In *Proceedings of 5th International Conference on Database Theory, Prague, Czech Republic, January 1995*, pages 397-410 (LNCS #893)

Increment Boundedness and Nonrecursive Incremental Evaluation of Datalog Queries

(Extended Abstract)

Guozhu Dong[1] * and Jianwen Su[2] **

[1] Department of Computer Science
University of Melbourne
Parkville, Vic. 3052, Australia
dong@cs.mu.oz.au

[2] Department of Computer Science
University of California
Santa Barbara, CA 93106, U.S.A.
su@cs.ucsb.edu

Abstract. Given a recursive (datalog) query, the nonrecursive incremental evaluation approach uses nonrecursive (datalog) programs to compute the difference of the answers to the query against successive databases between updates. The mechanism used in this approach is called a "First-Order Incremental Evaluation System" (FOIES). We show that for two large classes of datalog queries, called "generalized (weakly) regular queries", FOIES always exist. We also define "increment boundedness" and its variations, which generalize boundedness. Increment bounded queries are shown to have FOIES of certain forms. We also relate increment boundedness to structural recursion, which was proposed for bulk data types. We characterize increment boundedness using the "insertion idempotency", "insertion commutativity", and "determinism" properties of structural recursion. Finally, we show that the increment boundedness notions are undecidable and a decidable sufficient condition is given.

1 Introduction

Recursive query optimization has been a focus of the study on the datalog language over the last several years (see [17] and recent PODS, SIGMOD, VLDB, DOOD proceedings). Most recently, query evaluation against databases that are frequently updated is studied in [8, 6, 5, 7, 10]. Specifically, to repeatedly evaluate the same (computationally expensive) recursive query on a database that is being updated between successive query requests, 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 entire query in the next state. It is considered as "desirable" if one can use nonrecursive queries to

* This author gratefully acknowledges support of Australian Research Council (ARC) through research grants and the Centre for Intelligent Decision Systems.

** Work supported in part by NSF grants IRI-9109520 and IRI-9117094. Part of work was done while visiting the University of Melbourne with partial support from an ARC grant to G. Dong.

compute the differences. In [8, 6, 5, 7], these ideas were abstracted in the so-called "first-order incremental query evaluation system" (or FOIES), and some interesting results were given; and in [15] a similar class called Dyn-FO was investigated as a complexity class. In this paper we continue our investigation in this direction by studying further on FOIES and by exploring some related query optimization issues.

Boundedness of datalog programs is one of the most-studied issues in the deductive database community [13, 14, 19, 18, 1, 12, 4]. In this paper we introduce and examine a related property, "increment boundedness", of datalog programs. Intuitively, a datalog program P is increment bounded if, to derive a new model after inserting one edb fact, one only needs to apply T_P^k for some fixed k (instead of T_P^ω).

On the other hand, the notion of "structural recursion" by insertion was proposed in [3] as a paradigm to specify database queries. Briefly speaking, a query defined by structural recursion consists of a base clause on the empty set, and a recursive clause on "inserting an element to a set".

In this paper, we establish close relationships between the three notions: FOIES, increment boundedness, and structural recursion. This paper has the following contributions: First, we extend our earlier results [7] on the existence of FOIES for (weakly) regular queries to wider classes of queries, "generalized (weakly) regular queries". These queries allow predicates of arbitrary arities, unlike in the (weakly) regular case where only binary chain rules are permitted in the recursive part. Second, increment boundedness and its variations are defined and shown to be equivalent to some subclasses of FOIES. Third, we characterize increment boundedness by the "insertion idempotency", "insertion commutativity", and "determinism" properties of structural recursion and show that commutativity is equivalent to increment boundedness for a syntactic class of programs. Fourth, undecidability results for increment boundedness and its variations are presented and a decidable sufficient condition is given. Increment boundedness turns out to be remarkably different from boundedness.

This abstract is organized as follows. Section 2 reviews some basic terminologies. Section 3 discusses generalized (weakly) regular queries. Section 4 presents the notion of increment boundedness and some results on them. Section 5 provides the characterizations of increment boundedness through structural recursion; and Section 6 focuses on the decision issues. Proofs are sketched or omitted in this extended abstract; full proofs will be included in the full version.

2 Preliminaries

We briefly review datalog and some necessary notions, and then present the definition of a FOIES.

We assume the existence of three pairwise disjoint infinite sets of *constants*, *variables*, and *predicates*. Built-in predicates such as equality are disallowed. Each predicate has a positive *arity*. An *atom* is a formula of the form $q(t_1, ..., t_k)$, where q is a predicate and $t_1, ..., t_k$ are variables or constants. A *fact* is an atom without variables. A *(datalog) program* is a finite set of rules of the form $A \leftarrow A_1, ..., A_n$, where $n \geq 1$, A and $A_1, ..., A_n$ are atoms and each variable in A occurs in some A_i. Let P be a program. We call a predicate occurring in P *intensional* (*idb*) if it occurs in the head of some rule in

P, and *extensional* (*edb*) otherwise. We use $idb(P)$ to denote the set of all idb predicates in P. For each set D of facts and each nonnegative integer i, let $T_P^i(D)$ denote the set of facts having derivation trees of depth at most i; each fact in D is considered to have a derivation tree of depth zero; and let $T_P^\omega(D) = \cup_{i=0}^\infty T_P^i(D)$. A *(datalog) query* is a pair (P, p), where P is a program, and p an (answer) idb predicate; for each set D of edb facts, the *answer* to the query is the set of facts over p in $T_P^\omega(D)$.

