An improved algorithm for the incremental recomputation of active relational expressions

Timothy Griffin, Leonid Libkin, and Howard Trickey

Abstract— **In [1] Qian and Wiederhold presented an algorithm for the incremental recomputation of relational algebra expressions that was claimed to preserve a certain *minimality* condition. This condition guarantees that the incremental change sets do not contain any unnecessary tuples and so redundant computations are not performed. We show that in fact their algorithm violates this condition. We present an improved algorithm that does preserve this notion of minimality.**

Keywords— **Relational algebra, incremental recomputation, change propagation, view maintenance.**

I. Introduction

Let R, R_1, R_2, ... denote names of relations in a database scheme. Let p range over quantifier-free predicates, and A range over sets of attribute names. Relational algebra expressions are generated by the grammar

S	$::=$	R	base relation
	$\mid$	$\sigma_p(S)$	selection
	$\mid$	$\Pi_A(S)$	projection
	$\mid$	$S \times S$	cartesian product
	$\mid$	$S \cup S$	union
	$\mid$	$S \cap S$	intersection
	$\mid$	$S - S$	difference
	$\mid$	$S \bowtie S$	natural join

The symbols S, T and Q will be used to denote arbitrary relational expressions.

If s is a database state, that is, a partial map from relation names to finite sets of tuples, and T is a relational expression such that s is defined on all relation names mentioned in T, then $s(T)$ denotes the set resulting from evaluating T in the state s. (Note that s is a function, so we consider evaluating T in s as the result of applying s to T.) The notation $T =_r S$ means that for all database states s, if s is defined on all relation names mentioned in S and T, then $s(T) = s(S)$. The notation $T \subseteq_r S$ means that for all database states s, if s is defined on all relation names mentioned in S and T, then $s(T) \subseteq s(S)$.

Abstract transactions, viewed as functions from states to states, are assumed to be of the form

$$t = \begin{cases} R_1 & \leftarrow \quad (R_1 - \nabla R_1) \cup \triangle R_1, \\ & \cdots \\ R_n & \leftarrow \quad (R_n - \nabla R_n) \cup \triangle R_n \end{cases}$$

The expressions ∇R_i and $\triangle R_i$ represent the sets deleted from and inserted into base relation R_i. More formally, when transaction t is executed in state s, then the value of R_i in state $t(s)$ becomes $s((R_i - \nabla R_i) \cup \triangle R_i)$.

The expression T is a *pre-expression* of S w.r.t. t if for every database state s we have $s(T) = t(s)(S)$. In other words, we can evaluate the pre-expression T *before* we execute t in order to determine the value that S will have *afterwards.* It is easy to see that

$$\text{pre}(t, S) \stackrel{\text{def}}{=} S\begin{pmatrix} (R_1 - \nabla R_1) \cup \triangle R_1, \\ \cdots \\ (R_n - \nabla R_n) \cup \triangle R_n \end{pmatrix} \qquad (1)$$

is a pre-expression of S w.r.t. t.

Example. Let $Q \stackrel{\text{def}}{=} R_1 \cup R_2$ and $t \stackrel{\text{def}}{=} \{R_1 \leftarrow R_1 - \nabla R_1, R_2 \leftarrow R_2 \cup \triangle R_2.\}$. Then

$$\text{pre}(t, Q) \stackrel{\text{def}}{=} (R_1 - \nabla R_1) \cup (R_2 \cup \triangle R_2)$$

can be evaluated before t is committed in order to determine the value that Q will have afterwards.

Problem Statement. Suppose $S(R_1, \cdots, R_n)$ is a relational expression and t is a transaction. We would like to determine how t's changes to the base relations propagate to changes in the value of S. In particular, we seek to construct relational expressions $\triangle S$ and ∇S, called a *solution* for $\text{pre}(t, S)$, such that

$$\text{pre}(t, S) =_r (S - \nabla S) \cup \triangle S. \qquad (2)$$

Any algorithm for producing such a solution will be called a *change propagation algorithm.*

Note that the expressions ∇S and $\triangle S$ are to be evaluated before t is executed and committed. These solutions can be used in many applications involving the maintenance of derived data. For example, in the case of view maintenance, representation (2) allows us to recompute the value of S in the new state from its value in the old state and the values of ∇S and $\triangle S$. In general, we can expect this to be computationally less expensive than recomputing S in the new state or computing (1) in the pre-commit state. For integrity maintenance it allows us to check data integrity *before* a transaction is committed, thus allowing for the transaction to be aborted without the expense of a roll-back operation.

