph vertex v_i, and whose heads return all the variables occurring in the bodies. Q_1 has a rewriting using Q_2 iff there is a *safe substitution* [4] of Q_2 in Q_1. But such a safe substitution exists iff there is a subset of the body of Q_1 which is isomorphic to the body of Q_2, which is true iff there is a subgraph of G_1 which is isomorphic to G_2.

(ii) *NP*-hardness when Q and $\mathcal{V}$ are CQs is again established by a reduction from the subgraph isomorphism problem. This time, we work with a slightly specialized version of the problem: if m is the number of edges in G_1, and n is the number of edges in G_2, we assume that $m - n$ does not divide m. It can be shown that this version of the problem remains *NP*-complete. Next, given directed graphs G_1 and G_2 satisfying this assumption, we construct a Boolean CQ Q encoding the edges of G_1, a CQ view V_1 encoding the edges of G_2, and whose head contains all the vertices of G_2 as distinguished variables, and an additional CQ view V_2 defined as follows:

$$V_2(x_1, \ldots, x_{2(m-n)}) \text{ :- } E(x_1, x_2), E(x_3, x_4), \ldots, E(x_{2(m-n)-1}, x_{2(m-n)})$$

Intuitively, V_2 will be used to "cover the remainder" of a reformulation of Q using V_1. Now we need to show that there exists a complete reformulation of Q using $\{V_1, V_2\}$ iff there exists a subgraph isomorphism from G_1 to G_2. Suppose there is a complete reformulation of Q using $\{V_1, V_2\}$. Then, the reformulation must use V_1 at least once (a reformulation using V_2 is not possible since V_2 contains $m - n$ subgoals, V_1 contains m subgoals, and we have assumed that $m - n$ does not divide m). The safe substitution associated with the use of V_1 in the reformulation yields the required subgraph isomorphism. In the other direction, if there is a subgraph of G_1 that is isomorphic to G_2, then there is a reformulation of Q using V_1. It is easy to see that this can be extended to a reformulation of Q using V_1 and V_2 each exactly once. □

By the way, for DUCQs, the first question in the theorem has a trivial answer: a DUCQ Q can always be rewritten as the equivalent $(V \cup -V) \cup Q$ (equivalent under bag semantics, $\mathbb{Z}$-semantics, and even set semantics). As for the second question in the theorem, for DUCQs, we do not even know if this problem is decidable.

4.2 Term Rewriting System and Decidability of Equivalence under Views

For the dual purposes of checking equivalence under views and—as we shall see later—enumerating reformulations we introduce here a *term rewriting system* [26] for DUCQs under $\mathbb{Z}$-semantics. We fix a relational schema Σ and a set of views $\mathcal{V}$ given by DUCQs over the relations in Σ. The terms of our rewrite system are the DUCQs over any combination of the source predicates in Σ and the view predicates in $\mathcal{V}$.

In the rewrite rules we use an auxiliary *view unfolding relation* on CQs, defined $Q \to_V Q'$, if Q' can be obtained from a CQ Q by *unfolding* (in the standard way) a single occurrence of the view predicate V in Q. This can be extended to CQs containing "$-$" marks by defining the result to have a "$-$" mark iff exactly one of the query and view have a "$-$" mark (in other words, we "multiply" the signs). For instance, if U is the CQ

$$-U(x) \text{ :- } S(x, y, z)$$

and Q is the query

$$Q() \text{ :- } U(x), W(x)$$

then $Q \to_V Q'$ where Q' is the CQ

$$-Q() \text{ :- } S(x, y, z), W(x)$$

We extend $\to_V$ to work with DUCQ views by unfolding repeatedly (once for each CQ in the view) and producing a DUCQ as output (with the same number of rules as the view). For example, if V is the DUCQ view:

$$V(x, y) \text{ :- } R(x, z), R(z, y)$$
$$-V(x, y) \text{ :- } R(x, u), R(u, v), R(v, y)$$

and Q is the CQ:

$$-Q(x, y) \text{ :- } V(x, z), V(z, y)$$

then $Q \to_V Q'$ where Q' is the UCQ:

$$-Q'(x, y) \text{ :- } V(x, z), R(z, w), R(w, y)$$
$$Q'(x, y) \text{ :- } V(x, z), R(z, u), R(u, v), R(v, y)$$

Note that the second rule for Q' has no "$-$" mark because the marks on Q and the second rule for V have cancelled each other.

Now we define a rewrite relation $\to$ on terms as follows:

$$\frac{P, Q, R \in \text{DUCQ} \quad V \in \mathcal{V} \quad P \to_V Q}{P \cup R \to Q \cup R} \qquad \text{(UNFOLD)}$$

$$\frac{A, B \in \text{CQ} \quad Q \in \text{DUCQ} \quad A \cong B}{Q \cup A \cup (-B) \to Q} \qquad \text{(CANCEL)}$$

Next we establish the salient properties of our rewrite system. We denote the transitive reflexive closure of $\rightarrow$ by $\stackrel{*}{\rightarrow}$. A term is a *normal form* if no rewrite rule applies to it. A *reduction sequence* $Q_1 \rightarrow \cdots \rightarrow Q_n$ is *terminating* if Q_n is a normal form.

Proposition 4.2 *The rewrite system above is* uniquely terminating, *i.e., it satisfies the following two properties*:

1. (*Confluence*) *For all* $Q, Q_1, Q_2 \in$ DUCQ *if* $Q \stackrel{*}{\rightarrow} Q_1$ *and* $Q \stackrel{*}{\rightarrow} Q_2$ *then there exist* $Q_3, Q_3' \in$ DUCQ *such that* $Q_1 \stackrel{*}{\rightarrow} Q_3$ *and* $Q_2 \stackrel{*}{\rightarrow} Q_3'$ *and* $Q_3 \cong Q_3'$.
2. (*Termination*) *Every reduction sequence* $Q_1 \rightarrow Q_2 \rightarrow \cdots$ *eventually must terminate.*

Proposition 4.3 *The rewrite system above is* sound and complete *w.r.t.* $\mathbb{Z}$*-equivalence with respect to* $\mathcal{V}$, *i.e., for any* $Q_1, Q_2 \in$ DUCQ *we have*

1. (*Soundness*) *If* $Q_1 \stackrel{*}{\rightarrow} Q_2$ *then* $Q_1 \equiv_{\mathbb{Z}}^{\mathcal{V}} Q_2$.
2. (*Completeness*) *If* $Q_1 \equiv_{\mathbb{Z}}^{\mathcal{V}} Q_2$ *then there exist* Q_1', Q_2' *such that* $Q_1 \stackrel{*}{\rightarrow} Q_1'$ *and* $Q_2 \stackrel{*}{\rightarrow} Q_2'$ *and* $Q_1' \cong Q_2'$.

