1995

Nonrecursive Incremental Evaluation of Datalog Queries

Guozhu Dong

Wright State University - Main Campus, guozhu.dong@wright.edu

Jianwen Su

Rodney Topor

Follow this and additional works at: <https://corescholar.libraries.wright.edu/knoesis>

Part of the [Bioinformatics Commons](#), [Communication Technology and New Media Commons](#), [Databases and Information Systems Commons](#), [OS and Networks Commons](#), and the [Science and Technology Studies Commons](#)

Repository Citation

Dong, G., Su, J., & Topor, R. (1995). Nonrecursive Incremental Evaluation of Datalog Queries. *Annals of Mathematics and Artificial Intelligence*, 14 (2-4), 187-223.
<https://corescholar.libraries.wright.edu/knoesis/366>

This Article is brought to you for free and open access by the The Ohio Center of Excellence in Knowledge-Enabled Computing (Kno.e.sis) at CORE Scholar. It has been accepted for inclusion in Kno.e.sis Publications by an authorized administrator of CORE Scholar. For more information, please contact library-corescholar@wright.edu.

Nonrecursive Incremental Evaluation of Datalog Queries*

Guozhu Dong

Department of Computer Science
University of Melbourne
Parkville, Vic. 3052, Australia

Jianwen Su

Department of Computer Science
University of California
Santa Barbara, CA 93106 USA

Rodney Topor

School of Computing and Information Technology
Griffith University
Nathan, Qld 4111, Australia

Computer Science TR 93/3, University of Melbourne
To appear in Annals of Mathematics and Artificial Intelligence
Revised: May 1994

---

*Parts of the results in this paper appeared as extended abstracts in *Proceedings of the 1992 International Conference on Database Theory* (LNCS 646, Springer-Verlag), and in *Proceedings of the 1993 International Workshop on Database Programming Languages* (Workshops in Computing, Springer-Verlag).

Guozhu Dong gratefully acknowledges support of Australian Research Council through research grants and the Centre for Intelligent Decision Systems.

Work by Jianwen Su was supported in part by NSF grants IRI-9109520 and IRI-9117094.

Abstract

We consider the problem of repeatedly evaluating the same (computationally expensive) query to a database that is being updated between successive query requests. In this situation, it should be possible to use the difference between successive database states and the answer to the query in one state to reduce the cost of evaluating the query in the next state. We use nonrecursive Datalog (which are unions of conjunctive queries) to compute the differences, and call this process “incremental query evaluation using conjunctive queries.”

After formalizing the notion of incremental query evaluation using conjunctive queries, we give an algorithm that constructs, for each regular chain query (including transitive closure as a special case), a nonrecursive Datalog program to compute the difference between the answer after an update and the answer before the update. We then extend this result to weakly regular queries, which are regular chain programs augmented with conjunctive queries having the so-called cartesian-closed increment property, and to the case of unbounded-set insertions where the sets are binary cartesian products. Finally, we show that the class of conjunctive queries with the cartesian-closed increment property is decidable.

Key words: Datalog query, database, optimization, on-line evaluation, incremental evaluation, view maintenance, conjunctive query, bounded query, program transformation, cartesian-closed increment, inductive definition.

1 Introduction

Relational query languages have limited power since they cannot express recursive queries such as transitive closure queries [5]. Datalog provides a way of incorporating recursion into a query language. However, it also raises the complexity of query evaluation. The problem of efficiently computing Datalog queries has attracted a great deal of attention in the database and logic programming communities e.g., [6, 8, 12, 14, 17, 18, 24, 25, 27].

In this paper, we consider the problem of repeatedly evaluating the same (computationally expensive) Datalog query to a database that is being updated between successive query requests. In this case, it should be possible to use the difference between successive database states and the answer to the query in one state to reduce the cost of evaluating the query in the next state. We use nonrecursive Datalog (which are unions of conjunctive queries) to compute the differences, and call this process “incremental query evaluation using conjunctive queries.”

This optimization approach is analogous to the incremental checking of integrity constraint satisfaction by using (i) database updates and (ii) the fact that the integrity constraints were satisfied prior to the updates [9, 28, 29]. Our task is closely related to the problem of efficiently updating the standard model [2] of a definite or more generally stratified database [4, 25]. Our approach is very useful in maintaining materialized views upon updates. It is also closely related to the problem of partially evaluating definite logic programs [27]. Finally, when restricted to standard transitive closure programs, our task can be viewed as solving the incremental transitive closure computation problem for graphs [10, 14, 20, 21]. More detailed comparison will be given in Section 6.

In general, all these optimization approaches store extra information to reduce the time required for subsequent computations. In our case, we store the answer to the query in one database state (and possibly additional derived facts) to reduce the cost of evaluating the query in subsequent database states.

Informally, the idea of incremental query evaluation using conjunctive queries is as follows. Let Q be a Datalog query, D an initial database state, $Q(D)$ the answer to query Q in database state D , and Δ a set of at most k facts to be inserted (k a fixed integer). Then our approach is to store $Q(D)$ (and possibly additional derived facts), and compute the answer to Q in the new database state $D \cup \Delta$ by using a nonrecursive “incremental query” Q' satisfying $Q'(Q(D) \cup \Delta) \cup Q(D) = Q(D \cup \Delta)$. Using incremental evaluation, the task of evaluating Q is replaced by the task of evaluating the computationally cheaper Q' .

Nonrecursive Datalog programs are effectively unions of conjunctive queries, which permit efficient computation methods [31] and are more suitable for parallel computation than recursive Datalog and recursive algorithms embedding relational operations. For database applications, we believe that nonrecursive Datalog programs are much better than recursive graph algorithms using elaborate data structures even though the latter have lower sequential complexity. Indeed, a nonrecursive Datalog program can be evaluated by a bounded number of relational join operations, whereas a recursive algorithm needs an unbounded number of iterations. Furthermore, nonrecursive Datalog programs are a subset of relational queries and are thus readily programmable in common database programming languages, whereas recursive algorithms

with elaborate data structures are not easily expressible in most such languages.

Queries allowing incremental evaluation using conjunctive queries form a strict generalization of bounded Datalog queries.

Besides introducing the idea of incremental evaluation using conjunctive queries, one of our main contributions is an algorithm to provide conjunctive queries for incrementally evaluating regular chain queries (which are associated with chain Datalog programs). We also extend the result on regular chain queries to weakly regular queries where the regular chain programs are *augmented* with conjunctive queries having the so-called “cartesian-closed increment” property, and to the case of unbounded-set insertions where the sets are “cartesian closed”. We show that the cartesian-closed increment property is decidable for nonrecursive programs.

The remainder of the paper is organized as follows. Section 2 defines the above concepts in more detail and discusses some elementary properties of incremental evaluation using conjunctive queries. Section 3 presents our incremental query construction algorithm for regular chain queries and the proof of its properties, and Section 4 describes our results for the extended cases. Section 5 presents the decidability results on the cartesian-closed increment property and weakly regular queries. Section 6 compares our results with related work, and Section 7 concludes and suggests some directions for future research.

2 Incremental Evaluation System Using Conjunctive Queries

After briefly reviewing definitions of queries and answers, we introduce the central concepts of the paper, i.e., incremental evaluation (system) using conjunctive queries. We shall illustrate the concepts by several examples. We consider subclasses of such systems with special forms and establish a relationship between predicate boundedness and these subclasses. It turns out that the existence of such systems in these subclasses is undecidable. We also show that there cannot be such systems in these subclasses for queries such as “or gate” and “same generation,” although these subclasses all include the transitive closure query. But it remains open if there are Datalog queries that do not permit such incremental evaluation.

In this section, we limit the insertions to be singletons. The results of this paper with respect to such insertions can be easily extended to the case where the number of inserted tuples is bounded by a fixed integer. Later in Section 4 we will discuss incremental evaluation using conjunctive queries with respect to unbounded insertions with a certain property. However, the results cannot be extended to the case where an arbitrary set of tuples is inserted, since for example the transitive closure query (on the inserted set) cannot be computed by a nonrecursive program.

Note that nonrecursive programs define unions of conjunctive queries [11]. Such queries allow very efficient computations, and have received extensive attention in the literature [3, 31]. As we shall argue later, queries permitting incremental evaluation using conjunctive queries strictly generalize queries computable by nonrecursive programs.

We assume familiarity with the relational databases and the Datalog language [31].

We assume the existence of three pairwise disjoint infinite sets of *constants*, *variables*, and *predicates*. Predicates are divided into *extensional* (or *EDB*) predicates and *intensional* (or *IDB*) predicates. Built-in predicates such as equality are disallowed. Each predicate has a positive *arity*. A *term* is either a variable or a constant. An *atom* is a formula of the form $q(t_1, \dots, t_k)$, where q is a predicate and $t_1, \dots, t_k$ are terms. A *fact* is an atom whose terms are all constants. A *database* is a finite set of facts over EDB predicates.

A *Datalog program* or simply *program* is a finite set of rules of the form $A \leftarrow A_1, \dots, A_n$, where $n \geq 1$, A and $A_1, \dots, A_n$ are atoms and each variable occurring in A occurs in some A_i . Only IDB predicates can occur in the heads of such rules. A *Datalog query* Q is a pair (Π, p) , where Π is a Datalog program and p is a (*query*) predicate symbol.

The result $\Pi(D)$ of applying a Datalog program Π to a set D of facts is the set of IDB facts in the least (Herbrand) model for $\Pi \cup D$ or, equivalently, the set of IDB facts that are logical consequences of $\Pi \cup D$ [32]. The *answer* $Q(D)$ to a query $Q = (\Pi, p)$ on a database D is simply the set¹ of facts $\Pi(D)|_p$.

Given a program Π and a rule $A \leftarrow A_1, \dots, A_n$ in Π , the predicate symbol in A is said to *depend on* q in Π where q is a predicate symbol occurring in $A_1, \dots, A_n$; and a predicate symbol p is called *recursive* in Π if p transitively depends on itself in Π , i.e., there is a sequence $p_1, \dots, p_k$ ($k \geq 1$) of predicate symbols occurring in Π such that $p_1 = p = p_k$ and p_i depends on p_{i+1} in Π for all $i \in [1..k - 1]$. A program is called *nonrecursive* if it does not contain any predicate symbol that is recursive in the program, and is called *recursive* otherwise.

To differentiate old facts in the old state from new facts in a current state, for each predicate q , we shall use q° (o for old) as a new predicate to represent facts over q computed or stored in the previous database state. For each set I of facts, let I° be the set of facts obtained from I by replacing each predicate q with q° . These old facts will then be used for computing the new facts in the query answer after the updates. An illustration is given in Example 2.1.

We now introduce the central notions of the paper, namely incremental evaluation (system) using conjunctive queries.

Definition Let $Q = (\Pi, p)$ be a Datalog query. An *incremental evaluation system using conjunctive queries* (or *IEC*) for Q is a triple $\langle \Pi_p, S, \Pi_\delta \rangle$, where:

- Π_p is a (possibly recursive) Datalog program, called the *initial program*, such that $\Pi_p(D)|_p = \Pi(D)|_p$ for each database D ;
- S is a set of IDB predicate symbols containing p ; and
- Π_δ is a nonrecursive Datalog program, called the *incremental program*, such that $\Pi_p(D \cup \Delta)|_S = \Pi_\delta([\Pi_p(D)|_S \cup D]^\circ \cup \Delta)|_S \cup \Pi_p(D)|_S$ for each database D and each set Δ consisting of one EDB fact.

We say Q *permits* incremental evaluation using conjunctive queries if there is such a system for Q .

---

¹For each set S of predicate symbols, the restriction of a set I of facts to those with predicate in S is denoted $I|_S$. We also write $I|_p$ for $I|_{\{p\}}$.

As indicated in the introduction, we store $\Pi_p(D)|_S$ to reduce the cost of evaluating $\Pi_p(D \cup \Delta)|_S$. This is intended to avoid recomputing the facts in $\Pi_p(D)|_S$ after inserting the fact in Δ . Note that $\Pi_p(D \cup \Delta)|_p = (\Pi_p(D \cup \Delta)|_S)|_p$.

The benefits that can be achieved by incremental evaluation depend mostly on the choice of Π_δ , the program used to compute the new facts in the answer to the query in the updated database. To concentrate on the benefit of efficiency, we require Π_δ to be nonrecursive.

The following example is used to illustrate the above concepts.

Example 2.1 Consider the query $Q = (\Pi, path)$, where Π is the program

$$\begin{aligned} path(x, z) &\leftarrow edge(x, z) \\ path(x, z) &\leftarrow edge(x, y), path(y, z) \end{aligned}$$

Let $\Pi_p = \Pi$, $S = \{path\}$, and Π_δ be the program

$$\begin{aligned} path(x, z) &\leftarrow edge(x, z) \\ path(x, z) &\leftarrow edge(x, y), path^o(y, z) \\ path(x, z) &\leftarrow path^o(x, y), edge(y, z) \\ path(x, z) &\leftarrow path^o(x, y_1), edge(y_1, y_2), path^o(y_2, z) \end{aligned}$$

To illustrate incremental evaluation, suppose $D = \{edge(1, 2), edge(2, 3), edge(4, 5), edge(5, 6)\}$, and $\Delta = \{edge(3, 4)\}$. Then $\Pi_p(D) = \{path(1, 2), path(2, 3), path(1, 3), path(4, 5), path(5, 6), path(4, 6)\}$. To compute $\Pi_p(D \cup \Delta)$ from $\Pi_p(D)$ using Π_δ , the facts in $\Pi_p(D) \cup D$ are marked with the superscript o to indicate that they were facts in the state before inserting the fact in Δ ; the predicate $edge$ (resp., $path$) in Π_δ denotes the additional set of facts that are added (resp., derived) for $edge$ (resp. $path$). Thus, the additional fact for $edge$ is $\{edge(3, 4)\}$, and the additional facts for $path$ are $\{path(i, j) \mid i \in [1..3] \text{ and } j \in [4..6]\}$.

It will be seen from Lemma 3.6 below that $\langle \Pi_p, S, \Pi_\delta \rangle$ is an IEC for Q . Intuitively, in computing the new path after an edge is added, one only needs to do four joins (by directly using Π_δ ; it could be reduced to three by using the results of a previous join). Thus we have transformed the computation of a recursive program into the computation of a nonrecursive program (with the help of stored results).

