Incremental Recomputation in Local Languages[*][†]

Guozhu Dong
Dept of Comp. Sc. and Engr
Wright State University
Dayton, OH 45435
Email: gdong@cs.wright.edu

Leonid Libkin[§]
Department of Computer Science
University of Toronto
Toronto, Ontario, M5S 3H5, Canada
Email: libkin@cs.toronto.edu

Limsoon Wong
Kent Ridge Digital Labs
21 Heng Mui Keng Terrace
Singapore 119613
Email: limsoon@krdl.org.sg

Abstract

We study the problem of maintaining recursively-defined views, such as the transitive closure of a relation, in traditional relational languages that do not have recursion mechanisms. The main results of this paper are negative ones: we show that a certain property of query languages implies impossibility of such incremental maintenance. The property we use is locality of queries, which is known to hold for relational calculus and various extensions, including those with grouping and aggregate constructs (essentially, plain SQL).

1 Introduction

It is well known that relational calculus, or first-order logic, cannot express recursive queries such as transitive closure or same-generation, cf. [1]. This is one of the main reasons why languages extending first-order logic, such as various fixpoint logics, have been so extensively studied in database theory. However, most practical database systems still use query languages with limited expressive power. Indeed, the plain SQL that is used for writing the majority of queries is essentially first-order logic extended with grouping and aggregation, and as such it cannot code recursion mechanisms.

What can one do if one needs to know the result of a recursive query? One possibility is to use a general-purpose programming language to compute such a query. However, this may not be desirable,

[*]Part of this work was done when the authors were visiting each other in 5 out of 6 possible combinations (Libkin never visited Dong.). A revised version was made during Libkin's visit to INRIA-Rocquencourt.

[†]Contact author: Leonid Libkin, Department of Computer Science, University of Toronto, Toronto, Ontario, M5S 3H5, Canada. Phone: (416) 978-4158, Fax: (416) 978-4765. Email: libkin@cs.toronto.edu.

[§]Research affiliation: Bell Laboratories.

as one no longer has access to a declarative query language and to a query language optimizer. An alternative solution is to use a general-purpose programming language to compute the initial result of a query, and then update the result every time the database changes. For example, for the transitive closure query this amounts to updating the transitive closure of a graph every time an edge is inserted or deleted.

The problem of updating the results of queries (called *views*) when the underlying database changes is known under the name of *view maintenance*, or *incremental recomputation*. There is also extensive literature on dynamic algorithms (see, for example, [19, 27]) which does not consider the issue of a query language in which updates are expressed. Since databases are normally queried and updated by languages of limited expressive power, this issue becomes important for view maintenance. There is a large body of literature on view maintenance that assumes that views are defined and maintained using the *same* language. Numerous algorithms exist dealing with fragments of relational algebra [2], full relational algebra [29, 13], bag (multiset) languages [12], languages with grouping and aggregation [16, 30] and others; see [15] for a survey.

However, much less is known in the case when a view is defined in one, more powerful language, and is maintained in another one, less powerful. Those papers that do consider this situation deal with the case when a recursive query is computable in polynomial time and definable in a language such as recursive datalog, and the maintenance is done in relational calculus.

The query that received most attention is the transitive closure. It can be easily shown that the transitive closure can be maintained under the insertion of edges [5, 3]. A more interesting result of [28, 6] shows that transitive closure of *undirected* graphs can always be maintained, provided some auxiliary (binary) relations can be used. For directed graphs, the situation is more complex. It is known [4] that the transitive closure of acyclic graphs can be maintained in relational calculus, but the question is still open for arbitrary directed graphs.

In general, it is known that every query that can be incrementally maintained in relational calculus has PTIME data complexity [28, 6]. It is conjectured that the containment is strict, but, as was shown in [9], when auxiliary relations of arity 2 or higher are allowed, proving such bounds amounts to proving lower bounds for a general model of computation, and bounds of this kind are *extremely hard* to obtain.

For auxiliary relations of arity 1 (or no auxiliary relations), some bounds have been reported for relational calculus [6]. These results are not completely satisfactory as they are very closely tied to a particular language; in fact, the proofs rely on Ehrenfeucht-Fraïssé games. It would thus be impossible to extend proof techniques from [6] to cover languages that more closely resemble the commercial lingua franca of the database world—SQL. Nor it is clear how to extend those results to deal with operations such as a built-in linear order.