We now turn to FOIES, which use nonrecursive programs to maintain models of programs after insertion of facts. The nonrecursive programs need to differentiate facts in the old state (before an insertion) and facts in the new state (after the insertion). We adopt the following notation: for each predicate q, we shall use (i) q^o (o for old) as a predicate to represent facts over q computed or stored in the old database state, and (ii) q to represent facts over q that are asserted (either through insertion or through derivation) in the new state. For each set I of facts, let I^o be the set of facts obtained from I by replacing each predicate q with q^o. If S is a set of predicates, the restriction of I to those with predicate in S is denoted $I|_S$. We write $I|_p$ for $I|_{\{p\}}$.

Definition. A *FOIES* for a query (P, p) is a triple $\langle P_p, S, P_\delta \rangle$, where:

- S is a set of idb predicates containing p;
- P_p is a (possibly recursive) program, called the *initial program*, such that P_p and P are equivalent regarding their common idb predicates, i.e., $T_{P_p}^\omega(D)|_{idb(P)\cap S} = T_P^\omega(D)|_{idb(P)\cap S}$ for each set D of edb facts; and
- P_δ is a nonrecursive program, called the *incremental program*, such that, for each set D and each singleton set Δ of edb facts, P_δ derives the new least model of P_p containing $D \cup \Delta$ from Δ and the old least model of P_p containing D, i.e., $T_{P_p}^\omega(D \cup \Delta)|_S = T_{P_\delta}^\omega\big([T_{P_p}^\omega(D)|_S]^o \cup \Delta\big)|_S \cup T_{P_p}^\omega(D)|_S$.

A FOIES $\langle P_p, S, P_\delta \rangle$ for (P, p) is *space-free* if $S \subseteq idb(P)$.

Intuitively, a FOIES of a query always stores the answer on the current database. It may also store some auxiliary relations (predicates in S other than p). When the database changes, the FOIES uses the stored answer (and possibly auxiliary relations) to non recursively compute the new answer (and the new auxiliary relations). Space-free FOIES do not use any auxiliary relations. The following example illustrates the notion of FOIES.

Example 2.1. Consider the program P_1 computing the transitive closure of an edb predicate q: $\{p(x,z) \leftarrow q(x,z);\ \ p(x,z) \leftarrow q(x,y), p(y,z)\}$. Let $P_p = P_1$, $S = \{p\}$, and P_δ be the program

$$p(x,z) \leftarrow q(x,z) \qquad\qquad p(x,z) \leftarrow q(x,y), p^o(y,z)$$
$$p(x,z) \leftarrow p^o(x,y), q(y,z) \qquad\qquad p(x,z) \leftarrow p^o(x,y_1), q(y_1,y_2), p^o(y_2,z)$$

Then $\langle P_p, S, P_\delta \rangle$ is a FOIES for (P_1, p). To illustrate how the FOIES works, suppose $D = \{q(1,2), q(2,3),\ q(4,5), q(5,6)\}$ and $\Delta = \{q(3,4)\}$. Then $T_{P_p}^\omega(D) = D \cup \{p(i,j) \mid 1 \le i < j \le 3, 4 \le i < j \le 6\}$. To compute $T_{P_p}^\omega(D \cup \Delta)$ from $T_{P_p}^\omega(D)$ using P_δ, the facts in $T_{P_p}^\omega(D)$ are marked with the superscript o to indicate that they were facts before inserting the fact $q(3,4)$; the predicate q (resp. p) in P_δ denotes the additional

set of facts that are added (resp. derived) for q (resp. p). Thus, the additional fact for q is $q(3,4)$, and the additional facts for p are $\{p(i,j) \mid i \in [1..3] \text{ and } j \in [4..6]\}$.

Note that, to get the new paths after a q edge is added using this FOIES, only four (one if the new edge is treated as a pair of constants) joins are needed. Thus we have transformed the computation of a recursive program into the computation of a nonrecursive one (with the help of stored results). □

3 Generalized Regular Queries

In this section, we define "generalized (weakly) regular queries", which are subclasses of generalized chain queries, and extend the results on FOIES for subclasses of chain queries reported in [7] to generalized (weakly) regular queries. The results are new and the proofs require nontrivial new techniques.

The main results are Theorems 3.3, 3.5 and 3.6. Theorem 3.3 says that generalized regular queries (defined syntactically) have FOIES, Theorem 3.5 that generalized regular queries augmented by adding non recursive initializations having a property called "cci" have FOIES, and Theorem 3.6 that the cci property is decidable for non recursive programs which implies that the extended class, generalized weakly regular queries, is decidable. (This is because the class is defined syntactically except the cci property.)

A datalog rule r is a *generalized chain rule* if it has the following form:

$$r : p_0(\bar{x}_0, \bar{x}_n) \leftarrow p_1(\bar{x}_0, \bar{z}_0, \bar{x}_1), ..., p_i(\bar{x}_{i-1}, \bar{z}_{i-1}, \bar{x}_i), ..., p_n(\bar{x}_{n-1}, \bar{z}_{n-1}, \bar{x}_n) \quad (1)$$

where $\bar{x}_0, ..., \bar{x}_n, \bar{z}_0, ..., \bar{z}_{n-1}$ are disjoint (possibly empty) sequences of variables. Note that *chain rules* are special generalized chain rules, where $\bar{x}_0, ..., \bar{x}_n$ are distinct single variables and $\bar{z}_0, ..., \bar{z}_{n-1}$ are all empty sequences of variables. A *generalized chain program* is a finite set of generalized chain rules; and a *generalized chain query* is a query (P, p), where P is a generalized chain program. Similar to chain queries, each generalized chain query $Q = (P, p)$ can be associated with a context-free grammar G_Q constructed as follows. The terminals (nonterminals) are the edb (idb) predicates; the start nonterminal is p; and for each rule in P of the form (1) there is a production of the form $p_0 \rightarrow p_1 p_2 \cdots p_n$.