There are several properties that we could require of a "good" solution. It is usually required that no unnecessary computations be involved in the evaluation of ∇S and $\triangle S$. Formally, a solution will be called *minimal* if

(a) $\nabla S \subseteq_r S$: All deleted tuples are in S.

AT&T Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974, USA.

1	$\sigma_p(S - \bigtriangledown S) =_r \sigma_p(S) - \sigma_p(\bigtriangledown S)$
2	$\sigma_p(S \cup \triangle S) =_r \sigma_p(S) \cup \sigma_p(\triangle S)$
3	$\Pi_A(S - \bigtriangledown S) =_r \Pi_A(S) - (\Pi_A(S) - \Pi_A(S - \bigtriangledown S))$
4	$\Pi_A(S \cup \triangle S) =_r \Pi_A(S) \cup (\Pi_A(\triangle S) - \Pi_A(S))$
5	$(S - \bigtriangledown S) \times T =_r (S \times T) - (\bigtriangledown S \times T)$
6	$(S \cup \triangle S) \times T =_r (S \times T) \cup (\triangle S \times T)$
7	$(S - \bigtriangledown S) \cup T =_r (S \cup T) - (\bigtriangledown S - T)$
8	$(S \cup \triangle S) \cup T =_r (S \cup T) \cup (\triangle S - T)$
9	$(S - \bigtriangledown S) - T =_r (S - T) - (\bigtriangledown S - T)$
10	$S - (T - \bigtriangledown T) =_r (S - T) \cup (S \cap \bigtriangledown T)$
11	$(S \cup \triangle S) - T =_r (S - T) \cup (\triangle S - T)$
12	$S - (T \cup \triangle T) =_r (S - T) - (S \cap \triangle T)$
13	$(S - \bigtriangledown S) \cap T =_r (S \cap T) - (\bigtriangledown S \cap T)$
14	$(S \cup \triangle S) \cap T =_r (S \cap T) \cup (\triangle S \cap T)$
15	$(S - \bigtriangledown S) \bowtie T =_r (S \bowtie T) - (\bigtriangledown S \bowtie T)$
16	$(S \cup \triangle S) \bowtie T =_r (S \bowtie T) \cup (\triangle S \bowtie T)$

Fig. 1 Relational equations used in change propagation.

(b) $S \cap \triangle S =_r \phi$: All inserted tuples are new.

Informally, minimality guarantees that no unnecessary tuples are generated in the change sets.

A violation of minimality could be more serious than a matter of lost efficiency. The correctness of a given application of change propagation could very well depend on minimality. For example, suppose that we want to maintain the integrity constraint $\text{size}(S) = \text{size}(T)$. Assuming that the constraint is valid, then using a change propagation algorithm we could simply check that $\text{size}(\triangle S) - \text{size}(\bigtriangledown S) = \text{size}(\triangle T) - \text{size}(\bigtriangledown T)$ before committing a transaction that changes some of the relations mentioned in S and T. However, this would be incorrect if our algorithm violated minimality.

II. The Qian-Wiederhold algorithm

We now present a simplified description of the change propagation algorithm of [1]. This algorithm is based on the equations listed in figure 1. Each equation, when read from left to right, is interpreted as a *rewrite rule* that describes how to propagate upward change expressions (labeled with symbols $\bigtriangledown$ and $\triangle$). Given the expression $\text{pre}(t, Q)$, the algorithm repeatedly applies the rules to propagate change expressions to the top of a relational expression, starting with expressions of the form $\bigtriangledown R_i$ or $\triangle R_i$. This is done until all subexpressions of the form $\bigtriangledown R_i$ or $\triangle R_i$ have been removed. These propagations are performed in two stages:

$$\text{pre}(t, Q) \Longrightarrow^* Q^- \cup \triangle Q$$

(first propagate all positive changes)

$$\Longrightarrow^* (Q - \bigtriangledown Q) \cup \triangle Q$$

(then propagate all negative changes)

The precise order of propagations within these two stages is not specified.