Proof of Propositions 4.2 and 4.3 (*Termination*) Let n be the number of occurrences of view predicate symbols in DUCQ Q_1, and let m be the maximum number of CQs in a DUCQ view definition. Then one can show that any reduction sequence $Q_1 \rightarrow Q_2 \rightarrow \cdots$ uses UNFOLD at most m^n times. Next, note that if $Q_1 \rightarrow Q_2$ using CANCEL, and Q_1 has k CQs, then $k \geq 2$ and Q_2 has $k-2$ CQs. Thus CANCEL may only be applied $\lfloor k/2 \rfloor$ times in a row to Q_1. It follows that any reduction sequence is finite and therefore terminating.

(*Soundness*) It is easy to verify that UNFOLD and CANCEL are equivalence-preserving, and the general result follows by induction on the reduction sequence.

(*Confluence*) Suppose $Q \stackrel{*}{\rightarrow} Q_1$ and $Q \stackrel{*}{\rightarrow} Q_2$. Then there are terminating sequences $Q \rightarrow Q_1 \stackrel{*}{\rightarrow} Q_1'$ and $Q \rightarrow Q_2 \stackrel{*}{\rightarrow} Q_2'$. Since the rewrite system is sound, $Q_1' \equiv_{\mathbb{Z}}^{\mathcal{V}} Q_2'$. Moreover, Q_1' and Q_2' contain no view predicates (else UNFOLD would have applied), so $Q_1' \equiv_{\mathbb{Z}} Q_2'$. Letting $Q_1' = A - B$ and $Q_2' = C - D$, Proposition 3.8 implies that $A \cup D \cong B \cup C$. Since the reduction sequences were terminating, CANCEL does not apply to Q_1' or Q_2', hence no CQ in A (resp. C) is isomorphic to a CQ in B (resp. D). It follows that $A - B \cong C - D$.

(*Completeness*) Suppose $Q_1 \equiv_{\mathbb{Z}}^{\mathcal{V}} Q_2$. Then there exist terminating sequences $Q_1 \stackrel{*}{\rightarrow} Q_1'$ and $Q_2 \stackrel{*}{\rightarrow} Q_2'$, and $Q_1' \equiv_{\mathbb{Z}}^{\mathcal{V}} Q_2'$. However, Q_1' and Q_2' contain no view predicate symbols (else UNFOLD would have applied). It follows that $Q_1' \equiv_{\mathbb{Z}} Q_2'$. □

Corollary 4.4 $\mathbb{Z}$*-equivalence of* DUCQs (*and thus* $\mathcal{RA}$ *queries*) *with respect to a set of* DUCQ (*and thus* $\mathcal{RA}$) *views* $\mathcal{V}$ *is decidable.*

The corollary holds for $\mathcal{RA}$ queries and views because we can always convert them first to DUCQs. Note that converting them to DUCQs may increase the size exponentially, and there can be a separate exponential blowup when unfolding the views in the DUCQs. We leave open the exact complexity of the problems in Corollary 4.4.

Remark 4.5 Proposition 3.8 gives a *GI*-hard lower bound on the problems, and we conjecture that $\mathbb{Z}$-equivalence of $\mathcal{RA}$ queries w.r.t. $\mathcal{RA}$ views can be checked in *PSPACE* (by comparing counts of isomorphic CQs in the converted and unfolded DUCQs, without actually performing the full conversion and unfolding). Establishing the exact complexity may be difficult, for the same reasons we gave in Remark 3.5.

In addition to their use for showing decidability, the term rewrite rules can be very valuable in the search for reformulations. By the soundness property above, using the rules guarantees that we only explore equivalent rewritings. By the completeness property, if an equivalent rewriting exists, it will be reachable by a sequence of rewrite steps or—and this is the main difficulty we will have to face—*converse* rewrite steps.

Thus, in Sect. 5 we develop an enumeration algorithm for $\mathbb{Z}$-equivalent reformulations, which uses the above rewrite system combined with the converse of the UNFOLD and CANCEL operations, called FOLD and AUGMENT, respectively. FOLD rewrites a query by removing some of its CQs and replacing them with an equivalent view (recall that under bag semantics two UCQs are equivalent iff they are isomorphic.) In general we may need to AUGMENT first before FOLD is applicable.

$$\frac{P, Q, R \in \text{DUCQ} \quad V \in \mathcal{V} \quad P \to_V Q}{Q \cup R \to P \cup R} \qquad \text{(FOLD)}$$

$$\frac{A, B \in \text{CQ} \quad Q \in \text{DUCQ} \quad A \cong B}{Q \to Q \cup A \cup (-B)} \qquad \text{(AUGMENT)}$$

4.3 Diff-Irredundant Rewritings

While UNFOLD and CANCEL only give terminating sequences of rewritings, the addition of the converse rules destroys this property. For example AUGMENT can be used to "grow" a DUCQ arbitrarily by adding more and more CQs: for example, $Q \to Q \cup A \cup -A \to Q \cup A \cup B \cup -A \cup -B \to \cdots$.

Therefore the space of queries reachable in this way is infinite. How do we cut it down to a finite size? Given the success of considering locally minimal rewritings for set semantics, we could try to find a similar notion of minimality for DUCQs, hopefully one that is compatible with useful cost models. In this subsection, we define such a notion based on removing certain redundant computations, and we consider whether the space of non-redundant rewritings is finite.

Definition 4.6 A DUCQ $Q = A - B$ is said to be *diff-redundant* if for some subset of CQs $A' \subseteq A$ and $B' \subseteq B$, we have $A' \equiv_{\mathbb{Z}}^{\mathcal{V}} B'$. In this case the pair of terms A', B' is said to be *diff-redundant*.

In the example above, $Q \cup A \cup -A$ and $Q \cup A \cup B \cup -A \cup -B$ are both diff-redundant.