We can transform Π_δ into a more efficient program by instantiating Π_δ with the specific fact $edge(a, b)$ in Δ . The resulting rules are: $path(a, z) \leftarrow path^o(b, z)$; $path(x, b) \leftarrow path^o(x, a)$ and $path(x, z) \leftarrow path^o(x, a), path^o(b, z)$. This technique reduces the number of joins needed by the incremental program from four to one, and reduces the number of tuples accessed in the joins. This technique also applies to other examples described below. We will discuss the complexity of more general IEC in Section 3. $\square$

Example 2.2 We now compare our incremental method with the semi-naive evaluation method [6, 31], by considering their computations of the transitive closure query Q after $edge(3, 4)$ is added to the database D in Example 2.1. To make the comparison fair for the semi-naive method, we assume that the semi-naive

method also starts with $path^\circ$ available. We use Δ to denote the relation containing the new $edge$, and use $\delta path$ to contain the new facts derived from each iteration. The initialization for $\delta path$ is tricky: If we initialize $\delta path$ to Δ , then semi-naive will not produce the desired fixpoint (e.g., $(3, 5)$ cannot be derived). If we initialize $\delta path$ to $path^\circ$, then semi-naive will not produce the desired fixpoint either (e.g., $(1, 4)$ cannot be produced). We choose to initialize $\delta path$ to be $\Delta \cup path^\circ$. Hence we assume that the semi-naive algorithm is as follows:

```

 $\delta path := \Delta \cup path^\circ;$ 
while  $\delta path$  is not empty do begin
   $\delta path := \pi_{1,3}((edge^\circ \cup \Delta) \bowtie_{2=1} \delta path);$ 
   $\delta path := \delta path - path;$ 
   $path := \delta path \cup path$ 
end

```

Here the semi-naive method will use three iterations (more iterations will be needed for longer paths). The semi-naive evaluation proceeds as follows:

- $\delta path$ is initialized to: $\{(1, 2), (2, 3), (1, 3), (3, 4), (4, 5), (5, 6), (4, 6)\}$.
- In iteration one, $\delta path$ first becomes $\{(1, 3), (2, 4), (3, 5), (3, 6), (4, 6)\}$, and after removing old facts it becomes $\{(2, 4), (3, 5), (3, 6)\}$.
- In iteration two, $\delta path$ becomes $\{(1, 4), (2, 5), (2, 6)\}$, and no old fact is derived.
- In iteration three, $\delta path$ becomes $\{(1, 5), (1, 6)\}$, and that concludes the computation.

This is more expensive in two aspects: (1) more joins are needed, (2) joins produce larger relations with more duplicate facts (2 such facts here). In contrast, using our improved method, we need just one join, and no old facts are produced. Even with our initial (nonimproved) method, we need three joins, and no old facts are produced. Since the (new) $edge$ relation contains exactly one fact, the joins corresponding to the second and the third rules of Π_δ are actually like selections, and thus only the join corresponding to the last rule is expensive.

For this example program at least, our incremental method is superior to the semi-naive method. The reasons are: First, our method produces only facts with new derivations (using the inserted edge); all these derivations will happen in semi-naive evaluation. Second, semi-naive evaluation does more derivations, at least in iteration one, because of its large initialization. $\square$

In the remainder of the section, we discuss the equivalence between “predicate boundedness” and the existence of IEC of a certain form. Predicate boundedness is a special case of boundedness [22, 18]. A Datalog program Π is called p -bounded, where p is a predicate, if there is a nonrecursive Datalog program Π' such that $\Pi'(D)|_p = \Pi(D)|_p$ for each database D .

Lemma 2.3 Suppose Π is an arbitrary Datalog program and p is an arbitrary IDB predicate occurring in Π . Let Π' be the program constructed by adding $q_0(y)$ to the body of each rule in Π , where q_0 is a new unary EDB predicate and y is a new variable. Then Π is p -bounded iff there is an IEC of the form $\langle \Pi_p, \{p\}, \Pi_\delta \rangle$ for (Π', p) .

Proof If Π is p -bounded, then so is Π' . By appropriately adding the superscript o to rules in Π' we can obtain a nonrecursive program Π_δ such that $\langle \Pi', \{p\}, \Pi_\delta \rangle$ is an IEC for (Π', p) .

Conversely, suppose $\langle \Pi_p, \{p\}, \Pi_\delta \rangle$ is an IEC for (Π', p) . Then, for each set D of facts over EDB predicates occurring in Π and each set Δ consisting of one fact over q_0 , $\Pi_p(D)|_p = \Pi'(D)|_p = \emptyset$, and hence $\Pi'(D \cup \Delta)|_p = \Pi_p(D \cup \Delta)|_p = \Pi_\delta((\Pi_p(D)|_p \cup D)^o \cup \Delta)|_p \cup \Pi_p(D)|_p = \Pi_\delta(D^o \cup \Delta)|_p$. We now construct from Π_δ a nonrecursive Datalog program equivalent to Π with respect to p . Let Π_1 consist of rules in Π_δ that only contains atoms over q_0 and atoms over predicates of the form q^0 where q is an EDB predicate occurring in Π . Since in $D^o \cup \Delta$ there are only facts over q_0 and atoms over predicates of the form q^0 where q is an EDB predicate occurring in Π , we have $\Pi_1(D^o \cup \Delta)|_p = \Pi_\delta(D^o \cup \Delta)|_p$. Clearly, Π_1 is also nonrecursive. Without loss of generality, we can assume that p is the only IDB predicate in Π_1 . Let Π_2 be obtained from Π_1 by removing the superscript o . Then Π_2 is nonrecursive and $\Pi_2(D \cup \Delta)|_p = \Pi_\delta(D^o \cup \Delta)|_p$. Since $\Pi'(D)|_p = \emptyset$, it is easily seen that q_0 must occur in every rule in Π_2 . Let Π_3 be obtained by, for each rule r in Π_2 , (i) choosing a variable (say x_r) that occurs in a q_0 atom in r , (ii) changing all variables that occur in any q_0 atom in r to x_r , and (iii) removing all atoms over q_0 . Then Π_3 is nonrecursive. It can be easily verified that $\Pi_3(D)|_p = \Pi(D)|_p$. Hence Π is p -bounded. $\square$

Note that Example 2.1 showed that there are queries which have IEC but which are not predicate bounded. Combining this with Lemma 2.3, we see that queries having IEC strictly generalizes bounded Datalog queries.

Since it is undecidable whether an arbitrary Datalog program is predicate bounded [18] (even when the program has only one IDB predicate and has only binary IDB predicates [33]), Lemma 2.3 implies the following result:

Theorem 2.4 It is undecidable for each arbitrary query (Π, p) whether there is an IEC of the form $\langle \Pi_p, \{p\}, \Pi_\delta \rangle$. This holds even for Π having only one IDB predicate and/or having only binary IDB predicates. $\square$

The following result, used twice below, says that the existence of IEC with three special forms are equivalent for Datalog queries (Π, p) where p is the only IDB predicate occurring in Π .

Proposition 2.5 For each Datalog query (Π, p) where p is the only IDB predicate occurring in Π , the following three conditions are equivalent:

1. There is an IEC of the form $\langle \Pi, S, \Pi_\delta \rangle$.
2. There is an IEC of the form $\langle \Pi, \{p\}, \Pi_\delta \rangle$.

3. There is an IEC of the form $\langle \Pi_p, \{p\}, \Pi_\delta \rangle$.

Proof Clearly (2) implies (1) and (3). It suffices to establish that (1) implies (2) and (3) implies (2).

To verify that (1) implies (2), suppose (1) holds. For arbitrary database D and arbitrary set Δ of one EDB fact, since p is the only predicate symbol occurring in Π , $\Pi(D)|_S = \Pi(D)|_p$, and thus $\Pi(D \cup \Delta)|_p = \Pi(D \cup \Delta)|_S = \Pi_\delta((\Pi(D)|_S \cup D)^o \cup \Delta)|_S \cup \Pi(D)|_S = \Pi_\delta((\Pi(D)|_p \cup D)^o \cup \Delta)|_p \cup \Pi(D)|_p$. Therefore $\langle \Pi, \{p\}, \Pi_\delta \rangle$ is an IEC for (Π, p) .

To verify that (3) implies (2), suppose (3) holds. For arbitrary database D and arbitrary set Δ of one EDB fact, $\Pi(D)|_p = \Pi_p(D)|_p$, and $\Pi(D \cup \Delta)|_p = \Pi_p(D \cup \Delta)|_p = \Pi_\delta((\Pi_p(D)|_p \cup D)^o \cup \Delta) \cup \Pi_p(D)|_p = \Pi_\delta((\Pi(D)|_p \cup D)^o \cup \Delta) \cup \Pi(D)|_p$. Therefore $\langle \Pi, \{p\}, \Pi_\delta \rangle$ is an IEC for (Π, p) . $\square$

We now give an IEC for a Datalog query associated with a Datalog program that is not a chain program. This example will also be used to show the necessity of some subtle conditions in the definition of IEC.

Example 2.6 Consider the query $Q = (\Pi, p)$, where Π is the following program that represents the propagation of signals p on wires x, y, z through a network of logical OR gates s (illustrated below) with inputs q .

Figure 1: The Or-Gate Program

For instance, $\Pi(\{s(1, 2, 3), s(2, 4, 5), s(3, 5, 6), q(4), q(6)\}) = \{p(1), p(2), p(3), p(4), p(6)\}$.

By using a program $\Pi_p \neq \Pi$ and storing derived facts (for a new predicate) in addition to the derived facts for p , we can construct an IEC for Q . Indeed, let Π_p be the program:

$$\begin{array}{ll}
 t(x, y) \leftarrow s(x, y, z) & p(x) \leftarrow q(x) \\
 t(x, z) \leftarrow s(x, y, z) & p(x) \leftarrow t(x, y), q(y) \\
 t(x, z) \leftarrow t(x, y), t(y, z) &
 \end{array}$$

and $S = \{p, t\}$ (not simply $\{p\}$). Here, $t(x, y)$ is true if x is “on” whenever y is “on.” Let Π_δ be the following program:

$$\begin{array}{ll}
p(x) \leftarrow q(x) & p(x) \leftarrow t^o(x, y), q(y) \\
p(x) \leftarrow t(x, y), q^o(y) & \\
t_1(x, y) \leftarrow s(x, y, z) & t_1(x, z) \leftarrow s(x, y, z) \\
t(x, y) \leftarrow t_1(x, y) & t(x, z) \leftarrow t^o(x, y), t_1(y, z) \\
t(x, z) \leftarrow t_1(x, y), t^o(y, z) & t(x, z) \leftarrow t^o(x, y_1), t_1(y_1, y_2), t^o(y_2, z)
\end{array}$$

Then it can be verified that $\langle \Pi_p, S, \Pi_\delta \rangle$ is an IEC for Q . Note that, if Δ consists of a fact over q , then one only needs to use the first two rules; if Δ consists of a fact over s , then one only needs to use the other rules.

□

The above example combined with the next result illustrates why it is sometimes necessary to use an initial program Π_p different from Π , and why it is sometimes necessary to store the set of facts $\Pi_p(D)|_S$ instead of the query answer $\Pi(D)|_p$.

Proposition 2.7 For the query Q in Example 2.6, there is no IEC $\langle \Pi_p, S, \Pi_\delta \rangle$ with (i) $\Pi_p = \Pi$ or (ii) with $S = \{p\}$.

Proof By Proposition 2.5, (i) and (ii) are equivalent. Hence it suffices to prove (i). Assume there is an IEC $\langle \Pi, S, \Pi_\delta \rangle$ for Q . To reach a contradiction, we show that the path problem (defined below) of graphs can be solved by nonrecursive queries.

The path problem is: For a given directed graph G and two vertices a, b , decide if there is a path from a to b in G . The problem is a variation of the transitive closure query and it can be shown not expressible by nonrecursive queries using an argument of playing Ehrenfeucht-Fraïssé games similar to the proof for the graph connectivity query in [3].

We map each instance of the path problem $\langle G, a, b \rangle$ to a database D and an insertion Δ such that (i) both D and Δ can be constructed from G and (a, b) using nonrecursive queries (given below), (ii) $\Pi(D)|_S = \emptyset$, and (iii) $p(b) \in \Pi(D \cup \Delta)|_S$ if and only if there is a path from a to b in G . It then follows that $\Pi(D \cup \Delta)|_S = \Pi_\delta((\Pi(D)|_S \cup D)^o \cup \Delta)|_S \cup \Pi(D)|_S = \Pi_\delta(D^o \cup \Delta)|_S$ and, in particular, $\Pi(D \cup \Delta)|_p = \Pi_\delta(D^o \cup \Delta)|_p$. Since Π_δ is a nonrecursive query, and D^o and Δ are constructible from G and (a, b) by nonrecursive queries, it follows that the path problem can be expressed by nonrecursive queries, a contradiction.

We construct D and Δ by letting $D = \{s(c, c', c') \mid (c', c) \in G\}$ and $\Delta = \{q(a)\}$. Obviously, D and Δ are expressible by nonrecursive queries from $\langle G, a, b \rangle$. It is also easy to verify $\Pi(D)|_S = \emptyset$ since there is no q fact in D . Finally we have to show $p(b) \in \Pi(D \cup \Delta)|_S$ if and only if there is a path from a to b in G . Since $\Delta = \{r(a)\}$, $p(a) \in \Pi(D \cup \Delta)|_S$. By the construction of D , there is a path $a = c_0, c_1, \dots, c_n = b$ from a to b in G iff there are constants $a = c_0, c_1, \dots, c_n = b$ such that for each $i \in [1..n]$, $s(c_i, c_{i-1}, c_{i-1}) \in D$, since $p(c_i) \in \Pi(D \cup \Delta)|_S$ by inductively applying the first rule on facts $s(c_i, c_{i-1}, c_{i-1})$ and $p(c_{i-1})$, iff $p(b) \in \Pi(D \cup \Delta)|_S$. □

Example 2.8 We will refer to the same-generation query (Π, sg) several times, where Π is the following program:

$$\begin{aligned} sg(x, x) &\leftarrow person(x) \\ sg(x, y) &\leftarrow parent(x, z_1), sg(z_1, z_2), parent(y, z_2) \end{aligned}$$

□

We can also reduce the path-existence problem to the same-generation query as follows: For each graph G and vertices a and b , let $D = \{parent(c, c') \mid (c, c') \text{ is an edge of } G\}$ and $\Delta = \{s(b)\}$. Then there is a path from a to b iff $sg(a, a)$ belongs to $\Pi(D \cup \Delta)$. Using a proof similar to that of Proposition 2.7, we have the following:

Proposition 2.9 For the same-generation query (Π, sg) , there is no IEC $\langle \Pi_p, S, \Pi_\delta \rangle$ with (i) $\Pi_p = \Pi$ or (ii) with $S = \{sg\}$. □

3 Regular Chain Queries

In this section we consider the incremental evaluation of queries in the class of “regular chain Datalog programs.” The main result of the section (Theorem 3.1) states that each regular chain query has an IEC. As the primary step of the proof, we present an algorithm for constructing an IEC for each regular query. We also discuss some complexity issues associated with the IEC constructed by our algorithm. Extensions of the regular chain queries are studied in the next section.