Example. Returning to our running example, $Q \stackrel{\text{def}}{=} R_1 \cup R_2$ and $t \stackrel{\text{def}}{=} \{R_1 \leftarrow R_1 - \bigtriangledown R_1, R_2 \leftarrow R_2 \cup \triangle R_2.\}$, the Qian-Wiederhold algorithm propagates these changes as follows:

$$\begin{array}{cl} & \text{pre}(t, Q) \\ \stackrel{\text{def}}{=} & (R_1 - \bigtriangledown R_1) \cup (R_2 \cup \triangle R_2) \\ \stackrel{8}{\Longrightarrow} & ((R_1 - \bigtriangledown R_1) \cup R_2) \cup (\triangle R_2 - (R_1 - \bigtriangledown R_1)) \\ \stackrel{7}{\Longrightarrow} & ((R_1 \cup R_2) - (\bigtriangledown R_1 - R_2)) \cup (\triangle R_2 - (R_1 - \bigtriangledown R_1)) \\ = & (Q - \bigtriangledown Q) \cup \triangle Q \end{array}$$

where $\triangle Q \stackrel{\text{def}}{=} \triangle R_2 - (R_1 - \bigtriangledown R_1)$ and $\bigtriangledown Q \stackrel{\text{def}}{=} \bigtriangledown R_1 - R_2$.

It is claimed on page 340 of [1] (assuming each pair $\bigtriangledown R_i$, $\triangle R_i$ represents a minimal change to base relation R_i) that

> The result of the algorithm is minimal in the sense that $\bigtriangledown \mathcal{E}$ is contained in $\mathcal{E}$ while $\triangle \mathcal{E}$ is disjoint from $\mathcal{E}$.

However, this is not the case.

Proposition 1: *The Qian-Wiederhold algorithm violates minimality.*

Proof. The example presented above serves as a counter-example since

$$Q \cap \triangle Q =_r \bigtriangledown R_1 \cap \triangle R_2,$$

which is not guaranteed to be empty. To see this, consider the following example. Let $R_1 = \{a, b\}$, $R_2 = \{a\}$, $\bigtriangledown R_1 = \{b\}$, and $\triangle R_2 = \{b, c\}$. Then $Q = \{a, b\}$, and $\triangle Q = \{b, c\} - (\{a, b\} - \{b\}) = \{b, c\}$, which is *not* disjoint from Q. □

III. An Improved Change Propagation Algorithm

In this section we present a simple recursive algorithm for relational change propagation that does preserve minimality.

Repeated application of the rules of figure 1 guarantees a solution, as proved in [1]. However, we have shown that such a solution is not necessarily a minimal one. How can this be repaired? The following claim tells us that any solution can be transformed into a minimal one:

Proposition 2: *Suppose that* $S =_r (Q - \bigtriangledown_1 Q) \cup \triangle_1 Q$. *Let*

$$\bigtriangledown_2 Q \stackrel{\text{def}}{=} (Q \cap \bigtriangledown_1 Q) - \triangle_1 Q \qquad \textit{and} \qquad \triangle_2 Q \stackrel{\text{def}}{=} \triangle_1 Q - Q$$

Then

a) $S =_r (Q - \bigtriangledown_2 Q) \cup \triangle_2 Q$
b) $\bigtriangledown_2 Q \subseteq_r Q$
c) $Q \cap \triangle_2 Q =_r \phi$.

Proof. That b) and c) hold follows immediately from the definitions, so we only have to prove a):

$$\begin{array}{cl} & (Q - \bigtriangledown_2 Q) \cup \triangle_2 Q \\ = & (Q - ((Q \cap \bigtriangledown_1 Q) - \triangle_1 Q)) \cup (\triangle_1 Q - Q) \\ =_r & (Q - (Q \cap \bigtriangledown_1 Q)) \cup (Q \cap \triangle_1 Q) \cup (\triangle_1 Q - Q) \text{ by } (10) \\ =_r & (Q - (Q \cap \bigtriangledown_1 Q)) \cup \triangle_1 Q \\ =_r & (Q - \bigtriangledown_1 Q) \cup \triangle_1 Q \end{array}$$

□

Note that proposition 2 could be applied directly to the results of the algorithm of [1] to obtain a minimal solution. However, we prefer to present a new algorithm that preserves minimality at each step.

