University of Pennsylvania
ScholarlyCommons

Database Research Group (CIS) | Department of Computer & Information Science

3-23-2009

Reconcilable Differences

Todd J. Green
University of Pennsylvania, tjgreen@cis.upenn.edu

Zachary G. Ives
University of Pennsylvania, zives@cis.upenn.edu

Val Tannen
University of Pennsylvania, val@cis.upenn.edu

Follow this and additional works at: http://repository.upenn.edu/db_research

 Part of the Databases and Information Systems Commons

Green, Todd J.; Ives, Zachary G.; and Tannen, Val, "Reconcilable Differences" (2009). *Database Research Group (CIS)*. 45.
http://repository.upenn.edu/db_research/45

Postprint version. Copyright ACM, 2009. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in:
Reconcilable Differences. Todd Green, Zachary Ives, and Val Tannen. *Proceedings of the 12th International Conference on Database Theory 2009(CDT 2009)*, March 23-26 2009, Saint Petersburg, Russia
Publisher URL: http://www.edbt.org/Proceedings/2009-StPetersburg/icdt/papers/N11F58.html

This paper is posted at ScholarlyCommons. http://repository.upenn.edu/db_research/45
For more information, please contact libraryrepository@pobox.upenn.edu.

Reconcilable Differences

Abstract
Exact query reformulation using views in positive relational languages is well understood, and has a variety of applications in query optimization and data sharing. Generalizations to larger fragments of the relational algebra (RA) --- specifically, support for the difference operator --- would increase the options available for query reformulation, and also apply to view adaptation (updating a materialized view in response to a modified view definition) and view maintenance. Unfortunately, most questions about queries become undecidable in the presence of difference/negation. We present a novel way of managing this difficulty via an excursion through a non-standard semantics, Z-relations, where tuples are annotated with positive or negative integers.

We show that under Z-semantics RA queries have a normal form as a single difference of positive queries and this leads to the decidability of equivalence. In most real-world settings with difference, it is possible to convert the queries to this normal form. We give a sound and complete algorithm that explores all reformulations of an RA query (under Z-semantics) using a set of RA views, finitely bounding the search space with a simple and natural cost model. We investigate related complexity questions, and we also extend our results to queries with built-in predicates.

Z-relations are interesting in their own right because they capture updates and data uniformly. However, our algorithm turns out to be sound and complete also for bag semantics, albeit necessarily only for a subclass of RA. This subclass turns out to be quite large and covers generously the applications of interest to us. We also show a subclass of RA where reformulation and evaluation under Z-semantics can be combined with duplicate elimination to obtain the answer under set semantics.

Keywords
database theory, view maintenance, view adaptation

Disciplines
Databases and Information Systems

Comments
Postprint version. Copyright ACM, 2009. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in:

Reconcilable Differences. Todd Green, Zachary Ives, and Val Tannen. *Proceedings of the 12th International Conference on Database Theory 2009(CDT 2009)*, March 23-26 2009, Saint Petersburg, Russia

Publisher URL: http://www.edbt.org/Proceedings/2009-StPetersburg/icdt/papers/N11F58.html

This conference paper is available at ScholarlyCommons: http://repository.upenn.edu/db_research/45

Reconcilable Differences

Todd J. Green Zachary G. Ives Val Tannen
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104
{tjgreen,zives,val}@cis.upenn.edu

ABSTRACT

Exact query reformulation using views in positive relational languages is well understood, and has a variety of applications in query optimization and data sharing. Generalizations to larger fragments of the relational algebra ($\mathcal{RA}$) — specifically, support for the difference operator — would increase the options available for query reformulation, and also apply to *view adaptation* (updating a materialized view in response to a modified view definition) and *view maintenance*. Unfortunately, most questions about queries become undecidable in the presence of difference/negation. We present a novel way of managing this difficulty via an excursion through a non-standard semantics, $\mathbb{Z}$*-relations*, where tuples are annotated with positive or negative integers.

We show that under $\mathbb{Z}$-semantics $\mathcal{RA}$ queries have a normal form as a single difference of positive queries and this leads to the decidability of equivalence. In most real-world settings with difference, it is possible to convert the queries to this normal form. We give a sound and complete algorithm that explores all reformulations of an $\mathcal{RA}$ query (under $\mathbb{Z}$-semantics) using a set of $\mathcal{RA}$ views, finitely bounding the search space with a simple and natural cost model. We investigate related complexity questions, and we also extend our results to queries with built-in predicates.

$\mathbb{Z}$-relations are interesting in their own right because they capture updates and data uniformly. However, 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.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the ACM. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permissions from the publisher, ACM.
ICDT 2009, March 23–25, 2009, Saint Petersburg, Russia.
Copyright 2009 ACM 978-1-60558-423-2/09/0003 ...$5.00

1. INTRODUCTION

Query reformulation (rewriting) 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 under bag semantics (see, e.g., [4, 24]). Reformulation is of both theoretical and practical interest, due to its importance to data integration (specifically, local-as-view and GLAV query reformulation [19]) and to query optimization with materialized views. Our interest is with *exact* reformulation, which finds only equivalent rewritings of the query.