We start by defining the class of “regular chain queries.”

A *chain Datalog program* is a finite set of *chain rules* of the form

$$q(x, z) \leftarrow q_1(x, y_1), q_2(y_1, y_2), \dots, q_k(y_{k-1}, z) \quad (1)$$

where $k \geq 1$ and $x, y_1, \dots, y_{k-1}$, and z are distinct variables. Note that chain Datalog programs contain only variables and binary predicate symbols.

Chain Datalog programs and generalizations allow special optimization techniques. Indeed, several papers have considered efficiency issues of such programs [1, 12, 13]. The current paper also explores such possibilities.

It is well known that, for each chain Datalog program Π , the query (Π, p) can be associated with a context-free grammar² G constructed as follows. The terminal (resp., nonterminal) symbols are the EDB (resp., IDB) predicates; the start nonterminal is the query predicate p ; and for each rule in Π of the form (1) there is a production of the form $q \rightarrow q_1 q_2 \dots q_k$.

---

²We assume familiarity with the elements of the formal language theory.

Definition A Datalog query (Π, p) is called *regular* if Π is a chain Datalog program and the context-free grammar associated with it is right-linear.³

The standard edge-path query $(\Pi, path)$ given in Example 2.1 is regular, whereas the standard same-generation query in Example 2.8 is not.

Our main result of this section is now stated.

Theorem 3.1 Each regular chain query has an IEC. □

We first provide an auxiliary notion and establish a key lemma (Lemma 3.2). The algorithm (Algorithm 3.4) constructing an incremental evaluation system using conjunctive queries for each regular chain query is then presented, along with its correctness proof (Lemma 3.6). Theorem 3.1 follows immediately from Lemma 3.6.

We regard a database D over a set of binary EDB predicates as a directed graph whose vertices are constants and edges are labelled by EDB predicates such that there is an edge labelled p from a to b in the graph if and only if $p(a, b) \in D$. Let $\mathcal{L}$ be an ϵ -free⁴ regular language over the alphabet of binary EDB predicates. For each directed graph D , an $\mathcal{L}$ -path from c_0 to c_k is an expression of the form “ $q_1(c_0, c_1)q_2(c_1, c_2) \cdots q_k(c_{k-1}, c_k)$ ” where each $q_i(c_{i-1}, c_i)$ ($i \in [1..(k-1)]$) is in D and $q_1 \cdots q_k \in \mathcal{L}$. Suppose E is a regular expression. We denote by $\#_q(E)$ the number of occurrences of the symbol q in E and by $L(E)$ the language of E . For example, $L(edge^+) = \{edge^i \mid i \geq 1\}$ and $\#_{edge}(edge^+) = 1$; for $D = \{edge(1, 2), edge(2, 3)\}$, $edge(1, 2)edge(2, 3)$ is an $L(edge^+)$ -path from 1 to 3.

Lemma 3.2 Let D be a labelled directed graph, $q(a_1, a_2)$ a labelled edge in D , E a $\{*, \epsilon, \emptyset\}$ -free regular expression, and b_1 and b_2 two vertices. If there is an $L(E)$ -path in D from b_1 to b_2 , then there is such a path in which $q(a_1, a_2)$ occurs at most $\#_q(E)$ times.

Proof Let Λ be the set of labels appearing in E and $n = \sum_{x \in \Lambda} \#_x(E)$. Then $n \geq 1$. For each $i \in [1..n]$, we replace the i th occurrence of symbols in E from Λ by i . Let $\hat{E}$ denote the resulting regular expression. Clearly, no terminal symbol occurs in $\hat{E}$ twice. Let f be the mapping from $[1..n]$ to Λ such that $f(\hat{E}) = E$. Then f is an homomorphism and it follows that $f(L(\hat{E})) = L(E)$.

Suppose there exists an $L(E)$ -path $q_1(c_0, c_1) \cdots q_k(c_{k-1}, c_k)$ from b_1 to b_2 . Let m be the number of occurrences of $q(a_1, a_2)$ in this path.

It suffices to assume $m > \#_q(E)$. Since $q_1 \cdots q_k$ is in $L(E)$ and $f(L(\hat{E})) = L(E)$, there exists a word $i_1 \cdots i_k$ in $L(\hat{E})$ such that $f(i_1 \cdots i_k) = q_1 \cdots q_k$. Since $m > \#_q(E)$, there exist $\rho, \rho' \in [1..k]$ such that $\rho < \rho'$, $i_\rho = i_{\rho'}$, and $q_\rho(c_{\rho-1}, c_\rho) = q_{\rho'}(c_{\rho'-1}, c_{\rho'}) = q(a_1, a_2)$. Intuitively, the two equations mean that $q(a, b)$ appears at the “position” i_ρ in E twice. It can be verified (using an automata-theoretic argument) that $i_1 \cdots i_\rho i_{\rho'+1} \cdots i_k$ is in $L(\hat{E})$. Since $q_1 \cdots q_\rho q_{\rho'+1} \cdots q_k = f(i_1 \cdots i_\rho i_{\rho'+1} \cdots i_k)$, $q_1 \cdots q_\rho q_{\rho'+1} \cdots q_k$

³A grammar is right-linear if the only nonterminal symbol in the right hand side of each production is the rightmost symbol.

⁴ ϵ denotes the empty word.

is in $L(E)$. Since $c_\rho = c_{\rho'} = a_2$, it follows that

$$q_1(c_0, c_1) \cdots q_\rho(c_{\rho-1}, c_\rho) q_{\rho'+1}(c_{\rho'}, c_{\rho'+1}) \cdots q_k(c_{k-1}, c_k)$$

is an $L(E)$ -path from b_1 to b_2 . Clearly, $q(a_1, a_2)$ occurs in this path at most $m - 1$ times. Repeating the above argument, one ultimately obtains a desired $L(E)$ -path. $\square$

Example 3.3 We now illustrate the construction used in the proof of the above lemma. Let $E = (qs \cup q^+t)^+$. Then $n = 4$, $\hat{E} = (12 \cup 3^+4)^+$, and $f(1) = f(3) = q$, $f(2) = s$, and $f(4) = t$.

Suppose the following $L(E)$ -path is in graph D :

$$q(c_0, c_1)q(c_1, c_2)t(c_2, a_1)q(a_1, a_2)s(a_2, a_1)q(a_1, a_2)t(a_2, a_1)q(a_1, a_2)s(a_2, a_5)q(a_5, a_6)t(a_6, a_7)$$

Then the word w in $L(\hat{E})$ such that $f(w) = qqtqsqtqsqt$ is $w = 33412341234$. The edge $q(a_1, a_2)$ occurs three times in the path, and $\#_q(E) = 2$. We observe that the first and third usages of $q(a_1, a_2)$ correspond to the same integer 1 in $\hat{E}$. (The second usage corresponds to integer 3 in $\hat{E}$.) By removing 1234, we get the shorter word 3341234 in $L(\hat{E})$. The corresponding $L(E)$ -path from c_0 to a_7 is

$$q(c_0, c_1)q(c_1, c_2)t(c_2, a_1)q(a_1, a_2)s(a_2, a_5)q(a_5, a_6)t(a_6, a_7).$$

Note that $q(a_1, a_2)$ now occurs only once in this $L(E)$ -path. $\square$

We now present the main algorithm, assuming each increment Δ contains one new fact. A generalization allowing a set of facts is discussed later.

Algorithm 3.4 (IEC)

INPUT: A regular chain query $Q = (\Pi, p)$.

OUTPUT: An IEC $\langle \Pi_p, S, \Pi_\delta \rangle$ for Q .

METHOD:

Step 1: Construct a regular expression E' from Q such that the associated grammar of Q generates $L(E')$.

Step 2: Construct a $\{*, \epsilon, \emptyset\}$ -free regular expression E such that $L(E) = L(E')$.

Step 3: For each regular expression e occurring in E , define a predicate symbol p_e such that $p_t = t$ for each EDB predicate symbol t , $p_E = p$, and p_e is new otherwise. Let Π_p consist of the following rules:

- a. $p_e(x, z) \leftarrow p_{e_1}(x, y_1), p_{e_2}(y_1, y_2), \dots, p_{e_k}(y_{k-1}, z)$, if $e = e_1 \cdots e_k$ ($k \geq 2$).
- b. $p_e(x, z) \leftarrow p_{e_i}(x, z)$ for each $i \in [1..k]$, if $e = e_1 \cup \dots \cup e_k$ ($k \geq 2$).
- c. $p_e(x, z) \leftarrow p_{e_1}(x, z)$ and $p_e(x, z) \leftarrow p_{e_1}(x, y), p_e(y, z)$, if $e = e_1^+$.
- d. $p_E(x, z) \leftarrow t(x, z)$, if $E = t$ for some EDB predicate t .

Let S be the set of all IDB predicate symbols of Π_p .

Step 4: We use the predicate symbols occurring in Π_p together with their “old” versions of the form p_e^o . For each EDB predicate q , let Π_δ^q consist of the following rules:

- a. $p_e(x, z) \leftarrow p_1(x, y_1), p_2(y_1, y_2), \dots, p_k(y_{k-1}, z)$, for each sequence $p_1, \dots, p_k$ such that each $p_i \in \{p_{e_i}, p_{e_i}^o\}$ and there is at least one $i \in [1..k]$ such that (*) $p_i = p_{e_i}$ and (**) q occurs in e_i , if $e = e_1 \dots e_k$ ($k \geq 2$).
- b. $p_e(x, z) \leftarrow p_{e_i}(x, z)$ for each $i \in [1..k]$, if $e = e_1 \cup \dots \cup e_k$ ($k \geq 2$).
- c. $p_e(x, z) \leftarrow p_1(x, y_1) \dots p_k(y_{k-1}, z)$ for each subsequence $p_1 \dots p_k$ of $(p_e^o p_{e_1})^{\#_q(e)} p_e^o$ such that (i) there is at least one j such that $p_j = p_{e_1}$, and (ii) there are no consecutive p_e^o 's, if $e = e_1^+$. (A subsequence of a sequence or word $s_1 s_2 \dots s_k$ is a sequence $s_{i_1} s_{i_2} \dots s_{i_j}$, where $j \geq 1$ and $1 \leq i_1 < i_2 < \dots < i_j \leq k$.)
- d. $p_E(x, z) \leftarrow t(x, z)$, if $E = t$ for some EDB predicate t .

Let Π_δ be $\bigcup_q \Pi_\delta^q$, where the union is over all EDB predicates q of Π . □

The program Π_δ constructed by this algorithm computes the new facts for predicates of the form p_e , by using the old facts for p_e in the state before the fact in Δ was inserted.

Note that the incremental program Π_δ constructed by the above algorithm is nonrecursive. Furthermore, Π_p and Π_δ do not correspond to right-linear grammars in general.

If an increment Δ has more than one fact, then we change $\#_q(e)$ in Step 4 to $m = |\Delta| \max\{\#_q(e) \mid q \text{ is a predicate symbol occurring in } \Delta\}$, where $|\Delta|$ denotes the cardinality of Δ .

Example 2.1 illustrated the construction applied to the standard edge-path query. The following example uses Algorithm 3.4 to construct an IEC for a more involved regular chain query.

Example 3.5 Consider the regular chain query $Q = (\Pi, p)$, where Π is the following program:

$$\begin{aligned}
 p_1(x, y) &\leftarrow q(x, y) \\
 p_1(x, y) &\leftarrow q(x, z), p_1(z, y) \\
 p_2(x, y) &\leftarrow q(x, z), s(z, y) \\
 p_2(x, y) &\leftarrow p_1(x, z), t(z, y) \\
 p(x, y) &\leftarrow p_2(x, y) \\
 p(x, y) &\leftarrow p_2(x, z), p(z, y)
 \end{aligned}$$

Suppose the first two steps of Algorithm 3.4 yield the regular expression $E = (qs \cup q^+t)^+$. Let $e_1 = qs$, $e_2 = q^+$, $e_3 = e_2t$, $e_4 = e_1 \cup e_3$, and $e_5 = e_4^+$. Then $S = \{p_{e_i} \mid 1 \leq i \leq 5\}$, Π_p is the program

$$\begin{aligned}
 p_{e_1}(x, z) &\leftarrow q(x, y), s(y, z) & p_{e_4}(x, z) &\leftarrow p_{e_1}(x, z) \\
 p_{e_2}(x, z) &\leftarrow q(x, z) & p_{e_4}(x, z) &\leftarrow p_{e_3}(x, z) \\
 p_{e_2}(x, z) &\leftarrow q(x, y), p_{e_2}(y, z) & p_{e_5}(x, z) &\leftarrow p_{e_4}(x, z) \\
 p_{e_3}(x, z) &\leftarrow p_{e_2}(x, y), t(y, z) & p_{e_5}(x, z) &\leftarrow p_{e_4}(x, y), p_{e_5}(y, z)
 \end{aligned}$$

and Π_δ^q is the program

$$\begin{array}{ll}
p_{e_1}(x, z) \leftarrow q(x, y), s^o(y, z) & p_{e_3}(x, z) \leftarrow p_{e_2}(x, y), t^o(y, z) \\
p_{e_2}(x, z) \leftarrow q(x, z) & p_{e_4}(x, z) \leftarrow p_{e_1}(x, z) \\
p_{e_2}(x, z) \leftarrow q(x, y), p_{e_2}^o(y, z) & p_{e_4}(x, z) \leftarrow p_{e_3}(x, z) \\
p_{e_2}(x, z) \leftarrow p_{e_2}^o(x, y), q(y, z) & p_{e_5}(x, z) \leftarrow p_{e_4}(x, z) \\
p_{e_2}(x, z) \leftarrow p_{e_2}^o(x, y_1), q(y_1, y_2), p_{e_2}^o(y_2, z) & \text{(10 other rules defining } p_{e_5}) \\
p_{e_5}(x, z) \leftarrow p_{e_5}^o(x, y_1), p_{e_4}(y_1, y_2), p_{e_5}^o(y_2, y_3), p_{e_4}(y_3, y_4), p_{e_5}^o(y_4, z)
\end{array}$$

Note the two rules defining p_{e_5} above correspond to the predicate sequences of p_{e_4} and $p_{e_5}^o p_{e_4} p_{e_5}^o p_{e_4} p_{e_5}^o$. The 10 other rules defining p_{e_5} correspond to the following predicate sequences: $p_{e_5}^o p_{e_4}$, $p_{e_4} p_{e_5}^o$, $p_{e_4} p_{e_4}$, $p_{e_5}^o p_{e_4} p_{e_4}$, $p_{e_4} p_{e_4} p_{e_5}^o$, $p_{e_4} p_{e_5}^o p_{e_4}$, $p_{e_5}^o p_{e_4} p_{e_5}^o$, $p_{e_4} p_{e_5}^o p_{e_4} p_{e_5}^o$, $p_{e_5}^o p_{e_4} p_{e_5}^o p_{e_4}$, and $p_{e_5}^o p_{e_4} p_{e_4} p_{e_5}^o$. Π_δ^s and Π_δ^t can be constructed similarly. $\square$

We now prove the correctness of the construction of Algorithm 3.4.