A query Q is *generalized regular (g.r.)* if G_Q is right-linear, i.e., the only nonterminal in the right hand side of each production is the rightmost symbol.

Obviously, each chain query (program) is also generalized chain; each regular (chain) query [7] is also generalized regular. In particular, transitive closure can be expressed by a g.r. query. For nonrecursive generalized chain queries, the next proposition follows from a result in [7].

Proposition 3.1. Each nonrecursive generalized chain query is a conjunctive query, but not vice versa. □

We now give an example of a g.r. chain query which is not a chain query, together with a FOIES for it.

Example 3.2. Let $Q = (P_2, p)$ be a query, where $P_2 = \{p(x) \leftarrow s(x, y), p(y);\ p(x) \leftarrow q(x)\}$. Q represents the propagation of signals q through a network s of logical "or"

gates ($s(x,y)$ means the gate x has y as an input wire). Q is g.r. but not a (regular) chain query. Q also has a FOIES. Indeed, let $S = \{p, t\}$, where $t(x,y)$ denotes x is "on" whenever y is "on" and

$$P_p = \left\{ \begin{array}{l} t(x,y) \leftarrow s(x,y) \\ t(x,z) \leftarrow t(x,y), t(y,z) \\ p(x) \leftarrow q(x) \\ p(x) \leftarrow t(x,y), q(y) \end{array} \right\},$$

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

It can be verified that $\langle P_p, S, P_\delta \rangle$ is a FOIES for Q but Q has no space-free FOIES. Observe that P_p computes the entire transitive closure of s, whereas P_2 only computes the part reachable from constants in q.

Theorem 3.3. Each g.r. (generalized regular) query has a FOIES.

The proof idea is similar to the one used in [7] for regular (chain) queries. To construct a FOIES for a g.r. query, we first find a regular expression E corresponding to the grammar, and E uses only symbols from the set of idb predicates, "(", ")", "$\cup$", and "$^+$". This can be done by standard procedures [2]. For example, for P_2, the regular expression is $(s^+ q) \cup q$. We then use an auxiliary relation for each subexpression of E and the incremental program is obtained inductively according to (roughly) the syntax tree of E. The basis for both [7] and here is that all new derived facts after inserting a fact A can be obtained by using the old derived facts and A a bounded number of times. The proof and construction for the g.r. case, however, require some new techniques. The modification to this proof, as well as to the proofs of other results, is needed due to the following difference between chain and generalized chain programs:

Note that chain programs consist of rules of the form (1) where all $\overline{z}_i$'s are empty and each $\overline{x}_i$ is a single variable. Consequently, chain queries operate on graphs where each edge has two constants as nodes. On the other hand, generalized chain queries operate on hypergraphs where each hyperedge may have three nodes, each being a sequence of constants. Specifically, a fact $q(a_1, ..., a_k)$ corresponds to many hyperedges, e.g., for each $j \in [1..(k-1)]$, it can be used as a hyperedge from $a_1, ..., a_j$ to $a_{j+1}, \ldots, a_k$ (there are many other ways where the middle constants do not connect other constants in the edges before and after this edge).

We now consider "generalized weakly regular queries".

We say $p[j_1, j_2, j_3]$ is a *partition* of a predicate p if j_1, j_2, j_3 are nonnegative integers such that $j_1 + j_2 + j_3$ = arity of p. Each rule r of form (1) above is said to *use* the partition $p_i[|\overline{x}_{i-1}|, |\overline{z}_{i-1}|, |\overline{x}_i|]$ for each $i \in [1..n]$ ($|\overline{x}|$ denotes the length of $\overline{x}$). Intuitively, r uses $p_i[j_1, j_2, j_3]$ if facts over p_i may be unified with an atom $p_i(\overline{x}_{i-1}, \overline{z}_{i-1}, \overline{x}_i)$ such that the first j_1 constants are identical to the last j_1 constants of the preceding fact, the last j_3 constants are identical to the first j_3 constants of the following fact, and the middle j_2 constants are not restricted.

A set D of facts over a predicate q is called $q[j_1, j_2, j_3]$-*cartesian closed* if, $q[j_1, j_2, j_3]$ is a partition of q and, whenever $q(\bar{a}_1, \bar{c}_1, \bar{b}_1)$ and $q(\bar{a}_2, \bar{c}_2, \bar{b}_2)$ are in D where $|\bar{a}_1| = |\bar{a}_2| = j_1$ and $|\bar{b}_1| = |\bar{b}_2| = j_3$, there is some $\bar{c}$ so that $q(\bar{a}_1, \bar{c}, \bar{b}_2)$ belongs to D.

This notion is a special case of embedded multivalued dependency (with an empty left-hand-side [16]). Note that D is always $q[j_1, j_2, j_3]$-cartesian closed if either D is a singleton set, or $j_1 = 0$ or $j_3 = 0$.

The notion of cartesian-closed sets is important for incremental evaluation since it allows us to extend results on single fact to sets: the insertion of a cartesian-closed set will behave like the insertion of a single fact in incremental evaluation, which can be dealt with by FOIES for generalized regular queries.