Under bag-semantics ("real" queries [5]), given a UCQ Q and a finite set $\mathcal{V}$ of UCQs, there are only finitely many queries equivalent to Q under $\mathcal{V}$. This follows from the fact that there is a simple bound on the size of UCQs equivalent to Q and using at least one view.[1] In turn, this is a consequence of the result that under bag semantics UCQs are equivalent iff they are isomorphic. (see [13] where this result is discussed).[2]

For set semantics, the problem is more complicated, since even CQs can be arbitrarily extended with redundant atoms while preserving equivalence. Nonetheless, the results of [24] imply that there is a simple bound on the size of *locally minimal* UCQs equivalent to Q and using at least one view.

Abstracting the salient features of these approaches, we find that they imply the decidability of query equivalence under $\mathcal{V}$ (see Appendix).

Naturally, it would be desirable to extend query reformulation techniques to the full relational algebra ($\mathcal{RA}$) by adding *difference*. This would yield a variety of benefits:

- *Optimization using materialized views* could be done over a broader space of plans (even if the original query and view were just CQs/UCQs!). Rewritings could *subtract* one view from a larger view in

[1] This is the foundation of optimization of queries with views under bag semantics [4].
[2] Under bag semantics, *containment* of UCQs is undecidable [21]. For CQs, equivalence is the same as isomorphism [5] but the status of containment is still open.

order to return a query answer.

- *View adaptation* [16], 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.
- *Incremental view maintenance* [17] could be seen as a reformulation using views, since insertions and deletions could be treated as unions and differences.

Unfortunately, introducing the difference operator destroys the decidablity of query equivalence under views. Indeed, even without views the equivalence of relational algebra ($\mathcal{RA}$) queries is undecidable, for both set and bag semantics.[3] Since the decidability of equivalence is a necessary condition, we cannot hope for the approaches to reformulation under bag or set semantics to extend to the entire $\mathcal{RA}$.

A natural question to ask is whether there is a slightly less expressive class of queries — 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 [14, 15]. In addition, difference has an obvious, natural definition on $\mathbb{Z}$-relations.

$\mathbb{Z}$-relations are a natural representation for the *updates* to source relations (collections of tuple insertions and deletions, a.k.a. *deltas*) which must be propagated in incremental view maintenance applications. Indeed, both data and updates can be uniformly represented using $\mathbb{Z}$-relations, and "application" of a delta to a relation corresponds to simply computing a union. We discuss this further in Section 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. 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. Although membership in this class of queries is necessarily undecidable, there are many useful cases with simple sufficient conditions for membership, in the three classes of applications outlined above.

[3]The latter follows, e.g., from the undecidability of bag-containment of unions of conjunctive queries (UCQs) [21], since for UCQs Q, Q' we have Q is contained in Q' iff $Q - Q'$ is equivalent to the empty answer query.

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 $\mathcal{RA}^+$ queries. 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 for 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 natural, instance-independent notion of "inherent 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.
- Next we examine when we can use the $\mathbb{Z}$-semantics reformulation strategy to obtain results that work for the bag semantics. We show that the reformulation procedure is closed for queries/views in this class. We also give simple membership conditions.
- 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 Section 2. We define the semantics of $\mathcal{RA}$ on $\mathbb{Z}$-relations and establish the decidability of $\mathbb{Z}$-equivalence of $\mathcal{RA}$ queries in Section 3. We introduce DUCQs and the rewrite system for queries using views in Section 4. We present reformulation algorithms and strategies in Section 5. We discuss reformulation for bag semantics/set semantics via $\mathbb{Z}$-semantics in Section 6. We extend our $\mathbb{Z}$-equivalence results to $\mathcal{RA}$ with built-in predicates in Section 7. We discuss related work in Section 8 and conclude in Section 9.

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*, 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.

Optimizing queries using views [4, 24]. 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:

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

and a query Q for paths of length 3:

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

To answer Q using V, we might compute paths of length 2, then subtract those paths from V under bag semantics. If our end goal is set semantics, we would add a duplicate elimination step at the end.

View adaptation [16]. 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:

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

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

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

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 [17]. 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:

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

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. [17] 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*. A set of delta rules for V correspond to the UCQ:

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

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.:

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

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 the materialized view 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.

A unified treatment of these three applications requires 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*).

A *bag relation* over attributes U is a mapping $R : U\text{-}\mathsf{Tup} \to \mathbb{N}$ 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 $R : U\text{-}\mathsf{Tup} \to \mathbb{Z}$ 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* (or $\mathbb{B}$-*instance*) is a mapping from predicate symbols to $\mathbb{B}$-relations.

We define the positive relational algebra ($\mathcal{RA}^+$) on bag instances, $\mathbb{Z}$-instances, and set instances according to the semiring-annotated semantics used in our previous [15, 14] (repeated in the Appendix). 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 Section 7.)