A diff-redundant DUCQ can be *minimized* by repeatedly finding and removing diff-redundant pairs of terms until a *diff-irredundant* query is obtained. In the examples above, this leads back to Q (assuming Q itself is irredundant). More generally:

Proposition 4.7 *If Q, Q' are* DUCQ*s that do not contain view predicates and $Q \equiv_{\mathbb{Z}} Q'$, then minimizing Q and minimizing Q' produces the same query* (*up to isomorphism*).

However, when queries may contain view predicates, this property fails dramatically:

Theorem 4.8 *The set $\mathsf{Irr}_{\mathcal{V}}(Q)$ of diff-irredundant rewritings of a* DUCQ *Q with respect to a set of views $\mathcal{V}$ is in general infinite.*

Proof If R and S are binary relations, then denote by RS the *relational composition* of R and S, i.e., the query:

$$Q(x, y) \text{ :- } R(x, z), S(z, y)$$

We will use exponents to mean repeated relational composition (e.g., $R^3 = RRR$), and to simplify notation we use here $+$ for union.

Now let $Q = R^2$, and let $\mathcal{V}$ contain the single view $V = R + R^3$. Then we have:

$$\begin{aligned} Q &= R^2 \\ &\equiv^{\mathcal{V}}_{\mathbb{Z}} VR - R^4 \\ &\equiv^{\mathcal{V}}_{\mathbb{Z}} VR - VR^3 + R^6 \\ &\equiv^{\mathcal{V}}_{\mathbb{Z}} VR - VR^3 + VR^5 - R^8 \end{aligned}$$

and more generally:

$$Q \equiv^{\mathcal{V}}_{\mathbb{Z}} (-1)^n R^{2(n+1)} + \sum_{i=1}^{n} (-1)^{i+1} VR^{2i-1}$$

for all $n \geq 0$. Clearly, there are infinitely many rewritings of this form. Moreover, one can check that every such rewriting is diff-irredundant. It follows that $\mathsf{Irr}_{\mathcal{V}}(Q)$ is infinite. □

Thus, considering diff-irredundant queries does not suffice to establish a finite bound on the set of possible rewritings for a DUCQ.

4.4 A Cost Model

Another notion of minimality we might consider is the *global minimality* of [28], where the goal is to minimize the *total number of atoms* in a CQ reformulated using views. We find this notion problematic for several reasons. First, in contrast to the classical CQ minimization techniques (where minimizing the number of atoms coincides with computing the *core* of the query), the mathematical justification is unclear. Second, it is not clear how global minimality should be extended to UCQs/DUCQs (total number of atoms in all CQs?). Third, in the practical applications of interest

to us, the real goal is to minimize the *cost* of executing a query, and the number of atoms is often not indicative of query performance. This is because of many factors, especially the different costs of computing with different source relations (which may be of drastically different cardinalities, have different indexes, etc.).

To illustrate this last point, recall that in Sect. 2 we had a view maintenance example, where the original view was defined as:

$$V(x, y) \text{ :- } R(x, y), R(z, y)$$

and our updated view could be defined as:

$$V'(x, y) \text{ :- } R'(x, y), R'(z, y)$$

where $R'(x, y)$ represents $R(x, y) \cup R^{\Delta}(x, y)$. The reformulation problem has two materialized views: the original V in terms of the base R, and R' (R after updates are applied, which we can compute either before or after computing V'). If R^{Δ} is *large*, then V' may be most efficient to compute in terms of R', as specified in the second rule above. Alternatively, if R^{Δ} is *small*, then it may be more efficient to compute V' using V and delta rules:

$$V'(x, y) \text{ :- } V(x, y)$$
$$V'(x, y) \text{ :- } R^{\Delta}(x, z), R(z, y)$$
$$V'(x, y) \text{ :- } R(x, z), R^{\Delta}(z, y)$$

Note that the latter query has three rules compared to only one in the first case, and five atoms to only two. Yet, intuitively, for small R^{Δ}s, the second case is more efficient.

In practical DBMS implementations, a *cost model* [35] predicts query performance given known properties (such as cardinalities) of the input relations. Thus we will seek here an abstract cost model that captures the essence of the detailed models implemented in real optimizers while allowing us to establish finite bounds on the set of possible rewritings for DUCQs using the FOLD + AUGMENT + UNFOLD + CANCEL rules.

Our cost model, $\mathsf{cost} : \mathcal{RA} \to \mathbb{N}$, is instance-dependent, and makes use of external calls to a *cardinality estimation function*, $\mathsf{card} : \mathcal{RA} \to \mathbb{N}$, which returns the estimated number of tuples in the result of a subquery (for the current database instance). We define cost inductively on $\mathcal{RA}$ expressions:

$$\begin{aligned}
\mathsf{cost}(R) &= \mathsf{card}(R) \\
\mathsf{cost}(\pi E) &= \mathsf{cost}(E) \\
\mathsf{cost}(\sigma E) &= \mathsf{cost}(E) \\
\mathsf{cost}(E_1 \cup E_2) &= \mathsf{cost}(E_1) + \mathsf{cost}(E_2) \\
\mathsf{cost}(E_1 - E_2) &= \mathsf{cost}(E_1) + \mathsf{cost}(E_2) \\
\mathsf{cost}(E_1 \ltimes E_2) &= \mathsf{cost}(E_1) + \mathsf{cost}(E_2) \\
&\quad + \mathsf{card}(E_1 \ltimes E_2)
\end{aligned}$$

This cost model intuitively focuses on the cost of *producing* new tuples in expensive operations, namely joins and tablescans. (Union in $\mathbb{Z}$-semantics is inexpensive, as it is in bag-semantics; difference in $\mathbb{Z}$-semantics is in fact a union operation that negates counts, and hence it is also inexpensive.) Our cost model essentially computes a lower bound on the amount of work a join must perform: it considers tuple creation cost but ignores the cost of matching tuples. (For an index nested loops join or a hash join, matching is in fact inexpensive, so this is a fairly realistic lower bound.) Our model satisfies the *principle of optimality* [11] required by a real query optimizer cost model.

The cost model above is based on queries represented as relational algebra expressions (with a specific order of evaluation). However it can be extended to CQs by defining the cost of a CQ as the cost of the cheapest equivalent algebraic expression,[4] and to UCQs and DUCQs by summing the costs of all the CQs in the UCQ or DUCQ. Assuming every source relation is non-empty, a simple observation is that the cost of any CQ is at least equal to the number of atoms in the CQ, and any DUCQ Q has a cost that is at least equal to the number of CQs in Q.