Thus, the main goal of this paper is to find properties of query languages (describing their expressiveness) that would imply unmaintainability of certain recursive views. The main property we use is that of *locality*. The ideas we describe here are the ones typically used as tools in finite-model theory for proving inexpressibility results. In particular, the fact that they are possessed by relational calculus (even with aggregate functions) is known. In terms of recursive queries, we concentrate on the two most famous examples of queries expressible in datalog but not in relational calculus: transitive closure and same-generation. Given the fact that some recursive queries can be maintained in relational calculus, it is probably impossible to find general characterizations of this kind, and thus one has to concentrate on some particular queries. However, we believe that the techniques developed in this paper are easily extendible to deal with other queries.

The rest of the paper is organized as follows. We define the framework for incremental maintenance in Section 2. In Section 3, we define locality of query languages. In Section 4, we show how to use locality to derive bounds on incremental maintenance in query languages.

2 Definition of incremental maintenance

In this section, we describe our setting for the problem of incremental maintenance of views. We assume that all values that can appear in a database are drawn from a countably infinite domain U. A relational schema SC is a collection $\{R_1, \ldots, R_l\}$ of relation names, each name R_i having arity $m_i > 0$. By $Inst(SC)$ we denote the set of instances of the schema SC, that is, the set of families of finite relations $R_1^D \subset U^{m_1}, \ldots, R_l^D \subset U^{m_l}$. We shall write just R_i in place of R_i^D when it does not lead to confusion.

Let Q be a query, that is, a map that associates to every database D in $Inst(SC_{in})$ an instance of an output schema SC_{out}. When SC_{out} consists of a single n-ary relation symbol, we speak of an n-ary query.

We say that Q can be (incrementally) maintained under insertions in a language $\mathcal{L}$, if for each m-ary symbol R in SC_{in}, there exists a $\mathcal{L}$ query Q^R_{ins} that takes as its inputs $D \in Inst(SC_{in})$, $Q(D) \in Inst(SC_{out})$, and an m-ary tuple $\vec{t}$, and returns the result of Q on D updated in such a way that $\vec{t}$ is inserted into R. In other words,

$$Q^R_{ins}(D, Q(D), \vec{t}) \quad = \quad Q(D[R := R \cup \{\vec{t}\}])$$

where $D[R := S]$) means D in which the relation R is replaced with the relation S.

Similarly, Q can be (incrementally) maintained under deletions in a language $\mathcal{L}$, if for each m-ary symbol R in SC_{in}, there exists a $\mathcal{L}$ query Q^R_{del} that takes as its inputs $D \in Inst(SC_{in})$, $Q(D) \in Inst(SC_{out})$, and an m-ary tuple $\vec{t}$, and returns the result of Q on D updated in such a way that $\vec{t}$ is deleted from R. In other words,

$$Q^R_{del}(D, Q(D), \vec{t}) \quad = \quad Q(D[R := R - \{\vec{t}\}]).$$

This definition assumes that no auxiliary data are kept. That is, to recompute the value of Q after a single insertion or deletion, only the old value of Q, the old database and the inserted (deleted) tuple are needed. There are example of queries that cannot be recomputed in such a way, but can be recomputed in the presence of some auxiliary data. To capture this situation, we say that Q is (incrementally) maintainable under insertions (deletions) in the presence of auxiliary relations if, there exists a schema SC_{aux}, disjoint from SC_{in} and SC_{out}, and a query Q' from $Inst(SC_{in} \cup SC_{out} \cup SC_{aux})$ to $Inst(SC_{out} \cup SC_{aux})$ such that Q' is maintainable under insertions (deletions) to SC_{in}-relations, and Q' is an extension of Q.

To define this latter notion precisely, we have to explain how the initial value of the output of Q and of the auxiliary relations is obtained. In one model, we start with the empty database, and keep inserting and deleting tuples. In the other model, we are given the initial value of $D, Q(D)$ and the auxiliary data. Note that this is the model that makes sense when we deal with maintenance under deletions only. When we say that Q' is an extension of Q, we mean that for every $D \in \mathit{Inst}(SC)$, the associated auxiliary relations $V \in \mathit{Inst}(SC_{aux})$, and $Q(D)$, for any sequence of updates $u_1, \ldots, u_n$ to the database D, and the sequence $D_0 = D, Z_0 = (Q(D), V), \ldots, D_{i+1} = u_{i+1}(D_i), Z_{i+1} = Q'_{u_{i+1}}(D_i, Z_i)$, it holds that the values of SC_{out}-relations in Z_i are $Q(D_i)$, $i \leq n$. Here Q'_u is the query maintaining Q' under the update u.