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: if $R_1 : U\text{-}\mathsf{Tup} \to \mathbb{Z}$ and $R_2 : U\text{-}\mathsf{Tup} \to \mathbb{Z}$ then $R_1 - R_2 : U\text{-}\mathsf{Tup} \to \mathbb{Z}$ is defined by

$$(R_1 - R_2)(t) \stackrel{\text{def}}{=} R_1(t) - R_2(t)$$

For bag semantics, the subtraction in the definition above is replaced by *proper subtraction* (negative numbers are truncated to 0). For set semantics, subtraction corresponds to set difference.

Every bag instance is also a $\mathbb{Z}$-instance, and relational queries on bag instances can be evaluated under bag semantics or $\mathbb{Z}$-semantics. To disambiguate we use the notation $\mathsf{eval}_K(Q, I)$ 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. "$\Rightarrow$" 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. "$\Leftarrow$" follows from the fact that bag equivalent positive queries, when transformed into UCQs, are isomorphic (see Section 7.5 in [13]). □

3.1 Normal Form and Decidability

Equivalence of relational queries under set semantics has long been known to be undecidable [26]. We have also seen (cf. footnote in Section 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 is that $\mathbb{Z}$-equivalence of relational queries is actually *decidable*.

$$\begin{aligned}
(A - B) - C &\equiv_{\mathbb{Z}} A - (B \cup C)\\
A \bowtie (B - C) &\equiv_{\mathbb{Z}} (A \bowtie B) - (A \bowtie C)\\
A - (B - C) &\equiv_{\mathbb{Z}} (A \cup C) - B\\
(A - B) \bowtie C &\equiv_{\mathbb{Z}} (A \bowtie C) - (B \bowtie C)\\
A \cup (B - C) &\equiv_{\mathbb{Z}} (A \cup B) - C\\
\sigma_P(A - B) &\equiv_{\mathbb{Z}} \sigma_P(A) - \sigma_P(B)\\
(A - B) \cup C &\equiv_{\mathbb{Z}} (A \cup C) - B\\
\pi_X(A - B) &\equiv_{\mathbb{Z}} \pi_X(A) - \pi_X(B)\\
\rho_\beta(A - B) &\equiv_{\mathbb{Z}} \rho_\beta(A) - \rho_\beta(B)
\end{aligned}$$

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

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:

THEOREM 3.2 (NORMALIZATION). *For any $Q \in \mathcal{RA}$ we can find $A, B \in \mathcal{RA}^+$ such that $Q \equiv_{\mathbb{Z}} A - B$.*

PROOF. Straightforward induction on Q, using the algebraic identities in Figure 1. □

Note that several of these identities fail for both set and bag semantics and indeed Theorem 3.2 fails for set or bag semantics.

The procedure in the proof of Theorem 3.2 can be easily determinized and is clearly effective. Thus, we have a function that computes for any $Q \in \mathcal{RA}$ a *difference normal form* denoted by $\mathsf{diffNF}(Q)$ which has the form $A - B$ with $A, B \in \mathcal{RA}^+$ and such that $Q \equiv_{\mathbb{Z}} \mathsf{diffNF}(Q)$. Note that in general $\mathsf{diffNF}(Q)$ may be of size exponential in the size of Q.

COROLLARY 3.3. *$\mathbb{Z}$-equivalence of $\mathcal{RA}$ queries is decidable.*

PROOF. Let $Q, Q' \in \mathcal{RA}$. By Theorem 3.2, $Q \equiv_{\mathbb{Z}} Q'$ 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 [13]. □

Corollary 3.3 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 4.1 which shows that the problem is at least *GI*-hard.[4]

[4] *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*.

4. DUCQS AND REFORMULATION

Our reformulation algorithm does not work directly on queries in $\mathcal{RA}$. Rather, it is more convenient here to adopt a Datalog-style syntax. We will define *differences of unions of conjunctive queries* (DUCQs) for this purpose, for example:

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

The $\tilde{}$ marks a "negated" CQ, so this example in algebraic form corresponds (roughly) to $Q = (\pi(R \bowtie S) \cup (R \bowtie R \bowtie R)) - (R \bowtie T)$. DUCQs are related to the *elementary differences* of [26]. Under $\mathbb{Z}$-semantics, any $\mathcal{RA}$ query can be written equivalently as a DUCQ; this follows from Theorem 3.2 and the fact that any $\mathcal{RA}^+$ query can be rewritten as a union of projections of selections of joins, i.e., as a UCQ. This justifies our use of DUCQs as an alternative representation of relational algebra queries. The semantics of DUCQs on bag relations/$\mathbb{Z}$-relations/set relations can be given formally by translation to $\mathcal{RA}$.

In this section, we first consider the complexity of reformulation using DUCQs. Then we discuss how we can explore the space of rewritings using a term rewrite system, we consider how to limit our search to *diff-irredundant rewritings*, and we develop a cost model and a procedure for limiting the search space further.