Now we can establish a bound on the search space in which we wish to look for rewritings of *minimum cost*:

Proposition 4.9 (Completeness under cost) *Let $Q \in$ DUCQ, and let $\mathcal{V}$ be a set of* DUCQ *views. Let $k = \mathsf{cost}(Q)$, and let w be the maximum number of* CQs *in Q or in a view in $\mathcal{V}$. Assume w.l.o.g. that Q does not contain any view predicates* (*they can always be removed by applying* UNFOLD *repeatedly*). *Then, any* DUCQ *of* minimum cost *among those equivalent to Q using $\mathcal{V}$ can be reached from Q in at most $O(kw^k)$ steps of* (FOLD + AUGMENT)*-rewriting.*

Proof Suppose that Q' is a DUCQ reformulation of Q of minimum cost. Then using our observations above, Q' contains at most k CQs each containing at most k view predicates. We can rewrite Q' into Q by a sequence of u UNFOLD steps (removing all view predicate occurrences), followed by a sequence of v CANCEL steps. (Read in the reverse direction, this will yield a sequence of v AUGMENT steps followed by a sequence of u FOLD steps, rewriting Q into Q'.)

Consider a single CQ in Q'. We claim that UNFOLD can be applied to it at most w^k times, replacing the single CQ with at most w^k CQs. Doing this for all of the $\leq k$ CQs in Q', we have $u \leq kw^k$ UNFOLD steps, and the result is a DUCQ containing $\leq kw^k$ CQs.

Next, since each application of CANCEL removes 2 CQs, it follows that CANCEL can be applied at most $\lfloor kw^k/2 \rfloor$ times. Thus, the total length of the sequence rewriting Q' to Q is at most $kw^k + \lfloor kw^k/2 \rfloor = O(kw^k)$. □

This yields a finite bound (which can be effectively computed from Q, $\mathcal{V}$, and cost) on the region of the rewrite space that must be explored in order to find a rewriting of minimum cost (for a given instance).

[4] This is in fact done by dynamic programming in real query optimizers.

5 Finding Query Rewritings

Now that we understand the conditions under which reformulation can be bounded to a finite search space, we develop an enumeration algorithm for exploring the space of possible query reformulations, for the problems of *optimizing queries using views*, *view adaptation*, and *view maintenance*. Recall from Sect. 2 that the *optimizing queries using views* problem takes a set of materialized views, plus a query to be reformulated, as input. *View adaptation* takes a single materialized view (the "old" view definition and materialized instance), with the modified view definition as the query to be reformulated. *View maintenance* takes the pre-updated view instance and updated versions of the base relations as input views, with the maintained view (i.e., the view computed over the updated base relations) as the query to be reformulated.

5.1 Reformulation Algorithm

Section 4.2 presented four rewrite rules, UNFOLD, CANCEL, AUGMENT and FOLD, which are required for enumerating the space of plans. However, as they are currently specified, they can lead to inefficient exploration. We observe that in isolation AUGMENT can add/subtract arbitrary CQs to the query, regardless of whether this will turn out to be "useful" (by enabling a subsequent FOLD). Hence we do not use AUGMENT directly, but rather define a compound AUGMENT-FOLD operation, which augments the query *only* with the CQs necessary to perform a FOLD operation with a given view. (The AUGMENT step of AUGMENT-FOLD may be skipped if FOLD can be directly applied to the query.) Additionally, we restrict AUGMENT-FOLD to apply only if there exists at least one rule in common between the query and the view. After applying AUGMENT-FOLD, we may be able to CANCEL some rules introduced into the query. Hence, we will also always apply CANCEL after an AUGMENT-FOLD repeatedly until it is no longer applicable. We denote the rule which corresponds to this sequence of AUGMENT-FOLD followed by repeated CANCEL by AUGMENT-FOLD-CANCEL.

Algorithm 1 shows how the UNFOLD, CANCEL, and AUGMENT-FOLD-CANCEL primitives can be composed to enumerate the space of plans. The *normalize* function (omitted) repeatedly applies UNFOLD and CANCEL to the input query until they are no longer applicable. Next, the main loop simply applies AUGMENT-FOLD-CANCEL to rewrite portions of each "frontier" query q in F, in terms of any views that overlap with q.

The *prune* function determines whether the rewritten query q' should be added to the frontier set, or disregarded during exploration. For the rewrite algorithm to be guaranteed to terminate and find an optimal rewriting according to the cost model, it suffices to define $prune(q, k)$ to return **true** iff k is less than the bound given by Proposition 4.9. (In practice, we would add additional heuristics to *prune* to limit the search space, sacrificing the guarantee of a minimum-cost rewriting.) Once the full space has been explored, *reformulate* returns the rewriting from Q with the lowest cost according to our cost model.

Algorithm 1 *Reformulate* (query q, set of views V) returns query

```
1: for each view v in V do
2:     v := normalize(v)
3: end for
4: Let q_i := normalize(q)
5: Let Q := ∅
6: Let F := {(q_i, 0)}
7: for each (q, k) in F do
8:     Remove (q, k) from F and add it to Q
9:     for each v in V do
10:        Let q' := AUGMENT-FOLD-CANCEL(q,v)
11:        if q' not in Q and not prune(q', k + 1) then
12:            Add (q', k + 1) to F
13:        end if
14:    end for
15: end for
16: return the q in Q with lowest cost
```

5.2 Rewriting in a Query Optimizer

In principle, one could search the space of query rewritings by building a layer above the query optimizer, which enumerates possible rewritings; then separately optimizes each. However, a more efficient approach is to *extend an existing optimizer* to incorporate the rewriting system into its enumeration. Unlike with rewriting of conjunctive queries, most existing cost-based optimizers cannot easily be extended to DUCQ rewritings. The System-R optimizer [35] only does cost-based optimization of *joins*, instead relying on heuristics for applying unions and differences. Starburst [21] can rewrite queries with unions and differences, but only at its heuristics-based *query rewrite* stage. Starburst only has limited facilities for cost-based comparison of alternative rewritings.

Fortunately, the Volcano [14] optimizer generator (as well as its successors) can be modified to incorporate our rewrite scheme within an optimizer. Volcano models a query initially as a plan of *logical operators* representing the algebraic operations, including unions and differences; it uses *transformation rules* to describe algebraic equivalences that can be used to find alternate plans. *Implementation rules* describe how to rewrite a logical operator (or set of operators) into a series of physical algorithms, which have associated costs.