Note that there are some subtle differences between the two ways of initializing auxiliary data [6, 28]. However, it will be clear from the proofs in the next section that our results are not affected by the way in which data is initialized.

If all relations in SC_{aux} are at most unary (binary, etc.) then we say that Q is maintainable in the presence of unary (binary, etc.) auxiliary relations. In this paper we concentrate on maintenance with at most unary auxiliary relations. As pointed out in the introduction, proving bounds for maintenance with auxiliary relations of arity 2 and higher is probably beyond reach. Note also that in the algorithmic literature on view maintenance one typically considers maintenance without auxiliary data, see [2, 13, 12, 15, 16].

Another parameter of incremental maintenance is whether the value of auxiliary relations is the same for any sequence of updates that leads to a given database. It was shown in [7] that fewer queries can be incrementally maintained under this restriction. In what follows, we thus consider this more powerful model of incremental recomputation, as we are interested in proving negative results.

3 Query languages and locality

In the rest of the paper, when we say "language $\mathcal{L}$," we always assume that the following is true of $\mathcal{L}$:

1. $\mathcal{L}$ contains relational calculus, or first-order logic, as a sublanguage.
2. $\mathcal{L}$ is closed under first-order operations.
3. $\mathcal{L}$ is closed under substitutions. That is, assume that there is a $\mathcal{L}$ query $Q : \mathit{Inst}(SC_{in}) \to \mathit{Inst}(SC_{out})$. Assume that some of the relations $R_1, \ldots, R_k$ are defined by means of other queries, $Q_1, \ldots, Q_k$ on input databases of schemas $SC_1, \ldots, SC_k$. Let $SC' = SC - \{R_1, \ldots, R_k\}$. Then there exists a $\mathcal{L}$ query $Q' : \mathit{Inst}(SC' \cup SC_1 \cup \ldots \cup SC_k) \to \mathit{Inst}(SC_{out})$ such that

   $$Q'(D) \;=\; Q(D[R_1 := Q_1(D_1), \ldots, R_k := Q_k(D_k)])$$

   where D_i is the SC_i part of D.

For example, relational calculus and plain SQL are such languages.

Next we define the concept of local queries and local languages. Given a schema SC_{in} and $D \in Inst(SC_{in})$, its *active domain*, $adom(D)$, is the set of all elements from D that occur in relations from D. The *Gaifman graph* [8, 11, 10] of D, $\mathcal{G}(D)$ is defined as a graph $\langle A, E \rangle$, where $A = adom(D)$, and (a, b) is in E iff there is a tuple $\vec{t} \in R_i^D$ for some i such that both a and b are in $\vec{t}$. The distance $d(a, b)$ is defined as the length of the shortest path from a to b in $\mathcal{G}(D)$; we assume $d(a, a) = 0$. If $\vec{a} = (a_1, \ldots, a_n)$, then $d(\vec{a}, b) = \min_{ij} d(a_i, b)$.

Given a tuple $\vec{a}$ of elements of $A = adom(D)$, its *r-ball* $S_r^D(\vec{a})$ is $\{b \in A \mid d(\vec{a}, b) \leq r\}$. Its *r-neighborhood* $N_r^D(\vec{a})$ is defined as an instance of SC_{in} where each relation symbol R_i is interpreted as a set of tuples $\vec{t} \in R_i^D$ where all elements in $\vec{t}$ are from $S_r^D(\vec{a})$. Furthermore, we treat $\vec{a}$ as distinguished constants.

We write $\vec{a} \approx_r^D \vec{b}$ if $N_r^D(\vec{a})$ and $N_r^D(\vec{b})$ are isomorphic; that is, if there exists a one-to-one map $h : S_r^D(\vec{a}) \to S_r^D(\vec{b})$ such that $h(\vec{a}) = \vec{b}$ and $\vec{t} \in R_i$ iff $h(\vec{t}) \in R_i$, for every $i \leq l$ and a tuple $\vec{t}$ of elements of $S_r^D(\vec{a})$.

Definition 1 (cf. [11, 17]) *An n-ary query Q is called* local *if there exists a number $r \geq 0$ such that, for any database $D \in Inst(SC_{in})$ and any $\vec{a}, \vec{b} \in adom(D)^n$,*

$$\vec{a} \approx_r^D \vec{b} \quad \textit{implies} \quad \vec{a} \in Q(Å) \;\; \textit{iff} \;\; \vec{b} \in Q(Å).$$

The minimum such r is called the locality rank *of Q, and is denoted by* $\mathsf{lr}(Q)$.