Lemma 3.6 Algorithm 3.4 is correct, i.e., for each regular chain query $Q = (\Pi, p)$, Algorithm 3.4 produces an IEC for Q .

Proof Suppose $q(c, d)$ is an arbitrary EDB fact and $\Delta = \{q(c, d)\}$. We need to verify the two equations in the definition of an IEC. To this end, let D be an arbitrary database, E the $\{*, \epsilon, \emptyset\}$ -free regular expression constructed in Steps 1 and 2, and Π_p the program constructed in Step 3 of Algorithm 3.4. It is then straightforward to verify that $\Pi_p(D)|_p = \Pi(D)|_p$ holds.

To establish the other equation,

$$\Pi_p(D \cup \Delta)|_S = \Pi_\delta([\Pi_p(D)|_S \cup D]^o \cup \Delta)|_S \cup \Pi_p(D)|_S$$

we first prove that the right hand side is contained in the left hand side. Due to monotonicity, $\Pi_p(D)|_S \subseteq \Pi_p(D \cup \Delta)|_S$. Observe that every (bottom-up) derivation of a fact F from the database $[\Pi_p(D)|_S \cup D]^o \cup \Delta$ using the program Π_δ can be transformed into a derivation of F from $D \cup \Delta$ using Π_p by deriving each atom of the form $p_e^o(\dots)$ from D using Π_p . Hence $\Pi_\delta([\Pi_p(D)|_S \cup D]^o \cup \Delta)|_S \subseteq \Pi_p(D \cup \Delta)|_S$.

To prove the reverse containment, it suffices to assume $\Delta \cap D = \emptyset$. First it is easy to observe that:

(†) For each regular subexpression e of E using at least one operator (concatenation, $\cup$, or $+$), $p_e(a, b)$ is in $\Pi_p(D \cup \Delta)|_S$ if and only if there is an $L(e)$ -path in $D \cup \Delta$.

We then show by an induction on the number of operators that for each regular subexpression e of E ,

(‡) if $p_e(a, b)$ is in $\Pi_p(D \cup \Delta) - \Pi_p(D)$, then $p_e(a, b)$ is in $\Pi_\delta([\Pi_p(D)|_S \cup D]^o \cup \Delta)$.

Basis (Zero operators) e is an EDB predicate symbol. Then (‡) holds trivially because there is no EDB fact in $\Pi_p(D \cup \Delta)$.

Induction (One or more operators) Assume (‡) holds for all regular subexpressions of E with fewer than $i \geq 1$ operators. Let e be a regular subexpression of E with i operators.

Case 1: $e = e_1 \cup \dots \cup e_k$. Suppose $p_e(a, b) \in \Pi_p(D \cup \Delta) - \Pi_p(D)$. By the construction of Π_p , there exists $j \in [1..k]$ such that $p_{e_j}(a, b) \in [\Pi_p(D \cup \Delta) \cup \Delta] - \Pi_p(D)$. Then either $p_{e_j}(a, b) \in \Delta$ or, by the induction hypothesis, $p_{e_j}(a, b) \in \Pi_\delta([\Pi_p(D)|_S \cup D]^\circ \cup \Delta)$. Since $p_e(x, z) \leftarrow p_{e_j}(x, z)$ is a rule in Π_δ , $p_e(a, b) \in \Pi_\delta([\Pi_p(D)|_S \cup D]^\circ \cup \Delta)$ as desired.

Case 2: $e = e_1 \dots e_k$. Suppose $p_e(a, b)$ is in $\Pi_p(D \cup \Delta) - \Pi_p(D)$. Then there exist facts $p_{e_1}(a, c_1), \dots, p_{e_k}(c_{k-1}, b)$ in $D \cup \Delta \cup \Pi_p(D \cup \Delta)$. Since the rule $p_e(x, z) \leftarrow p_{e_1}(x, y_1), \dots, p_{e_k}(y_{k-1}, z)$ is in Π_p and $p_e(a, b)$ is not in $\Pi_p(D)$, at least one of these facts is not in $\Pi_p(D) \cup D$. By the induction hypothesis, all these facts are in $\Pi_\delta([\Pi_p(D)|_S \cup D]^\circ \cup \Delta) \cup D \cup \Delta$. Let $c_0 = a$ and $c_k = b$. Consider the rule

$$p_e(x, z) \leftarrow p_1(x, y_1), \dots, p_k(y_{k-1}, z)$$

in Π_δ , where p_j is p_{e_j} if $p_{e_j}(c_{j-1}, c_j)$ is in $[\Delta \cup \Pi_p(D \cup \Delta)] - \Pi_p(D)$, and $p_j = p_{e_j}^\circ$ otherwise. Clearly, an application of the rule yields $p_e(a, b)$. Thus $p_e(a, b)$ is in $\Pi_\delta([\Pi_p(D)|_S \cup D]^\circ \cup \Delta)$.

Case 3: $e = e_1^+$. Suppose $p_e(a, b)$ is in $\Pi_p(D \cup \Delta) - \Pi_p(D)$. By $(\dagger)$, there is an $L(e)$ -path from a to b . By Lemma 3.2, there exists an $L(e)$ -path P using Δ at most $\#_q(e)$ times. Let $P'_1, \dots, P'_n$ be $L(e_1)$ -paths such that $P = P'_1 \dots P'_n$. We combine the consecutive $L(e_1)$ -paths not using Δ to form $L(e)$ -paths. As a result, we obtain $L(e)$ -paths not using Δ and $L(e_1)$ -paths that use Δ . Let $P_1, \dots, P_k$ be those $L(e)$ or $L(e_1)$ paths such that $P = P_1 \dots P_k$. Note that, for each $i < k$, if P_i does not use Δ then P_{i+1} uses Δ . (But P_{i+1} may use Δ if P_i uses Δ .) Let $c_0 = a$, $c_k = b$, and $c_2, \dots, c_{k-1}$ be the constants such that P_i is from c_{i-1} to c_i . Let i be fixed. If P_i does not use Δ , then it is an $L(e)$ -path in D , and thus $p_e(c_{i-1}, c_i)$ is in $\Pi_p(D)$. Suppose P_i uses Δ . Then it is an $L(e_1)$ -path in $D \cup \Delta$, and thus $p_{e_1}(c_{i-1}, c_i)$ is in $[\Delta \cup \Pi_p(D \cup \Delta)] - \Pi_p(D)$ by $(\dagger)$. By the induction hypothesis, $p_{e_1}(c_{i-1}, c_i)$ is in $\Delta \cup \Pi_\delta([\Pi_p(D)|_S \cup D]^\circ \cup \Delta)$. Let $p_e(x, z) \leftarrow p_1(x, y_1), \dots, p_k(y_{k-1}, z)$ be the rule in Π_δ where $p_j = p_{e_1}$ if P_i uses Δ and $p_j = p_{e_j}^\circ$ otherwise. Clearly an application of this rule yields $p_e(a, b)$. Thus $p_e(a, b)$ is in $\Pi_\delta([\Pi_p(D)|_S \cup D]^\circ \cup \Delta)$. $\square$

In the remainder of this section we discuss a number of complexity measures on the IEC constructed by Algorithm 3.4. Suppose E is a $\{*, \epsilon, \emptyset\}$ -free regular expression constructed by Step 2 of Algorithm 3.4 for a regular query $Q = (\Pi, p)$. We focus on the interesting case where Π is recursive, i.e., the case where E contains some occurrence of $+$. We consider the complexity of Π_δ by examining Π_δ^q , where q is an EDB predicate occurring in Π .

We first consider the number of rules in Π_δ^q .

Proposition 3.7 The number of rules in Π_δ^q is the sum of (i) the number of union operations in E , (ii) $\sum (2^{\#_q(e)} - 1)$ where the sum ranges over E 's maximal subexpressions e of the form $e_1 \dots e_k$ built by using concatenation operations at the top level, and (iii) $\sum (2^{\#_q(e)+2} - 4)$, where the sum ranges over E 's subexpressions e with the form e_1^+ for some e_1 .

Proof First we observe that (i) Step 4.b produces one rule for each union operation in E , and (ii) Step 4.a produces $2^{\#_q(e)} - 1$ rules for each maximal subexpression e of the form $e_1 \dots e_k$ built by using concatenation operations at the top level.

It now suffices to show that Step 4.c produces $2^{\#_q(e)+2} - 4$ rules for $e = e_1^+$. For each $k \in [1..\#_q(e)]$, let $\mathcal{S}_k$ denote the set of sequences s over p_e^o and p_{e_1} such that in s there are exactly k occurrences of p_{e_1} and there are no adjacent occurrences of p_e^o . Then $\mathcal{S}_1 = \{p_{e_1}, p_{e_1}p_e^o, p_e^op_{e_1}, p_e^op_{e_1}p_e^o\}$, and so it contains $2^{1+1} = 4$ sequences. By induction, it is easily shown that (i) $\mathcal{S}_{k+1}$ consists of exactly those sequences obtained by appending either p_{e_1} or $p_{e_1}p_e^o$ to sequences in $\mathcal{S}_k$, (ii) sequences thus obtained are all distinct, and (iii) thus there are exactly 2^{k+2} sequences in $\mathcal{S}_{k+1}$. Consequently, the number of rules constructed by Step 4.c is $\sum_{k=1}^{\#_q(e)} 2^{k+1} = 2^{\#_q(e)+2} - 4$. $\square$

We next consider the number of joins needed by Π_δ^q to compute the new answer following the insertion of Δ , where Δ consists of an arbitrary q fact. It is easily observed that each rule constructed at Step 4.c needs at most $2\#_q(e)$ joins. Hence we have:

Proposition 3.8 The number of joins needed by Π_δ^q is bounded by the sum of (i) $\sum (k_e - 1)(2^{\#_q(e)} - 1)$ where the sum ranges over E 's maximal subexpressions e of the form $e_1 \cdots e_{k_e}$ built by using concatenation operations at the top level, and (ii) $2 \sum \#_q(e)(2^{\#_q(e)+2} - 4)$, where the sum ranges over E 's subexpressions e with the form e_1^+ for some e_1 . $\square$

Note that Π may need an unbounded number of joins whereas Π_δ always needs only a bounded number of joins.

The worst case time complexity of computing $\Pi_\delta((\Pi(D)|_S \cup D)^o \cup \Delta)$ by using Π_δ^q is n^{2j} , where n is the number of constants in D and j is the number of joins. Hence it cannot compete with incremental graph algorithms in this aspect. However, as was argued earlier, for database applications the number of joins is the desirable measure for efficiency. Furthermore, Π_δ^q can make use of the constants in Δ (as we indicated in Example 2.1) to improve efficiency.

The most desirable aspect of Π_δ^q is its parallel efficiency. Indeed, membership of facts in the answer to the query can be checked in constant time using Π_δ^q since Π_δ is a nonrecursive query [3].

We illustrate the above discussion using Examples 2.1 and 3.5. For Example 2.1, $E = edge^+$, there are four rules in Π_δ and the number of joins needed is four. By making use of the constants in Δ , the number of joins (now one) and the number of facts used in joins are significantly reduced (as remarked in Example 2.1).

For Example 3.5, the regular expression produced by Step 2 of Algorithm 3.4 is $E = (qs \cup q^+t)^+$. The number of rules in Π_δ^q is 20, where 2 rules are constructed by Step 4.a (1 each for the maximal subexpressions qs and q^+t constructed from concatenation at the top level), 2 by Step 4.b for subexpressions built from union at the top level, 4 by Step 4.c for q^+ , and 12 by Step 4.c for $(qs \cup q^+t)^+$. The actual number of joins needed is 31.

4 Weakly Regular Queries

In the previous section we constructed IEC for regular chain Datalog queries. Such queries are limited to binary predicates and chain rules. In this section we partially remove both of these restrictions by allowing nonrecursive predicates to be defined in terms of arbitrary conjunctive queries involving predicates of arbitrary arities. We generalize the IEC result to this case and to a special case of unbounded set insertion for regular chain queries.

Both generalized results are based on a useful key notion. To introduce that notion, two auxiliary concepts are first defined.

A set D of facts over a binary predicate q is called *cartesian closed* if $q(a_1, b_2)$ belongs to D for all $q(a_1, b_1)$ and $q(a_2, b_2)$ in D . (Hence every singleton set of the form $\{q(a, b)\}$ is cartesian closed.)

As we shall see in Lemma 4.4, each cartesian-closed set of facts can be treated as “one fact” in incremental computations.

The key notion is:

Definition Let $k \geq 0$ be an integer and p a predicate. Then a program Π has *k-cartesian-closed increment* (or *k-CCI* for short) with respect to p if for each database D and each set Δ of one EDB fact, there exist k cartesian-closed sets $C_1, \dots, C_k$ satisfying

$$\Pi(D \cup \Delta)|_p - \Pi(D)|_p \subseteq \bigcup_{i=1}^k C_i \subseteq \Pi(D \cup \Delta)|_p.$$

Π has *cartesian-closed increment* (or *CCI*) with respect to p if it has *k-CCI* with respect to p for some k .