Two mutually recursive functions, $\bigtriangledown(t,Q)$ and $\bigtriangleup(t,Q)$, are presented in figure 2. For readability, we use the abbreviations $\mathrm{add}(t,S)$ for $S \cup \bigtriangleup(t,S)$, $\mathrm{sub}(t,S)$ for $S - \bigtriangledown(t,S)$, and $\mathrm{mod}(t,S)$ for $(S - \bigtriangledown(t,S)) \cup \bigtriangleup(t,S)$. It is important to emphasize that these expressions are to be evaluated in the database state *before* the execution of t.

Here is a sketch of the process by which these functions were derived. Consider the query $Q = S \cup T$. First note that $\mathrm{pre}(t,Q) = \mathrm{pre}(t,S) \cup \mathrm{pre}(t,T)$. Now, assume that we are able to construct minimal solutions for the subexpressions S and T so that $\mathrm{pre}(t,Q) =_r ((S - \bigtriangledown S) \cup \bigtriangleup S) \cup ((T - \bigtriangledown T) \cup \bigtriangleup T)$. Next, we obtain a general solution by repeated applications of the propagation rules of figure 1 to obtain $\mathrm{pre}(t,Q) =_r (Q - \bigtriangledown_1 Q) \cup \bigtriangleup_1 Q$, where $\bigtriangledown_1 Q = (\bigtriangledown S - \mathrm{sub}(t,T)) \cup (\bigtriangledown T - S)$ and $\bigtriangleup_1 Q = (\bigtriangleup S - \mathrm{mod}(t,T)) \cup (\bigtriangleup T - \mathrm{sub}(t,S))$. Finally, by application of proposition 2, together with simplification that relies on minimality, we obtain $\bigtriangledown Q = (\bigtriangledown S - \mathrm{mod}(t,T)) \cup (\bigtriangledown T - \mathrm{mod}(t,S))$ and $\bigtriangleup Q = (\bigtriangleup S - T) \cup (\bigtriangleup T - S)$. This last step is quite important since the assumptions of minimality would not be available to a query optimizer at a later stage. When applied to all relational operators, this process naturally gives rise to two mutually recursive functions of figure 2.

Q	$\bigtriangledown(t,Q)$
R	$\begin{cases} \bigtriangledown R & \text{if } R \leftarrow (R - \bigtriangledown R) \cup \bigtriangleup R \text{ is in } t \\ \phi & \text{otherwise} \end{cases}$
$\sigma_p(S)$	$\sigma_p(\bigtriangledown(t,S))$
$\Pi_A(S)$	$\Pi_A(\bigtriangledown(t,S)) - \Pi_A(\mathrm{mod}(t,S))$
$S \times T$	$(\bigtriangledown(t,S) \times T) \cup (S \times \bigtriangledown(t,T))$
$S \cup T$	$(\bigtriangledown(t,S) - \mathrm{mod}(t,T)) \cup (\bigtriangledown(t,T) - \mathrm{mod}(t,S))$
$S \cap T$	$(\bigtriangledown(t,S) \cap T) \cup (\bigtriangledown(t,T) \cap S)$
$S - T$	$(\bigtriangledown(t,S) - T) \cup (\bigtriangleup(t,T) \cap S)$
$S \bowtie T$	$(\bigtriangledown(t,S) \bowtie T) \cup (S \bowtie \bigtriangledown(t,T))$