A language is called local if every m-ary query definable in it, $m > 0$, is local. □

Gaifman's theorem [11] on locality of first-order queries implies that every query definable in relational calculus is local.

It is rather pleasant that locality can be established for the language that is essentially plain SQL. SQL, the dominant language of commercial databases, adds two main features to the relational calculus: grouping and aggregation. In a number of papers [23, 25, 18] we studied a theoretical reconstruction of plain SQL and its expressive power. Our approach was as follows. To model the grouping feature, we considered a *nested* relational language, as in [3]. If one deals with the usual queries from flat relational databases to flat relational databases, then nested sets can appear as intermediate results. It is known that the nested relational algebra is an extension of relational algebra that has enough power to express the `GROUPBY` and `HAVING` clauses of SQL. To model aggregation, we made the language two-sorted. In other words, it has two base types, one of them being the type of rational numbers. By graph queries we meant queries of the type $\{b \times b\} \to \{b \times b\}$, where b is the other base type. We assumed that the usual rational arithmetic is present. Furthermore, we added an operator for summation of function values over a column, and showed that such a language computes the standard aggregate functions such as `AVG`, `TOTAL`, `COUNT`.

Then [25] established locality of *relational queries* in such a language (that is, queries that do not have values of the numerical type in their input and output, but can use them for intermediate steps of the computation). Furthermore, [18] showed (a stronger form of) locality under the assumption that *every* arithmetic function and *every* aggregate operator is present in the language.

Another very useful result is that queries definable in relational calculus in the presence of a built-in order relation are local, provided they are order-invariant [14]. Normally, adding order as one of the relations in D would render the concept of locality meaningless, as for every a, its unit ball $S_1(a)$ would contain the entire active domain. However, one can also define the concept of neighborhoods with respect to the original database, and use order as an additional built-in predicate, and restrict one's attention to queries that do not depend on the particular interpretation of this built-in order. The result is the *order-invariant* relational calculus, which is known to be a proper extension of the relational calculus [1]. The result of [14] shows that it is still local.

4 Incremental recomputation of recursive queries

In this section we prove our main results showing that certain recursive queries cannot be incrementally maintained in local languages. The queries we choose are prototypical recursive queries that can be expressed in languages such as datalog, but not in relational calculus: the transitive closure query tc, and the same-generation query sg. In both cases, the input is a directed graph (binary relation) R. The transitive closure query is given by the following datalog program:

$$\begin{array}{lcl} tc(x,y) & \text{:-} & R(x,y) \\ tc(x,y) & \text{:-} & R(x,z), tc(z,y). \end{array}$$

The same-generation query is given by the program

$$\begin{array}{lcl} sg(x,x) & \text{:-} & \\ sg(x,y) & \text{:-} & sg(x',y'), R(x,x'), R(y,y'). \end{array}$$

That is, a pair (a,b) belongs to the output of the same-generation query iff there is a node c and two equi-distant walks in the graph, one from c to a, and the other from c to b.

It is well known that the transitive closure query can be incrementally maintained in relational calculus under insertions (essentially, by coding Warshall's algorithm, cf. [3, 5]). Here we show that other maintenance queries are impossible. The proof applies to all local languages, and, unlike the techniques of [6], it is not limited to relational calculus.

Theorem 2 *Let $\mathcal{L}$ be a local language. Then it cannot incrementally maintain the transitive closure query under deletions, nor the same-generation query under either deletions or insertions, even in the presence of unary auxiliary relations.*

Proof. Throughout the proof, R is a binary relation symbol for the input graph, and G denotes the graph itself. The main technique is the following. Let $\mathcal{C}$ be a class of graphs. We say that an n-ary query Q on graphs is *$\mathcal{L}$-definable on $\mathcal{C}$ with unary relations* if there exists a number m, a schema $SC_m = \{R, V_1, \ldots, V_m\}$, and an n-ary query Q' on $Inst(SC_m)$ definable in $\mathcal{L}$ such that, for every graph $G \in \mathcal{C}$, there exists $D \in Inst(SC_m)$ with $R^D = G$, satisfying

$$Q'(D) = Q(G).$$

That is, there is a way to define m unary predicates on the nodes of G such that Q' on the resulting colored graph yields $Q(G)$. Now, in each case, we first show that, assuming that a query can be maintained, a certain query would be $\mathcal{L}$-definable on some class of graphs with unary relations. Then we would show that such definability contradicts locality.