Definition. A program P has *k-cartesian-closed increment* (*k-cci*) *w.r.t.* a partition $p[j_1, j_2, j_3]$ if $k \geq 0$ and for each set D and singleton set Δ of edb facts, there are k $p[j_1, j_2, j_3]$-cartesian-closed sets $C_1, ..., C_k$ satisfying

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

P has *cci w.r.t.* $p[j_1, j_2, j_3]$ if it has k-cci w.r.t. $p[j_1, j_2, j_3]$ for some k.

Example 3.4. Program $P_3 = \{p_1(x, y) \leftarrow q_1(x, u, v), q_5(v, w, z), q_3(z, y)\}$ has 1-cci w.r.t. $p_1[1, 0, 1]$. Intuitively, for each database D and each set Δ of one edb fact, let

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

Then C is cartesian closed, and $P_3(D \cup \Delta) - P_3(D) \subseteq C \subseteq P_3(D \cup \Delta)$ hold.

An empty (binary) program has 1-cci w.r.t. $p_1[1, 0, 1]$, since the empty set is cartesian closed. Program $P_4 = \{p_1(x, y) \leftarrow q_1(x, u, v), q_5(v, u, z), q_6(z, u, y)\}$ also has 1-cci w.r.t. $p_1[1, 0, 1]$.

Program $P_5 = \{p_1(x, y) \leftarrow q_4(x, y), q_2(u, v)\}$ does not have cci w.r.t. $p_1[1, 0, 1]$. 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 $P_5(D \cup \{q_2(a, b)\}) - P_5(D)$ is not bounded by any k cartesian-closed sets in $P_5(D \cup \{q_2(a, b)\})$, violating the two containments in the above definition.

We next define "generalized weakly regular" queries which may allow nonrecursive initialization.

Definition. A query (P, p) is *generalized weakly regular (g.w.r.)* if $P = P_c \cup P_r$ satisfies the following:

1. P_c is non recursive;
2. Each predicate in the heads of rules in P_r does not occur in P_c;

3. For each $q \in idb(P_c)$, let P_c^q be the set of rules defining q. The program Q_q with only one idb-predicate (i.e., q), modified from P_c^q by eliminating all other idb predicates through expansion, has cci for each partition $q[j_1, j_2, j_3]$ of q used by P_r.
4. (P_r, p) is a g.r. query (viewing idb predicates in P_c as edb predicates).

Since the nonrecursive initialization part of a g.w.r. query has cci, Theorem 3.3 can be generalized:

Theorem 3.5. Each g.w.r. (generalized weakly regular) chain query has a FOIES. □

Finally, we consider the decision problem for cci. It turns out that the decidability and undecidability results in [7] can also be extended (nontrivially) to the case with partitions. In the following, we state the results and two key lemmas used in proving the decidability result.

Theorem 3.6. It is decidable if a nonrecursive program has cci w.r.t. a partition $p[i, j, k]$; undecidable if a recursive program has cci w.r.t. $p[i, j, k]$. □

A program P is nonredundant if P cannot be syntactically simplied by removing "elements" from it, i.e., (i) P is not equivalent to any of its proper subsets and (ii) P is not equivalent to any P' obtained by removing atoms from its rule bodies.

Lemma 3.7. A nonredundant, nonrecursive program P with a single idb-predicate has cci w.r.t. $p[i, j, k]$ iff $\{r\}$ has cci w.r.t. $p[i, j, k]$ for each $r \in P$. □

For each set S of atoms and each atom A, we say two variables x and y are (S, A)-*connected* if there is a sequence $A_1, ..., A_\ell$ ($\ell \geq 1$) of atoms from S such that, x occurs in A_1, y occurs in A_ℓ, and A_i and A_{i+1} share a variable x_i not occurring in A for each $1 \leq i < \ell$. For examples, for $S = \{q_1(x, u, v), q_2(z, u, y)\}$ and $A = q(v, v, z)$, x and y are (S, A)-connected; however, for $A' = q(v, u, z)$, x and y are not (S, A')-connected. If two variables occur in a common atom in S, then they are (S, A)-connected for every A. Note that (S, A)-connectivity reduces to the usual connectivity when A contains no variable.

For each sequence of variables $\overline{x}$, we define $V(\overline{x})$ to be the set of variables occurring in $\overline{x}$. We also extend V to atoms naturally.

Lemma 3.8. A rule $r : p(\overline{x}, \overline{y}, \overline{z}) \leftarrow A_1, ..., A_m$ has cci w.r.t. $p[|\overline{x}|, |\overline{y}|, |\overline{z}|]$ iff for each $i \in [1..m]$,

1. $V(\overline{x}) \cap V(\overline{z}) \subseteq V(A_i)$; and
2. either $V(\overline{x}) \subseteq V(A_i)$, or $V(\overline{z}) \subseteq V(A_i)$, or for each $x \in V(\overline{x}) - V(\overline{z})$ and each $z \in V(\overline{z}) - V(\overline{x})$, x, z are not (S, A_i)-connected (where $S = \{A_j | 1 \leq j \leq m\}$).

□

For example, P_3 in Example 3.4 has 1-cci w.r.t. $p[1, 0, 1]$ according to this lemma. Indeed, let $A_1 = q_1(x, u, v)$, $A_2 = q_5(v, w, z)$ and $A_3 = q_3(z, y)$. Then $V(\overline{x}) \cap V(\overline{z}) = \emptyset \subseteq V(A_i)$ for each $i \in [1..3]$. For each $i \in \{1, 3\}$, we have either $V(\overline{x}) \subseteq V(A_i)$, or $V(\overline{z}) \subseteq V(A_i)$. Furthermore, x, y are not (S, A_2)-connected where $S = \{A_1, A_2, A_3\}$.