4.1 Complexity Results

Using reasoning from the proof of Corollary 3.3 along with the fact that checking bag equivalence of UCQs is *GI*-complete [13], we can fully characterize the complexity of checking $\mathbb{Z}$-equivalence of DUCQs:

PROPOSITION 4.1. *For DUCQs Q, Q' checking $Q \equiv_{\mathbb{Z}} Q'$ is GI-complete.*

We next consider several complexity questions related to *answering queries using views*. A basic question posed in [24] is the following: given a query Q and a view V, can Q be rewritten to use V? For conjunctive queries under set semantics, it is shown in [24] the problem is *NP*-complete (and equivalent to the problem of checking whether Q is contained in V when Q and V are viewed as Boolean queries). Under bag semantics and $\mathbb{Z}$-semantics, the complexity remains *NP*-complete, for CQs or UCQs:

THEOREM 4.2. *Given a query $Q \in CQ$ and view $V \in CQ$, determining whether there exists a CQ rewriting of Q using V under bag semantics (or $\mathbb{Z}$-semantics) is NP-complete. It remains NP-complete when the query, view, and rewriting are UCQs.*

See proof in Appendix C.

For DUCQs, the same question has a trivial answer: a DUCQ Q can always be rewritten as the equivalent $(V \cup \tilde{}V) \cup Q$ (equivalent under bag semantics, $\mathbb{Z}$-semantics, and even set semantics).

Another question studied in [24] concerns *complete* rewritings. Given a set of views $\mathcal{V}$, Q has a *complete rewriting* if there is a rewriting of Q using only view predicates (and no source predicates).

THEOREM 4.3. *Given a query $Q \in CQ$ and set of views $\mathcal{V} \subseteq CQ$, determining whether there exists a CQ rewriting of Q using only predicates in $\mathcal{V}$ under bag semantics (or $\mathbb{Z}$-semantics) is NP-complete. It remains NP-complete when the query, view, and rewriting are UCQs.*

For DUCQs, we leave open the decidability of the above problem.

4.2 Term Rewriting System

The goal of a query reformulation algorithm is to *enumerate* equivalent rewritings of a query and find the minimum-cost rewriting (according to some cost model). If equivalence is decidable (for the query language of interest) a naive approach is to enumerate all queries, selecting only those equivalent to the original query. A more practical approach is to explore the space of rewritings by repeatedly applying a set of *transformation rules*. The main benefit is that only equivalent queries are ever considered.

We are also interested in checking equivalence and performing query reformulation *with respect to a set of a views* $\mathcal{V}$ over the source relations. In this case queries and their reformulations are allowed to use any combination of source predicates and view predicates. For $K \in \{\mathbb{B}, \mathbb{N}, \mathbb{Z}\}$ and $Q, Q \in$ DUCQ (possibly using views in $\mathcal{V} \subseteq$ DUCQ) we say that Q is *K-equivalent to Q' with respect to* $\mathcal{V}$, denoted $Q \equiv_K^{\mathcal{V}} Q'$, if Q and Q' agree on all instances of the source relations and derived views.

For these dual purposes (checking equivalence and enumerating rewritings), we introduce here a *term rewriting system* [23] 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 $Q \rightarrow_V Q'$, if Q' can be obtained from CQ Q by *unfolding* (in the standard way) a single occurrence of the view predicate V in Q. We extend $\rightarrow_V$ to work with UCQ views by unfolding repeatedly (once for each CQ in the view) and producing a UCQ as output (with the same number of rules as the view). For example, if V is the UCQ view:

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

and Q is the CQ:

$$Q(x,y) \quad :- \quad V(x,z), V(z,y)$$

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

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

Finally we extend $\rightarrow_V$ to work with DUCQs by propagating the $\tilde{}$ (e.g., if the second CQ in V above were marked $\tilde{}$, then the second CQ in Q' above would also be marked $\tilde{}$).

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

$$\frac{P, Q, R \in \text{DUCQ} \qquad V \in \mathcal{V} \qquad P \rightarrow_V Q}{P \cup R \rightarrow Q \cup R} \text{ (UNFOLD)}$$

$$\frac{A, B \in \text{CQ} \qquad Q \in \text{DUCQ} \qquad A \cong B}{Q \cup A \cup (\tilde{}B) \rightarrow Q} \text{ (CANCEL)}$$

Next we establish some basic properties of our rewrite system. We denote the transitive reflexive closure of $\rightarrow$ by $\xrightarrow{*}$. We say Q is a *normal form* if no rewrite rule applies. A *reduction sequence* $Q_1 \rightarrow \cdots \rightarrow Q_n$ is *terminating* if Q_n is a normal form.