a) Transitive closure under deletions. Let $\mathcal{C}$ be the class of chains, that is, graphs of the form $\{(a_0, a_1), (a_1, a_2), \ldots, (a_{k-1}, a_k)\}$, $k > 0$, where all a_is are distinct. Given any G, let $G^\top$ stand for the complete graph with the same set of nodes as that of G. Assuming that the tc query can be maintained under deletions (perhaps with unary auxiliary relations), we find a query Q^{tc}_{del} that takes in a graph, its transitive closure, an edge to be deleted, and the auxiliary relations, and produces the transitive closure after the deletion. In particular, if G is the chain as above, and $\vec{V}$ is the tuple of unary auxiliary relations, $Q^{tc}_{del}(G \cup \{(a_k, a_0)\}, G^\top, (a_k, a_0), \vec{V})$ returns $tc(G)$, since $G \cup \{(a_k, a_0)\}$ is a cycle, and its transitive closure is $G^\top$. Since the edge (a_k, a_0) is definable from G in relational calculus, and so is $G^\top$, we conclude that the transitive closure query is $\mathcal{L}$-definable on $\mathcal{C}$ with unary relations.

It thus remains to show that this is impossible. Let m be the number of unary relations, and Q' a query that computes the transitive closure of a chain G, given unary relations $V_1, \ldots, V_m$. We can view adding these unary relations as coloring the nodes of G with 2^m colors. Let $r = \mathsf{lr}(Q')$. For any node a in G at the distance at least r from the start and the end nodes, its r-neighborhood is a $2r + 1$-element chain, colored with 2^m colors according to the V_is. There are thus at most $2^{m(2r+1)}$ different types of r-neighborhoods of such nodes in terms of their colors. Hence, for any chain with $k > (2r + 3) \cdot 2^{m(2r+1)} + 2r$ there would be at least $2r + 3$ nodes at the distance at least r from the end nodes, and having the same neighborhood type, no matter how V_is are interpreted. In particular, one can then find two such nodes, a, b, with $d(a, b) > 2r + 1$. This implies that in $D \in \mathit{Inst}(SC_m)$ where the relation R is interpreted as G, we have $(a, b) \approx^D_r (b, a)$, and thus $(a, b) \in Q'(D)$ iff $(b, a) \in Q'(D)$. However, this contradicts the assumption that Q' computes the transitive closure, as exactly one of the pairs (a, b), (b, a), belongs to $tc(G)$. This contradiction proves case a).

b) Same-generation query under insertion. The class $\mathcal{C}$ consists of the graphs of the following form. Let G_0 be the union of a chain $\{(b_0, b_1), \ldots, (b_{p-1}, b_p)\}, p > 1$, and an edge (b_0, b_*), where all b_is and b_* are distinct. Let G_1 be similarly defined as the union of a chain $\{(a_0, a_1), \ldots, (a_{k-1}, a_k)\}, 0 < k < p$ and an edge (a_0, a_*), where $a_* \neq a_i, i = 0, \ldots, k$. We also assume that G_0 and G_1 are disjoint. Then graphs in $\mathcal{C}$ are those of the form $G = G_0 \cup G_1 \cup \{(a_0, b_0)\}$.

Let G' be $G_0 \cup G_1$. Then $sg(G') = \{(a, a) \mid a \text{ node of } G\} \cup \{(a_*, a_1), (a_1, a_*), (b_*, b_1), (b_1, b_*)\}$; in particular, it is definable in relational calculus with G as an input. Note also that the pair (a_0, b_0) is definable in relational calculus (when the input is G), since a_0 is the only node of outdegree 3, and b_0 is the only node of indegree 1 and outdegree 2. Furthermore, for $i, j \neq 0, *$, $(a_i, b_j) \in sg(G)$ iff $j = i - 1$.

Assume now that sg can be maintained under the insertion of edges (perhaps with auxiliary unary relations). Then there is a query Q^{sg}_{ins} that takes in a graph, an edge, and some unary relations $V_1, \ldots, V_m$, and returns the output of the same-generation query on the graph resulting from inserting the input edge into the input graph. In particular, $Q^{sg}_{ins}(G', (a_0, b_0), sg(G'), \vec{V})$ would return $sg(G)$. Since (a_0, b_0) and $sg(G')$ are definable in relational calculus (with G as input), this means that sg is $\mathcal{L}$-definable on $\mathcal{C}$ with unary relations.