Our rewrite rules of Sect. 4.2 can be expressed as transformation rules for Volcano, and we would not need to change any implementation rules. However, we would also need to modify Volcano's pruning algorithm: we must place a finite bound on the size of the rewritings explored, as with our *prune* function in the previous subsection.

6 Applications to Bag and Set Semantics

Having obtained a sound and complete algorithm for reformulation of $\mathcal{RA}$ queries using $\mathcal{RA}$ views under $\mathbb{Z}$-semantics, we wish to see for which $\mathcal{RA}$ queries/views

this same algorithm actually provides reformulations under bag or set semantics. We begin with bag semantics, where we are naturally led to study the following class of queries:

Definition 6.1 We denote by $\overline{\mathcal{RA}}$ the class of all queries $Q \in \mathcal{RA}$ such that for all bag-instances I, $[\![Q]\!]^I_{\mathbb{N}} = [\![Q]\!]^I_{\mathbb{Z}}$.

Right away we see that:

Lemma 6.2 *For any $Q_1, Q_2 \in \overline{\mathcal{RA}}$ we have $Q_1 \equiv_{\mathbb{Z}} Q_2$ iff $Q_1 \equiv_{\mathbb{N}} Q_2$.*

Proof Straightforward consequence of Lemma 3.6 and the definition of $\overline{\mathcal{RA}}$. □

Then

Lemma 6.3 *If $A, B \in \mathcal{RA}^+$ then $A - B \in \overline{\mathcal{RA}}$ if and only if $B \sqsubseteq_{\mathbb{N}} A$.*

Proof "⇒": suppose $A - B \in \overline{\mathcal{RA}}$ and consider an arbitrary bag instance I and tuple t. Since $[\![A - B]\!]^I_{\mathbb{Z}}(t) = [\![A - B]\!]^I_{\mathbb{N}}(t) \geq 0$, and by definition, $[\![A - B]\!]^I_{\mathbb{Z}}(t) = [\![A]\!]^I_{\mathbb{Z}}(t) - [\![B]\!]^I_{\mathbb{Z}}(t)$, it follows that $[\![A]\!]^I_{\mathbb{Z}}(t) \geq [\![B]\!]^I_{\mathbb{Z}}(t)$. Since A and B are positive queries, by Lemma 3.1, $[\![A]\!]^I_{\mathbb{Z}} = [\![A]\!]^I_{\mathbb{N}}$ and $[\![B]\!]^I_{\mathbb{Z}} = [\![B]\!]^I_{\mathbb{N}}$ and therefore $[\![A]\!]^I_{\mathbb{N}}(t) \geq [\![B]\!]^I_{\mathbb{N}}(t)$. Since I and t were chosen arbitrarily, it follows that $B \sqsubseteq_{\mathbb{N}} A$.

"⇐": suppose $B \sqsubseteq_{\mathbb{N}} N$ and consider an arbitrary bag instance I. By Lemma 3.1 $[\![A]\!]^I_{\mathbb{N}} = [\![A]\!]^I_{\mathbb{Z}}$ and $[\![B]\!]^I_{\mathbb{N}} = [\![B]\!]^I_{\mathbb{Z}}$. But since $[\![B]\!]^I_{\mathbb{N}} \leq [\![A]\!]^I_{\mathbb{N}}$, it follows that $[\![A - B]\!]^I_{\mathbb{Z}} = [\![A - B]\!]^I_{\mathbb{N}} \geq 0$. Since I was chosen arbitrarily, it follows that $A - B \in \overline{\mathcal{RA}}$. □

Proposition 6.4 *If $Q \in \overline{\mathcal{RA}}$ then $\mathsf{DiffNF}(Q) \in \overline{\mathcal{RA}}$ and $Q \equiv_{\mathbb{N}} \mathsf{DiffNF}(Q)$.*

Proof Suppose $Q \in \overline{\mathcal{RA}}$, let $\mathsf{DiffNF}(Q) = A - B$ (with $A, B \in \mathcal{RA}^+$), and choose an arbitrary bag instance I and tuple t. Then $[\![Q]\!]^I_{\mathbb{N}}(t) = [\![Q]\!]^I_{\mathbb{Z}}(t) = [\![A - B]\!]^I_{\mathbb{Z}}(t) = [\![A]\!]^I_{\mathbb{Z}}(t) - [\![B]\!]^I_{\mathbb{Z}}(t) \geq 0$. It follows that $[\![A]\!]^I_{\mathbb{Z}}(t) \geq [\![B]\!]^I_{\mathbb{Z}}(t)$ and therefore (using Lemma 3.1) that $[\![A - B]\!]^I_{\mathbb{Z}}(t) = [\![A - B]\!]^I_{\mathbb{N}}(t)$. Since I and t were chosen arbitrarily, it follows that $Q \equiv_{\mathbb{N}} A - B$, as required, and also that $B \sqsubseteq_{\mathbb{N}} A$. By Lemma 6.3 this in turn implies that $A - B \in \overline{\mathcal{RA}}$. □

Corollary 6.5 *Bag-equivalence of $\overline{\mathcal{RA}}$ queries is decidable.*

Next we show that if we start with a DUCQ in $\overline{\mathcal{RA}}$ and a set of DUCQ views also in $\overline{\mathcal{RA}}$, then the exploration of the space of reformulations prescribed by the algorithm of Sect. 5 examines only queries in $\overline{\mathcal{RA}}$ which are bag-equivalent under the views to the original DUCQ.

Suppose that $\mathcal{V}$ is a set of views in $\overline{\mathcal{RA}}$ expressed as DUCQs.

Theorem 6.6 *If Q and the views in $\mathcal{V}$ are* DUCQs *in $\overline{\mathcal{RA}}$, then the reformulation algorithm of Sect. 4.2 is sound and complete with respect to $\mathbb{N}$-equivalent $\overline{\mathcal{RA}}$ reformulations of Q.*

Proof Use Lemmas 6.2 and 6.3 and Proposition 4.3. □

Therefore, the reformulation algorithm can be used with $\overline{\mathcal{RA}}$ queries and views. Unfortunately, but not unexpectedly, it is undecidable whether an $\mathcal{RA}$ query or even a DUCQ is actually in $\overline{\mathcal{RA}}$ (Lemma 6.3 provides a reduction from bag-containment of UCQs). But there are interesting classes of queries for which membership in $\overline{\mathcal{RA}}$ is guaranteed. The simplest but still very useful case is based on the observation that $\mathcal{RA}^+ \subset \overline{\mathcal{RA}}$. It follows that our algorithm is also complete for finding DUCQ reformulations of UCQs using UCQ views, as in the example in Sect. 2.