4 Increment Boundedness and FOIES

We define a new property, called "increment boundedness", for datalog programs and establish two equivalence relationships with space-free FOIES and FOIES that are syntactically constructible from the programs. A characterization of increment boundedness using "structural recursion" is given in the next section and decision problems are discussed in Section 6.

Intuitively, a program P is ℓ-increment bounded ($\ell \geq 1$) if, upon an insertion of one edb fact A into a database D, the new minimal model $T_P^\omega(D \cup \{A\})$ can be obtained by apply T_P^ℓ (instead of the usual T_P^ω) to the previous minimal model $T_P^\omega(D)$ and the new fact A. Hence, an ℓ-increment bounded program needs no recursion to update its minimal model upon insertion. This concept is closely related to the popular bounded class [9] but with some interesting distinct behavior. Formally, we have:

Definition. Suppose P is a datalog program.

- For each $\ell \geq 1$ [and each edb predicate q], P is *ℓ-increment bounded* (*ℓ-IB*) [w.r.t. q] if
$$T_P^\omega(D \cup \{A\}) = T_P^\ell(T_P^\omega(D) \cup \{A\})$$
for each set D of edb facts and for each edb fact A [over q].
- P is *increment bounded* (*IB*) if it is ℓ-IB for some fixed ℓ.
- P is *generalized increment bounded* (*GIB*) if there is some nonrecursive program Q such that
$$T_P^\omega(D \cup \{A\}) = T_Q(T_P^\omega(D) \cup \{A\})$$
for each set D of edb facts and for each edb fact A.

We also denote by ℓ-IB (resp., IB, GIB) the set of programs which are ℓ-IB (resp., IB, GIB).

Example 4.1. Let P_6 be P_δ in Example 2.1 except the o superscript is removed. Note that the predicate p holds the transitive closure of q. P_6 is increment bounded (by Theorem 4.2). In fact, P_6 is 1-IB: Suppose p contained the transitive closure of q before the insertion. Then after inserting a q fact, the new closure is obtained by applying $T_{P_6}^1$ once. However, the equivalent program P_1 in Example 2.1 is not even increment bounded. Indeed, for each $\ell \geq 1$, $T_{P_1}^\omega(D \cup \{A\}) \neq T_{P_1}^\ell(T_{P_1}^\omega(D) \cup \{A\})$, where $D = \{q(i, i+1) \mid 0 \leq i \leq 2\ell, i \neq \ell\}$ and $A = q(\ell, \ell+1)$. □

Example 4.1 indicates that IB is not semantic. (A property is semantic if it is closed under substitution by equivalent programs [9].) Recall that a program P is *bounded* if there is an integer ℓ such that $T_P^\omega(D) = T_P^\ell(D)$ for each set D of edb facts. It follows that each bounded program is also increment bounded but not vice versa.

Theorem 4.2. Let (P, p) be a query. If P is ℓ-IB, then (P, p) has a FOIES (and the FOIES can be constructed from P by syntactic changes). Furthermore, P is GIB iff (P, p) has a space-free FOIES. Consequently, each IB query has a space-free FOIES; but the converse is not true.

For proving the first statement, for each ℓ-IB query (P, p), we construct a FOIES $\langle P, idb(P), Q_\ell \rangle$, where, roughly, Q_ℓ is obtained by "expanding" P up to ℓ times (all derivation trees of height $\leq \ell$) and marking appropriate predicates with superscript o. To be more precise, consider an arbitrary rule in the expansion:

$$A \leftarrow B_1, \ldots, B_m, C_1, \ldots, C_n$$

where each B_i is an idb atom and C_j is an edb atom. Then, for each proper subset M of $[1..n]$, Q_ℓ includes the following rule:

$$A \leftarrow B_1^o, \ldots, B_m^o, C_1', \ldots, C_n'$$

where C_j' is C_j if $j \notin M$ and C_j' is C_j^o otherwise. (If $C = p(t_1, \ldots, t_k)$, then by C^o we mean $p^o(t_1, \ldots, t_k)$.)

The second statement is easily verified from the definitions. For the last statement, the standard TC program P_1 (Example 2.1) has a space-free FOIES, but is not IB.

Example 4.3. Let P_1 be as in Example 2.1. Then, for (P_1, p), there is a space-free FOIES. Let P_6 be P_δ in Example 2.1 except the o superscript is removed, as considered in Example 4.1. Then there is a space-free FOIES $\langle P_6, \{p\}, P_6' \rangle$ for (P_6, p), where P_6' is constructed as above, and it happens that $P_6' = P_\delta$ (in Example 2.1).

It can be verified that the original transitive closure query (P_1, p) and the or-gate propagation query Q in Example 3.2 do not have FOIES constructed in this way. □

The following result shows that there is a strict hierarchy among the classes.

Theorem 4.4. $\forall \ell \geq 1$, ℓ-IB $\subsetneq$ $(\ell + 1)$-IB. Also, $\cup_{\ell \geq 0} \ell$-IB = IB $\subsetneq$ GIB.