To show that this is impossible, let Q' be a query defining sg on $\mathcal{C}$ with unary relations, and let $r = \mathsf{lr}(Q')$. As in the proof of a), we conclude that if k is large enough, there are two indices, $j > i > r$, such that $a_i \approx_r^D a_j$, where D is an extension of G with unary predicates V_is. This holds no matter what the interpretation of V_is is. This implies that $(a_i, b_{j-1}) \approx^D (a_j, b_{j-1})$, since the r-balls of a_i and b_{j-1} are disjoint (and likewise for a_j and b_{j-1}). Thus, locality of Q' would imply that $(a_i, b_{j-1}) \in Q'(D)$ iff $(a_j, b_{j-1}) \in Q'(D)$. Since $(a_i, b_{j-1}) \notin sg(G)$ and $(a_j, b_{j-1}) \in sg(G)$, this contradicts the assumption that sg can be maintained under insertions.

c) Same-generation under deletions. Let $\mathcal{C}$ consist of graphs of the from $\{(a_1, a_2), (a_2, a_3), \ldots, (a_{l-1}, a_l), (a_l, a_{l+1}), \ldots, (a_{2l-1}, a_{2l}), (a_*, a_1), (a_*, a_{l+1})\}$, $l > 1$. That is, the subgraph on the nodes a_i, $i \neq *$, is a chain, and we have edges from a_* to two nodes on this chain: the start a_1 and the middle a_l. Note that for such a graph G, $sg(G)$ is the union of $\{(a, a) \mid a \text{ node of } G\}$, $\{(a_{l+i}, a_i) \mid 1 \leq i \leq l\}$, and $\{(a_i, a_{l+i}) \mid 1 \leq i \leq l\}$.

Let G' be obtained from G by adding two edges: (a_{2l}, a_1) and (a_1, a_1). Then $sg(G') = \{(a_*, a_*)\} \cup \{(a_i, a_j) \mid i, j \neq *\}$. This is because for every $k < l$, and every $N > k$, there is a walk of length N from a_* to a_k of length N, simply by using the loop on a_1 sufficiently many times. Similarly, for every $k > l$, and every $N > 2l + k + 1$, there is a walk of length N from a_* to a_k: one moves to a_l, then to a_{2l}, uses the (a_{2l}, a_1) edge to move back to a_1, stay sufficiently long at a_1 and then moves to a_k along the chain. Hence, $(a_i, a_j) \in sg(G')$ for any $i, j \neq *$.

Now assume now that sg can be maintained under the deletion of edges (perhaps with auxiliary unary relations). Then there is a query Q^{sg}_{del} that takes in a graph, an edge, and some unary relations $V_1, \ldots, V_m$, and returns the output of the same-generation query on the graph resulting from deleting the input edge from the input graph, as well as new values $V'_1, \ldots, V'_m$ of the auxiliary relations. Both pairs (a_{2l}, a_1) and (a_1, a_1) are definable in relational calculus, given G as input (a_{2l} is the only node of outdegree 0, and a_1 is the successor of indegree 1 of the node of indegree 0), as well as $sg(G')$, if G is given as an input. Thus, we can first define $Q^{sg}_{del}(G \cup \{(a_1, a_1), (a_{2l}, a_1)\}, (a_{2l}, a_1), sg(G'), \vec{V})$ in $\mathcal{L}$, which produces $sg(G \cup \{(a_1, a_1)\})$ and the new values $\vec{V}'$ of auxiliary relations. Then, by compositionality, we can define, in $\mathcal{L}$, $Q^{sg}_{del}(G \cup \{(a_1, a_1)\}, (a_1, a_1), sg(G \cup \{(a_1, a_1)\}), \vec{V}')$, which produces $sg(G)$. Thus, the same-generation query is $\mathcal{L}$-definable on $\mathcal{C}$ with unary relations.