Next we identify a subclass of $\overline{\mathcal{RA}}$ for which, while still undecidable in general, membership may be easier to check in certain cases.

Definition 6.7 We denote by $\widehat{\mathcal{RA}}$ the class of all $\mathcal{RA}$ queries Q such that for every occurrence $A - B$ of the difference operator in Q, we have $B \sqsubseteq_{\mathbb{N}} A$.

Theorem 6.8 (Normalization for $\widehat{\mathcal{RA}}$) *For any $Q \in \widehat{\mathcal{RA}}$, one can find $A, B \in \mathcal{RA}^+$ such that $Q \equiv_{\mathbb{N}} A - B$.*

Proof Straightforward induction on Q, using the same algebraic identities as in Theorem 3.3. Although these identities fail under bag semantics for $\mathcal{RA}$ queries, they hold for $\widehat{\mathcal{RA}}$ queries. □

Corollary 6.9 *Bag-equivalence of $\widehat{\mathcal{RA}}$ queries is decidable.*

Theorem 6.10 $\widehat{\mathcal{RA}} \subset \overline{\mathcal{RA}}$.

Proof Checking $\widehat{\mathcal{RA}} \subseteq \overline{\mathcal{RA}}$ is a straightforward application of Theorem 6.8. To see that the inclusion is proper, consider the query $Q \stackrel{\text{def}}{=} (R - (R \cup R)) - (R - (R \cup R))$. Q is both $\mathbb{Z}$-equivalent and $\mathbb{N}$-equivalent to the unsatisfiable query $\emptyset$; it follows that Q is in $\overline{\mathcal{RA}}$. However, since $R \cup R \not\sqsubseteq_{\mathbb{N}} R$, it is clear that $Q \notin \widehat{\mathcal{RA}}$. □

Although $\overline{\mathcal{RA}}$ contains queries which are not in $\widehat{\mathcal{RA}}$ (because of their syntactic structure), it turns out that, semantically, $\widehat{\mathcal{RA}}$ *captures* $\overline{\mathcal{RA}}$, as the following theorem makes precise:

Theorem 6.11 *For all $Q \in \overline{\mathcal{RA}}$ there exists $Q' \in \widehat{\mathcal{RA}}$ such that $Q \equiv_{\mathbb{N}} Q'$.*

Proof For any $Q \in \overline{\mathcal{RA}}$, we show that there must exist $A, B \in \mathcal{RA}^+$ such that $B \sqsubseteq_{\mathbb{N}} A$ and $Q \equiv_{\mathbb{N}} A - B$. Fix a $Q \in \overline{\mathcal{RA}}$. By Theorem 3.3, there exist $A, B \in \mathcal{RA}^+$ such that $Q \equiv_{\mathbb{Z}} A - B$. We argue by contradiction that $B \sqsubseteq_{\mathbb{N}} A$. Suppose $B \not\sqsubseteq_{\mathbb{N}} A$. Then there exists a bag instance I and tuple t such that $[\![A]\!]^I_{\mathbb{N}}(t) < [\![B]\!]^I_{\mathbb{N}}(t)$. Then $[\![A - B]\!]^I_{\mathbb{Z}}(t) < 0$, i.e., $[\![A - B]\!]^I_{\mathbb{Z}} = [\![Q]\!]^I_{\mathbb{Z}}$ is not even a bag-instance. However, this is a

contradiction, because by assumption (since $Q \in \overline{\mathcal{RA}}$), we must have $[\![Q]\!]^I_{\mathbb{Z}} = [\![Q]\!]^I_{\mathbb{N}}$. Finally, we argue that $Q \equiv_{\mathbb{N}} A - B$. To see this, fix an arbitrary bag instance I. We want to show that $[\![Q]\!]^I_{\mathbb{N}} = [\![A - B]\!]^I_{\mathbb{N}}$. Since $Q \in \overline{\mathcal{RA}}$, we have $[\![Q]\!]^I_{\mathbb{N}} = [\![Q]\!]^I_{\mathbb{Z}}$. Since $Q \equiv_{\mathbb{Z}} A - B$ we have $[\![Q]\!]^I_{\mathbb{Z}} = [\![A - B]\!]^I_{\mathbb{Z}}$. Finally, since $A - B \in \widehat{\mathcal{RA}}$, by Theorem 6.10 we have $[\![A - B]\!]^I_{\mathbb{Z}} = [\![A - B]\!]^I_{\mathbb{N}}$. This completes the proof. □

Membership in $\widehat{\mathcal{RA}}$ is also undecidable. However in some practical situations, such as incremental view maintenance of $\mathcal{RA}^+$ views using delta rules [20], the difference operator is used in a very controlled way where the containment requirement is satisfied (e.g., it is just necessary for the system to enforce that only tuples actually present in source tables are ever deleted from the sources).

6.1 Set Semantics

We are also interested in reformulating and answering queries under $\mathbb{Z}$-semantics, but then "eliminating duplicates" to obtain the answer under set semantics. Even for $\widehat{\mathcal{RA}}$ queries, this is not in general straightforward: for example, consider the query $Q = (R \cup R) - R$. Under set semantics, this is equivalent to the unsatisfiable query, while under bag semantics or $\mathbb{Z}$-semantics, it is equivalent to the identity query R. We can, however, restrict the use of negation in $\widehat{\mathcal{RA}}$ further to obtain another fragment of $\mathcal{RA}$ suitable for this purpose.

Definition 6.12 An $\mathcal{RA}$ query Q over a schema Σ is said to be a *base-difference* query if $A - B$ can appear in Q only when A and B are both base relations (names in Σ). Further, a base-difference query Q is said to be *positive-difference* w.r.t. a set-instance I if for each $A - B$ appearing in Q we have $A^I \supseteq B^I$ (where A^I is the relation in I that corresponds to $A \in \Sigma$).