PROPOSITION 4.4. *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 $Q \xrightarrow{*} Q_1$ and $Q \xrightarrow{*} Q_2$ then there exist $Q_3, Q_3' \in DUCQ$ such that $Q_1 \xrightarrow{*} Q_3$ and $Q_2 \xrightarrow{*} Q_3'$ and $Q_3 \cong Q_3'$.*
2. **(termination)** *Every reduction sequence $Q_1 \rightarrow Q_2 \rightarrow \cdots$ eventually must terminate.*

PROPOSITION 4.5. *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 $Q_1 \xrightarrow{*} Q_2$ 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 $Q_1 \xrightarrow{*} Q_1'$ and $Q_2 \xrightarrow{*} Q_2'$ and $Q_1' \cong Q_2'$.*

COROLLARY 4.6. *$\mathbb{Z}$-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 4.6.

In Section 5 we develop an enumeration algorithm for $\mathbb{Z}$-equivalent rewritings, 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.

$$\frac{P, Q, R \in \text{DUCQ} \qquad V \in \mathcal{V} \qquad P \rightarrow_V Q}{Q \cup R \rightarrow P \cup R} \text{ (FOLD)}$$

$$\frac{A, B \in \text{CQ} \qquad Q \in \text{DUCQ} \qquad A \cong B}{Q \rightarrow Q \cup A \cup (\tilde{}B)} \text{ (AUGMENT)}$$

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, and write $Q \leftharpoonup Q'$ if Q' can be obtained from Q by an application of AUGMENT-FOLD-CANCEL. We denote the reflexive transitive closure of $\leftharpoonup$ by $\overset{*}{\leftharpoonup}$. Note that $Q \overset{*}{\leftharpoonup} Q'$ implies $Q \overset{*}{\leftarrow} Q'$ but the converse does not hold in general.

PROPOSITION 4.7. *If $Q \in DUCQ$ then checking whether there exists $Q' \in DUCQ$ s.t. $Q \rightarrow Q'$ (resp. $Q \leftharpoonup Q'$) is GI-complete (resp. NP-complete).*

4.3 Diff-Irredundant Rewritings

For UCQs (and CQs) under set semantics, the space of rewritings is bounded because we need only search for queries that satisfy a particular minimality property. The space of such minimal queries is finite. Naturally, we would like to find a similar notion of minimality for DUCQs, such that any "non-minimal" rewriting is always less desirable than a minimal one. In this subsection, we define such a notion based on removing redundant computation, and we consider whether the space of non-redundant rewritings is finite.

We saw in the previous section that AUGMENT (which we denote $\leftarrow$) can be used to "grow" a DUCQ arbitrarily by adding more and more CQs: for example, $Q \leftarrow Q \cup A \cup \tilde{}A \leftarrow Q \cup A \cup B \cup \tilde{}A \cup \tilde{}B \leftarrow \cdots$. Here, we introduce a notion of *diff-redundancy* which targets the kind of redundancy in these rewritings:

DEFINITION 4.8. *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 $A' \equiv^{\mathcal{V}}_{\mathbb{Z}} B'$. In this case the pair of terms A', B' is said to be* diff-redundant.

In the example above, $Q \cup A \cup \tilde{}A$ and $Q \cup A \cup B \cup \tilde{}A \cup \tilde{}B$ 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 4.9. *If Q, Q' 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 4.10. *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:

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

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 [24], 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 Section 2 we had a view maintenance example, where the original view was defined as:

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

and our updated view could be defined as:

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

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:

$$\begin{aligned} V'(x, y) &\quad \text{:-} \quad V(x, y) \\ V'(x, y) &\quad \text{:-} \quad R^{\Delta}(x, z), R(z, y) \\ V'(x, y) &\quad \text{:-} \quad R(x, z), R^{\Delta}(z, y) \end{aligned}$$

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* [27] predicts query performance given known properties (such as cardinalities) of the input relations. Here we seek an abstract cost model that captures the essence of the detailed models implemented in real optimizers. 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:

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

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,[5] 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 any CQ has cost at least 1, and any DUCQ Q has cost at least $n = |Q|$ (where $|Q|$ is the number of CQs in the DUCQ). This in turn leads to a bound on the size of a minimum-cost rewriting (for a given instance):

PROPOSITION 4.11. *For $Q \in DUCQ$, let $\mathcal{V}$ be a set of views, and let $k = \mathsf{cost}(Q)$ where cost is defined as above. Then for any $Q' \in DUCQ$, if $\mathsf{cost}(Q') \leq \mathsf{cost}(Q)$, then $|Q'| \leq k$.*

This bound on the size of a minimum-cost rewriting can be used to establish a bound on the *length* of a reduction sequence which produces the rewriting:

PROPOSITION 4.12. *Let $Q \in DUCQ$, let $\mathcal{V}$ be a set of views, let Q' be the result of rewriting Q to normal form, let Q'' be a rewriting of Q of minimum cost, let $k_1 = \mathsf{cost}(Q)$, let k_2 be the maximum number of atoms in the body of a CQ in Q and $\mathcal{V}$, and let k_3 be the maximum number of CQs in a DUCQ in Q and $\mathcal{V}$. Then there exists a reduction sequence $Q' \leftarrow Q_1 \leftarrow \cdots \leftarrow Q_n \leftarrow Q''$ from Q'' to Q' with $n \leq k_1 k_2 k_3$.*

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

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).