Q	$\bigtriangleup(t,Q)$
R	$\begin{cases} \bigtriangleup R & \text{if } R \leftarrow (R - \bigtriangledown R) \cup \bigtriangleup R \text{ is in } t \\ \phi & \text{otherwise} \end{cases}$
$\sigma_p(S)$	$\sigma_p(\bigtriangleup(t,S))$
$\Pi_A(S)$	$\Pi_A(\bigtriangleup S) - \Pi_A(S)$
$S \times T$	$(\mathrm{mod}(t,S) \times \bigtriangleup(t,T)) \cup (\bigtriangleup(t,S) \times \mathrm{mod}(t,T))$
$S \cup T$	$(\bigtriangleup(t,S) - T) \cup (\bigtriangleup(t,T) - S)$
$S \cap T$	$(\bigtriangleup(t,S) \cap \mathrm{mod}(t,T)) \cup (\bigtriangleup(t,T) \cap \mathrm{mod}(t,S))$
$S - T$	$(\bigtriangleup(t,S) - \mathrm{mod}(t,T)) \cup (\bigtriangledown(t,T) \cap \mathrm{sub}(t,S))$
$S \bowtie T$	$(\mathrm{mod}(t,S) \bowtie \bigtriangleup(t,T)) \cup (\bigtriangleup(t,S) \bowtie \mathrm{mod}(t,T))$

Fig. 2 Mutually Recursive functions $\bigtriangledown$ and $\bigtriangleup$.

Theorem 1: *Let t be a minimal transaction. That is, each pair $\bigtriangledown R_i$, $\bigtriangleup R_i$ in t represents a minimal change to base relation R_i. Let Q be any relational expression. Then*

(a) $\mathrm{pre}(t,Q) =_r (Q - \bigtriangledown(t,Q)) \cup \bigtriangleup(t,Q)$
(b) $\bigtriangledown(t,Q) \subseteq_r Q$
(c) $\bigtriangleup(t,Q) \cap Q =_r \phi$

Proof. All justifications of the form "by (m)" refer to the equations in figure 1.

To reduce the clutter, we introduce the following notation for the proof. First, we fix a transaction t. Then by S^- we mean $\mathrm{sub}(t,S)$, S^+ stands for $\mathrm{add}(t,S)$ and S^{n} is $\mathrm{mod}(t,S)$. Similarly for T we use T^-, T^+ and T^{n}. We also use $\bigtriangledown S$ and $\bigtriangleup S$ for $\bigtriangledown(t,S)$ and $\bigtriangleup(t,S)$, omitting t, and similarly for T. The proof proceeds by induction on the structure of Q. In each case we must show that the results of $\bigtriangleup(t,Q)$ and $\bigtriangledown(t,Q)$ satisfy conditions (a), (b), and (c).

Base case. If $Q = R$, then $\mathrm{pre}(t,Q) = \mathrm{pre}(t,R) = (R - \bigtriangledown R) \cup \bigtriangleup R = (R - \bigtriangledown(t,R)) \cup \bigtriangleup(t,R) = (Q - \bigtriangledown(t,Q)) \cup \bigtriangleup(t,Q)$. The theorem follows from the assumption that t is a minimal transaction.

Induction step. Assume that (a), (b), and (c) hold for any expression smaller than Q. Now proceed by case analysis of the structure of Q. We present two cases for illustration; others are similar.

Projection. Case $Q = \Pi_A(S)$. By propagation we obtain:

$$\begin{aligned}
& \mathrm{pre}(t,Q) \\
= \; & \Pi_A(\mathrm{pre}(t,S)) \\
=_r \; & \Pi_A((S - \bigtriangledown S) \cup \bigtriangleup S) \\
=_r \; & \Pi_A(S - \bigtriangledown S) \cup (\Pi_A(\bigtriangleup S) - \Pi_A(S^-)) \\
=_r \; & (\Pi_A(S) - (\Pi_A(S) - \Pi_A(S^-))) \cup (\Pi_A(\bigtriangleup S) - \Pi_A(S^-)) \\
\stackrel{\mathrm{def}}{=} \; & (Q - \bigtriangledown_1 Q) \cup \bigtriangleup_1 Q
\end{aligned}$$

where $\bigtriangledown_1 Q = \Pi_A(S) - \Pi_A(S^-)$ and $\bigtriangleup_1 Q = \Pi_A(\bigtriangleup S) - \Pi_A(S^-)$. The first equation is obtained by definition, the second by induction, using (a), the third by (4) and the fourth by (3). It is clear that $\bigtriangledown_1 Q \subseteq_r Q$. Forcing minimality (by application of proposition 2) we get