The two containments basically say that the increment computed by Π following the insertion of Δ is “bounded” by the k cartesian-closed sets.

When no ambiguity arises (for example if there is only one IDB predicate) we will omit the phrase “with respect to p .”

We shall give characterizations (Propositions 5.2 and 5.4) for when nonrecursive one-rule Datalog programs (or equivalently, conjunctive queries) have CCI. Those results can be used to directly verify the five programs in the following example.

Example 4.1 Program $\Pi_1 = \{p_1(x, y) \leftarrow q_1(x, u, v), q_5(v, w, z), q_3(z, y)\}$ has 1-CCI. We verify the statement using a construction detailed in the proof for Proposition 5.4. For each database D and each set Δ of one EDB fact, let

$$C = \begin{cases} \{p_1(a, d_2) \mid q_5(c, d, d_1) \text{ and } q_3(d_1, d_2) \text{ in } D \text{ for some } d \text{ and } d_1\} & \text{if } \Delta \text{ has the form } \{q_1(a, b, c)\} \\ \{p_1(a_2, b_1) \mid q_1(a_2, a_1, a) \text{ and } q_3(b, b_1) \text{ in } D \text{ for some } a_1\} & \text{if } \Delta \text{ has the form } \{q_5(a, c, b)\} \\ \{p_1(c, b) \mid q_1(c, d_1, d_2) \text{ and } q_5(d_2, d, a) \text{ in } D \text{ for some } d, d_1 \text{ and } d_2\} & \text{if } \Delta \text{ has the form } \{q_3(a, b)\} \end{cases}$$

Then C is cartesian closed, and $\Pi_1(D \cup \Delta) - \Pi_1(D) \subseteq C \subseteq \Pi_1(D \cup \Delta)$ hold.

Program $\Pi_2 = \{p_1(x, y) \leftarrow q_1(x, u, v), q_5(v, u, z), q_6(z, u, y)\}$ also has 1-CCI.

The empty program Π_3 has 1-CCI. (The empty set is cartesian closed.)

Program $\Pi_4 = \{p_1(x, y) \leftarrow q_4(x, y), q_2(u, v)\}$ does not have CCI. Indeed, for each $k \geq 0$, suppose D is a set of q_4 facts such that D is not the union of any k cartesian-closed sets. Then $\Pi_4(D \cup \{q_2(a, b)\}) - \Pi_4(D)$ is not bounded by any k cartesian-closed sets in $\Pi_4(D \cup \{q_2(a, b)\})$, violating the two containments in the above definition.

Program $\Pi_5 = \{p(x, y) \leftarrow q_1(x, u, v), q_2(z, v), q_6(z, u, y)\}$ does not have CCI. (Compare Π_5 with Π_2 above.) A database D and increment Δ are shown in Figure 2 where the increased answer is not a union of $\leq k$ cartesian closed sets. $\square$

	q_1	q_2	q'_3
$D =$	00b 11b ⋮ kkb	$\emptyset$	aa1 a12 ⋮ akk+1
$\Delta = \{q_2(a,b)\}$			
$\Pi_5(D) = \emptyset$			
$\Pi_5(D \cup \Delta) _p =$	01 12 ⋮ kk+1		

Figure 2: Π_5 does not have k -CCI

We now give a one-rule program and a two-rule program that have 2-CCI but not 1-CCI. For each positive integer k , one can easily modify the example to give programs having $(k + 1)$ -CCI but not having k -CCI.

Example 4.2 Let Π be the program consisting of the following rule:

$$p(x, y) \leftarrow q(x, z_1), q(z_1, y).$$

Then Π has 2-CCI (by Proposition 5.4). To show that Π does not have 1-CCI, let $D = \{q(0, 1), q(2, 3)\}$

and $\Delta = \{q(1, 2)\}$, then $\Pi(D) = \emptyset$ and $\Pi(D \cup \Delta) - \Pi(D) = \Pi(D \cup \Delta) = \{p(0, 2), p(1, 3)\}$, which is not cartesian closed.

Let $\Pi = \Pi_1 \cup \Pi_2$. Π has 2-CCI since Π_1 and Π_2 have 1-CCI. However, Π does not have 1-CCI. Let $D = \{q_1(a_1, b_1, a), q_3(c, c_1), q_1(a_2, b, a), q_6(c, b, c_2)\}$ and $\Delta = \{q_5(a, b, c)\}$. Then $\Pi(D) = \emptyset$ and $\Pi(D \cup \Delta) = \{p_1(a_1, c_1), p_1(a_2, c_2)\}$. Hence there cannot be a cartesian-closed set C such that $\Pi(D \cup \Delta) = \Pi(D \cup \Delta) - \Pi(D) \subseteq C \subseteq \Pi(D \cup \Delta)$. $\square$

We shall generalize the IEC existence result to queries whose program component are the union of a regular chain program and, for each nonrecursive predicate, one program with CCI defining that nonrecursive predicate.

Definition A query (Π, p) is *weakly regular* if $\Pi = \Pi_1 \cup \Pi_2$ satisfying the following:

1. (Π_1, p) is a regular query (viewing IDB predicates of Π_2 as EDB predicates);
2. Each predicate in the heads of rules in Π_1 does not occur in Π_2 ;
3. For each EDB predicate q of Π_1 , let Π^q be the set of rules in Π_2 defining q . The single IDB-query equivalent to Π^q (obtained by eliminating other IDB predicates) has CCI.

Example 4.3 Let Π consist of the following rules:

$$\begin{aligned} p(x, y) &\leftarrow p_1(x, z), p(z, y) \\ p(x, y) &\leftarrow p_1(x, y) \\ p_1(x, y) &\leftarrow q_1(x, u, v), q_5(v, w, z), q_3(z, y) \end{aligned}$$

Then (Π, p) is a weakly regular query. Indeed, the first two rules form a regular chain program, and the last rule is a nonrecursive rule (Π_1 of Example 4.1) with CCI defining the nonrecursive predicate p_1 . $\square$

In order to establish the generalized results, we now generalize the key lemma for the regular query case, Lemma 3.2.

Lemma 4.4 Let D be a labelled directed graph, $C \subseteq D$ a cartesian-closed set of edges labelled by q , E be a $\{*, \epsilon, \emptyset\}$ -free regular expression, and b_1 and b_2 be nodes. If there is an $L(E)$ -path in D from b_1 to b_2 , then there is such a path in which edges from C occur at most $\#_q(E)$ times.

Proof The main idea is the same as the proof for Lemma 3.2. The distinct feature is the use of the cartesian-closed set to achieve path “contraction.”

Suppose there exists an $L(E)$ -path $q_1(c_0, c_1) \cdots q_k(c_{k-1}, c_k)$ from b_1 to b_2 . As for Lemma 3.2, suppose there is a position i_0 such that two facts $q(a_1, d_1)$ and $q(a_2, d_2)$ from C appear in the path corresponding to the position i_0 in E . Then $q(a_1, d_2)$ is also in C since C is cartesian closed. This allows us to shorten the

path by replacing the path interval from $q(a_1, d_2)$ to $q(a_2, d_2)$ by $q(a_1, d_2)$, thus decreasing the number of occurrences of facts from C . $\square$

Similar to the regular chain query case, we have the following result:

Theorem 4.5 There is an IEC for each weakly regular query (Π, p) .

Proof For each nonrecursive binary predicates q in Π , let Π_q be the program defining q , and let k_q be an integer such that Π_q has k_q -CCI. Let k be the sum of all such k_q (including duplicates). For the insertion of one EDB fact, we view each of the k or fewer cartesian-closed sets derived by the Π_q 's as one inserted "fact" to some nonrecursive predicate q . Since there are at most k cartesian-closed sets for all the nonrecursive predicates, we can use the construction for the regular chain query case for k inserted facts. The correctness of this construction is guaranteed by Lemma 4.4. $\square$

Note that this result can be generalized (with the same proof) to queries (Π, p) where Π is the union of a regular chain program Π_1 and, for each predicate q that is not recursive in Π_1 , an arbitrary (possibly recursive) Datalog program Π_q with CCI that defines q and predicates $q' \neq q$ occurring in Π_q do not depend on predicates in Π_1 .

As a particular case of Theorem 4.5, for transitive closure we have the following:

Corollary 4.6 If (Π, p) is a weakly regular query such that the rules in Π with recursive predicates in rule heads are the following rules computing transitive closure:

$$\begin{aligned} p(x, y) &\leftarrow q(x, y) \\ p(x, y) &\leftarrow q(x, z), p(z, y) \end{aligned}$$

then there is an IEC for (Π, p) . $\square$

Since (Π, p) of Example 4.3 is a weakly regular query, by Theorem 4.5 there is an IEC for (Π, p) . Example 4.7 below shows a non-weakly regular query (Π, p) that has an IEC. It is still open whether there is a Datalog query (Π, p) that does not have an IEC. However, one can easily verify that the construction given in Theorem 4.5 does not work for non-weakly regular queries such as (Π, p) , where Π is the following program:

$$\begin{aligned} p_0(x, y) &\leftarrow q_1(u, x, z), q_2(z, v), q_6(v, u, y) \\ p(x, y) &\leftarrow p_0(x, y) \\ p(x, y) &\leftarrow p_0(x, z), p(z, y) \end{aligned}$$

Example 4.7 Indeed, let Π be the following program:

$$\begin{aligned} p(x, y) &\leftarrow p_1(x, z), p(z, y) \\ p(x, y) &\leftarrow p_1(x, y) \\ p_1(x, y) &\leftarrow q_4(x, y), q_2(u, v) \end{aligned}$$

Then the first two rules form a regular chain program, and the last rule defines the nonrecursive predicate p_1 and does not have CCI (Π_4 of Example 4.1). Query (Π, p) is equivalent to (Π', p) , where Π' is the following program:

$$\begin{aligned} r_1: \quad & p(x, y) \leftarrow p_2(x, y), q_2(u, v) \\ r_2: \quad & p_2(x, y) \leftarrow q_4(x, y) \\ r_3: \quad & p_2(x, y) \leftarrow q_4(x, z), p_2(z, y) \end{aligned}$$

Let $\Pi_1 = \{r_2, r_3\}$. Then there is an IEC $\langle \Pi_{1p}, S_1, \Pi_{1\delta} \rangle$ for (Π_1, p_2) . Let $\Pi_p = \Pi_{1p} \cup \{r_1\}$, $S = S_1 \cup \{p\}$, and $\Pi_\delta = \Pi_{1\delta} \cup \{r_1\}$. Then one can easily check that $\langle \Pi_p, S, \Pi_\delta \rangle$ is an IEC for (Π, p) . $\square$

So far we have considered IEC that computes the increment after the insertion of one fact. For set-at-a-time insertions, we have the following result:

Theorem 4.8 For each regular chain query $Q = (\Pi, p)$, there is an IEC $\langle \Pi_p, S, \Pi_\delta \rangle$ that can compute the increment after each insertion of a cartesian-closed set, that is, $\Pi_p(D \cup \Delta)|_S = \Pi_\delta([\Pi_p(D)|_S \cup D]^o \cup \Delta)|_S \cup \Pi_p(D)|_S$ for each database D and each cartesian-closed set Δ of EDB facts.

Proof We view a cartesian-closed set as one inserted “fact”, and use the construction for the regular chain query case. The correctness of this construction is guaranteed by Lemma 4.4. $\square$

It is also of interest to extend the results beyond weakly regular queries.

5 Properties of Weak Regularity

To enhance the conclusion that weakly regular queries have IEC, the main result of this section shows that it is decidable if an arbitrary query (Π, p) is weakly regular, where Π is the union of a regular chain program Π_1 and, for each predicate q that is not recursive in Π_1 , a nonrecursive program Π_q defining q . In particular, we completely characterize CCI for nonrecursive one-rule programs, and then establish a characterization and the decidability result for CCI of nonrecursive multi-rule programs. In contrast, the decidability result cannot be extended to queries with recursive Π_q programs.

The main result of this section is now stated.

Theorem 5.1 It is decidable if an arbitrary query (Π, p) is weakly regular, where Π is the union of a regular chain program Π_1 and, for each predicate q that is not recursive in Π_1 , a nonrecursive program Π_q defining q .

This theorem is a direct consequence of Corollary 5.6. Recall that a query is weakly regular if the recursive part is a regular chain program Π_1 and each nonrecursive IDB predicate occurring in Π_1 has

the CCI property. Since the former is a purely syntactic condition and the latter is not, the main part is to determine, for each nonrecursive query if it has CCI. It shall be shown that the CCI property is indeed decidable (Corollary 5.6). In the following, we first present the characterizations for each rule to have CCI according to two cases, depending on whether the two variables in the rule head are the same or not. We then present the decidability result.

We shall need the notion of nonredundancy. A program Π_1 is *contained* in another program Π_2 , denoted $\Pi_1 \sqsubseteq \Pi_2$, if $\Pi_1(D) \subseteq \Pi_2(D)$ for each database D . Two programs Π_1 and Π_2 are *equivalent*, denoted $\Pi_1 \equiv \Pi_2$, if $\Pi_1 \sqsubseteq \Pi_2$ and $\Pi_2 \sqsubseteq \Pi_1$. A rule $r : A_0 \leftarrow A_1, \dots, A_m$ is *nonredundant* if, for each $i \in [1..m]$, $\{r\} \not\equiv \{A_0 \leftarrow A_1, \dots, A_{i-1}, A_{i+1}, \dots, A_m\}$. A program Π is *nonredundant* if, for each rule r in Π , r is nonredundant and $\Pi \not\equiv \Pi - \{r\}$.

In [31] nonredundant rules (programs) are termed “minimal” rules (programs), and a detailed discussion and a method (based on containment substitution defined below) for obtaining minimal equivalent program of a nonrecursive program are given. In fact, suppose we are given two nonrecursive rules:

$$\begin{aligned} r : A &\leftarrow A_1, \dots, A_m \\ r' : B &\leftarrow B_1, \dots, B_n \end{aligned}$$

We define a *containment substitution*⁵ from r' to r as a substitution σ such that $\sigma(B) = A$ and, for each B_i there is an A_j with $\sigma(B_i) = A_j$. Then $r \sqsubseteq r'$ iff there is a containment substitution from r' to r .