We now show that this is impossible. Again, assume that the same-generation query is definable by a query Q' with $\mathsf{lr}(Q') = r$, using m auxiliary relations. Let D refer to the extension of G with unary relations V_is. As before, we can show that for large enough l, there exist two indices $r < i < j < l - r$ such that $a_i \approx_r^D a_j$, no matter what the interpretation of V_is is (since r-neighborhoods of a_i and a_j are $2r + 1$ chains colored with 2^m colors). Therefore, $(a_i, a_{j+l}) \approx_r^D (a_j, a_{j+l})$, as elements in these pairs at the distance at least $2r + 1$ from each other. By the locality of Q', $(a_i, a_{j+l}) \in Q'(G, \vec{V})$ iff $(a_j, a_{j+l}) \in Q'(G, \vec{V})$, and thus Q' cannot define the same-generation query, since $(a_j, a_{j+l}) \in sg(G)$ and $(a_i, a_{j+l}) \notin sg(G)$. This completes the proof. $\Box$

Remark It follows from the proof of a) that transitive closure can be replaced by *deterministic* transitive closure [21] (every node on a path, except the final one, is required to have outdegree 1)—this query is complete for deterministic logspace.

Corollaries

Since relational calculus is local [11], we immediately obtain:

Corollary 3 *It is impossible to incrementally maintain, in relational calculus, the transitive closure query under deletions and the same-generation query under either deletions or insertions, even in the presence of unary auxiliary relations.* $\square$

As we explained in the introduction, proving bounds in the presence of binary auxiliary relations is probably beyond reach. One particular binary relation used very often is a linear order on the domain U. While a linear order can be maintained with binary relations [9, 24], it is often available as a basic operation in relational calculus. In the case when one can use a linear order in relational calculus (first-order) formulae, we refer to relational calculus with a built-in linear order. It turns out that the previous corollary extends to it:

Corollary 4 *It is impossible to incrementally maintain, in relational calculus with built-in linear order, the transitive closure query under deletions and the same-generation query under either deletions or insertions, even in the presence of unary auxiliary relations.*

Proof. We follow the proof of Theorem 2 and observe that every query that we construct in order to contradict locality, is order-invariant, that is, its result is independent of a particular interpretation of the linear order. Thus, the proof of Theorem 2 applies verbatim, since [14] shows that order-invariant first-order queries are local. $\square$

We now turn our attention to plain SQL, that is, an extension of relational calculus with grouping and aggregation, described briefly at the end of section 3. We assume that there are two base types: type b, whose domain is U, and type $\mathbb{Q}$ of rational numbers. When we talk about graph queries, we mean queries of the type $\{b \times b\} \to \{b \times b\}$, that is, queries that take a finite graph over U and return another finite graph over U. We then can show:

Corollary 5 *It is impossible to incrementally maintain, in plain SQL, the transitive closure query under deletions and the same-generation query under either deletions or insertions, even in the presence of non-numerical unary auxiliary relations.*

Proof. Again, we follow the proof of Theorem 2. Every query that we construct in order to contradict locality, is relational: that is, its input has relations interpreted over U but not $\mathbb{Q}$ (due to the restriction that auxiliary relations are nonnumerical). The result now follows from the fact that such queries are local [18]. $\square$

Note that if we consider numerical input relations, or even a built-in linear order on the non-numerical base type, and no auxiliary relations in the setting of plain SQL, then proving bounds not only on incremental maintenance but even on the expressive power is extremely hard. This follows from the fact that the language can then express every query whose data complexity is in uniform TC^0 [18, 22], which so far has not been separated even from NP. On the other hand, in the incremental maintenance framework in which one starts with the empty database, it is possible to maintain, in plain SQL, every query whose data complexity is in the polynomial hierarchy, using a built-in linear order and unary auxiliary relations [26]. The result of [26] assumes the setting in which there is no *a priori* bound on the number of elements that can be stored in a database (e.g., the number of nodes of graphs). It was recently shown in [20] that if the number of nodes of graphs is fixed in advance, then transitive closure can be incrementally maintained in TC^0. It is still unknown whether TC^0 can be replaced by a smaller complexity class, e.g., first-order queries.

5 Conclusion

The primary objective of this note was to investigate general properties of query languages that render the unmaintainability of certain recursive views. The property we focused on is locality. It is known that proving bounds on incremental recomputation with auxiliary relations of arity 2 and higher is extremely hard [9], so we considered auxiliary data of arity at most 1. We showed that locality implies unmaintainability of two typical recursive queries: transitive closure, and same generation, even in the presence of unary auxiliary relations. The results apply to relational calculus, relational calculus with built-in order, and plain SQL.

Acknowledgement The authors wish to thank Neil Immerman for his comments on the paper.

References

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

[2] J. Blakeley, P.-Å. Larson, and F. W. Tompa. Efficiently updating materialized views. In *Proceedings of the 1986 ACM-SIGMOD International Conference on Management of Data*, ACM Press, 1986, pages 61–71.