$$\begin{aligned}
& \bigtriangledown_2 Q \\
\stackrel{\mathrm{def}}{=} \; & (Q \cap \bigtriangledown_1 Q) - \bigtriangleup_1 Q \\
=_r \; & \bigtriangledown_1 Q - \bigtriangleup_1 Q \\
= \; & (\Pi_A(S) - \Pi_A(S^-)) - (\Pi_A(\bigtriangleup S) - \Pi_A(S^-)) \\
=_r \; & \Pi_A(S) - (\Pi_A(S^-) \cup (\Pi_A(\bigtriangleup S) - \Pi_A(S^-))) \\
=_r \; & \Pi_A(S) - (\Pi_A(S^-) \cup \Pi_A(\bigtriangleup S)) \\
=_r \; & \Pi_A(S^- \cup \bigtriangledown S) - (\Pi_A(S^-) \cup \Pi_A(\bigtriangleup S)) \\
=_r \; & (\Pi_A(S^-) \cup \Pi_A(\bigtriangledown S)) - (\Pi_A(S^-) \cup \Pi_A(\bigtriangleup S)) && \text{by (4)} \\
=_r \; & \Pi_A(\bigtriangledown S) - (\Pi_A(S^-) \cup \Pi_A(\bigtriangleup S)) \\
=_r \; & \Pi_A(\bigtriangledown S) - \Pi_A(S^{\mathrm{n}}) && \text{by (4)} \\
= \; & \bigtriangledown Q
\end{aligned}$$

and

$$\begin{aligned}
& \bigtriangleup_2 Q \\
\stackrel{\mathrm{def}}{=} \; & \bigtriangleup_1 Q - Q \\
= \; & (\Pi_A(\bigtriangleup S) - \Pi_A(S^-)) - \Pi_A(S) \\
=_r \; & \Pi_A(\bigtriangleup S) - (\Pi_A(S^-) \cup \Pi_A(S)) \\
=_r \; & \Pi_A(\bigtriangleup S) - \Pi_A(S) && \text{since } \Pi_A(S^-) \subseteq_r \Pi_A(S) \\
= \; & \bigtriangleup Q
\end{aligned}$$

Difference. Case $Q = S - T$. By propagation we obtain:

$$\begin{array}{ll} & \mathrm{pre}(t,Q) \\ = & \mathrm{pre}(t,S) - \mathrm{pre}(t,T) \\ =_r & ((S - \triangledown S) \cup \triangle S) - ((T - \triangledown T) \cup \triangle T) \\ =_r & ((S - \triangledown S) - ((T - \triangledown T) \cup \triangle T)) \cup (\triangle S - T^{\mathrm{n}}) \\ =_r & ((S - \triangledown S) - ((T \cup \triangle T) - \triangledown T)) \cup (\triangle S - T^{\mathrm{n}}) \\ =_r & ((S - \triangledown S) - (T \cup \triangle T)) \cup ((\triangle S - T^{\mathrm{n}}) \cup (\triangledown T \cap S^-)) \\ = & ((S - \triangledown S) - (T \cup \triangle T)) \cup \triangle Q \\ =_r & ((S - (T \cup \triangle T)) - (\triangledown S - T^+)) \cup \triangle Q \\ =_r & ((S - T) - ((\triangledown S - T^+) \cup (\triangle T \cap S))) \cup \triangle Q \\ \stackrel{\mathrm{def}}{=} & (Q - \triangledown_1 Q) \cup \triangle Q \end{array}$$

where $\triangledown_1 Q = (\triangledown S - T^+) \cup (\triangle T \cap S)$ and $\triangle Q = (\triangle S - T^{\mathrm{n}}) \cup (\triangledown T \cap S^-)$. The first equation is obtained by the definition of pre-expressions, the second by induction, using (a), the third by (11), the fourth by (b) and (c), the fifth by (10), the seventh by (9) and the eighth by (12).