The characterization for one rule with equal variables in the rule head is as follows:

Proposition 5.2 Suppose $r : p(x_1, x_1) \leftarrow A_1, \dots, A_m$ is a nonredundant and nonrecursive rule, and $\Pi = \{r\}$. Then the following three conditions are equivalent:

1. Π has CCI.
2. x_1 occurs in A_i for each $i \in [1..n]$.
3. There exists an integer k such that $\Pi(D \cup \Delta) - \Pi(D)$ contains at most k facts for each database D and each set Δ of one EDB fact.

Proof Clearly (3) implies (1).

Suppose (2) holds. Let k be the maximum of arities of EDB predicates in Π . Then for each database D and each set Δ of one EDB fact, there are at most k constants in Δ . By (2), there are at most k facts in $\Pi(D \cup \Delta) - \Pi(D)$. Hence (3) holds.

Suppose (1) holds but (2) is false. Then there is an integer i such that x_1 does not occur in A_i . Suppose k is an integer such that Π has k -CCI. Let τ be a one-to-one assignment which maps each variable in A_i to some constant. (The one-to-one property will be used to establish Proposition 5.5.) For each

---

⁵Substitutions are mappings from variables to terms which are the identity on constants, and are extended naturally to atoms.

$j \in [1..k + 1]$, let τ_j be an assignment defined (for each variable x) by $\tau_j(x)$ to a unique new constant if x does not occur in A_i and $\tau_j(x) = \tau(x)$ otherwise. Let $D = \{\tau_{j_1}(A_{j_2}) \mid j_1 \in [1..k + 1], j_2 \in [1..m]\}$ and $\Delta = \{\tau(A_i)\}$. Then $\Pi(D \cup \Delta) - \Pi(D)$ contains at least $k + 1$ facts of the form $p(c, c)$, and contains no fact of the form $p(c, d)$ where $c \neq d$. Hence there cannot be k cartesian-closed sets $C_1, \dots, C_k$ such that $\Pi(D \cup \Delta) - \Pi(D) \subseteq \cup_{j=1}^k C_j \subseteq \Pi(D \cup \Delta)$, contradicting (1). $\square$

For the rule r in the statement of Proposition 5.2, one can compute the minimal number ρ such that $\{r\}$ has ρ -CCI. In fact, for each EDB predicate q occurring in r , suppose k is its arity. Let $a_1, \dots, a_k$ be k distinct constants, and let ρ_q be the size of the set $\{\mu(x_1) \mid i \in [1..m], \mu \text{ a most general unifier (MGU) for } q(a_1, \dots, a_k) \text{ and } A_i\}$.⁶ Then one can show that ρ equals the maximum of such ρ_q 's.

To present our characterization for rules whose heads contain two distinct variables, we need an auxiliary notion concerning a variant of connectivity, and a technical result.

We first present the concept. For all sets $\mathcal{S}$ and $\mathcal{S}'$ of atoms, two variables x and y are called $(\mathcal{S}, \mathcal{S}')$ -connected if there is a sequence $A_1, \dots, A_m$ of atoms in $\mathcal{S}$ (repetitions permitted) such that x occurs in A_1 , y in A_m , and A_i and A_{i+1} have a common variable not occurring in $\mathcal{S}'$ for each $i \in [1..m - 1]$. We write $(\mathcal{S}, A)$ -connected for $(\mathcal{S}, \{A\})$ -connected.

For example, for $\mathcal{S} = \{q_1(x, u, v), q_6(z, u, y)\}$ and $A = q_2(v, z)$, x and y are $(\mathcal{S}, A)$ -connected. Indeed, the sequence $q_1(x, u, v), q_6(z, u, y)$ verifies the $(\mathcal{S}, A)$ -connectivity since x occurs in $q_1(x, u, v)$, y in $q_6(z, u, y)$, and u occurs in both of the two atoms and does not occur in A . In contrast, for $\mathcal{S} = \{q_1(x, u, v), q_6(z, u, y)\}$ and $A = q_5(v, u, z)$, x and y are not $(\mathcal{S}, A)$ -connected. The sequence $q_1(x, u, v), q_6(z, u, y)$ no longer verifies the $(\mathcal{S}, A)$ -connectivity since u now occurs in A and there is no variable which is not in A and which occurs in both $q_1(x, u, v)$ and $q_6(z, u, y)$.

Note that $(\mathcal{S}, \emptyset)$ -connectivity reduces to the usual connectivity.

We now establish the technical lemma. Intuitively it says, if there are certain k grounded copies of the body of r and one can generate $p(\tau_{i_1}(x_1), \tau_{i_2}(x_2))$ from these copies using r' , then $\{r\} \sqsubseteq \{r'\}$.

Lemma 5.3 Let $r : p(x_1, x_2) \leftarrow A_1, \dots, A_m$ be a nonredundant and nonrecursive rule. Suppose $\mathcal{S}$ is a set of atoms and $\tau_1, \dots, \tau_k$ are one-to-one assignments such that, for all distinct $i, j \in [1..k]$, $\tau_i(x) = \tau_j(y)$ iff x and y occur in $\mathcal{S}$ and $x = y$. If $r' : p(y_1, y_2) \leftarrow B_1, \dots, B_n$ is a nonrecursive rule such that $p(\tau_{i_1}(x_1), \tau_{i_2}(x_2)) \in \{r'\}(D_k)$, where $D_k = \{\tau_i(A_j) \mid i \in [1..k] \wedge j \in [1..m]\}$ and $i_1, i_2 \in [1..k]$, then $\{r\} \sqsubseteq \{r'\}$.

Proof We prove the lemma by induction on k . First consider the basis case, $k = 1$. Since $p(\tau_1(x_1), \tau_1(x_2)) \in \{r'\}(D_1)$, there exists an assignment α such that for each $i \in [1..n]$, $\alpha(B_i) \in D_1 = \{\tau_1(A_j) \mid j \in [1..m]\}$ and $\alpha(y_1, y_2) = \tau_1(x_1, x_2)$. By the assumption that τ_1 is one-to-one, $\tau_1^{-1} \circ \alpha$ is⁷ a containment substitution from r' to r . Hence, $\{r\} \sqsubseteq \{r'\}$.

⁶Two atoms A and B are *unifiable* if there is a substitution ψ such that $\psi(A) = \psi(B)$. And ψ is a *most general unifier* (MGU) for A and B if ψ is their unifier and, for each unifier ϕ for A and B , there is a substitution γ such that $\phi = \gamma\psi$.

⁷ $f \circ g$ is the mapping defined (for all x) by $f \circ g(x) = f(g(x))$.

Now suppose the lemma holds for the $k - 1$ assignments $\tau_1, \dots, \tau_{k-1}$ and consider the case for k assignments $\tau_1, \dots, \tau_k$. Let $D_{k-1} = \{\tau_i(A_j) \mid i \in [1..k-1], j \in [1..m]\}$. Define the mapping μ on constants such that, for each x occurring in r , $\mu(\tau_k(x)) = \tau_1(x)$, and μ is the identity mapping elsewhere. We see that $\mu(D_k) \subseteq D_{k-1}$ since, for all distinct $i, j \in [1..k]$, $\tau_i(x) = \tau_j(y)$ iff x and y occur in $\mathcal{S}$ and $x = y$. Indeed, let A be an atom in $\{A_1, \dots, A_m\}$. Clearly $\mu(\tau_1(A)) = \mu(\tau_k(A)) = \tau_1(A)$. Let $i \in [2..k-1]$. Let x be a variable occurring in A . If x occurs in $\mathcal{S}$, then $\tau_i(x) = \tau_k(x) = \tau_1(x)$, and thus $\mu(\tau_i(x)) = \tau_1(x)$. Otherwise, $\tau_i(x) \neq \tau_k(y)$ for any y , and thus $\mu(\tau_i(x)) = \tau_i(x)$. Consequently, $\mu(\tau_i(A)) = \tau_i(A)$.

Since $p(\tau_{i_1}(x_1), \tau_{i_2}(x_2)) \in \{r'\}(D_k)$, there exists an assignment α such that $\alpha(y_1, y_2) = (\tau_{i_1}(x_1), \tau_{i_2}(x_2))$ and $\alpha(\{B_i \mid i \in [1..n]\}) \subseteq D_k$. Let $\alpha' = \mu \circ \alpha$. Obviously,

$$\alpha'(y_1, y_2) = \mu(\alpha(y_1, y_2)) = \mu(\tau_{i_1}(x_1), \tau_{i_2}(x_2)) = (\tau_{j_1}(x_1), \tau_{j_2}(x_2))$$

where for each $\ell \in \{1, 2\}$, $j_\ell = 1$ if $\tau_{i_\ell}(x_\ell) = \tau_k(x_\ell)$ and $j_\ell = i_\ell$ otherwise; and

$$\alpha'(\{B_i \mid i \in [1..n]\}) = \mu(\alpha(\{B_i \mid i \in [1..n]\})) \subseteq \mu(D_k) \subseteq D_{k-1}.$$

Hence $p(\tau_{j_1}(x_1), \tau_{j_2}(x_2)) \in \{r'\}(D_{k-1})$. Clearly, both j_1 and j_2 belong to $[1..k-1]$. By the induction hypothesis on $k - 1$ assignments $\tau_1, \dots, \tau_{k-1}$, $\{r\} \sqsubseteq \{r'\}$. $\square$

The characterization for one rule whose head has distinct variables is as follows:

Proposition 5.4 Suppose $r : p(x_1, x_2) \leftarrow A_1, \dots, A_m$ is a nonredundant and nonrecursive rule where $x_1 \neq x_2$, and $\Pi = \{r\}$. Then Π has CCI iff

- (*) for each $i \in [1..m]$, either (1) A_i contains x_1 or x_2 ,
or (2) x_1 and x_2 are not $(\mathcal{S}_i, A_i)$ -connected,

where $\mathcal{S}_i = \{A_j \mid 1 \leq j \leq m, A_j \neq A_i\}$. Furthermore, if (*) holds then Π has ρ -CCI, where ρ is the maximum number of atoms in the body of r with a common EDB predicate.

Proof For the “if”, suppose (*) holds. Let D be an arbitrary database, A an arbitrary EDB fact, $\Delta = \{A\}$, and $i \in [1..m]$. We will use the non $(\mathcal{S}_i, A_i)$ -connectivity to construct a cartesian-closed set C_i .

If A_i and A are not unifiable, let $C_i = \emptyset$. Otherwise, let τ_i be an MGU for A_i and A , and $C_i = \{p(c_1, c_2) \mid c_1 \in W_{i1}, c_2 \in W_{i2}\}$, where W_{i1} and W_{i2} are defined according to two cases:

- (a) At least one of x_1 and x_2 occurs in A_i . For each $k \in \{1, 2\}$, define W_{ik} as

$$\{\sigma(\tau_i(x_k)) \mid \sigma \text{ an assignment with } \sigma(\tau_i(A_j)) \in D \cup \Delta \text{ for all } j \in [1..m]\}.$$

To verify $C_i \subseteq \Pi(D \cup \Delta)$ for case (a), let $p(c_1, c_2)$ be in C_i . Three subcases arise. (a1) Both x_1 and x_2 occur in A_i . Since $C_i \neq \emptyset$, there is an assignment σ such that $\sigma(\tau_i(A_j)) \in D \cup \Delta$ for all $j \in [1..m]$. For each $k \in \{1, 2\}$, since τ_i is an MGU for A and A_i and since x_k occurs in A_i , $\tau_i(x_k)$ is a constant. Therefore

$p(c_1, c_2) = \tau_i(p(x_1, x_2)) = \sigma(\tau_i(p(x_1, x_2))) \in \Pi(D \cup \Delta)$. (a2) x_1 occurs in A_i but x_2 does not. Since $c_2 \in W_{i2}$, there is an assignment σ such that $\sigma(\tau_i(x_2)) = c_2$ and $\sigma(\tau_i(A_j)) \in D \cup \Delta$ for all $j \in [1..m]$. As in (a1), we can show that $c_1 = \tau_i(x_1)$. Hence, $p(c_1, c_2) = \sigma(\tau_i(p(x_1, x_2))) \in \Pi(D \cup \Delta)$. (a3) x_2 occurs in A_i but x_1 does not. This case is symmetrical to case (a2).

(b) Neither x_1 nor x_2 occurs in A_i . Let $\mathcal{S}_{i1}$ be a maximal subset of $\mathcal{S}_i$ such that every variable occurring in $\mathcal{S}_{i1}$ is $(\mathcal{S}_i, A_i)$ -connected with x_1 . Let $\mathcal{S}_{i2} = \mathcal{S}_i - (\mathcal{S}_{i1} \cup \{A_i\})$. Since x_2 is not $(\mathcal{S}_i, A_i)$ -connected with x_1 , $\mathcal{S}_{i2}$ contains an atom where x_2 occurs. For each $k \in \{1, 2\}$, let $W_{ik} = \{\sigma(\tau_i(x_k)) \mid \sigma \text{ an assignment such that } \sigma(\tau_i(A_j)) \in D \cup \Delta \text{ for each } A_j \in \mathcal{S}_{ik}\}$.

To verify $C_i \subseteq \Pi(D \cup \Delta)$ for case (b), let $p(c_1, c_2) \in C_i$. For each $k \in \{1, 2\}$, there exists an assignment σ_k such that $\sigma_k(\tau_i(A_j)) \in D \cup \Delta$ for each $A_j \in \mathcal{S}_{ik}$ and $\sigma_k(\tau_i(x_k)) = c_k$. Let σ be defined (for each variable u) by $\sigma(u) = \sigma_1(u)$ if u occurs in $\mathcal{S}_{i1}$ and $\sigma(u) = \sigma_2(u)$ otherwise. Since variables in $\mathcal{S}_{i1}$ are not $(\mathcal{S}_i, A_i)$ -connected with variables in $\mathcal{S}_{i2}$, $\sigma\tau_i$ is a well-defined assignment, and $\sigma(\tau_i(A_j)) \in D \cup \Delta$ for each $A_j \in \mathcal{S}_i$. Since τ_i is an MGU for A_i and A , $\sigma(\tau_i(A_i)) = \tau_i(A_i) = A \in D \cup \Delta$. Hence $p(c_1, c_2) = \sigma(\tau_i(p(x_1, x_2))) \in \Pi(D \cup \Delta)$.