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 Section 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

From Section 4.2 we have a set of rewrite rules, namely UNFOLD, CANCEL, and AUGMENT-FOLD-CANCEL. Algorithm 1 shows how these 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 4.12. (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.

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 [27] only does cost-based optimization of *joins*, instead relying on heuristics for applying unions and differences. Starburst [18] can rewrite queries with unions and differences, but only at its heuristics-based *query rewrite* stage. Starburst only has limited

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
```

facilities for cost-based comparison of alternative rewritings.

Fortunately, the Volcano [12] 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 Section 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 semantics. Thus 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, $\mathsf{eval}_{\mathbb{N}}(Q, I) = \mathsf{eval}_{\mathbb{Z}}(Q, I)$.*

Right away we see that:

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

Then (see proofs in Appendix C):

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

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

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 Section 5 examines only queries in $\overline{\mathcal{RA}}$ which are $\equiv_{\mathbb{N}}$ under the views to the original DUCQ.

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

PROPOSITION 6.6. *If Q is a DUCQ in $\overline{\mathcal{RA}}$ and Q' is a DUCQ obtained from Q either by a step of augmentation or by a step of folding with a view from $\mathcal{V}$ then $Q' \in \overline{\mathcal{RA}}$ and Q' is $\equiv_{\mathbb{N}}$ to Q under $\mathcal{V}$.*

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 Section 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 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 $Q \in \widehat{\mathcal{RA}}$ 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 3.2. 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 $Q \notin \widehat{\mathcal{RA}}$. $\square$

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 $Q \equiv_{\mathbb{N}} Q'$.*

See proof in Appendix C.

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 [17], 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).

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 *wrt 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 wrt 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 $Q(I)$ by viewing I as a $\mathbb{Z}$-instance, computing $Q(I)$ under $\mathbb{Z}$ semantics 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 [9, 8], 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:

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

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. $\mathit{lin}(Q) \cong \mathit{lin}(Q')$

PROOF. (2) $\Rightarrow$ (1) because every bag instance is a $\mathbb{Z}$-instance. (1) $\Rightarrow$ (3) is stated in [9] for bag-set semantics but it also holds for bag semantics (see Section 7.5 in [13]). (3) $\Rightarrow$ (1) follows from the observation that $Q \equiv_{\mathbb{Z}} \mathsf{lin}(Q)$. $\square$

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 w.r.t. 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. [24] 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 [26], where the undecidability of set-equivalence of $\mathcal{RA}$ queries was also established.

The papers by Ioannidis and Ramakrishnan [21] 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 [9, 6] and also from results on provenance-annotated semantics (see the discussion in [13]). The decidability of bag-containment of CQs remains open. Recent progress was made on the problem by Jayram, Kolaitis and Vee [22] 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 [25] (see also [20]). 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 [16], 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 [17]. That paper did not consider query equivalence for deltas or make a general study of query reformulation.

9. CONCLUSIONS