Although the use of negation in base-difference queries considered on instances w.r.t. which they are positive-difference is highly restricted, it still captures the form needed for incremental view maintenance, where negation just relates old and new versions of source relations via the tables of deleted and inserted tuples.

For conversion between bag semantics/$\mathbb{Z}$-semantics and set semantics we also define the *duplicate elimination* operator $\delta : \mathbb{Z} \to \mathbb{B}$ which maps 0 to false and everything else (positive or negative) to true. Conversely, we can view any set instance as a bag/$\mathbb{Z}$ instance by replacing false with 0 and true with 1. With this we can state the salient property of base-difference queries:

Proposition 6.13 *. Let $Q \in \mathcal{RA}$ be a base-difference query and let I be a set instance w.r.t. which Q is positive-difference. Then, we can compute $[\![Q]\!]^I$ under set semantics by viewing I as a $\mathbb{Z}$-instance, computing $[\![Q]\!]^I_{\mathbb{Z}}$, and finally applying δ.*

Consequently, the optimization techniques in this paper (which replaces a query with a $\mathbb{Z}$-equivalent one) will also apply to set semantics, provided we restrict ourselves to base-difference queries applied to instances w.r.t. which they are positive-difference.

7 Built-in Predicates

To this point, our approach to query rewriting has assumed equality predicates only. Clearly, any practical implementation would also consider inequality ($<, \leq$) and non-equality ($\neq$) predicates. In this section we discuss the extensions necessary to support such *built-in predicates*.

We assume our domain $\mathbb{D}$ comes equipped with a dense linear order $<$, and we define $\mathcal{RA}^<$, $\widehat{\mathcal{RA}}^<$, CQ$^<$, etc. as the previously defined classes of queries extended to allow use of the predicates $<, \leq, =$, and $\neq$. In general, the predicates in a CQ$^<$ induce only a partial order on the variables. We shall call a CQ$^<$ *total* if the predicates in the query induce a total order on the variables, and *partial* otherwise. To facilitate syntactic comparison of queries we shall assume w.l.o.g. for total CQ$^<$s that a minimal number of predicate atoms are used, i.e., if the predicates induce the total order $x < y < z$ then the predicate atoms $x < y$ and $y < z$ and no others appear in the query. A UCQ$^<$ or DUCQ$^<$ is *total* if all of its CQ$^<$s are total, and *partial* if it contains a partial CQ$^<$.

As in [8, 9], we note that a partial CQ$^<$ Q can always be converted into an equivalent total UCQ$^<$, denoted $\mathsf{lin}(Q)$, that contains one CQ$^<$ for each *linearization* of the partial order on the variables. For example:

$$Q(x, y) \text{ :- } R(x, y), R(y, z), x < y, x \leq z$$

can be rewritten into:

$$Q(x, y) \text{ :- } R(x, y), R(y, z), x = z, x < y$$
$$Q(x, y) \text{ :- } R(x, y), R(y, z), x < y, y = z$$
$$Q(x, y) \text{ :- } R(x, y), R(y, z), x < y, y < z$$
$$Q(x, y) \text{ :- } R(x, y), R(y, z), x < z, z < y$$

Likewise a partial UCQ$^<$ (DUCQ$^<$) Q can be converted into an equivalent total UCQ$^<$ (DUCQ$^<$) $\mathsf{lin}(Q)$ by replacing each partial CQ$^<$ with its equivalent total UCQ$^<$. Note that if Q is already total, then $Q = \mathsf{lin}(Q)$.

Theorem 7.1 *For all $Q, Q' \in$ UCQ$^<$ the following are equivalent*:

1. $Q \equiv_{\mathbb{N}} Q'$
2. $Q \equiv_{\mathbb{Z}} Q'$
3. $\mathsf{lin}(Q) \cong \mathsf{lin}(Q')$

Proof (2) $\Rightarrow$ (1) because every bag instance is a $\mathbb{Z}$-instance. (1) $\Rightarrow$ (3) follows by results of [9]. (3) $\Rightarrow$ (1) follows from the observation that $Q \equiv_{\mathbb{Z}} \mathsf{lin}(Q)$. □

Corollary 7.2 *$\mathbb{Z}$-equivalence of $\mathcal{RA}^<$ queries is decidable and so is bag-equivalence of $\widehat{\mathcal{RA}}^<$ queries.*

This leads to an approach to enumerating rewritings of queries with predicates with respect to views: linearize the queries and views into total UCQs/DUCQs as above

and reformulate using the linearized representations. As an optional final step, the reformulated query could then be "de-linearized" to a partial query.

8 Related Work

Exact query reformulation using views has been studied extensively, due to its applications in query optimization, data integration, and view maintenance, starting with the papers by Levy et al. [28] and Chaudhuri et al. [4]. The former paper established fundamental results for UCQ$^<$ under set semantics. The latter paper considered CQ$^<$s under bag semantics, but it did not provide a complete reformulation algorithm or consider UCQs or UCQ$^<$s. Cohen, Nutt, and Sagiv [7] considered the problem for CQ$^<$s with aggregate operators and built-in predicates under bag-set semantics, and developed sound and complete reformulation algorithms. In contrast to our term-rewrite system based approach, which considers only equivalent rewritings, their algorithm considers non-equivalent candidate rewritings, using an equivalence check to filter candidates. Afrati and Pavlaki [2] give results on rewriting with views for CQs with *safe negation*. These queries/views are considered under set semantics but their expressive power seems to be incomparable to that of the queries we consider in Proposition 6.13.

The seminal paper by Chandra and Merlin [3] introduced the fundamental concepts of containment mappings and canonical databases in showing the decidability of containment of CQs under set semantics and identifying its complexity as *NP*-complete. The extension to UCQs is due to Sagiv and Yannakakis [34], where the undecidability of set-equivalence of $\mathcal{RA}$ queries was also established.

The papers by Ioannidis and Ramakrishnan [24] and Chaudhuri and Vardi [5] initiated the study of query optimization under bag semantics. Chaudhuri and Vardi showed that bag-equivalence of CQs is the same as isomorphism and established the Π_2^p-hardness of checking bag-containment of CQs. Ioannidis and Ramakrishnan showed that bag-containment of UCQs is undecidable. The decidability of bag-equivalence of UCQs can be derived from the results on bag-set semantics in [6, 9] and also from results on provenance-annotated semantics (see the discussion in [15]). The decidability of bag-containment of CQs remains open. Recent progress was made on the problem by Jayram, Kolaitis and Vee [25] who established the undecidability of checking bag-containment of CQs with built-in predicates (our CQ$^<$s).