To complete the proof of the “if”, it suffices to verify for cases (a) and (b) that

$$\Pi(D \cup \Delta) - \Pi(D) \subseteq \cup_{i=1}^m C_i.$$

Let A_0 be a fact in $\Pi(D \cup \Delta) - \Pi(D)$. Then there exist $i \in [1..m]$ and assignment ϕ such that $\phi(p(x_1, x_2)) = A_0$, $\phi(A_i) = A$ and $\phi(A_j) \in D \cup \Delta$ for each $j \in [1..m]$. Since τ_i is an MGU for A_i and A , there is an assignment σ such that $\sigma(\tau_i(A_j)) = \phi(A_j)$ for each $j \in [1..m]$. One can now easily see that $A_0 \in C_i$ for cases (a) and (b), completing the proof for the “if.”

From the construction above, clearly Π has ρ -CCI.

To verify the “only if”, suppose (*) is false. Assume to the contrary that k is an integer such that Π has k -CCI. Then there exists some $i \in [1..m]$ such that A_i contains neither x_1 nor x_2 , and x_1 and x_2 are $(\mathcal{S}_i, A_i)$ -connected. Let $\mathcal{S}_{i1}$ be the maximal subset of $\mathcal{S}_i - \{A_i\}$ such that each variable occurring in $\mathcal{S}_{i1}$ is $(\mathcal{S}_i, A_i)$ -connected to x_1 . Let $\mathcal{S}_{i2} = (\mathcal{S}_i - \mathcal{S}_{i1}) \cup \{A_i\}$.

Let τ be a substitution that maps distinct variables in $\mathcal{S}_{i2}$ to distinct constants not in $[1..2k+2]$. For each $j \in [1..k+1]$, let τ_j be a substitution such that $\tau_j(x_1) = j$, $\tau_j(x_2) = 2j$, $\tau_j(u) = \tau(u)$ if u occurs in $\mathcal{S}_{i2}$, and $\tau_j(v)$ is a distinct new constant for each other variable v .

Let $\Delta = \{\tau(A_i)\}$ and $D = \{\tau_j(A_n) \mid j \in [1..k+1], n \in [1..m]\} - \Delta$. Since Π has k -CCI, there are k cartesian-closed sets $C_1, \dots, C_k$ such that

$$\Pi(D \cup \Delta) - \Pi(D) \subseteq \cup_{i=1}^k C_i \subseteq \Pi(D \cup \Delta).$$

Claim: If there is an assignment σ such that $\sigma(A_\ell) \in D \cup \Delta$ for each $\ell \in [1..m]$ and $\sigma(x_1) = j$, then $\sigma(u) = \tau_j(u)$ for all variables occurring in $\mathcal{S}_{i1}$.

Proof: Suppose y is $(\mathcal{S}_i, A_i)$ -connected to x_1 . Then there is a sequence $B_1, \dots, B_n$ of atoms from $\mathcal{S}_i$ such that x_1 occurs in B_1 , x_2 in B_n , and for each $j \in [1..n-1]$, there exists a variable y_j that occurs

in both B_j and B_{j+1} but not in A_i . Since $\sigma(x_1) = j$ and no other substitution from $\tau_1, \dots, \tau_{k+1}$ maps any variable to j , $\sigma(x_1) = \tau_j(x_1)$ and $\sigma(B_1) = \tau_j(B_1)$. Since y_1 occurs in B_1 , $\sigma(y_1) = \tau_j(y_1)$. Since y_1 occurs in B_2 and $\tau_j(y_1)$ is a constant not in the image of any assignment $\tau_{j'}$ with $j \neq j'$, we similarly conclude that $\sigma(B_2) = \tau_j(B_2)$ and $\sigma(y_2) = \tau_j(y_2)$. Continuing this way we conclude that $\sigma(y) = \tau_j(y)$.

We now verify that, for each $j \in [1..k+1]$, $p(j, 2j)$ belongs to $\Pi(D \cup \Delta) - \Pi(D)$. From the definition of τ_j , $p(j, 2j)$ belongs to $\Pi(D \cup \Delta)$. To reach a contradiction, assume $p(j, 2j) \in \Pi(D)$. Then there exists an assignment σ such that σ maps each atom in the body of r to a fact in D , $\sigma(x_1) = j$, and $\sigma(x_2) = 2j$. By Claim, $\sigma(u) = \tau_j(u)$ for each u occurring in $\mathcal{S}_{i1}$. Two cases arise:

(a) There is a variable x_0 occurring in A_i which is $(\mathcal{S}_i, A_i)$ -connected with x_1 .

Clearly, x_0 occurs in $\mathcal{S}_{i1}$. So $\sigma(x_0) = \tau_j(x_0)$. Since x_0 occurs in A_i , $\tau_j(x_0) = \tau(x_0)$. Since $\tau(x_0)$ is unique in the image of τ , we see that $\sigma(A_i) = \tau(A_i) \notin D$, a contradiction.

(b) There is no variable occurring in A_i which is $(\mathcal{S}_i, A_i)$ -connected with x_1 .

Let r' be the rule $p(x_1, x_2) \leftarrow A_1, \dots, A_{i-1}, A_{i+1}, \dots, A_m$. Clearly, $\{r\} \subseteq \{r'\}$. By Lemma 5.3 we see that $\{r'\} \subseteq \{r\}$. Hence $r \equiv r'$, contradicting the nonredundancy of r .

It now suffices to show that, for distinct j_1 and j_2 in $[1..k+1]$, $p(j_1, 2j_2) \notin \Pi(D \cup \Delta)$. (Note that $p(j_1, 2j_2) \notin \Pi(D \cup \Delta)$ implies that $p(j_1, 2j_1)$ and $p(j_2, 2j_2)$ cannot be in one cartesian-closed set.) Suppose there is an assignment ψ such that $\psi(A_j) \in D \cup \Delta$ for each j and $\psi(A_0) = p(j_1, 2j_2)$. Then Claim implies that $2j_2 = \psi(x_2) = \tau_{j_1}(x_2) = 2j_1$, a contradiction. $\square$

Three remarks are in order. (1) In the “if” direction the nonredundancy condition is not used. (2) The characterization can be used directly to verify the five example programs given in Example 4.1. (3) As another application of the characterization, we see that the relational join operation corresponding to $p(x, y) \leftarrow q_4(x, z), q_2(z, y)$ has 1-CCI.

We assume each nonrecursive multi-rule program is a *single IDB-predicate* program in the sense that there is at most one predicate symbol occurring in the heads of all its rules. (The rule heads can have different variable patterns.) No generality is lost due to this restriction. Indeed, for each predicate q , each nonrecursive program Π_q can be converted into a nonrecursive single IDB-predicate program Π'_q defining q satisfying $\Pi_q(D)|_q = \Pi'_q(D)$ for each database D .

The characterization for the multi-rule case is as follows.

Proposition 5.5 A nonredundant and nonrecursive, single IDB-predicate program Π with a binary IDB predicate has CCI iff $\{r\}$ has CCI for each $r \in \Pi$.

Proof We first note that the “if” is a special case of the following statement: A multi-rule nonrecursive program Π has CCI if it is the union of single IDB-predicate programs (over a common IDB-predicate) with CCI. This more general statement holds since $\cup_{i=1}^n \Pi_i$ has $(\sum_{i=1}^n k_i)$ -CCI if each Π_i ($1 \leq i \leq n$) has k_i -CCI.

For the “only if”, suppose $k > 0$ and Π has k -CCI. To reach a contradiction, assume $r : p(x_1, x_2) \leftarrow A_1, \dots, A_m$ is a rule in Π such that $\{r\}$ does not have CCI. Two cases arise:

(a) $x_1 = x_2$. By Proposition 5.2, there exists an i such that x_1 does not occur in A_i . Let τ, τ_j ($j \in [1..k+1]$), D and Δ be constructed as in the proof for “(1) implies (2)” in Proposition 5.2. Note that $\{r\}(D \cup \Delta) - \{r\}(D)$ contains at least $k+1$ facts of the form $p(c, c)$ and it contains no fact of the form $p(c, d)$ with $c \neq d$. Since Π has k -CCI, there must be a rule $r' : B \leftarrow B_1, \dots, B_n$ in Π and an assignment α with $\alpha(B_j) \in D \cup \Delta$ for each $j \in [1..n]$ and $\alpha(B) = p(c_1, c_2)$, where $c_1 = \tau_{j_1}(x_1)$ and $c_2 = \tau_{j_2}(x_2)$ for some j_1 and j_2 . By Lemma 5.3, $r \sqsubseteq r'$, and thus $\Pi \sqsubseteq \Pi - \{r\}$. Clearly, $\Pi - \{r\} \sqsubseteq \Pi$. Therefore $\Pi \equiv \Pi - \{r\}$, and so Π is redundant, a contradiction.

(b) $x_1 \neq x_2$. Using the notation of Proposition 5.4 and by that proposition, there is an integer i such that x_1 and x_2 are $(\mathcal{S}_i, A_i)$ -connected. Let $\mathcal{S}_{i1}, \mathcal{S}_{i2}, \tau, \tau_j$ ($j \in [1..k+1]$), D and Δ be constructed as in the proof for “ Π has CCI implies (*)” for Proposition 5.4. As was shown in Proposition 5.4, $\Pi(D \cup \Delta) - \Pi(D)$ contains at least $k+1$ facts $p(j, 2j)$ ($j \in [1..k+1]$), and contains no fact of the form $p(j_1, 2j_2)$ where $j_1 \neq j_2$. Since Π has k -CCI, there must be a rule $r' : B \leftarrow B_1, \dots, B_n$ in Π and an assignment α with $\alpha(B_j) \in D \cup \Delta$ for each $j \in [1..n]$ and $\alpha(B) = p(c_1, c_2)$, where $c_1 = \tau_{j_1}(x_1)$ and $c_2 = \tau_{j_2}(x_2)$ for some j_1 and j_2 . Again by Lemma 5.3 Π is redundant, a contradiction. $\square$

Since a nonredundant equivalent of a nonrecursive, single IDB-predicate program can be constructed [31], we have the following:

Corollary 5.6 It is decidable whether arbitrary nonrecursive, single IDB-predicate programs with binary IDB predicates (or unions of binary conjunctive queries) have CCI. $\square$

Although CCI for unions of conjunctive queries is decidable, it is also interesting to know, for a fixed k , if a nonrecursive program has k -CCI. For a single rule r , the integer ρ specified in the statement of Proposition 5.4 may not be the minimal integer k such that $\{r\}$ has k -CCI. For multi-rule programs and for $k = 1$, the following result provides a decidability result; while for $k > 1$ it is still open.

Proposition 5.7 It is decidable whether arbitrary nonrecursive, single IDB-predicate programs Π with binary IDB predicates have 1-CCI.

Proof Suppose p is the IDB predicate in Π . Let q_1 and q_2 be new unary IDB predicates. Then Π has 1-CCI iff $\Pi_{a_1 \dots a_k}^q \sqsubseteq \Pi$, where $\Pi_{a_1 \dots a_k}^q$ is constructed below, q an arbitrary EDB predicate in Π and k its arity, and $a_1, \dots, a_k$ are k arbitrary constants.

Let $r : p(x_1, x_2) \leftarrow A_1, \dots, A_m$ be in Π . First we split this rule into two rules: $r_1 : q_1(x_1) \leftarrow A_1, \dots, A_m$ and $r_2 : q_2(x_2) \leftarrow A_1, \dots, A_m$. Let Π' consists of $p(x_1, x_2) \leftarrow p_1(x_1), p_2(x_2)$ and all rules which can be obtained from r_1 and r_2 by unifying (using MGU) one or more q atoms in their bodies with $q(a_1, \dots, a_k)$.

Note that there are only a finite number of EDB predicates in Π and, by genericity of Datalog, we

only need to consider a finite number of “patterns” of constants in the above test. Further the above test is decidable. $\square$

We mentioned earlier that IEC exist for queries (Π, p) where nonrecursive programs Π_q are extended to recursive ones with CCI. One would naturally ask if it is decidable whether a recursive program has CCI. Unfortunately, the answer is no.

Proposition 5.8 It is undecidable whether an arbitrary program Π has CCI with respect to a given predicate.

Proof The proof is based on a reduction from the halting problem of Turing machines on empty inputs. The reduction is modified from the one used by Vardi et al [33, 19] in proving the undecidability of boundedness for binary Datalog programs. We briefly describe their reduction and the changes below.

Let M be a Turing machine with one-way infinite tape, an alphabet Σ , a set K of states, and a starting state $s \in K$. Configurations of M can be represented by words over the extended alphabet $\Sigma' = \Sigma \cup (K \times \Sigma)$, and computations of M by words over $\Sigma' \cup \{\#\}$ of the form “ $C_1\# \dots \#C_n$ ” where C_i ($i \in [1..n]$) is a configuration. The EDB predicates are described as follows. For each $a \in \Sigma' \cup \{\#\}$, there is a unary predicate q_a . A constant c encodes a iff $q_a(c)$ is true. The predicates *succ* represents the adjacency relation and *first* the first symbol. Intuitively, *succ* “represents” a word over $\Sigma' \cup \{\#\}$ which possibly encodes a computation of M .

In [33, 19], it was shown that a Datalog program Π' with only one binary IDB predicate *FING* can be constructed such that when the input is not a proper encoding or is an encoded halting computation, then Π' “floods” *FING*, i.e., inserting every pair of constants into *FING*. Otherwise, *FING* traverses the chains in *succ*. In particular, Π' has *encoding*, *halting*, *error detecting*, *finger pointing* and *moving* rules.

Now let q_0 be a new binary EDB predicate, and p a new binary IDB predicate. We construct a query (Π, p) where the program Π is as follows.

1. Π has all the encoding, error detecting, finger pointing and moving rules.
2. For each encoding and each error detecting rule $FING(u, v) \leftarrow B_1, \dots, B_n$, Π has a rule (u', v', u'', v'' are new variables):

$$p(u', v'') \leftarrow B_1, \dots, B_n, q_0(u', v'), q_0(u'', v'')$$

3. For each finger moving rule that moves into the halting state $FING(u, v) \leftarrow B_1, \dots, B_n$, Π has a rule (u' and v' are new variables):

$$p(u', v') \leftarrow q_0(u', v'), B_1, \dots, B_n$$

When M halts on the empty input, then there is a database D^h which has an encoding of the halting computation. Suppose D^h is a minimal database of this kind. Let $\Delta = \{succ(a, b)\}$ such that $succ(a, b) \in D$ and let $D = D^h - \Delta$. Obviously $\Pi(D)|_p = \emptyset$ while $\Pi(D \cup \Delta)|_p = \{p(c, d) \mid q_0(c, d) \in D\}$. Hence Π does not have k -CCI for any k . When M does not halt on the empty input, for each database D , either D

encodes a good but nonhalting computation or D is not an encoding. In the first case, $\Pi(D)|_p$ is empty and in the second case $\Pi(D)|_p$ is either flooded to be a cartesian-closed set (by rules of type 2 above) or empty. Hence Π has 1-CCI. $\square$

As an aside, by using a reduction to the undecidable problem of satisfiability of relational calculus [3], it can be shown that it is undecidable if a binary relational calculus (algebra) query has CCI (the detail is omitted).