Proof. Obviously the containments hold. The first containment is proper because we can write inefficient datalog programs that delays output; for example, for $\ell = 1$, the two-rule program $\{p_1(x) \leftarrow q(x);\ \ p_2(x) \leftarrow q(x), p_1(y)\}$ is 2-IB, but not 1-IB. The second containment is proper since the standard TC program P_1 (Example 2.1) belongs to GIB but not to IB. □

In terms of the number of iterations needed for evaluating IB queries, we have:

Theorem 4.5. If (P, p) is an IB query, then the number of iterations needed to derive $T_P^\omega(D)|_p$ using semi-naive evaluation method is at most linear in the number of facts in D.

Finally, similar to the boundedness problem of whether a given program is equivalent to some nonrecursive program, it is also interesting to know when a given program is equivalent to some IB program.

5 FOIES and Structural Recursion

Structural recursion [3] was proposed as a database programming paradigm on "bulk" types such as sets or relations. Structural recursion can be performed based either on insertion or on union. Under the insertion approach, the computation (query) on an input set S is divided into the computation on a subset which contains all but one element of S, followed by the computation for inserting the remaining element. The computation on the subsets of S is done recursively in that manner.

Since both FOIES and structural recursion accomplish computation in a similar way, it is interesting to know how FOIES relates to structural recursion.

We consider Datalog programs having the special forms of space-free FOIES corresponding to the IB property as discussed Theorem 4.2. We give two characterizations of such IB programs using structural recursion. The first shows the equivalence of four properties: (i) IB, (ii) determinism, (iii) idempotency, and (iv) the combination of weak idempotency and commutativity. The second refines the first by establishing the equivalence of IB and commutativity for connected datalog programs. The restrictions on programs are shown necessary to guarantee this equivalence.

Roughly, structural recursion by insertion allows inductive definition of mappings on sets by inserting one element at a time, where each element may be inserted more than once. Formally, given a constant e and a function ϕ, we define a mapping g as follows[3]:

$$\begin{array}{ll} \textit{fun} & g(\emptyset) = e \\ \mid & g(\textit{Insert}(A, S)) = \phi(A, g(S)) \end{array}$$

Following [3], we will write in combinator style $\Psi(e, \phi)$ for g. The typing is $\Psi(e, \phi) : \{\alpha\} \rightarrow \beta$, provided that $e : \beta$ and $\phi : \alpha \times \beta \rightarrow \beta$.

The mapping g is called *deterministic* if, for all appropriate S, $g(S)$ has exactly one value, that is, $\phi(A_1, \phi(A_2, ..., \phi(A_m, e))) = \phi(B_1, \phi(B_2, ..., \phi(B_n, e)))$ whenever $A_1, ..., A_m$ and $B_1, ..., B_n$ are sequences satisfying $\{A_1, ..., A_m\} = S = \{B_1, ..., B_n\}$. The following two properties imply determinism [3]: If ϕ is a function whose typing is $\alpha \times \beta \rightarrow \beta$, then (i) ϕ is called *(insertion) idempotent* if

$$\phi(A, \phi(A, \phi(A_1, \phi(A_2, ..., \phi(A_m, e)...)))) = \phi(A, \phi(A_1, \phi(A_2, ..., \phi(A_m, e)...)))$$

for all $A, A_1, ..., A_m$ ($m \geq 0$) of type α, and (ii) ϕ is called *(insertion) commutative* if

$$\phi(A, \phi(B, \phi(A_1, \phi(A_2, ..., \phi(A_m, e)...)))) = \phi(B, \phi(A, \phi(A_1, \phi(A_2, ..., \phi(A_m, e)...))))$$

for all $A, B, A_1, ..., A_m$ ($m \geq 0$) of type α. We further define ϕ to be *weakly idempotent* w.r.t. e if $\phi(A, \phi(A, e)) = \phi(A, e)$ for all A.

The proposition below shows the relationship among determinism, idempotency, and commutativity for general functions. As it will be seen later, (b) shows an interesting difference between structural recursion using general function and using datalog mappings.

Proposition 5.1. (a) Determinism $\equiv$ idempotency + commutativity. (b) Idempotency and commutativity are incomparable. (c) Commutativity + weakly idempotency imply idempotency. □

[3] *Insert* is a set constructor so that *Insert*(A, S) returns the set obtained by inserting A (an element) into a set S.

For each program P and each positive integer ℓ, define I_P^ℓ by $I_P^\ell(A, S) = T_P^\ell(S \cup \{A\})$. We will write I_P^1 as I_P. From here on we assume that e is the empty set $\emptyset$.

The next theorem presents the first characterization for IB.

Theorem 5.2. If ℓ is a positive integer, then (a) P is ℓ-IB *iff* (b) I_P^ℓ is idempotent *iff* (c) $\Psi(\emptyset, I_P^\ell)$ is deterministic *iff* (d) I_P^ℓ is commutative and weakly idempotent.

Proof. By Proposition 5.1, we have (d) implies (b), and so (c) and (d) are equivalent; and we have (c) implies (b). It is easy to prove that (a) and (b) are equivalent and that (b) implies (c). □

As an aside, from the equivalence of (b) and (c) it is easily seen that the expressive power of structural recursion using datalog is confined to datalog. That is, all deterministic mappings defined by using datalog mappings in structural recursion by insertion are themselves datalog mappings.

Corollary 5.3. A query (P, p) has a space-free FOIES if for some ℓ, one of the following conditions holds: (1) I_P^ℓ is idempotent, (2) $\Psi(\emptyset, I_P^\ell)$ is deterministic, or (3) I_P^ℓ is commutative and weakly idempotent. □

We now give results on the relationships between commutativity and IB for datalog programs.