By induction, using (b) and (c), it is easy to check that $\triangledown_1 Q \subseteq_r Q$. and $\triangle Q \cap Q =_r \phi$. The expression $\triangledown_1 Q$ can be further simplified as

$$\begin{array}{lll} & (\triangledown S - T^+) \cup (\triangle T \cap S) & \\ =_r & ((\triangledown S - T) - (\triangledown S \cap \triangle T)) \cup (\triangle T \cap S) & \text{by } (12) \\ =_r & (\triangledown S - T) \cup (\triangle T \cap S) & \\ = & \triangledown Q & \end{array}$$

The last equation is valid since $\triangledown S \cap \triangle T \subseteq_r \triangle T \cap S$, by (b). Theorem 1 is proved. □

Algorithm. Our algorithm is simply this: given inputs t and Q, use the functions $\triangledown(t,Q)$ and $\triangle(t,Q)$ to compute a solution for $\mathrm{pre}(t,Q)$. Note that in an actual implementation $\triangledown(t,Q)$ and $\triangle(t,Q)$ could be combined into one recursive function. Thus the algorithm requires only one pass over the expression Q.

Example. Recall the counter-example presented in proposition 1: $Q \stackrel{\mathrm{def}}{=} R_1 \cup R_2$ and $t \stackrel{\mathrm{def}}{=} \{R_1 \leftarrow R_1 - \triangledown R_1, R_2 \leftarrow R_2 \cup \triangle R_2.\}$. Our algorithm produces

$$\begin{array}{ll} & \triangledown(t,Q) \\ = & \triangledown(t, R_1 \cup R_2) \\ = & (\triangledown(t,R_1) - \mathrm{mod}(t,R_2)) \cup (\triangledown(t,R_2) - \mathrm{mod}(t,R_1)) \\ = & (\triangledown R_1 - (R_2 \cup \triangle R_2)) \cup (\phi - \mathrm{mod}(t,R_1)) \\ =_r & \triangledown R_1 - (R_2 \cup \triangle R_2) \end{array}$$

and

$$\begin{array}{ll} & \triangle(t,Q) \\ = & \triangle(t, R_1 \cup R_2) \\ = & (\triangle(t,R_1) - R_2) \cup (\triangle(t,R_2) - R_1) \\ = & (\phi - R_2) \cup (\triangle R_2 - R_1) \\ =_r & \triangle R_2 - R_1 \end{array}$$

For the concrete example, $R_1 = \{a,b\}$, $R_2 = \{a\}$, $\triangledown R_1 = \{b\}$, and $\triangle R_2 = \{b,c\}$, it is easy to see that $\triangledown(t,Q) =_r \phi$ and $\triangle(t,Q) =_r \{c\}$. In particular, $\triangle(t,Q) \cap Q = \phi$.

IV. Remarks

We believe that our algorithm has some advantages. The recursive form of the algorithm lends itself to a correctness proof that proceeds by a straightforward structural induction, and also allows us to exploit the invariant of minimality in the simplification of the results. In addition, it is easy to see that the change sets produced by the algorithm are of a special form – they are "controlled" in some sense by the changes to base relations. For instance, to compute $Q = \triangledown(t,S) - \mathrm{mod}(t,T)$, we compute $\triangledown(t,S)$, $\triangledown(t,T)$ and $\triangle(t,T)$. To compute Q, we iterate over $\triangledown(t,S)$, and for each $x \in \triangledown(t,S)$ check if either $x \in T$ and $x \notin \triangledown(t,T)$, or $x \in \triangle(t,T)$. Note that we do not have to compute and materialize $\mathrm{mod}(t,T)$ in order to compute Q. In fact, the complexity of incremental recomputation is controlled by the size of change sets, which are typically smaller than relations and views. This point is explored in more detail in the context of a multiset calculus presented by the authors in [2].

References

[1] X. Qian and G. Wiederhold. Incremental recomputation of active relational expressions. *IEEE Transactions on Knowledge and Data Engineering*, 3(3):337–341, 1991.

[2] T. Griffin and L. Libkin. Incremental maintenance of views with duplicates. *Proceedings of the 1995 ACM-SIGMOD International Conference on Management of Data*, ACM Press, 1995, pages 328-339.