In this paper we investigated the problem of exact query reformulation using views, when queries and views are given with the full $\mathcal{RA}$ including difference. We introduced $\mathbb{Z}$-relations, we showed that under $\mathbb{Z}$-semantics, every $\mathcal{RA}$ query may be expressed as a difference of two $\mathcal{RA}^+$ queries, and we saw that this leads to the decidability of $\mathbb{Z}$-equivalence of $\mathcal{RA}$ queries. (In contrast, equivalence of $\mathcal{RA}$ queries is undecidable under set or bag semantics). We presented a query reformulation algorithm based on a term rewrite system and saw that it is sound and complete for $\mathcal{RA}$ queries under $\mathbb{Z}$-semantics. We also saw that query reformulation on $\mathbb{Z}$-relations allows us to unify in one framework several interesting applications, such as optimizing queries using materialized views, view adaptation, and incremental view maintenance. We studied related complexity questions, and gave conditions under which our techniques our results can be used to find reformulations under bag semantics and set semantics. Finally, we showed that $\mathbb{Z}$-equivalence of $\mathcal{RA}$ queries extended with built-in predicates, and bag-equivalence of $\mathcal{RA}^+$ queries with built-in predicates, are also decidable.

As future work, we hope to develop a practical implementation of our reformulation algorithm and evaluate experimentally its effectiveness in view maintenance, adaptation, and optimization.

10. REFERENCES

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

[2] F. Afrati and V. Pavlaki. Rewriting queries using views with negation. *AI Communications*, 19:229–237, 2006.

[3] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, 1977.

[4] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing queries with materialized views. In *ICDE*, 1995.

[5] S. Chaudhuri and M. Y. Vardi. Optimization of *real* conjunctive queries. In *PODS*, pages 59–70, 1993.

[6] S. Cohen. Containment of aggregate queries. *SIGMOD Record*, 34(1):77–85, 2005.

[7] S. Cohen, W. Nutt, and Y. Sagiv. Rewriting queries with arbitrary aggregation functions using views. *ACM Trans. Database Syst.*, 31(2):672–715, 2006.

[8] S. Cohen, W. Nutt, and Y. Sagiv. Deciding equivalences among conjunctive aggregate queries. *J. ACM*, 54(2), 2007.

[9] S. Cohen, W. Nutt, and A. Serebrenik. Rewriting aggregate queries using views. In *PODS*, 1999.

[10] S. Cohen, Y. Sagiv, and W. Nutt. Equivalences among aggregate queries with negation. *ACM TOCL*, 6(2):328–360, April 2005.

[11] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. McGraw-Hill / MIT Press, 1990.

[12] G. Graefe and W. J. McKenna. The Volcano optimizer generator: Extensibility and efficient search. In *ICDE*, 1993.

[13] T. J. Green. Containment of conjunctive queries on annotated relations. In *ICDT*, 2009.

[14] T. J. Green, G. Karvounarakis, Z. G. Ives, and V. Tannen. Update exchange with mappings and provenance. In *VLDB*, 2007. Amended version available as Univ. of Pennsylvania report MS-CIS-07-26.

[15] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, 2007.

[16] A. Gupta, I. S. Mumick, J. Rao, and K. A. Ross. Adapting materialized views after redefinitions: Techniques and a performance study. *Information Systems*, 26(5):323–362, 2001.

[17] A. Gupta, I. S. Mumick, and V. S. Subrahmanian. Maintaining views incrementally. In *SIGMOD*, 1993.

[18] L. M. Haas, J. C. Freytag, G. M. Lohman, and H. Pirahesh. Extensible query processing in Starburst. In *SIGMOD*, 1989.

[19] A. Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4), 2001.

[20] P. Hell and J. Nešetřil. *Graphs and Homomorphisms*. Oxford University Press, 2004.

[21] Y. E. Ioannidis and R. Ramakrishnan. Containment of conjunctive queries: Beyond relations as sets. *TODS*, 20(3):288–324, 1995.

[22] T. S. Jayram, P. G. Kolaitis, and E. Vee. The containment problem for *real* conjunctive queries with inequalities. In *PODS*, 2006.

[23] J. W. Klop. *Handbook of Logic in Computer Science*, volume 2, chapter 1. Oxford University Press, 1992.

[24] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *PODS*, 1995.

[25] L. Lovász. Operations with structures. *Acta Mathematica Hungarica*, 18(3–4):321–328, 1967.

[26] Y. Sagiv and M. Yannakakis. Equivalences among relational expressions with the union and difference operators. *J. ACM*, 27(4):633–655, 1980.

[27] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *SIGMOD*, 1979.

APPENDIX

A. POSITIVE RELATIONAL ALGEBRA ON $\mathbb{Z}$-RELATIONS

The operations of the **positive algebra** on $\mathbb{Z}$-relations are defined as follows:

empty relation For any set of attributes U, there is $\emptyset : U\text{-}\mathsf{Tup} \to \mathbb{Z}$ such that $\emptyset(t) = 0$.

union If $R_1, R_2 : U\text{-}\mathsf{Tup} \to \mathbb{Z}$ then $R_1 \cup R_2 : U\text{-}\mathsf{Tup} \to \mathbb{Z}$ is defined by

$$(R_1 \cup R_2)(t) \stackrel{\text{def}}{=} R_1(t) + R_2(t)$$

projection If $R : U\text{-}\mathsf{Tup} \to \mathbb{Z}$ and $V \subseteq U$ then $\pi_V R : V\text{-}\mathsf{Tup} \to \mathbb{Z}$ is defined by

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

(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 since R has finite support)

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