Proposition 5.4. Commutativity does not imply IB in general: There are constant-free (nonrecursive or recursive) programs P such that I_P is commutative but not idempotent. Consequently, commutativity for datalog mappings does not imply IB and determinism.

Proof. Let P_7 be the nonrecursive program consisting of the following two rules:

$$\begin{array}{ll} r_1: & p_1(x) \leftarrow q(x) \\ r_2: & p_2(x) \leftarrow q(x), p_1(y) \end{array}$$

Then I_{P_7} is not idempotent:

$$I_{P_7}(q(1), \emptyset) = \{q(1), p_1(1)\} \neq I_{P_7}(q(1), I_{P_7}(q(1), \emptyset)) = \{q(1), p_1(1), p_2(1)\}.$$

To see that I_{P_7} is commutative, let $A_0, \ldots, A_m$ ($m \geq 1$) be a sequence of facts. Let c be a constant occurring in some A_i. From r_1 we see that the following two statements hold:

$$p_1(c) \in I_{P_7}(A_0, I_{P_7}(A_1, I_{P_7}(A_2, \ldots, I_{P_7}(A_m, \emptyset)\ldots))),$$
$$p_1(c) \in I_{P_7}(A_1, I_{P_7}(A_0, I_{P_7}(A_2, \ldots, I_{P_7}(A_m, \emptyset)\ldots))).$$

Since $m \geq 1$, there are p_1 facts in each of the following two sets:

$$I_{P_7}(A_1, I_{P_7}(A_2, \ldots, I_{P_7}(A_m, \emptyset)\ldots)),$$
$$I_{P_7}(A_0, I_{P_7}(A_2, \ldots, I_{P_7}(A_m, \emptyset)\ldots)).$$

Hence, from r_2 we see that the following two statements hold:

$$p_2(c) \in I_{P_7}(A_0, I_{P_7}(A_1, I_{P_7}(A_2, \ldots, I_{P_7}(A_m, \emptyset)\ldots))),$$
$$p_2(c) \in I_{P_7}(A_1, I_{P_7}(A_0, I_{P_7}(A_2, \ldots, I_{P_7}(A_m, \emptyset)\ldots))).$$

So commutativity follows. □

Theorem 5.5. If P is a connected[4] and constant-free datalog program, then P is ℓ-IB *iff* I^ℓ_P is idempotent *iff* I^ℓ_P is commutative.

Proof. Suppose P is a connected and constant-free datalog program. By Theorem 5.2, it suffices to show that I^ℓ_P is weakly idempotent.

We call two sets of facts S_1 and S_2 disconnected if there are no facts $A_1 \in S_1$ and $A_2 \in S_2$ such that A_1 is connected to A_2 in $S_1 \cup S_2$. Clearly T^ℓ_P maps disconnected sets to disconnected sets, and $T^\ell_P(D_1 \cup D_2) = T^\ell_P(D_1) \cup T^\ell_P(D_2)$ for all disjoint sets D_1 and D_2 of facts.

Let A be an arbitrary fact, and B a fact not[5] using any constants in A. Then

$$\begin{aligned} & I^\ell_P(A, I^\ell_P(B, \emptyset)) = T^\ell_P(T^\ell_P(\{B\}) \cup \{A\}), \quad \text{by definition of } I^\ell_P \\ = \; & T^\ell_P(T^\ell_P(\{B\})) \cup T^\ell_P(\{A\}), \quad \text{since } \{A\} \text{ and } \{B\} \text{ are disconnected.} \end{aligned}$$

Similarly, $I^\ell_P(B, I^\ell_P(A, \emptyset)) = T^\ell_P(\{B\}) \cup T^\ell_P(T^\ell_P(\{A\}))$. Suppose I^ℓ_P is commutative. Then

$$I^\ell_P(A, I^\ell_P(B, \emptyset)) = I^\ell_P(B, I^\ell_P(A, \emptyset)).$$

Since P is connected and A and B are disconnected, we get $T^\ell_P(\{A\}) = T^\ell_P(T^\ell_P(\{A\}))$, that is, $I^\ell_P(A, \emptyset) = I^\ell_P(A, I^\ell_P(A, \emptyset))$. □

Corollary 5.6. A query (P, p) has a space-free FOIES if P is connected and constant-free and I^ℓ_P is commutative for some ℓ. □

6 Decision Problems for Increment Boundedness

We consider here the decision issues for the increment bounded properties. The main results of the section show that they are all undecidable. The results also raise an interesting contrast between boundedness and increment boundedness: While they are both undecidable for the general case, for a fixed ℓ, ℓ-boundedness is decidable [4] but ℓ-IB is still undecidable.

In [9], it was shown that undecidability of boundedness can be translated into undecidability for many other properties. The translation, however, is not applicable to our context because none of the IB notions is semantic.

Theorem 6.1. It is undecidable for an arbitrary datalog query Q if Q is GIB; if Q is IB.

Proof. In [7], it was shown that it is undecidable if a query has a space-free FOIES. By Theorem 4.2, it follows easily that GIB is also undecidable. By using a direct reduction from the halting problem, IB can also be shown undecidable. □

[4] Two atoms are *directly connected* if they share a variable or constant. Given a set S of atoms, two atoms in S are *connected in* S if either they are directly connected, or there is an atom C in S which is connected to both of them in S. A rule $A \leftarrow A_1, ..., A_m$ is *connected* if for each $i, j \in [1..m]$, A_i, A_j are connected in $\{A_1, ..., A_m\}$. A program is *connected* if each rule is connected.