[3] P. Buneman, S. Naqvi, V. Tannen and L. Wong. Principles of programming with complex objects and collection types. *Theoretical Computer Science*, 149(1):3–48, 1995.

[4] G. Dong and J. Su. Incremental and decremental evaluation of transitive closure by first-order queries. *Information and Computation*; 120(1):101–106, July 1995.

[5] G. Dong and R. Topor. Incremental evaluation of datalog queries. In *LNCS 646: Proceedings of 4th International Conference on Database Theory, Berlin, Germany, October 1992*, pages 282–296.

[6] G. Dong and J. Su. Arity bounds in first-order incremental evaluation and definition of polynomial time database queries. *JCSS*, 57 (1998), 289–308.

[7] G. Dong and J. Su. Deterministic FOIES are strictly weaker. *Annals of Mathematics and Artificial Intelligence*, 19(1-2):127–146, 1997.

[8] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory.* Springer Verlag, 1995.

[9] K. Etessami. Dynamic tree isomorphism via first-order updates. In *PODS'98*, pages 235–243.

[10] R. Fagin, L. Stockmeyer, M. Vardi, On monadic NP vs monadic co-NP, *Information and Computation*, 120 (1994), 78–92.

[11] H. Gaifman. On local and non-local properties. In *Proceedings of the Herbrand Symposium, Logic Colloquium '81*, pages 105–135. North Holland, 1982.

[12] 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.

[13] T. Griffin, L. Libkin, and H. Trickey. An improved algorithm for incremental recomputation of active relational expressions. *IEEE Transactions on Knowledge and Data Engineering*, 9 (1997), 508–511.

[14] M. Grohe and T. Schwentick. Locality of order-invariant first-order formulas. *ACM Transactions on Computational Logic*, 1 (2000), 112–130.

[15] A. Gupta and I. S. Mumick. Maintenance of Materialized Views: Problems, Techniques, and Applications. *IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Data Warehousing*, 18(2):3–19, June 1995.

[16] A. Gupta, I. S. Mumick, and V. S. Subrahmanian. Maintaining views incrementally. In *Proceedings of the 1993 ACM-SIGMOD International Conference on Management of Data*, ACM Press, 1993, pages 157–166.

[17] L. Hella, L. Libkin and J. Nurmonen. Notions of locality and their logical characterizations over finite models. *Journal of Symbolic Logic*, 64 (1999), 1751–1773.

[18] L. Hella, L. Libkin, J. Nurmonen and L. Wong. Logics with aggregate operators. *Journal of the ACM*, to appear. Extended abstract in *LICS'99*, pages 35–44.

[19] M. R. Henzinger and V. King. Fully dynamic biconnectivity and transitive closure. In *Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science*, pages 664–672, 1995.

[20] W. Hesse. The dynamic complexity of transitive closure is in DynTC^0. In *Proceedings of International Conference on Database Theory*, Springer LNCS 1973, 2001, pages 234–247.

[21] N. Immerman. Languages that capture complexity classes. *SIAM Journal of Computing*, 16:760–778, 1987.

[22] N. Immerman. *Descriptive Complexity.* Springer-Verlag, 1999.

[23] L. Libkin and L. Wong. Query languages for bags and aggregate functions. *JCSS*, 55 (1997), 241–272.

[24] L. Libkin and L. Wong. Incremental recomputation of recursive queries with nested sets and aggregate functions. In *Proc. Database Programming Languages 1997*, Springer LNCS 1369, pages 222–238.

[25] L. Libkin and L. Wong. On the power of aggregation in relational query languages. In *Proc. Database Programming Languages 1997*, Springer LNCS 1369, pages 260–280.

[26] L. Libkin and L. Wong. On the power of incremental evaluation in SQL-like languages. In *Proc. Database Programming Languages 1999*, Springer LNCS 1949, pages 17–30.

[27] P. Miltersen, S. Subramanian, J. S. Vitter and R. Tamassia. Complexity models for incremental computation. *Theoretical Computer Science* 130 (1994), 203–236.

[28] S. Patnaik and N. Immerman. Dyn-FO: A parallel dynamic complexity class. *JCSS*, 55 (1997), 199–209.

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

[30] D. Quass. Maintenance expressions for views with aggregation. In *Proceedings of the SIGMOD'96 Workshop on Materialized Views*, pages 110–118.