Chaudhuri and Vardi also introduced in [5] the study of bag-set semantics (where source tuple multiplicities are 0 or 1 only, and queries are evaluated under bag semantics), and showed that bag-set equivalence of CQs is the same as isomorphism. This was essentially a rediscovery of a much earlier result due to Lovász [30] (see also [23]). Cohen, Nutt, and Sagiv [8] give decidability results for bag-set equivalence of CQs with comparisons and aggregate operators. Cohen, Sagiv, and Nutt [10] give decidability results for bag-set equivalence of UCQs with comparisons, aggregate operators, and a limited form of negation (only on extensional predicates).

The view adaptation problem was introduced in [19], which gives a case-based algorithm for adapting materialized views under changes to view definitions (under bag semantics). In contrast, our methods apply to view adaptation, but use a more

general term rewrite system to develop a sound and complete query reformulation algorithm.

Our $\mathbb{Z}$-relations appeared in an early form as the *deltas* in the count incremental view maintenance algorithm for UCQs of [20]. That paper did not consider query equivalence for deltas or make a general study of query reformulation.

More recently, a paper by Koch [27] presents new strategies for performing incremental view maintenance based on a "ring of databases." Like us, the approach of [27] involves constructing an analogue of bag relational algebra with a difference operator that behaves as an inverse of the union operator. However, there is a fundamental technical difference in that [27] considers untyped relations, allowing union and join to be defined as total operations and yielding a ring structure over the domain of untyped database instances.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading (1995)
2. Afrati, F., Pavlaki, V.: Rewriting queries using views with negation. AI Commun. **19**, 229–237 (2006)
3. Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational data bases. In: STOC (1977)
4. Chaudhuri, S., Krishnamurthy, R., Potamianos, S., Shim, K.: Optimizing queries with materialized views. In: ICDE (1995)
5. Chaudhuri, S., Vardi, M.Y.: Optimization of *real* conjunctive queries. In: PODS, pp. 59–70 (1993)
6. Cohen, S.: Containment of aggregate queries. SIGMOD Rec. **34**(1), 77–85 (2005)
7. Cohen, S., Nutt, W., Sagiv, Y.: Rewriting queries with arbitrary aggregation functions using views. ACM Trans. Database Syst. **31**(2), 672–715 (2006)
8. Cohen, S., Nutt, W., Sagiv, Y.: Deciding equivalences among conjunctive aggregate queries. J. ACM 54(2) (2007)
9. Cohen, S., Nutt, W., Serebrenik, A.: Rewriting aggregate queries using views. In: PODS (1999)
10. Cohen, S., Sagiv, Y., Nutt, W.: Equivalences among aggregate queries with negation. ACM Trans. Comput. Log. **6**(2), 328–360 (2005)
11. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. McGraw-Hill/MIT Press, New York/Cambridge (1990)
12. Deutsch, A., Popa, L., Tannen, V.: Query reformulation with constraints. SIGMOD Rec. **35**(1), 65–73 (2006)
13. Duschka, O.M., Genesereth, M.R.: Answering recursive queries using views. In: PODS (1997)
14. Graefe, G., McKenna, W.J.: The Volcano optimizer generator: extensibility and efficient search. In: ICDE (1993)
15. Green, T.J.: Containment of conjunctive queries on annotated relations. In: ICDT (2009)
16. Green, T.J., Ives, Z.G., Tannen, V.: Reconcilable differences. In: ICDT (2009)
17. Green, T.J., Karvounarakis, G., Ives, Z.G., Tannen, V.: Update exchange with mappings and provenance. In: VLDB (2007). Amended version available as Univ. of Pennsylvania report MS-CIS-07-26
18. Green, T.J., Karvounarakis, G., Tannen, V.: Provenance semirings. In: PODS (2007)
19. Gupta, A., Mumick, I.S., Rao, J., Ross, K.A.: Adapting materialized views after redefinitions: techniques and a performance study. Inf. Syst. **26**(5), 323–362 (2001)
20. Gupta, A., Mumick, I.S., Subrahmanian, V.S.: Maintaining views incrementally. In: SIGMOD (1993)
21. Haas, L.M., Freytag, J.C., Lohman, G.M., Pirahesh, H.: Extensible query processing in Starburst. In: SIGMOD (1989)
22. Halevy, A.Y.: Answering queries using views: a survey. VLDB J. **10**(4) (2001)
23. Hell, P., Nešetřil, J.: Graphs and Homomorphisms. Oxford University Press, London (2004)
24. Ioannidis, Y.E., Ramakrishnan, R.: Containment of conjunctive queries: beyond relations as sets. TODS **20**(3), 288–324 (1995)
25. Jayram, T.S., Kolaitis, P.G., Vee, E.: The containment problem for *real* conjunctive queries with inequalities. In: PODS (2006)

26. Klop, J.W.: Handbook of Logic in Computer Science, vol. 2. Oxford University Press, London (1992), Chap. 1
27. Koch, C.: Incremental query evaluation in a ring of databases. In: PODS, pp. 87–98 (2010)
28. Levy, A.Y., Mendelzon, A.O., Sagiv, Y., Srivastava, D.: Answering queries using views. In: PODS (1995)
29. Levy, A.Y., Rajaraman, A., Ordille, J.J.: Querying heterogeneous information sources using source descriptions. In: VLDB (1996)
30. Lovász, L.: Operations with structures. Acta Math. Hung. **18**(3–4), 321–328 (1967)
31. Mitra, P.: An algorithm for answering queries efficiently using views. In: Proceedings of the Australasian Database Conference (2001)
32. Pottinger, R., Levy, A.: A scalable algorithm for answering queries using views. In: VLDB (2000)
33. Raz, R., Shpilka, A.: Deterministic polynomial identity testing in non-commutative models. Comput. Complex. **14**(1), 1–19 (2005)
34. Sagiv, Y., Yannakakis, M.: Equivalences among relational expressions with the union and difference operators. J. ACM **27**(4), 633–655 (1980)
35. Selinger, P.G., Astrahan, M.M., Chamberlin, D.D., Lorie, R.A., Price, T.G.: Access path selection in a relational database management system. In: SIGMOD (1979)