[5] Here we assume that there are unbounded number of constants.

We next consider ℓ-IB for a fixed ℓ. It turns out to be again undecidable which is interesting compared to the decidability of ℓ-boundedness.

Theorem 6.2. It is undecidable whether P is ℓ-IB w.r.t. q for arbitrary datalog programs P and arbitrary edb predicate q.

The proof is based on the following lemma, which shows that ℓ-boundedness for databases that are least models of some datalog program is undecidable.

Definition. For datalog programs P_α, P_β, P_α is *ℓ-bounded* ($\ell \geq 1$) *w.r.t.* P_β if for each set D of edb facts $T_{P_\alpha}^{\ell+1}(T_{P_\beta}^{\omega}(D)) \subseteq T_{P_\alpha}^{\ell}(T_{P_\beta}^{\omega}(D))$.

Lemma 6.3. Let $\ell \geq 1$ be fixed. It is undecidable if a program is ℓ-bounded w.r.t. another.

Proof. The proof is based on a reduction using a similar technique as in [11]. (The reduction simulates the checking of whether the input database describes a halting configuration of a Turing machine starting from an empty tape.) □

In view of the undecidability results, it is interesting to give some sufficient yet decidable conditions for ℓ-IB. In the following we provide a sufficient condition (in Proposition 6.5) which is based on "uniform query containment relative to a program P" ($\subseteq_P^u$).

Definition. Let P be a datalog program and Q_1, Q_2 two queries. Q_1 is *uniformly contained in Q_2 relative to P*, denoted $Q_1 \subseteq_P^u Q_2$, if for each model D of P, $Q_1(D) \subseteq Q_2(D)$.

Intuitively, the problem of uniform relative query containment is to consider containment under certain "integrity" constraints specified by some datalog program.

Lemma 6.4. For an arbitrary program P and two conjunctive queries Q_1, Q_2, it is decidable if $Q_1 \subseteq_P^u Q_2$. □

Thus the following provides a decidable sufficient condition for ℓ-IB:

Proposition 6.5. Suppose P is an arbitrary datalog program. Then P is ℓ-IB if, for each edb fact $A = q(\overline{a})$ where q occurs in P, $T_{P_A}^{\ell+1} \subseteq_P^u T_{P_A}^{\ell}$, where P_A is obtained as follows: For each rule r containing q in the rule body, add each rule r' to P_A, where r' is obtained by unifying one or more q atoms in the body of r with A. □

Acknowledge

The authors are grateful to the anonymous referees for their helpful comments on an earlier version of the paper.

References

1. S. Abiteboul. Boundness is undecidable for datalog programs with a single recursive rule. *Inf. Process. Lett.*, 32(6), October 1989.
2. A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Company, 1974.
3. V. Breazu-Tannen, P. Buneman, and S. Naqvi. Structural recursion as a query language. In *Proceedings of the Third International Workshop on Database Programming Languages: Bulk Types and Persistent Data*, pages 9–19, 1991.
4. S. Chaudhuri and M. Vardi. On the equivalence of recursive and nonrecursive datalog programs. In *Proc. ACM Symp. on Principles of Database Systems*, 1992.
5. G. Dong and J. Su. First-order incremental evaluation of datalog queries (extended abstract). In *Proc. 4th Int. Workshop on Database Programming Languages*, 1993.
6. 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.
7. G. Dong, J. Su, and R. Topor. First-order incremental evaluation of datalog queries. Technical report, Dept. of Comp. Sci., Univ. of Melbourne, 1993. (Submitted to AMAI).
8. G. Dong and R. Topor. Incremental evaluation of datalog queries. In *Proc. Int'l Conference on Database Theory*, pages 282–296, Berlin, Germany, Oct. 1992.
9. H. Gaifman, H. Mairson, Y. Sagiv, and M. Y. Vardi. Undecidable optimization problems for database logic programs. *Journal of the Association for Computing Machinery*, 40(3):683–713, 1993.
10. A. Gupta, I. S. Mumick, and V. S. Subrahmanian. Maintaining views incrementally. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 157–166, 1993.
11. 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.
12. G. Hillerbrand, P. Kanellakis, H. Mairson, and M. Vardi. Tools for Datalog boundedness. In *Proc. ACM Symp. on Principles of Database Systems*, 1991.
13. Y. Ioannidis. A time bound on the materialization of some recursively defined views. In *Proc. of Int. Conf. on Very Large Data Bases*, 1985.
14. J. Naughton and Y. Sagiv. A decidable class of bounded recursion. In *Proc. ACM Symp. on Principles of Database Systems*, pages 227–236, 1987.
15. S. Patnaik and N. Immerman. Dyn-FO: A parallel dynamic complexity class. In *Proc. ACM Symp. on Principles of Database Systems*, pages 210–221, 1994.
16. J. D. Ullman. *Database and Knowledge-Base Systems*, volume 1. Computer Science Press, 1988.
17. J. D. Ullman. *Database and Knowledge-Base Systems*, volume 2. Computer Science Press, 1989.
18. R. van der Meyden. Predicate boundedness of linear monadic datalog is in PSPACE. Technical Report SCS&E Report 9205, Univ. of New South Wales, 1992.
19. M. Vardi. Decidability and undecidability results for boundedness of linear recursive programs. In *Proc. ACM Symp. on Principles of Database Systems*, pages 341–351, 1988.

This article was processed using the LaTeX macro package with LLNCS style