We note that the notion of CCI can be generalized to predicates with arity > 2 and all the results on CCI reported here can also be generalized. For example, consider the case for rules. Suppose $r : p(x_1, \dots, x_k) \leftarrow A_1, \dots, A_m$ is a nonredundant and nonrecursive rule. Then $\{r\}$ has CCI iff, for each $i \in [1..m]$, (a) variables x_n with multiple occurrences in $p(x_1, \dots, x_k)$ must occur in A_i , and (b) for all distinct variables x_j and x_ℓ , either (b1) A_i contains x_j or x_ℓ , or (b2) x_j and x_ℓ are not $(\mathcal{S}_i, A_i)$ -connected (where $\mathcal{S}_i = \{A_1, \dots, A_m\} - \{A_i\}$).

6 Comparison with Related Work

The problem of incremental computation, in its most general form, can be stated as follows: after a state is changed to a new state, how can answers to some question in the new state be computed by changing the answer to the same question in the old state as little as possible? This incremental computation approach has been investigated for many different computational problems, such as transitive closure computation, database integrity checking, computation of models of stratified logic programs, and computation of models of Datalog programs. We will describe these in more detail below.

Our approach is an incremental computation approach according to the above description: we store derived relations for reuse after updates. More importantly, we emphasize the transformation of the original program into a more efficient nonrecursive program. The second basis is designed to fit database application by virtue of efficiency and easy programmability in database query languages.

We now briefly compare our approach with related work.

Semi-naive evaluation [6]. The basic idea of semi-naive evaluation is, in each iteration in the bottom-up evaluation, to compute only those facts that depend on at least one fact computed in the previous iteration. This approach differs from our approach in three ways: (i) the evaluation is incremental between iterations rather than between models, (ii) the changes that transfer one state to another state is internal to the database rather than external updates, and (iii) the original program is used with an iteration procedure rather than using a new program. As illustrated in Example 2.1, at least for the transitive closure case, semi-naive evaluation usually does more to get the new model than our incremental approach. This is because the incremental approach only produces facts that use the newly inserted fact in their proofs, and the semi-naive method leads to duplicated derivations at least from the first iteration.

Integrity constraint simplification [9, 28, 29]. The basic idea of integrity constraint simplification is to use an update to determine a simplified set of constraint instances that need to be checked after the update.

It is similar to our approach in using the information that the constraint was *satisfied* in a previous database state and in propagating the effect of an *update* to transform (and simplify) the constraint to be checked. Our approach, for query evaluation rather than for constraint checking, differs in storing previous derived relations and in transforming the programs used in query evaluation.

Efficient maintenance of (stratified) databases [4, 25]. The goal of this approach is to efficiently compute the standard model of a stratified database after a database update. It is similar to our approach in using the previous standard model (analogous to our stored relations) to simplify the task of computing the standard model (query answer) after the update. Our approach differs by storing intermediate relations rather than reasons (or “supports”) for including computed facts [4], by not using meta-programs to compute the difference between successive models [25], and by transforming the programs used in query evaluation. Our approach is, however, more restricted as it does not allow negation in rules and queries.

Incremental evaluation by counting [17]. The basic idea of this work is to use the number of derivation trees to achieve incremental evaluation of Datalog queries. In contrast, among other things, our incremental approach do not create new constants not in the original input database.

Incremental evaluation after deletion [14]. Complementary to the insertion case presented in this paper, [14] considers the computation of the transitive closure of graphs after the deletion of an edge, and gives nonrecursive queries for such computations for two classes of graphs (including the acyclic graphs).

Incremental evaluation of Datalog⁻ and its application to parallelism [34]. The approach in this work associates with each derived fact a collection of records of counters, one for each iteration in bottom-up evaluation. The counters remember the number of times the fact is derived, and the number of times the fact is deleted. The algorithms can handle general Datalog⁻ programs by using these counters from the appropriate iterations, but at the price of using recursive algorithms.

Incremental evaluation of arbitrary Datalog [16]. An algorithm is given in [16] for transforming an arbitrary Datalog query into an incremental query for arbitrary updates, but which is not in general nonrecursive.

Graph algorithms [21, 20, 26, 24]. Graph algorithms for on-line evaluation of transitive closure of graphs are given in [21, 20], and a method to optimize transitive queries by using subtrees in graphs constructed in previous evaluations is presented in [24]. The main difference is that they use more elaborate data structures and recursive algorithms, whereas we only use relations and nonrecursive Datalog programs.

We now compare with some other optimization approaches which are not incremental.

Partial evaluation in logic programming [27]. The idea of partial evaluation is to propagate given facts into programs so that subsequent queries involving those facts can be evaluated more efficiently. In this sense, this approach is also similar to ours, though it does not involve database updates or storage of derived relations. Our results may contribute to research on partial evaluation.

Magic sets [8]. Our incremental approach differs considerably from approaches such as the magic set approach [8] to query optimization. Indeed, incremental query evaluation is driven by anticipation, whereas magic set evaluation is driven by need. Consequently, it is difficult to combine the two approaches.

To see this, consider the path problem in Example 2.1. Suppose that we want to find all nodes reachable from a given node, say 1. Suppose further that our old set of facts contains two connected components such that 1 is in one component, and suppose the inserted fact connects the two components in some way. Since the magic set approach is driven by need, reachable nodes in the component not containing 1 must be computed from the *beginning* in an unbounded number of iterations depending on the original facts. In the incremental approach only one or two joins are necessary since the needed steps have previously been computed in *anticipation*.

Structural induction [7]. The idea is to build a simple programming language whose main computational engine is structural recursion on sets. Our work can be viewed as special cases of structural induction where structural recursion is deterministic and one fact at a time, and uses nonrecursive queries to compute the increment.

Computing Datalog queries using IEC is also related to the bounded iteration constructs [30] and the more general treatment of database states and their differences (deltas) [23].

7 Conclusions and Research Problems

We have considered the incremental evaluation problem for Datalog queries. The main idea is to use the facts computed in one state to reduce the cost of computing the answer to the same query after the insertion of a bounded number of facts. In an incremental evaluation system using conjunctive queries (IEC), such incremental evaluation is carried out by a nonrecursive Datalog program. We first presented an algorithm to construct an IEC for each regular chain query. This result was then extended to programs consisting of (i) regular chain rules and (ii) arbitrary nonrecursive rules (which are not necessarily chain rules and which may use predicates of arbitrary arities) defining nonrecursive predicates. Another extension was given for regular chain queries on the insertion of unbounded sets which are cartesian closed. We considered some complexity issues associated with the incremental programs. We also gave decidability result on weak regularity and results on when programs have the cartesian-closed increment property.

Queries permitting incremental evaluation using conjunctive queries can be viewed as a strict generalization of *bounded recursive Datalog queries* [22, 18]. In fact, such queries can perhaps be appropriately called “incrementally bounded queries.”

Several problems for future research are listed below.

- Incremental evaluation can compute more facts than computation using the original programs. Although such increased computation is amortized or “evenly distributed” over a number of query requests, it would be of interest to know when we should use incremental evaluation and when should we avoid using it.
- Can IEC be constructed for classes of Datalog programs substantially larger than weakly regular chain programs? (Note that the signal propagation program considered in Example 2.6 is roughly

equivalent to a regular chain program.) For example, can we have an IEC for the same-generation query?

- What other properties can be used to replace nonrecursiveness to measure the efficiency of the incremental program, especially for queries not associated with regular chain programs?

Acknowledgement The first author thanks V. Vianu for encouraging discussions, and H. Jakobsson for providing some useful references.

References

- [1] F. Afrati and S.S. Cosmadakis. Expressiveness of restricted recursive queries. In *Proc. ACM SIGACT Symp. on the Theory of Computing*, pages 113–126, 1989.
- [2] K. R. Apt, H. A. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann, 1988.
- [3] S. Abiteboul, R. Hull, and V. Vianu. *Database Theory: From A to F*. Manuscript. 1994.
- [4] K. R. Apt and J.-M. Pugin. Maintenance of stratified databases viewed as a belief revision system. In *Proc. Sixth ACM Symp. on Principles of Database Systems*, pages 136–145, 1987.
- [5] A. Aho and J. Ullman. Universality of data retrieval languages. In *Proc. ACM Symp. on Principles of Programming Languages*, pages 110–120, 1979.
- [6] F. Bancilhon. Naive evaluation of recursively defined relations. In M. L. Brodie and J. Mylopoulos, editors, *On Knowledge Base Management Systems: Integrating Artificial Intelligence and Database Technologies*. Springer-Verlag, 1985.
- [7] V. Breazu-Tannen, P. Buneman, and L. Wong. Naturally embedded query languages. In *Proc. 1992 Int. Conference on Database Theory*, LNCS 646, pages 140–154. Springer-Verlag, 1992.
- [8] F. Bancilhon, D. Maier, Y. Sagiv, and J. D. Ullman. Magic sets and other strange ways to implement logic programs. In *Proc. Fifth ACM Symp. on Principles of Database Systems*, pages 1–15, 1986.
- [9] F. Bry, H. Decker and R. Manthey. A uniform approach to constraint satisfaction and constraint satisfiability in deductive databases. In *Proc. First Int. Conf. on Extending Database Technology*, pages 488–505, 1988.
- [10] A.L. Buchsbaum, P.C. Kanellakis and J.S. Vitter. A data structure for arc insertion and regular path finding. In *Proc. ACM-SIAM Symp. on Discrete Algorithms*, 1990.

- [11] A. Chandra and P. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proc. ACM SIGACT Symp. on the Theory of Computing*, pages 77–90, 1977.
- [12] G. Dong. On Datalog linearization of chain queries. In J.D. Ullman, editor, *Theoretical Studies in Computer Science*, pages 181–206. Academic Press, 1991.
- [13] G. Dong. Datalog expressiveness of chain queries: Grammar tools and characterizations. In *Proc. Eleventh ACM Symp. on Principles of Database Systems*, pages 81–90, 1992.
- [14] G. Dong and J. Su. First-order on-line computation of transitive closure queries. In *Proc. of 16th Australian Computer Science Conference*, pages 721–729, 1993.
- [15] G. Dong and J. Su. First-Order Incremental Evaluation of Datalog Queries. *Proceedings of 4th International Workshop on Database Programming Languages*, 1993.
- [16] G. Dong and R. Topor. Incremental evaluation of Datalog queries. In *Proc. Int. Conference on Database Theory*, LNCS 646, pages 282–296. Springer-Verlag, 1992.
- [17] A. Gupta, D. Katiyar, and I. S. Mumick. Maintaining views incrementally. In *Proceedings of ACM SIGMOD Conference*, pages 157–166.
- [18] G. G. Hillerbrand, P. C. Kanellakis, H. G. Mairson, and M. Y. Vardi. Tools for Datalog boundedness. In *Proc. Tenth ACM Symp. on Principles of Database Systems*, pages 1–12, 1991.
- [19] G. Hillebrand, P. Kanellakis, H. Mairson, and M. Vardi. Undecidable boundedness problems for Datalog programs. Technical Report RJ 8739, IBM Almaden Research Center, San Jose, CA, Apr. 1992.
- [20] T. Ibaraki and N. Katoh. On-line computation of transitive closure of graphs. *Information Processing Letters*, 16:95–97, 1983.
- [21] G.F. Italiano. Amortized efficiency of a path retrieval data structure. *Theoretical Computer Science*, 48:273–281, 1986.
- [22] Y. Ioannidis. A time bound on the materialization of some recursively defined views. In *Proc. of International Conference on Very Large Data Bases*, 1985.
- [23] D. Jacobs and R. Hull. Database programming with delayed updates. In *Proc. 3rd Int. Workshop on Database Programming Languages*, pages 416–428, 1991.
- [24] H. Jakobsson. On materializing views and on-line queries. In *Proc. Int. Conference on Database Theory*, LNCS 646, pages 407–420. Springer-Verlag, 1992.
- [25] V. Küchenhoff. On the efficient computation of the difference between consecutive database states. In C. Delobel, M. Kifer, and Y. Masunaga, editors, *Proc. Second Int. Conf. on Deductive Object-Oriented Databases*, LNCS 566, pages 478–502. Springer-Verlag, 1991.

- [26] J. La Poutre and J. van Leeuwen. Maintenance of transitive closures and transitive reductions of graphs. Technical Report RUU-CS-87-25, Department of Computer Science, University of Utrecht, The Netherlands, 1987. An extended abstract of this paper appeared in LNCS 314, pp. 106-120.
- [27] J. W. Lloyd and J. C. Shepherdson. Partial evaluation in logic programming. *Journal of Logic Programming*, 11:217–242, 1991.
- [28] J. W. Lloyd, E. A. Sonenberg, and R. W. Topor. Integrity constraint checking in stratified databases. *Journal of Logic Programming*, 4(4):331–343, 1987.
- [29] J-M. Nicolas. Logic for improving integrity checking in relational data bases. *Acta Informatica*, 18(3):227–253, 1982.
- [30] X. Qian. On the expressive power of the bounded iteration construct. In *Proc. 2nd Int. Workshop on Database Programming Languages*, pages 411–421, 1989.
- [31] J. D. Ullman. *Principles of Database and Knowledge-base Systems, Vols I and II*. 1989, Computer Science Press.
- [32] M. H. van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976.
- [33] M. Vardi. Decidability and undecidability results for boundedness of linear recursive programs. In *Proc. of the ACM Symposium on Principles of Database Systems*, pages 341–351, 1988.
- [34] O. Wolfson, H.M. Dewan, S.J. Stolfo, and Y. Yemini. Incremental Evaluation of Rules and Its Relationship to Parallelism. In *Proc. of ACM SIGMOD Conference*, pages 78–87, 1991.