$$(\sigma_{\mathbf{P}} R)(t) \stackrel{\text{def}}{=} R(t) \cdot \mathbf{P}(t)$$

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 $R_i : U_i\text{-}\mathsf{Tup} \to \mathbb{Z}$ $i = 1, 2$ then $R_1 \bowtie R_2$ is the $\mathbb{Z}$-relation over $U_1 \cup U_2$ defined by

$$(R_1 \bowtie R_2)(t) \stackrel{\text{def}}{=} R_1(t_1) \cdot R_2(t_2)$$

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 $R : U\text{-}\mathsf{Tup} \to \mathbb{Z}$ and $\beta : U \to U'$ is a bijection then $\rho_\beta R$ is a $\mathbb{Z}$-relation over U' defined by

$$(\rho_\beta R)(t) \stackrel{\text{def}}{=} R(t \circ \beta)$$

B. COMPUTABILITY ASPECTS OF REFORMULATION

Let us summarize the common abstract elements of the approaches to query reformulation using views for CQs and UCQs, for bag and set semantics. Let Σ be the relational schema and let us denote by $\Sigma_{\mathcal{V}}$ the schema consisting of the view names (assumed disjunct from Σ). In all these approaches to reformulation we have:

- A total recursive function β that associates to each Σ-query a natural number.
- For each Σ-query Q a total recursive function μ_Q (abstracting local minimization) that associates to each $\Sigma \cup \Sigma\mathcal{V}$-query Q' an equivalent $\Sigma \cup \Sigma_{\mathcal{V}}$-query such that if $Q' \equiv_{\mathbb{Z}}^{\mathcal{V}} Q$ then $\mu_Q(Q') \equiv_{\mathbb{Z}}^{\mathcal{V}} Q$ and the size of $\mu_Q(Q')$ is bounded by $\beta(Q)$.
- The total function that associates to each Σ-query Q the (finite) set of $\Sigma \cup \Sigma_{\mathcal{V}}$-query that are equivalent to Q under $\mathcal{V}$ and whose size is bounded by $\beta(Q)$ is recursive.

It is easy to see that these imply the decidability of query equivalence under $\mathcal{V}$.

C. PROOFS

PROOF. (of Theorem 4.2) (Sketch) Membership in *NP* in both cases is easy to see. *NP*-hardness when Q and V are CQs (and hence also for the case where Q and V are UCQs) is established by a straightforward reduction from the *subgraph isomorphism* problem: given 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.) □

PROOF. (of Lemma 6.3) "⇒": suppose $A - B \in \overline{\mathcal{RA}}$ and consider an arbitrary bag instance I and tuple t. Since $\mathsf{eval}_{\mathbb{Z}}(A-B, I)(t) = \mathsf{eval}_{\mathbb{N}}(A-B, I)(t) \geq 0$, and by definition, $\mathsf{eval}_{\mathbb{Z}}(A-B, I)(t) = \mathsf{eval}_{\mathbb{Z}}(A, I)(t) - \mathsf{eval}_{\mathbb{Z}}(B, I)(t)$, it follows that $\mathsf{eval}_A(\mathbb{Z}, I)(t) \geq \mathsf{eval}_{\mathbb{Z}}(B, I)(t)$. Since A and B are positive queries, by Lemma 3.1, $\mathsf{eval}_{\mathbb{Z}}(A, I) = \mathsf{eval}_{\mathbb{N}}(A, I)$ and $\mathsf{eval}_{\mathbb{Z}}(B, I) = \mathsf{eval}_{\mathbb{N}}(B, I)$ and therefore $\mathsf{eval}_{\mathbb{N}}(A, I)(t) \geq \mathsf{eval}_{\mathbb{N}}(B, I)(t)$. Since I and t were chosen arbitrarily, it follows that $B \sqsubseteq_{\mathbb{N}} A$.

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

PROOF. (of Proposition 6.4) 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 $\mathsf{eval}_{\mathbb{N}}(Q, I)(t) = \mathsf{eval}_{\mathbb{Z}}(Q, I)(t) = \mathsf{eval}_{\mathbb{Z}}(A - B, I)(t) = \mathsf{eval}_{\mathbb{Z}}(A, I)(t) - \mathsf{eval}_{\mathbb{Z}}(B, I)(t) \geq 0$. It follows that $\mathsf{eval}_{\mathbb{Z}}(A, I)(t) \geq \mathsf{eval}_{\mathbb{Z}}(B, I)(t)$ and therefore (using Lemma 3.1) that $\mathsf{eval}_{\mathbb{Z}}(A-B, I)(t) = \mathsf{eval}_{\mathbb{N}}(A - B, I)(t)$. 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 $A - B \in \overline{\mathcal{RA}}$. □

PROOF. (of Theorem 6.11) 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 3.2, 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 $\mathsf{eval}_{\mathbb{N}}(A, I)(t) < \mathsf{eval}_{\mathbb{N}}(B, I)$. Then $\mathsf{eval}_{\mathbb{Z}}(A - B, I)(t) < 0$, i.e., $\mathsf{eval}_{\mathbb{Z}}(A - B, I) = \mathsf{eval}_{\mathbb{Z}}(Q, I)$ is not even a bag-instance. However, this is a contradiction, because by assumption (since $Q \in \overline{\mathcal{RA}}$), we must have $\mathsf{eval}_{\mathbb{Z}}(Q, I) = \mathsf{eval}_{\mathbb{N}}(Q, I)$. Finally, we argue that $Q \equiv_{\mathbb{N}} A - B$. To see this, fix an arbitrary bag instance I. We want to show that $\mathsf{eval}_{\mathbb{N}}(Q, I) = \mathsf{eval}_{\mathbb{N}}(A - B, I)$. Since $Q \in \overline{\mathcal{RA}}$, we have $\mathsf{eval}_{\mathbb{N}}(Q, I) = \mathsf{eval}_{\mathbb{Z}}(Q, I)$. Since $Q \equiv_{\mathbb{Z}} A - B$ we have $\mathsf{eval}_{\mathbb{Z}}(Q, I) = \mathsf{eval}_{\mathbb{Z}}(A - B, I)$. Finally, since $A-B \in \widehat{\mathcal{RA}}$, by Theorem 6.10 we have $\mathsf{eval}_{\mathbb{Z}}(A-B, I) = \mathsf{eval}_{\mathbb{N}}(A - B, I)$. This completes the proof. □